Distributed Multi-Robot Exploration under Complex Constraints

Alberto Viseras

Distributed Multi-Robot Exploration under Complex Constraints

Escuela de Doctorado
Universidad Pablo de Olavide
Crta. Utrera, km. 1
41013 Seville, Spain

Distributed Multi-Robot Exploration under Complex Constraints

Tesis Doctoral

para la obtención del título de
Doctor por la Universidad Pablo de Olavide
Sevilla, 5 de abril de 2018

por

Alberto Viseras

Directores:
Dmitriy Shutin, German Aerospace Centre
Luis Merino, Universidad Pablo de Olavide

Acknowledgments

A Michael, por introducirme en el mundo de la robótica y la exploración con múltiples robots.

A Luis, por los ánimos, las discusiones técnicas y por confiar desde el principio en la idea que presentamos en esta tesis.

A Uwe, Dmitriy y mis compañeros en DLR, por su apoyo y por hacerme el día a día tan sencillo.

A los estudiantes que he tenido la oportunidad de supervisar, por transmitirme su motivación y por ayudarme a replantearme mis ideas.

A mi familia, por apoyarme durante todo el proceso, a pesar de no haber llegado a entender nunca lo que pretendía resolver en esta tesis.

A cosiii, por estar siempre ahí y, ante todo, por aguantarme y no echarme de casa.

Múnich Alberto Viseras
Febrero de 2018.

Abstract

Mobile robots have emerged as a prime alternative to explore physical processes of interest. This is particularly relevant in situations that have a high risk for humans, like e.g. in search and rescue missions, and for applications in which it is desirable to reduce the required time and manpower to gather information, like e.g. for environmental analysis. In such context, exploration tasks can clearly benefit from multi-robot coordination. In particular, distributed multi-robot coordination strategies offer enormous advantages in terms of both system's efficiency and robustness, compared to single-robot systems. However, most state-of-the-art strategies employ discretization of robots' state and action spaces. This makes them computationally intractable for robots with complex dynamics, and limits their generality. Moreover, most strategies cannot handle complex inter-robot constraints like e.g. communication constraints.

The goal of this thesis is to develop a distributed multi-robot exploration algorithm that tackles the two aforementioned issues. To achieve this goal we first propose a single-robot myopic approach, in which we build to develop a non-myopic informative path planner. In a second step, we extend our non-myopic single-robot algorithm to the multi-robot case. Our proposed algorithms build on the following techniques: (i) Gaussian Processes (GPs) to model the spatial dependencies of a physical process of interest; (ii) sampling-based planners to calculate feasible paths; (iii) information metrics to guide robots towards informative locations; and (iv) distributed constraint optimization techniques for multi-robot coordination.

We validated our proposed algorithms in simulations and experiments. Specifically, we carried out the following experiments: mapping of a magnetic field with a ground-based robot, mapping of a terrain profile with two quadcopters equipped with an ultrasound sensor, and exploration of a simulated wind field with three quadcopters. Results demonstrate the effectiveness of our approach to perform exploration tasks under complex constraints.

En la actualidad, los robots móviles son una de las principales alternativas para explorar procesos físicos de interés. Una de las razones más relevantes de esta elección es su utilización en aplicaciones que suponen un alto riesgo para los humanos, como por ejemplo en misiones de búsqueda y rescate. Además, su uso brinda otras ventajas como la reducción del tiempo y recursos humanos necesarios para obtener información, por ejemplo, en el caso de análisis medioambiental. En este contexto, las tareas de exploración se pueden claramente beneficiar de la coordinación entre múltiples robots. En particular, las estrategias distribuidas de coordinación presentan grandes ventajas en términos de la eficiencia y robustez del sistema, en comparación con las estrategias que emplean un único robot. Sin embargo, la mayoría de las estrategias de la literatura emplean una discretización del espacio de estados y acciones del robot, lo cual las hace computacionalmente irresolubles y limita su generalidad. Además, la mayor parte de las estrategias no es capaz de lidiar con restricciones complejas entre robots como, por ejemplo, restricciones de comunicación.

El objetivo de esta tesis es desarrollar un algoritmo de exploración distribuido para múltiples robots que sea capaz de solucionar los dos problemas mencionados anteriormente. Para lograr este objetivo, primero proponemos una estrategia miópica para un único robot, en la cual nos basamos para desarrollar un planificador de caminos informativo no miópico. En un segundo paso, extendemos nuestro algoritmo no miópico de un sólo robot para considerar múltiples robots. El algoritmo que proponemos se basa en las siguientes técnicas: (i) procesos Gaussianos (GPs) para modelar las dependencias espaciales del proceso físico de interés; (ii) planificadores por muestreo para calcular posibles caminos; (iii) métricas de información para dirigir a los robots hacia localizaciones informativas; y (iv) técnicas distribuidas de optimización con restricciones para lograr la coordinación entre múltiples robots.

En esta tesis hemos validado los algoritmos propuestos tanto en simulaciones como en experimentos. En particular, hemos llevado a cabo los siguientes experimentos: mapeado de la intensidad de un campo mágnetico con un robot de superficie, mapeado de un perfil de altura con dos drones equipados con sensores de ultrasonido, y exploración de un campo de viento simulado con tres drones. Los resultados obtenidos nos han permitido demostrar la efectividad de nuestro método para llevar a cabo tareas de exploración bajo restricciones complejas.

Contents

List of Figures

List of Tables

Acronyms

ACO	Ant Colony Optimization.
DCOP	Distributed Constraint Optimization.
DFS	Depth-First Search.
DLR	Deutsches zentrum für Luft- und Raumfahrt (German Aerospace Center).
GP	Gaussian Process.
LML	Log-Marginal-Likelihood.
MI	Mutual Information.
RL	Reinforcement Learning.
RMSE	Root Mean Squared Error.
ROS	Robot Operating System.
RRT	Rapidly exploring Random Trees.
SE	Squared Exponential covariance function.

Notation

A	random variable.
$\mathcal{A} \setminus \mathcal{B}$	elements of set $\mathcal{A}$ that are not part of $\mathcal{B}$.
$\mathcal{A}$	set/graph.
$\mathcal{N}_{\mathcal{A}}(\mathbf{a})$	neighborhood of a node $\mathbf{a}$ in graph $\mathcal{A}$.
$\mathbf{A}$	matrix.
$\mathbf{a}$	vector.
a	constant.
$\lvert\mathcal{A}\rvert$	cardinality of set $\mathcal{A}$.

List of Symbols

Decision maker

N_p	parameter that sets the maximum number of iterations that we let a planning algorithm run.
η	parameter that sets the maximum branch size in RRT/RRT* algorithms.
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	planning graph (tree), with a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$.
$\mathcal{L}_{\mathcal{X}_{free}}(\mathcal{V}_{\mathcal{X}_{free}}, \mathcal{E}_{\mathcal{X}_{free}})$	lattice graph defined over $\mathcal{X}_{free}$, with $\mathcal{V}_{\mathcal{X}_{free}}$ the set of vertices and $\mathcal{E}_{\mathcal{X}_{free}}$ the set of edges.
$\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$	path that consists of an ordered list of waypoints from $\mathbf{x}_A$ to $\mathbf{x}_B$; $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} = \{\mathbf{x}_A, ..., \mathbf{x}_i, ..., \mathbf{x}_B\}$, with $\mathbf{x}_i \in \mathcal{X}_{free}$.

Gaussian processes

$\mathbf{\Sigma}_*$	covariance matrix of the predictive distribution that results from the Gaussian processes regression.
$\boldsymbol{\mu}_*$	mean vector of the predictive distribution that results from the Gaussian processes regression.

$\boldsymbol{\theta}_*$	optimal hyperparameters of a Gaussian process that better fit the training data.
$\boldsymbol{\theta}$	hyperparameters that define a Gaussian process covariance function, $\boldsymbol{\theta} = [\sigma_f^2, l, \sigma_n^2]^T$.
σ_f^2	hyperparameter that models the variations in the magnitude of the physical process.
σ_n^2	hyperparameter that models the sensor's noise.
$\mathbf{m}(\mathbf{x})$	mean function of the Gaussian processes for regression.
$k(\mathbf{x}, \mathbf{x}')$	covariance function of the Gaussian processes for regression defined between positions $\mathbf{x}$ and $\mathbf{x}'$.
l	hyperparameter that models the length scale of the spatial correlation.

Information metrics

$H(X)$	entropy of random variable X.
$H(X, Y)$	joint entropy of random variables X and Y.
$H(X\|Y)$	conditional entropy of random variable X given a random variable Y.
$I(X; Y)$	mutual information of random variables X and Y.

Informative path planning

$I(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$	measure of the expected information, according to a pre-specified information metric, at $\mathcal{P}_{\mathbf{x},\mathbf{x}'}$.
$I(\mathbf{x})$	measure of the expected information, according to a pre-specified information metric, at $\mathbf{x}$.
$\mathbf{s}_*$	position $\mathbf{x} \in \mathcal{X}_{free}$ that is highly informative according to a pre-specified information metric.

b	trajectory budget.
$c(\mathcal{P}_{\mathbf{x},\mathbf{x}'}(\mathcal{G}))$	cost to reach sample $\mathbf{x}'$ from $\mathbf{x}$ following the tree $\mathcal{G}$.
$c(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$	cost of traversing path $\mathcal{P}_{\mathbf{x},\mathbf{x}'}$.
$f(I(\mathcal{P}_{\mathbf{x},\mathbf{x}'}), c(\mathcal{P}_{\mathbf{x},\mathbf{x}'}))$	function that evaluates an information-cost trade-off of $\mathcal{P}_{\mathbf{x},\mathbf{x}'}$.

Multi-robot system

N	number of robots in a multi-robot system.
$U(\cdot)$	global utility function that we aim to maximize.
$U_C(\cdot)$	constraint satisfaction utility that we aim to maximize.
$U_I(\cdot)$	information gathering utility that we aim to maximize.
$U_i(\cdot)$	utility of robot i.
$\mathcal{A}$	assignment of decision variables in max-sum, with $\mathcal{A}^*$ the optimal assignment.
$\mathcal{D}$	set of decision variables for which we aim to solve a multi-robot coordination problem. $\mathcal{D} = \{d_1, d_2, ..., d_N\}$, with d_i the decision variable of robot i.
$\mathcal{G}_c(\mathcal{V}_t, \mathcal{E}_t)$	network communication graph at time t, with vertices $\mathcal{V}_t$ and edges $\mathcal{E}_t$.
$\mathbf{x}_{r_i}$	robot's i current position, for a system of N robots, with $\mathbf{x}_{r_i} \in \mathcal{X}_{free}$ and $i \in [1, 2, ..., N]$.
k_c	maximum number of iterations we allow a robots' network to be disconnected.
r_c	communication radius. It specifies the maximum communication range between robots.
r_s	safety distance. It specifies the minimum inter-robot distance.

Physical process of interest

$Y_{\mathbf{X}'}$	random variable that represents a process $y(\mathbf{x})$, with $\mathbf{x} \in \mathcal{X}_{free}$, at positions $\mathbf{X}'$.

d_s	dimensionality of the environment in which a process of interest takes place.
$y(\mathbf{x})$	unobserved process a robot aims to explore at position $\mathbf{x}$.

Robot and sensor model

$\epsilon(\mathbf{x})$	noise factor at position $\mathbf{x}$, with $\epsilon(\mathbf{x}) \sim \mathcal{N}(0, \sigma_n^2)$ a gaussian probability density function of mean 0 and variance σ_n^2.
$\mathcal{X}_{free}$	free space in the robot's configuration space. It contains all states that are reachable by the robot considering the environment.
$\mathbf{f}_m(\cdot)$	robot's motion model, which relates the robot's current position $\mathbf{x}_r(t)$ and future position $\mathbf{x}_r(t + dt)$ given a control input $\mathbf{u}$: $\mathbf{x}_r(t + dt) = \mathbf{f}_m(\mathbf{x}_r(t), \mathbf{u})$.
$\mathbf{x}_r$	robot's current position, with $\mathbf{x}_r \in \mathcal{X}_{free}$.
n	number of measurements collected by a robot.
p	number of points where a robot aims to make a prediction. In our information gathering setting, p is the number of potential next measurements.
$\mathbf{X}_*$	matrix with each of the rows corresponding to a position where a robot aims to predict a physical process value.
$\mathbf{X}$	matrix that contains all positions where a robot collected measurements.
$\mathbf{z}$	magnitude of measurements taken at positions $\mathbf{X}$.

Part I

Introduction and Background

Chapter 1

Introduction

1.1 Motivation

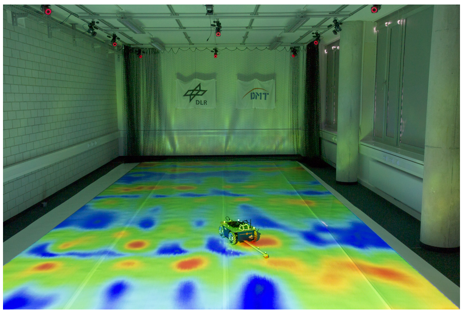
> ***2011***. *"Fukushima nuclear plant blast puts Japan on high alert. Warnings of possible meltdown amid radiation leaks. Tens of thousands evacuated after plant explosion. Up to* 1300 *killed in earthquake and tsunami" (theguardian, 2011).*
>
> ***2017***. *"Six years after Fukushima, robots finally find reactors' melted uranium fuel. The Japanese government and companies used radiation-hardened machines to search for the fuel that escaped the plant's ruined reactors" (nytimes, 2017).*

Fukushima was a major disaster that made us rethink about the use of nuclear energy. It also showed the world, and the robotics community in particular, that there is still an enormous gap between robotics research, and capabilities of robots to perform real-world missions. We would like to remark at this point that it took a team of experts *more than six years* to find Fukushima nuclear plant reactors. For that, experts used a mobile robot that was remotely controlled.

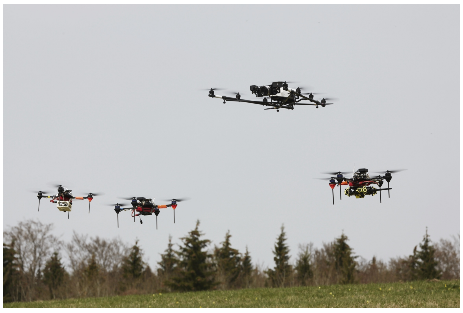
Mobile robots have emerged as a prime alternative to explore a physical process of interest, like e.g. fuel in Fukushima's accident, in situations that have a high risk for humans. Also mobile robots are becoming increasingly popular for applications in which it is desirable to reduce the required time and manpower to gather information of a physical process, like e.g. for prospecting or environmental analysis.

In this thesis, we focus on exploration/information gathering[1] methods for autonomous mobile robots. Especially in situations where the interaction of a hu-

[1]In this thesis we will employ the terms exploration and information gathering indistinctively.



(a) One robot measuring a magnetic field.



(b) Multiple robots measuring a wind field.

Figure 1.1: Examples of two physical processes of interest considered in the thesis. The pictures correspond to actual experiments that we performed to validate the algorithms proposed in this work.

man operator with a robot is difficult or impossible, e.g. in space exploration, or search and rescue missions, it is crucial that a robot is able to make autonomous decisions. While robots could follow brute-force systematic measuring strategies to explore a process of interest, we are motivated to derive algorithms that, without previous knowledge of the process, allow robots to obtain desired exploration results more rapidly, and/or with fewer resources. This may be economically advantageous in e.g. prospecting or environmental analysis applications, or even life-critical in search and rescue missions, like e.g. Fukushima's accident.

Besides autonomy, the information gathering task can clearly benefit from multi-robot coordination. Specifically, we are interested in distributed coordination strategies, as they offer enormous advantages in terms of both system's efficiency and robustness. However, most distributed coordination strategies employ discretization of the robots state and action spaces. This makes them computationally intractable for robotic systems with complex dynamics, and limits the generality of the strategies. In addition, most strategies cannot handle inter-robot constraints like e.g. communication constraints.

In this thesis we propose a solution that tackles the aforementioned issues - generality of the strategies, and handling inter-robot constraints. Our solution allows multiple robots to cooperate in order to autonomously gather information of a physical process. As a consequence, we expect to be one step closer to use mobile robots in real-world information gathering tasks, like the one from our motivational example of Fukushima's accident.

1.2 Gaussian Processes Based Exploration

In robotics, exploration covers a broad range of problems such as tracking, intruder detection, etc. (Thrun et al., 2005). In particular, in this thesis we focus on the exploration of physical processes that are spatially distributed. Examples of spatially distributed processes are fuel concentration, temperature, pollution, radioactivity, terrain profiles, magnetic fields (see Fig. 1.1a), wind fields (see Fig. 1.1b), etc. Gathering information of such processes is a fundamental task in a wide range of applications, such as:

- *prospecting*, where information about the terrain profile, and magnetic field is crucial;
- *environmental analysis*, where indicators such as temperature, pollution, fuel concentration, radioactivity, and wind are fundamental; or
- *search and rescue*, where knowledge about the terrain profile, and the temperature is needed to e.g. identify victims of a natural disaster with a thermal camera.

One of our prime goals is exploration efficiency. Therefore, we are interested in obtaining a map, i.e. a spatial representation of the process, with sufficient information in the least amount of time. To this end, we propose to model the process' spatial dependencies. One of the most popular methods to model spatial dependencies of a physical process are Gaussian Process (GP) for regression (Rasmussen and Williams, 2005), also referred as kriging in the field of geostatistics (Stein, 1999).

GPs are state-of-the-art for a broad range of information gathering tasks such as environmental monitoring (Marchant and Ramos, 2012), exploration of unknown environments (Jadidi et al., 2014), radio source localization (Fink and Kumar, 2010), real time traffic monitoring (Chen et al., 2012), persistent monitoring (Kai-Chieh et al., 2016), or autonomous soaring of an unpowered aerial glider (Chung et al., 2014). The applicability of GPs to a large class of problems is our fundamental motivation to use GPs as underlying model of a physical process of interest.

Before we specify the individual objectives that we want to achieve in this thesis, let us formulate this thesis' research problem.

1.3 Research Problem

We aim to explore an *a priori* unknown physical process with N cooperative robots autonomously, and as accurately as possible, in the sense of minimizing the difference between a process estimate and the (unknown) ground truth.

Additionally, the exploration task should be efficient provided the (i) available resources, and (ii) complex constraints such as inter-robot collision avoidance and communication constraints. To this end, we devise a decentralized coordination strategy that relies on the optimization of an information metric to control robots actions. Such strategies have been shown to be effective to reduce the difference between a process estimate and the (unknown) ground truth (Krause and Guestrin, 2007a; Hollinger and Sukhatme, 2014).

The aforementioned information gathering problem is subject to the following assumptions:

1. The physical process of interest can be modeled as a GP.
2. This process is time-invariant during the information gathering task. Time-variant processes could be considered within the proposed framework (Rasmussen and Williams, 2005). However, it is out of the scope of this thesis.
3. The process typically takes place in an environment populated with obstacles. The borders and obstacles that define the environment are *a priori* known. This assumption allows us to isolate the exploration of the physical process from the perception and mapping of the environment.
4. The robots' position is known exactly and noise-free. We assume that there exists an external positioning system that provides us with a highly accurate localization, e.g., a Global Positioning System for outdoor scenarios, or a motion tracking system for indoor environments. Uncertainty in positioning could also be accounted for using GPs (Muppirisetty et al., 2016), but it is out of the scope of this work.

The research problem formulated in this section corresponds to the final goal of this thesis. In order to solve the formulated research problem, we performed incremental steps towards our goal. In next section, we describe each of the steps in more detail.

1.4 Thesis Overview and Objectives

GPs allow us to model, and subsequently to exploit, the spatial dependencies of a physical process. In the literature, the exploitation of spatial dependencies with GPs focuses on two main aspects. First, exploitation in the sense of predicting the value of the process at not yet visited positions, with the aim of obtaining a map of the process with sufficient information as fast as possible. Second, exploitation in the sense of predicting the informativeness of potential measurement positions within the environment, with the aim of driving the robot to highly informative positions.

The informativeness of a potential measurement position is commonly measured via a so-called informativeness measure, also referred in this thesis as *information metric*. An information metric, together with an underlying model of the process of interest – GPs in this thesis, allows a robot to decide where to move next by weighting the informativeness of potential next positions. Here we distinguish two different approaches to select potential next positions: myopic and non-myopic approaches.

Myopic vs. non-myopic approaches. On the one hand, *myopic* approaches plan a potential next movement, without taking into account how this could affect subsequent movements. On the other hand, *non-myopic* approaches plan over a longer horizon. Non-myopic approaches generally result in a higher computational complexity. However, they outperform myopic approaches, as the future impact of robot's movement is considered (see Sec. 3.3).

In this thesis, we first propose myopic algorithms as a first step, and then extend them to non-myopic approaches. For the latter ones, we introduce efficient algorithms that can deal with the high computational requirements without sacrificing exploration performance. Non-myopic approaches require an additional component compared to myopic approaches: an action planner.

Action planners for exploration. Essentially, an action planner calculates an ordered set of potential positions – a path – that the robot could traverse, subject to the robot's kinematic constraints. Most of state-of-the-art works for information gathering with GPs employ discrete graph-based planners, see e.g. Dijkstra (Dijkstra, 1959) or A* (Hart et al., 1968) algorithms, to plan the robots motion. However, discrete graph-based planners present two major limitations. First, they are not suited to robots that have a large state space, typically consisting of more than four states (LaValle and Kuffner, 2001). Second, a discrete graph-based planner requires a discretization of the exploration environment. The first aspect limits the generalization of the algorithms to a reduced class of robots. Furthermore, the second aspect introduces an additional parameter to the algorithm: the discretization factor. Let us point out that an incorrect selection of the discretization factor could significantly reduce the algorithm's performance (LaValle and Kuffner, 2001).

The two aforementioned limitations motivate us to consider sampling-based planners as, in contrast to discrete graph-based planners, they do not require any discretization, and allow us to generalize the proposed algorithms to a larger class of robots.

Single-robot exploration. The three main elements of the model-based information gathering algorithms proposed in this thesis are:

1. *GPs* to model the spatial dependencies of a physical process of interest,
2. a *sampling-based action planner* to calculate paths that are feasible given a robot's kinematic constraints, and
3. an *information metric* to guide a robot towards highly informative locations.

Previous works such as (Hollinger and Sukhatme, 2014; Yang et al., 2013) considered the combination of the three aforementioned elements to perform exploration with a single robot. However, the current literature lacks an algorithm that considers those three elements to gather information with multiple cooperative robots.

Multi-robot exploration. Information gathering with multiple robots offers two main advantages compared to a single-robot system, as pointed out in (Burgard et al., 2000). First, by means of parallelization, as the exploration tasks could be splitted between robots. Second, in terms of robustness, as the tasks of a faulty robot could be overtaken by the rest of the team. However, the development of a multi-robot information gathering system presents an additional challenge: the inclusion of complex multi-robot constraints. In particular, in this thesis we focus on inter-robot collision avoidance, and inter-robot communication constraints. In the literature, most of the algorithms do not consider such constraints (see Sec. 3.4 for an overview), which prohibits those algorithms to be transferred from simulations to real world experiments.

We can distinguish two different architectures to exploit the advantages, as well as to deal with the challenges, of a multi-robot system. The most common approach in the literature is a centralized architecture (see Sec. 3.4 for related works). A centralized architecture consists of the combination of multiple identical robots plus a "central" robot. The central robot typically collects information from the other robots, and coordinates the subsequent robots' movements. In this thesis, we go one step forward. To the best of our knowledge, we propose the first non-myopic decentralized coordination architecture for exploration that can deal with (i) complex multi-robot constraints, and (ii) robots whose motion can be planned with a sampling-based planner. A decentralized architecture, in contrast to a centralized one, does not require a "central" robot. This implies that coordination is performed by means of local communication between neighboring robots.

Experimental evaluation. We validate our proposed algorithms in simulations, as well as in experiments with robots in the loop that consider either simulated or actual sensors, depending on the concrete experiment. In particular, we performed the following experiments:

- mapping of a magnetic field intensity in an indoor environment with a ground-based robot (see Fig. 1.1a), fundamental for e.g. indoor positioning tasks (Angermann et al., 2012),
- mapping of a terrain profile with two quadcopters equipped with an ultrasound sensor, relevant in e.g. agriculture (Robinson et al., 2009), and
- exploration of a simulated wind field with three quadcopters (see Fig. 1.1b), crucial to e.g. autonomous soaring (Lawrance and Sukkarieh, 2011).

Summary. In this thesis, we start proposing a myopic discrete graph-based single-robot exploration algorithm. Then, we analyze its performance and evolve the algorithm in subsequent chapters to incorporate non-myopic planners, sampling-based planners, multiple robots, and inter-robot constraints. We finalize with a non-myopic sampling-based decentralized multi-robot exploration algorithm that deals with complex constraints.

In addition to the cited contributions that are purely related to information gathering, we also developed algorithms that contribute to the state of the art in path planning. Path planning is a fundamental task for exploration algorithms. These contributions are included in the appendices.

1.5 Thesis Contributions

We classified in Fig. 1.2 the content of the thesis according to the number of robots, and decision makers employed in each of the chapters. Next we list the main contribution of each of the chapters, together with the publications that resulted out of each chapter.

- In *Chapter 2* we introduce the three fundamental methods that we will employ in this work: (i) GPs model for spatial data, (ii) information metrics for exploration, and (iii) decision makers.
- In *Chapter 3* we present an overview of the state-of-the-art in information gathering with GPs.
- In *Chapter 4* we employ a graph-based myopic single-robot algorithm to explore a magnetic field intensity in an indoor environment. To the best

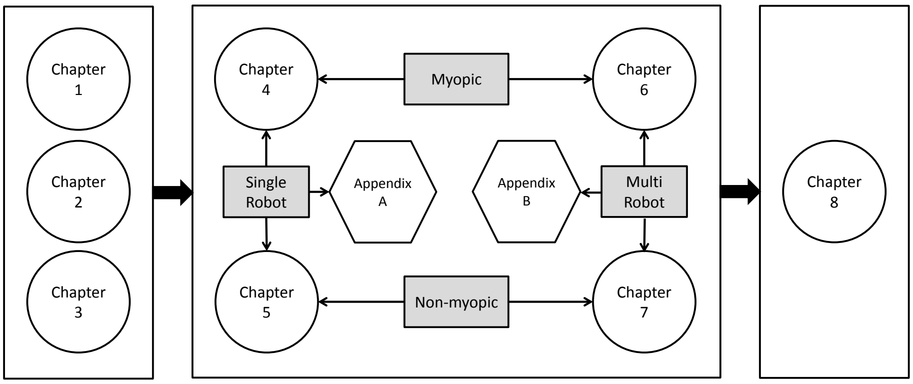


Figure 1.2: Classification of the thesis content according to the number of robots, and the nature of the decision maker employed in each chapter.

of our knowledge, this is the first algorithm that explores a magnetic field intensity in an indoor environment. Results of this chapter were published in the following papers:

- Alberto Viseras, Michael Angermann, Iris Wieser, Martin Frassl, and Joachim Mueller. Efficient multi-agent exploration with Gaussian processes. In *Robotics and Automation (ACRA), 2014 Australasian Conference on*, 2014
- Alberto Viseras, Thomas Wiedemann, Christoph Manss, Lukas Magel, Joachim Mueller, Dmitriy Shutin, and Luis Merino. Decentralized multi-agent exploration with online-learning of Gaussian processes. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4222–4229. IEEE, 2016b

• In *Chapter 5* we derive a sampling-based non-myopic single-robot algorithm that offers a nine-fold improvement respect to a greedy and a random walk trajectories. Preliminary concepts related to non-myopic information gathering, in which this chapter builds, were published in:

- Alberto Viseras and Calin Olariu. A general algorithm for exploration with Gaussian processes in complex, unknown environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3388–3393. IEEE, 2015

Results of this chapter are published and under review in the following conference and journal, respectively:

  - Alberto Viseras, Dmitriy Shutin, and Luis Merino. Online information gathering using sampling-based planners and GPs: An information theoretic approach. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 123–130, Sept 2017b
  - Alberto Viseras, Dmitriy Shutin, and Luis Merino. Online information gathering using RRT-based planners and GPs. *Robotics and Autonomous Systems*, (under review) 2017a

- In *Chapter 6* we extend the algorithm proposed in Chapter 4 to handle multiple robots. The content of this chapter was published in the aforementioned papers (Viseras et al., 2014, 2016b).
- In *Chapter 7* we propose a sampling-based non-myopic algorithm for information gathering with multiple robots, which permits incorporating complex constraints such as inter-robot collisions and communication constraints. To the best of our knowledge, this is the first algorithm with the aforementioned features in the literature. The content of this chapter is accepted and under review in the following conference and journal, respectively:

  - Alberto Viseras, Zhe Xu, and Luis Merino. Distributed multi-robot cooperation for information gathering under communication constraints. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, (accepted) 2017
  - Alberto Viseras, Zhe Xu, and Luis Merino. Distributed multi-robot information gathering under complex constraints. *Autonomous Robots*, (under review) 2017b

  The theory and results presented in this chapter were obtained during a research stay at the Australian Centre for Field Robotics (ACFR) at the University of Sydney.

- In *Chapter 8* we present the conclusions of this thesis, as well as potential venues for future work.
- In *Appendix A* we introduce an improvement into the sampling process of the Rapidly exploring Random Trees (RRT)/RRT* algorithms. By incorporating an ant colony optimization algorithm, which learns the sampling distribution of RRT/RRT*, we obtain a gain of performance of factor 3.6 respect to the standard algorithms. This was published in the following journal:

- Alberto Viseras, Rafael Ortiz Losada, and Luis Merino. Planning with ants: Efficient path planning with rapidly exploring random trees and ant colony optimization. *International Journal of Advanced Robotic Systems*, 13(5):1729881416664078, 2016a

- In *Appendix B* we present, the first multi-robot RRT-based path planning algorithm that only requires local communication between neighboring robots. The content of this appendix was published in:

  - Alberto Viseras, Valentina Karolj, and Luis Merino. An asynchronous distributed constraint optimization approach to multi-robot path planning with complex constraints. In *Proceedings of the Symposium on Applied Computing*, pages 268–275. ACM, 2017a

Chapter 2

Methods

We describe in this chapter the three main tools that will be considered in this thesis. First, we give an overview of GPs regression to model spatial data in Section 2.1. Then, we discuss in Section 2.2 several methods to calculate a robot's trajectory. To finalize, we describe in Section 2.3 information metrics that exploit a GP model to evaluate the relevance of a robot's trajectory. These three techniques will then be combined in the next chapters to offer solutions to exploration tasks of increasing complexity.

2.1 Gaussian Process Model for Spatial Data

2.1.1 Signal and Sensor Models

Let us consider a signal $y(\mathbf{x}) \in \mathbb{R}$ that takes values in positions $\mathbf{x} \in \mathbb{R}^{d_s}$, where d_s is the dimensionality of the environment in which a process of interest takes place. For example, a signal could be a physical process like a magnetic field intensity within a two-dimensional environment, where each of positions $\mathbf{x}$ would correspond to a position of the environment. Typically, however, a signal is not observed directly, but is measured using some sensors. Here we assume a simple sensor model that represents a measured signal as:

$$z(\mathbf{x}) = y(\mathbf{x}) + \epsilon(\mathbf{x}), \tag{2.1}$$

where $z(\mathbf{x})$ is a signal sample, $y(\mathbf{x})$ is the unobserved signal value, and $\epsilon(\mathbf{x})$ is a random noise. In the following we will assume that, for different measurements, noise samples $\epsilon(\mathbf{x})$ are independent and identically distributed according to $\mathcal{N}(0, \sigma_n^2)$; i.e. they follow a Gaussian distribution with zero mean and variance σ_n^2.

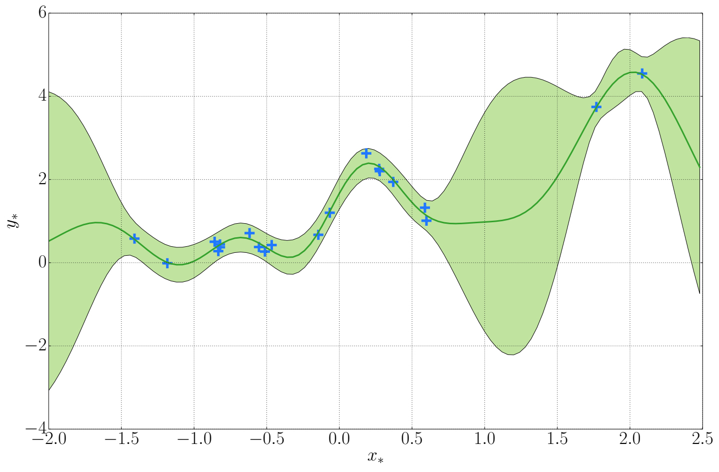


Figure 2.1: Example of GPs regression. Blue crosses represent measurements taken by a robot. Green bold line corresponds to the mean of the resulting prediction, and shaded area is the 95% confidence interval of the prediction.

2.1.2 Gaussian Processes for Regression

In this section we introduce basic theory about GPs for regression, and give some hints about their applicability in exploration tasks. Before explaining the details of GPs, we provide an example of GPs regression for a one-dimensional space ($x \in \mathbb{R}$). This is illustrated in Figure 2.1. Blue crosses represent measurements taken by a robot, according to model (2.1). Given those measurements, we aim to predict the process value in the range $x \in [-2.0, 2.5]$. The green bold line corresponds to the mean of the prediction calculated from the GPs regression, and green shaded area represents the uncertainty about the prediction. The green shaded area encloses predicted values that are within a range of two sigmas from the mean of the prediction, which corresponds to a 95% confidence interval. We observe that the predictions at locations that are close to the measurements locations have a lower uncertainty than those that are in areas where we have not measured yet. This property gives us a first hint about how to exploit GPs regression for exploration tasks, since our goal in exploration is typically to reduce uncertainty about our model, under the assumption that the model is representative of the physical phenomena of interest.

GPs notation. We now state GPs regression formally. Essentially, a GP is a collection of random variables, any finite number of which have a joint multivariate Gaussian distribution (Rasmussen and Williams, 2005). As such, it is fully specified by:

- a mean function $\mathbf{m}(\mathbf{x})$ that encodes the average value of $y(\mathbf{x})$. Without loss of generality, here we assume that $\mathbf{m}(\mathbf{x})$ is set to zero, which implies an absence of prior values of the observed process. Alternative mean functions could be used as suggested in (Rasmussen and Williams, 2005); and
- a covariance function $k(\mathbf{x}, \mathbf{x}')$ for any given positions $\mathbf{x}$ and $\mathbf{x}'$. $k(\mathbf{x}, \mathbf{x}')$ encodes information about the shape and structure we expect $y(\mathbf{x})$ to have. $k(\mathbf{x}, \mathbf{x}') = f(\mathbf{x}, \mathbf{x}', \boldsymbol{\theta})$ is a function of $\mathbf{x}$ and $\mathbf{x}'$, and model hyperparameters $\boldsymbol{\theta}$. Hyperparameters $\boldsymbol{\theta}$ define the GPs model and completely specify its properties.

Now, let us make the following definitions:

- $\mathbf{X} = \begin{bmatrix}\mathbf{x}^{[1]} & \mathbf{x}^{[2]} & \cdots & \mathbf{x}^{[n]}\end{bmatrix}^T$ is a matrix with n rows corresponding to spatial locations where the robot has taken measurements. For example, in Figure 2.1 matrix $\mathbf{X}$ would correspond to the vector $\mathbf{x} = \begin{bmatrix}-1.4 & 1.2 & \cdots & 2.1\end{bmatrix}^T$.
- $\mathbf{z} = \begin{bmatrix}z^{[1]} & z^{[2]} & \cdots & z^{[n]}\end{bmatrix}^T$ are the corresponding n measurements (e.g. $\mathbf{z} = \begin{bmatrix}0.4 & 0 & \cdots & 4.4\end{bmatrix}^T$ in Figure 2.1).
- $\mathbf{X}_* = \begin{bmatrix}\mathbf{x}_*^{[1]} & \mathbf{x}_*^{[2]} & \cdots & \mathbf{x}_*^{[p]}\end{bmatrix}^T$ is a matrix with p rows corresponding to "probe" locations – points in space where we predict the process value using the learned model. In Figure 2.1, $\mathbf{X}_*$ would correspond to an uniformly sampled subset of all positions in the horizontal axis.

Furthermore, using the covariance function $k(\mathbf{x}, \mathbf{x}')$ we define

$$\begin{aligned}
\mathbf{K} &= \begin{bmatrix} k(\mathbf{x}^{[1]}, \mathbf{x}^{[1]}) & \cdots & k(\mathbf{x}^{[1]}, \mathbf{x}^{[n]}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^{[n]}, \mathbf{x}^{[1]}) & \cdots & k(\mathbf{x}^{[m]}, \mathbf{x}^{[n]}) \end{bmatrix}, \\
\mathbf{K}_* &= \begin{bmatrix} k(\mathbf{x}^{[1]}, \mathbf{x}_{i*}^{[1]}) & \cdots & k(\mathbf{x}^{[1]}, \mathbf{x}_{i*}^{[p]}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}^{[n]}, \mathbf{x}_{i*}^{[1]}) & \cdots & k(\mathbf{x}^{[n]}, \mathbf{x}_{i*}^{[p]}) \end{bmatrix}, \qquad (2.2) \\
\mathbf{K}_{**} &= \begin{bmatrix} k(\mathbf{x}_{i*}^{[1]}, \mathbf{x}_{i*}^{[1]}) & \cdots & k(\mathbf{x}_{i*}^{[1]}, \mathbf{x}_{i*}^{[p]}) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_{i*}^{[p]}, \mathbf{x}_{i*}^{[1]}) & \cdots & k(\mathbf{x}_{i*}^{[p]}, \mathbf{x}_{i*}^{[p]}) \end{bmatrix}.
\end{aligned}$$

We would like to stress that $\mathbf{K}$, $\mathbf{K}_*$, and $\mathbf{K}_{**}$ are all functions of the hyperparameters $\boldsymbol{\theta}$, and of sampling locations $\mathbf{X}$ and $\mathbf{X}_*$. Yet we leave this dependency implicit in the following to simplify the notation.

GPs regression. Given $\mathbf{z}$ and $\mathbf{X}$, we can now predict the process value $\mathbf{y}_*$ and the corresponding uncertainty at probe locations $\mathbf{X}_*$. The elements in $\mathbf{y}_*$ are distributed according to: $p(\mathbf{y}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{z}) = \mathcal{N}(\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$. The mean vector $\boldsymbol{\mu}_*$ and the covariance matrix $\boldsymbol{\Sigma}_*$ of the predictive distribution are calculated as (Rasmussen and Williams, 2005):

$$\begin{aligned}\boldsymbol{\mu}_* &= \mathbf{m}(\mathbf{X}_*) + \mathbf{K}_*^T\mathbf{K}^{-1}(\mathbf{z} - \mathbf{m}(\mathbf{X})),\\ \boldsymbol{\Sigma}_* &= \mathbf{K}_{**} - \mathbf{K}_*^T\mathbf{K}^{-1}\mathbf{K}_*.\end{aligned} \tag{2.3}$$

GPs covariance function. The effectiveness of regression strongly depends on the covariance function $k(\mathbf{x}, \mathbf{x}')$. Specifically, we employ a Squared Exponential covariance function (SE) because of its ability to model smooth processes, as the ones we often aim to explore. SE is stationary and isotropic. Let us also note that the proposed scheme is not restricted to SE; spatially non-stationary non-isotropic covariances or other choices (Rasmussen and Williams, 2005) can be easily incorporated in the proposed scheme.

The definition of SE relies on the notion of similarity, which implies that we expect that closer points are more likely to be similar. For SE, a measure of similarity is merely a form of Euclidian distance between two points $\mathbf{x}$, $\mathbf{x}'$. Specifically:

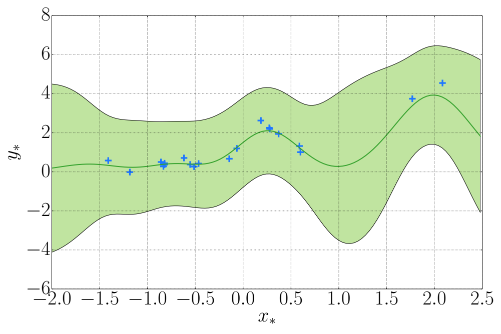
$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \cdot \exp\left(\frac{-\left|\left|\mathbf{x} - \mathbf{x}'\right|\right|^2}{2l^2}\right) + \sigma_n^2\delta_{\mathbf{x}\mathbf{x}'}, \tag{2.4}$$

where $\delta_{\mathbf{x}\mathbf{x}'} = 1$ iff $\mathbf{x} = \mathbf{x}'$ is the Kronecker's delta, and $\boldsymbol{\theta} = [\sigma_f^2, l, \sigma_n^2]^T \in \mathbb{R}_{>0}$.

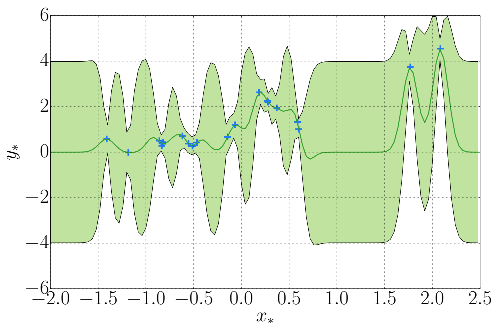
In the following, we give a physical interpretation of $\boldsymbol{\theta}$ and analyze their impact in the regression:

- σ_f^2 is a scaling factor that models the variations in the magnitude of the physical process; i.e. a process whose magnitude has a slow rate of variation will be modeled with a small σ_f^2.
- l models the covariance between variables separated by a certain distance. A high value of l means that variables that are separated a large distance are correlated. This factor is used to trade-off between model complexity and representability of the data.
- σ_n^2 models the process noise, and allows us to account for the measurements noise caused by the sensors.

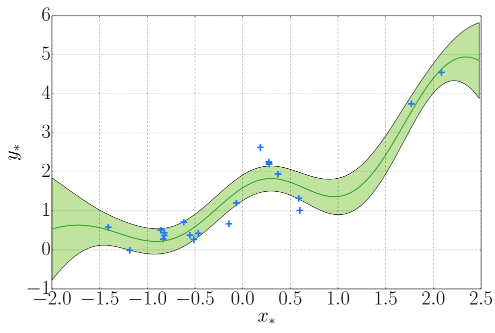
We showed in Figure 2.1 the result of GPs regression employing the optimal hyperparameters that better fit the training data (blue crosses). These hyperparameters corresponds to $\sigma_f^2 = 3.39, l = 0.37, \sigma_n^2 = 0.02$. Now, in order to



(a) $\sigma_f^2 = 3.39, l = 0.37, \sigma_n^2 = 1$.



(b) $\sigma_f^2 = 3.39, l = 0.08, \sigma_n^2 = 0.02$.



(c) $\sigma_f^2 = 3.39, l = 1, \sigma_n^2 = 0.02$.

Figure 2.2: Impact of the model's hyperparameters. Blue crosses represent the measurements taken by the robot. Green bold line corresponds to the mean of the resulting prediction, and shaded area is the 95% confidence interval of the prediction.

understand the impact of the model's hyperparameters, we show in Figure 2.2 the result of the regression when we modify the values of l and σ_n^2. We fix σ_f^2, since it is merely a scaling factor. We can observe in Figure 2.2a that the mean of the prediction becomes smoother when we increase the process noise σ_n^2. In Figures 2.2b and 2.2c we evaluate the impact of a small and a large l parameter, respectively. We notice that for a small value of l the correlation is restricted to neighboring locations. Therefore, the uncertainty of the prediction heavily increases as we get farther from the training points. We observe the opposite behavior when we increase the value of l. Now the range in which measurements are correlated increases and the mean and variance of the prediction becomes smoother. Another important point is that the model's complexity is reduced as we increase l. However, the training data loses representability.

2.1.3 Learning of Gaussian Process Model

GPs represent a powerful method to perform regression of spatially correlated data. In addition they allow us to learn the process model from the measured data. Given some measurements as training data, we can compute the hyperparameters $\boldsymbol{\theta}_*$ that better fit the collected measurements. This is achieved via the maximization of the Log-Marginal-Likelihood (LML) with respect to $\boldsymbol{\theta}$ (Rasmussen and Williams, 2005):

$$\boldsymbol{\theta}_* = \underset{\boldsymbol{\theta}_*}{\operatorname{argmax}}\, LML(\boldsymbol{\theta}|\mathbf{X}, \mathbf{z}), \tag{2.5}$$

with

$$LML(\boldsymbol{\theta}|\mathbf{X}, \mathbf{z}) = \log p(\mathbf{z}|\mathbf{X}, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{z}^T\mathbf{K}^{-1}\mathbf{z} - \frac{1}{2}\log|\mathbf{K}| - \frac{n}{2}\log 2\pi, \tag{2.6}$$

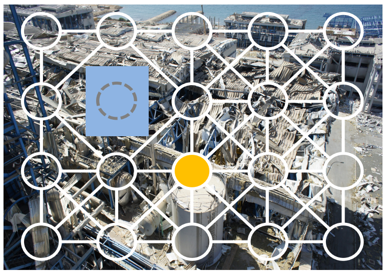
where matrix $\mathbf{K}$ is constructed by evaluating the covariance function, defined by the hyperparameters $\boldsymbol{\theta}$, at positions $\mathbf{X}$ (2.2). The LML is composed by three terms. The first term corresponds to the data-fit. The second term is a complexity penalty that decreases proportionally with the model's complexity. The third term is a normalization constant. The LML is a differentiable function and, therefore, multiple optimization methods such as gradient descent, conjugate gradients, newton method etc. could be used to find $\boldsymbol{\theta}_*$.

2.2 Decision Making

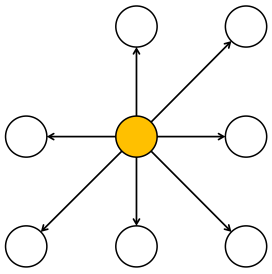
The second fundamental element that we employ in this thesis is the decision making component, which permits robots planning their next actions. In particular, we describe first in Section 2.2.1 a discrete graph-based myopic approach. This is followed by a sampling-based non-myopic approach in Section 2.2.2. For a more complete overview of planning methods we refer the reader to (LaValle, 2006).

2.2.1 Discrete Graph-Based Myopic Approach

The first alternative we consider in this thesis to plan robots actions is a discrete graph-based myopic approach. A graph-based approach relies on a graph representation of the robot's state space. We denote the graph representation as $\mathcal{L}(\mathcal{V}, \mathcal{E})$, with $\mathcal{V}$ the graph vertices, and $\mathcal{E}$ the graph edges. $\mathcal{V}$ contains all possible robot states, and edges contained in $\mathcal{E}$ represent transitions between two states; i.e. two vertices are linked with and edge iff a robot can transition between them. Specifically, in this thesis we define $\mathcal{L}(\mathcal{V}, \mathcal{E})$ as a lattice graph. This is a



(a) Environment graph's representation.



(b) Robot's planning graph.

Figure 2.3: Example of an environment's graph representation, and a robot's action graph for a myopic planning approach. Orange node represents a robot current position, and white nodes represent all possible next positions.

common approach in the literature that better suits holonomic robots, as they do not have kinematic constraints. Nevertheless, a lattice graph representation have been also employed to plan actions for robots that are subject to kinematic constraints, as in (Pivtoraiko et al., 2009).

Lattice graph. A lattice graph is typically defined over the full robot's state space. However, robots perform exploration tasks in environments populated with obstacles, which limits the exploration space to the free space in the robot's configuration space, denoted as $\mathcal{X}_{free}$. $\mathcal{X}_{free}$ contains all states that are reachable by the robot considering the environment. To make explicit that $\mathcal{L}(\mathcal{V}, \mathcal{E})$ is defined over $\mathcal{X}_{free}$, we denote the lattice graph as $\mathcal{L}_{\mathcal{X}_{free}}(\mathcal{V}_{\mathcal{X}_{free}}, \mathcal{E}_{\mathcal{X}_{free}})$. As an example, in Fig. 2.3a we depict a lattice graph representation of a holonomic robot that operates in a two-dimensional environment, where the blue square corresponds to an obstacle.

Myopic planning. Given a graph representation of the environment, we introduce next a myopic approach to select a robot's next action. A myopic approach is a particularization to decision making of a greedy paradigm. Greedy refers to an algorithmic paradigm that follows the heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy approach may yield locally optimal solutions that approximate a global optimal solution in a reasonable time. Specifically, for robot decision making, a greedy approach would select as next position one of the nodes in the robot's neighbourhood, according to a user-defined cost. Multiple costs could be considered according to the application of interest such as

Euclidian distance between two nodes, expected energy consumption while transitioning between two states, or any of the information metrics described latter in Section 2.3.

Let us assume a robot located at $\mathbf{x}_r$, and a cost $c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}})$ associated to $\mathcal{P}_{\mathbf{x}_r,\mathbf{x}}$, with $\mathcal{P}_{\mathbf{x}_r,\mathbf{x}}$ a straight line between $\mathbf{x}_r, \mathbf{x}$; and $\mathbf{x} \in \mathcal{N}_{\mathcal{L}_{\mathcal{X}_{free}}}(\mathbf{x}_r)$ the node's $\mathbf{x}_r$ neighbourhood in graph $\mathcal{L}_{\mathcal{X}_{free}}$. A myopic approach would select the robot's next position $\mathbf{x}_{next}$ as:

$$\mathbf{x}_{next} = \underset{\mathbf{x} \in \mathcal{N}_{\mathcal{L}_{\mathcal{X}_{free}}}(\mathbf{x}_r)}{\operatorname{argmin}} \; c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}}). \tag{2.7}$$

An example of a robot's myopic motion graph that corresponds to Figure 2.3a is depicted in Figure 2.3b. Orange node represents $\mathbf{x}_r$, white nodes represent $\mathbf{x} \in \mathcal{N}_{\mathcal{L}_{\mathcal{X}_{free}}}(\mathbf{x}_r)$, and arrows indicate the direction of movement. As we can observe, a myopic approach is restricted for this particular situation to seven possible movements that the robot can perform in the current iteration. For example, for a robot located in one corner of the environment, there exists only three possible movements.

2.2.2 Sampling-Based Non-Myopic Actions Planning: RRTs

Limitations of graph-based planners (discussed in Section 1.1), regarding the dimensionality of the robot's state space and the need of an environment discretization, motivates us to extend exploration algorithms to handle a robot's continuous motion and arbitrary robot kinematics. Sampling-based path planning algorithms (LaValle and Kuffner, 2001; Kavraki et al., 1996; Karaman and Frazzoli, 2011) are natural candidates for solving such problems. Specifically, we propose the use of RRT (LaValle and Kuffner, 2001) and RRT* (Karaman and Frazzoli, 2011) for exploration tasks. In the following we briefly review the contents of (LaValle and Kuffner, 2001; Karaman and Frazzoli, 2011).

RRT algorithm. The RRT algorithm is a solution to the path planning problem in complex high dimensional spaces (LaValle and Kuffner, 2001). The RRT algorithm iteratively constructs a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ (tree), with a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$, with the goal of establishing a path between an initial state $\mathbf{x}_A$ and a goal state $\mathbf{x}_B$ in the state space - a feasible trajectory $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$. $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$ corresponds to an ordered list of waypoints:

$$\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} = \{\mathbf{x}_A, ..., \mathbf{x}_i, ..., \mathbf{x}_B\} \tag{2.8}$$

with $\mathbf{x}_i \in \mathcal{X}_{free}$.

Algorithm 2.1. RRT-Path Planner($\mathbf{x}_A, \mathbf{x}_B, N_p, \mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{X}_{free}$)

```
1: $\mathcal{V} \leftarrow \{\mathbf{x}_A\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \leftarrow \emptyset$;
2: for $i = 1, \dots, N_p$ do
3:     $\mathbf{x}_{rand} \leftarrow$ SampleFree($\mathcal{X}_{free}$);
4:     $\mathbf{x}_{nearest} \leftarrow$ Nearest($\mathbf{x}_{rand}, \mathcal{V}$);
5:     $\mathbf{x}_{new} \leftarrow$ Steer($\mathbf{x}_{nearest}, \mathbf{x}_{rand}, \eta$);
6:     if CollisionFree($\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{new}}, \mathcal{X}_{free}$) then
7:         $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{x}_{new}\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{new}}\}$;
8:     $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \leftarrow$ FindBestPath($\mathbf{x}_A, \mathbf{x}_B, \mathcal{G}$);
9: return $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$;
```

The key steps of the RRT algorithm are summarized in Algorithm 2.1. In the following we explain Alg. 2.1 in more details. First, we draw a sample $\mathbf{x}_{rand}$ randomly from an uniform distribution defined over $\mathcal{X}_{free}$ using function `SampleFree`. Then the `Nearest` function finds the nearest neighbor (in terms of the cost-to-reach) of $\mathbf{x}_{rand}$ from the set of vertices $\mathcal{V}$. We use the function `Steer` to simulate driving the robot from $\mathbf{x}_{nearest}$ to $\mathbf{x}_{rand}$ according to our controller. We drive the robot a maximum distance η. This is a user-selected parameter that sets the maximum branch size. As output we obtain $\mathbf{x}_{new}$. If the trajectory $\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{new}}$ does not collide with any obstacles (line 6, Alg. 2.1), we add the vertex $\mathbf{x}_{new}$ and the edge $\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{new}}$ to $\mathcal{G}$. Given the current tree, we search the best path $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$, in terms of an user-defined cost metric, using function `FindBestPath`. In case there is no feasible path, the output would be a void set. We repeat this process during N_p iterations.

RRT* algorithm. The RRT* algorithm is an evolution of the RRT algorithm that finds asymptotically optimal solutions to the path planning problem (Karaman and Frazzoli, 2011). RRT* is a cost-based algorithm that allows a robot to plan actions that minimize a desired cost such as distance, energy consumption, or a process entropy. There exist alternative asymptotically optimal cost-based algorithms like e.g. T-RRT (Jaillet et al., 2008). However, we employ RRT* due to its wide adoption in the robotics community.

We describe the RRT* in Algorithm 2.2. It differs from RRT in two aspects: choosing a parent and rewiring. In contrast to RRT, RRT* chooses the parent of $\mathbf{x}_{new}$ as the node from the set $\mathcal{X}_{near}$ that allows us to reach $\mathbf{x}_{new}$ with the minimum cost. $\mathcal{X}_{near}$ is calculated using the function `Near` that is defined as

$$\texttt{Near}(\mathbf{x}, \mathcal{V}) := \left\{ \|\mathbf{x} - \mathbf{x}'\| \leq r(|\mathcal{V}|) \right\}, \tag{2.9}$$

with

$$r(|\mathcal{V}|) = \min\{\gamma(\log(|\mathcal{V}|)/|\mathcal{V}|)^{1/d_s}, \eta\}, \tag{2.10}$$

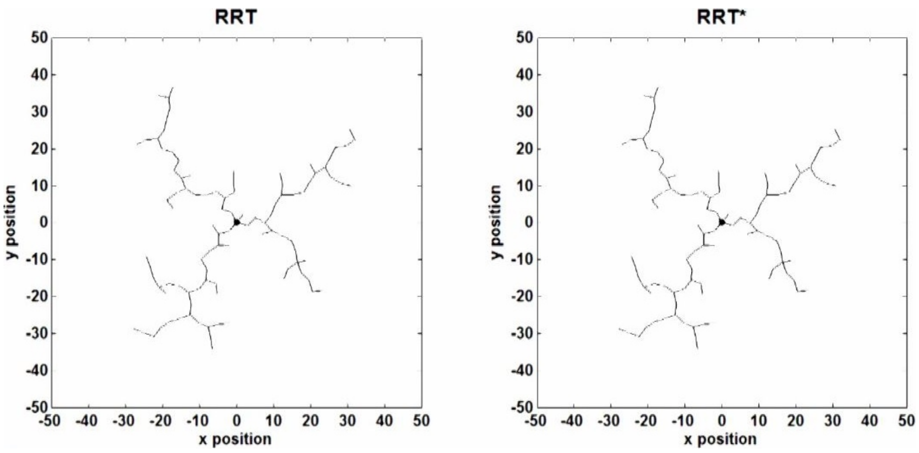
Algorithm 2.2. RRT*-Path Planner($\mathbf{x}_A, \mathbf{x}_B, N_p, \mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{X}_{free}$)

```
1: $\mathcal{V} \leftarrow \{\mathbf{x}_A\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \leftarrow \emptyset$;
2: for $i = 1, \ldots, N_p$ do
3:     $\mathbf{x}_{rand} \leftarrow \texttt{SampleFree}(\mathcal{X}_{free})$;
4:     $\mathbf{x}_{nearest} \leftarrow \texttt{Nearest}(\mathbf{x}_{rand}, \mathcal{V})$;
5:     $\mathbf{x}_{new} \leftarrow \texttt{Steer}(\mathbf{x}_{nearest}, \mathbf{x}_{rand}, \eta)$;
6:     if $\texttt{CollisionFree}(\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{new}}, \mathcal{X}_{free})$ then
7:         $\mathbf{x}_{min} \leftarrow \mathbf{x}_{nearest}$;
8:         $c_{min} \leftarrow c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_{nearest}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{new}})$;
9:         $\mathcal{X}_{near} \leftarrow \texttt{Near}(\mathbf{x}_{new}, \mathcal{V})$;
10:        for $\mathbf{x}_{near} \in X_{near}$ do                              ▷ choose parent
11:            $c_{new} \leftarrow c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_{near}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{near},\mathbf{x}_{new}})$;
12:            if $c_{new} < c_{min}$ then
13:                if $\texttt{CollisionFree}(\mathcal{P}_{\mathbf{x}_{near},\mathbf{x}_{new}}, \mathcal{X}_{free})$ then
14:                    $\mathbf{x}_{min} \leftarrow \mathbf{x}_{near}$; $c_{min} \leftarrow c_{new}$;
15:        $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{x}_{new}\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{P}_{\mathbf{x}_{min},\mathbf{x}_{new}}\}$;
16:        for $\mathbf{x}_{near} \in \mathcal{X}_{near}$ do                    ▷ rewire near nodes
17:            $c_{near} \leftarrow c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_{new}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{new},\mathbf{x}_{near}})$;
18:            if $c_{near} < c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_{near}}(\mathcal{G}))$ then
19:                if $\texttt{CollisionFree}(\mathcal{P}_{\mathbf{x}_{new},\mathbf{x}_{near}}, \mathcal{X}_{free})$ then
20:                    $\mathbf{x}_{parent} \leftarrow \texttt{Parent}(\mathbf{x}_{near})$;
21:                    $\mathcal{E} \leftarrow \mathcal{E} \setminus \{\mathcal{P}_{\mathbf{x}_{parent},\mathbf{x}_{near}}\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{P}_{\mathbf{x}_{new},\mathbf{x}_{near}}\}$;
22:    $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \leftarrow \texttt{FindBestPath}(\mathbf{x}_A, \mathbf{x}_B, \mathcal{G})$;
23: return $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$;
```

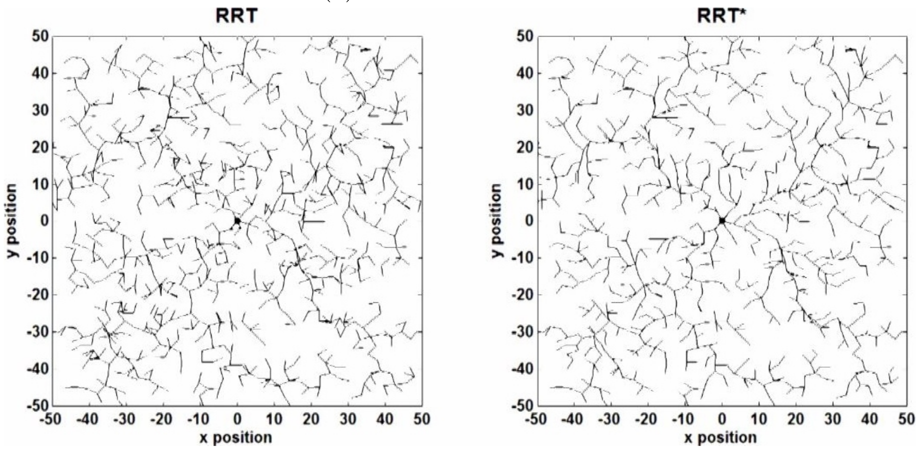
where $|\mathcal{V}|$ is the cardinality of set $\mathcal{V}$, γ is a constant. $r(|\mathcal{V}|)$ controls the tree branch's size. In particular, it reduces the branch's size as $\mathcal{V}$ grows and, as a consequence, paths are refined over time.

RRT* incorporates a rewiring process in order to find an optimal trajectory. This is done by finding minimum cost sub-paths. Here, we define two different costs: $c(\mathcal{P}_{\mathbf{x},\mathbf{x}'}(\mathcal{G}))$ is the cost to reach sample $\mathbf{x}'$ from $\mathbf{x}$ following the tree $\mathcal{G}$. $c(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ would be the cost of path $\mathcal{P}_{\mathbf{x},\mathbf{x}'} = [\mathbf{x}, \mathbf{x}']$. For example, a cost between two samples $c(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ could be defined as the Euclidean distance between them. In this case $c(\mathcal{P}_{\mathbf{x},\mathbf{x}'}(\mathcal{G}))$ would be the sum of these Euclidean distances along the edges towards $\mathbf{x}$. Alternative costs could be defined depending on the application, as we will show in next chapters. More information about the properties that a cost must meet can be found in (LaValle and Kuffner, 2001; Karaman and Frazzoli, 2011). Let us remark that RRT* requires that two states must be connected exactly during the rewiring process. This implies that the system must be controllable.

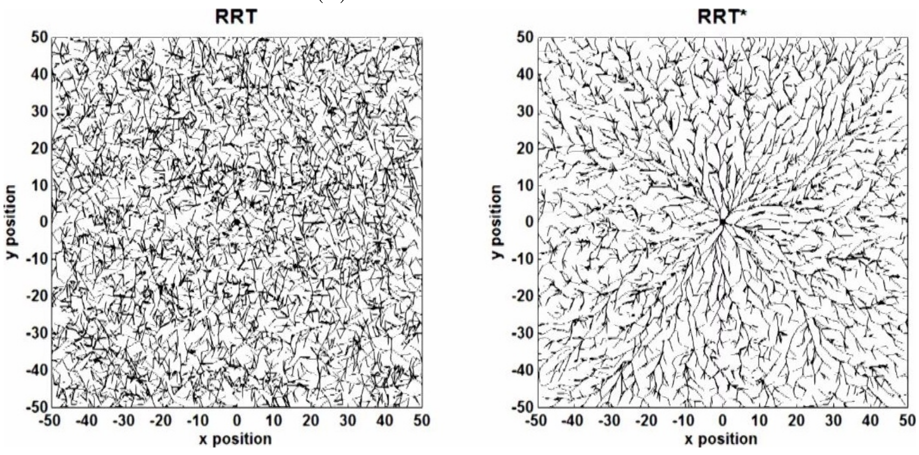
To finalize, we depict in Figure 2.4 the evolution of RRT and RRT* algo-



(a) Nodes number: 100.



(b) Nodes number: 1000.



(c) Nodes number: 5000.

Figure 2.4: Evolution of RRT and RRT* algorithms as we increase the number of iterations. These figures were extracted from (Yiqun Dong, 2015).

rithms as we increase the number of nodes. We consider an initial position $\mathbf{x}_A = \begin{bmatrix} 0 & 0 \end{bmatrix}$ and no goal; i.e. $\mathbf{x}_B$ is not defined. The total number of nodes is set to 5000, and we show the current tree at iterations 100, 1000 and 5000. We can observe that the tree created with RRT* algorithm is more structured than the one generated with RRT. This is due to the rewiring process that rewires branches to create minimum-cost subpaths.

2.3 Information Metrics for Exploration

One definition of exploration is "the act of searching for the purpose of discovery of information or resources". In a mobile robotics context, discovery of information typically translates in where to move next in order to obtain more information. Multiple criteria have been defined in the literature, as we indicate in Chapter 3, to decide where a robot should move next. In this thesis we focus on information-theoretic metrics (Cover and Thomas, 2012) to decide about a robot's next actions.

Information metrics have been widely employed in combination with GPs (see introduction in (Krause et al., 2008) for a detailed overview). In particular, we employ in this work the following information metrics: (i) differential entropy, and (ii) Mutual Information (MI); as they have been shown to perform well together with GPs (Krause and Guestrin, 2007a; Krause et al., 2008; Hollinger and Sukhatme, 2014). Next we highlight the most relevant properties of differential entropy and mutual information. For a deeper analysis, we refer the reader to (Cover and Thomas, 2012). Then we particularize in Section 2.3.3 the two metrics for random variables that are distributed as GPs.

2.3.1 Differential Entropy

Differential entropy is an extension of the idea of entropy from discrete to continuous probability distributions. Differential entropy of a random variable X with probability density function f whose support is a set $\mathcal{X}$ is defined as:

$$H(X) = -\int_{\mathcal{X}} f(x) \log f(x) dx. \tag{2.11}$$

A conditional entropy of X given a random variable Y with support $\mathcal{Y}$ and a joint probability density function $f(x, y)$ is given by:

$$H(X|Y) = -\int_{\mathcal{Y}} \int_{\mathcal{X}} f(x, y) \log f(x|y) dx dy. \tag{2.12}$$

Given the definitions of differential and conditional entropy, we can now review some relevant properties that we will use in the remainder of the thesis:

- Differential entropy $H(X)$ can take negative values, in contrast to entropy that only takes positive values.
- Conditioning never increases differential entropy: $H(X|Y) \leq H(X)$. In other words "information never hurts".
- Joint entropy $H(X,Y)$ can be expressed as $H(X,Y) = H(X|Y) + H(Y)$.
- The chain rule for differential entropy of p random variables $X^{[1]}, \dots, X^{[p]}$ is given by:

$$H(X^{[1]}, X^{[2]}, \dots, X^{[p]}) = \sum_{i=1}^{p} H(X^{[i]}|X^{[1]}, ..., X^{[i-1]}). \tag{2.13}$$

  Note that $H(X^{[1]}, \dots, X^{[p]}) \leq \sum_{i=1}^{p} H(X^{[i]})$, which implies that joint entropy is upper-bounded by individual entropies.

A final remark: in this work, we focus on random variables that are distributed as GPs, which implies that random variables are continuous. Therefore we will employ differential entropy as information metric; not entropy. For simplicity, we will use the terms differential entropy and entropy interchangeably.

2.3.2 Mutual Information

MI of two random variables is a measure of the interdependence between the two variables. More specifically, it quantifies the "amount of information" obtained about one random variable, through the other random variable. For two continuous random variables X, Y, MI is defined as:

$$I(X;Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} f(x,y) \log \frac{f(x,y)}{f(x)f(y)} dxdy. \tag{2.14}$$

The concept of MI is intrinsically linked to that of entropy of a random variable. Specifically, we can reformulate the previous definition to express MI in terms of entropies:

$$\begin{align} I(X;Y) &= H(Y) - H(Y|X) \tag{2.15} \\ &= H(X) - H(X|Y) \tag{2.16} \\ &= H(X) + H(Y) - H(X,Y). \tag{2.17} \end{align}$$

The relationship between MI and entropy can be better understood in terms of a Venn diagram. In Figure 2.5, we depict a Venn diagram that illustrates that relationship, where colors represent different elements of equations (2.15-2.17).

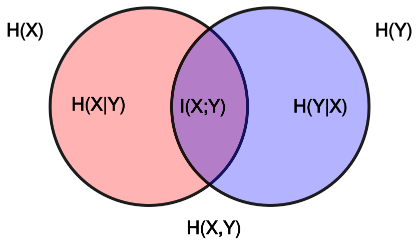


Figure 2.5: Venn diagram, from Wikipedia, the free encyclopedia (2017), for several information metrics associated with random variables X and Y. The area contained by both circles is the joint entropy $H(X, Y)$. The circle on the left (red and violet) is the individual entropy $H(X)$, with the red area being the conditional entropy $H(X|Y)$. The circle on the right (blue and violet) is the individual entropy $H(Y)$, with the blue area being the conditional entropy $H(Y|X)$. The violet area is the MI $I(X;Y)$.

Typically, in the context of information gathering we calculate MI given some prior knowledge. For example, some previously gathered measurements of the process we aim to explore, or some previous knowledge about the process properties. Calculation of MI given prior knowledge leads us to the concept of conditional MI. The conditional MI of two random variables X, Y given a random variable Z is given by:

$$I(X;Y|Z) = H(Y|Z) - H(Y|X,Z). \tag{2.18}$$

Let us now highlight some of the most relevant properties of MI:

- MI is always equal or greater than zero; i.e. $I(X;Y) \geq 0$.
- The chain rule for MI is given by:

$$I(X;Y;Z) = I(X;Y|Z) + I(X;Z). \tag{2.19}$$

- $I(X;Y)$ can in general be greater or lower than conditional MI $I(X;Y|Z)$. This is particularly relevant, since adding *a priori* knowledge does not necessarily translates into a decrease of MI; as opposed to entropy.

2.3.3 Differential Entropy and Mutual Information for GPs

Definitions and properties of entropy and MI, described in Sections 2.3.1 and 2.3.2, hold in general for any continuous random variable. Now let us particularize entropy and MI for the specific case we consider in this thesis: random variables that are distributed as GPs.

For example, let us imagine that we aim to calculate the entropy of a process at a single probe location $\mathbf{x}_*^{[1]}$ given some previously acquired measurements at positions $\mathbf{X}$. This would correspond to calculating entropy $H(Y_*|\mathbf{X})$, where Y_* is a Gaussian random variable that represents the process at $\mathbf{x}_*^{[1]}$. Then, entropy $H(Y_*|\mathbf{X})$ would be given by the following expression:

$$H(Y_*|\mathbf{X}) = \frac{1}{2}\log(2\pi e\sigma_*^2), \tag{2.20}$$

with σ_*^2 a variance calculated according to (2.3). Let us emphasize that we refer to σ_*^2 instead to $\mathbf{\Sigma}_*$ because we consider a single probe location. Also note that entropy is independent of the actual measurements values, given a GPs model. This is a property that will be exploited latter in the thesis.

Now let us extend previous definition to multiple probe locations. For example, this could be relevant to evaluate the entropy of a potential path composed of p waypoints. In this case, the random variable Y_* would correspond to a set of random variables $Y_* \triangleq \left\{Y_*^{[1]}, Y_*^{[2]}, \ldots, Y_*^{[p]}\right\}$, where $Y_*^{[i]}$, with $i = 1, 2, ..., p$, is a random variable associated to a process at a probe location $\mathbf{x}_*^{[i]}$. For multiple probe locations entropy is given by:

$$H(Y_*|\mathbf{X}) = \frac{1}{2}\log((2\pi e)^p|\mathbf{\Sigma}_*|), \tag{2.21}$$

with $|\mathbf{\Sigma}_*|$ being the determinant of covariance matrix $\mathbf{\Sigma}_*$ computed according to (2.3). Note that $H(Y_*|\mathbf{X})$ is independent of the order of variables in Y_* and $\mathbf{X}$.

An final remark: as MI can be expressed as a difference of entropies (2.15-2.17), we can easily compute MI for GPs with (2.20, 2.21).

Computational complexity. To finalize, we would like to add a few remarks about the computational complexity of the calculation of entropy and MI for GPs. Two most complex operations to calculate entropy are the matrix inversion required in (2.3), and calculation of determinant from (2.21). Both of them have a computational complexity that scales cubically with the matrix dimensions if we use standard methods; i.e. Gauss-Jordan elimination for the matrix inversion, and LU decomposition for the calculation of determinant. Let us also point out that faster methods could be considered, which would reduce computational complexity to a 2.373 factor. For simplicity, here we restrict to standard methods previously mentioned. This implies that computational complexity of a $n \times n$ matrix inversion and determinant is $\mathcal{O}(n^3)$. These two operations define the computational complexity of the information metrics we consider in this thesis. In Table 2.1, we summarize the computational complexity of several definitions

	Computational complexity
$H(Y_A)$	$\mathcal{O}(n_A^3)$
$H(Y_A\|Y_B)$	$\mathcal{O}(n_A^3 + n_B^3)$
$H(Y_A\|Y_B, Y_C)$	$\mathcal{O}(n_A^3 + (n_B + n_C)^3)$
$I(Y_A; Y_B)$	$\mathcal{O}(2n_A^3 + n_B^3)$
$I(Y_A; Y_B\|Y_C)$	$\mathcal{O}(2n_B^3 + 2n_C^3 + (n_A + n_C)^3)$

Table 2.1: Computational complexity of several definitions of entropy and MI for GPs. We employ the big O notation to represent the computational complexity. Note that we include scalars to exemplify the actual complexity and to differentiate entropy and MI.

of entropy and MI. Specifically, we consider as an example random variables Y_A, Y_B and Y_C containing n_A, n_B, n_C positions respectively. We have chosen the big O notation to represent the order of the computational complexity. According to Table 2.1, we can conclude that MI has a higher computational complexity than entropy because it requires the calculation of an entropy twice.

Chapter 3

Robotic Exploration using GPs

Exploration is one of the fundamental problems in mobile robotics. In this chapter we review the most relevant approaches, to the best of our knowledge, which have been proposed in the literature in the context of model-based exploration of spatially distributed physical processes. In particular, we focus here on processes than can be modeled as a GP. To facilitate the state-of-the-art review we classified the different approaches according to four categories: (i) underlying model of the process of interest, (ii) information metric that determines robots actions, (iii) planning algorithm that calculates robots' paths, and (iv) multi-robot architecture and inter-robot constraints that the algorithms can handle.

3.1 Model-based Exploration

A model that can accurately represent the information we aim to gather – process of interest – is crucial to perform an efficient exploration. Obviously, the better we are able to represent the process, the better we can exploit the gathered measurements to take decisions about robots' actions. Multiple models have been considered in the literature to represent physical processes. Examples of such models are: occupancy grid maps (Julian et al., 2014), a grid-based hierarchical decomposition of the environment (Dames et al., 2015), certainty grids together with information filters (Grocholsky et al., 2006), Gaussian mixture models (Merino et al., 2010), Markov random fields (Williams and Sukhatme, 2012), or GPs (Krause and Guestrin, 2007b). In this thesis we focus on the latter ones – GPs, as we pointed out in Sec. 1.1.

GPs hyperparameters learning. One of the fundamental aspects in GPs is the learning of the hyperparameters that characterize the GPs mean and covariance function (see Section 2.1.3). In the information gathering literature we can find two main approaches that deal with this problem. On the one hand, works

Approach	Employed by...
Pre-learned	Doo-Hyun et al. (2016), Kai-Chieh et al. (2016), Chen et al. (2015), Jadidi et al. (2015), Jadidi et al. (2014), Hollinger and Sukhatme (2014), Patten et al. (2013), Yang et al. (2013), Kemppainen et al. (2010), Stranders et al. (2010), Singh et al. (2009), Stranders et al. (2009), Krause et al. (2008), Low et al. (2008).
Online update	Ouyang et al. (2014), Marchant and Ramos (2012), Fink and Kumar (2010), Singh et al. (2010), Stranders et al. (2008), Krause and Guestrin (2007b).

Table 3.1: GPs-based exploration. The table classifies works according to how they deal with the hyperparameters that characterize the GPs mean and covariance function. In particular, two approaches are considered: approaches that assume the hyperparameters are pre-learned prior to the exploration task, and approaches that update the hyperparameters online as robots collect measurements.

such as (Singh et al., 2009; Hollinger and Sukhatme, 2014; Doo-Hyun et al., 2016) assume that the hyperparameters are known prior to the start of the information gathering task. This assumption implies that we posses some prior knowledge of the process we aim to explore. In contrast, works such as (Krause and Guestrin, 2007b; Marchant and Ramos, 2012; Ouyang et al., 2014) perform an online update of the hyperparameters. That is, the hyperparameters are updated as robots collect measurements, which allows robots to gather information without prior knowledge of the process of interest. Singh et al. (2010) go one step further and derive an algorithm that, in addition to performing an online update of the hyperparameters, selects a covariance function that better fits the measured data.

An online update of the hyperparameters is crucial to generalize algorithms to multiple information gathering applications. However, algorithms that incorporate an online update of the hyperparameters typically offer a lower performance at the initial phase of the exploration task, compared to those methods that assume the hyperparameters as pre-learned. This lies on the fact that algorithms that perform an online update must first learn the process spatial variation in order to exploit such knowledge. We tackle this issue in Chapter 6 of this thesis. Specifically, we evaluate the proposed algorithm's performance, with and without prior knowledge of the process, in an experiment where quadcopters explore a terrain profile.

To conclude we present in Table 3.1 a classification, according to the update of the model hyperparameters, of works that deal with information gathering

Approach	Employed by...
Entropy	Chen et al. (2015), Cliff et al. (2015), Jadidi et al. (2014), Ouyang et al. (2014), Merino et al. (2010), Stranders et al. (2009), Stranders et al. (2008), Krause and Guestrin (2007b).
Mutual information	Doo-Hyun et al. (2016), Dames et al. (2015), Jadidi et al. (2015), Choi and Lee (2015), Julian et al. (2014), Patten et al. (2013), Yang et al. (2013), Choi and How (2010), Hoffmann and Tomlin (2010), Kemppainen et al. (2010), Singh et al. (2010), Singh et al. (2009), Krause et al. (2008), Krause and Guestrin (2007b), Grocholsky et al. (2006).
Others	Kai-Chieh et al. (2016), Charrow et al. (2015), Miller and Murphey (2015), Gan et al. (2014), Lan and Schwager (2013), Carrillo et al. (2012), Marchant and Ramos (2012), Williams and Sukhatme (2012), Fink and Kumar (2010), Levine (2010), Low et al. (2008).

Table 3.2: Classification of works according to the information metric employed to select robots actions during the information gathering task.

with GPs.

3.2 Information Metrics for Exploration

Autonomous robotic exploration encompass a class of algorithms that allow a robot to autonomously decide where to move next in order to get a better understanding of a physical process of interest. The decision about where to move next could obviously be a random choice from a pool of possible potential positions, or could be based on a pre-defined pattern. However, such choices do not offer in general a high performance. In the literature, multiple metrics have been proposed, which outperform a random and a pre-defined trajectory.

Frontier cells. For example, one classic exploration approach is based on the concept of frontier cells (Yamauchi, 1997). A frontier cell denotes an explored cell that is an immediate neighbor of an unknown, unexplored cell. In (Yamauchi, 1997), the authors showed that by guiding a robot towards frontier cells they were able to obtain a superior performance compared to state-of-the-art methods. Another example is the algorithm proposed in (Williams and Sukhatme, 2012). In (Williams and Sukhatme, 2012) the authors aim to track the level curve of a process of interest. To this end they employ a Lyapunov control law that ensures that the system is stable at the level curve, which leads to a selection of a robot's control inputs that results in system's stability and, as a consequence, in an efficient tracking.

Information theoretic exploration. In contrast to (Yamauchi, 1997; Williams and Sukhatme, 2012), in this thesis we focus on information theoretic metrics to decide robots' actions, as such metrics have been proven to be effective in conjunction with GPs. Examples of information metrics, together with a respective work where they were utilized, are Fisher information (Levine, 2010; Miller and Murphey, 2015), predictive variance of the process estimation (Lan and Schwager, 2013), Upper Confidence Bound and Distance-based Upper Confidence Bound (Marchant and Ramos, 2012), A/D-optimality (Carrillo et al., 2012), entropy (Cliff et al., 2015), or MI (Krause et al., 2008).

Entropy and mutual information. Specifically, in this thesis we consider entropy and MI as information metrics (Section 2.3). These have been widely used in the information gathering with GPs literature, and have been shown to yield a superior performance compared to another metrics.

The most remarkable works that employ entropy and MI for information gathering with GPs are (Krause and Guestrin, 2007b) and (Krause et al., 2008), respectively. In (Krause and Guestrin, 2007b), the authors address the question of when an active sensing strategy, where locations are selected based on previous measurements, will perform better than a strategy where sensing takes place at an *a priori* specified set of locations. The main result of (Krause and Guestrin, 2007b) is a bound that trades off exploration and exploitation respect to the GPs hyperparameters. On the one hand, exploration aims to decrease the uncertainty about the model parameters. On the other hand, exploitation aims to select the best observations given a fixed model.

The work from (Krause et al., 2008) extends (Krause and Guestrin, 2007b) by deriving a near-optimal strategy for the exploitation phase. In particular, Krause et al. (2008) define a grid of measurements locations, and employ as information metric the MI between the potential measurements, i.e. positions where we aim to measure, and the not measured locations. By exploiting submodularity properties of MI, the authors derive an algorithm that selects a next measurement position that is within $1 - 1/e$ of the optimum.

Krause and Guestrin (2007b) and Krause et al. (2008) present algorithms that offer important theoretical guarantees. However, these algorithms employ a discretization of the environment, which is not suitable for real world applications, as we pointed out in Sec. 1.1. In addition to (Krause et al., 2008) and (Krause and Guestrin, 2007b), there are multiple papers that employ information met-

Approach	Employed by...
Discrete	Doo-Hyun et al. (2016), Kai-Chieh et al. (2016), Chen et al. (2015), Choi and Lee (2015), Cliff et al. (2015), Dames et al. (2015), Jadidi et al. (2015), Jadidi et al. (2014), Julian et al. (2014), Ouyang et al. (2014), Patten et al. (2013), Williams and Sukhatme (2012), Kemppainen et al. (2010), Stranders et al. (2010), Singh et al. (2009), Stranders et al. (2009), Krause et al. (2008), Low et al. (2008), Stranders et al. (2008), Krause and Guestrin (2007b), Grocholsky et al. (2006).
Continuous	Charrow et al. (2015), Miller and Murphey (2015), Nguyen et al. (2015), Gan et al. (2014), Hollinger and Sukhatme (2014), Lan and Schwager (2013), Yang et al. (2013), Marchant and Ramos (2012), Choi and How (2010), Fink and Kumar (2010), Hoffmann and Tomlin (2010), Levine (2010), Merino et al. (2010), Singh et al. (2010), Meliou et al. (2007).

Table 3.3: Classification of informative path planners according to the planning space representation.

rics for information gathering tasks. For a more complete classification of the information gathering literature according to the employed information metric we refer the reader to Table 3.2.

3.3 Path Planners for Information Gathering

Robotic information gathering algorithms aim to plan robots' paths that maximize the information gathered along the path while avoiding collisions with robots/obstacles. This is commonly referred in the literature as informative path planning. Informative path planning is one of the key components in any information gathering algorithm. In particular, here we classify informative path planners according to two features that are the most relevant for our work. These features are: planning space representation, and planning horizon.

Planning space representation. The representation of the planning space can be either discrete, i.e. robots plan on a discrete graph that is overlaid on top of the robot's configuration space, or continuous, i.e. robots plan in the full configuration space. Discrete planners like e.g. A* (Hart et al., 1968) are suited for low-dimensional problems (Sec. 2.2.1). In contrast, continuous planners like e.g. potential fields (Hwang and Ahuja, 1992) or sampling-based planners (LaValle and Kuffner, 2001; Kavraki et al., 1996) are state-of-the-art

Approach	Employed by...
Myopic	Doo-Hyun et al. (2016), Choi and Lee (2015), Cliff et al. (2015), Dames et al. (2015), Jadidi et al. (2015), Jadidi et al. (2014), Julian et al. (2014), Ouyang et al. (2014), Patten et al. (2013), Marchant and Ramos (2012), Williams and Sukhatme (2012), Fink and Kumar (2010), Hoffmann and Tomlin (2010), Kemppainen et al. (2010), Merino et al. (2010), Singh et al. (2010), Krause et al. (2008), Stranders et al. (2008), Grocholsky et al. (2006).
Non-myopic	Kai-Chieh et al. (2016), Charrow et al. (2015), Chen et al. (2015), Miller and Murphey (2015), Nguyen et al. (2015), Gan et al. (2014), Hollinger and Sukhatme (2014), Lan and Schwager (2013), Yang et al. (2013), Choi and How (2010), Levine (2010), Stranders et al. (2010), Singh et al. (2009), Stranders et al. (2009), Low et al. (2008), Krause and Guestrin (2007b), Meliou et al. (2007).

Table 3.4: Classification of informative path planners according to the planning horizon.

to solve high-dimensional problems (Sec. 2.2.2).

State-of-the-art informative path planners employ either discrete or continuous path planners, depending on the specific problem (see Table 3.3 for a detailed classification). In this thesis, our goal is to develop information gathering algorithms that can be generalized to a large class of robots. Therefore, as robots may have arbitrarily large state spaces, we propose the use of continuous planners for exploration tasks.

Planning horizon. The second feature that we consider to classify exploration algorithms is the planning horizon. Here we distinguish between two approaches: myopic, and non-myopic. Myopic approaches select the immediate next best action without accounting for the impact of that action into the future behaviour of the algorithm (Sec. 2.2.1). Alternatively, non-myopic approaches plan over a longer horizon several steps ahead by evaluating the impact of the actions over the horizon (Sec. 2.2.2).

Myopic approaches are in general suboptimal as they only select the next best action. If the action taken leads to a poor future behaviour, this is not considered while planning. An exception of a myopic algorithm that is able to find a solution that is close to optimal is (Krause et al., 2008), which guarantees that the solution is within $1 - 1/e$ of the optimum. However, as pointed out in Sec. 3.2, (Krause et al., 2008) does not satisfy our problem requirements (Sec. 1.3).

Approach	Employed by...
Centralized	Doo-Hyun et al. (2016), Kai-Chieh et al. (2016), Dames et al. (2015), Singh et al. (2009), Krause et al. (2008), Low et al. (2008).
Decentralized	Chen et al. (2015), Choi and Lee (2015), Gan et al. (2014), Patten et al. (2013), Williams and Sukhatme (2012), Hoffmann and Tomlin (2010), Stranders et al. (2010), Stranders et al. (2009), Stranders et al. (2008), Grocholsky et al. (2006).
Semi-decentralized	Ouyang et al. (2014), Levine (2010).

Table 3.5: Multi-robot state-of-the-art algorithms sorted according to the employed multi-robot architecture.

In contrast, non-myopic algorithms offer a higher performance compared to myopic approaches (see references in Table 3.4). A higher performance comes at the cost of a higher computational complexity, as planning over a longer horizon involves a larger number of states and actions to be considered.

In Table 3.4 we classify state-of-the-art informative path planners according to the planning horizon. By observing Tables 3.3, 3.4 we can conclude that there exist few algorithms in the literature that plan in a continuous space, and are non-myopic. This gap in the current literature is due to the difficulty to derive an algorithm that can deal with the computational complexity required to plan non-myopically in a continuous space. In this thesis, we overcome this issue and propose algorithms that are able to plan *online* and non-myopically in continuous spaces.

3.4 Multi-Robot Architectures and Inter-Robot Constraints

The objective of this thesis is to derive a multi-robot exploration algorithm (Sec. 1.3). Multi-robot systems offer clear advantages in terms of efficiency to gather information, and robustness to robots failures; respect to a single-robot system (Burgard et al., 2000). However, multi-robot systems introduce new aspects with which algorithms have to cope with. Specifically, we address in this thesis two aspects that, to our understanding, are the most relevant for multi-robot systems. These are the following:

- *multi-robot architecture*, i.e. how robots organize themselves to carry out an exploration task; and

Approach	Employed by...
Collision avoidance	Dames et al. (2015), Gan et al. (2014), Williams and Sukhatme (2012).
Network connectivity	Stranders et al. (2009).
Others	Doo-Hyun et al. (2016), Gan et al. (2014).
None	Kai-Chieh et al. (2016), Chen et al. (2015), Choi and Lee (2015), Ouyang et al. (2014), Patten et al. (2013), Hoffmann and Tomlin (2010), Levine (2010), Stranders et al. (2010), Singh et al. (2009), Krause et al. (2008), Low et al. (2008), Stranders et al. (2008), Grocholsky et al. (2006).

Table 3.6: Classification of multi-robot works according to the inter-robot constraints they can handle.

- *inter-robot constraints*, i.e. which constraints must algorithms handle because of the presence of multiple robots.

Next we discuss in detail the two aforementioned aspects.

Multi-robot architecture. We distinguish between three classes of multi-robot architectures:

(a) *centralized*, which consists of the combination of multiple identical robots plus a "central" robot. The central robot typically collects information from the other robots, and coordinates the subsequent robots movements (see Fig. 3.1a).

(b) *decentralized*, which, in contrast to a centralized one, does not require a "central" robot. This implies that the cooperation is performed by means of local communication between neighboring robots (see Fig. 3.1b).

(c) *semi-decentralized*, where most of the system is decentralized but it has some instances that run in a centralized fashion. An example of a semi-decentralized architecture could be a system in which robots decide their movements just by "talking" to their neighbors, but the data fusion is performed in a central node and then broadcasted to the team.

In Table 3.5 we classify state-of-the-art information gathering works according to the multi-robot architecture. In particular, we would like to comment about two works that employ a semi-decentralized architecture: (Levine, 2010)

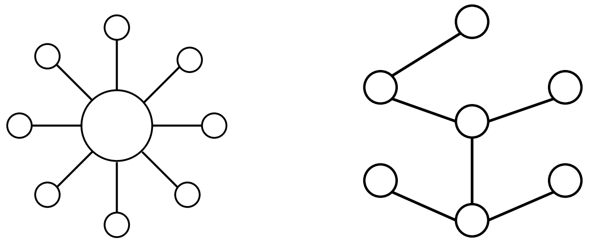

(a) Centralized. (b) Decentralized.

Figure 3.1: Multi-robot architectures. We depict robots by circles, with the size of the circle representing the computational load of the robot. Lines link two circles if the corresponding robots interact with each other.

and (Ouyang et al., 2014). Levine (2010) proposes an algorithm that incorporates a decentralized planning architecture in which gathered measurements are filtered locally by individual robots. Then the estimates of the individual robots are shared among the team in a consensus framework that employs local communication between robots. However, measurements gathered by robots are processed in a central node that broadcasts the measurements to the individual robots. In contrast, in (Ouyang et al., 2014) the data fusion is realized in a decentralized fashion while the multi-robot cooperation is partially decentralized. This implies that cooperation is not solvable only with local communication between robots, and, therefore, requires either a central node or a broadcast mechanism.

Inter-robot constraints. A multi-robot exploration algorithm must meet additional constraints compared to a single-robot system. For example, two fundamental constraints are inter-robot collision avoidance, and maintenance of network connectivity. Let us remark that not considering multi-robot constraints could lead to e.g. unfeasible trajectories, and/or network disconnections that could interrupt robots' cooperation procedures.

In Table 3.6 we classify algorithms according to the inter-robot constraints they can handle. In particular, we consider collision avoidance, network connectivity, other constraints, and no constraints that can be handled by the algorithm. We would like to highlight (Doo-Hyun et al., 2016) and (Gan et al., 2014) as an example of works that consider other constraints. In (Doo-Hyun et al., 2016), the authors assume that flying robots are affected by wind, which the authors take into account for inter-robot collision. Also Gan et al. (2014) present an approach that is general in the sense that they can incorporate objective and constraints functions, with the restriction that the resulting function is differentiable. However, differentiability required by (Gan et al., 2014) limits their approach to a certain class of objective functions and constraints.

3.5 Final remarks

The review of the state-of-the-art lets us conclude that there is a need in the current literature of a multi-robot exploration algorithm that (i) plans in a continuous space, (ii) is non-myopic, (iii) is decentralized, and (iv) deals with complex inter-robot constraints. These four points are our motivation to formulate our research problem (Sec. 1.3), and to write this thesis. In the following chapters we develop a step-by-step concept that guides us towards our objective.

Part II

Single-Robot Exploration

Chapter 4

Myopic Single-Robot Exploration

This chapter introduces a first method to explore a physical process with an autonomous mobile robot. Specifically, we explore a magnetic field intensity in this chapter. Our method combines the three main ingredients described in Chapter 2: (i) a GPs underlying model to perform effective regression, (ii) a decision maker to identify potential robot actions, (iii) and an information metric to drive the robot's motion. In addition to the proposed algorithm description, we introduce as well in this section some of the notation and basic ideas that will be used in the remainder of the thesis.

We organize this chapter as follows: first, we describe the robot and sensor model in Section 4.1. Then we introduce the concept of model-based information-driven exploration in Section 4.2. We demonstrate the effectiveness of the proposed algorithm for the exploration of a magnetic field intensity within an indoor environment, both in simulations (Section 4.3) and experiments (Section 4.4). The use of the proposed algorithm to explore a magnetic field intensity represents the main contribution of this chapter.

4.1 Robot and Sensor Model

We aim to explore a physical process $y(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}_{free}$, autonomously with a robot. The robot measures $y(\mathbf{x})$ (e.g. a magnetic field intensity or a wind field) with a sensor that is characterized by (2.1). In this thesis we assume that $y(\mathbf{x})$ can be modeled with a GP of zero mean and covariance function $k(\mathbf{x}, \mathbf{x}')$. That is, $y(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}'))$ (see Section 2.1).

Let us denote the robot's position $\mathbf{x}_r \in \mathcal{X}_{free} \subset \mathbb{R}^{d_s}$. The robot's motion model is given by a function

$$\mathbf{x}_r(t + dt) = \mathbf{f}_m(\mathbf{x}_r(t), \mathbf{u}), \tag{4.1}$$

which relates the robot's current position $\mathbf{x}_r(t)$ and future position $\mathbf{x}_r(t + dt)$ given a control input $\mathbf{u}$. This corresponds to a generic motion model that could correspond to any robot, such as a holonomic robot, a car-like robot, or an aircraft.

Note that the definition of the robot and sensor model is rather general. This allows us to employ this notation throughout the thesis for multiple classes of robots and sensors. For each of the chapters, we particularize the algorithms for specific classes of robots. In the following, we present an algorithm, which uses the proposed notation, to explore a magnetic field intensity.

4.2 Model-Based Information-Driven Myopic Exploration using GPs

In this section we introduce the concept of model-based information-driven exploration. Information-driven exploration means that robot's actions are determined by an information metric (Section 2.3). This works as follows: first the robot calculates potential next actions. Then it ranks those actions according to an information theoretic function, and follows the trajectory that maximizes this function. In this chapter we consider a myopic decision maker to select potential next actions (Section 2.2.1), and differential entropy as information metric (Section 2.3.1).

Calculation of entropy, and of any information metric in general, requires an underlying model of $y(\mathbf{x})$. Given a model we can calculate, for example, the entropy at a position of interest (see Section 2.3.1). Obviously, the better the model represents $y(\mathbf{x})$, the better the information metric will guide the robot to perform an efficient exploration. As we pointed out in Chapter 2.1, we employ in this work GPs for regression because of their ability to model phenomena with spatial variations. Next, we show in Section 4.2.1 some first insights on the performance of the regression with GPs for exploration tasks. This is followed by a description of the proposed algorithm in Section 4.2.2.

4.2.1 GPs Regression for Exploration

We depict in Figure 4.1 a magnetic field intensity within an indoor environment. The magnetic field was measured in our Deutsches zentrum für Luft- und Raumfahrt (German Aerospace Center) (DLR) lab with a spatial resolution of 10 cm. That is, we considered an horizontal and vertical separation of 10 cm between any two consecutive measurements. We would like to point out that exploring a magnetic field intensity is motivated by research carried out at DLR on indoor localization systems (Angermann et al., 2012).

We denote the measured magnetic field intensity as $\mathbf{y}_G(\mathbf{X}_G)$, measured at positions $\mathbf{X}_G \subset \mathcal{X}_{free}$. $\mathbf{y}_G(\mathbf{X}_G)$ is a subset sampled from $y(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}_{free}$. We

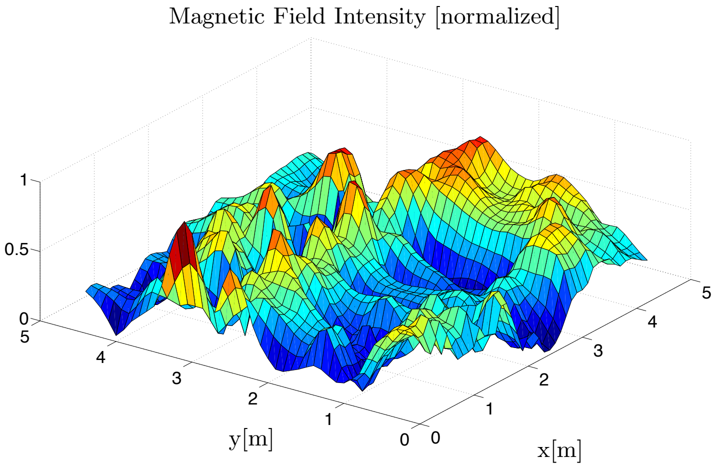


Figure 4.1: Magnetic field intensity captured on the ground in an indoor environment with a resolution of 10 cm. The data is normalized, with values ranging between 0 and 1.

assume $\mathbf{y}_G(\mathbf{X}_G) = [y_G^{[1]}, ..., y_G^{[n_G]}]$, with $y_G^{[i]} \in \mathbb{R}$ the magnetic field intensity value at position $\mathbf{x}_G^{[i]} \in \mathbf{X}_G$, to be the ground truth for the subsequent experiments.

Here we investigate the impact of the number of measurements $\mathbf{z}$ we take, and the reconstruction error. The reconstruction error is measured as the Root Mean Squared Error (RMSE) between $\mathbf{y}_G(\mathbf{X}_G)$ and the process estimate $\boldsymbol{\mu}_* = [\boldsymbol{\mu}_*^{[1]}, ..., \boldsymbol{\mu}_*^{[n_G]}]$, calculated with (2.3) at $\mathbf{X}_* = \mathbf{X}_G$ given $\mathbf{z}$, $\mathbf{X}$. Specifically, the RMSE corresponds to the following expresion:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n_G} \left(\boldsymbol{\mu}_*^{[i]} - y_G^{[i]}\right)^2}{n_G}}. \tag{4.2}$$

We perform the following experiment: first a robot measures at random vertices $\mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}}$, with $|\mathcal{V}_{\mathcal{X}_{free}}| = n_G$, and calculates $\boldsymbol{\mu}_*$. Then we compute the RMSE with (4.2) for three different classes of GPs covariance functions: squared exponential (`SEiso`), and Matèrn covariance function with $\nu = \frac{3}{2}$ (`Matern3`) and $\nu = \frac{5}{2}$ (`Matern5`). We refer to (Rasmussen and Williams, 2005) for more details in the later covariance function. Figure 4.2 shows the result of this experiment. We evaluate the RMSE as the robot increases the percentage of measured values, and compare the aforementioned GPs covariance functions against the non-predictive case. Here we term it `NoPrediction`. This takes the measured value for the measured cells and predicts the mean value of the process, calculated a priori from $\mathbf{y}_G(\mathbf{X}_G)$, for the rest.

We can observe that estimation with GPs offers clear advantages in terms

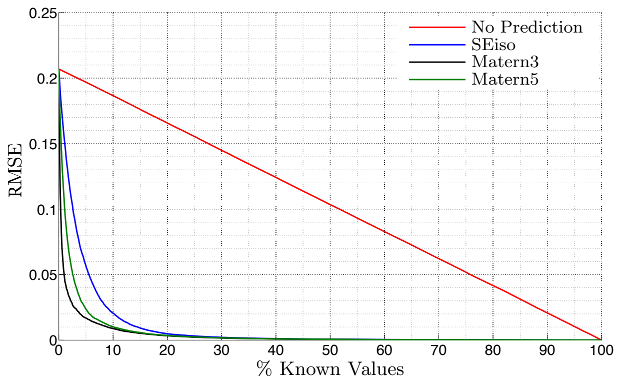


Figure 4.2: RMSE between estimate and ground truth as we increase the number of measurements. Here we select the measurements positions randomly. We compare a non-predictive case (`NoPrediction`) against GPs regression (`SEiso`,`Matern3`,`Matern5`).

of prediction capabilities since we are able to decrease the RMSE more rapidly. However, in previous experiment we chose the measured positions simply randomly. This arises the question: could we select the sampling locations in a more intelligent way to decrease the RMSE even faster? The answer is yes. In next section we describe an algorithm that performs an intelligent and more efficient sampling.

4.2.2 Entropy-Driven Exploration with GPs

The algorithm we describe next was first proposed by Shewry and Wynn (1987) and Cressie (1992). Here we review it and adapt the notation to meet our problem requirements. The algorithm is based on the principle of maximum entropy. That is, the robot moves next to the position, from a set of potential positions, that has a maximum entropy. Entropy can be easily calculated with (2.20).

Before describing the algorithm in detail, let us illustrate the concept of entropy-driven exploration with an example. We show in Figure 4.3 the entropy at each individual $\mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}}$ after taking some measurements with a robot along a path. The color scale represents the process entropy. Regions coloured in dark blue (low entropy) correspond to the robot's path, and regions coloured in red (high entropy) correspond to areas that are far from the measured positions. This has a clear interpretation: areas that are far from the explored regions are more informative and, therefore, have a higher entropy.

Our goal is to drive the robot to unexplored areas; i.e. to areas of high entropy. This can be realized with the algorithm introduced in Fig. 4.4, which

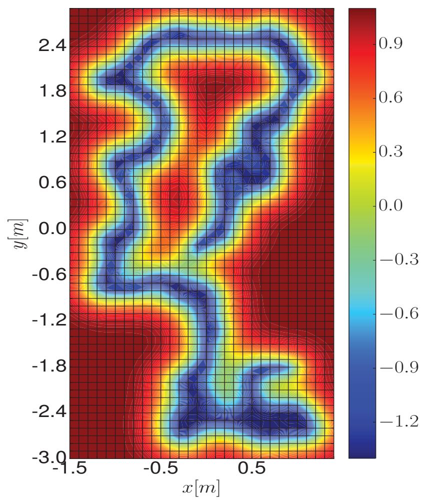


Figure 4.3: Entropy (measured in bits) at each individual cell after taking some measurements with a robot along a path. Dark blue areas correspond to the robot's path – low entropy – and red areas correspond to non-measured positions – high entropy.

we summarize in Algorithm 4.1. This takes as inputs the following parameters:

- Robot's current position $\mathbf{x}_r$.
- GPs hyperparameters $\boldsymbol{\theta}$ that define the model. In this chapter we assume we know *a priori* the hyperparameters; i.e. we assume we have some prior knowledge about the process spatial correlations. In the following chapters we will extend the algorithm to incorporate the learning of the GPs hyperparameters online as the robot takes measurements.
- Graph $\mathcal{L}_{\mathcal{X}_{free}}(\mathcal{V}_{\mathcal{X}_{free}}, \mathcal{E}_{\mathcal{X}_{free}})$ that determines how the robot can move (see Fig. 2.3a).
- Stop criterion that stops the algorithm's execution. Possible stop criteria could be an user-defined number of iterations, a fixed running time of the algorithm, or remaining uncertainty about the explored process.

Algorithm 4.1 works as follows: first, the robot takes a measurement at position $\mathbf{x}_r$ with a sensor. The sensor choice will be motivated by the concrete application. For example, for exploring the magnetic field intensity we use a magnetometer. Then the robot incorporates the new measurement z and its location $\mathbf{x}_r$ into a measurements' vector $\mathbf{z}$ and positions' vector $\mathbf{X}$ (line 5).

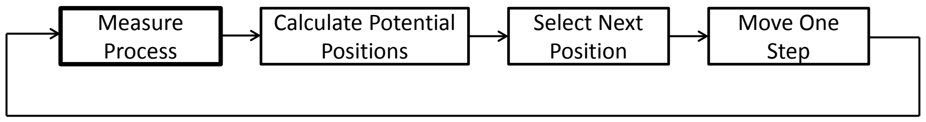


Figure 4.4: Algorithm block diagram. First step corresponds to measuring the process of interest at the robots' current locations. Then the algorithm is executed in a loop.

Algorithm 4.1. `SingleRobotExploration`$(\mathbf{x}_r, \boldsymbol{\theta}, \mathcal{L}_{\mathcal{X}_{free}}, StopAlgorithm)$

```
1: $\mathbf{z} \leftarrow NULL; \mathbf{X} \leftarrow NULL$
2:
3: while ! $StopAlgorithm$ do
4:     $z \leftarrow \texttt{Measure}(\mathbf{x}_r)$
5:     $\mathbf{z} \leftarrow [\mathbf{z};\ z]; \mathbf{X} \leftarrow [\mathbf{X}; \mathbf{x}_r^T]$
6:     $\mathbf{X}_* \leftarrow \texttt{CalcNextPotentialPositions}(\mathcal{L}_{\mathcal{X}_{free}}, \mathbf{x}_r)$        ▷ with (2.7)
7:     ▷ Calculate maximum entropy position
8:     $h_{max} \leftarrow -\infty$
9:     for $\mathbf{x}_*^{[j]} \in \mathbf{X}_*$ do
10:        $\mu_*, \sigma_*^2 \leftarrow \texttt{PredictGP}(\mathbf{z}, \mathbf{X}, \mathbf{x}_*^{[j]}, \boldsymbol{\theta})$        ▷ with (2.3)
11:        $h_* \leftarrow \texttt{CalculateEntropy}(\sigma_*^2)$        ▷ with (2.20)
12:        if $h_* > h_{max}$ then
13:            $h_{max} \leftarrow h_*;\ \mathbf{x}_{next} \leftarrow \mathbf{x}_*^{[j]}$
14:     $\mathbf{x}_r \leftarrow \texttt{MoveTo}(\mathbf{x}_{next})$
15: $\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_* \leftarrow \texttt{ReconstructProcess}(\mathbf{z}, \mathbf{X}, \mathcal{V}_{\mathcal{X}_{free}})$
```

Next, the robot selects the set of potential positions $\mathbf{X}_*$ where it can move in the current iteration (line 6). In this chapter we consider a discrete graph-based myopic approach (Section 2.2.1) to select the agent's next position with (2.7) (see Figure 4.5). We would like to stress that the use of a myopic approach makes the exploration algorithm myopic, as the robot plans sequentially the immediate next best action towards its exploration goal (reduction of the RMSE between a process estimate and $\mathbf{y}_G(\cdot)$).

Once the robot calculates the set of potential positions $\mathbf{X}_*$, it selects the position from $\mathbf{X}_*$ with the highest entropy (lines 8-13). This is done as follows: first it predicts the process mean μ and variance σ_* for each of the positions $\mathbf{x}_*^{[j]} \in \mathbf{X}_*$ using a GPs model defined by hyperparameters $\boldsymbol{\theta}$. Mean and variance are calculated according to (2.3), where the inputs are $\mathbf{z}$, $\mathbf{X}$ (line 10). Variance is then used in line 11 to calculate entropy at position $\mathbf{x}_*^{[j]}$. This is done with (2.20). After calculating entropy at all locations in $\mathbf{X}_*$, the robot moves to position $\mathbf{x}_{next}$ that has the highest entropy.

The robot continues running the algorithm by moving towards positions with

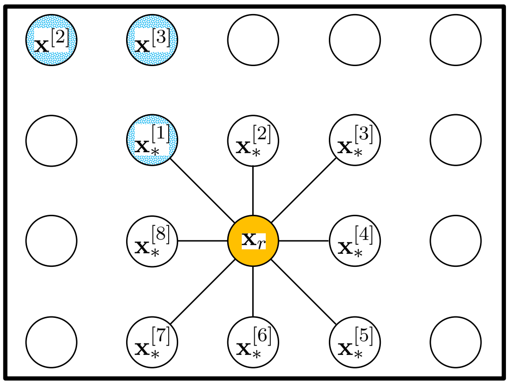


Figure 4.5: Graphical representation of an exploration scenario. Nodes correspond to $\mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}}$. Blue dotted nodes are positions where a robot took a measurement. The orange node is the robot's current position, and nodes linked to the orange one are potential robot's next actions.

high entropy. Once the algorithm stops, the collected measurements $\mathbf{z}, \mathbf{X}$ are used to predict $\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*$ at positions $\mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}}$ (line 15). Following a maximum likelihood criterion we select $\boldsymbol{\mu}_*$ as our exploration result.

4.3 Simulations and Discussion of Results

In this section we evaluate the performance of Algorithm 4.1 to explore a magnetic field intensity within an indoor environment. Here $\mathbf{y}_G(\mathbf{X}_G)$ corresponds to the magnetic field depicted in Figure 4.1. This was measured with a holonomic robot (see Figure 4.6) that is a modified version of the commercially available Slider platform by Commonplace Robotics. Due to its four mecanum wheels the platform is able to perform omnidirectional movements, following input commands for forward, lateral and rotational velocities. The magnetic field sensor module used in the reported experiments is part of a commercial integrated sensor package (Xsens MTx). We mounted the sensor on a wooden beam that extended 0.75 m from the center of the robot. The purpose of this beam is to separate the sensor from the robot's ferromagnetic components and electromagnetic field generating devices. We employ a commercial motion capture system (Vicon) to provide ground truth information of the robot's position. Our particular setup consists of 16 infrared sensitive cameras and infrared strobes.

Benchmark trajectories. We compare the performance of Algorithm 4.1 against three predefined trajectories and a random walk. Trajectories are de-

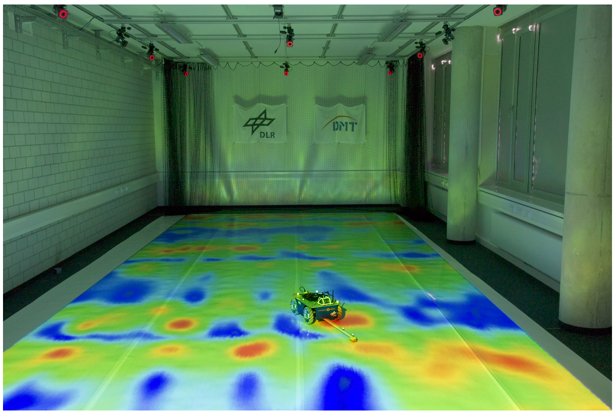


Figure 4.6: Experimental setup employed to test the algorithm's performance. We show a mobile robot and projection on the ground of the magnetic field intensity within our DLR lab.

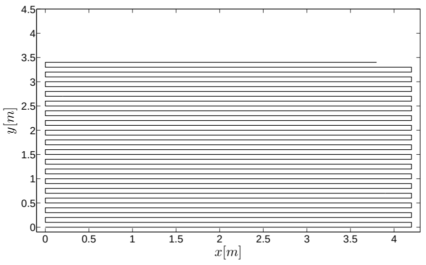
picted in Figure 4.7. In the following, we explain each of the trajectories in detail:

- `Meander`: meander-like trajectory starting from the left bottom corner.
- `Spiral`: spiral-like trajectory starting from the left bottom corner.
- `SplitTwo`: trajectory that consists of halving the space consecutively.
- `Random`: random walk that selects the next position randomly given the robot's motion graph. We add some intelligence to the random trajectory so that it does not measure twice at the same position. It would only measure twice if it has no other choice. Adding some intelligence makes the comparison against the other algorithms fairer.

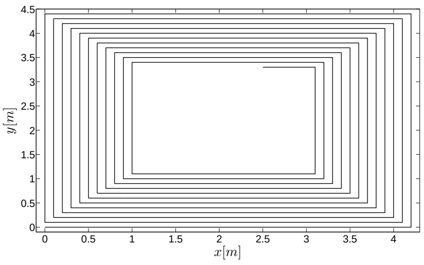
For all trajectories, including Alg. 4.1, horizontal and spatial separation between two consecutive measurements at $\mathbf{x}, \mathbf{x}' \in \mathcal{V}_{\mathcal{X}_{free}}$ is equal to the process spatial resolution; i.e. 10 cm in our particular setup. Predefined trajectories correspond to a single run of the algorithm. On the contrary, results of Alg. 4.1 and the random walk strategy correspond to the average, together with corresponding 2σ error bars, over 100 simulation runs.

4.3.1 RMSE Evolution

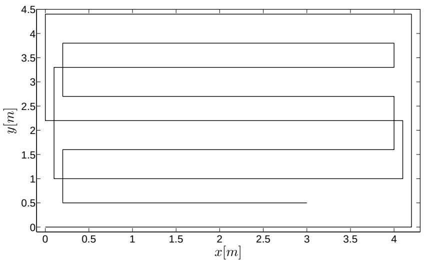
We evaluate the evolution with time of the RMSE, calculated with (4.2). Specifically, we employ the following $\boldsymbol{\theta}$: $l = 0.2, \sigma_f^2 = 0.07, \sigma_n^2 = 4e^{-4}$. These were



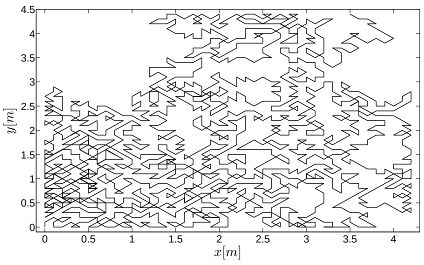
(a) Meander.



(b) Spiral.



(c) SplitTwo.



(d) Random.

Figure 4.7: Trajectories employed to benchmark our algorithm for the exploration of a magnetic field intensity.

learned from a subset of the data using (2.5). Let us also add that we employ the pyGPs[1] library to carry out all operations related to GPs. We show the RMSE evolution with time for the analyzed trajectories in Figure 4.8. We can observe that Algorithm 4.1 offers the best performance in terms of the RMSE of the considered algorithms. It is also relevant to point out that `SplitTwo` trajectory performs very well. However, it requires a rectangular environment to plan the trajectory. Notice that an environment with, for example, one obstacle in the middle would not allow planning this trajectory. On the contrary, the proposed algorithm would allow exploring the magnetic field intensity in an environment of arbitrary shape.

4.3.2 Algorithm's Scalability

Another key feature of Alg. 4.1 is its scalability with respect to the environment's size in which the physical process of interest takes place. We define the scalability respect to the environment's size in terms of the exploration time, measured as the time needed to obtain an RMSE of 0.005. We carried out simulations for

[1]pyGPs - A Package for Gaussian Processes - http://www-ai.cs.uni-dortmund.de/weblab/static/api_docs/pyGPs/

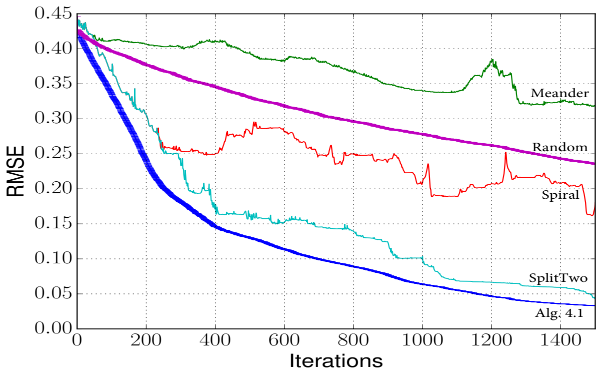


Figure 4.8: Evolution with time of the RMSE between estimate, calculated with GPs regression given the gathered measurements, and ground truth. We benchmark Alg. 4.1 against three predefined trajectories and a random walk.

different environment sizes. For each of the environment sizes we performed 100 simulation runs. Figure 4.9 shows the obtained results. Red dots correspond to the mean over the performed simulation runs. The blue line corresponds to a linear regression performed over the blue dots. Results illustrate that algorithm's scalability respect to the environment size is linear. Moreover, the slope of the obtained regression line is below one; 0.25 to be precise. This indicates that, for example, an environment that it is four times bigger than another one would only take double time to explore it with the proposed algorithm.

4.4 Experiments and Discussion of Results

In addition to the simulation results, we carried out an experiment [2] to demonstrate the effectiveness of Alg. 4.1 to explore a magnetic field intensity. We use the same hardware setup as in Section 4.3. Figure 4.6 illustrates a snapshot of the experiment. We show the robot, together with a projection on the floor of the previously measured ground truth data.

We compare the proposed algorithm with a meander-like and a random trajectory. Specifically, we evaluate the evolution of the RMSE with time, depicted in Figure 4.10. On the one hand, we can observe that the performance of a random and Alg. 4.1 trajectory are similar at the beginning of the algorithm's execution. This lies on the fact that, before taking a few measurements, is not relevant to move in an intelligent way since we can not exploit the inter-

[2]A video that shows the experiment execution can be found in: `https://vimeo.com/253575604`; `https://rebrand.ly/myopi596a`.

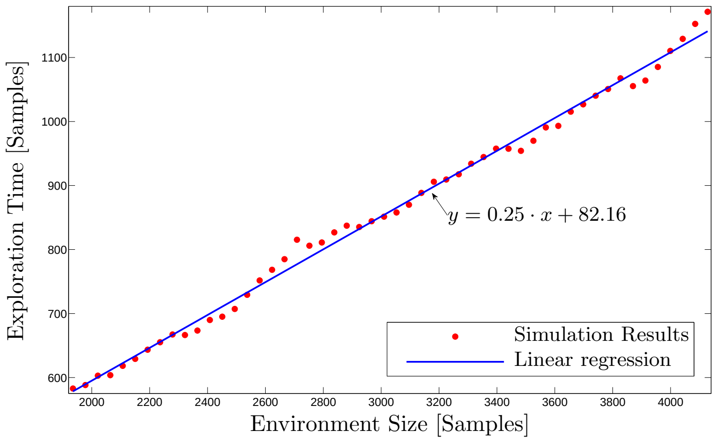


Figure 4.9: Scalability of the algorithm respect to the environment's size.

measurements correlation. Instead, it is sufficient to move and collect data to perform a good estimation. On the other hand, we can conclude that once the robot has collected sufficient measurements (approx. after 500 seconds), difference between random and Alg. 4.1 performance gets larger. This translates in a reduction on the RMSE of 0.4 (four-fold improvement) after a 1200 seconds exploration task.

4.5 Summary and Outlook

In this chapter we described a first approach that allows a mobile robot to autonomously explore a physical process of interest – magnetic field intensity in an indoor environment. We evaluated the performance of the algorithm both in simulations, and experiments with a robot in the loop. Results illustrate that the proposed approach is able to explore a magnetic field intensity much more efficiently than with a trajectory consisting of a predefined pattern – four-fold improvement in the experimental evaluation.

However there are a few aspects that will be considered in this thesis to improve the algorithm performance. These are the following:

1. We are considering only one robot. An extension of this algorithm to multiple robots will be proposed in Chapter 6.

2. The robot plans in a myopic fashion, which is in general suboptimal (see Section 1.4). Alternative path planning schemes will be considered in Chapters 5 and 7 to improve the algorithm performance.

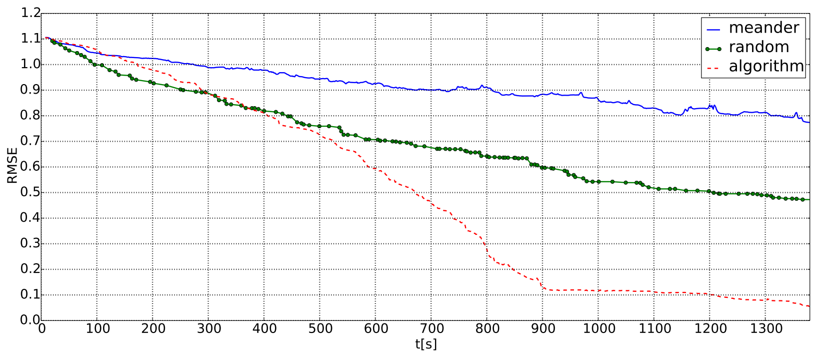


Figure 4.10: Evolution with time of the RMSE between estimate, calculated with GPs regression given the collected measurements, and ground truth. We test three different algorithm's in an experiment with a robot in the loop equipped with a magnetometer to measure a magnetic field intensity.

3. The use of entropy as information metric leads to a superior performance compared to predefined trajectories. However, more complex metrics have been proposed in the literature, like e.g MI (see Section 2.3.2), which we expect would yield a better performance. In Chapters 5 and 7 we consider some of those metrics. In addition we will propose alternative metrics for exploration.

4. We employed a holonomic robot to carry out the exploration task. In Chapters 5 and 7 we expand the algorithm to handle a larger class of robots.

In next chapter (Chapter 5) we will consider points 2,3,4 and will propose a more efficient exploration algorithm. In particular, in Chapter 5 we introduce an information gathering algorithm based on RRT* (see Section 2.2.2) that plans non-myopically in a continuous space, and trades off path informativeness and path cost.

Chapter 5

Sampling-Based Single-Robot Exploration

We proposed in Chapter 4 an algorithm that allows a robot to autonomously explore a physical process. The algorithm proposed in Chapter 4 employs a discrete graph-based myopic approach to drive the robot. In contrast, here we extend the algorithm proposed in Chapter 4 to plan non-myopically in a continuous space. This allows us to account for robots with large state spaces, i.e. larger than four states (LaValle and Kuffner, 2000), which are subject to differential constraints, like e.g. UAVs (Nguyen et al., 2015), as pointed out in Section 1.1.

Many recent works have proposed algorithms for information gathering in continuous spaces (see Section 3.3). However, these algorithms are not suited for online information gathering tasks, as we demonstrate in Section 5.6 of this chapter. Thus, in this chapter we propose a novel approach that allows an online exploration of a physical process. Specifically, we introduce an algorithm that consists of two main steps: (i) a search mechanism to identify locations that are highly informative – stations – as measured by an information metric (Section 2.3), and (ii) a planning mechanism to find a path towards the station that trades-off informativeness and cost. We realize this last step via an adaptation of the RRT* planner (Section 2.2.2).

The algorithm proposed in this chapter adds an additional feature respect to Chapter 4. Here the robot learns the GPs hyperparameters as it collects measurements, which allows a robot to explore a process without prior information.

We organize this chapter as follows: first, we describe the proposed algorithm in detail in Sections 5.1-5.4. Then we evaluate the algorithm's computational complexity in Section 5.5, and its performance through simulations in Section 5.6. This is followed in Section 5.7 by an experiment in which a robot autonomously gathers information of a magnetic field intensity in an indoor environment. We finalize with a summary and outlook of the chapter in Section 5.8.

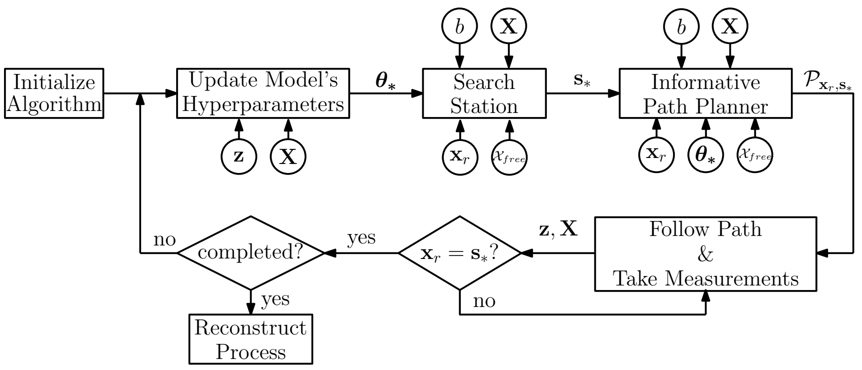


Figure 5.1: Algorithm block diagram.

5.1 Efficient Information Gathering using RRT-Based Planners and GPs

We aim to explore with a robot an unknown process $y(\mathbf{x})$, for $\mathbf{x} \in \mathcal{X}_{free}$. As we employ GPs as underlying model of $y(\mathbf{x})$, the lack of prior information about $y(\mathbf{x})$ implies that hyperparameters $\boldsymbol{\theta}$ need to be estimated and updated as the robot collect measurements. In fact, the spatial distribution of the information metric is directly related to the values of $\boldsymbol{\theta}$. An adaptation of $\boldsymbol{\theta}$ while the robot moves will essentially make any planning suboptimal, since the information metric computed at any region in space will follow the fluctuations of the hyperparameter estimates $\boldsymbol{\theta}_*$. Instead, we propose updating $\boldsymbol{\theta}$ only at some point in the vicinity of the robot's current location that maximizes the information gained about the modeled process. This point we name it a *station*, a concept inspired by frontiers in autonomous robotic exploration (Yamauchi, 1997). Then, given $\boldsymbol{\theta}_*$, the robot plans a route towards the station so as to further increase the amount of information about $y(\mathbf{x})$. In this case the resulting information metric calculated at all points in space is fixed and thus planning (conditioned on $\boldsymbol{\theta}_*$) will optimize the desired utility function. In the following, we describe the exploration strategy in detail.

Algorithm overview. A block diagram of the whole scheme is shown in Figure 5.1. We present in Algorithm 5.1 a detailed pseudo-code. Our proposed algorithm works as follows. First, the robot learns the $\boldsymbol{\theta}_*$ that best model the acquired measurements $\mathbf{z}, \mathbf{X}$ (line 4 in Alg. 5.1). This is done with (2.5), finding $\boldsymbol{\theta}_*$ that maximize the LML.

Once the robot estimates $\boldsymbol{\theta}_*$, it searches for a highly informative station $\mathbf{s}_*$ (line 5) using Algorithm 5.2. Algorithm 5.2 takes as input $\mathbf{x}_r$ and $\mathcal{X}_{free}$, a

Algorithm 5.1. `SBSRE Algorithm`$(\mathbf{x}_r, \mathcal{X}_{free}, b, StopAlgorithm)$

```
1: $\mathbf{z} \leftarrow NULL; \mathbf{X} \leftarrow NULL;$
2:
3: while ! $StopAlgorithm$ do
4:     $\boldsymbol{\theta}_* \leftarrow$ LearnHyp$(\mathbf{z}, \mathbf{X})$;                    ▷ update process model with (2.5)
5:     $\mathbf{s}_*, \mathcal{P}_s, u_s \leftarrow$ SearchStation$(\mathbf{x}_r, \boldsymbol{\theta}_*, \mathbf{X}, b, \mathcal{X}_{free})$;    ▷ with Algorithm 5.2
6:     $\mathcal{P}_p, u_p \leftarrow$ InformativePlanner$(\mathbf{x}_r, \mathbf{s}_*, \boldsymbol{\theta}_*, \mathbf{X}, b, \mathcal{X}_{free})$;    ▷ with Algorithm 5.3
7:     if $\mathcal{P}_p \neq \emptyset$ AND $u_p > u_s$ then                 ▷ choose the best solution
8:         $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*} \leftarrow \mathcal{P}_p$;
9:     else
10:        $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*} \leftarrow \mathcal{P}_s$;
11:    for $\mathbf{x}_i \in \mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ do                  ▷ follow and measure along the path
12:        $\mathbf{x}_r \leftarrow$ MoveTo$(\mathbf{x}_r, \mathbf{x}_i)$;
13:        $z \leftarrow$ Measure$(\mathbf{x}_r)$;
14:        $\mathbf{z} \leftarrow [\mathbf{z}; z]$;  $\mathbf{X} \leftarrow [\mathbf{X}; \mathbf{x}_r^T]$;
15: $\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_* \leftarrow$ ReconstructProcess$(\mathbf{z}, \mathbf{X}, \mathcal{X} \subset \mathcal{X}_{free})$;    ▷ with (2.3)
```

budget constraint on the path cost b, and $\boldsymbol{\theta}_*$ that allow the robot to calculate the expected information contained at a station. In addition to $\mathbf{s}_*$, the algorithm outputs a suboptimal, yet feasible path $\mathcal{P}_s = [\mathbf{x}_r, ..., \mathbf{x}_i, ..., \mathbf{s}_*]$, with $\mathbf{x}_i \in \mathcal{X}_{free}$, and the corresponding path utility u_s. More on the computation of the utility and its properties will be discussed in Sec. 5.3.

Then the robot plans a trajectory from $\mathbf{x}_r$ to $\mathbf{s}_*$ (line 6) using an informative path planner (Alg. 5.3) in order to refine $\mathcal{P}_s$. In Sec. 5.3 we describe Alg. 5.3 in more details. The algorithm result is a trajectory $\mathcal{P}_p$, together with its corresponding utility u_p that trades off the information gathering with the path cost.

Alg. 5.3 has an anytime nature; i.e. it aims to find a feasible solution and then to improve it with time. Note however, that it is possible that, for the stop criterion pre-defined by the user, e.g. planning time, Alg. 5.3 is either not able to find a path or the found path is of worse quality (in terms of used utility) than $\mathcal{P}_s$. To guarantee that a solution is found, the robot compares solutions from Algorithms 5.2 and 5.3 in lines 7-10 of Alg. 5.1, and selects the best path $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ according to an information metric.

Finally, the robot follows $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ until it reaches $\mathbf{s}_*$, while it measures $y(\mathbf{x}_i)$, for all $\mathbf{x}_i \in \mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$, and incorporates $\mathbf{z}$ and $\mathbf{X}$ to its knowledge database (lines 11-14, Alg. 5.1). Then, the main loop is repeated until some stopping criterion is fulfilled, e.g., maximum exploration time, or the remaining process uncertainty. Once the robot finishes gathering information, it can predict the value of the process $\boldsymbol{\mu}_*$ and the associated uncertainty $\boldsymbol{\Sigma}_*$ of the prediction for any $\mathcal{X} \subset \mathcal{X}_{free}$ using (2.3) (line 15, Alg. 5.1).

Algorithm 5.2. `SearchStation`$(\mathbf{x}_r, \boldsymbol{\theta}_*, \mathbf{X}, b, \mathcal{X}_{free})$

```
1: $\mathcal{V} \leftarrow \{\mathbf{x}_r\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathbf{s}_* \leftarrow \{\mathbf{x}_r\}$; $\mathcal{P}_s \leftarrow \emptyset$;
2: $I_{\max} \leftarrow -\infty$; $u_s \leftarrow -\infty$;
3:
4: while ! StopStation do
5:     $\mathbf{x}_{\text{rand}} \leftarrow$ SampleFree($\mathcal{X}_{free}$);
6:     $\mathbf{x}_{\text{nearest}} \leftarrow$ Nearest($\mathbf{x}_{\text{rand}}, \mathcal{V}$);
7:     $\mathbf{x}_{\text{new}} \leftarrow$ Steer($\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \eta$);
8:     if CollisionFree($\mathcal{P}_{\mathbf{x}_{\text{nearest}},\mathbf{x}_{\text{new}}}, \mathcal{X}_{free}$) then
9:         $c_{\text{new}} \leftarrow c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}_{\text{nearest}}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{\text{nearest}},\mathbf{x}_{\text{new}}})$;
10:        if $c_{\text{new}} < b$ then                                        ▷ budget constraint
11:            $I_{\text{new}} \leftarrow$ InformationS($\mathbf{x}_{\text{new}}, \boldsymbol{\theta}_*, \mathbf{X}$);      ▷ calculate node informativeness
12:            $\mathcal{N}_{\text{new}} \leftarrow < \mathbf{x}_{\text{new}}, I_{\text{new}}, c_{\text{new}} >$;                          ▷ add node to tree
13:            $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathcal{N}_{\text{new}}\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{P}_{\mathbf{x}_{\mathcal{N}_{\text{nearest}}},\mathbf{x}_{\text{new}}}\}$;
14:            if $I_{\text{new}} > I_{\max}$ then                       ▷ search most informative node
15:                $\mathbf{s}_* \leftarrow \mathbf{x}_{\mathcal{N}_{\text{new}}}$; $I_{\max} \leftarrow I_{\text{new}}$;
16: $\mathcal{P}_s, I_s, c_s \leftarrow$ FindPath($\mathbf{x}_r, \mathbf{s}_*, \mathcal{G}$);                    ▷ find path from root to station
17: $u_s \leftarrow f(I_s, c_s)$;                                                  ▷ calculate path utility
18: return $\mathbf{s}_*, \mathcal{P}_s, u_s$;          ▷ return the station, path to station, and path utility
```

5.2 Search for Highly Informative Stations

Let us now consider the algorithm to search for a highly informative station in more detail. A station is a $\mathbf{x}' \in \mathcal{X}_{free}$ that is highly informative according to a pre-specified information metric. In addition, the search of a station must fulfill the following two requirements: (i) it must be reachable for the robot; (ii) its calculation must have an anytime nature to allow the online realization of the algorithm. To realize these requirements, we propose an adaptation of the kynodinamic RRT algorithm (Section 2.2.2) where we extend the RRT nodes to incorporate an information metric. The RRT algorithm has an anytime nature, and fulfills the first requirement since it is able to account for the robot's kinematics and avoid possible collisions with obstacles. Note that in `SearchStation` we are not concerned about the optimality of the path, but rather about reachability of the next point of interest – the station. Therefore we employ a simple, yet suboptimal path planner such as RRT, which provides a quick way to "sort out" stations that are not reachable by the robot. Using e.g. RRT* (Section 2.2.2) for realizing this test is possible, but computationally less efficient.

Formally, the search of $\mathbf{s}_*$ can be formulated as:

$$\mathbf{s}_* = \underset{\mathbf{x}' \in \mathcal{X}_{free}}{\operatorname{argmax}} I(\mathbf{x}') \text{ s.t. } c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}'}) \leq b, \tag{5.1}$$

where $I(\mathbf{x}')$ is a measure of the expected information at $\mathbf{x}'$ (the particular mea-

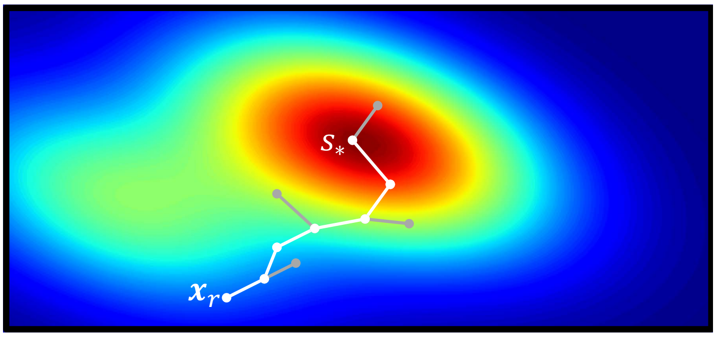


Figure 5.2: Search for highly informative stations. The color scale represents the informativeness, as measured by a predefined information metric, at a particular location. In particular, dark blue corresponds to low informativeness and red represents high informativeness. Algorithm 5.2 selects $\mathbf{s}_*$ as the location with the highest informativeness among all $\mathbf{x} \in \mathcal{V}$.

sure employed is described in Section 5.4), $c(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ is the cost of traversing $\mathcal{P}_{\mathbf{x},\mathbf{x}'}$, and b is a trajectory budget. We assume $c(\cdot)$ as strictly positive, additive[1] and monotonically increasing. Examples of such cost measures are the total time or fuel required to traverse the path; the number of measurements taken along the path can also serve as valid cost function. Here we choose time as path cost.

SearchStation. The key steps of the `SearchStation` algorithm are summarized in Algorithm 5.2. Algorithm 5.2 is an extension of the RRT algorithm for information gathering tasks. The first modification with respect to RRT consists of an extension of the standard RRT node $\mathcal{N} \in \mathcal{V}$. Like in (Hollinger and Sukhatme, 2014), in Alg. 5.2 a node includes (i) the spatial location of node $\mathbf{x}_{\text{new}}$, (ii) the expected information I_{new} at $\mathbf{x}_{\text{new}}$, and (iii) the cost c_{new} of reaching $\mathbf{x}_{\text{new}}$ from $\mathbf{x}_r$ (line 12). The latter is computed using the robot motion model, while I_{new} is calculated with function `InformationS` (line 11). For more details on the information calculation we refer the reader to Section 5.4. In addition to RRT we include as well a budget constraint b (line 10).

The ultimate goal of Alg. 5.2 is selecting $\mathbf{s}_*$ that has the highest informativeness (lines 14-15). In addition we calculate the path $\mathcal{P}_s$ that drives the robot from its current position to the station with function `FindPath` (line 16). This function also outputs the information I_s and the total cost c_s of $\mathcal{P}_s$. Given I_s and c_s we calculate the utility u_s of $\mathcal{P}_s$ (line 17). Details about the calculation of utility function (5.3) and information of the path (5.4) are given in Section 5.3

[1]If we have two partial trajectories $\mathcal{P}_{1,2}$ and $\mathcal{P}_{2,3}$ that can be concatenated to yield a trajectory $\mathcal{P}_{1,3}$, a cost function is considered additive if $c(\mathcal{P}_{1,3}) = c(\mathcal{P}_{1,2}) + c(\mathcal{P}_{2,3})$.

and Section 5.4, respectively.

To finalize this section, we depict the concept behind Alg. 5.2 in Fig. 5.2.

5.3 Informative Path Planner using RRT*

The goal of the informative path planner is to refine $\mathcal{P}_s$ calculated with Alg. 5.2. Here we aim to calculate a path that fulfills the following two requirements: (i) it is feasible given the robot's kinematics and does not incur collisions with obstacles; and (ii) it is efficient, in the sense of maximizing the information gathering, while minimizing the path cost. Formally, we aim to find the optimal path $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ between states $\mathbf{x}_r$ and $\mathbf{s}_*$. This can be formulated as the following optimization problem:

$$\begin{aligned} \underset{\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*} \subset \mathcal{X}_{free}}{\operatorname{argmax}} \quad & f(I(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}), c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})), \\ \text{s.t.:} \quad & c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}) < b. \end{aligned} \tag{5.2}$$

Here $I(\cdot)$ and $c(\cdot)$ are functions that evaluate the information and cost of the path, respectively, $f(\cdot,\cdot)$ is a function that evaluates the information-cost trade-off (the utility), and b is a budget for the path cost. We summarize the `InformativePlanner` in Algorithm 5.3.

InformativePlanner. Algorithm 5.3 is an extension of RRT* (Section 2.2.2) that allows a robot to gather information efficiently and autonomously. In contrast to RRT*, here we replace the concept of the path cost by the concept of utility. The utility u of a path is a value that weights the importance of a path. In this paper, we formulate the utility, given by $f(\cdot,\cdot)$, so that it fulfills our information gathering objective. That is, we aim to gather as much information as possible along the path towards $\mathbf{s}_*$ while generating trajectories with the minimum cost. This implies that $f(I(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}), c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}))$ should grow with $I(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})$ and decrease as $c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})$ becomes large. We represent this trade-off with the following function:

$$f(I(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}), c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})) = \alpha \frac{I(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})}{c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})}, \tag{5.3}$$

with α a coefficient that determines the trade-off. $c(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})$ is chosen as a time needed to traverse $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$. $I(\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})$, which we calculate with function `InformationP`, will be explained in detail in Section 5.4. Let us also emphasize that the formulation of (5.3), which includes both the information and cost, allows us to extend the algorithm to applications where taking a measurement incurs a high cost. That is, in such an application, gathering information becomes

Algorithm 5.3. `InformativePlanner`$(\mathbf{x}_r, \mathbf{s}_*, \boldsymbol{\theta}_*, \mathbf{X}, b, \mathcal{X}_{free})$

```
1: $\mathcal{V} \leftarrow \{\mathbf{x}_r\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathcal{P}_p \leftarrow \emptyset$; $u_p \leftarrow -\infty$;
2:
3: while ! StopPlanner do
4:     $\mathbf{x}_{rand} \leftarrow$ SampleFree($\mathcal{X}_{free}$);
5:     $\mathcal{N}_{nearest} \leftarrow$ Nearest($\mathbf{x}_{rand}, \mathcal{V}$);
6:     $\mathbf{x}_{\text{new}} \leftarrow$ Steer($\mathbf{x}_{nearest}, \mathbf{x}_{rand}, \eta$);
7:     ▷ Check trajectory feasibility and budget constraint
8:     if CollisionFree($\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{\text{new}}}, \mathcal{X}_{free}$) AND $c_{\max} < b$ then
9:         $\mathbf{x}_{\max} \leftarrow \mathbf{x}_{nearest}$;
10:        ▷ Calculate cost and information of path to node
11:        $c_{\max} \leftarrow c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}_{\text{nearest}}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{nearest},\mathbf{x}_{\text{new}}})$;
12:        $I_{\max} \leftarrow$ InformationP($\mathbf{x}_{\text{new}}, \mathcal{N}_{nearest}, \mathcal{G}, \boldsymbol{\theta}_*, \mathbf{X}$);
13:        $u_{\max} \leftarrow f(I_{\max}, c_{\max})$;                          ▷ calculate path utility
14:        $\mathcal{V}_{\text{near}} \leftarrow$ Near($\mathbf{x}_{\text{new}}, \mathcal{V}$);
15:        for $\mathcal{N}_{\text{near}} \in \mathcal{V}_{\text{near}}$ do                         ▷ choose parent
16:            if CollisionFree($\mathcal{P}_{\mathbf{x}_{near},\mathbf{x}_{\text{new}}}, \mathcal{X}_{free}$) then
17:                $c_{\text{new}} \leftarrow c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}_{\text{near}}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{near},\mathbf{x}_{\text{new}}})$;
18:                $I_{\text{new}} \leftarrow$ InformationP($\mathbf{x}_{\text{new}}, \mathcal{N}_{\text{near}}, \mathcal{G}, \boldsymbol{\theta}_*, \mathbf{X}$);
19:                $u_{\text{new}} \leftarrow f(I_{\text{new}}, c_{\text{new}})$;
20:                if $u_{\text{new}} > u_{\max}$ then          ▷ parent that results in highest path utility
21:                    $\mathbf{x}_{\max} \leftarrow \mathbf{x}_{\mathcal{N}_{\text{near}}}$; $c_{\max} \leftarrow c_{\text{new}}$; $I_{\max} \leftarrow I_{\text{new}}$; $u_{\max} \leftarrow u_{\text{new}}$;
22:        $\mathcal{N}_{\text{new}} \leftarrow < \mathbf{x}_{new}, I_{\max}, c_{max} >$;                   ▷ add node to tree
23:        $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathcal{N}_{\text{new}}\}$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{\mathcal{P}_{\mathbf{x}_{\max},\mathbf{x}_{\text{new}}}\}$;
24:        $\mathcal{V}'_{\text{near}} \leftarrow$ CyclesFree($\mathbf{x}_{\text{new}}, \mathcal{V}_{\text{near}}, \mathcal{G}$);   ▷ discard the nodes that will create a cycle
25:        for $\mathcal{N}'_{\text{near}} \in \mathcal{V}'_{\text{near}}$ do                        ▷ rewire near nodes
26:            $c_{\text{new}} \leftarrow c(\mathcal{P}_{\mathbf{x}_r,\mathbf{x}_{\text{new}}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{new},\mathbf{x}'_{near}})$;
27:            $I_{\text{new}} \leftarrow$ InformationP($\mathbf{x}'_{near}, \mathcal{N}_{\text{new}}, \mathcal{G}, \boldsymbol{\theta}_*, \mathbf{X}$);
28:            $u_{\text{new}} \leftarrow f(I_{\text{new}}, c_{\text{new}})$;
29:            if CollisionFree($\mathcal{P}_{\mathbf{x}_{new},\mathbf{x}'_{near}}, \mathcal{X}_{free}$) then
30:                if $u_{\text{new}} > f(I'_{near}, c'_{near})$ then           ▷ rewire if higher path utility
31:                    $I'_{near} \leftarrow I_{\text{new}}$; $c'_{near} \leftarrow c_{\text{new}}$;
32:                    $\mathcal{N}_{\text{parent}} \leftarrow$ Parent($\mathcal{N}'_{\text{near}}$);
33:                    $\mathcal{E} \leftarrow \mathcal{E} \setminus \{\mathcal{P}_{\mathbf{x}_{parent},\mathbf{x}'_{near}}\} \cup \{\mathcal{P}_{\mathbf{x}_{\text{new}},\mathbf{x}'_{near}}\}$;
34: $\mathcal{P}_p, u_p \leftarrow$ FindBestPath($\mathbf{x}_r, \mathbf{s}_*, \mathcal{G}$);                  ▷ find best path to station
35: return $\mathcal{P}_p, u_p$;
```

expensive and therefore the exploration algorithm should be able to balance the trade-off information-cost. In contrast to prior work, our algorithm incorporates this trade-off. The definition of (5.3) introduces an additional difference between RRT* and Alg. 5.3, since (5.3) is non-monotonic.

Non-monotonicity of the utility function. The non-monotonicity of (5.3) compromises the optimality guaranty of RRT* (Section 2.2.2). Despite this, our simulation results suggest that Alg. 5.3 is still able to approach the optimal solution, as shown in Sec. 5.6.2. Furthermore, the non-monotonicity of our utility function requires the inclusion of a mechanism to avoid the creation of cycles in the tree. A cycle is a sequence of vertices starting and ending at the same vertex (Godsil and Royle, 2013). Cycles might appear in the graph due to our specific choice of the utility function, which is not guaranteed to increase monotonically with the growing tree. Cycles can be created during the rewiring process if a node $\mathcal{N}'_{\text{near}}$, which belongs to the path that connects the robot's position with $\mathcal{N}_{\text{new}}$, could be reached with a higher utility from $\mathcal{N}_{\text{new}}$ than its previous utility. This problem does not arise with a monotonic utility function, since the inclusion of a new node always incurs a higher cost. Here, however, a longer path could have a higher utility if we gather more information along it. In order to generalize RRT* to a larger class of utility functions, such as (5.3) or the one used by Charrow et al. (2015), we propose a procedure to eliminate cycles in $\mathcal{G}$. This is implemented in function `CyclesFree` (line 24). This function takes as input $\mathbf{x}_{\text{new}}$ and $\mathcal{V}_{\text{near}}$. Then it iterates over $\mathcal{N}_{near} \in \mathcal{V}_{\text{near}}$, and removes those $\mathcal{N}_{near} = \{\mathbf{x}_{near}, I_{near}, c_{near}\}$ where $\mathbf{x}_{near} \in \mathcal{P}_{\mathbf{x}_r,\mathbf{x}_{\text{new}}}(\mathcal{G})$.

Best path calculation. Once the robot finishes the execution of the algorithm, which is given by the *StopPlanner* criterion, it calculates the best path $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ in terms of utility with function `FindBestPath`. This function connects $\mathbf{s}_*$ to $\mathbf{x} \in \mathcal{V} \in \mathcal{G}$ that are closer than a distance η from it. Then we calculate the utility of all those possible paths and choose the one with the highest utility. The utility, together with the computed path form the algorithm output. In case the algorithm does not find a suitable path, it outputs an empty path with minus infinite utility.

5.4 Information Metric

Algorithm 5.1 relies on an information metric to evaluate the informativeness of a $\mathbf{x} \in \mathcal{X}_{free}$ and a $\mathcal{P}_{\mathbf{x},\mathbf{x}'} \subset \mathcal{X}_{free}$. In Chapter 4 we employed entropy at individual locations as information metric. However, in Chapter 4 we also discussed that more complex information metrics could be considered to, e.g., take into account the cross-correlations between $\mathbf{x}, \mathbf{x}' \in \mathcal{X}_{free}$. In this section, we extend the discussion about possible information metrics. In particular, we argue about the use of MI (Section 2.3.2) and mean entropy as information metrics. To finalize, we motivate our particular choice: mean entropy as information metric.

Mutual information. MI has been extensively employed in the information gathering literature (see Section 3.2). Indeed, it seems like a perfect fit for selecting information sampling locations because it takes into account the cross-correlations of the test points. However, we observed that MI is not adequate for algorithms that require an extensive computation of the information metric, as it is pointed out in (Stranders et al., 2009). Due to the long computation time of the MI (see Table 2.1), the addition of nodes to the tree in Algs. 5.2 and 5.3 would be sacrificed to calculate the information metric. This reduces the size of the explored region of the state space given a certain exploration time, resulting in a loss of performance. There exists efficient algorithms to calculate MI, like e.g. the one introduced in (Ramirez-Paredes et al., 2016). However, to the best of our knowledge, they are not applicable to GPs.

Mean entropy. As a viable alternative, we decided to use mean entropy $\bar{H}(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ as information metric. This is primarily motivated by the concept of entropy rate, which is a limit of the joint entropy as the number of observations grows (Cover and Thomas, 2012). Entropy rate converges to the mean entropy as a special case. Formally we define $\bar{H}(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ as:

$$I(\mathcal{P}_{\mathbf{x},\mathbf{x}'}) = \bar{H}(\mathcal{P}_{\mathbf{x},\mathbf{x}'}) = \frac{1}{|\mathcal{P}_{\mathbf{x},\mathbf{x}'}|} \sum_{\mathbf{x}_i \in \mathcal{P}_{\mathbf{x},\mathbf{x}'}} H(\mathbf{x}_i), \tag{5.4}$$

with $H(\mathbf{x}_i)$ calculated according to (2.20).

$\bar{H}(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ presents a diminishing property (Krause et al., 2008) similar to the MI. For example, imagine two paths that have the same sum of entropies. An averaged entropy would favor the one that requires fewer measurements. This is a desirable property that would be crucial if we considered the cost of taking a measurement. Note also that $H(\mathbf{x}_i)$ only needs to be calculated once for each $\mathbf{x}_i \in \mathcal{V}$, as we can save $H(\mathbf{x}_i)$ and reuse it each time we evaluate a path that includes $\mathbf{x}_i$. In summary, $\bar{H}(\mathcal{P}_{\mathbf{x},\mathbf{x}'})$ is computationally efficient and favors those paths that have higher information at smaller cost.

In the following, we specify the calculation of functions `InformationP`, `InformationS`, and `FindPath`. `InformationP` computes the information of $\mathcal{P}_{\mathbf{x}_r,\mathbf{x}_{new}}$ with (5.4). In contrast `InformationS` computes the information of a single $\mathbf{x}_{new} \in \mathcal{X}_{free}$ – a potential $\mathbf{s}_*$. This is also calculated with (5.4), where $\mathcal{P}_{\mathbf{x},\mathbf{x}'} = [\mathbf{x}_{new}]$. Finally, `FindPath` searches $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ with highest utility. Notice that in this case the information does not need to be calculated since it is stored in each of the individual nodes.

5.5 Computational Complexity

In this section we perform an assessment of the algorithm's overall time complexity. To this end we split the analysis in two main parts: (i) updating the GPs model, which corresponds to estimating $\boldsymbol{\theta}$, and (ii) planning $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$.

The complexity of the estimation of $\boldsymbol{\theta}$ is given by the inversion of the $\mathbf{K}$ matrix, which is calculated with (2.2). The basic complexity of this matrix inversion is $\mathcal{O}(n^3)$, with n the number of collected measurements (Section 2.3.3).

The planning of $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ corresponds to the execution of Algs. 5.2 and 5.3, which are based on RRT and RRT*, respectively. For N_p tree's nodes, the basic complexity of RRT and RRT* is $\mathcal{O}(N_p \log N_p)$ to create the tree, and $\mathcal{O}(N_p)$ to query the best path from the tree. We refer to (LaValle and Kuffner, 2000) and (Karaman and Frazzoli, 2011) for a detailed analysis of the algorithms' time complexity.

In addition to the basic RRT and RRT* algorithms we must include the computation of the information metric. As we explained in Sections 5.2 and 5.3, the calculation of the information metric is delimited to $\{\mathbf{x} \mid ||\mathbf{x} - \mathbf{x}_r||_2^2 \leq b\}$. Moreover, we assume that $\mathbf{x}, \mathbf{x}'$ are uncorrelated if $k(\mathbf{x}, \mathbf{x}') \leq \sigma_n/10$, with $k(\mathbf{x}, \mathbf{x}')$ given by (2.4). This simplification is possible because of the definition of $k(\mathbf{x}, \mathbf{x}')$ that decreases with distance. The presence of a trajectory budget, together with the simplification regarding $k(\mathbf{x}, \mathbf{x}')$, allows us to reduce the time complexity of calculation of the information metric from $\mathcal{O}(n^3)$, which is unbounded and grows with the number of measurements, to $\mathcal{O}(n_b^3)$, which is bounded by parameter b and does not necessarily grow as the number of measurements increases. Let us also remark that we employ this simplification for the estimation of $\boldsymbol{\theta}$ as well. This implies that we only consider those measurements that are correlated with those that lie within the bounded region.

In summary, the computational complexity of the overall algorithm (one iteration of Algorithm 5.1) is $\mathcal{O}(n_b^3 + N_p \log N_p)$. We remark that RRT and RRT* algorithms have an anytime nature. Therefore, N_p will depend on the running time of the planning algorithm, which is a user-defined criterion.

5.6 Simulations and Discussion of Results

In this section we present the simulations setup and performance results of Alg. 5.1. We divide this analysis in two main parts. First, we compare Alg. 5.3 against two state-of-the-art algorithms. Second, we evaluate Alg. 5.1 in an information gathering task of an unknown physical process that takes place within a complex environment.

5.6.1 Simulations Setup

Here we describe the simulation setup used to validate Alg. 5.1 with synthetic and real data. We test our algorithm with simulated and real data as ground truth $\mathbf{y}_G(\mathbf{X}_G)$. These data are stored as grids of 20×20 cells with a resolution of 5 and 10 centimetres. For the simulation we assume a round-shaped holonomic robot with 5 centimetres radius that moves with a constant speed of 0.2 metres per second. Here we employ a holonomic-robot (e.g. robot from Fig. 4.6) to abstract the active sensing strategy from the robot's motion. This is a common strategy employed in works like e.g. (Hollinger and Sukhatme, 2014), one of our benchmark algorithms. Nevertheless, Alg. 5.1 is valid for arbitrary robot's dynamics given the restrictions imposed by RRT or RRT* algorithms. We also assume that the robot needs an infinitesimally small time to take a measurement. The robot can move in a continuous space, and we assume that measurements taken within one cell of the grid are equal.

For the `SearchStation` and `InformativePlanner` algorithms we select the following parameters: the parameter η in the `Steer` function and the distance employed by the function `Near` to search neighbors nodes are both set to the measurement's resolution, i.e., 5 or 10 centimetres depending on the concrete simulation. We select the running time as stop criterion for the `SearchStation` and `InformativePlanner` algorithms, with a value of 5 and 10 seconds respectively. We consider a trajectory budget $b = 10$ seconds, which corresponds to a planning horizon of 2 metres given the robot's speed. We initialize $\boldsymbol{\theta}$ to the following values $l = 1$, $\sigma_f = 1$, $\sigma_n = 0.1$. We repeat each of the simulations 40 times.

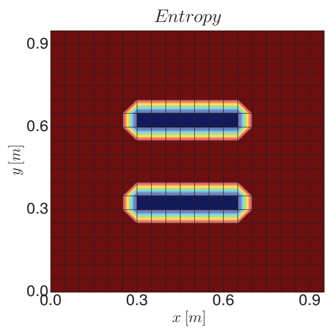
5.6.2 Analysis of the Informative Path Planner

5.6.2.1 Setup

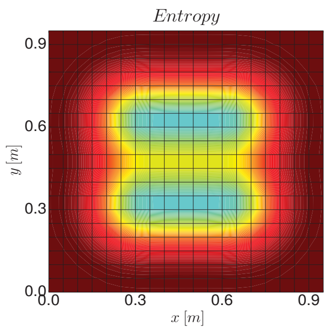
In this section, the objective is to analyze the individual performance of Alg. 5.3. This assumes an available GP model with fixed hyperparameters. The goal of Alg. 5.3 is, given this model, to find the trajectory that optimizes (5.2) as fast as possible. Since our information metric corresponds to the mean entropy, we simulate two scenarios with distinct entropy structures (see Figure 5.3):

- Scenario 1 recreates a physical process with low spatial correlation in which a robot has already gathered two patches of measurements. The blue areas correspond to the measured areas and the red areas to the non-measured positions. We employ the following $\boldsymbol{\theta}$: $l = 0.02$, $\sigma_f = 0.084$, $\sigma_n = 0.02$.

- Scenario 2 recreates the same scenario, but now we consider a process with higher spatial correlation. Here we set $l = 0.13$, $\sigma_f = 0.084$, $\sigma_n = 0.09$.



(a) Scenario 1.



(b) Scenario 2.

Figure 5.3: Simulation scenarios used to test the performance of our informative path planner algorithm.

We fix $\mathbf{x}_r$ to $(x = 0.2, y = 0.5)$ and $\mathbf{s}_*$ to $(x = 0.8, y = 0.5)$ for all the simulations runs.

5.6.2.2 Choice of the Information Function

In this section we compare our proposed information metric (5.4) with MI. We carry out the analysis for scenario 2, as it is the one that presents a higher spatial correlation, which translates into a higher informativeness of (5.4). Let us remark that here we run Alg. 5.3 and replace (5.4) with the mutual information. Results correspond to the average over 40 simulations runs for a planning time of 180 seconds. We compare (5.4) with MI, in terms of: (i) time to find a first $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ (t_{first}); (ii) posterior entropy that results after measuring along $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$ output by Alg. 5.3; and (iii) cost of $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$. Note that posterior entropy $H(Y_{\mathbf{X}_G}|\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})$ can be calculated with (2.21), where $Y_{\mathbf{X}_G}$ is a random variable that estimates $\mathbf{y}_G(\mathbf{X}_G)$.

We show the results of the evaluation in Table 5.1. According to Table 5.1, (5.4) finds a first path seven times faster than MI, reduces the posterior entropy by one half, and the path calculated with (5.4) has a slightly smaller cost than the one calculated with MI. This lets us conclude that our proposed information metric, given by (5.4), outperforms the MI in an online sensing setting that requires an extensive computation of the information metric.

5.6.2.3 Performance Analysis

We benchmark Alg. 5.3 against two state-of-the-art sampling-based informative path planning algorithms:

	t_{first}[s]	Entropy [bits]	Cost [s]
Mean Entropy	**6.31**	**−6.93**	**6.79**
Mutual Information	46.71	−3.54	6.86

Table 5.1: Analysis of the information function. We compare (5.4) with MI.

- (i) the technique of Yang et al. (2013), where multiple paths are obtained by running an RRT planner several times, and then paths are evaluated according to the information metric. This algorithm we will term `Multiples RRT`;
- (ii) the RIG-tree planner (Hollinger and Sukhatme, 2014), to which we will refer as `RIG Algorithm`.

In both cases, (5.3) is employed. For the RIG-tree we use one of the approaches suggested by the authors in (Hollinger and Sukhatme, 2014). Specifically, we consider the pruning based on the heuristic that the utility function is modular. Here, we defined two nodes as co-located if they are within the same cell of the grid. For more details about the implementation, we refer the reader to the original paper (Hollinger and Sukhatme, 2014). We also tested the other alternatives proposed by the authors, but they offered a lower performance in our particular setup.

Moreover, the two benchmark algorithms are not designed to meet a goal constraint; they explore the environment with no goal. This makes a comparison with our algorithm difficult. We solve this by selecting all samples that are closer than a distance η from the goal $\mathbf{s}_*$, and then connecting them to $\mathbf{s}_*$. This results in paths that link $\mathbf{x}_r$ with $\mathbf{s}_*$. We analyze in Figure 5.4 the performance of compared algorithms as a function of planning time.

Utility analysis. The difference in terms of utility (see first row of Fig. 5.4) with respect to the other algorithms ranges between 0.05 and 0.15 bits per second. We notice as well that the RIG algorithm only presents a minor improvement of the utility as the planning time increases. We believe this is due to the inclusion of the goal constraint, which the RIG algorithm is not able to handle.

Algorithm complexity analysis. Another important figure that characterizes the algorithm is the number of nodes spanned by the path planner. We observe in the second row of Fig. 5.4 that the `Multiples RRT` variant has a limited number of nodes since we reset the algorithm each time we find a new path. Furthermore, Alg. 5.3 requires a larger number of nodes than the `RIG Algorithm`.

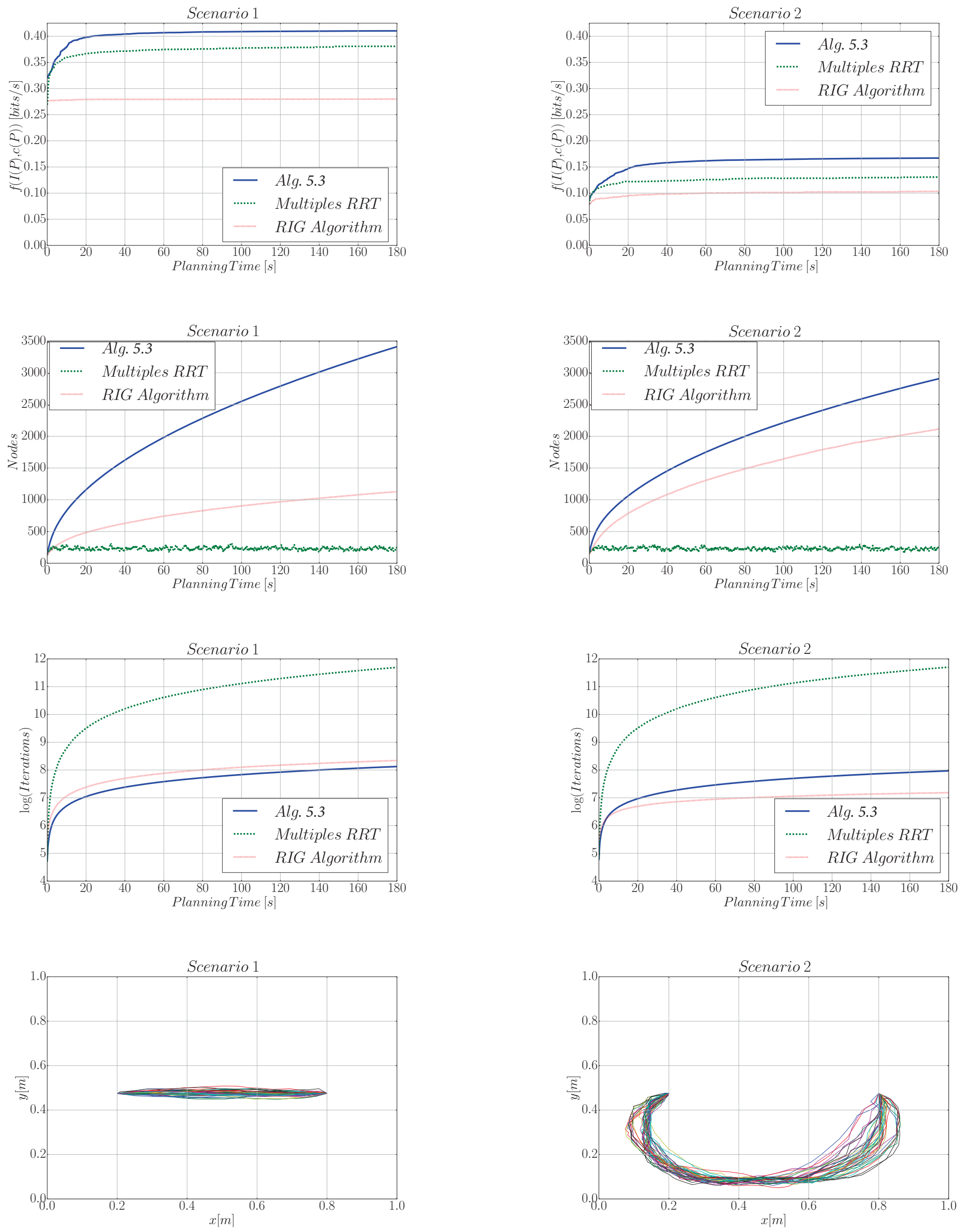


Figure 5.4: Performance analysis of Alg. 5.3 as we increase the planning time. Here, from the first to the last row, we evaluate the utility of the best path, the number of nodes spanned by the tree, and the algorithm's complexity that is represented as the curve number of iterations vs. planning time. In addition, we plot the 40 paths output by Alg. 5.3 given 180 seconds of planning time.

The latter employs a smaller number of nodes because of the pruning strategy that removes those co-located nodes that have a smaller utility than the new added node. However, this does not lead to a higher complexity per iteration, as we can observe in the third row of Fig. 5.4, which shows the number of iterations of the algorithm vs. the planning time. Here, the `Multiples RRT` alternative offers the lowest complexity.

Paths output by Alg. 5.3. In the last row of Fig. 5.4 we depict the paths output by Alg. 5.3. We observe that, for scenario 1, the robot takes the path that has the most information and takes the least time, which results in a straight line. However, in scenario 2 the straight line corresponds to a path that has little information, and therefore the robot takes a path that is longer but allows it to gather more information as it visits not yet measured locations. These results illustrate the need of defining an utility function that trades off the information gathering and the path's cost (5.3). Moreover paths output suggest that Alg. 5.3 is able to approach the optimal solution of (5.2).

Posterior entropy analysis. We showed in Figure 5.4 that Alg. 5.3 outperforms the considered state-of-the-art approaches in terms of our information function. However, this does not necessarily imply that our algorithm can find a more informative path. In order to make a fair comparison between the three considered algorithms we evaluate them in terms of an independent metric: $H(Y_{\mathbf{X}_G}|\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*})$. In addition, we compare the cost of the resulting paths. Table 5.2 shows the results for the three algorithms in the two scenarios, for 180 seconds of planning time. We can conclude that Alg. 5.3 offers the best ratio entropy-cost for all scenarios. Specifically, the most relevant scenario for this evaluation is scenario 2 since it presents a higher spatial correlation. Here Alg. 5.3 results in a twofold and sevenfold increase respect to `RIG Algorithm` and `Multiples RRT`, respectively, while offering a similar path cost.

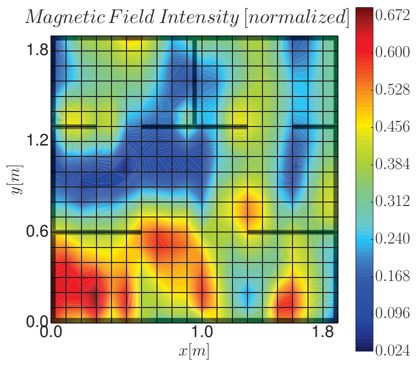
5.6.3 Analysis of the Exploration Strategy

5.6.3.1 Setup

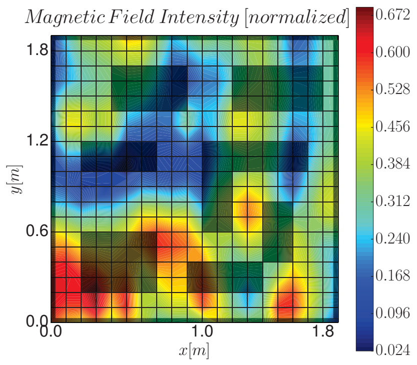
We validate in this section Alg. 5.1 in an environment populated with obstacles. To this end we carried out simulations in two environments that are shown in Figure 5.5. Scenario A emulates a corridor with different rooms, while scenario B considers thicker block-like obstacles. The simulations employ real data, collected with a ground-based robot at DLR (see Fig. 4.6), which corresponds to a magnetic field intensity in an indoor environment.

		Entropy [bits]	Path Cost [s]
Scenario 1	Alg. 5.3	**383.25**	**3.03**
	`Multiples RRT`	383.75	3.26
	`RIG Algorithm`	383.86	4.24
Scenario 2	Alg. 5.3	$-$**6.93**	6.79
	`Multiples RRT`	-3.68	6.23
	`RIG Algorithm`	0.03	**5.65**

Table 5.2: Posterior entropy and path cost evaluated over the complete environment after measuring $\mathcal{P}_{\mathbf{x}_r,\mathbf{s}_*}$, calculated for 180 seconds of planning time.



(a) Scenario A.



(b) Scenario B.

Figure 5.5: Scenarios employed to test Alg. 5.1. Black polygons correspond to the obstacles and the underlying picture is the magnetic field intensity we aim to explore.

5.6.3.2 Performance Analysis

The goal of our information gathering algorithm is to reduce the RMSE, calculated with (4.2), as fast as possible. In this section, we compare Alg. 5.1 with the following strategies:

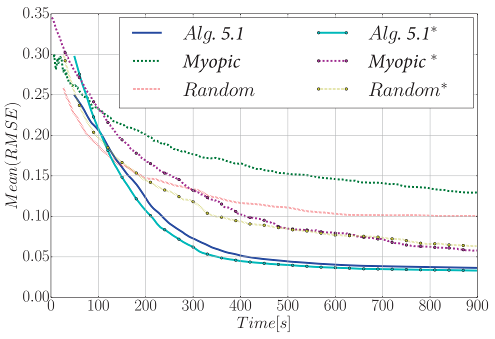
- Myopic approach: according to the notation introduced in Sec. 2.2.1, we assume $\mathcal{L}_{\mathcal{X}_{free}}(\mathcal{V}_{\mathcal{X}_{free}}, \mathcal{E}_{\mathcal{X}_{free}})$ defined over scenarios A and B, where $\mathcal{V}_{\mathcal{X}_{free}}$ corresponds to the cells depicted in Fig. 5.5. A myopic approach corresponds to a widely used state-of-the-art strategy for information gathering that has been employed in Chapter 4, as well as in works such as (Krause and Guestrin, 2007b; Marchant and Ramos, 2012).

- Random approach: an RRT is grown from $\mathbf{x}_r$ for the same planning time and budget b as in `SearchStation` algorithm. The next station is selected randomly from the leaves of the RRT and the associated path is employed to reach the station.

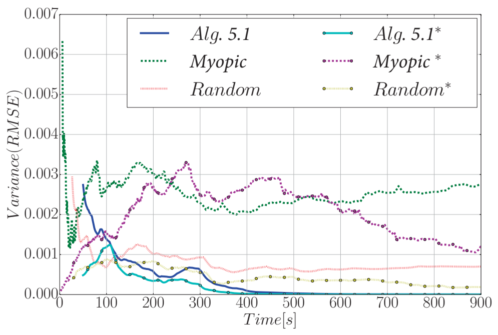
Let us clarify our motivation to employ a myopic and random benchmarks. The algorithms `Multiples RRT` and `RIG Algorithm` are one-shot algorithms. In other words, given an *a priori* known model the algorithms run for some user pre-defined time and produce a path. This is also what our informative planner (Algorithm 5.3) is designed to do. This was the motivation to compare the algorithms with each other. In this section we evaluate our full exploration strategy (Algorithm 5.1) that is able to explore an *a priori* unknown process. Let us remark again that `Multiples RRT` and `RIG Algorithm` do not consider this feature. In the literature, two common approaches to deal with an exploration of an unknown process are a random trajectory and a myopic one. Therefore, we used these two to benchmark Alg. 5.1. We believe that the proposed evaluation is the fairest to compare each of the different approaches according to its capabilities.

RMSE analysis. Figure 5.6 shows the mean and variance of the RMSE for all executions. This is done for the different strategies and for both scenarios. We also test the methods under assumption that the optimal $\boldsymbol{\theta}$ are known and fixed (listed with an asterisk sign). Our goal is to shift the Mean(RMSE) curve to the left bottom corner. This implies a small RMSE that is achieved efficiently in terms of time resources. Observe that Alg. 5.1 clearly outperforms the other strategies. The myopic approach with optimal $\boldsymbol{\theta}$ is the only one that offers a comparable performance after the 900 seconds mission. Note, however, that it assumes an *a priori* known model, which is unrealistic for an actual information gathering task.

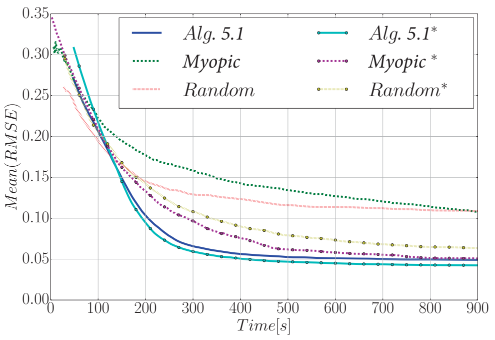
Solution quality. We analyze in Figure 5.7 the quality of the solution respect to the best possible performance that we could obtain by systematic sampling. We consider the best possible solution as the estimation over the complete environment that results after measuring at all $\mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}}$. Let us remark that this solution considers optimal $\boldsymbol{\theta}$. We show in Figure 5.7 the percentage in terms of the RMSE respect to the best solution that we obtain with greedy, random, and Alg. 5.1. A percentage of 100% indicates that Alg. 5.1 is able to achieve an RMSE that is equal to the best possible RMSE that we could obtain. According to Figure 5.7, after 900 seconds Alg. 5.1 is able to obtain a RMSE that is the 90%, while the greedy and random approach achieve only 40%.



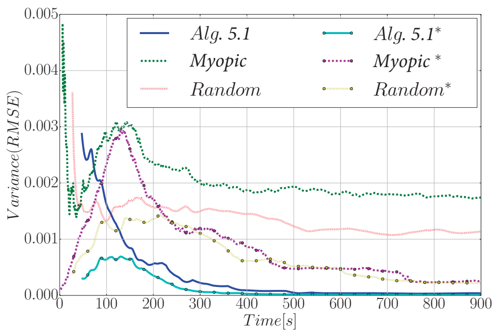
(a) Scenario A. Mean.



(b) Scenario A. Variance.



(c) Scenario B. Mean



(d) Scenario B. Variance.

Figure 5.6: RMSE between the estimation of the process and the ground truth. Top: scenario A. Bottom: scenario B. We represent the mean and variance of the RMSE over the 40 simulations we carried out. Here we test three different trajectories: (i) Alg. 5.1, (ii) a myopic approach, and (iii) random trajectories. For all of them we compare their performance assuming: (i) no prior knowledge about the process, which implies an online learning of $\boldsymbol{\theta}$, and (ii) assuming they know the optimal $\boldsymbol{\theta}$ *a priori* (marked with an asterisk).

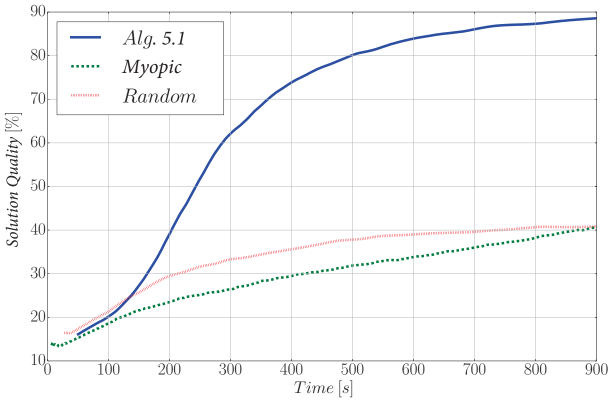


Figure 5.7: Quality of the solution achieved by Alg. 5.1, a myopic, and a random trajectories after a 900 seconds mission.

Comparison with RIG algorithm. According to the simulation results, we can conclude that Alg. 5.1 outperforms the myopic and random approaches. In order to get a better understanding of our proposed algorithm capabilities we compare our full exploration strategy with the `RIG Algorithm`. Specifically, we consider the following for the `RIG Algorithm`: (i) the model is *a priori* known; i.e. we know the GPs hyperparameters and they do not need no be estimated, (ii) the utility function corresponds to the MI, as suggested by the authors in (Hollinger and Sukhatme, 2014), and (iii) the planning time is 600 seconds and then we let the robot follow and measure along the planned path. Let us remark that these are favorable conditions for the `RIG Algorithm` as our algorithm assumes an *a priori* unknown model that needs to be estimated online. We run the simulation 40 times starting from different positions in the environment. Then we calculated the RMSE after measuring along the calculated path. The average RMSE that we obtained for the `RIG Algorithm` is 0.27, which is much higher than the one obtained by Alg. 5.1 that is 0.05 (see Table 5.3). We believe that the lower performance of the `RIG Algorithm` lies on the fact that the algorithm grows a single tree to explore the complete environment. Notice that the complexity of adding a new sample grows exponentially as the tree grows, which difficult the exploration of the complete environment. In contrast, our algorithm runs multiple consecutive trees using our devised two-step approach that permits an efficient online exploration.

5.6.3.3 Hyperparameters Analysis

Finally, in Figure 5.8 we show the evolution of the estimated $\boldsymbol{\theta}$ (and their variances) for Scenario A. To estimate $\boldsymbol{\theta}$ we use the LML (2.5). The LML is a

	RMSE at $t = 600s$
Alg. 5.1	0.05
`RIG Algorithm`	0.27

Table 5.3: RMSE at $t = 600s$ resulting after exploring scenario B. For this comparison Alg. 5.1 employs (5.3), (5.4), and `RIG Algorithm` uses MI.

differentiable function and, therefore, conjugate gradients are a proper alternative to obtain $\boldsymbol{\theta}_*$. Let us remark that the non-convexity of the LML could drive the optimizer to local minima. To overcome this issue we run the optimization algorithm several times (10 to be specific) with initial values drawn randomly from a uniform distribution defined over the set of feasible hyperparameter values. Then we pick the best solution.

We can observe that Alg. 5.1 converges slightly slower than the other approaches. In the myopic and random approach, the process is re-estimated more often as compared to Alg. 5.1. This can explain a faster convergence of $\boldsymbol{\theta}$. However, a slower convergence of $\boldsymbol{\theta}$ does not imply an inferior performance in terms of the RMSE respect to these two strategies, as shown in Figs. 5.6 and 5.7. We can also observe that σ_n for the random trajectory converges to a slightly higher value compared to the myopic approach and Alg. 5.1. We believe this is due to the fact that the random trajectory often repeats measurements at the same positions and this has an impact on the learned $\boldsymbol{\theta}$.

5.7 Experiments and Discussion of Results

Now we test Alg. 5.1 in an experiment [2] employing a real ground-based holonomic robot that is used to autonomously explore a magnetic field intensity within an indoor laboratory environment populated with obstacles (see Figure 5.9).

5.7.1 Experimental Setup

The information gathering takes place in an environment that measures 3 by 6 metres. It contains 8 boxes of different sizes that are arbitrarily placed. We remark again that we assume an *a priori* known location of obstacles in order to abstract from the mapping of the environment. Also, we employ the same sensor and hardware setup as described in Section 4.3.

[2] A video that shows the experiment execution can be found in: `https://vimeo.com/253576265`; `https://rebrand.ly/sampld3c1`.

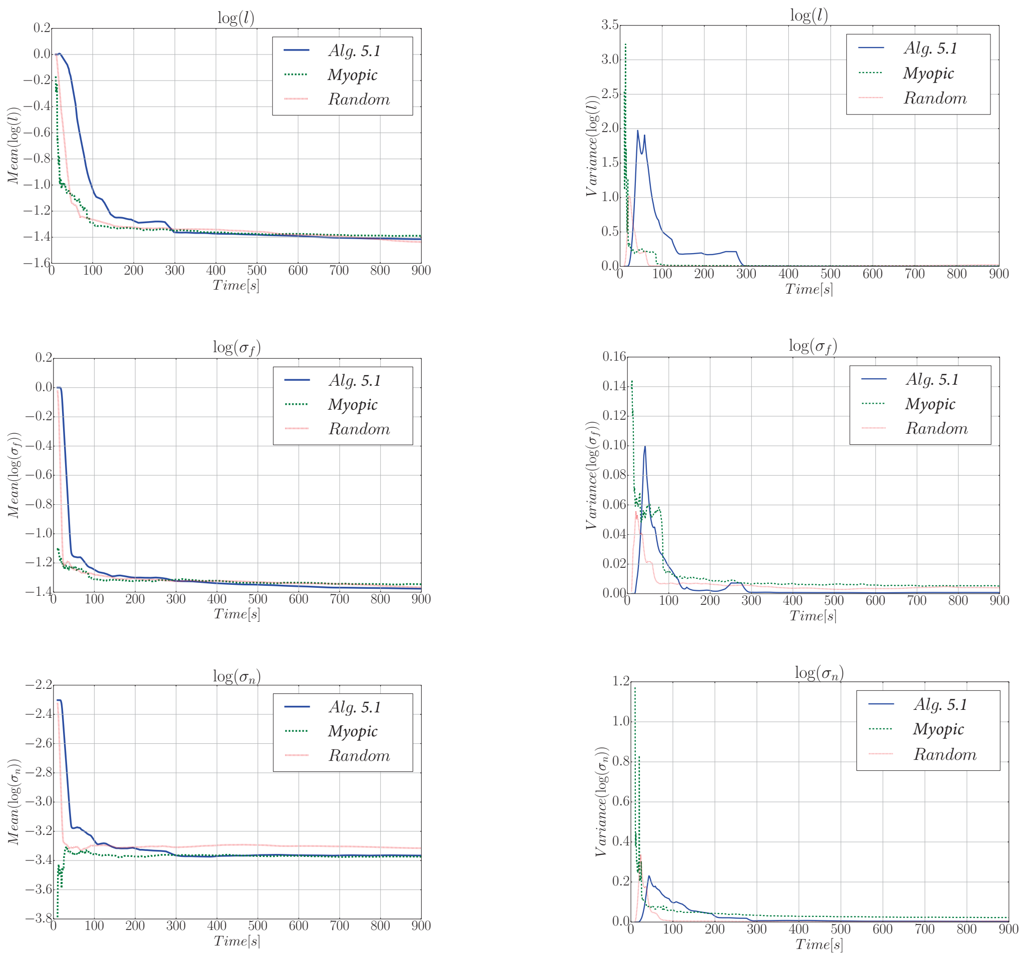


Figure 5.8: GPs hyperparameters learned during the information gathering task for Alg. 5.1, a myopic, and a random trajectory. We represent the mean and variance over 40 simulation runs. We show the hyperparameters $\boldsymbol{\theta} = [\sigma_f^2, l, \sigma_n^2]^T$ in logarithmic scale.

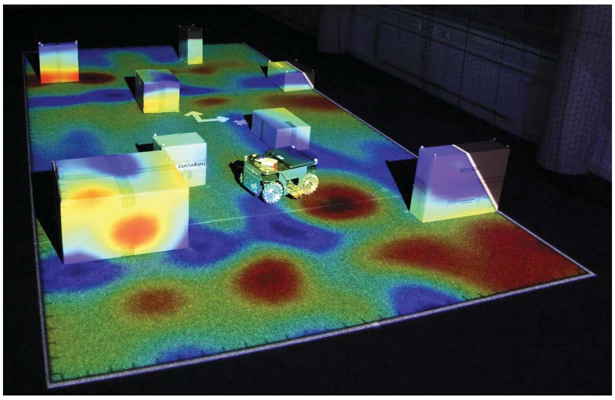


Figure 5.9: Ground-based robot exploring the magnetic field intensity within an indoor environment populated with obstacles. The projection corresponds to the ground truth data that we previously measured before performing the experiment.

Prior to the experiment we scanned the magnetic field intensity with a resolution of 10 cm. Given the dimensions of the environment this corresponds to a grid of 1800 cells. The magnetic field intensity ranges between 5 and 84 μT. This data is then used as a ground truth to test the performance of our algorithm. Considering these measurements as ground truth is a realistic assumption since the sensor can be considered as almost noise-free according to its specifications.

For the algorithm parameters we use the same ones as described in Sec. 5.6.1. We run the algorithm in a central computer and then we send the corresponding waypoints to the robot using the Robot Operating System (ROS) (Quigley et al., 2009) with a WiFi connection. The robot is equipped with a Raspberry Pi that runs the robot's controller to guide the robot to the desired position.

5.7.2 Experimental Results

We benchmark the performance of Alg. 5.1 with a greedy and a random trajectory. We show in Figure 5.11 screenshots of the algorithm execution for three instants of time. Specifically, we show the estimation of the physical process, the entropy of the process model, and the tree that was produced by the robot to plan a path toward the next station. The robot is represented with a circle and the path planned by our RRT*-based informative planner is indicated with a thick red line. We can also observe that the estimation of the robot at 946 seconds, which corresponds to the end of the experiment, is really close to the ground truth data shown in Figure 5.9. Let us also remark that the blue areas in the figures that represent the model's entropy correspond to the already ex-

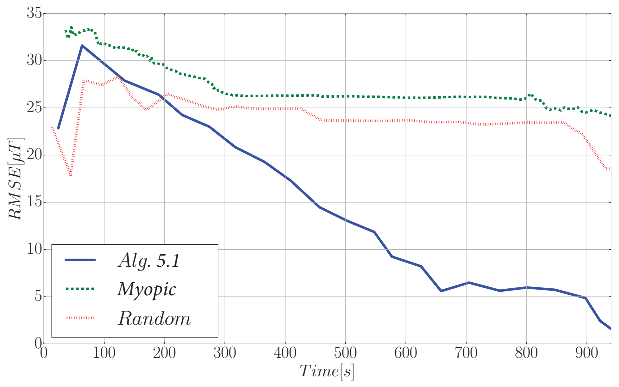


Figure 5.10: Evolution of the RMSE during a 940 seconds exploration task that was carried out with a ground-based robot.

plored positions. Therefore, the entropy resulting at the end of the execution shows that the robot covered all areas except the ones that are occupied by the obstacles.

Finally, Figure 5.10 illustrates the evolution of the RMSE, calculated with (4.2), for the three compared strategies. We remark that the RMSE is computed over all $\mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}}$. As we concluded in simulations, here we verify that our algorithm greatly outperforms the myopic and random trajectories. The proposed strategy is able to decrease the RMSE to approx. 1.24 μT, which represents a nine-fold improvement respect to the other approaches.

5.8 Summary and Outlook

In this chapter we described an extension of the algorithm proposed in Chapter 4 that allows a robot to explore an unknown process that takes place in an environment populated with obstacles. The proposed algorithm employs a sampling-based method (RRT/RRT*) to plan a robot's trajectory in a continuous space. In contrast to Chapter 4, where we employ a myopic approach, the algorithm proposed in this chapter allows us to incorporate any robot whose motion can be planned with an RRT-like algorithm. Moreover, we proposed the introduction of an utility function into the standard RRT/RRT* algorithms. This utility function trades-off informativeness, measured as a mean entropy, and path cost. Performance of the algorithm was benchmarked both in simulations and experiments against several state-of-the-art algorithms. Results illustrate that the proposed approach represents a nine-fold improvement respect to state-of-the-art methods.

However there are a few aspects that will be considered in this thesis to

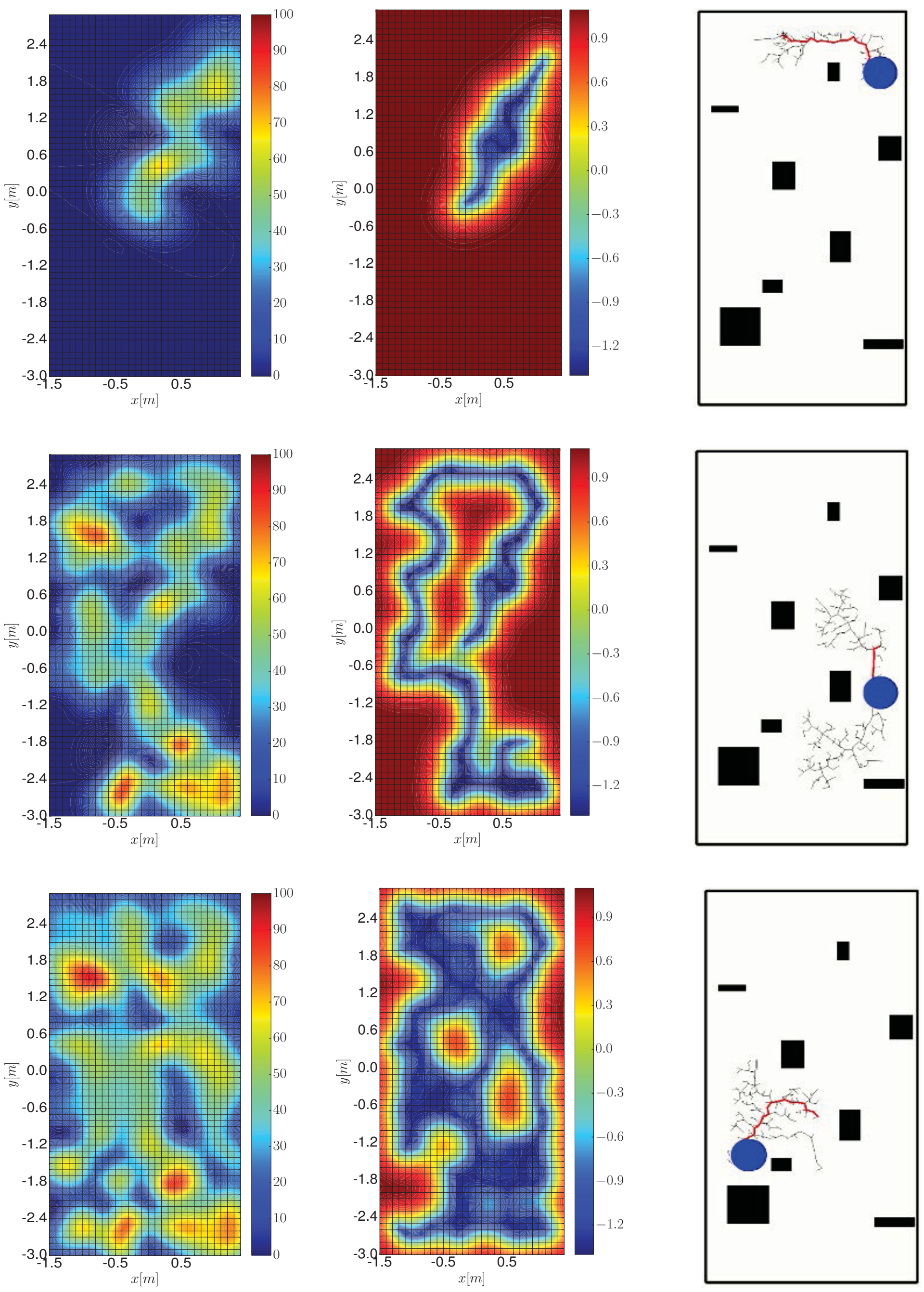


Figure 5.11: Screenshots showing the algorithm execution as we run the experiment. The three rows correspond to three instants of times: 133, 502 and 946 seconds. From left to right, the columns are the estimation of the process (measured in μT), the entropy, and the planned path using Alg. 5.3.

improve the algorithm performance. These are the following:

1. We are considering only one robot. An extension of this algorithm to multiple robots will be proposed in Chapter 7.

2. The proposed algorithm requires an intensive computation of the information metric. Therefore, we motivated the use of the mean entropy against MI. However, mean entropy still does not consider the cross-correlation between different points. We overcome this issue in Chapter 7 by introducing a clustering method that reduces the dimensionality of the RRT. This way, we do not require a so intensive computation and more complex information functions, like MI, can be considered as information metric.

In next chapter we consider point 1, and propose a first approach to gather information with multiple robots. Specifically, we propose an extension of the algorithm described in Chapter 4 that allows multiple robots to communicate with each other and cooperate to gather information more efficiently.

Part III

Multi-Robot Exploration

Chapter 6

Myopic Multi-Robot Exploration

We proposed in Part II of this thesis two algorithms to autonomously gather information with a single robot. We showed both in simulations and experiments that our proposed algorithms outperform state-of-the-art solutions. Nevertheless, we strongly believe that the task of information gathering can benefit from the use of a team of multiple robots, instead of a single robot. A multi-robot system offers two main advantages for exploration tasks over a single-robot system (Burgard et al., 2000):

- *robustness*, as a multi-robot system does not suffer from the "single point of failure" problem. That is, if designed properly, a multi-robot system is able to perform an assigned task even if an individual robot fails.
- *efficiency*, as an increase in the number of robots translates into an increase of resources available, such as computational resources or number of sensors. For example, a multi-robot system could distribute the information processing among all robots in the system, which results in a more efficient information processing. Moreover, an increase in the number of robots translates into an increase in the number of sensors available. Therefore, in a fixed time a multi-robot system is able to gather more information than a single-robot system.

Thus, a multi-robot system offers two essential advantages for information gathering tasks: robustness and efficiency. However, a multi-robot system presents additional challenges, which we must deal with, that are related to the inclusion of a higher number of robots. In particular, there are two aspects that are crucial in a multi-robot system: (i) inter-robot cooperation, and (ii) inter-robot coordination. Inter-robot cooperation refers to how a team of robots should jointly perform a task, like e.g. maximize information gathering. Inter-robot coordination is associated to how robots, which share common resources, should fulfill team specific constraints, like e.g. collision avoidance or communication constraints.

To achieve inter-robot cooperation and coordination multiple approaches have been proposed in the literature. One example of such approaches are communication-less methods, where robots may reason without communication by simply observing others robots states (Balch and Parker, 2002). In contrast, and based on the fact that our robots are equipped with communication devices, here we employ communication-based methods to allow robots to "talk to each other".

We have introduced the main advantages and challenges of a multi-robot system. In this chapter, we propose a GPs-based algorithm that allows multiple robots to gather information of an unknown physical process. The algorithm introduced in this chapter is an extension of Alg. 4.1 to the multi-robot case. We validate the proposed algorithm in simulations, and in an experiment in which two quadcopters, equipped with an ultrasound sensor that faces down, explore an unknown terrain profile. An efficient GPs-based multi-robot cooperation algorithm that runs online, together with the application of mapping a terrain profile, are the two main contributions of this chapter.

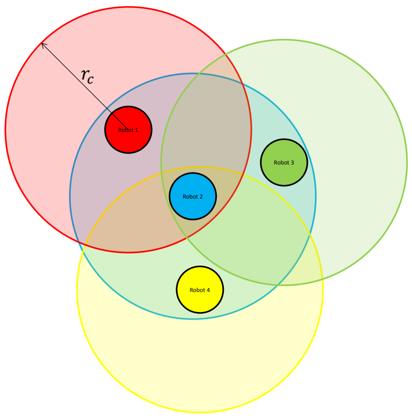
This chapter is organized as follows: in Section 6.1, we introduce notation required to describe a multi-robot system, as well as the inter-robot communication model that we will employ in the remainder of this thesis. Then we describe the proposed algorithm in detail in Section 6.2. This is followed by an evaluation of the algorithm in simulations (Section 6.3), and experiments (Section 6.4). We conclude this chapter with a summary and outlook in Section 6.5.

6.1 Inter-Robot Communication

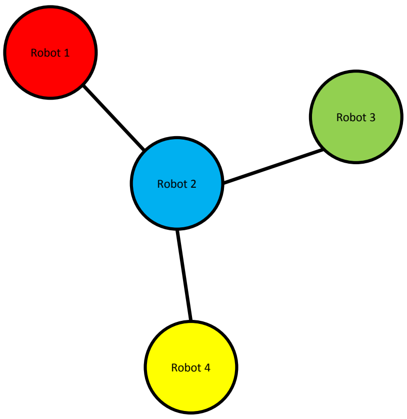
We introduce in this section the basics of inter-robot communication that we will need for the upcoming chapters. First, we describe in Section 6.1.1 a model that characterizes the communication between any pair of robots. Communication between robots allows them to conform a network. A formal definition of a robotic network, together with an overview of network topologies, is given in Section 6.1.2.

6.1.1 Communication Model

In the literature, we can find multiple inter-robot communication models that range over a huge spectrum in terms of complexity, and representability of the actual communication channel (Zavlanos et al., 2011; Tekdas et al., 2012; Hollinger et al., 2012; Wang et al., 2017). For simplicity, as research in inter-robot communication is out of the scope of the thesis, we rely on a simple communication model. A simple communication model that is widely used in the multi-robot



(a) Communication disk.



(b) Communication network.

Figure 6.1: Example that illustrates the concept of a disk communication model.

cooperation literature is a disk communication model (Dixon and Frew, 2009; Sabattini et al., 2013).

Definition 6.1. ***(Disk communication model).*** *A disk communication model specifies that there is a communication link between two agents i and j, located at positions $\mathbf{x}_{r_i}$, $\mathbf{x}_{r_j}$, respectively, iff,*

$$||\mathbf{x}_{r_i} - \mathbf{x}_{r_j}||_2 \leq r_c, \tag{6.1}$$

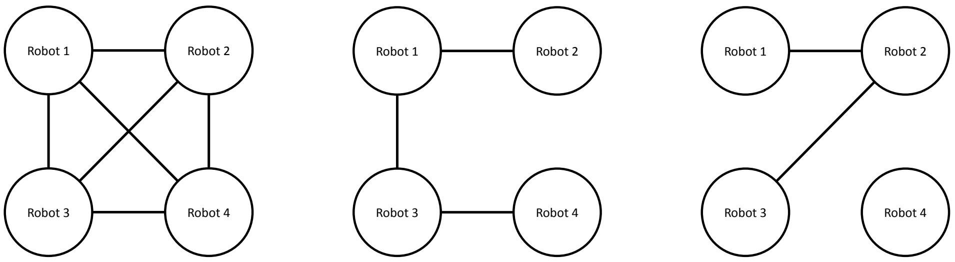
where r_c defines the communication radius.

We show in Figure 6.1 an example that illustrates the concept of a disk communication model. On the left hand side, we depict four robots together with their corresponding communication disk. On the right hand side, we represent the resulting communication network where two robots are linked iff they can communicate with each other.

The example from Fig. 6.1 allows us to understand the concept of a disk communication model. Also, in this work we introduce two additional assumptions in our communication model that are widely used in the multi-robot cooperation literature. That is, we assume that communication between a pair of robots is instantaneous and error-free (Dixon and Frew, 2009; Sabattini et al., 2013).

6.1.2 Communication Network Topology

Let us denote a system composed of N robots, where each robot i is located at position $\mathbf{x}_{r_i} \in \mathcal{X}_{free}$, with $i \in [1, 2, ..., N]$. We assume that robots are equipped



(a) Fully connected. (b) Connected. (c) Disconnected.

Figure 6.2: Communication graphs of the network topologies, classified according to its connectivity properties.

with a communication system that allows them to communicate with each other, conforming a communication network.

A communication network is typically represented as a graph, where nodes correspond to robots' positions, and edges link two nodes iff there exist a physical communication link between the two robots associated to the nodes. We term the graph, which characterizes the communication network, the communication graph.

Definition 6.2. ***(Communication graph).*** *A communication graph $\mathcal{G}_c(\mathcal{V}_t, \mathcal{E}_t)$ at time t, with vertices $\mathcal{V}_t$ and edges $\mathcal{E}_t$, of N robots, located at positions $\mathbf{x}_{r_i}(t)$, is given by:*

$$\mathcal{V}_t = \{\mathbf{x}_{r_1}(t), \mathbf{x}_{r_2}(t), ..., \mathbf{x}_{r_N}(t)\}, \tag{6.2}$$

$$\mathcal{E}_t = \left\{(\mathbf{x}_{r_i}(t), \mathbf{x}_{r_j}(t))\right\}_{i,j\in[1:N];\ i\neq j;\ i,j\ \textit{are connected},} \tag{6.3}$$

where connectivity implies that two robots i, j can communicate with each other, with inter-robot communication given by Def. 6.1.

The communication graph allows us, for example, to check whether two robots are connected, or whether a robot is disconnected from the rest of the network. Network connectivity is a topology property that is crucial for multi-robot cooperation/coordination algorithms, as these require a certain degree of connectivity to allow robots to cooperate with each other. In this thesis we classify connectivity in terms of the communication graph that defines the network topology (Godsil and Royle, 2013). In this sense we can identify three different topologies:

- A *fully connected* topology implies that the communication graph is fully connected (Figure 6.2a). That is, there exists an edge between any two nodes in the communication graph.

- A *connected* topology implies that the communication graph is connected (Figure 6.2b). That is, there exists a direct or indirect path between any two nodes in the communication graph.

- A *disconnected* topology is a topology that is not connected (Figure 6.2c).

In this chapter we assume $r_c = \infty$, which specifies a fully connected network. Then we consider in next chapters a more realistic system that is subject to communication constraints, yielding any of the three afore-described topologies.

6.2 Multi-Robot Exploration with Online-Learning of GPs

We introduce in this chapter an algorithm that extends Alg. 4.1 to a multi-robot system composed by N robots. As Alg. 4.1, the algorithm proposed in this chapter also exploits GPs (Section 2.1) to model the physical process under study, relies on a discrete graph-based myopic decision maker (see Section 2.2.1), and employs entropy (Section 2.3) as information metric. In contrast to Alg. 4.1, here we incorporate an online learning of the GPs hyperparameters, as we did in Alg. 5.1. Let us recall that the online learning of the hyperparameters allows robots to explore an *a priori* unknown process, as robots do not require prior information about the process.

The proposed algorithm is able to coordinate multiple robots while avoiding inter-robot collisions. In particular, we propose a semi-decentralized realization of the algorithm (see Sec. 3.4), such that each robot takes its own decisions based on the currently available information. A semi-decentralized realization of the algorithm is crucial both in terms of robustness and efficiency as we pointed out in this chapter's introduction. Let us clarify that we consider our approach as semi-decentralized, and not decentralized, because robots have an infinite communication range ($r_c = \infty$), and employ a broadcast mechanism to exchange information. In contrast, we understand as decentralized a system that relies solely on local inter-robot communication, according to the definitions in (Grime and Durrant-Whyte, 1994; Capitan et al., 2011). In Chapter 7 we extend the concept proposed in this chapter, and propose a system that performs a decentralized coordination.

A block diagram of the whole scheme is shown in Figure 6.3. We present in Algorithm 6.1 a detailed pseudo-code. Both the block diagram and algorithm correspond to a single robot. That is, each individual robot runs Algorithm 6.1 independently and communicates with the other robots, through module "Broadcast & Receive Info" (see Figure 6.3), until a stopping criterion is fulfilled.

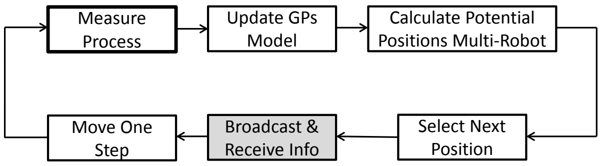


Figure 6.3: Algorithm block diagram. The shadowed block represents a module that requires communication between robots.

Algorithm 6.1. `MyopicMultiRobotExploration`($\mathbf{x}_{r_i}, \mathcal{L}_{\mathcal{X}_{free}}, StopAlgorithm$)

```
1: $\mathbf{z} \leftarrow NULL; \mathbf{X} \leftarrow NULL; \mathbf{x}_{next} \leftarrow NULL;$
2:
3: while ! $StopAlgorithm$ do
4:     $\mathbf{z}_{othLast}, \mathbf{X}_{othLast}, \mathbf{X}_{othNext} \leftarrow$ ReceiveInfo;        ▷ from all other robots
5:     $z \leftarrow$ Measure($\mathbf{x}_{r_i}$);
6:     $\mathbf{z} \leftarrow [\mathbf{z}; \mathbf{z}_{othLast}; z]$;  $\mathbf{X} \leftarrow [\mathbf{X}; \mathbf{X}_{othLast}; \mathbf{x}^T]$;
7:     $\boldsymbol{\theta}_* \leftarrow$ LearnHyp($\mathbf{z}, \mathbf{X}$);        ▷ with (2.5)
8:     $\mathbf{X}_* \leftarrow$ CalcNextPotentialPositionsMR($\mathcal{L}_{\mathcal{X}_{free}}, \mathbf{X}_{othLast}, \mathbf{X}_{othNext}, \mathbf{x}_{r_i}$);
9:     ▷ Calculate maximum entropy position
10:    $h_{max} \leftarrow -\infty$;
11:    for $\mathbf{x}_*^{[j]} \in \mathbf{X}_*$ do
12:        $\mu_*, \sigma_*^2 \leftarrow$ PredictGP($\mathbf{z}, \mathbf{X}, \mathbf{x}_*^{[j]}, \boldsymbol{\theta}_*$);        ▷ with (2.3)
13:        $h_* \leftarrow$ CalculateEntropy($\sigma_*^2$);        ▷ with (2.20)
14:        if $h_* > h_{max}$ then
15:            $h_{max} \leftarrow h_*$;  $\mathbf{x}_{next} \leftarrow \mathbf{x}_*^{[j]}$;
16:    BroadcastInfo($z, \mathbf{x}_{r_i}, \mathbf{x}_{next}$);        ▷ to all other robots
17:    $\mathbf{x}_{r_i} \leftarrow$ MoveTo($\mathbf{x}_{next}$);
18: $\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_* \leftarrow$ ReconstructProcess($\mathbf{z}, \mathbf{X}, \mathcal{V}_{\mathcal{X}_{free}}$);
```

Algorithm 6.1. Algorithm 6.1 works as follows: in a first step, each robot receives the information broadcasted by the other robots (line 4). It consists of the last measurement taken by each of the other robots, $\mathbf{z}_{othLast}$, as well as the positions where those measurements were taken, $\mathbf{X}_{othLast}$, and the next positions where the robots are heading to, $\mathbf{X}_{othNext}$. This information is sufficient to achieve inter-robot coordination. On the one hand, the knowledge about $\mathbf{X}_{othLast}$ and $\mathbf{X}_{othNext}$ allows us to implement a collision avoidance mechanism. On the other hand, $\mathbf{z}_{othLast}$ and $\mathbf{X}_{othLast}$ act as an indirect mechanism to coordinate the exploration efforts. Since all robots share the same data model, each of them can reproduce what the other robots' information look like. This leads to an implicit cooperation that avoids, for example, that two robots measure at the same position if it is not strictly necessary. It is important to notice that the proposed inter-robot coordination mechanism is valid as along as robots do not

execute Alg. 6.1 perfectly synchronously, which is an assumption that in practice always holds.

In a second step, the robot takes a measurement z (line 5), and incorporates z to its own vector of measurements $\mathbf{z}$ (line 6). Then it estimates $\boldsymbol{\theta}_*$ with (2.5) to update the GPs model given $\mathbf{z}, \mathbf{X}$ (line 7). The update of the GPs model is essential to better model the process we aim to explore, and, therefore, to better predict the process entropy that will guide the robots' exploration. Next the robot computes the position where it will move next. The computation of the next position is carried out in lines 8-15, and is similar to the process described in Alg. 4.1. However, it differs on a fundamental aspect. Here we consider a system composed by multiple robots that share a common environment. Therefore, inter-robot collisions must be prevented. This is done in line 8 with function `CalcNextPotentialPositionsMR`, which calculates a set of next potential positions $\mathbf{X}_*$. Here, in contrast to Alg. 4.1, we incorporate an additional constraint to calculate $\mathbf{X}_*$. That is, we forbid robots to be closer than a safety distance r_s to each other. Given these constraints, the robot calculates $\mathbf{X}_*$ as follows:

$$\mathbf{X}_* = \left\{ \mathbf{x} \in \mathcal{V}_{\mathcal{X}_{free}} \;\middle|\; \begin{array}{l} \mathbf{x} \in \mathcal{N}_{\mathcal{L}_{\mathcal{X}_{free}}}(\mathbf{x}_{r_i}), \\ \min(||\mathbf{x} - \mathbf{X}_{othLast}||_2) > r_s, \\ \min(||\mathbf{x} - \mathbf{X}_{othNext}||_2) > r_s, \end{array} \right\} \tag{6.4}$$

where $\min(||\mathbf{x} - \mathbf{X}||_2)$ is the minimum euclidean distance between $\mathbf{x}$ and any of the rows in $\mathbf{X}$.

Once a robot calculates its next position $\mathbf{x}_{next}$, it broadcasts z and $\mathbf{x}_{next}$ (line 16). Robots continue executing Alg. 6.1 loop until they reach their stopping criteria. Then each individual robot is able to reconstruct the physical process (line 18) as we described in Alg. 4.1.

6.3 Simulations and Discussion of Results

First, we evaluate Alg. 6.1 in simulations. Specifically, we explore a magnetic field intensity, and employ the setup described in Sec. 4.3. In this chapter, we introduced an extension of Alg. 4.1 that differs in three aspects: (i) online learning of GPs hyperparameters, (ii) cooperation between multiple robots for gathering information, and (iii) inter-robot collision avoidance. We evaluated first aspect in Chapter 5. Moreover, we will extend the evaluation in an experiment in Sec. 6.4. Therefore, we do not consider the first aspect in the simulations evaluation. Also, we do not evaluate the last aspect in this section as we demonstrate in Sec. 6.4 that robots do not collide while using Alg. 6.1. Therefore, in this section we focus on the evaluation of the second aspect: multi-robot cooperation for information gathering.

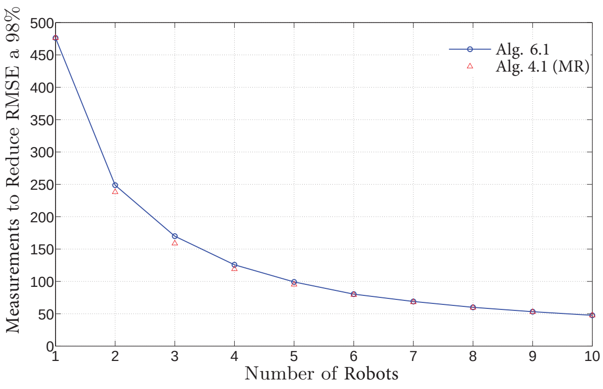


Figure 6.4: Performance evaluation of Alg. 6.1 as we increase the number of robots in the system.

Multi-robot cooperation for information gathering. We test Alg. 6.1 performance as we increase the number of robots in the system from one to ten. As performance metric, we choose the number of total measurements per robot, which robots need to take in order to reduce the initial RMSE a 98%. We clarify that Alg. 6.1 with one robot is identical to Alg. 4.1. Also we test an strategy where we divide the environment in a number of sectors equal to the number of robots available. Then we assign a robot to a sector, which each robot will explore with Alg. 4.1. This, strategy we term it Alg. 4.1 (MR). We carried out each of the simulations 100 times, and results correspond to the average calculated from all simulations. For Alg. 4.1 (MR) each robot starts at a random position within its assigned sector, while for Alg. 6.1 robots start at a random position in the environment.

In Figure 6.4 we show simulation results for both Alg. 6.1 and Alg. 4.1 (MR). First fact is that performance of Alg. 6.1 increases as we increase the number of robots, which demonstrates the effectiveness of the proposed multi-robot cooperation. Also we can observe that performance of Alg. 6.1 and Alg. 4.1 (MR) are almost identical. This allows us to conclude that Alg. 6.1 is able to cover the complete environment without the need of any sectorization. Notice that in a real application, the environment is typically unknown beforehand. Here Alg. 4.1 (MR) would not be applicable. In constrast, with Alg. 6.1 we obtain the same performance as if we knew the environment before starting the exploration task.

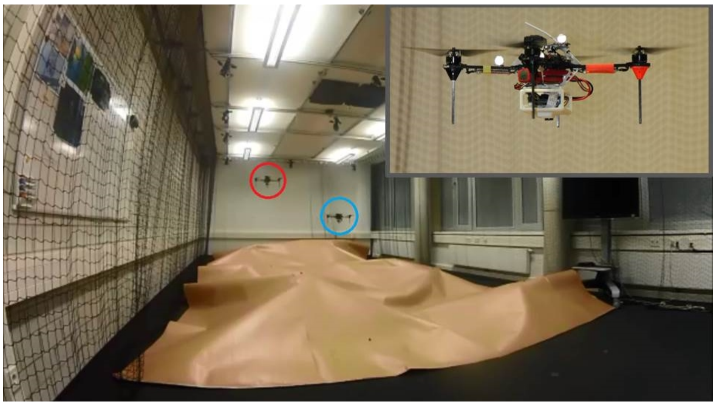


Figure 6.5: Two quadcopters explore an unknown terrain profile with Alg. 6.1.

6.4 Experiments and Discussion of Results

6.4.1 Experimental Setup

We validate Alg. 6.1 in an experiment [1] in which two quadcopters explore an *a priori* unknown terrain profile (see Figure 6.5) with a lateral resolution of 20 cm. The built terrain profile measures approx. 60 cm from top to bottom, and has a size of 8×3 m^2. Each of the quadcopters flies at a different constant height of 1 m and 1.5 m above the floor to avoid the risk of collisions, although they are not aware of it.

Ultrasound sensor. Quadcopters calculate the profile's height as a difference between the robot's actual height, which is provided by a Vicon system and can be assumed to be noise-free (see Sec. 4.3), and the range between the quadcopter and the height profile. To compute this range we employ an ultrasound sensor. Specifically, we mount a commercial ultrasound sensor from MaxBotix that faces down to measure the range to the floor. The sensor has a nominal range of approx. 7.5 m with opening angles of 45° in the near field and a cylindrical measurement profile for ranges greater than 1.5 m. An ultrasound sensor works as follows: it sends an impulse and waits for the echo to calculate the distance to the closest object within the sensor's footprint (see Figure 6.6). It could happen that reflections in the environment and oscillations of the quadcopter's pose could lead to missing measurements (the echo does not return to

[1] A video that shows the experiment execution can be found in: `https://vimeo.com/253576445`; `https://rebrand.ly/myopi389d`.

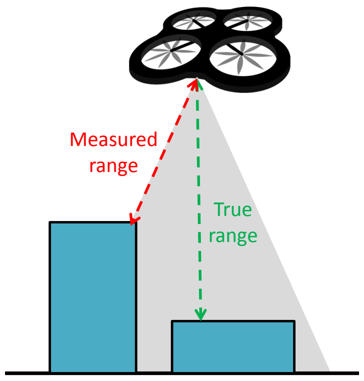


Figure 6.6: Ultrasound sensor measurement. Blue rectangles correspond to sample boxes, and the grey cone represents the sensor's footprint.

the ultrasound sensor) or not-plausible measurements (the impulse gets affected by multiple reflections and the resulting measurement is inconsistent considering the environment structure). Therefore, multiple measurements need to be taken to mitigate such effects. We solve this problem by averaging over 10 valid measurements taken at the position of interest; i.e. we discard those measurements that are not plausible, such as negative heights and heights that are above the quadcopter actual z position. In case we obtain no valid measurements during an interval of 30 seconds, we average over the last 10 valid measurements, which are stored in a buffer.

The algorithm uses ROS (Quigley et al., 2009) and WiFi for the communication between quadcopters. It is important to notice that, although the algorithm itself is decentralized, only parts of it run on the Raspberry Pi mounted on the quadcopters due to the computational limitations of these devices. Instead, the GPs regression and learning of hyperparameters run in one separate central computer, but in a decentralized manner. This decentralization is possible because of the nature of ROS, where nodes run in different threads. We employ the pyGPs library to perform the GPs-related calculations.

Benchmark strategies. We benchmark the following five exploration strategies:

1. meander-like trajectory with a single robot (see Sec. 4.3) with pre-set hyperparameters,

2. random trajectory with a single robot (see Sec. 4.3) and online learning of hyperparameters,

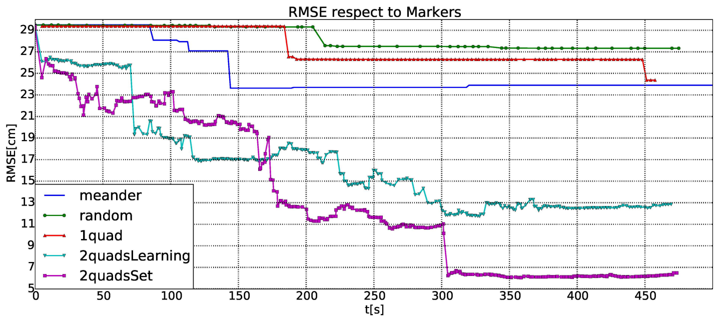


Figure 6.7: RMSE with respect to the ground truth for five different algorithms during a 8 minutes exploration experiment.

3. an extension of Alg. 4.1 (single-robot) that includes an online learning of hyperparameters,
4. Alg. 6.1, which is a multi-robot exploration algorithm with online learning of hyperparameters, and
5. Alg. 6.1 with pre-set hyperparameters.

Let us clarify that both strategies 1 and 2 employ GPs regression to predict the physical process values at positions that were not yet measured. Strategies 1 and 5 carry out the regression using the optimal hyperparameters learned from the data collected with the meander trajectory in a previous exploration run.

Ground truth. For the evaluation, the terrain profile ground truth corresponds to 13 markers placed randomly distributed along the terrain profile. The markers positions were recorded using the Vicon tracking system.

6.4.2 Experimental Results

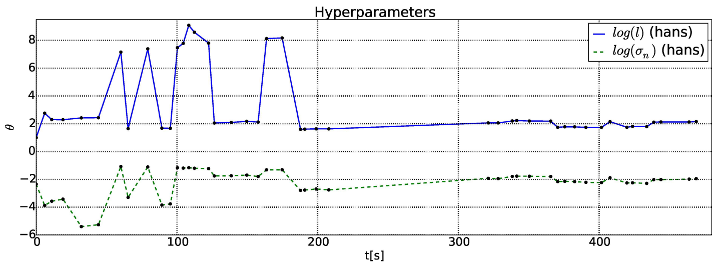
First, we analyze the RMSE between estimate and ground truth after running Alg. 6.1 during the lifetime of the quadcopter's battery, which is approx. 8 minutes in our case. Exchanging the batteries of the quadcopter is a highly time-consuming process. Therefore, our goal is developing exploration algorithms that are able to reconstruct the original process during this battery's life-time. This is done by both developing intelligent exploration strategies and increasing the number of robots in the swarm. We present evaluation results in Figure 6.7.

Ultrasound sensing error. First thing we observe is that the minimum error we achieve with any of the strategies is approx. 7 cm. This error may seem relatively large. However, it is equal to the best performance we can obtain with the ultrasound sensor in the explored environment. To verify that, we measured all positions in the environment (we needed 3 batteries and 53 minutes) with the meander trajectory and calculated the RMSE as benchmark. The resulting RMSE was 7.64 *cm*, although we measured the complete physical process. This error is considerably large and is due to the sensor's footprint and sensor's characteristics, and to the oscillations of the quadcopter while flying. The sensor takes the minimum range – maximum height – within its footprint. Therefore it is not able to distinguish between different heights that lay within the sensor's footprint (see Figure 6.6). This fact induces the error. Since that is the best performance we can get with the sensor without any postprocessing of the measurements, we assume this error as the best possible solution we can obtain.

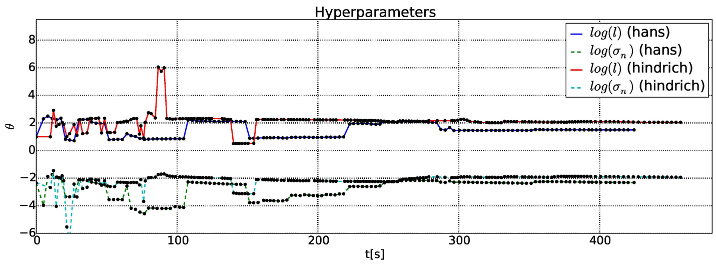
Comparison with predefined trajectories for a single robot. Next fact we notice is that, in contrast to the results obtained for the exploration of a magnetic field intensity (Figure 4.10), the error with strategies 2 (random) and 3 (single-robot) is larger than the one obtained with strategy 1 (meander). The difference respect to Fig. 4.10 is that here strategies 2 and 3 learn the model's hyperparameters online, while in Fig. 4.10 they were pre-set to the optimal values. In strategies 2 and 3 the amount of measurements collected with a single quadcopter in the initial phase of the exploration run is not enough to learn the process model fast, which incurs in an initial loss of performance that is dragged during the rest of the exploration.

However, attending at results from Figure 6.7, performance of strategy 2 (random) is comparable to the performance of strategy 3 (single-robot). Figure 6.8a demonstrates that this performance of the random trajectory is a mere coincidence, since this trajectory does not guarantee the convergence of the hyperparameters learning – see interval between 210 and 320 seconds, where the quadcopter could not converge to any hyperparameter value. In contrast, as we show in Figure 6.8b, Alg. 6.1 converges fast to the optimal hyperparameters given the available measurements because of the myopic nature of the algorithm.

Multi-robot exploration. Since one quadcopter is not sufficient to explore this physical process, we use Alg. 6.1 (strategy 4) with two quadcopters. For this case we observe in Figure 6.7 that the error has been reduced by one half respect to strategies 1-3, which proves the correct coordination between robots. However, this error of 12.84 *cm* is still larger than our benchmark error of 7.64 *cm*. Furthermore, the exploration time was 8 minutes, which is approx. seven times smaller than the time needed to achieve our benchmark error. In order to understand where this remaining error lies, we run Alg. 6.1 with two quadcopters



(a) Hyperparameters learned while following a random trajectory.



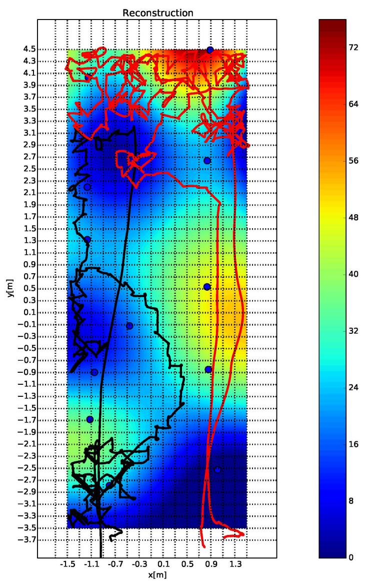
(b) Hyperparameters learned by two quads while exploring with Alg. 6.1.

Figure 6.8: Hyperparameters learned during the exploration run. The black points represent the actual values. The color lines are the result of a linear interpolation of those points.

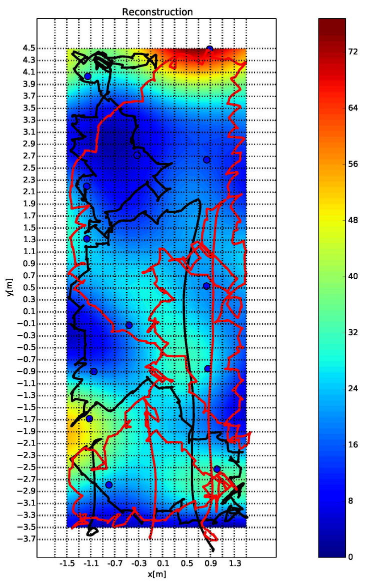
but with the optimal hyperparameters (strategy 5).

Exploration with optimal hyperparameters. Here, with strategy 5, we notice that the error is approx. equal as our best possible solution while the exploration time is approx. seven times smaller. Figure 6.9 shows the trajectories of the two robots for strategies 4 and 5. We can see that for the online learning case (strategy 4), the robots fly around in a local area at the starting phase. This response is due to the fact that robots did not learn a proper model and cannot decide correctly where to measure next. We observe this behavior as well in Figure 6.8b. In the first 220 seconds, the hyperparameters learning does not converge and this causes an inferior performance compared to the set hyperparameters trajectory.

Last remarks. We can conclude according to the simulations and experimental results that Alg. 6.1 is able to achieve the correct coordination between the robots. However, experimental results showed that, as part of the future work, we must improve the learning phase to get closer to the performance obtained while using the optimal hyperparameters. For example, active learning approaches



(a) Hyperparameters learning.



(b) Hyperparameters set.

Figure 6.9: Reconstruction of the terrain profile (see Figure 6.5) after running Alg. 6.1 for two different setups: online learning (left) and pre-set hyperparameters (right). On top we show the trajectories of the two quadcopters. Blue dots correspond to the markers' positions that serve us as ground truth. The color scale represents the estimated height (measured in cm), being red the highest value.

could be considered to better learn the model's hyperparameters (Krause and Guestrin, 2007b).

6.5 Summary and Outlook

In this chapter we described a first approach that lets multiple mobile robots autonomously explore an unknown physical process. The performance of the algorithm has been shown in simulations, and in an experiment where two quadcopters, equipped with an ultrasound sensor facing down, explore a terrain profile. Results illustrate that the proposed approach is able to explore a process more efficiently than several benchmark methods. Moreover, results show how robots correctly learned the GPs hyperparameters online while performing the exploration task.

However there are a few aspects that will be considered in this thesis to improve the algorithm performance. These are the following:

1. The robots plan in a myopic fashion, which is in general suboptimal as pointed out in Chapter 5. A sampling-based path planning approach (Section 2.2.2) will be considered in Chapter 7 to improve the algorithm performance.

2. The use of entropy as information metric leads to a superior performance compared to a random and a meander trajectory. However, more complex metrics that yield a better performance have been proposed in the literature, like e.g. MI. In Chapter 7 we argue about the use of MI in exploration algorithms.

3. We employed a holonomic robot to carry out the exploration task. In Chapter 7 we expand the algorithm to handle a larger class of robots.

4. In this chapter, we assumed that the communication graph that defines the robots network is fully connected. This assumption allowed us to analyze the cooperation capabilities of the proposed algorithm, regardless the communication aspect. However it is an unrealistic assumption, as all communication devices have a limited communication range. In Chapter 7 we propose an algorithm that does not require a fully connected network, and relies uniquely in local communication to coordinate robots during the information gathering task.

In next chapter (Chapter 7) we consider the four aforementioned points, and propose a more efficient exploration algorithm. This algorithm can handle a larger class of robots, and is able to deal with more complex constraints, such as collision avoidance and inter-robot communication constraints.

Chapter 7

Sampling-Based Multi-Robot Constrained Exploration

We proposed in Chapter 6 Alg. 6.1, which allows multiple robots to autonomously gather information of an unknown physical process. Algorithm 6.1 has two major constraints: (i) it requires a discrete graph-based representation of the environment, and (ii) it requires the communication network, which robots conform, to be fully connected. On the one hand, the former limits the generality of the algorithm, as we pointed out in Chapter 5, where we first introduced the concept of sampling-based exploration. On the other hand, the latter constraint is unrealistic in a real-world scenario, as robots have a limited communication range. This implies that, in practice, robots cannot conform a fully connected network if they are far apart.

In this chapter we introduce an algorithm that overcomes the two aforementioned issues. To this end, we build on previous concepts and employ (i) GPs to model the process we aim to explore, (ii) sampling-based algorithms (RRT) to plan robots actions, and (iii) a Distributed Constraint Optimization (DCOP) algorithm – max-sum (Farinelli et al., 2008) – to achieve multi-robot cooperation and meet problem-specific constraints. In particular, in this chapter we consider the following constraints: collision avoidance constraints to avoid inter-robot collisions, and network connectivity constraints to derive an algorithm based on local communication between robots. Let us remark that network connectivity allows communication between robots and, therefore, permits multi-robot cooperation. Moreover, it is a requirement in a large class of information gathering applications as indicated in (Hollinger and Singh, 2010; Gan et al., 2014).

This chapter extends concepts presented in previous chapters in order to fulfill this thesis objectives, as stated in Section 1.3. To the best of our knowledge, we introduce here the first non-myopic sampling-based multi-robot algorithm for information gathering.

We organize this chapter as follows: first, we state the problem formally in Section 7.1. Then we present an overview of our algorithm in Section 7.2. Next we introduce in Section 7.3 max-sum. Our algorithm relies on an information metric to decide robots' action. Therefore, we include in Section 7.4 a discussion about the suitability of several information metrics for our specific problem, and introduce a definition of MI that meets our problem requirements. This is followed in Sections 7.5 and 7.6 by a detailed explanation of the different subsystems that compose our system, and an assessment of the algorithm's computational complexity, respectively. Then we test the algorithm in simulations in Section 7.7, and verify in Section 7.8 its performance with a field experiment in which three quadcopters explore an unknown simulated wind field. We finalize with a summary and outlook of the chapter in Section 7.9.

7.1 Problem Statement

We wish to explore a physical process with N cooperative robots. Let us assume that the physical process and robots are subject to the assumptions introduced in Sec. 1.3. In addition, we consider the following constraints:

1. *Inter-robot collision avoidance*: two robots collide if they are separated less than a distance r_s.

2. *Network connectivity*: the network of robots requires a periodic connectivity, with a maximum disconnection time of $k_c dt$ seconds.

The problem described in Sec. 1.3 corresponds to an infinite horizon information gathering task. A common approach in information gathering is to divide an infinite horizon problem into multiple finite horizon problems that can be solved individually and sequentially. That is, first, robots solve an information gathering problem for the next k_c iterations. Note that we assume here that the horizon has a length k_c equal to the maximum number of iterations we allow the network to be disconnected. The solution to the information gathering problem is a combination of robots' paths $P = \{\mathcal{P}_1, ..., \mathcal{P}_N\}$, with $\mathcal{P}_i \in \mathcal{X}_{free}$, which maximizes a global utility function $U_I(\mathcal{P}, \mathbf{X})$, which depends on P and the gathered measurements $\mathbf{X}$. Once robots find a solution, they follow $\mathcal{P}_i$, and repeat the procedure again.

More formally, this chapter proposes a solution to the following finite horizon

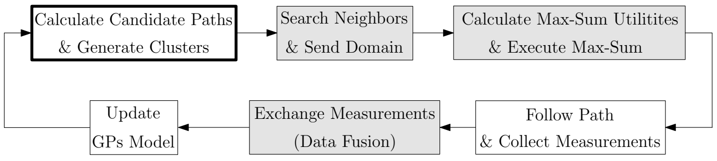


Figure 7.1: Algorithm block diagram. Shadowed blocks represent modules that require communication between robots.

problem:

$$\begin{aligned}
\underset{\mathcal{P}}{\text{maximize}} \quad & U_I(\mathcal{P}, \mathbf{X}) \\
\text{subject to} \quad & \mathcal{P}_i = \{\mathbf{x}_i(t), \mathbf{x}_i(t+dt), ..., \mathbf{x}_i(t+k_c dt)\}, \\
& \mathbf{x}_i(t+(k_t+1)dt) = \mathbf{f}_m(\mathbf{x}_i(t+k_t dt), \mathbf{u}), \\
& ||\mathbf{x}_i(t+k_t dt) - \mathbf{x}_j(t+k_t dt)||_2 \geq r_s \\
& \qquad\qquad \forall k_t = 1, 2, ..., k_c; i \neq j, \\
& \mathcal{G}_c(\mathcal{V}_{t+k_c dt}, \mathcal{E}_{t+k_c dt}) \text{ is connected,}
\end{aligned} \tag{7.1}$$

where $\mathbf{f}_m(\cdot,\cdot)$ describes the robot's motion model (Section 4.1); and graph $\mathcal{G}_c(\mathcal{V}_{t+k_c dt}, \mathcal{E}_{t+k_c dt})$ is the communication graph, according to definition 6.2, that results at iteration $t + k_c dt$ given the robots positions.

7.2 Algorithm Overview

We present in this section the algorithm that we propose to solve (7.1). In Fig. 7.1 we depict a block diagram of the proposed algorithm. In particular, the diagram corresponds to the modules that each single robot executes. Modules are executed in a loop, where each loop iteration solves (7.1).

Our proposed algorithm works as follows: first, robots plan a set of potential actions – paths – that they could follow (Sec. 7.5.1). Specifically, each robot generates an RRT, whose root is the robot's current position. Next, robots cooperate in order to select a path that maximizes $U_I(\cdot)$ subject to constraints from (7.1). Here we define $U_I(\cdot)$ as an information theoretic function. To solve this multi-robot cooperation problem we propose the use of a DCOP algorithm: max-sum.

Max-sum requires that each robot knows its own set of potential actions, as well as its neighbors' set of potential actions. Here we term this: robot's domain. To this end, we include a module that allows robots to find its neighbours, and to send its domain (Sec. 7.5.2).

Once a robot receives its neighbors' domain, it executes max-sum (Sec. 7.5.3). Max-sum is a combinatorial optimization algorithm. That is, in order to solve the optimization problem, each robot must evaluate all combinations of potential paths from the received domains (including its own domain). As we previously mentioned, here we consider a robot's domain as the set of all paths that are contained in the generated RRT. Since RRTs could grow large, the number of total paths could increase as well. This would result in an increase of the complexity of the combinatorial optimization, which could make the optimization computationally intractable. To solve this issue, we propose a procedure in which each robot groups the RRT paths into clusters, which reduces the robots' domain size (Section 7.5.2).

Max-sum outputs a cluster for each individual robot that solves (7.1). Then, each robot selects a path within its cluster. This is realized by evaluating an information-theoretic function (Section 7.5.4).

Next robots follow the selected paths while taking measurements along them. Measurements encode the knowledge robots have about the process of interest. Therefore, they exchange the gathered measurements through the network; i.e. they perform data fusion (Section 7.5.5). Finally, robots update their GPs model (its hyperparameters) with the new measurements in order to improvement the process model (Section 7.5.6).

7.3 Distributed Constraint Optimization: Max-Sum

The core of the system that we present in this chapter is "Calculate Max-Sum Utilities and Execute Max-Sum" (see Fig. 7.1). This module allows robots to cooperate; i.e. to determine the actions the team should take in order to maximize a given utility function $U_I(\cdot)$ subject to constraints from (7.1). To achieve cooperation between multiple robots we propose in this chapter the use of DCOP techniques.

A number of DCOP techniques have been proposed in past years (Leite et al., 2014). We can dichotomize such techniques between complete algorithms, which generate an optimal solution, and approximate algorithms, which generate an approximate solution.

Complete DCOPs. Complete algorithms are well exemplified by ADOPT (Modi et al., 2005) and DPOP (Petcu and Faltings, 2005). Now, while these algorithms represent significant contributions in their own domain, they do not address many of challenges that are present in autonomous robotic systems. In particular, optimality implies that some aspect of these algorithms scales exponentially, which compromises a real time implementation of the system.

In fact, first, before considering max-sum, we employed ADOPT within a multi-robot path planning algorithm. This was motivated by ADOPT optimality guarantees. The proposed algorithm, together with an analysis of ADOPT performance can be found in Appendix B. In Appendix B we conclude that ADOPT is not suited for real time applications that involve a large number of cooperative robots. Therefore, we investigated further DCOP algorithms. Specifically, we focused on an approximate DCOP: max-sum.

Max-sum. Max-sum, in contrast to another approximate algorithms like e.g. (Zhang et al., 2003), makes efficient use of the computational and communication resources. In addition, it offers approximate solutions that are close to optimal for many applications of interest like e.g. exploration, or tracking (Stranders et al., 2010). Next we explain max-sum in detail.

Let us consider a team of N robots, where each robot can control a decision variable d_i that can take values from domain $\mathcal{C}_i = \{\mathcal{C}_i^{[1]}, \mathcal{C}_i^{[2]}, ..., \mathcal{C}_i^{[k_i]}\}$ that denotes k_i sets of potential measurement locations that robot i could visit. We denote the set of variables for which we aim to solve the optimization problem as $\mathcal{D} = \{d_1, d_2, ..., d_N\}$. The goal of the robots is to minimize a global utility function $U(\mathcal{A})$, where $\mathcal{A}$ denotes a possible assignment for the variables. For example, for a team of three robots with identical domain size $k_i = 4$, a possible assignment for the variables could be $\mathcal{A} : \{d_1 : \mathcal{C}_1^{[1]};\ d_2 : \mathcal{C}_2^{[4]};\ d_3 : \mathcal{C}_3^{[1]}\}$.

In max-sum, each robot interacts locally with its neighboring robots such that the utility of an individual robot, $U_i(\bar{d}_i)$, is dependent on $\bar{d}_i$, which denotes d_i and the decision variables of neighboring robots. We remark that max-sum does not assume any structure of $U_i(\cdot)$, and does not require $U_i(\cdot)$ to be known by other robots. Within this setting, we wish to find the optimal assignment of paths $\mathcal{A}^*$ such that social welfare of the whole system (i.e. maximising the sum of the utilities of each individual robot) is maximised:

$$\mathcal{A}^* = \underset{\mathcal{A}}{\operatorname{argmax}} \sum_{i=1}^{N} U_i(\bar{d}_i). \tag{7.2}$$

Max-sum formulates this assignment problem as a *factor graph* (Kschischang et al., 2001). A factor graph is a bi-partite graph where there are two sets of nodes: variables and utilities. Edges in this graph represent the dependencies of utilities on variables. For instance, the factor graph in Fig. 7.2 represents $U(d_1, d_2, d_3)$ if it can be expressed as:

$$U(\mathcal{D}) = U_1(d_1, d_3) + U_2(d_2, d_3) + U_3(d_1, d_2, d_3), \tag{7.3}$$

where $\bar{d}_1 = \{d_1, d_3\}$, $\bar{d}_2 = \{d_2, d_3\}$ and $\bar{d}_3 = \{d_1, d_2, d_3\}$.

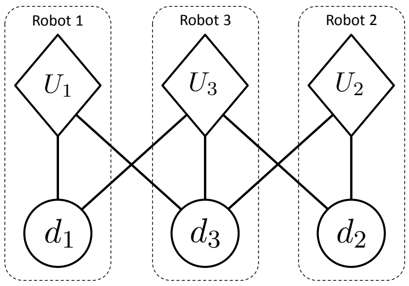


Figure 7.2: A factor graph representing a utility function that can be decomposed as in (7.3).

Message passing on factor graphs. Max-sum is a message passing algorithm on factor graphs. Messages are passed along the edges of the factor graph in order to determine the variable values that maximise $U(\cdot)$. We distinguish between two types of messages:

- Function to variable message. It encodes the maximum value of utility function U_i for each possible value of d_j.
- Variable to function message. It encodes the maximum utility of the neighboring utility nodes for each possible value of d_j.

In particular, the message passed from U_i to d_j, $r_{i\to j}$, is defined as:

$$r_{i\to j}(d_j) = \max_{\bar{d}_i \backslash d_i} \left(U_i\left(\bar{d}_i\right) + \sum_{j' \in adj(i)\backslash j} q_{j'\to i}(d_{j'}) \right), \tag{7.4}$$

where the *adj* operation is the set of adjacent nodes, and $q_{j'\to i}$ is the message going from variable j' to function i, defined as follows:

$$q_{j\to i}(d_j) = \alpha_{ji} + \sum_{i' \in adj(j)\backslash i} r_{i'\to j}(d_j), \tag{7.5}$$

where α_{ji} is a normalisation constant to ensure that the sum of outgoing messages is zero.

Provided the definitions of $r_{i\to j}$ and $q_{j\to i}$ messages we can now summarize the execution of max-sum algorithm.

Max-sum execution. First, each of the robots arbitrarily initializes message $q_{j\to i}$ and sends it to its adjacent function nodes. This triggers an exchange of messages between variable and utility function nodes. The messages exchange will continue until message values converge, or after an user-defined number of iterations. Next, each of the robots evaluates the marginal function of variable d_j:

$$z_j(d_j) = \sum_{i \in adj(j)} r_{i \to j}(d_j). \tag{7.6}$$

Then, by simply finding $\operatorname{argmax}_{d_j} z_j(d_j)$, each individual robot i is able to determine which $\mathcal{C}_i^{[i']}$ it should visit such that $U(\cdot)$ is maximized.

The max-sum algorithm exploits the fact that the factor graph is often not fully connected; i.e. the influence of a robot only exists within a local area. In cases where the factor graph is acyclic, the max-sum algorithm delivers an exact solution. Otherwise, if the factor graph is cyclic (as in e.g. Fig. 7.2), max-sum has been empirically shown to converge to an approximation of the exact solution (Farinelli et al., 2008).

In this chapter the utility function $U(\cdot)$, which we aim to maximize with max-sum, corresponds to an information metric subject to constraints. Next we discuss and motivate our specific choice of the information metric.

7.4 Information Metric

A key aspect in active sensing, and information gathering in particular, consists of deciding where to move next in order to obtain a better representation of a process of interest (see Figure 7.3). Several information metrics have been proposed in the literature for active sensing (see Sec. 3.2). Specifically, in previous chapters we employed entropy and mean entropy as information metric. In this chapter we extend the range of information metrics considered, and analyze the use of MI for information gathering. For a more thorough analysis of the properties of the individual metrics analyzed in this section we refer the reader to Section 2.3 and (Cover and Thomas, 2012).

Here we compare information metrics in the context of GPs based on two properties that are fundamental for information gathering. These two properties are:

- *monotonicity* with the number p of potential measurement locations $\mathbf{X}_*$. That is, we are interested in information metrics that yield a higher value as we consider a higher p. This is the so-called "information never hurts" principle; and

- *submodularity* respect to p. In short, a submodular information metric offers diminishing returns as we increase p. This justifies the use of finite horizon approaches (as we do in this thesis), as the amount of information

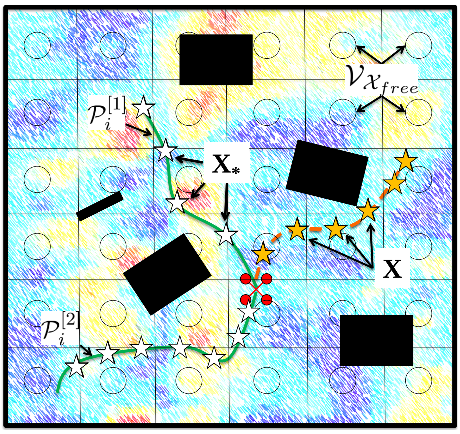


Figure 7.3: Graphical representation of the notation employed in this chapter. We depict an scenario in which a robot i (colored red) aims to explore a process (in the background) in an environment populated with obstacles (colored black). Orange stars correspond to measurements that were previously gathered by the robot at positions $\mathbf{X}$. White stars are potential measurements locations $\mathbf{X}_*$. As in this chapter we consider a path planning mechanism, $\mathbf{X}_*$ belong to potential paths $\mathcal{P}_i^{[1]}$, $\mathcal{P}_i^{[2]}$ that could be traversed by the robot. Information metrics are utilized here to quantify the informativeness of potential paths. In addition, we also represent $\mathcal{V}_{\mathcal{X}_{free}}$, together with associated grid cells, which are needed to compute some metrics.

obtained by increasing p becomes irrelevant from a certain value of p. For a detailed overview of submodularity applications in the context of GPs, we refer the reader to (Krause and Guestrin, 2011).

We analyze three information metrics that we employ in this thesis, as well as they have been employed in recent works such as (Stranders et al., 2009; Krause et al., 2008). These metrics are: (i) `Differential Entropy`, (ii) `Mutual Information Non − Measured`, and (iii) `Mutual Information All`. In the following we describe each of them in detail.

7.4.1 Differential Entropy

We denote differential entropy of a process given by random variable $Y_{\mathbf{X}_*}$, defined at $\mathbf{X}_*$, as $H(Y_{\mathbf{X}_*}|\mathbf{X})$, with $\mathbf{X}$ the location of measurements gathered by the robot up to now. $H(Y_{\mathbf{X}_*}|\mathbf{X})$ can be calculated with (2.21). A property of differential entropy that is highly relevant for information gathering is its non-monotonicity with p. That is, considering longer paths, which implies a higher p, could result in a lower differential entropy. Note also that entropy is not submodular (Cover and Thomas, 2012).

7.4.2 Mutual Information Non-Measured

We define `Mutual Information Non-Measured` as the mutual information between:

- a random variable $Y_{\mathbf{X}_*}$; and
- a random variable $Y_{\mathcal{V}_{\mathcal{X}_{free}} \backslash \mathbf{X}, \mathbf{X}_*}$ that represents the physical process at $\mathcal{V}_{\mathcal{X}_{free}}$ that would remain unmeasured after visiting $\mathbf{X}_*$.

We would like to clarify that we employ here $\mathcal{V}_{\mathcal{X}_{free}}$ instead of $\mathcal{X}_{free}$ because MI for GPs must be evaluated at a set of discrete locations (see Section 2.3). Therefore, to calculate MI we discretize $\mathcal{X}_{free}$ by overlaying a lattice graph with vertices $\mathcal{V}_{\mathcal{X}_{free}}$ (see Section 2.2.1). Also note that, for $\mathbf{x} \in \mathbf{X}, \mathbf{X}_*$ and $\mathbf{x}' \in \mathcal{V}_{\mathcal{X}_{free}}$, we assume that $\mathbf{x} = \mathbf{x}'$ if $\mathbf{x}$ lies within the cell associated to $\mathbf{x}'$ (see Figure 7.3). For example, to calculate $\mathcal{V}_{\mathcal{X}_{free}} \setminus \mathbf{X}, \mathbf{X}_*$ for $\mathcal{P}_i^{[1]}$ we would discard all $\mathbf{x}' \in \mathcal{V}_{\mathcal{X}_{free}}$ where there is an orange star or a white star that belongs to $\mathcal{P}_i^{[1]}$.

`Mutual Information Non-Measured` is given by: $I(Y_{\mathcal{V}_{\mathcal{X}_{free}} \backslash \mathbf{X}, \mathbf{X}_*}; Y_{\mathbf{X}_*} | \mathbf{X}) = H(Y_{\mathbf{X}_*} | \mathbf{X}) - H(Y_{\mathbf{X}_*} | \mathbf{X}, Y_{\mathcal{V}_{\mathcal{X}_{free}} \backslash \mathbf{X}, \mathbf{X}_*})$, which can be calculated with (2.21). This expression has a clear interpretation for information gathering: we aim to sample at locations $\mathbf{X}_*$ that yield a maximum inter-dependence with process $Y_{\mathcal{V}_{\mathcal{X}_{free}} \backslash \mathbf{X}, \mathbf{X}_*}$, defined at all positions in the environment that will remain unmeasured.

`Mutual Information Non-Measured` was first proposed by Krause et al. (2008). It is submodular, which permits deriving theoretical guarantees for information gathering with GPs for robots that perform a greedy exploration, as it was shown in (Krause et al., 2008). However, the aforementioned definition of mutual information is not monotonic as we increase p, as pointed out in (Krause et al., 2008). Thus longer paths may result in a lower value of the information metric.

7.4.3 Mutual Information All

In this chapter we propose an alternative use of mutual information – `Mutual Information All` – that is monotonic and submodular. `Mutual Information All` calculates the MI between:

- a random variable $Y_{\mathbf{X}_*}$; and

- a random variable $Y_{\mathcal{V}_{\mathcal{X}_{free}}}$.

`Mutual Information All` is given by the following expression:

$$I(Y_{\mathcal{V}_{\mathcal{X}_{free}}}; Y_{\mathbf{X}_*}|\mathbf{X}) = H(Y_{\mathbf{X}_*}|\mathbf{X}) - H(Y_{\mathbf{X}_*}|\mathbf{X}, \mathcal{V}_{\mathcal{X}_{free}}), \tag{7.7}$$

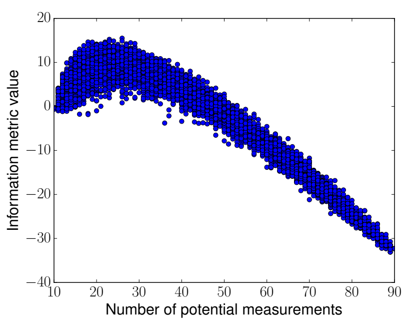
which can be calculated with (2.21). This definition is submodular, as well as monotonic with p.

7.4.4 Choice of Information Metric

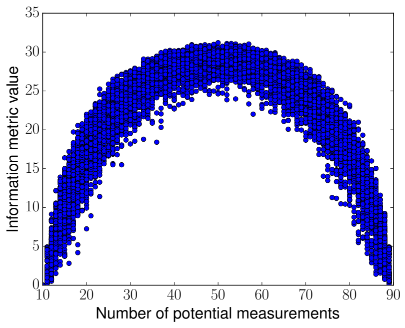
We have discussed about monotonicity and submodularity of three information metrics. In order to illustrate these two properties we carried out a simple simulation. Specifically, we considered a one-dimensional space $\mathcal{V}_{\mathcal{X}_{free}}$ that consists of 90 equally separated positions. Then we assumed that a robot already took ten measurements, drawn from a GP at positions $\mathbf{X}$ randomly selected from $\mathcal{V}_{\mathcal{X}_{free}}$. For this setup, we evaluated the afore-described information metrics as we increase p (illustrating longer planing horizons). That is, we randomly selected from $\mathcal{V}_{\mathcal{X}_{free}}$ a number of potential measurements positions, which is given by $\mathbf{X}_*$. Results from this experiment are depicted in Figure 7.4, where each dot corresponds to a realization of the experiment. From Figure 7.4 we can draw the following conclusions:

Differential entropy. `Differential Entropy` is non-monotonic, which goes against the principle of "information never hurts". Non-monotonocity is a property that is particularly undesirable for algorithms that aim to plan over an horizon longer than one step, as it is the case in (7.1). Nevertheless, `Differential Entropy` is the metric, from the three considered, that offers the lowest computational complexity (see Table 2.1). According to the two characteristics – non-monotonicity and low computational complexity compared to MI – we can argue about the use of entropy as information metric.

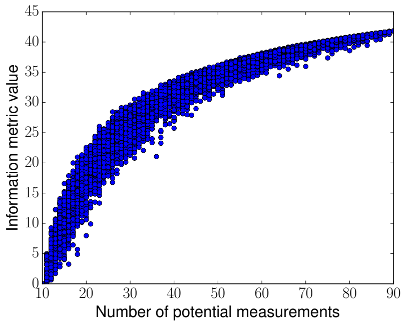
For example, in Chapter 5 we employed entropy as information metric although our goal was to plan over a long horizon. This was essentially motivated by the need of an intensive computation of the information metric. That is, in Chapter 5 the algorithm requires computing the informativeness of a complete path from $\mathbf{x}_r$ for each RRT* node addition. As the number of RRT* nodes is typically large for complex problems, this led us to use entropy instead of MI due to entropy's lower computational complexity. In contrast, in this chapter we aim to plan over long horizons but we do not need to compute the information metric that often as in Chapter 5. This led us to use MI here.



(a) Differential Entropy.



(b) Mutual Information Non-Measured.



(c) Mutual Information All.

Figure 7.4: Evaluation of several information metrics as we increase the number of potential measurements. (a) `Differential Entropy`; (b) `Mutual Information Non-Measured`; (c) `Mutual Information All`. Figure 7.4c corresponds to our proposed metric.

Mutual information. In addition to entropy, we analyzed two uses of MI: `Mutual Information Non-Measured` and `Mutual Information All`. `Mutual Information Non-Measured` is monotonic in the first part of the curve, which is an important property. However, it does not allow us to plan over a long horizon, which could be a requirement of some applications. In this chapter we proposed the use of `Mutual Information All` as information metric to tackle this problem. In contrast to `Mutual Information Non-Measured`, `Mutual Information All` is monotonic, which is an ideal choice for information gathering purposes. Also, we would like to point out that MI is a computationally expensive metric, even if it does not require an intensive computation. However, we alleviate this issue by proposing some approximations in Section 7.6.

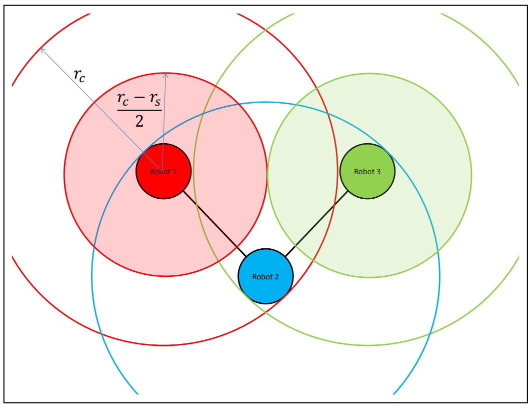


Figure 7.5: Illustration of a constraint that we introduce in the RRT to guarantee collision-free paths between robots that cannot directly communicate with each other. Specifically, we show an example for robots 1,3; where empty circles correspond to the robots communication range, shaded circles delimit the robots's planning area, and black lines represent communication links between two robots.

7.5 Algorithm Subsystems

7.5.1 Calculate Candidate Paths and Generate Clusters

The first step of the algorithm is the computation of a set of feasible paths given $\mathbf{x}_{r_i}$ and $\mathbf{f}_m(\cdot)$. This is realized with the RRT algorithm (Alg. 2.1). In particular, we introduce a constraint in the RRT that guarantees collision-free paths between robots that cannot directly communicate with each other. We realize this by limiting the RRT planning horizon to a maximum distance of $(r_c - r_s)/2$ (see Figure 7.5).

Let us denote the set of paths generated by robot i with RRT as $\mathcal{P}_{i,rrt}$. Ideally, we would like robots to exchange $\mathcal{P}_{i,rrt}$, and calculate $\mathcal{P}_i \in \mathcal{P}_{i,rrt}$ that solves (7.1). However, as we pointed out in Sec. 7.2, this would translate in evaluating multiple combinations of paths, which is computationally intractable. Therefore, inspired by (Stranders et al., 2010), we introduce the concept of spatio-temporal clusters.

Spatio-temporal clusters. Spatio-temporal clusters give us flexibility to adapt our algorithm to the robot's computational capabilities: as we increase the number of spatial and temporal divisions, we get closer to the actual RRT. However, clusters may lead to a lose of performance when optimizing $U(\cdot)$, since robots mask several paths into a cluster during the cooperation procedure. Nevertheless, we demonstrate in Section 7.7.4 that performance decrease is negligible

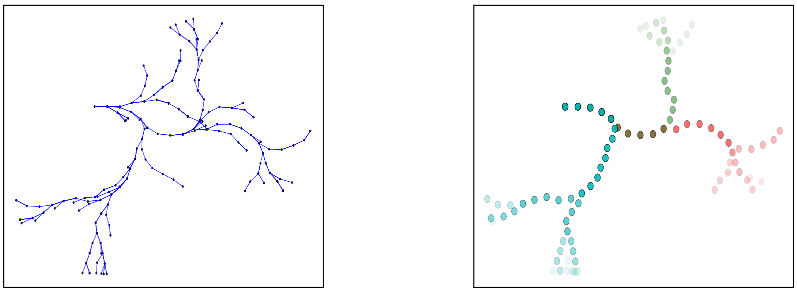

(a) Tree. (b) Clusters.

Figure 7.6: Spatio-temporal clustering. On the left hand side we depict an RRT. On the right hand side we depict the clusters calculated with our proposed clustering procedure for the RRT. Specifically, we considered one temporal horizon and three spatial clusters; i.e. $k_t = 1$, $k_s = 3$, respectively. Each of the colors represent a spatio-temporal cluster.

for a sufficiently large number of clusters (approx. 20 clusters for our setup).

Next we explain the clustering procedure in detail. First, we define k_t temporal horizons. Temporal horizons represent time spans of a path. For example, a path that needs $5s$ to be traversed by a robot has a temporal horizon of $5s$. For each of the temporal horizons, we extract the corresponding paths from the RRT. Then, we group paths of equal temporal horizon into k_s spatial clusters. This last step is realized running the k-means technique (MacQueen et al., 1967) over the complete paths. K-means requires a definition of state. In this work, we define a k-means state as a state that represents a complete path. This implies that a k-means state has a number of dimensions equal to the length of the path – number of waypoints the path contains – times d_s. It is important to remark that an RRT node could be part of different clusters, as we depict in Figure 7.6.

The clustering procedure is executed by each of the robots individually, yielding $k_t \times k_s$ clusters. Let us denote the clusters of robot i by $C_i = \left\{C_i^{[1]}, C_i^{[2]}, ..., C_i^{[k_t k_s]}\right\}$, with $C_i^{[j]} \subset \mathcal{P}_{i,rrt}$.

7.5.2 Search Neighbors and Exchange Domains

Robots move as they explore the process of interest, which results in a variation of the network topology. Therefore, we introduce a neighbors search mechanism. Robots realize this by sending an identification message with its ID. Robots that receive the identification message add the corresponding robot's ID to its set of neighbors. Then robots send C_i to their neighbors. The set of C_j received by a robot, including its own one, is denoted as a robot's domain $\bar{d}_i$, which is the element with which robots cooperate. Next, we explain the multi-robot cooperation procedure in detail.

7.5.3 Calculate Robot Utilities and Execute Max-Sum

Once robots exchange domains, they execute a cooperation algorithm – max-sum – to perform an assignment of clusters that solves (7.1). Specifically, robots cooperate to find the individual assignment $d_i : C_i^{[i']} \in C_i$ that each robot should select in order to maximize a global utility function $U(\cdot)$. Here we define $U(\cdot)$ so that it consists of two terms:

- an information gathering term, denoted as $U_I(\cdot)$, which measures the informativeness of a particular assignment of clusters; and
- a constraint satisfaction term, denoted as $U_C(\cdot)$, which takes a positive high value if problems constraints are not met.

The combination of these terms yields our proposed utility function:

$$U(\mathcal{D}, \mathbf{X}) = U_I(\mathcal{D}, \mathbf{X}) - U_C(\mathcal{D}), \tag{7.8}$$

with $\mathcal{D} = \{d_1, d_2, ..., d_N\}$.

Let us next describe $U_I(\cdot)$, $U_C(\cdot)$ in more detail.

Information gathering utility. We define $U_I(\cdot)$ as the MI between $Y_{\mathcal{V}_{\mathcal{X}_{free}}}$, and a joint assignment of clusters $\mathcal{D}$, conditioned on $\mathbf{X}$. This corresponds to `Mutual Information All`, described in Section 7.4.3, and is given by the following expression: $U_I(\cdot) = I(Y_{\mathcal{V}_{\mathcal{X}_{free}}}, Y_{d_1}, Y_{d_2}, ..., Y_{d_N}|\mathbf{X})$, with Y_{d_i} a GPs that represents $y(\mathbf{x})$ for all $\mathbf{x} \in d_i$.

Our goal is to solve $U_I(\cdot)$ in a decentralized fashion with max-sum. To this end we map our problem to (7.2), and express $U_I(\cdot)$ as a sum of functions that are associated to each individual robot. By applying the chain rule for MI, and decomposing $I(Y_{\mathcal{V}_{\mathcal{X}_{free}}}, Y_{d_1}, Y_{d_2}, ..., Y_{d_N}|\mathbf{X})$ as a difference of conditional entropies, we can express $U_I(\cdot)$ as:

$$\begin{aligned} U_I(\mathcal{D}, \mathbf{X}) &= I(Y_{\mathcal{V}_{\mathcal{X}_{free}}}, Y_{d_1}, Y_{d_2}, ..., Y_{d_N}|\mathbf{X}) \\ &= \sum_{i=1}^{N} I(Y_{\mathcal{V}_{\mathcal{X}_{free}}}, Y_{d_i}|Y_{d_{i+1}}, ..., Y_{d_N}, \mathbf{X}) \\ &= \sum_{i=1}^{N} H(Y_{d_i}|Y_{d_i+1}, ..., Y_{d_N}, \mathbf{X}) - \\ &\quad - H(Y_{d_i}|Y_{d_{i+1}}, ..., Y_{d_N}, \mathbf{X}, Y_{\mathcal{V}_{\mathcal{X}_{free}}}), \end{aligned} \tag{7.9}$$

where conditional entropies of random variables that are GPs can be easily calculated with (2.21).

***Locality* assumption.** We decomposed in (7.9) a global MI as a sum of MI of individual robots. However, this formulation, as robots only have information about their neighbors, cannot be directly applied for a system that relies on local communication between robots. We solve this issue by applying the principle of *locality* (Stranders et al., 2009; Ouyang et al., 2014). This allows us to assume that two random variables Y_{d_i}, Y_{d_l} are uncorrelated if they correspond to two robots i, l that are not in direct communication. Let us clarify that the *locality* assumption holds for a large class of applications; e.g. in this chapter's motivating problem of mapping a wind field. In this case, the structures (thermals) are only a few hundred meters in size, and the characteristic length-scales tend to be shorter. In contrast, the robots communications distances tend to be in the order of kilometers due to practical considerations.

By considering the *locality* assumption we can now formulate (7.8) as:

$$U(\mathcal{D}, \mathbf{X}) = \sum_{i=1}^{N} H(Y_{d_i} | Y_{\mathcal{N}(d_{i;i+1:N})}, \mathbf{X}) - \\ -H(Y_{d_i} | Y_{\mathcal{N}(d_{i;i+1:N})}, \mathbf{X}, Y_{\mathcal{V}_{\mathcal{X}_{free}}}) - U_C(d_i, \mathcal{N}(d_i)), \tag{7.10}$$

where $\mathcal{N}(d_{i;i+1:N})$ denotes assignment variables associated to neighbors of robot i with a higher ID, and $\mathcal{N}(d_i)$ denotes assignment variables that are associated to neighbors of the i-th robot.

Constraint satisfaction utility. The role of $U_C(\cdot)$ is to satisfy that problem specific constraints are not violated. To this end, we set $U_C(\cdot) = 0$ if robots are in a configuration that is far from violating the constraints. Otherwise, we set $U_C(\cdot)$ to a value that increases within a "scape" distance r_e as robots get closer to a configuration where constraints could be violated (see Figure 7.7).

This definition of r_e allows robots to avoid states that may result in a future violation of constraints. For example, two robots could plan paths that lead them to a configuration where, due to robots' kinematic constraints, robots can not move without colliding. Such situation is alleviated with r_e, as robots would try to avoid such blockades.

We described the general approach that we propose to account for problem specific constraints. Next let us particularize $U_C(\cdot)$ to satisfy constraints from (7.1). Specifically, in this chapter we account for the following constraints:

1. *Inter-robot collision avoidance*: it penalizes robots that are separated a distance smaller than $r_s + r_e$.

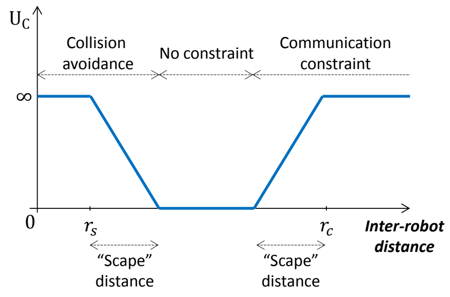


Figure 7.7: Graphical illustration of $U_C(\cdot)$.

2. *Periodic network connectivity*: it penalizes robots configurations that could lead to a disconnected network at $t + k_c dt$; i.e. at the end of robots' paths.

The last constraint can be incorporated into our scheme by separating it in two sub-constraints. First, we add a constraint that penalizes robots whenever the complete team does not choose a path of equal length k_c; i.e. a common temporal horizon. Second, we add a constraint that penalizes network configurations at $t + k_c dt$ that result in a disconnection. As robots rely on local communication, it is not trivial to check network connectivity (Michael et al., 2009). Therefore, as in (Gan et al., 2014), we guarantee connectivity by forcing robots to form a minimal topology – chain topology. That is, we encourage robots to be at least connected to their peers that have an immediate lower and higher ID. This way, we can solve the communication constraint only with local communication. We would like to remark that more complex mechanisms like eg. (Yang et al., 2010; Michael et al., 2009) could be introduced in the cooperation procedure to guarantee network connectivity, but it is left for future work.

Path selection. We explained how we optimize $U(\cdot)$ in a decentralized fashion with max-sum. For our specific formulation max-sum outputs, for each robot, an optimal cluster $d_i^* \in C_i$, which englobes multiple paths. Therefore, robots must select a path to follow from d_i^*. This is done by calculating the MI between $Y_{\mathcal{V}_{\mathcal{X}_{free}}}$, and a random variable $Y_{\mathcal{P}_i}$ that represents all possible path assignations within the selected cluster, with $\mathcal{P}_i \in d_i^*$. We condition MI on the knowledge about the selection of clusters $\left\{d_j^*\right\}_{j \in \mathcal{N}_i}$ of neighboring robots $\mathcal{N}_i$, and previously gathered measurements $\mathbf{X}$. This yields the following MI: $I(Y_{\mathcal{V}_{\mathcal{X}_{free}}}, Y_{\mathcal{P}_i} | \left\{d_j^*\right\}_{j \in \mathcal{N}_i}, \mathbf{X})$. Let us remark that this procedure is done by each of the robots independently.

7.5.4 Follow Path and Collect Measurements

The output of the cooperation stage is a path $\mathcal{P}_i$. Then robots follow $\mathcal{P}_i$, and collect measurements along it. Robots add measurements values to $\mathbf{z}$, and measurements positions to $\mathbf{X}$.

7.5.5 Exchange Measurements (Data Fusion)

Data fusion allows robots to have a common understanding about the process of interest. In this thesis we focus on multi-robot cooperation/coordination strategies, and consider decentralized data fusion out of the scope of this work. Therefore, we implement a simple flooding algorithm to carry out the data fusion. It works as follows. First, robots broadcast $\mathbf{z}, \mathbf{X}$ to their neighbors. Second, once a robot receives measurements it will broadcast those new measurements again, but only if this is the first time that they were received. This will continue till all robots have received measurements of the complete team.

We would like to remark at this point that the data fusion subsystem is the only subsystem that employs a broadcast to distribute information. Except the data fusion subsystem, the rest of the algorithm runs in a fully decentralized fashion with local inter-robot communication. Nevertheless, decentralized data fusion approaches like e.g. (Chen et al., 2012) could be considered to obtain a fully decentralized system.

7.5.6 Update GPs Model

The last step of the algorithm is an update of the GPs model with the new measurements. This is done by each of the robots individually by optimizing (2.5). Notice that the larger the number of measurements, the better the model, and the better our utility function will guide robots to perform an efficient exploration. For a detailed analysis of the GPs model update, we refer the reader to the GPs hyperparameters analysis performed in Figures 5.8 and 6.8.

7.6 Computational Complexity

In this section, we carry out an study of the computational complexity of the proposed algorithm. For this study, we build on some of the computational complexity results introduced in Chapter 2. We divide this study in three variants of the algorithm in order to highlight different aspects of the approach:

- `NoCluster`. This corresponds to the algorithm described in Section 7.5, but without considering the clustering method. That is, we consider there are as many clusters as paths resulting from the RRT for all time horizons (that is, as nodes in the RRT), where each cluster has a single path. This allows us to highlight the complexity in terms of the number of collected measurements, and total number of robots.

- `Cluster`. This is the algorithm described in Section 7.5. Here we highlight complexity reduction that results by introducing a clustering method.
- `ClusterSimplified`. This corresponds to algorithm described in Section 7.5 plus additional techniques that reduce the computational complexity. These techniques: are kd-trees, sparse GPs (Quiñonero-Candela and Rasmussen, 2005), and the principle of *locality* (Guestrin et al., 2005). Let us remark that the last two techniques discussed in this point (sparse GPs, and principle of *locality*) are approximations that do not yield exact solutions. Nevertheless, these techniques have been shown to work well in practice in a large domain of problems as discussed in (Quiñonero-Candela and Rasmussen, 2005; Guestrin et al., 2005).

Next we analyze the worst-case computational complexity for the three aforementioned algorithm variants.

7.6.1 NoCluster

The `NoCluster` variant has three main components that define the algorithm's computational complexity: (i) RRT planner, (ii) calculation of max-sum utilities, and (iii) update of the GPs model.

- The complexity of RRT is given by $\mathcal{O}(N_p \log N_p)$, with N_p the number of RRT planner iterations (LaValle and Kuffner, 2001).
- The complexity of max-sum is determined first by the calculation of $H(Y_{d_i}|Y_{\mathcal{N}(d_{i;i+1:N})}, \mathbf{X}, Y_{\mathcal{V}_{\mathcal{X}_{free}}})$ in (7.10). The complexity of this calculation is determined by that of the GPs regression, which is cubic on the total number of elements m contained in $\mathcal{N}(d_{i;i+1:N})$, $\mathbf{X}$ and $\mathcal{X}_{free}$ (Table 2.1). Since a robot calculates the utility of each combination of clusters (in this case, equal to the number of samples in the RRT), the overall complexity of max-sum is $\mathcal{O}(m^3 N_p^{|\mathcal{N}_C|})$, with $|\mathcal{N}_C|$ the number of elements of $\mathcal{N}_C \triangleq \mathcal{N}(d_{i;i+1:N})$ (proportional to the number of neighboring robots).
- The last component is the update of the GPs model in (2.5). This is given by $\mathcal{O}(n^3 i_G)$, where n is the total number of gathered measurements, and i_G is a user-defined parameter that sets the number of iterations we allow the optimizer to calculate the GPs hyperparameters.

The complexity of the `NoCluster` variant is thus determined by the calculation of max-sum utilities, as N_p is typically larger than n. The benefit of using a decentralized approach such as max-sum is illustrated by noticing that the complexity scales with the number of neighbors, and not with N. However, it is clearly influenced by N_p, which is typically large for robots with complex dynamics, or environments with multiple obstacles. Therefore, in order to reduce the algorithm's computational complexity we proposed in this chapter a concept of clustering.

7.6.2 Cluster

In the `Cluster` variant, the RRT structure is exploited to group N_p nodes in $k_s \times k_t$ clusters. This yields a max-sum complexity of $\mathcal{O}(m^3(k_s k_t)^{|\mathcal{N}_C|})$. The complexity is thus now dependent on the total number of clusters, which is typically much smaller than N_p due to the tree structure (as shown in Figure 7.6). Of course, the clustering method adds additional complexity to the algorithm. However, this is residual compared to the reduction obtained in max-sum. Specifically, the clustering method corresponds to k-means Lloyd's algorithm (Lloyd, 1982), which, for our specific problem, has a running time of $\mathcal{O}(N_p k_s k_t d_c i_c)$, with d_c the maximum number of dimensions of a k-means state, and i_c the number of iterations of Lloyd's algorithm. Let us remark that d_c is the largest temporal horizon of the clustering method times d_s.

The complexity reduction of the `Cluster` variant is vital for an online algorithm. However, it could not be sufficient for an exploration algorithm that must run in real time and cover large areas. Specifically, the `Cluster` variant faces two main problems: (i) the complexity increase of GPs regression in max-sum that results as m grows, and (ii) the complexity increase in the update of GPs that results as n grows.

7.6.3 ClusterSimplified

In order to alleviate the two aforementioned problems we propose a solution that we term `ClusterSimplified`. On the one hand, we exploit the principle of *locality* to reduce the complexity, assuming that $\mathbf{x}, \mathbf{x}'$ that are far apart are uncorrelated, and therefore do not need to be considered to carry out regression. In particular, in this work we assume that $\mathbf{x}, \mathbf{x}'$ are far if the evaluation of the covariance function results in a value that is ten times smaller than the noise level. Let us point out that this is a reasonable assumption as in this thesis we consider sensors with a negligible noise level. In order to efficiently search for locations that are correlated, we structure the data in a kd-tree.

The complexity of GPs regression is further alleviated by employing sparse

	`NoCluster`	`Cluster`	`ClusterSimplified`
Path planner	$\mathcal{O}(N_p \log N_p)$	$\mathcal{O}(N_p \log N_p)$	$\mathcal{O}(N_p \log N_p)$
Clustering method	-	$\mathcal{O}(N_p k_s k_t d_c i_c)$	$\mathcal{O}(N_p k_s k_t d_c i_c)$
Max-sum	$\mathcal{O}(m^3 N_p^{\lvert\mathcal{N}_C\rvert})$	$\mathcal{O}(m^3 (k_s k_t)^{\lvert\mathcal{N}_C\rvert})$	$\mathcal{O}(m_s^3 (k_s k_t)^{\lvert\mathcal{N}_C\rvert})$
Updating GPs	$\mathcal{O}(n^3 i_G)$	$\mathcal{O}(n^3 i_G)$	$\mathcal{O}(n_s^3 i_G)$

Table 7.1: Evaluation of algorithm's complexity. For clarification, let us add that typically $m_s << m$, $k_s k_t << N_p$, and $n_s << n$.

GPs (Quiñonero-Candela and Rasmussen, 2005). Specifically, we use the FITC method, with inducing points selected randomly from the set of potential measurements. For more detail on the FITC method we refer the reader to the original paper (Quiñonero-Candela and Rasmussen, 2005). Since the number of inducing points is typically set to be much smaller than the number of potential measurements, sparse GPs incur into an enormous reduction of complexity (Quiñonero-Candela and Rasmussen, 2005).

7.6.4 Summary

To finalize, we summarize in Table 7.1 the complexity of the three algorithm variants that we proposed in this section. Let us point out that m_s, n_s in Table 7.1 are the number of potential measurements, and actual measurements, respectively, which result after applying sparse GPs and *locality* approximations.

Motivated by a lower computational complexity and a similar performance, compared to other alternatives, we decided to employ `ClusterSimplified` in our simulations and experiments.

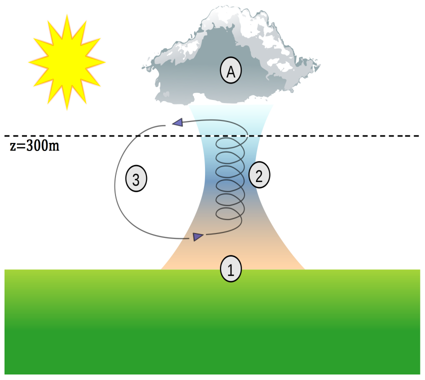
7.7 Simulations and Discussion of Results

7.7.1 Simulations Setup

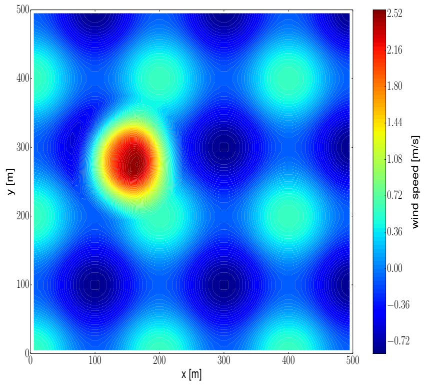
7.7.1.1 Process to Explore: Two-Dimensional Wind Field

We first validate our algorithm in simulations for the following application: multi-robot exploration of the vertical component of a wind field (see Figure 7.8a[1]). This is a first step towards performing autonomous soaring of multiple gliders in a distributed fashion. Since gliders use the wind energy to fly, estimating the position of thermals can be used to exploit them to gain energy (Chung et al.,

[1]By Dake (Self-made illustration) [CC BY 2.5 (http://creativecommons.org/licenses/by/2.5)], via Wikimedia Commons.



(a) Thermal.



(b) Wind field.

Figure 7.8: Illustration of a thermal (7.8a) and the two dimensional wind field to be explored (7.8b).

2014) and, therefore, would allow us to explore larger environments than, for example, with quadcopters.

The wind field is simulated using the model proposed by (Allen, 2006). (Allen, 2006) is an statistical model that characterizes updrafts, and was developed from data gathered by balloon and surface measurements. In addition, like in (Lawrance and Sukkarieh, 2011), we added a sinusoidal component in both x and y directions to increase the complexity of the information gathering task. Figure 7.8b shows the wind field to be explored. This corresponds to a $500 \times 500\ m^2$ two dimensional slice at $300\ m$ of a three dimensional wind field.

7.7.1.2 Robot Model: Simplified Aircraft

We employ a simplified aircraft model similar to the ones used by Renzaglia et al. (2016); Owen et al. (2015). We made further simplifications to adapt it to a two-dimensional environment, and assumed that the wind field does not affect the aircraft's motion. These simplifications are still far from a realistic model. However, they allow us to demonstrate the effectiveness of the proposed exploration approach. In the future, we aim to consider more realistic models of the aircraft's dynamics, as for instance in (Doo-Hyun et al., 2016).

Given these assumptions, our model is defined by the following equations:

$$\mathbf{x}(t + dt) = \mathbf{x}(t) + V_{in}(t)dt \tag{7.11}$$

$$\psi(t + dt) = \psi(t) + \dot{\psi}(t)dt, \tag{7.12}$$

with $\mathbf{x}(t)$ the aircraft's position, $V_{in}(t)$ the aircraft's inertial velocity, and ψ the heading angle. For airspeed V, commanded flight path angle θ, and commanded bank angle ϕ the components of the velocity $V_{in}(t)$ and $\dot{\psi}(t)$ are given by:

$$v_x = V \cos\theta \cos\psi \tag{7.13}$$

$$v_y = V \cos\theta \sin\psi \tag{7.14}$$

$$\dot{\psi} = \frac{g}{V} \tan(\phi). \tag{7.15}$$

Let us point out that the aircraft is fully controlled by the commanded bank angle ϕ, and flight path angle θ. For the simulations we assumed an aircraft defined by the following parameters: $dt = 0.5\ s$, $V = 15\ ms^{-1}$, $g = 9.8\ ms^{-2}$, $\theta = 0$ (constant height), $\phi \in [-\pi/5, \pi/5]\ rad.$

7.7.1.3 Algorithm Parameters

We consider a fleet of four robots – aircrafts – to explore the wind field. The aircrafts' motion can be approximated by the model described in Section 7.7.1.2. We define a communication range $r_c = 200\ m$, a safety distance $r_s = 10\ m$, and a scape distance $r_e = 20\ m$. For the simulations we run RRT for $N_p = 1000$ iterations, and max-sum for 5 seconds. The stopping criteria for both RRT and max-sum were set according to prior experience to guarantee that a solution is found. We consider four temporal horizons at $2, 5, 7, 10\ s$, and three spatial divisions. This makes 12 clusters in total for each robot.

We run Monte Carlo simulations to test our approach for systems of one, two, three and four robots. The initial position of the robots is fixed, and it is 250 m at the y direction, and $100, 200, 300, 400\ m$ at the x direction for robot $1, 2, 3, 4$, respectively. For each of the algorithms we average over 100 simulations runs. The algorithm is implemented in Python, and we use ROS (Quigley et al., 2009) to simulate the algorithm in a decentralized fashion.

7.7.2 Analysis of the Exploration Strategy

First we evaluate the performance of our proposed algorithm for an information gathering task that is subject to any constraints from (7.1). This implies that robots run our algorithm with $U_C(\cdot) = 0$. This algorithm variant we term it "SBMRE Alg. No Constraints". With this study we aim to proof the following two hypothesis:

1. The proposed cooperation procedure, which builds on MI as information metric and max-sum as decentralized cooperation technique, outperforms a benchmark algorithm.

2. Our proposed algorithm scales super-linearly as we increase the number of robots in the system. That is, as the number of robots increases the performance gap between a benchmark and our algorithm grows.

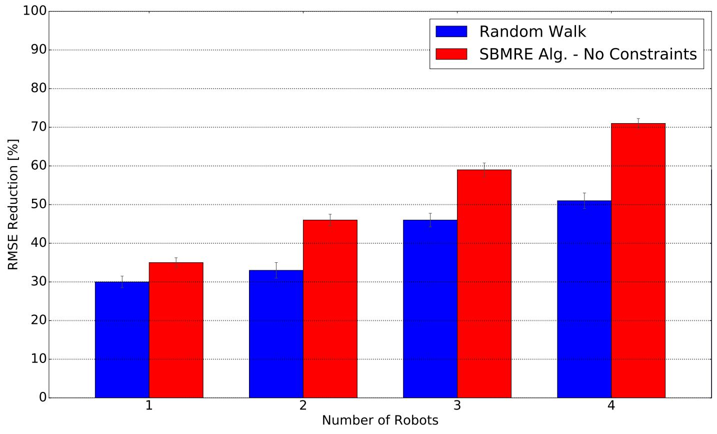


Figure 7.9: RMSE reduction during an exploration task as we increase the number of robots in the system. We benchmark our proposed algorithm (without constraints) against a random walk.

To the best of our knowledge there are no algorithms in the literature that solve (7.1); also without constraints (see Chapter 3). Therefore we selected a random walk as benchmark. A random walk implies that robots move independently following a random path, constrained by the robot motion, generated with RRT. The random walk neither aims to meet constraints nor to exchange measurements with the rest of the team. Let us remark that the random walk does not perform any data fusion, which implies that each of the robots only has the measurements taken by themselves. So, in order to obtain a fair comparison with our algorithm, which fuses data online, we perform a data fusion during post processing for the random walk benchmark.

Here we study our exploration strategy by evaluating the reduction of the RMSE, calculated with (4.2), after a $300s$ exploration run. We compute the RMSE with respect to a set of points $\mathcal{V}_{\mathcal{X}_{free}}$ that correspond to nodes of an overlaid lattice graph with a spatial resolution of 10 m. We use these points to compare the difference between our estimate $\boldsymbol{\mu}_*$, which is the result of GPs regression given $\mathbf{z}, \mathbf{X}$, and ground truth $\mathbf{y}_G(\mathbf{X}_G)$, with $\mathbf{X}_G := \mathcal{V}_{\mathcal{X}_{free}}$.

We depict in Figure 7.9 the RMSE reduction for one, two, three and four robots. First fact that we observe is that our algorithm offers an increase of performance with respect to a random walk of a 6% with one robot, and increases up to a 20% with four robots. Next fact is that the gap between our algorithm's performance and a random walk increases as we add more robots to the team. According to results from Figure 7.9 we can confirm our two hypothesis.

7.7.3 Analysis of the Multi-Robot Coordination Strategy

We demonstrated our algorithm's cooperation capabilities to gather information. Next we analyze our algorithm's coordination capabilities to meet problem specific constraints from (7.1). Therefore, here we aim to proof two hypothesis, which correspond to the inter-robot constraints considered in this work. These are the following:

1. Our algorithm meets the collision avoidance constraint, and outputs collision-free trajectories.
2. The network connectivity constraint is fulfilled, and our algorithm guarantees a higher connectivity than a random walk benchmark.

7.7.3.1 Collision Avoidance

This section evaluates the collision avoidance capabilities of our algorithm. In particular, we calculate the percentage of time that the constraint is not met during all simulation runs. This number ranges between 0.1% and 0.2%. We obtained this number by calculating the distance between each pair of robots for each iteration of the algorithm. Let us remark here that a low percentage is still possible as the scape distance could be violated. In this sense, local safety measures and obstacle avoidance mechanisms (Alejo et al., 2014) could be employed to solve such conflicts.

A fundamental feature of our algorithm is that the violation of the collision avoidance constraint can be detected in advance by evaluating the robot individual utility function. In contrast, a random walk has no means to anticipate a future possible collision without an external collision avoidance system.

7.7.3.2 Network Connectivity

Next we evaluate the fulfillment of the network connectivity constraint. To this end, we calculate the percentage of iterations in which the network is not connected at the end of robots' paths (during max-sum execution) for all simulation runs. This means that there are robots or subsets of robots that cannot communicate with the rest of the team, and therefore they violate the periodic connectivity constraint. As pointed out before, non-connectivity is an undesirable characteristic for most applications (Hollinger and Singh, 2010; Gan et al., 2014).

Figure 7.10 shows the network connectivity for our algorithm and a random walk. Our proposed algorithm achieves a network connectivity that ranges between 91% and 98%. In contrast, the random walk achieves a connectivity that ranges between 42% and 60%.

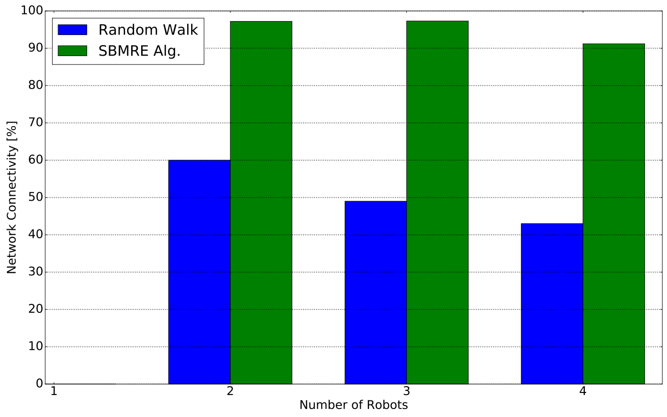


Figure 7.10: Network connectivity. We depict the percentage of iterations in which the network fulfills a periodic connectivity constraint.

7.7.4 Analysis of the Clustering Procedure

The evaluation of the exploration and coordination strategies illustrate the effectiveness of our approach to solve problem (7.1): performing an information gathering task while fulfilling problem specific constraints. However, we analyzed the algorithm's performance for a fixed configuration of parameters. In this section we evaluate the algorithm's sensitivity to changes in parameters values. In particular, we focus the study on the two most relevant parameters: number of spatial-temporal clusters, and communication radius. For these two parameters we analyze: (i) the resulting RMSE between estimation and ground truth after three iterations of the algorithm, and (ii) the solution feasibility; i.e. how often the algorithm is able to find a solution that meets the constraints imposed in problem (7.1).

We carry out the analysis for an environment that measures 1000×1000 square meters, with a wind field that is similar to the one shown in Figure 7.8b but it contains two thermals. For that scenario, we run 5000 Monte Carlo simulations with randomly chosen parameters. Specifically, the number of clusters ranges from 1 to 36, and we consider a communication radius of 200, 300, 400, 500 and 2000 meters. Let us also add that we let the max-sum algorithm run for 180 seconds each algorithm iteration in order to being able to calculate all utilities for up to 25 clusters.

Figure 7.11 depicts the results of the parameters analysis. The depicted curves are the result of a quadratic curve fitting done on the original data. This was done to improve the clarity of the figure. We also include the original data with the individual curve fittings in Appendix 7.A.

From Figure 7.11 we can extract four main conclusions:

1. The softer the constraints, i.e. a larger communication radius, the better the algorithm's performance both in terms of the RMSE and the solution feasibility.

2. Our algorithm's performance increases, i.e. lower RMSE and higher solution feasibility, as we increase the number of clusters up to approximately 20 clusters. This demonstrates that the larger the number of clusters, the better we represent the original RRT, which translates into a more efficient multi-robot cooperation.

3. Our algorithm is scalable with the number of clusters; which implies that there is no need of employing a very large number of clusters since it will not lead to a better solution. This property leads to an enormous reduction of the algorithm's computational complexity as indicated in Section 7.6. This is exemplified by the fact that performance of the algorithm remains approximately constant for a number larger than approximately 20 clusters. This lies on the fact that adding new clusters does not improve the representability of the original RRT, since clusters start containing paths that are very similar.

4. Performance of the algorithm decreases if we do not let max-sum algorithm converge due to insufficient running time. Then, the solution that it outputs is suboptimal. This happens as we use more than 25 clusters in this setup, approximately. This emphasizes the importance of point 3, since according to Figure 7.11 with a number of clusters equal to approximately 20 we obtain the best performance both in terms of RMSE and solution feasibility.

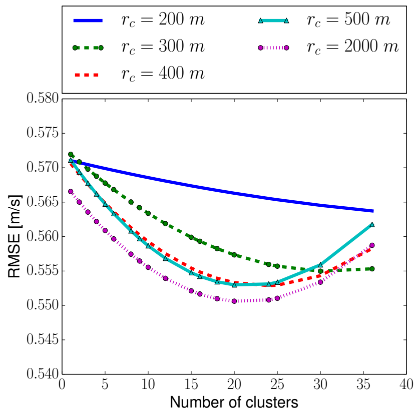
This section concludes the analysis of the algorithm in simulations. Next we present experimental results.

7.8 Experiments and Discussion of Results

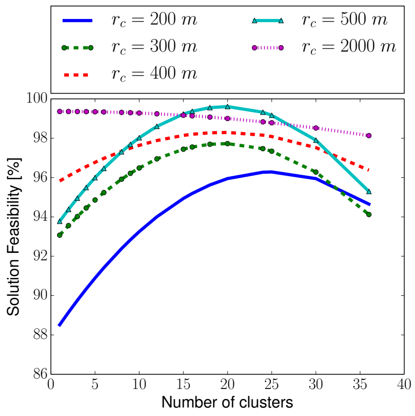
In addition to simulations, we carried out a field experiment [2] with flying robots. Specifically, we performed a simulation with robots in the loop, where we explored a simulated two-dimensional wind field with quadcopters emulating a fixed-wing aircraft's dynamics.

In particular, we aim to proof the following statements:

[2]A video that shows the experiment execution can be found in: `https://vimeo.com/253607385`; `https://rebrand.ly/sampl7137`.



(a) RMSE.



(b) Solution Feasibility.

Figure 7.11: Algorithm's performance as we vary the two most relevant parameters: number of spatial-temporal clusters, and communication radius. For these two parameters we analyze: (i) in Figure 7.11a the resulting RMSE between estimation and ground truth after three iterations of the algorithm, and (ii) in Figure 7.11b the solution feasibility; i.e. how often the algorithm is able to find a solution that meets the constraints imposed in problem (7.1).

1. the system is able to perform an active sensing, online, according to the measured values.
2. the algorithm is robust against inaccuracies in robots' position.

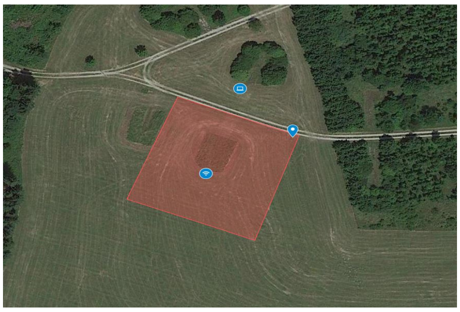
Next we describe in detail the experimental setup and results.

7.8.1 Experimental Setup

We describe in this section the experimental setup employed for this experiment. First, we describe the wind field model and algorithm parameters. Then, we give details about the robots we used to perform the exploration. We finalize with a description of the required hardware components, and system's infrastructure.

7.8.1.1 Wind field model

We simulated a wind field, instead of measuring an actual field, in order to simplify the overall experiment. This allows us to abstract ourselves from the particular sensor characteristics, and evaluate the algorithm's performance in a real scenario. As part of the future work, we aim to extend the work proposed in this thesis to account for a real sensor that measures a wind field.


(a) Exploration area.

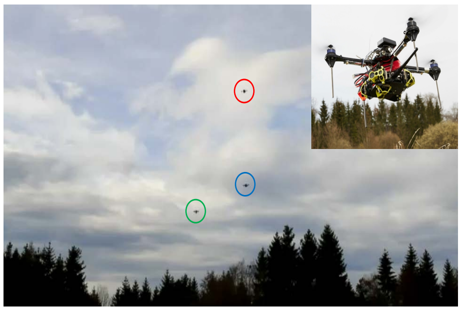

(b) Quadcopters during the experiment.

Figure 7.12: Experiment's environment.

In this paper, the wind field corresponds to a scaled down version of the one described in Section 7.7.1 (see Figure 7.8). Specifically, we reduced the size of the environment by a factor 10. This results in a wind field over an area of 50×50 square meters. For the exploration, we employ the same parameters as in Section 7.7.1; except dt that we set it to $dt = 0.2$ to account for a smaller environment.

7.8.1.2 System architecture

To explore the afore-described wind field, we propose a system architecture that is composed of the following main elements (see Fig. 7.12): (i) quadcopters to perform an active sensing of the wind field, (ii) a central computer to run the core of the algorithm and monitor the system, (iii) a Real-Time Kinematic navigation for global positioning systems (GPS-RTK) to provide robots position, and (iv) a communication infrastructure. Next we provide details of each of the components.

Quadcopters. We use three quadcopters that emulate the dynamics of a simple fixed-wing aircraft. In particular, we employ (7.12) to plan the quadcopters trajectories. A trajectory can be represented as a set of waypoints that the quadcopters can follow using their onboard controllers. This way, quadcopters will perform a flight path that is close to the one performed by a fixed-wing aircraft. For the experiments, we let the quadcopters fly at different heights of 8, 12 and 15 meters. Although we included a collision avoidance mechanism into the planner, inaccuracies in the positioning system could result in unexpected collisions. Therefore, for safety reasons, we flew at different heights. Let us remark that quadcopters are not aware of the height at which they fly.

Figure 7.12b shows one of the quadcopters used for the experiment. Quadcopters are a modified version of an AscTec Hummingbird from Ascending Technologies. We equipped them with a Raspberry Pi 2 Model B to perform calculations onboard. However, the core of the algorithm runs in a central computer due to the insufficient computational capabilities of a Raspberry Pi.

Central computer. A laptop situated outside the exploration area monitors the complete system, and runs the core of the algorithm. Specifically, it executes the algorithm that coordinates robots, and then sends waypoints to quacopters. Quadcopters will then fly to the commanded waypoints, using the onboard controller that runs in the Raspberry Pi. It is important to remark that the algorithm runs in a distributed fashion where each quadcopter runs in a separate software module – ROS node.

GPS-RTK. Quadcopters are also equipped with a GPS-RTK (Misra and Enge, 2006). Specifically, we mounted the Piksi 1 modules from Swift Navigation. It allows us to achieve a sub-meter-level accuracy in the position.

A GPS-RTK system requires a base station to transmit correction data (Misra and Enge, 2006). The base station receives positioning information from both the quadcopters and a GPS receiver that is mounted on it. Then it uses that information to compute correction data that is broadcasted back to the quadcopters. However, due to hardware issues, the GPS-RTK receiver mounted on the quad is sometimes not able to calculate the position solution with the highest precision. This is so-called float solution, which offers an accuracy in the order of meters. In Section 7.8.2 we discuss these issues, and show that the proposed algorithm is able to handle uncertainty in quadcopter's position.

Communication. We need a communication channel to send waypoints that must be flown by quadcopters, and to transmit correction data from the GPS-RTK station to quadcopters. To this end, we built two communication channels. On the one hand, we use a Wifi router to transmit information to quadcopters, since quadcopters are not equipped with peer-to-peer communication modules. Based on previous experience, we noticed that range of the Wifi signal extends only up to 50 meters, which requires a careful placement of the router, given the size of our exploration area and the quadcopters flying height. We chose to locate the router in the center of the exploration area to guarantee a correct communication. On the other hand, Ethernet cables connect peer-to-peer the laptop, Wifi router, and GPS-RTK base station.

7.8.2 Experimental Results

Here we show results for the experiment we described in Sec. 7.8.1. Specifically, we evaluate: (i) the robots' trajectories, (ii) the process reconstruction and the remaining uncertainty after the exploration task, and (iii) the RMSE between the process reconstruction and ground truth. For the last one we compare runs with one and three quadcopters to highlight the benefits of a multi-robot system.

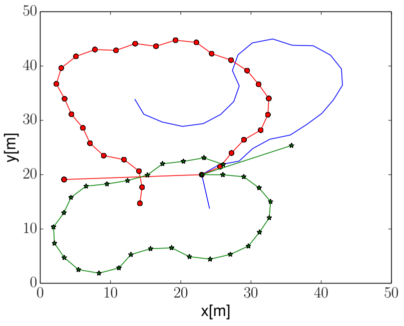
7.8.2.1 Robots Trajectories

First, we depict in Figure 7.13 the trajectories that quadcopters flew during the exploration run. Figure 7.13a corresponds to robots' nominal position. That is, waypoints that the algorithm commands to quadcopters. We can observe that the shape of the trajectories resembles those of a simple fixed-wing aircraft. Moreover, the robots achieve a good coverage of the exploration area. However, we notice that the bottom right corner of the area was not explored. This was due to battery life constraints, which caused that robots did not have time to cover the complete area. We would like to point out that the effective flight battery life of quadcopters for a system composed of three of them lasts approximately 5 minutes with our current setup. The remaining battery life is employed for take off, landing, and safety procedures.

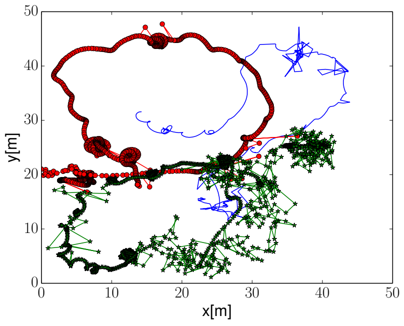
On the other hand, Figure 7.13b depicts quadcopters positions as output from the GPS-RTK system. The trajectories are similar to the nominal ones. However, we observe inaccuracies in the position solution. For example, we could concentrate in Figure 7.13b on a large concentration of dots at coordinates $x = 15$, $y = 45$ meters. These dots correspond to a single commanded waypoint where a quadcopter tries to stay at. This happens during the coordination step of algorithm when robots negotiate about their next move while staying at their current position. Ideally, we would like quadcopters to hold their position. However, this is not possible due to the combined effect of the robot's controller, which relies on external sensors to calculate its position, and the GPS-RTK solution. Both systems have inaccuracies that result in quadcopters flying in small circles around commanded waypoints. However, let us emphasize that these inaccuracies in position do not result in a low performance as we will show next.

7.8.2.2 Wind Field Estimation

Figure 7.14 shows the wind field reconstruction and entropy resulting from measurements gathered during an exploration run. Measurements are obtained at positions associated to commanded waypoints. Specifically, each quadcopter measures a wind field component with its simulated sensor once it reaches the



(a) Nominal position.



(b) Actual position.

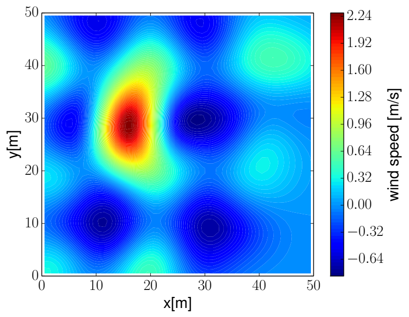
Figure 7.13: Nominal position and actual position of the quadcopters during the exploration task. Each of the three quadcopters is represented with a different color.

commanded waypoint. We set the accuracy – difference between commanded waypoint and GPS-RTK positions – to reach a waypoint as 2 meters. Then the quadcopter stores the measurement together with its current GPS-RTK position.

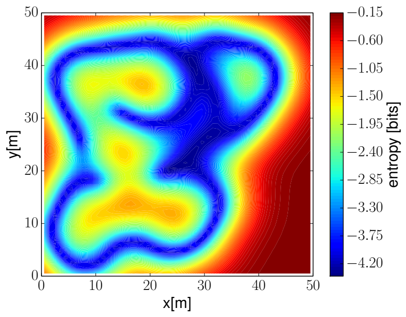
We illustrate in Figure 7.14a the estimated wind field. It corresponds to the mean prediction of the GPs at each of the positions of the environment given the collected measurements. The estimated wind field can be compared to the ground truth (depicted in Figure 7.8b). First, we observe that estimation and ground truth are almost identical, and we can easily identify the thermal. This exemplifies the algorithm robustness to uncertainty in robot's position. Second, we notice that the estimation is worse at those areas that were not covered during the exploration run; i.e. bottom right corner (also noticeable in Fig. 7.14b). However, even on that area the algorithm achieves a good reconstruction accuracy.

7.8.2.3 Error between Estimate and Ground Truth

We show in Figure 7.15 an evaluation of the RMSE between estimate and ground truth resulting from the field experiment. We show curves for one and three robots running the algorithm proposed in this work. As we showed in simulations, the system with three robots achieves a much lower RMSE compared to one robot. Specifically, three robots achieve a three-fold improvement compared to one robot. This confirms the benefits, in terms of efficiency, of a multi-robot system.



(a) Process reconstruction.



(b) Process entropy.

Figure 7.14: Process reconstruction and entropy after performing an exploration run. Process reconstruction corresponds to mean prediction at each of the positions, resulting from GPs regression. Process entropy is calculated from variance output from GPs.

7.9 Summary and Outlook

In this chapter we presented an approach for multi-robot information gathering. It considers GPs as underlying model of the process to explore, information utilities for active perception, and the max-sum algorithm for multi-robot cooperation. The approach extends the state of the art by allowing to take into account the motion constraints of the robots. This is realized through the use of motion planners able to handle such constraints, such as RRT. Furthermore, the method is able to handle mission team constraints as well, such as network connectivity and collision avoidance restrictions. We achieved this by including additional terms into the utility functions in max-sum. The whole approach is distributed, not requiring a central entity for processing. All the decision-making is decentralized, and, in our current implementation, only the data fusion component requires a broadcast mechanism at the network level (even though the system can work if the network connectivity is not fulfilled). As future work, we will consider decentralized data fusion approaches for GPs, as in (Ouyang et al., 2014; Chen et al., 2015), for a fully decentralized system.

We validated our approach in simulations for the exploration of a wind field. We also tested the methods in experiments with real robots in the loop for the same application. The results show how inter-robot cooperation permits a more efficient exploration, more evident when the number of robots grow. Furthermore, the results show how the approach can handle constraints that are relevant for real scenarios, in particular maintaining the network connectivity in the fleet.

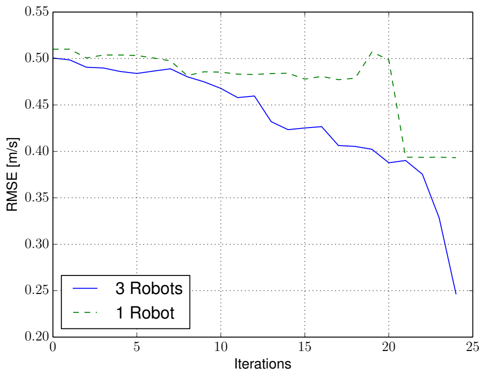


Figure 7.15: RMSE between estimate and ground truth resulting from a field experiment that we carried out with one and three quadcopters. Quadcopters run the algorithm proposed in this work.

There are still a few aspects that should be considered to improve the algorithm's performance, which constitute venues for future work. We remark that we do not implement them in this thesis. These are the following:

1. One of the limitations of the presented application is the use of a fixed chain network topology, which constraints the ability of the fleet to explore. More dynamic and flexible network topologies would definitely allow for better information gathering efficiency. Please notice that a fixed chain network topology is not a restriction of the decision making method itself, but degrades the overall algorithm's performance.

2. The vehicle models employed in this work are a simplified version of a fixed-wing aircraft. We plan to extend those models to full 3D models that consider also aerodynamic effects, as a next step to apply the approach for the autonomous soaring of gliders. We will consider exploration in 3D, and combining the exploration techniques presented with the exploitation of the wind information for longer endurance of the flight. Exploitation terms can be easily included into our utility functions.

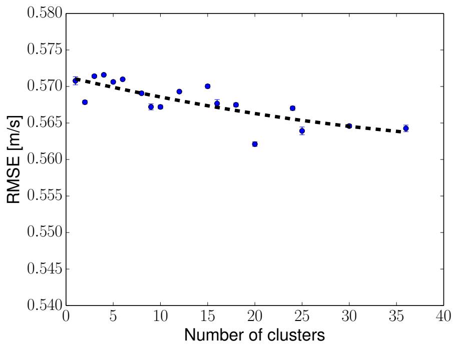
3. In this chapter, we did not considered the impact of the wind field on the robots' paths. That is, we assumed that the robot's controller is able to perfectly follow the paths planned by the robots independently of the actual wind field. As part of the future work, we aim to consider the wind field into the path planner. Specifically, our goal is to incorporate the prediction delivered by the GPs to plan paths in areas where the wind field has not been measured yet.

4. We proposed the use of RRT to plan the robot's motion. However, RRT is a path planning algorithm that outputs a feasible but suboptimal solution. In contrast, we proposed in Chapter 5 a sampling-based exploration algorithm that uses RRT* to approach an optimal solution. Therefore, a natural next step is to fuse algorithms from this chapter and Chapter 5 to obtain a solution to the multi-robot information gathering problem that is closer to optimality.

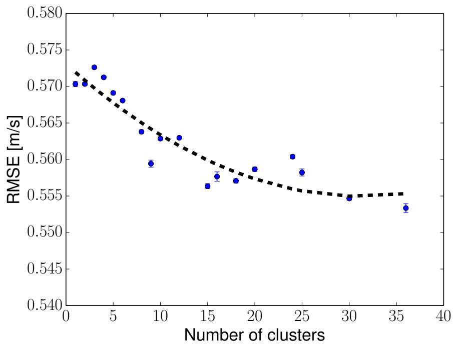
5. We performed an experiment with real robots to demonstrate the capabilities of our proposed algorithm. However we simulated the sensor. Next step would be for us using a real wind field sensor in order to get closer to a real-world scenario.

7.A Appendix - Analysis of Algorithm Parameters

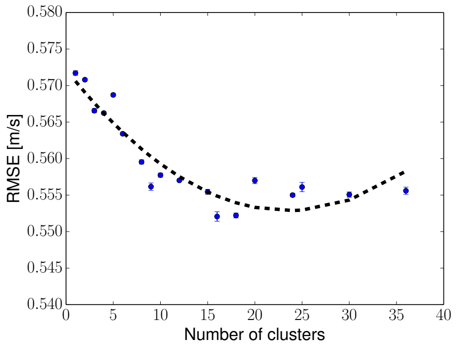
Original data used to carry out the quadratic curve fitting required for Figure 7.11. Data is depicted in Figures 7.16 and 7.17. Specifically, we carry out the curve fitting over the mean for each of the individual number of clusters. This is possible, since for this data the mean is an statistically relevant parameter as indicated by the small error bars. In blue we show the original data, and the black line corresponds to the quadratic curve fitted to the data.



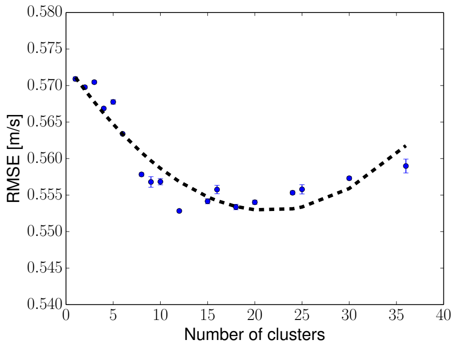
(a) $r_c = 200\ m$.



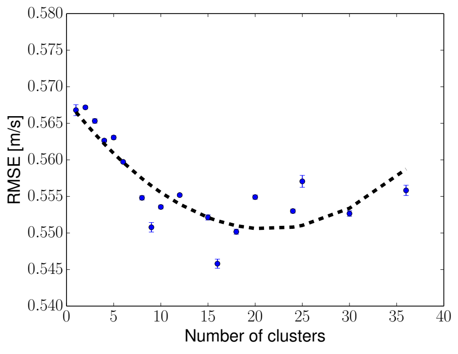
(b) $r_c = 300\ m$.



(c) $r_c = 400\ m$.

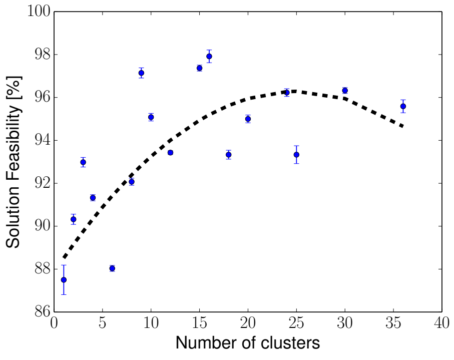


(d) $r_c = 500\ m$.

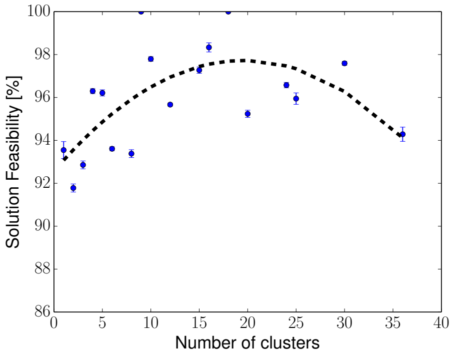


(e) $r_c = 2000\ m$.

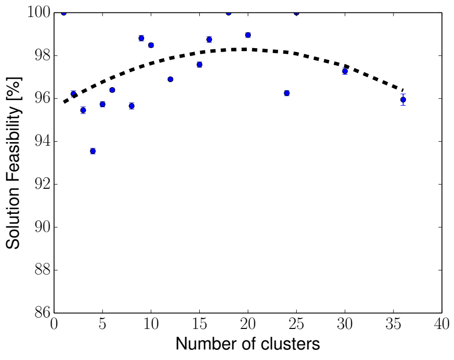
Figure 7.16: Original data employed to carry out the curve fitting needed for Figure 7.11a. The dotted lines correspond to the actual curve fitting.



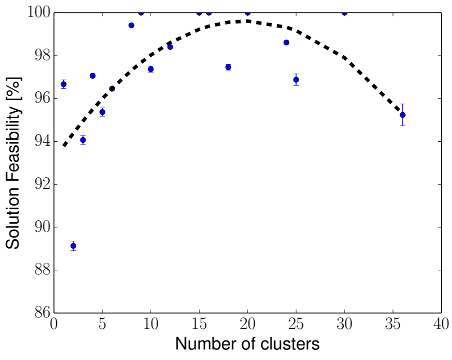
(a) $r_c = 200\ m$.



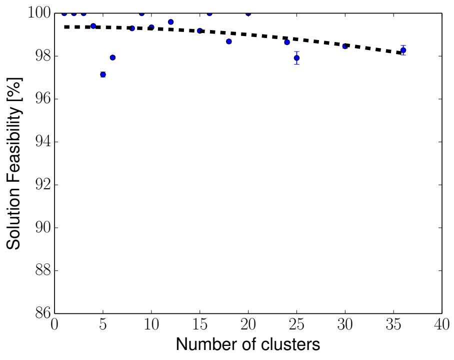
(b) $r_c = 300\ m$.



(c) $r_c = 400\ m$.



(d) $r_c = 500\ m$.



(e) $r_c = 2000\ m$.

Figure 7.17: Original data employed to carry out the curve fitting needed for Figure 7.11b. The dotted lines correspond to the actual curve fitting.

Part IV

Conclusion and Future Work

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Autonomy is one of the major driving forces behind modern robotic exploration. Especially in situations when the interaction of a human operator with a robot is difficult or impossible, e.g. in space exploration or search and rescue missions, it is crucial that a robot is able to make intelligent decisions. Moreover, the use of multiple cooperative robots enhances autonomy by increasing the system's efficiency and robustness.

Motivated by the idea of full robotic autonomy, we developed an autonomous multi-robot information gathering algorithm. Specifically, we advanced the state-of-the-art by introducing, to the best of our knowledge, the first multi-robot exploration algorithm that

1. performs a decentralized multi-robot coordination,
2. plans non-myopically in a continuous space, and
3. handles complex inter-robot constraints, such as collision avoidance and network connectivity constraints.

Clearly, there existed a gap in the current literature of an algorithm with the aforementioned features. To our understanding, this gap was due to the absence of methods that could solve the problem efficiently in real time. We tackled this issue by introducing approximations and intelligent mechanisms that reduce the overall computational complexity, without sacrificing performance. This enables a real time implementation of our system, which we demonstrated in field experiments with multiple flying robots.

The aforementioned features are crucial for most multi-robot applications, as they allow us to design systems that are efficient, robust, and can be generalized to multiple classes of robots. We strongly believe that the algorithm proposed in this thesis will enable future robotic applications that require multiple complex robots. Specifically, we proposed an algorithm that relies on three main elements that we intertwine and adapt to comply to our thesis objective. These are the following:

- GPs to model the underlying process of interest,
- action planners to calculate potential robots' actions, and
- information metrics, which act as decision makers, to select the next robots' action/s.

These elements were the basis to formulate in Chapter 7 our multi-robot information gathering algorithm that fulfills this thesis objectives. This algorithm is the result of multiple previous studies and algorithms, which solved smaller subproblems of increasing complexity. In particular, we presented a first algorithm in Chapter 4 that was discrete, myopic, and single-robot. Then we evolved the concept from Chapter 4 in subsequent chapters to formulate an algorithm in Chapter 7 that is continuous, non-myopic, and multi-robot.

We tested and evaluated the algorithms proposed in this thesis in simulations, and field experiments with ground based and aerial robots. Results of these evaluations demonstrated a significant improvement respect to state-of-the-art.

As one can conclude from previous paragraphs, this thesis was a journey, which tackled a large variety of aspects that are inherent to multi-robot exploration. In particular, we would like to drive the final discussion of our work around the pros and cons of the following aspects: (i) discrete vs. continuous planning, (ii) myopic vs. non-myopic planning, (iii) single-robot vs. multi-robot systems, (iv) centralized vs. decentralized architectures, (v) constrained vs- non-constrained multi-robot algorithms, and (vi) simulations-based vs. experimental-based evaluation. Next we discuss each of the aforementioned aspects in detail.

Discrete vs. continuous planning. We extensively discussed in this thesis the advantages and disadvantages of discrete and continuous action planners. In short: on the one hand, discrete planners are computationally lighter than continuous planners in low-dimensional spaces. On the other hand, discrete planners are not suited to robots with large state spaces, in contrast to continuous planners.

The latter aspect compromises the generality of discrete planners systems, as their use is restricted to a small class of robots. Generality of the system is one of our major objectives, as our vision is that developed algorithms could be employed for a large variety of information gathering tasks with little algorithmic changes. This is particularly relevant for applications like the exploration of a wind field that we performed in Chapter 7. In Chapter 7 we employed an aircraft with a four-dimensional state space (more complex systems could be considered with the developed algorithms). Such a robot motion cannot be in general planned with a discrete planner due to the high computational complexity. This is only a small example, but a major motivation to extend the discrete planner algorithms proposed in Chapters 4, 6 in order to incorporate continuous planners in Chapters 5, 7.

Attending to the previous arguments, it seems obvious to employ continuous planners to calculate robots actions. However, most algorithms in the multi-robot literature employ a discrete representation of the action space (see Sections 3.3, 3.4). From our point of view, this is mainly motivated by the following fact: the derivation of theoretical bounds is much easier for discrete planners as the planning graph is fixed during the exploration task. In contrast, continuous planners generate a new graph each planning step. Sacrificing algorithm's generality in favor of theoretical bounds is a dilemma that, from our point of view, should be addressed for each specific exploration task.

There is another aspect inherent to discrete planners, which is the discretization factor. The discretization factor plays a crucial role in the algorithms performance. For example, let us imagine a process that has an, unknown, spatial correlation of 5 cm. Here, selecting a discretization factor that is larger, e.g. 10 cm, would not allow robots to exploit the process spatial correlation. This would result in a suboptimal exploration strategy. Most algorithms in the literature take a hand-tuned discretization factor as input. From our perspective, this factor should not be an algorithm input, but derived by the algorithm itself. We strongly believe that this issue should be appropriately addressed by the robotic information gathering community.

Myopic vs. non-myopic planning. The selection of a myopic or a non-myopic strategy is a trade-off between performance and computational resources. On the one hand, myopic approaches are typically light. On the other hand, non-myopic approaches are typically more complex, but they offer a higher performance compared to myopic ones.

One aspect that is particularly relevant in non-myopic planning is the definition of the planning horizon. In the literature, most approaches employ a predefined planning horizon that is hand-tuned for an specific problem. In contrast, in our work we only specify a range of possible planning horizons, and let robots select the one that better fits their information gathering objectives.

Another remarkable point that is inherent to the planning horizon is the increase of the planning computational complexity as the horizon grows. We identified in the thesis that this is an issue that is not properly addressed in the literature, as most algorithms do not pursue a real time implementation. As one of our main objectives was to develop algorithms that can run online, we had to come up with novel ideas to solve this problem. One example of such idea is the concept of clusters introduced in Chapter 7. This works as follows: we take paths generated from an RRT planner, and group similar paths into clusters. Then we perform the multi-robot cooperation by treating those clusters as an unique path. This way we reduce the number of variables required for multi-robot cooperation, and, therefore, reduce the algorithm's computational complexity.

Single-robot vs. multi-robot system. We developed in this thesis both single-robot and multi-robot systems. Multi-robot systems offer clear advantages in terms of efficiency and robustness, respect to single-robot systems. This statement was validated in our work with extensive simulations and experiments.

A multi-robot system is by nature more complex than a single robot-system, as it is composed of a higher number of modules (see Chapter 7). Typically, a multi-robot system requires the following additional modules respect to its single-robot counterpart: (i) a communication module for inter-robot communication, (ii) a module for inter-robot collision avoidance, (iii) a data fusion module to distribute the data gathered by robots, and (iv) a cooperation module to achieve multi-robot cooperation.

Each of the aforementioned modules represent a research problem by itself. Therefore, the design of an information gathering system should take into consideration the trade-off between the benefits of employing multiple robots respect to the additional effort required to design the system. This issue could be alleviated if there were more publicly available software. This way, researchers could focus on specific modules of a multi-robot system, without the need of coding each single module from scratch. We strongly believe that the release of the software under an open source license could advance dramatically the state-of-the-art in multi-robot exploration. In this respect, we expect to make the software developed in this thesis publicly available.

Centralized vs. decentralized architecture. We argued in this work that a decentralized architecture has countless advantages respect to a centralized architecture, being the robustness against the single-point-of-failure the most relevant one. Here, we do not focus on the advantages, as they were addressed in previous chapters, but rather on the open challenges that are associated to a decentralized architecture.

First, let us comment on the difficulties to deploy a decentralized system. Decentralized systems typically rely on communication as a mean for cooperation and coordination. In the market there are few communication systems that are lightweight to be mounted on a robot, have a sufficient data rate to transmit a high amount of data, and transmit up to a long range to cover large areas. These aspects clearly limit the applicability of the algorithms developed and tested in simulations to real world applications. For example, for the experiment that we carried out in Chapter 7, we employed WiFi as communication system. WiFi offers high data rates and antennas are lightweight. However, it has a short communication range, approximately in the order of 200 m, which limits the size of the exploration area.

Communication is the pillar of most decentralized exploration algorithms. However, by examining the literature we noticed that most works do not take into account the communication cost for the algorithm design. Essentially, state-of-the-art works focus on the reduction of the algorithm computational complexity. From our point of view, we, as a community, should start considering the communication aspect more seriously by defining a trade-off between algorithm complexity and communication cost. This will enable the development of future decentralized exploration algorithms.

To finalize, we would like to comment about the necessity of a decentralized ROS system. Although there exist decentralized multi-robot simulators (Nikolai and Madey, 2009), most roboticists employ ROS in their everyday work. Although ROS nodes are decentralized, ROS is by nature centralized as the roscore runs in a central machine. This implies that a ROS-based multi-robot system is technically centralized. From our point of view, we should put additional efforts to develop a decentralized ROS.

Constrained vs. non-constrained multi-robot algorithms. In Chapter 6 we introduced a first algorithm to handle inter-robot collision avoidance. Then we extended the algorithm in Chapter 7 to also incorporate network connectivity constraints.

The analysis of the proposed algorithms helped us to confirm that constraints typically degrade the algorithm's exploration performance. This can be explained by the fact that constraints reduce the exploration problem solution space. Therefore, as we increase the number of constraints, finding a feasible solution becomes more difficult.

There are two alternatives to handle inter-robot constraints. First, we can incorporate constraints into our problem formulation, and solve an exploration problem that is subject to constraints. This is the alternative that we chose in Chapters 6, 7. Second, we can separate exploration and constraints handling in two separate modules, where the last one would act reactively.

The second alternative is typically easier to implement, but it could lead to more deadlocks, compared to the first alternative, as constraints are not handled in advance. Nevertheless, it would allow robots to explore in situations where finding a solution to the full problem is costly. Therefore, we strongly believe that a robotic exploration system should consist on two steps. First, we should execute an algorithm that solves the full problem. Second, in case no solution could be found after an user-defined time, we should execute an algorithm to solve the single exploration problem, and solve constraints reactively. This will allow us to develop systems that can handle a higher number of complex constraints.

Simulation-based vs. experimental-based evaluation. The majority of the works in the multi-robot information gathering literature are solely evaluated in simulations. Moreover, most decentralized algorithms are simulated in a sequential manner in a single thread. That is, robots take actions sequentially, instead of having a single thread per robot with multiple robots threads running in parallel. We believe that such a simulation strategy is valid to illustrate the algorithm's performance. However, it is not valid to illustrate aspects such as inter-robot communication or robots asynchrony.

Those aspects could appear as irrelevant, but we noticed during the development of our algorithm in Chapter 7 that those aspects play a crucial role. They play a crucial role not only from an engineering perspective, but also for the algorithm design. Therefore, we decided to simulate our system in a decentralized fashion with robots running in separate threads. Such simulation approach caused us some difficulties to e.g. simulate a large fleet of robots. Nevertheless, we strongly believe that this is the only alternative to transfer our algorithms from paper to field experiments.

Field experiments were a great tool to identify practical difficulties that arise when deploying a multi-robot system in the real world. Specifically, we identified communication as the main challenge to transfer simulation results to real world applications. As we previously mentioned, there are few and expensive communication systems[1] in the market that could fulfill the demands of a large-scale real-world application. Therefore, we believe that the development of future communication systems will be key to advance multi-robot exploration technologies.

Final remarks. To finalize this section, we would like to add a final remark. During the development of this thesis it was difficult for us to find benchmark algorithms in the literature. Most of the algorithms are designed to solve a very specific problem. This implies that, in order to compare our algorithm with the state-of-the-art, we should solve tons of problems that are different from

[1]Examples of commercial systems are SC4200 from Silvus Technologies (`http://silvustechnologies.com/products/streamcaster-4200`), or MCU-30 from Mobilicom (`http://www.mobilicom.com/mcu-30-ruggedized`).

our objective, only to compare the algorithm's performance. Moreover, most algorithms are not publicly available, which represents an additional difficulty.

We firmly believe that the development of a library, like e.g. OMPL for path planning (Şucan et al., 2012), with multiple information gathering problems and algorithms, will help the community to benchmark algorithms and, subsequently, to advance our research.

8.2 Future Work

The development of this thesis helped us to identify possible research directions, which could advance the state-of-the-art in multi-robot exploration. Some of them are straightforward extensions of the work done in the thesis, while others rather focus on the multi-robot exploration problem from a general perspective. Next we would like to discuss these directions in more detail.

Process of interest. In this thesis we focused on spatially distributed processes that are time-invariant. Specifically, we focused on processes that can be described by a set of coordinates and a scalar value, like e.g. the magnetic field intensity. However, we could extend our framework to processes whose magnitude is a vector, like e.g. the magnetic field, which can be represented as a set of coordinates and a three dimensional vector. A common technique to model such processes is to describe each of the vector dimensions by an independent GP (Lawrance and Sukkarieh, 2011).

In addition, we would like to investigate spatially distributed processes that are time-variant. Such processes could be easily incorporated into our GPs framework by defining a suitable covariance function that models both the spatial and temporal correlation of the process. This would allow us to consider further processes like e.g. gas diffusion processes.

Process model. We employed in this work GPs to model a process of interest. GPs offer a high performance for a great number of applications. However, GPs fail to represent complex processes that cannot be easily modeled by a covariance function. An example of such a complex process could be an image. Here, alternative models should be considered, with deep neural networks being state-of-the-art. Nevertheless, for a deep neural network model, we should address the following question: how to compute an information metric, e.g. entropy or MI, from a deep neural network? To the best of our knowledge, this question is still to be answered.

In addition, we would like to investigate the use of Gaussian Markov Random Fields (Rue and Held, 2005), as they could offer interesting properties for decentralized estimation of a process.

Sensors. For our experiments, we employed three different sensors: a magnetic field sensor, an ultrasound sensor, and an anemometer (simulated). These sensors share the following properties: (i) they output a scalar value as sensor reading, (ii) their impulse response is essentially the actual measurement plus additive gaussian noise, and (iii) the noise level of the sensor is negligible compared to the process values.

We would like to further investigate how the algorithms proposed in this thesis could be adapted to account for more complex sensors, like e.g. a camera. If we could extract a feature from each camera picture, we could still employ the algorithm proposed here simply by assuming the feature as a measurement. However, if we wanted to describe pictures with multiple features we should think about different alternatives. An example could be to define a GP for each feature, and then combine all GPs into an exploration objective function. Alternative models to GPs could be used to model information from complex sensors. Examples of such models could be Markov random fields, or deep neural networks.

Constraints. We developed a system that can incorporate multiple complex constraints into the information gathering task. There is one constraint that we could not consider in Chapter 7. This is related to the extension of the work proposed in Chapter 7 to perform autonomous soaring. For autonomous soaring, we should be able to incorporate additional constraints such as aircraft energy, wind velocity, and minimum flying height. This is a venue for future work that we strongly believe could lead to the first system that performs autonomous soaring with multiple cooperative flying aircraft.

Experiments. In this work, we performed multiple experiments to validate our algorithms. Nevertheless, there are still experiments and applications that we believe could be tackled with the algorithms proposed in this work. Autonomous soaring is a possible application, as we previously mentioned. In addition, we have a particular interest in 3D map building (Suveg and Vosselman, 2004), and wildfires monitoring (Merino et al., 2006; Casbeer et al., 2005).

Data fusion. The algorithm proposed in Chapter 7 is decentralized, with the exception of the data fusion component that employs a broadcast mechanism. There exist methods to perform a decentralized data fusion for GPs, as pointed out in Chapter 7. However, those methods employ approximations to build a global GP covariance matrix. From our perspective, the study of data fusion algorithms to build an exact GP covariance matrix from measurements taken by individual robots is an extremely interesting venue for future developments.

Beyond spatially distributed processes. A natural extension of this work is tackling problems that go beyond spatially distributed processes. Examples of such problems are tracking (Stranders et al., 2010), coverage (Miller and Murphey, 2015), surveillance (Spaan et al., 2015), etc. These would require alternative models, and objective functions. Nevertheless, we could think about how to adapt our system for such applications. For example, for coverage we could define a covariance function that models the sensor footprint, and entropy to as objective function. This way, by reducing the entropy we would cover the area of interest. Also, for surveillance we could formulate the problem as a time-variant coverage problem. Nevertheless, we believe tracking could not be achieved with a straight-forward adaptation of our algorithm.

Beyond model-based exploration. Traditional exploration methods are based on the knowledge we have about the world. This is the strategy that we followed in this thesis. For example, although we learned the process spatial correlation online, we assumed a certain covariance function – a squared exponential function. Such a choice implies that we assume that our process of interest is smooth.

However, it is true that the world is too complex to be properly modeled. Moreover, there are applications, like e.g. extraterrestrial exploration, where we cannot model the world as we have no prior information about it. This opens the door towards machine learning model-free approaches. Specifically, we believe that Reinforcement Learning (RL) will be key in next years, and will allow us to solve novel and classic problems more efficiently.

RL is a machine learning paradigm that is model-free. In particular, it takes a world's state and a reward function that represents an specific goal. Then, by trial and error it learns how robots should behave in order to achieve the goal. RL could change how we understand exploration because it changes the question we should ask ourselves before developing an exploration algorithm. That is, we would go from the "how can I model the world? (and then hope that my model is correct to achieve the goal)" to the "what do I want to achieve? (and then let the robot figure out how to do it)".

Appendices

Appendix A
Single-Robot Path Planning

In this thesis we proposed several algorithms to solve information gathering tasks that are subject to constraints, like e.g. collision avoidance or communication constraints. The proposed algorithms rely on two main components: (i) a path planning method that calculates a range of trajectories that the robot could follow according to its motion model, and (ii) an information function, e.g. entropy or mutual information, that weights a trajectory according to its informativeness, measured by the information function. In Chapter 7 we treated the path planning and weighting of the trajectory with an information function as two separate components. That is, first robots generate a large amount of different feasible trajectories, and in a later step they select the ones that maximize mutual information for the team of robots. In contrast, Chapter 5 combines the two aforementioned components into one planning method that trades-off the path cost, defined by the robot's dynamics, and the path informativeness, as measured by an information function.

As we pointed out in the previous paragraph, path planning plays a crucial role in information gathering algorithms. As such, the work in the thesis led us to novel ideas that are purely related to path planning, which we strongly believe could enhance the state-of-the-art path planning methods. Therefore, in this appendix we slightly deviate from the original goal of this thesis – information gathering – and describe a contribution that is related to path planning. In particular, we propose a sampling-based path planning algorithm (ACO-RRT*) for a single robot.

We organize this appendix as follows: first, we introduce the problem we aim to solve and present a literature overview in Section A.1. Then we give in Section A.2 a short summary of one of the main tools that we employ to derive our path planning algorithm: Ant Colony Optimization (ACO). This is followed in Sections A.3 and A.4 by a detailed description of the proposed algorithm. In Section A.5 we evaluate the algorithm performance in simulations. We finalize with a summary and outlook of the chapter in Section A.6. Specifically, in this last section we give some insights about how the path planning algorithm proposed in this chapter could be extended for exploration tasks.

A.1 Introduction

Sampling-based path planning algorithms are widely used because of their efficiency to provide path planning solutions in high dimensional spaces, which permits the algorithms generalization to a large class of robots; as we pointed out in Chapters 5 and 7. Sampling-based path planning methods are well exemplified by probabilistic roadmaps (PRMs) (Kavraki et al., 1996) and RRT/RRT* algorithms (Section 2.2.2).

In recent years a great amount of sampling-based path planning algorithms have been proposed, such as (Arslan and Tsiotras, 2013, 2015; Gammell et al., 2014b,a; Karaman et al., 2011; Salzman and Halperin, 2014, 2015). These works have in common that they outperform the RRT* algorithm by modifying and optimizing some of the subroutines that compose the original RRT* algorithm. However, the cited algorithms are specifically designed to solve the optimal shortest path planning problem under certain restrictions. Here, we aim to go one step further and propose a framework that allows us to introduce some heuristics into the original RRT and RRT* algorithms. Specifically, we propose to incorporate heuristics into the algorithm by modifying the sampling distribution, which is learned online, according to the heuristics, as we sample the state space. The advantage of defining such heuristics is twofold: (i) it allows us to introduce some additional knowledge to solve the path planning problem more efficiently; (ii) it could be used in conjunction with any of the aforementioned works to improve their performance. Specifically, in this work we will show that our approach combined with shortest path heuristics outperforms RRT and RRT* algorithms.

Sampling-based path planners consist of several subroutines that can be optimized individually to improve the algorithm's performance, which also make the methods very attractive. Denny *et al.* proposed a "lazy planning" to improve the collision checking by the assumption that only 10% of the collisions checks are positive (Denny et al., 2013b). Furthermore, algorithms like RRT connect (Kuffner and LaValle, 2000) and the one proposed in (Urmson and Simmons, 2003) increase the algorithm performance by heuristically biasing the tree growth. The tree growth is also adapted by the authors of (Denny et al., 2013a), where they adapt the branch size according to the space in heterogeneous environments. In contrast to the previous works, here we focus on a modification of the sampling strategy. Essentially, if we can identify regions of higher importance, i.e. regions in the state space that could help us to improve our current path, then we should sample these regions more often.

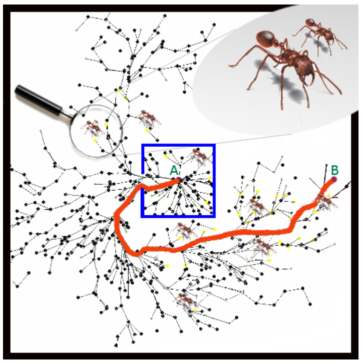


Figure A.1: Example of one path generated with the proposed ACO-RRT* algorithm. We build an RRT* based on a modified sampling strategy that learns from previous experience.

It is possible to dichotomize sampling strategies for path planning into importance sampling and adaptive sampling. Importance sampling methods exploit some pre-defined *a priori* sampling strategy. Examples include goal-biased sampling (Amato and Song, 2002), medial-axis sampling (Guibas et al., 1999), and the bridge test (Hsu et al., 2003) that is designed to solve narrow passages problems. These methods are specifically designed to solve concrete problems. Alternatively, in adaptive sampling methods, samples are drawn from a distribution that is adapted based on the information obtained from previous samples, which makes them more flexible. Siméon *et al.* propose the visibility PRM algorithm (Siméon et al., 2000) that just takes samples from the unexplored area within the planner visibility region. Although the constructed roadmaps are significantly smaller, the computation of the visibility region is expensive.

Adaptive dynamic domain RRT adapts the previous concept to the RRT algorithm (Jaillet et al., 2005). In this work, we additionally consider the importance of the previous samples that are not necessarily within the visibility region. This is exploited for PRMs through an utility-guided sampling in (Burns and Brock, 2005). There, the authors do not aim to learn the sampling distribution, but to perform a Monte-Carlo sampling and select the samples with a higher utility. However, our focus lies on RRT-based planners due to their efficiency, as they do not require any pre-computation time like in PRMs. Adaptive sampling within the RRTs framework has been also exploited in recent works (Jaillet et al., 2010; Janson et al., 2015; Kim et al., 2014; Nasir et al., 2013). In contrast to our proposed algorithm, these are not able neither to incorporate nor to learn arbitrary heuristics. Notice that the possibility of learning arbitrary heuristics is crucial to e.g. employ the proposed methods for information gathering tasks, where heuristics could drive a robot to highly informative locations.

Heuristics are incorporated into our algorithm via an utility function that weights the importance of samples. The definition of our utility function is inspired by the work from Rickert et al. (2014). In (Rickert et al., 2014), the authors propose the Exploring-Exploiting Tree (EET) algorithm that balances exploitation and exploration to construct the tree more effectively. Yet this method requires some environment dependent pre-computing time for growing the tree, which does not make it suitable for online planning. The exploration-exploitation trade-off has been also considered in several works, such as (Akgun and Stilman, 2011; Alterovitz et al., 2011; Persson and Sharf, 2014). In (Alterovitz et al., 2011), the authors propose the rapidly exploring roadmaps (RRMs). The algorithm finds first a solution like in RRTs and then refines this solution. Balancing exploration and exploitation is also employed in (Persson and Sharf, 2014). In (Persson and Sharf, 2014) the authors extend the A* algorithm to a sampling-based planner. Also in (Akgun and Stilman, 2011), the algorithm trades off exploration and exploitation to improve the RRT* in high dimensional spaces. This is done by introducing sampling heuristics. Our algorithm is also based on sampling heuristics that are learned using machine learning. In contrast to the aforementioned works (Akgun and Stilman, 2011; Alterovitz et al., 2011; Persson and Sharf, 2014), our framework allows us to introduce sampling heuristics that are not just specifically designed for the optimal shortest path planning but also for different applications, like e.g. information gathering.

Our goal in this appendix is not only to propose a framework that can incorporate user-defined heuristics. In addition, we aim to learn the sampling distribution of the planning algorithm that better fits to those heuristics. We realize that by introducing machine learning techniques. Machine learning was also employed in the path planners proposed in (Morales et al., 2004; Diankov and Kuffner, 2007), which require a discrete predefined set. In contrast, we aim to apply machine learning not to learn a discrete set but a continuous distribution.

The method presented here is strongly influenced by the idea of cross-entropy motion planning (Kobilarov, 2012). In (Kobilarov, 2012), the author learns the sampling distribution from previous samples by evaluating its entropy. Its limitation comes from the high computational requirement to calculate the sampling distribution for the environment, which does not make it feasible for real time applications. We improve this concept by using an ACO algorithm (Socha and Dorigo, 2008) to learn the sampling distribution. ACO has been previously used in the context of PRMs (Mohamad et al., 2006). In (Mohamad et al., 2006) their goal was to reduce the number of intermediate configurations from an initial to a goal position. Although with a different objective, that work serves us as inspiration to incorporate ACO into a sampling based path planner. Learning the

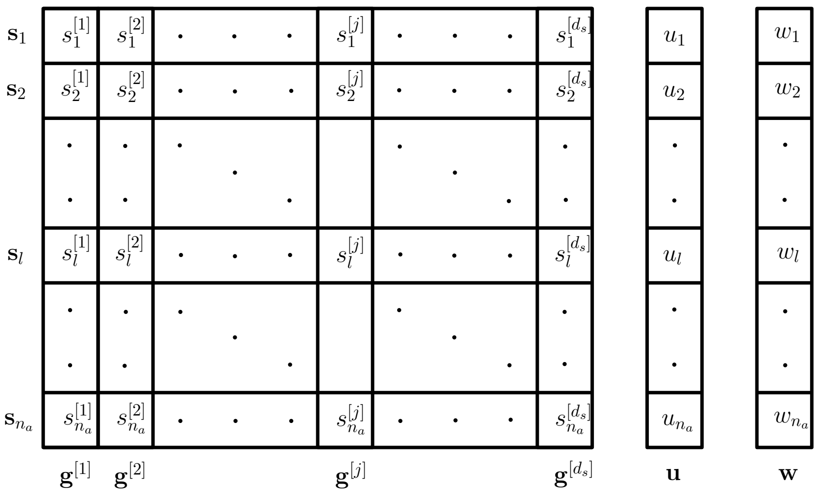

$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_l$
$\mathbf{s}_{n_a}$
$s_1^{[1]}$ $s_1^{[2]}$ $s_1^{[j]}$ $s_1^{[d_s]}$
$s_2^{[1]}$ $s_2^{[2]}$ $s_2^{[j]}$ $s_2^{[d_s]}$
$s_l^{[1]}$ $s_l^{[2]}$ $s_l^{[j]}$ $s_l^{[d_s]}$
$s_{n_a}^{[1]}$ $s_{n_a}^{[2]}$ $s_{n_a}^{[j]}$ $s_{n_a}^{[d_s]}$
u_1 u_2 u_l u_{n_a}
w_1 w_2 w_l w_{n_a}
$\mathbf{g}^{[1]}$ $\mathbf{g}^{[2]}$ $\mathbf{g}^{[j]}$ $\mathbf{g}^{[d_s]}$
$\mathbf{u}$
$\mathbf{w}$

Figure A.2: $\mathbf{T}$-table of the ACO$_{\mathbb{R}}$ algorithm.

sampling distribution, together with the definition of a novel utility function, let us derive an scalable algorithm suitable for real time path planning applications. To finalize, in Figure A.1 we present a motivating example that represents a path calculated with the proposed ACO-RRT* algorithm.

A.2 Ant Colony Optimization for Continuous Domains

Ant colony optimization is a nature-inspired algorithm to solve hard combinatorial optimization problems (Dorigo et al., 1996). Its driving principle comes from the behavior of ants when searching for food. That is, ants leave the nest walking in random directions. Once they find a food source they come back to the nest leaving a pheromone trail on the ground. The pheromone deposited depends on the quality and quantity of the food and guide the other ants to the food source. Based on the same principle, ant colony optimization for continuous domains (ACO$_{\mathbb{R}}$) is proposed to solve continuous optimization problems (Socha and Dorigo, 2008). This work inspires our sampling strategy, in which ants, according to their utility, will decide where to sample next.

In (Socha and Dorigo, 2008) ants are represented by a table $\mathbf{T}$ as depicted in Figure A.2. Each row contains one of the n_a ants, where $\mathbf{s}_l = [s_l^{[j]}]_{j=1,\dots,d_s}$ is the vector of coordinates describing the l^{th} ant's location. The ant's utility is given by u_l, which determines the importance of the l^{th} ant. The utility is defined according to the algorithm's optimization objective.

The algorithm proposed in (Socha and Dorigo, 2008) works as follows. First,

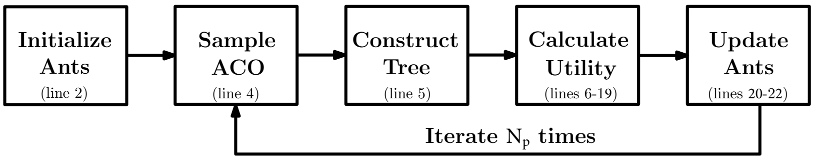


Figure A.3: ACO-RRT/RRT* algorithm block diagram. Each of the five blocks points to its respective lines from Algorithm A.1.

it takes a sample $\mathbf{x}_{rand} = [x_{rand}^{[j]}]_{j=1,\ldots,d_s}$, with $x_{rand}^{[j]}$ drawn from a Gaussian mixture model probability distribution:

$$G^{[j]}(x) = \sum_{l=1}^{n_a} w_l g_l^{[j]}(x) = \sum_{l=1}^{n_a} w_l \frac{1}{\sigma_l^{[j]}\sqrt{2\pi}} e^{-\frac{(x-s_l^{[j]})^2}{2\sigma_l^{[j]2}}}, \tag{A.1}$$

with w_l the l^{th} ant's weight, and $\sigma_l^{[j]}$ the l^{th} ant's standard deviation in dimension j. $\sigma_l^{[j]}$ is calculated as the average distance from the l^{th} ant to the rest of the ants stored in $\mathbf{T}$:

$$\sigma_l^{[j]} = \xi \sum_{e=1}^{n_a} \frac{|s_e^{[j]} - s_l^{[j]}|}{n_a - 1}, \tag{A.2}$$

where $\xi > 0$ is the pheromone evaporation rate, which avoids that the algorithm converges too fast before approaching the optimal solution. The weight w_l from vector of weights $\mathbf{w}$ is set as:

$$w_l = \frac{1}{qn_a\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2 n_a{}^2}}, \tag{A.3}$$

where q is a user-defined parameter. When q is small, the best ranked solutions are strongly preferred. $\mathbf{w}$ is normalized so that the integral of $G^{[j]}(x)$ over the entire space is equal to one. The value of w_l is initialized and is not modified during the execution of the algorithm.

Next, ACO sorts $\mathbf{T}$ in descending order according to $\mathbf{u}$, and inserts $\mathbf{x}_{rand}$. $\mathbf{x}_{rand}$ will become now an ant. In this way, samples with a higher utility will go up in the table and will be selected with a higher probability. In case the sample's utility is higher than the last ranked solution $\mathbf{s}_{n_a}$, $\mathbf{s}_{n_a}$ will be removed from $\mathbf{T}$ to keep n_a ants in $\mathbf{T}$.

A.3 ACO-RRT/RRT* Algorithm

We introduce here the ACO-RRT/RRT* algorithm. This consists of five steps that are depicted in a block diagram in Figure A.3. ACO-RRT/RRT* works

Algorithm A.1. ACO-RRT/RRT*$(\mathbf{x}_A, \mathbf{x}_B, N_p, n_a, \check{\alpha}, \hat{\alpha}, \mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{X}_{free})$

```
1: $\mathcal{V} \leftarrow \{\mathbf{x}_A\}$; $\mathcal{E} \leftarrow \emptyset$; $\mathbf{T} \leftarrow \emptyset$; $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \leftarrow \emptyset$; $c_{path} \leftarrow \infty$;
2: $\mathbf{T} \leftarrow$ InitializeAnts$(n_a, \mathbf{T}, \mathcal{X}_{free})$;    ▷ according to Sec. A.3.1
3: for $i = 1, \dots, N_p$ do
4:     $\mathbf{x}_{rand}, l \leftarrow$ SampleACO$(\mathbf{T})$;    ▷ according to Sec. A.3.2
5:     $\mathbf{x}_{new}, \mathcal{G}(\mathcal{V}, \mathcal{E}) \leftarrow$ ConstructTree$(\mathbf{x}_{rand}, \mathcal{G}, \mathcal{X}_{free})$;    ▷ according to Sec. A.3.3
6:     ▷ Calculate utility according to Sec. A.3.4
7:     $U_{explore} \leftarrow U_{explore}(\mathbf{x}_{new}, \mathcal{G})$;    ▷ with (A.6)
8:     if $c_{path} \neq \infty$ then    ▷ path found
9:         if $c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_{new}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x}_{new},\mathbf{x}_B}) < c_{path}$ then    ▷ if improves the current solution
10:            $\alpha \leftarrow \check{\alpha}$;
11:            if $c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G})) < c_{path}$ then    ▷ path improvement
12:                $U_{exploit} \leftarrow \dot{\check{U}}_{exploit}(\mathbf{x}_{new}, \mathcal{G})$;    ▷ with (A.8)
13:            else    ▷ no path improvement
14:                $U_{exploit} \leftarrow \bar{\check{U}}_{exploit}(\mathbf{x}_{new}, \mathcal{G})$;    ▷ with (A.9)
15:        else
16:            $\alpha \leftarrow NULL, U_{exploit} \leftarrow NULL, U_{explore} \leftarrow NULL$;
17:    else    ▷ no path found
18:        $\alpha \leftarrow \hat{\alpha}$;
19:        $U_{exploit} \leftarrow \hat{U}_{exploit}(\mathbf{x}_{new}, \mathcal{G})$;    ▷ with (A.7)
20:    $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \leftarrow$ FindBestPath$(\mathbf{x}_A, \mathbf{x}_B, \mathcal{G})$;
21:    $\mathcal{I} \leftarrow \{U_{exploit}, U_{explore}, \mathbf{x}_{new}, \mathbf{x}_B, \alpha, l, \mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}, c_{path}\}$;
22:    $\mathbf{T} \leftarrow$ UpdateAnts$(\mathbf{T}, \mathcal{G}, \mathcal{I})$;    ▷ according to Sec. A.3.5
23:    if $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \neq \emptyset$ then    ▷ if we found a path
24:        $c_{path} \leftarrow c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G}))$;
25: return $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$;
```

as follows: first, we initialize $\mathbf{T}$ with an initial set of ants (Section A.3.1). The initialization is only carried out once at the beginning of the algorithm's execution. Then we obtain a sample $\mathbf{x}_{rand}$ from the distribution defined by $\mathbf{T}$ (Section A.3.2), and incorporate $\mathbf{x}_{rand}$ into a tree built with an RRT/RRT* algorithm (Section A.3.3). Next we calculate the utility of $\mathbf{x}_{rand}$. We consider two factors to calculate the utility: (i) exploitation of the current solution, and (ii) exploration of the state space to find a new, better solution (Section A.3.4). According to the calculated utility, we update $\mathbf{T}$ (Section A.3.5). This loops continues during N_p iterations. We formulate the ACO-RRT/RRT* in Alg. A.1. In the following sections we describe each of the algorithm steps in more detail.

A.3.1 Initialize Ants

First we fill $\mathbf{T}$ with n_a ants, as described in Alg. A.2. Alg. A.2 works as follows: we take a sample $\mathbf{x}_{rand}$ from an uniform distribution defined over $\mathcal{X}_{free}$ (line 2, Alg. A.2). Then we insert $\mathbf{x}_{rand}$ in row l that represents the l^{th} ant (lines 3-4, Alg. A.2). We initialize utility u_l to zero. For the calculation of the utility we require the exploitation and exploration utility as well as a parameter α that trades off the two factors (see Section A.3.4). These three elements $(\mathcal{F}_l.U_{exploit}, \mathcal{F}_l.U_{explore}, \mathcal{F}_l.\alpha)$[1] are stored in set $\mathcal{F}_l$, and initialized to zero. Parameter w_l is computed according to (A.3) (lines 5-6, Alg. A.2).

Algorithm A.2. `InitializeAnts`$(n_a, \mathbf{T}, \mathcal{X}_{free})$

```
1: for $l = 1, \dots, n_a$ do                                          ▷ for each ant
2:     $\mathbf{x}_{rand} \leftarrow$ SampleFree$(\mathcal{X}_{free})$;
3:     for $j = 1, \dots, d_s$ do                                      ▷ for each dimension
4:         $\mathbf{T}.s_l^{[j]} \leftarrow x_{rand}^{[j]}$;
5:     $\mathbf{T}.\mathcal{F}_l.\alpha \leftarrow NULL; \mathbf{T}.u_l \leftarrow NULL; \mathbf{T}.w_l \leftarrow w_l$;
6:     $\mathbf{T}.\mathcal{F}_l.U_{exploit} \leftarrow NULL; \mathbf{T}.\mathcal{F}_l.U_{explore} \leftarrow NULL$;
7: return $\mathbf{T}$;
```

A.3.2 Sample ACO

Once we have initialized $\mathbf{T}$, we draw a new sample $\mathbf{x}_{rand}$ (line 4, Alg. A.1). To this end, we employ the following procedure, which is equivalent to sampling directly from (A.1). First, we select an ant l with a probability:

$$p_l = \frac{w_l}{\sum_{e=1}^{n_a} w_e}. \tag{A.4}$$

The selected ant determines $\mathbf{x}_{rand}$, with $x_{rand}^{[j]} \sim \mathcal{N}(s_l^{[j]}, {\sigma_l^{[j]}}^2)$. To generate a $x_{rand}^{[j]} \in \mathcal{X}_{free}$, we employ rejection sampling. In addition to $\mathbf{x}_{rand}$, `SampleACO` outputs the selected ant index l.

Let us note that the modification of the RRT* sampling strategy does not sacrifice the asymptotic optimality guarantee of RRT* (Karaman and Frazzoli, 2011). Asymptotic optimality of ACO-RRT* lies on the fact that ants are associated to a Gaussian probability density function. Samples extracted from such a function can take values from an infinite domain, which results in sampling over the complete state space. Even in the worst case, with all ants converging to a single point, $\sigma_l^{[j]}$ will be slightly greater than zero. This fact guarantees that the state space will be fully sampled and, therefore, ACO-RRT* will approach the optimal solution.

[1] $\mathcal{A}.b$ denotes element b that is part of set $\mathcal{A}$.

A.3.3 Construct Tree

Next we construct a tree according to the RRT/RRT* path planner. This step corresponds to lines 4-7 in Algorithm 2.1 (RRT) and lines 4-21 in Algorithm 2.2 (RRT*). `ConstructTree` takes as input $\mathbf{x}_{rand}$ and current tree $\mathcal{G}$, and outputs the vertex $\mathbf{x}_{new}$ that was added to the tree, as well as the new constructed tree.

A.3.4 Calculate Utility

The key part of Alg. A.1 is the calculation of the $\mathbf{x}_{new}$ utility, which corresponds to lines 6-19 in Alg. A.1. Here we define the utility of a sample according to an exploitation-exploration trade-off. On the one hand, exploitation aims: (i) to reach $\mathbf{x}_B$ by following the shortest possible path, in case any path was found, and (ii) to improve the path to $\mathbf{x}_B$, once a path to $\mathbf{x}_B$ was found. On the other hand, exploration aims to sample at locations that have not been previously sampled. Exploration is necessary to: (i) find a first path to $\mathbf{x}_B$ by exploring the state space, and (ii) search for new better paths, once a path was found.

We calculate the utility $U(\mathbf{x}, \mathcal{G})$ of sample $\mathbf{x}$ given a tree $\mathcal{G}$ as follows:

$$U(\mathbf{x}, \mathcal{G}) = \alpha \cdot U_{exploit}(\mathbf{x}, \mathcal{G}) + (1 - \alpha) \cdot U_{explore}(\mathbf{x}, \mathcal{G}), \tag{A.5}$$

where α denotes the exploitation-exploration trade-off, $U_{exploit}(\mathbf{x}, \mathcal{G})$ is the exploitation utility, and $U_{explore}(\mathbf{x}, \mathcal{G})$ is the exploration utility. Next we describe each of the elements of (A.5) in detail.

A.3.4.1 Exploration Utility

We define $U_{explore}(\mathbf{x}, \mathcal{G})$ as a measure of the density of samples in $\mathcal{G}$ in the vicinity of sample $\mathbf{x}$. Formally, we calculate $U_{explore}(\cdot)$ as follows:

$$U_{explore}(\mathbf{x}, \mathcal{G}) = \frac{1}{|\mathcal{X}_{near}|} \left(\frac{R}{\eta}\right)^{d_s}, \tag{A.6}$$

with $\mathcal{X}_{near} \leftarrow \mathtt{Near}(\mathbf{x}, \mathcal{V})$ the set of neighbors of $\mathbf{x}$ given by (2.9), η a parameter of the RRT/RRT* path planner (Section 2.2.2), and R the connection radius. We define $R = r(|\mathcal{V}|)$ for RRT* with $r(\cdot)$ given by (2.10), and $R = \eta$ for RRT.

The first term of the product models a proportional decay of $U_{explore}(\cdot)$ with respect to $|\mathcal{X}_{near}|$. This implies that a sample that has a low number of neighbors in $\mathcal{G}$ will have a high exploration utility, which will bias the exploration towards the not-yet-sampled state space. However, $|\mathcal{X}_{near}|$ is strongly related to R. The bigger is R, the higher the probability to have a large $|\mathcal{X}_{near}|$. In RRT*, as the tree growths, R decreases. To make $U_{explore}(\cdot)$ independent of the tree's current state, we introduce a second term $(R/\eta)^{d_s}$ to act as a normalization factor.

A.3.4.2 Exploitation Utility

The exploitation utility aims to benefit from the knowledge acquired about the state space. Here, we distinguish two situations: utility before a first path was found, and utility after a first path was found. In order to add more flexibility to the algorithm, we assign to α, that was defined in (A.5), one of the two possible values: (a) $\alpha = \hat{\alpha}$ if no path was found ; (b) $\alpha = \check{\alpha}$ otherwise.

No path found. Before a first path is found, we bias the sampling to connect $\mathbf{x}$ with $\mathbf{x}_B$ as fast as possible regardless the obstacles (Amato and Song, 2002). We realize this by defining the utility as:

$$\hat{U}_{exploit}(\mathbf{x}, \mathcal{G}) = 1 - \frac{c(\mathcal{P}_{\mathbf{x},\mathbf{x}_B})}{c_{max}}, \tag{A.7}$$

where $c(\mathcal{P}_{\mathbf{x},\mathbf{x}_B})$ is normalized by the maximum cost c_{max} to reach $\mathbf{x}_B$ from any $\mathbf{x}' \in \mathcal{X}_{free}$. We point out that sampling at $\mathbf{x}_B$ has maximum utility since it directs the tree growth towards the goal position.

Path found. Once a first path is found, we exploit that information to derive a richer exploitation utility. We consider two possible situations: path improvement (see Figure A.4a), and no path improvement (see Figure A.4b).

Consider a sample $\mathbf{x}$ that leads to an improvement on $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G})$. In such situation, we could expect that the region of the state space around $\mathbf{x}$ could help us to improve the solution again in a future. Therefore, we formulate an exploitation utility that quantifies this improvement:

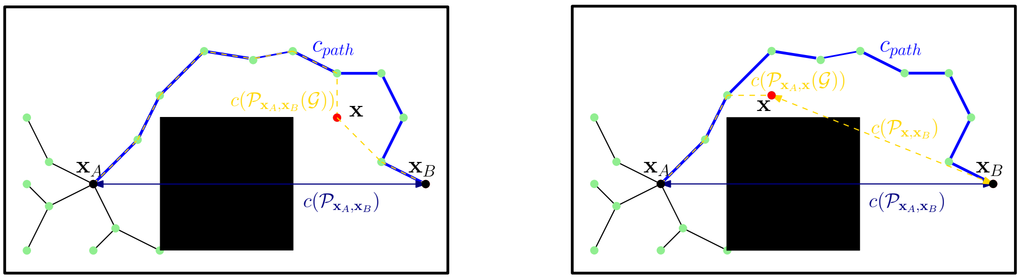
$$\dot{\check{U}}_{exploit}(\mathbf{x}, \mathcal{G}) = \frac{c_{path} - c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G}))}{c_{path} - c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B})}, \tag{A.8}$$

with $c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G}))$ the cost of the best path after sampling $\mathbf{x}$, and c_{path} the cost of the previous best path. The denominator normalizes the function so that it ranges between 0 (no path improvement) and 1 (best possible path).

In contrast, if sample $\mathbf{x}$ does not contribute to improve $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G})$, we define an exploitation utility that shapes the path as a straight line connecting $\mathbf{x}_A, \mathbf{x}_B$. This represents the best possible path regardless of the obstacles. In this case, the utility is given by:

$$\bar{\check{U}}_{exploit}(\mathbf{x}, \mathcal{G}) = \frac{c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G}))}{c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{x},\mathbf{x}_B})}. \tag{A.9}$$

It is important to remark that, once we find a first path, we only introduce the ant in $\mathbf{T}$ if it could improve the current solution (line 9, Alg. A.1). Otherwise


c_{path}
$c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G}))$
$\mathbf{x}$
$\mathbf{x}_A$
$\mathbf{x}_B$
$c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B})$
c_{path}
$c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}}(\mathcal{G}))$
$\mathbf{x}$
$c(\mathcal{P}_{\mathbf{x},\mathbf{x}_B})$
$\mathbf{x}_A$
$\mathbf{x}_B$
$c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B})$

(a) Path improvement.

(b) No path improvement.

Figure A.4: Graphical representation of the exploitation utility in the path found mode. A black square represents an obstacle. A red dot corresponds to $\mathbf{x}$. Black lines and green dots represent $\mathcal{G}$. The superposed thick blue line is the best found path before sampling $\mathbf{x}$. The dashed yellow line is the new best path after sampling $\mathbf{x}$. We represent with arrows the direct path between a state and $\mathbf{x}_B$.

we set the exploration and exploitation utilities to zero (line 16, Alg. A.1), which implies that the ant will be discarded once we update the table of ants. By doing that, we allow the algorithm to sample in the future again in a promising region, which could incur in a path improvement.

A.3.5 Update Ants

The last step of Alg. A.1 is the update of the ants, stored in $\mathbf{T}$, according to the updated utilities. The update is realized with Alg. A.3. This works as follows. The first time we find a path we reset the utility values $\mathbf{T}.\mathbf{u}$ and parameters $\mathbf{T}.\mathcal{F}$ to incorporate the utilities that are associated to that path (lines 2-3, Alg. A.3).

Next we update the l^{th} ant that drew sample $\mathbf{x}_{new}$. This update allows us to incorporate the current information provided by $\mathcal{G}$ (lines 5-13, Alg. A.3). For example, let us imagine an ant that had a great exploration utility when it was stored, but several iterations later the area associated to that ant is fully explored. In such a situation an update of the ants is crucial. Specifically, we realize this by introducing a soft pruning condition that allows the algorithm to shape the sampling distribution according to the most promising areas given the current knowledge about the state space (line 8, Alg. A.3). Then we insert the l^{th} ant in $\mathbf{T}$ according to the updated utility u_l, calculated with (A.5) from the elements of $\mathcal{F}_l$.

Once we update the l^{th} ant, we generate a new ant to incorporate the information provided by $\mathbf{x}_{new}$. We insert this ant in $\mathbf{T}$ at position i given by the ants utility u_i, calculated from $\mathcal{F}_i$ (lines 14-17, Alg. A.3). To incorporate the i^{th} ant we make a simplification that consists of two heuristics: (i) the utility of the i^{th}

Algorithm A.3. `UpdateAnts`$(\mathbf{T}, \mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{I})$

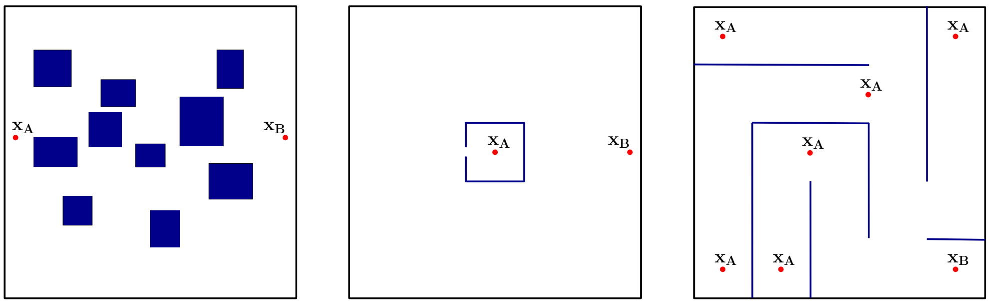
```
1: $\{U_{exploit}, U_{explore}, \mathbf{x}_{new}, \mathbf{x}_B, \alpha, l, \mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}, c_{path}\} \leftarrow \mathcal{I}$;
2: if $(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \neq \emptyset)$ and $(c_{path} = \infty)$ then        ▷ first path found
3:     $\mathbf{T}.\mathbf{u} \leftarrow NULL; \mathbf{T}.\mathcal{F} \leftarrow NULL$;
4: else
5:     $\mathcal{F}_l.U_{explore} \leftarrow U_{explore}$;
6:     $\mathbf{s}_l \leftarrow \mathbf{T}.\mathbf{s}_l; \mathcal{F}_l.\alpha \leftarrow \mathbf{T}.\mathcal{F}_l.\alpha; \mathcal{F}_l.U_{exploit} \leftarrow \mathbf{T}.\mathcal{F}_l.U_{exploit}$;
7:     if $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B} \neq \emptyset$ then        ▷ soft pruning condition
8:         if $c(\mathcal{P}_{\mathbf{x}_A,\mathbf{s}_l}(\mathcal{G})) + c(\mathcal{P}_{\mathbf{s}_l,\mathbf{x}_B}) > c(\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G}))$ then        ▷ soft pruning condition
9:             $\mathcal{F}_l.U_{exploit} \leftarrow NULL; \mathcal{F}_l.U_{explore} \leftarrow NULL$;
10:    $u_l \leftarrow$ CalculateUtility$(\mathcal{F}_l)$;        ▷ with (A.5)
11:    ▷ Update the $l^{th}$ ant
12:    $\mathbf{T} \leftarrow$ Extract$(l, \mathbf{T})$;
13:    $\mathbf{T} \leftarrow$ Insert$(\mathbf{s}_l, u_l, \mathcal{F}_l, \mathbf{T})$;
14:    ▷ Introduce $\mathbf{x}_{new}$ in the table
15:    $\mathcal{F}_i.\alpha \leftarrow \alpha; \mathcal{F}_i.U_{exploit} \leftarrow U_{exploit}; \mathcal{F}_i.U_{explore} \leftarrow U_{explore}$;
16:    $u_i \leftarrow$ CalculateUtility$(\mathcal{F}_i)$;        ▷ with (A.5)
17:    $\mathbf{T} \leftarrow$ Insert$(\mathbf{x}_{new}, u_i, \mathcal{F}_i, \mathbf{T})$;
18:    $\mathbf{T} \leftarrow$ Extract$(''last'', \mathbf{T})$;        ▷ extract last ant of the table
19: return $\mathbf{T}$;
```

ant is the same as the utility from the l^{th} ant; and (ii) the introduction of the i^{th} ant into the table does not incur in a modification of the exploration utility of the rest of the ants contained in $\mathbf{T}$. These two heuristics allow us to reduce the algorithm's computational complexity as we do not have to recalculate all utilities each time a new ant is introduced in $\mathbf{T}$. Despite this simplification, these heuristics have been shown to work well in practice since, next time an ant is selected, its utility will be recalculated. The last step of Alg. A.3 is the removal of the last row of $\mathbf{T}$ after a new ant is added (line 18, Alg. A.3). This is needed to keep n_a ants in $\mathbf{T}$.

A.4 Anytime ACO-RRT*

The main drawback of the ACO-RRT/RRT* algorithm is that it needs more time to find a first path compared to RRT/RRT*. This is due to the convergence time of the ants. However, the solution obtained by ACO-RRT/RRT* has a better quality; i.e. a smaller cost (as we will see in Sec. A.5).

There are situations, e.g. in exploration tasks or search and rescue missions, where finding a first solution rapidly is crucial. Then, if we had more time, we could improve it to reach our goal faster. Inspired by Ferguson and Stentz (2006), we exploit this concept in our Anytime ACO-RRT* algorithm. Anytime



(a) Scenario 1. (b) Scenario 2. (c) Scenario 3.

Figure A.5: Simulation scenarios. We aim to find the optimal path $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$. In A.5c we consider different $\mathbf{x}_A$ for each of the simulation runs.

ACO-RRT* works as follows. First, we run the fastest algorithm (RRT) to find a first solution $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G})$. Second, we initialize $\mathcal{G}$ as $\mathcal{G}(\mathcal{V},\mathcal{E}) = \mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G})$. Then we improve $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}(\mathcal{G})$ using ACO-RRT* taking $\mathcal{G}$ as input. This mechanism allows us to combine the best of both algorithms to increase the algorithm's performance.

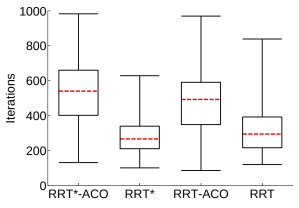
A.5 Simulations and Discussion of Results

We evaluate in simulations the ACO-RRT/RRT* algorithm performance to find the optimal $\mathcal{P}_{\mathbf{x}_A,\mathbf{x}_B}$. We employ an holonomic robot, since it allows us to abstract the algorithm capabilities from the robot's kinodynamic constraints. We assume the robot corresponds to a single point. However, more complex robot shapes could be easily introduced within this framework.

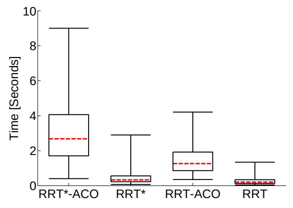
We consider three scenarios that correspond to realistic scenarios that could be encountered while navigating in an indoor facility (see Figure A.5). Similar scenarios have been considered to evaluate some of the most recent state-of-the-art methods, such as (Gammell et al., 2014b; Kim et al., 2014). Scenario 1 is composed of 10 rectangles of different size and the optimal path measures 88 meters. Scenario 2 contains a narrow passage that is a often considered as one of the most challenging path planning problems. The optimal path in this scenario is 63 meters. Scenario 3 corresponds to a maze-like environment. This last one allows us to test the algorithm performance in a more structured scenario. Since scenario 3 is more structured, the placement of the initial position plays a crucial role. Therefore, in scenario 3 we consider several $\mathbf{x}_A$ that are randomly selected in each simulation run. For the evaluation in the three scenarios we consider a goal region centered around $\mathbf{x}_B$, not just a single state. The three scenarios measure 100×100 meters.

N_p [s]	η [m]	n_a [ants]	q	ξ	$\check{\alpha}$	$\hat{\alpha}$
220	5	100	50	0.4	0.3	0.1

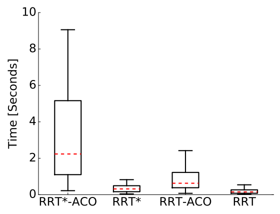
Table A.1: Simulation parameters.



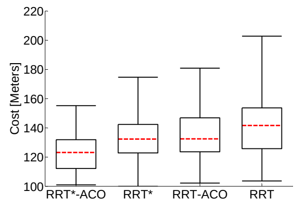
(a) Iterations to find first path.



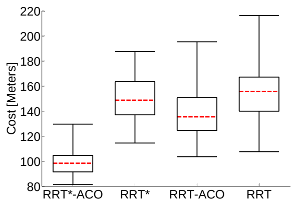
(b) Time to find first path.



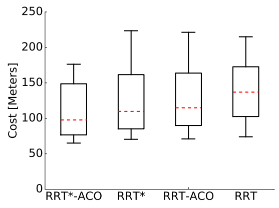
(c) Time to find first path.



(d) Cost of first path.



(e) Cost of first path.



(f) Cost of first path.

Figure A.6: Box plot representation of the number of iterations and time to find a first path, and its associated cost. (a,d) Scenario 1. (b,e) Scenario 2. (c,f) Scenario 3.

We carry out the simulations using a Intel Xenon E31225 processor at 3.10GHz with 8GB of RAM. We run each of the simulations 100 times according to the parameters shown in Table A.1.

We evaluate the following parameters: (i) time to find a first path and its associated cost; (ii) evolution of the cost of the best path found over time; (iii) performance of the anytime implementation; (iv) influence of the different parameters in the algorithm's performance.

A.5.1 Time to Find a First Path and Associated Cost

One of the key figures to evaluate the performance of Alg. A.1 is the number of iterations needed to find a first path. This is strongly correlated with the cost associated to that path. We evaluate both indicators for scenarios 1, 2 and 3. For scenarios 2 and 3, we represent the time instead of the number of iterations in order to proof the algorithm's performance in an actual system. We compare the ACO-RRT/RRT* algorithm with RRT and RRT* algorithms. We remark that we do not perform a comparison against the state-of-the-art works cited in Sec. A.1 because they follow a different goal. Their goal consists of approaching the optimal solution to the path planning problem as quickly as possible. On the contrary, our objective is to demonstrate that Alg. A.1 is able to learn user pre-defined heuristics, and used them to improve the solution of RRT/RRT* algorithms.

In Figure A.6, we employ a box plot representation of the obtained results, where the dashed red line is the median of the data, the bottom and top of each box represent the 25*th* and 75*th* percentile, and the two strokes encompass the minimum and maximum values. We observe that the ACO-RRT* algorithm finds a better solution when compared to the rest of alternatives. On the other hand, the ACO-RRT* algorithm is slower because ACO requires an additional time to place ants in the best positions to guide the tree's growth. During the first iterations of the algorithm, the ants are not correctly placed and therefore the planner cannot find a path between the initial and goal position.

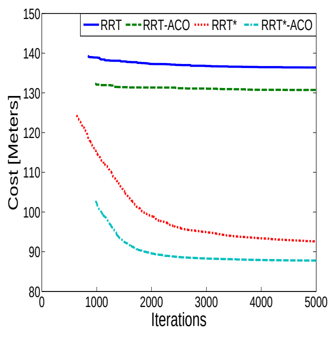
We can also conclude that RRT is the fastest algorithm to find a first solution to the path planning problem, although this solution has the highest cost compared to the other alternatives. We exploit this capability in our anytime implementation to find rapidly a first solution.

A.5.2 Algorithm Performance with Time

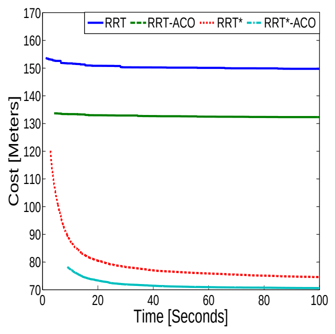
Once we find a first path, we aim to improve it to reach the optimal solution. Figure A.7 shows the evolution of the best path found over the number of iterations and time. We accompany these figures with a simulation of the algorithm's complexity for the three scenarios. Results of the proposed ACO-RRT/RRT* algorithm are compared with RRT* and RRT algorithms.

The curves correspond to the mean value calculated over 100 runs. For each of the curves in Figures A.7a, A.7b, A.7c, we considered the worst case; i.e. each of the curves starts when a path was found in all the 100 runs. We observe that the ACO-RRT* algorithm offers a superior performance over time. However, for scenario 3 the performance is similar to the one offered by RRT*. This is because scenario 3 is more structured and therefore, once the algorithm finds a first solution, it has little room for improvement. We remark that the introduction of the ACO increases the algorithm's computational complexity. This is mainly because of the time needed to compute the ant's utility and update $\mathbf{T}$. However, this additional complexity is beneficial since the ACO-RRT/RRT* offers a better performance.

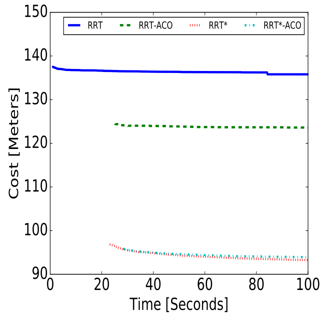
These results naturally lead us to formulate the anytime ACO-RRT* algo-



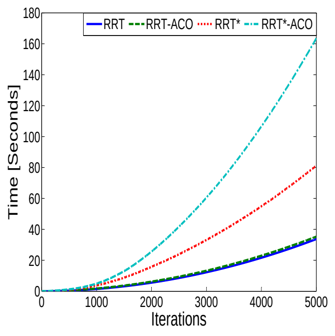
(a) Path Cost vs. Iterations.



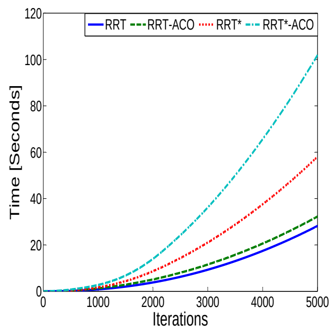
(b) Path Cost vs. Time.



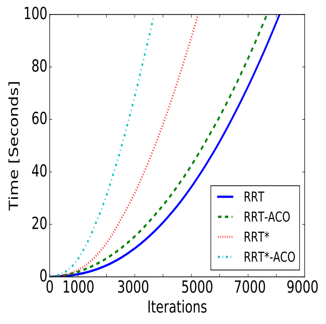
(c) Path Cost vs. Time.



(d) Time vs. Iterations.



(e) Time vs. Iterations.



(f) Time vs. Iterations.

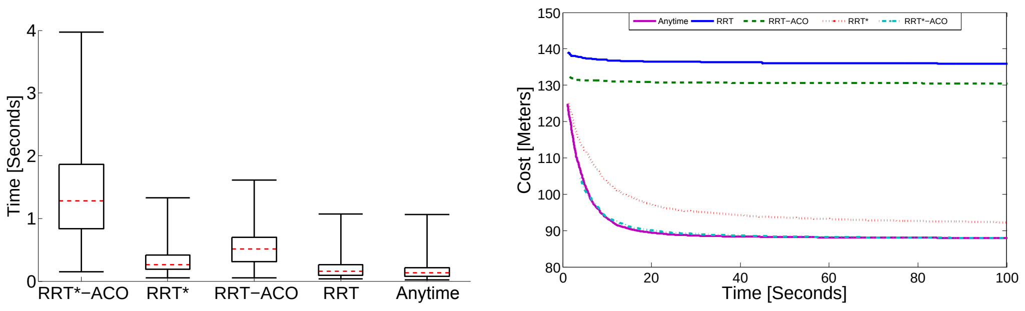
Figure A.7: Top: evolution of the best path cost over time and iterations once we found a first solution. Down: evaluation of the algorithm time complexity. (a,d) Scenario 1. (b,e) Scenario 2. (c,f) Scenario 3.

rithm. Although the solution offered by the RRT algorithm in a first place is of worse quality, we expect to improve it using ACO-RRT*. We expect this combination incurs in an increase of performance over time.

A.5.3 Anytime ACO-RRT* Performance

We show in Figure A.8 the performance of the anytime implementation of the algorithm. We would assume that the RRT and anytime curves should start at the same position, since both of them start running the RRT algorithm. However, here we represent the first moment in which we found a path for all the 100 algorithm runs. They would then start at the same point if the number of runs approaches infinity.

Anytime ACO-RRT* is the fastest algorithm to find a first path (equal to RRT) and has the same evolution of the performance over time (equal as ACO-



(a) Time to find first path. (b) Path Cost vs. Time.

Figure A.8: Anytime ACO-RRT* performance for scenario 1. (a) Box plot representation of the time to find a first path. (b) Evolution of the best path cost over time once we found a first solution.

RRT*).

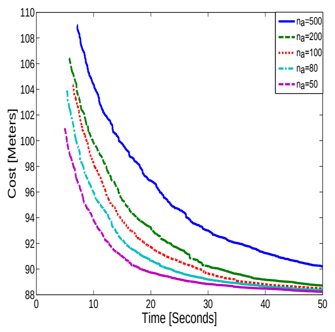
A.5.4 Performance with respect to the Algorithm Parameters

For scenario 1, we evaluate the evolution of the path cost over time with respect to the number of ants, the exploitation-exploration trade-off parameter, and the evaporation rate; while keeping the not-analyzed parameters constant according to Table A.1. Let us also remark that we do not simulate the influence of varying q since this is strongly correlated with the number of ants n_a. This allows us to keep it fixed and just modify n_a.

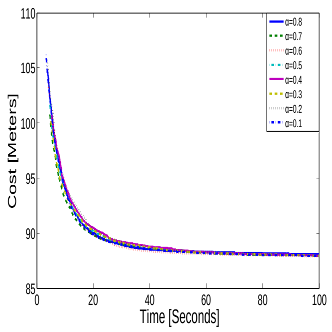
Figure A.9a shows the performance with respect to n_a. We performed as well simulations with a smaller n_a and the algorithm was not able to converge to any solution given the planning time. We observe that 50 ants corresponds to the best solution. Increasing n_a incurs however in a decrease of performance. The explanation of such behaviour comes from the trade-off that exists between including more ants to better learn the sampling distribution, and the complexity added at the sampling procedure when increasing n_a.

In order to analyze the impact of the α factor in the algorithm performance over time, we keep constant $\hat{\alpha}$ and vary $\breve{\alpha}$ between 0 and 1 (see Figure A.9b). As we could expect, for extreme values of $\breve{\alpha}$ the algorithm does not find a solution. For the rest of the values the performance varies only slightly.

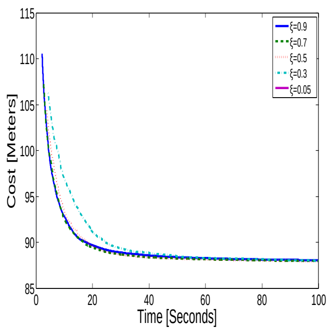
In Figure A.9c, we observe as well that the algorithm does not converge only for the extreme values of the convergence rate ξ. As in the previous case, it is important that performance does not drastically change as we vary this parameter.



(a) Evolution with n_a.



(b) Evolution with $\breve{\alpha}$.



(c) Evolution with ξ.

Figure A.9: Analysis of the algorithm performance for scenario 1 respect to: (a) number of ants, n_a; (b) exploitation-exploration trade-off parameter once we have found a first path, $\breve{\alpha}$; (c) evaporation rate, ξ.

A.5.5 Examples of Paths Planned with the ACO-RRT* Algorithm

We analyzed the different parameters that influence the algorithm's performance. In addition we include in Figure A.10 three snapshots of the paths planned after running our proposed ACO-RRT* algorithm [2]. The figures show the resulting trajectory, the samples that conform the tree, and the ants at the end of the algorithm's execution. We can observe that most of the ants are placed in the region of the state space that contains the optimal trajectory. This results in the presence of more samples in this region, which is the goal of our algorithm. In the case of scenario 3 it can be seen how the first path found is already close to the optimal path.

A.6 Summary and Outlook

In this appendix we proposed and analyzed a novel path planning algorithm – ACO-RRT* – based on RRT. We modified the RRT algorithm sampling strategy so that the current tree influences the sampling strategy. This is done by the definition of an utility function in combination with an ant colony optimization algorithm. We define the utility function so that it trades off between: (i) exploitation of the best current solution; (ii) exploration of the states' space. We compared the ACO-RRT* performance with the RRT and RRT* algorithms in three challenging scenarios. ACO-RRT* is able to find a first path of a quality

[2]A video that shows a simulation execution in two different scenarios (Scenarios 1, 2) can be found in: `https://vimeo.com/253576604`; `https://rebrand.ly/plann385a`.

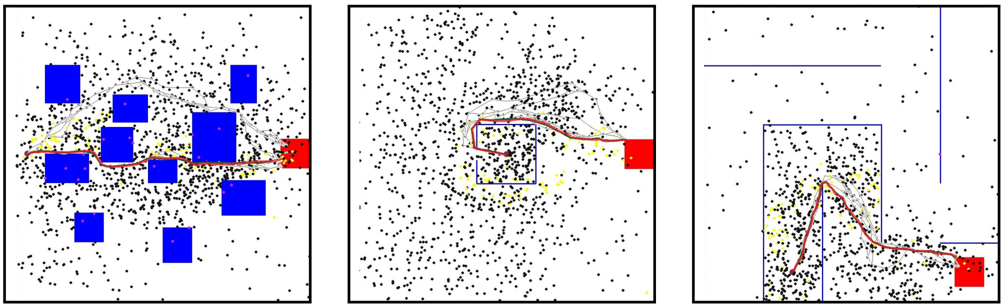

(a) Scenario 1. (b) Scenario 2. (c) Scenario 3.

Figure A.10: Example of one path planned with the ACO-RRT* algorithm for each of the analyzed scenarios. The big pink dot is the starting position. The red square is the goal region. The final path is coloured in red. The black dots are the samples generated by the algorithm. The yellow and pink dots represent the ants positions at the end of the algorithm's execution. Let us remark that the ants represented with the pink dots are the ones that were place on top of an obstacle during the algorithm's execution.

higher than the other alternatives (improvement factor between a 1.08 and 1.5). In addition, results suggest that our algorithm approaches the optimal solution 3.6 faster than RRT* algorithm. However ACO-RRT* requires more time to find a first path. In order to reduce this time, we extended the algorithm to an anytime version. Here, the algorithm searches a first path as fast as possible regardless of the path's cost, and then improves it using ACO-RRT*. Simulations results demonstrate that Anytime ACO-RRT* outperforms the state-of-the-art RRT/RRT* algorithms.

In the following we list future steps that we consider could enhance our proposed algorithm. We remark that we do not implement them in this thesis. These are the following:

1. Validating the algorithm in an experiment with a robot-in-the-loop.

2. Learning the optimal algorithm's parameters for an specific scenario. This could be done by means of RL, where a robot could automatically tune these parameters by analyzing the current solutions as it moves and interacts with the environment.

3. Incorporating some of the state-of-the-art path planning methods in this framework. In this appendix we proposed a framework that deals with the RRT algorithm. However, the proposed method is not restricted to RRT and further path planning methods could be easily incorporated, which we

strongly believe it could lead to a superior performance compared to RRT. To realize the proposed framework with a planning algorithm different from RRT, it is crucial to define a proper utility function that meets the planning algorithm's characteristics.

4. Extending the algorithm to a cooperative multi-robot case, as a multi-robot system would strongly benefit from exchanging ants between robots. This way, robots would exchange information about most promising areas in the environment which will lead to a more efficient path planning.

5. Handling more complex objective functions within our framework. In this appendix, our objective function corresponds to the cost of traversing a path, which we measure as path length. However, we could consider alternative objective functions. For example, given a robot subject to localization uncertainty, an objective function could steer the robot so that its uncertainty in the position its minimal while moving from an initial to a goal position. Another promising application, which is in line with this thesis, is the definition of an objective function to drive a robot to gather information efficiently.

This last application – the definition of an objective function to gather information – seems like the next natural step within the scope of this thesis. In fact, we implemented some ideas in this respect. Specifically, we employed ants in order to identify regions that are the most informative, by defining the ants' utility as the entropy at the ant's location. We also made the assumption that an informative point most likely belongs to a highly informative area. This way, sampling at a high informative location will involve placing an ant with a high utility in that area to sample there again in the future.

We performed some initial tests to validate the previous hypothesis. However, results yielded a performance slightly inferior than the algorithm proposed in Chapter 5. We believed that the inferior performance lies on the assumption we made: an informative point most likely belongs to a highly informative area. According to the simulation results, we concluded that this is only true for structured environments. For example, an office-like environment that is composed of several rooms would benefit of utilizing ACO. Here, a highly informative location will most likely correspond to an unexplored room. Then, by placing ants in that room we will explore it before moving to next one. In contrast, in an unstructured environment or in an environment with obstacles, a robot will move and leave small *patches* of unexplored locations. In case an ant identifies such a patch, placing more ants will not improve the algorithm's performance due to the small size of the patch relative to the number of ants.

To conclude: initial results suggest that we could benefit from ACO to gather information in structured environments. However, further investigations should be made in order to properly define suitable structured environments.

Appendix B
Multi-Robot Path Planning

In this appendix we present a preliminary study that we performed to better understand DCOP for multi-robot coordination, a technique that we latter exploited in Chapter 7 for information gathering. As in Appendix A, the application chosen to study DCOP is multi-robot path planning.

We would like to remark that this chapter does not only represent a preliminary study, but it is a major contribution by itself. Specifically, here we present, to the best of our knowledge, the first RRT-based multi-robot path planning algorithm that only requires local inter-robot communication. Next, we introduce the problem we aim to solve in detail.

B.1 Introduction

Multiple approaches have been proposed in the literature to deal with the multi-robot path planning problem. Approaches such as geometry-based algorithms (Siméon et al., 2002), dynamic programming (Swigart and Lall, 2010), linear programming (Ayanian and Kumar, 2010), mixed integer linear programming (MILP) (Habib et al., 2013), or decentralized model predictive control (DMPC) (Kuwata and How, 2011) have been shown to work well, but they suffer from a high computational complexity. This makes such approaches intractable for large teams of robots, and for missions that are subject to complex inter-robot constraints (Desaraju et al., 2012).

Efficient algorithms that offer a lower computational complexity, compared to the aforementioned ones, have been recently proposed in the literature (Zhang et al., 2016; Kim et al., 2015; Wei et al., 2014; Wagner and Choset, 2015). In particular, (Wagner and Choset, 2015) served us as inspiration for our approach. In (Wagner and Choset, 2015), the authors employ the concept of subdimensional expansion. That is: initially each of the robots plans a path individually, and then each robot coordinates its motion with the rest of the team as needed.

Despite the fact that the work of Wagner and Choset (2015) is a major contribution for multiple applications, it is not applicable to our problem. This is because it requires a discretization of the environment and robot's state space, which does not allow us to generalize the algorithm to a large class of robots (Section 2.2.1).

To solve this issue, we propose an algorithm that builds on RRTs (Section 2.2.2). Several works for multi-robot cooperation employed RRT to plan robots' paths (Ziyang et al., 2014; Saribatur et al., 2014; Levine et al., 2013). However, none of them focus on the specific problem of path planning for multiple robots.

The work by Desaraju et al. (2012) is the one that is closest to our proposed approach. They employ the closed-loop RRT algorithm (Kuwata et al., 2008) in combination with a token based approach to solve the multi-robot path planning problem in a decentralized manner. However, the authors assume full connectivity between robots. Such assumption is unrealistic, as communication links have a limited range. In contrast, in this work, we propose an algorithm that only requires local communication between robots. Like in (Wagner and Choset, 2015), our algorithm is based on a two steps approach: (i) first, robots plan several paths individually using the RRT algorithm, (ii) second, robots cooperate with their neighbors as needed to find the set of paths that minimizes the total distance travelled by robots. The second step is realized using DCOP (Leite et al., 2014).

The remainder of this paper is organized as follows. Section B.2 states formally the problem. Section B.3 summarizes the DCOP approach that we employ to achieve multi-robot coordination. Then we describe in Sections B.4 and B.5 our distributed multi-robot path planning algorithm. Sections B.6 and B.7 present the simulations and experiments performed, and is then followed by a summary and outlook in Section B.8.

B.2 Problem Statement

Let us consider a network of N robots. Each robot $i = 1, 2, ..., N$ aims to visit, in a predefined order, the set of stations $\mathcal{S}_i = \{\mathbf{s}_{i,1}, \mathbf{s}_{i,2}, ..., \mathbf{s}_{i,n_i}\}$ (for instance, a set of places where to take measurements), with n_i the number of stations that robot i aims to visit. We denote the path that links stations $\mathcal{S}_i$ as $\mathcal{P}_{\mathcal{S}_i} = [\mathcal{P}_{\mathbf{s}_{i,1},\mathbf{s}_{i,2}}, \mathcal{P}_{\mathbf{s}_{i,2},\mathbf{s}_{i,3}}, ..., \mathcal{P}_{\mathbf{s}_{i,n_i-1},\mathbf{s}_{i,n_i}}]$.

The goal of agent i is to find $\mathcal{P}_{\mathcal{S}_i}$ that minimizes function $U(\mathcal{P}_{\mathcal{S}_1}, \mathcal{P}_{\mathcal{S}_2}, ..., \mathcal{P}_{\mathcal{S}_N})$. $U(\cdot)$ is a global function that encodes the inter-dependencies of all robots that conform the network. In this work, we define $U(\cdot)$ as the total distance traveled by robots while avoiding collisions. Let us remark that a large class of functions can be considered within the framework proposed in this appendix, as we will describe in Section B.3. An example of such function is the information gathering utility (7.10), introduced in Chapter 7.

In this work, we consider the following constraints:

1. Robots move according to the model introduced in Section 4.1.
2. Inter-robot communication is given by a disc communication model of radius r_c (definition (6.1)).
3. Two robots collide if they are separated less than a distance r_s.

In addition, we assume the following:

1. The robot's position is known exactly and noise-free.
2. The borders and obstacles that define the environment are *a priori* known.
3. Robots initially know neither about the presence nor about the location of other robots.
4. Stations $\mathbf{s}_{i,j}, \mathbf{s}_{i,j+1}$ are separated a maximum distance $\frac{r_c - r_s}{2}$. This simplifying assumption allows us to easily cast our problem within the DCOP algorithm described in Section B.3. We discuss possible solutions to relax this assumption in Section B.5.

B.3 Asynchronous Distributed Constraint Optimization

In this appendix we propose the use of a DCOP algorithm – Adopt – (Modi et al., 2005) to tackle the multi-robot path planning problem described in Section B.2. In contrast to other DCOPs, Adopt provides theoretical guarantees on the global solution optimality while keeping communication between agents asynchronous and localized. This is our motivation to incorporate Adopt into our algorithm.

Adopt. In Adopt, each robot can control a decision variable d_i that can take values from domain $\mathcal{P}_i = \{\mathcal{P}_i^{[1]}, \mathcal{P}_i^{[2]}, ..., \mathcal{P}_i^{[k_i]}\}$, with k_i the number of elements in the domain. The goal of the robots is to minimize a global utility function $U(\mathcal{A})$, where $\mathcal{A}$ denotes a possible assignment for the variables. Adopt only allows us to solve problems that involve binary constraints between robots. We denote a constraint between robots i and j as $g_{i,j}(d_i, d_j)$. The goal is to find the optimal assignment $\mathcal{A}^*$ that minimizes the following function:

$$U(\mathcal{A}) = \sum_{d_i,d_j \in \mathcal{D}} g_{i,j}(d_i, d_j), \tag{B.1}$$

in a distributed fashion, where each agent i is only in control of its own variable d_i. The robots participating in Adopt must conform a connected subnetwork. Let us empathize that this framework allows us to formulate a large class of objective functions. For more details about the properties that (B.1) must meet, please refer to (Modi et al., 2005).

DFS tree. Adopt takes as input a Depth-First Search (DFS) tree that encodes the inter-robot constraints. Vertices of the tree represent robots, and edges define inter-robot constraints. In this work, we implemented the algorithm proposed in (Awerbuch, 1985) to create a DFS tree.

(Awerbuch, 1985) is fully distributed. However, one robot – the leader – must trigger the DFS tree creation. The leader will become then the root of the DFS tree. Here, for leader election we implemented the YO-YO algorithm (Santoro, 2006), since it is well suited to connected subnetworks of arbitrary topology, as it is the case in this work.

B.4 Algorithm Overview

Here we propose an algorithm that allows multiple robots to jointly plan their paths in a distributed fashion. Instead of optimizing $U(\cdot)$ in one step, we do it sequentially. That is, robots cooperate to plan their paths between their current station and the next one. Once they reach an station, they cooperate again. This problem relaxation allows us to keep the algorithm complexity bounded and independent of the number of stations. Moreover, solving $U(\cdot)$ in one step is not necessarily optimal. Imagine, for example, that a mission requires incorporating a new robot to the system. In this case, robots should replan their complete paths to consider the new constraints imposed by this new robot, which would make the optimization for the full set of stations useless. This strategy is identical to the one proposed in Chapter 7 since it is well suited to dynamic network topologies, which result as robots move to explore.

State machine. Figure B.1 shows the state machine that controls the algorithm's execution. First, when a robot reaches a station, it searches for neighbors by sending an identification message. Then neighboring robots will create connected subnetworks. Notice that, within a large environment, several isolated subnetworks may be created (see Fig. B.2). Next, each subnetwork will execute the algorithm proposed in Section B.5. This is realized in two steps. First,

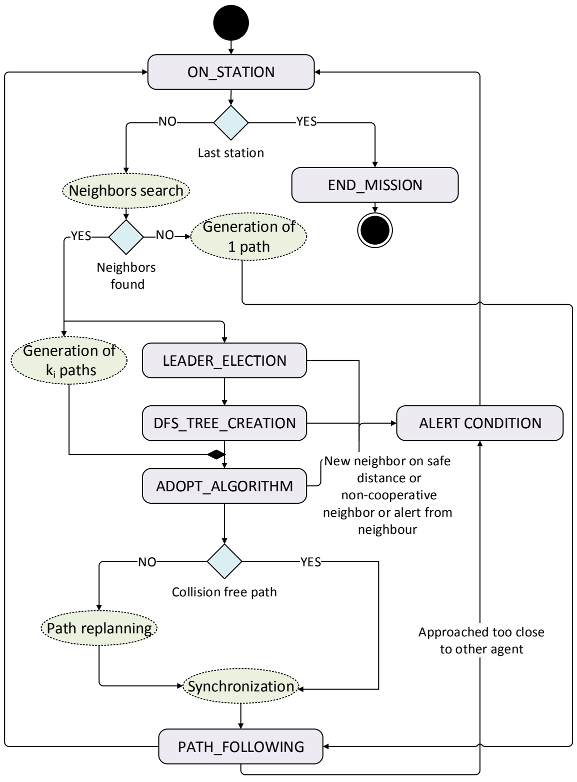


Figure B.1: State machine that describes the algorithm's execution.

robots plan several paths individually towards the next station. To this end, as in Chapter 7, we employ RRT (Section 2.2.2). Let us remark that this choice is motivated by the fact that, due to the random nature of RRT, it delivers different solutions each time we run it, which is a requirement for our algorithm. Second, robots cooperate to find $\mathcal{A}^*$ that minimizes $U(\cdot)$ while avoiding inter-robot collisions.

Path replanning. Each of the robots plans k_i paths to the next station. However, this does not guarantee the existence of a collision-free solution. This lies on the following: (i) RRT generates a finite number of paths k_i, which does not guarantee that there exists a collision-free combination of paths, or (ii) Adopt cannot converge to a solution before an user-defined time threshold. Therefore, we introduce a replanning strategy to deal with this problem (Section B.5.1).

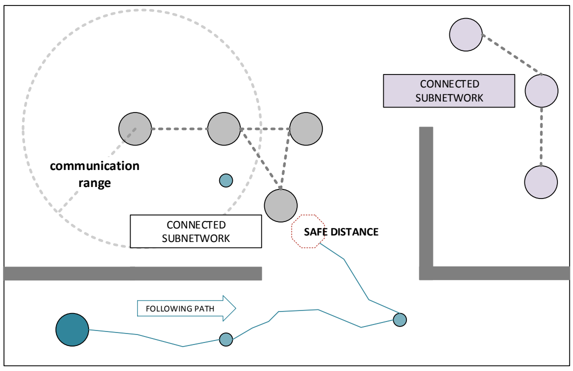


Figure B.2: Example of an scenario where multiple robots aim to visit a set of stations. On the one hand, robots that are close form a connected subnetwork to find a joint solution to move towards the next station. On the other hand, robots that are far plan their paths individually. This is realized with our proposed algorithm.

Robots synchronization. The last step of the algorithm for a connected subnetwork consists of a synchronization step that forces robots to start at the same time. This is required to avoid collisions. The robots will continue the algorithm's execution until they reach the last station. The aforementioned procedures are explained in further detail in Section B.5.

Single-robot subnetwork. Let us also emphasize that in the first step of the algorithm a robot could find no neighbors; for instance, in large environments or for robots that have a small communication range. Then, the robot will plan its path individually and follow it till the next station. This is a natural approach since robots that are not in communication range can not cooperate to find a joint solution. However, it could happen that while robots are moving they find another robots. Then, an alert condition will be triggered and robots involved will stop following their paths and will execute the second step of the algorithm; i.e. creation of a connected subnetwork.

B.5 Multi-Robot Path Planning for a Connected Subnetwork

This section explains the procedures that robots execute once they created a connected subnetwork. First, each of the robots generates k_i paths between $\mathbf{x}_{r_i}$

and their next station using RRT.

Second, robots elect a leader, create the DFS tree required by Adopt, and start the execution of Adopt (Section B.3). We employ Adopt to find $\mathcal{A}^*$ that minimizes the total distance travelled by robots while avoiding collision between robots. We denote the distance associated to path $\mathcal{P}_i^{[l]}$ as $\texttt{Dist}(\mathcal{P}_i^{[l]})$. Following the notation introduced in Section B.3, we first define $g_{i,j}(\cdot)$ for the collision-free case. This is as follows:

$$g_{i,j}(\mathcal{P}_i^{[l]}, \mathcal{P}_j^{[m]}) = \frac{\texttt{Dist}(\mathcal{P}_i^{[l]})}{|\mathcal{N}_i|} + \frac{\texttt{Dist}(\mathcal{P}_j^{[m]})}{|\mathcal{N}_j|}, \tag{B.2}$$

with $l \in [1, 2, ..., k_i], m \in [1, 2, ..., k_j]$, and $|\mathcal{N}_i|, |\mathcal{N}_j|$ the number of neighbors of robots i, j.

It is also possible that $\mathcal{P}_i^{[l]}$ and $\mathcal{P}_j^{[m]}$ incur in a collision; i.e. they are separated a distance smaller than r_s, at the same time instant. In case of a collision, $g_{i,j}(\mathcal{P}_i^{[l]}, \mathcal{P}_j^{[m]})$ takes a value equal to infinite as described in Section 7.5.3.

Let us remark that this definition of the utility function allows the robot to find a collision-free solution that minimizes the total traveled distance for subnetworks of arbitrary topology. For example, let us consider a subnetwork of 3 robots where robot 1 is connected to 2, robot 2 is connected to 1 and 3, and robot 3 is connected to 2. This results in $|\mathcal{N}_1| = 1, |\mathcal{N}_2| = 2, |\mathcal{N}_3| = 1$. We can then easily check by substituting those values in (B.2) that (B.1) is equal to the total distance traveled by all robots.

B.5.1 Path Replanning

We introduce a path replanning mechanism to guarantee the existence of a solution to the path planning problem. Since robots are ordered in a DFS tree, we exploit this hierarchy for the replanning mechanism. This is triggered by the tree root, which sends its shortest path to all its descendants. Once a node receives the path proposed by its father, it calculates a path that does not collide with the ones proposed by its ancestors; i.e. all nodes that link it to the root. Specifically, we calculate the path with RRT, and assume the ancestors' paths as spatio-temporal obstacles.

Replanning continues till robots at the leaves of the DFS tree are reached. Once robots finish replanning, they send through the DFS tree a message back to the root informing about the completion of replanning.

B.5.2 Robots Synchronization

The last step of the algorithm is a synchronization procedure. As we previously mentioned, collisions between paths are checked both in the spatial and temporal dimension. On the one hand, the consideration of the temporal dimension reduces the number of potential collisions, as we are adding an additional dimension. On the other hand, this requires that robots are synchronized; i.e. they must start following their paths at the same time instant. We would like to point out that the consideration of the time dimension is possible because this information is stored in the RRT, which is generated by considering the robots motion model.

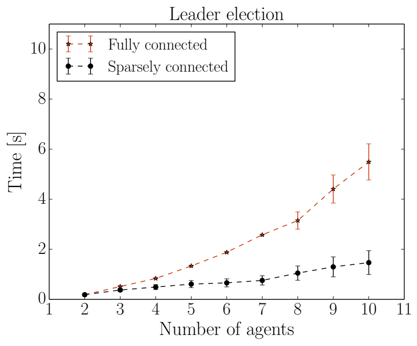
The root of the DFS tree is the one that will trigger the path following procedure. This message will be sent through the tree. Once a node receives this message it will start following its path. Since the message must travel through the network, we assume there will be a small delay between different nodes that will increase with the tree's depth. Let us remark that this can be taken into account by increasing the safety distance proportionally to the expected delay, which can be calculated given the communication protocol and the DFS tree topology.

B.5.3 Stations Separation

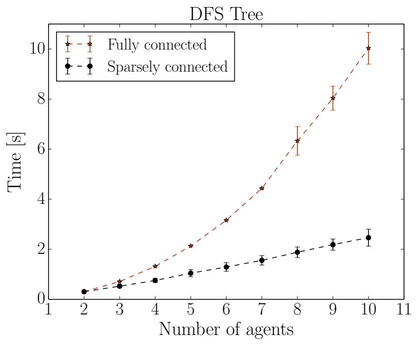
In Section B.2, we introduced a simplifying assumption that limits the maximum distance between two stations; this corresponds to assumption 4. This is motivated by the fact that Adopt can only accept binary constraints. Then, by limiting the planning horizon to $\frac{r_c - r_s}{2}$ we can guarantee that robots that are not in communication range will never collide. This assumption could be easily removed by defining an intermediate station between the two considered stations. The robot would plan first to this intermediate station and then to the final one.

B.6 Simulations and Discussion of Results

We validate our proposed algorithm in two steps. First, we validate the algorithm for a connected subnetwork (Section B.5) by performing Monte Carlo simulations. Specifically, we evaluate the performance of the leader election, the DFS tree creation, and the distributed assignment of paths. Second, we illustrate with an example the algorithm's behaviour for a system composed of several subnetworks, as described in Section B.4. Here, and without loss of generality, we consider a holonomic robot in order to abstract the robot's motion from the algorithm's behavior. We carry out all simulations in a central computer in a decentralized fashion using ROS (Quigley et al., 2009).



(a) Leader election.



(b) DFS tree creation.

Figure B.3: Leader election and DFS tree creation. Fully and sparsely connected subnetworks.

B.6.1 Leader Election and Depth-First Search Tree Creation

First we evaluate the convergence time of both the leader election and DFS tree creation algorithms as we increase the number of robots in the subnetwork. This is crucial to understand the scalability of the proposed algorithm.

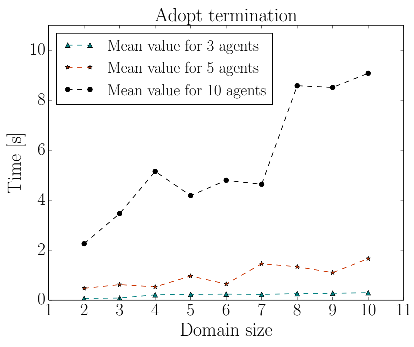
We perform simulations for two types of subnetworks: (i) a fully connected subnetwork where all robots can communicate with each other; (ii) a sparsely connected subnetwork where each robot has a maximum number of 5 neighbors. This last one corresponds to a more realistic scenario that we could encounter in a mission that takes place within a large environment. For each of the simulated number of agents, we repeat the simulation 100 times. Let us add that for the sparsely connected case a new random network is generated each time. Figure B.3 shows the resulting average and variance for the different scenarios.

We can conclude that the complexity of both algorithms grow exponentially for a fully connected subnetwork. However, the complexity is linear for a sparsely connected subnetwork, which makes both algorithms suitable for real world missions.

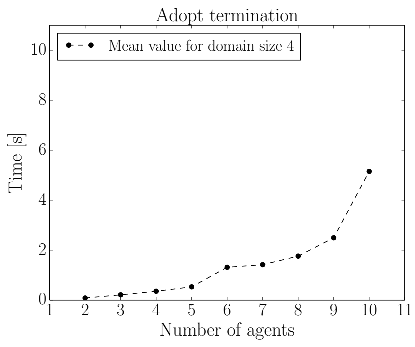
B.6.2 Distributed Assignment of Paths

Here we analyze the performance of Adopt as we vary the number of robots in the subnetwork, and the number of paths in a robot's domain. In both cases, we consider a sparse subnetwork with the same properties as in Sec, B.6.1. We repeat each of the simulations 100 times.

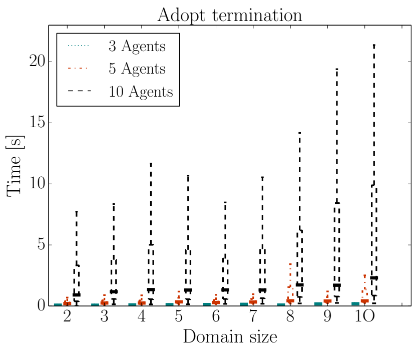
Figures B.4a,B.4c and B.5a,B.5c show the time and the total number of exchanged messages, respectively, that Adopt required to converge to the optimal solution as we increased the domain's size. We present the average and a box



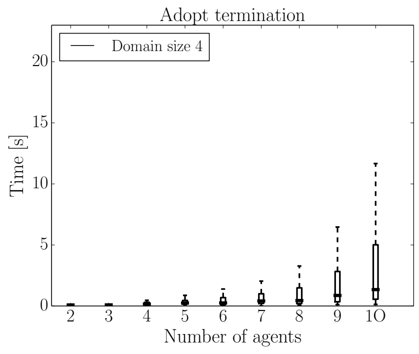
(a) Average performance with the number of paths in the domain.



(b) Average performance with the number of agents.



(c) Performance with the number of paths in the domain.

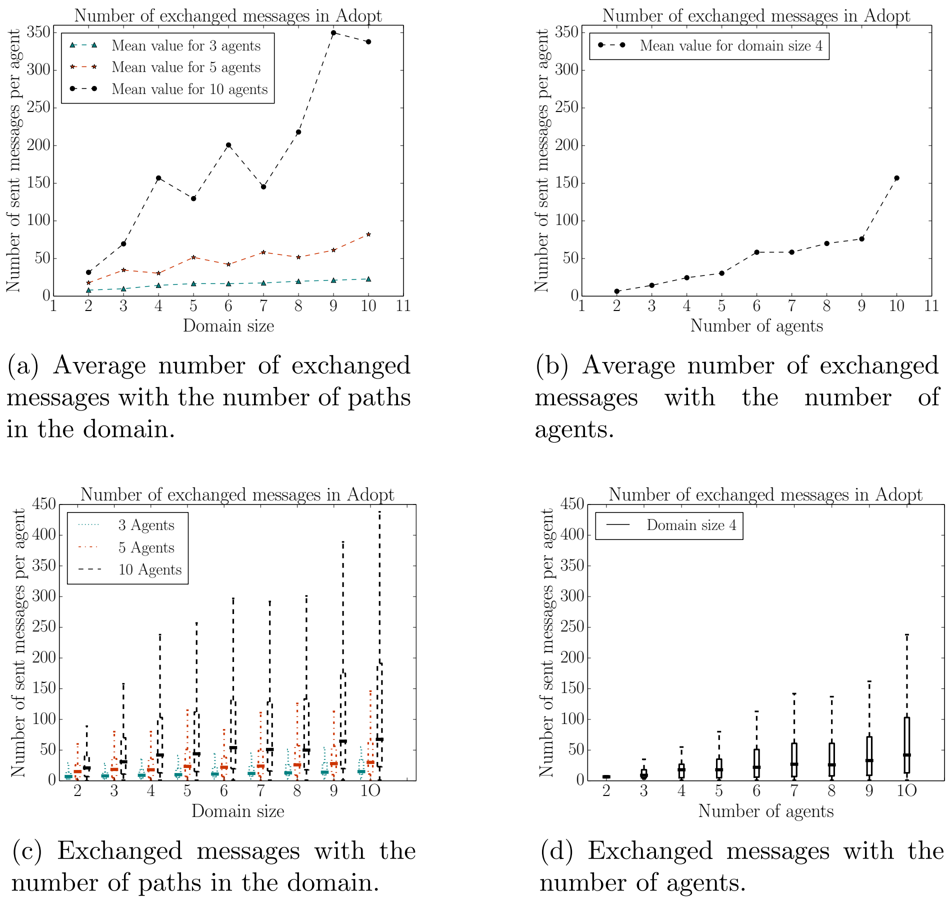


(d) Performance with the number of agents.

Figure B.4: Distributed Assignment of Paths. (a,c) Performance with the number of paths in the domain for a network with 3, 5 and 10 agents. (b,d) Performance with the number of agents given a fixed domain composed of 4 paths.

plot representation of the data. These simulations were carried out for a domain size – number of paths – ranging between 2 and 10, and we considered $3, 5$ and 10 robots.

We can observe in Figures B.4a,B.4c that for a small number of robots the algorithm's complexity remains quasi constant respect to the domain size. However, for a large number of robots (10) the performance increases linearly. The explanation for such behavior can be understood by analyzing Figures B.4b,B.4d and B.5b,B.5d. Here we show the algorithm's complexity as we vary the number of robots given a fixed domain composed of 4 paths. We can confirm that the algorithm's complexity grows exponentially as Modi et al. (2005) point out. Moreover, the box plot representation confirms that the algorithm's complexity



(a) Average number of exchanged messages with the number of paths in the domain.

(b) Average number of exchanged messages with the number of agents.

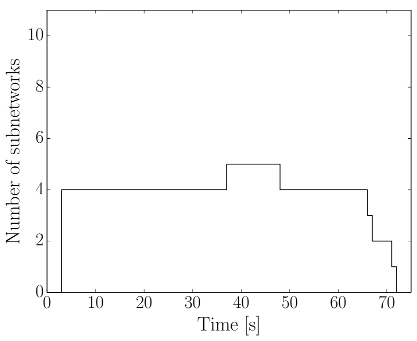
(c) Exchanged messages with the number of paths in the domain.

(d) Exchanged messages with the number of agents.

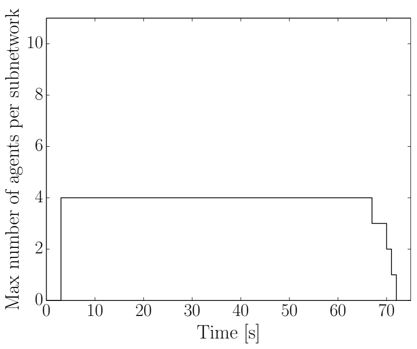
Figure B.5: Distributed Assignment of Paths. (a,c) Number of exchanged messages with the number of paths in the domain for a network with 3, 5 and 10 agents. (b,d) Number of exchanged messages with the number of agents given a fixed domain composed of 4 paths.

is highly dependent of the network topology. This is one of the main reasons why we introduced the replanning process in our algorithm; if Adopt takes too long to converge, the replanning method will be triggered to find a feasible solution.

Attending to results we can conclude that: (i) the use of Adopt within a small subnetwork results in a low computational complexity, and (ii) for a large subnetwork we should either consider alternative algorithms (like e.g. max-sum (Farinelli et al., 2008)), or introduce additional constraints into the algorithm's design to avoid the creation of large subnetworks. As our goal in this thesis is to develop an algorithm that is scalable with the number of robots, we decided to use max-sum in Chapter 7 for our informative path planning algo-



(a) Number of subnetworks.



(b) Maximum number of agents per subnetwork.

Figure B.6: Number of subnetworks and maximum number of robots per subnetwork for a typical scenario with 10 robots.

rithm.

B.6.3 Multi-Robot Path Planning for a System with Multiple Subnetworks

Finally, we show one example of the whole system's behavior for a typical scenario with 10 robots. In a typical scenario, given a limited communication radius, $\mathcal{G}_c(\cdot)$ is divided into several subgraphs (subnetworks), and robots cannot communicate with the rest of robots that compose the system. In this case, robots will execute the algorithm described in Section B.4.

We show in Figure B.6 the evolution with time of the number of subnetworks, and maximum number of robots per subnetwork, which results as robots move. We observe that the maximum number of robots per subnetwork is 4, although we have a system of 10 robots. This property implies that, even for a large system composed of multiple robots, the number of robots per subnetwork remains low. Therefore, the computational load of each of the robots remains also much lower, as we indicated in Fig. B.4.

B.7 Experiments and Discussion of Results

We validated the proposed algorithm in an experiment [1] with three holonomic robots (see Figure B.7). For a more detailed description of the robots, we refer the reader to Section 4.3. Like we did for the simulations, here we also run the

[1]A video that shows a simulation and an experiment execution can be found in: `https://vimeo.com/253576649`; `https://rebrand.ly/anasyaba6`.

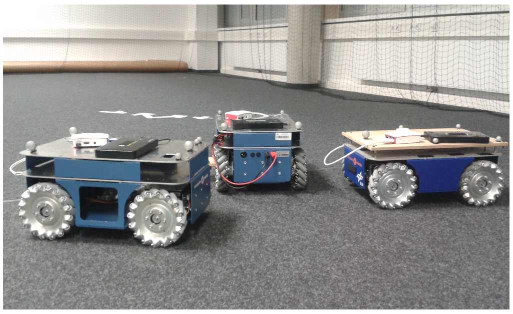


Figure B.7: The three holonomic robots employed to carry out the experimental validation of the proposed algorithm.

algorithm in a central computer in a decentralized fashion. Then we send the corresponding waypoints to the robot using ROS with a WiFi connection. Each robot is equipped with a Raspberry Pi that runs the robots controller to guide the robot to the desired wapoint.

For the experiment, we assume that robots initially conform a connected network. In particular, two robots are assigned to three stations, while the remaining one is assigned to two. We set a domain size of two for each of the robots; i.e. each robot proposes to its neighbors two possible paths to travel between stations.

Experiments demonstrated that robots were able to visit the assigned stations, using our proposed algorithm. We would also like to remark that the cooperation between robots took, in the worst case, less than 2 seconds.

B.8 Summary and Outlook

In this chapter, we proposed an approach for multi-robot path planning. Specifically, our approach builds on RRTs and a DCOP technique (Adopt). On the one hand, RRTs allow us to generalize our algorithm to a large class of robots. On the other hand, Adopt provides a mechanism to achieve multi-robot cooperation in an asynchronous and distributed fashion, and with local communication between robots.

Beyond this thesis, we foresee possible improvements that could enhance our proposed algorithm's performance. These are the following:

1. Adopt performance scales linearly with the number of robots, for a small subnetwork. However, Adopt scales exponentially for large subnetworks. Moreover, Adopt requires a DFS tree and a leader election algorithm, and both algorithms scale exponentially with the number of robots. This led

us to the conclusion that Adopt is not suited for systems composed by a large number of robots, and alternatives should be considered. In our case, we decided to use max-sum for multi-robot coordination (Farinelli et al., 2008), as it scales exponentially only with the number of neighbors, and it does not require any preprocessing steps. For a detailed description of max-sum, we refer the reader to Section 7.3.

2. Our algorithm can handle large robotic systems composed of several subnetworks. As robots moves, the network topology changes and robots leave/join subnetworks. To handle these changes in the network topology we introduced an alert condition that stops the subnetwork coordination procedure in case a new robot is discovered. This works in practice for a small number of robots. However, we can expect multiple interruptions for a large number of robots. Therefore, alternative mechanisms should be incorporated into the system. One alternative is to add additional constraints to force the system to be connected, as we propose in Chapter 7. Another alternative is to trigger the alarm condition only sporadically. This way we will have cooperative robots, and robots that act by themselves.

3. The experiment performed in this chapter was a toy example to better understand how our proposed algorithm performs with actual robots. In the future we would like to extend the experiments to a larger scale, as well as to consider a larger number of non-holonomic robots.

Bibliography

Baris Akgun and Mike Stilman. Sampling heuristics for optimal motion planning in high dimensions. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 2640–2645. IEEE, 2011.

D Alejo, JA Cobano, G Heredia, and A Ollero. Optimal reciprocal collision avoidance with mobile and static obstacles for multi-UAV systems. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 1259–1266. IEEE, 2014.

Michael J Allen. Updraft model for development of autonomous soaring uninhabited air vehicles. In *Forty Fourth AIAA Aerospace Sciences Meeting and Exhibit*, pages 1–19, 2006.

Ron Alterovitz, Sachin Patil, and Anna Derbakova. Rapidly-exploring roadmaps: Weighing exploration vs. refinement in optimal motion planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3706–3712. IEEE, 2011.

Nancy M Amato and Guang Song. Using motion planning to study protein folding pathways. *Journal of Computational Biology*, 9(2):149–168, 2002.

Michael Angermann, Martin Frassl, Marek Doniec, Brian J Julian, and Patrick Robertson. Characterization of the indoor magnetic field for applications in localization and mapping. In *Indoor Positioning and Indoor Navigation (IPIN), 2012 International Conference on*, pages 1–9. IEEE, 2012.

Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2421–2428. IEEE, 2013.

Oktay Arslan and Panagiotis Tsiotras. Dynamic programming guided exploration for sampling-based motion planning algorithms. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4819–4826. IEEE, 2015.

Baruch Awerbuch. A new distributed depth-first-search algorithm. *Information Processing Letters*, 20(3):147–150, 1985.

Nora Ayanian and Vijay Kumar. Decentralized feedback controllers for multiagent teams in environments with obstacles. *IEEE Transactions on Robotics*, 26(5):878–887, 2010. ISSN 15523098.

Tucker Balch and Lynne E Parker. *Robot teams: from diversity to polymorphism.* AK Peters, Ltd., 2002.

Wolfram Burgard, Mark Moors, Dieter Fox, Reid Simmons, and Sebastian Thrun. Collaborative multi-robot exploration. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 476–481. IEEE, 2000.

Brendan Burns and Oliver Brock. Toward optimal configuration space sampling. In *Robotics: Science and Systems*, pages 105–112. Citeseer, 2005.

J Capitan, L Merino, F Caballero, and A Ollero. Decentralized delayed-state information filter (DDSIF): A new approach for cooperative decentralized tracking. *Robotics and Autonomous Systems*, 59:376–388, 2011. doi: 10.1016/j.robot.2011.02.001.

Henry Carrillo, Ian Reid, and José A Castellanos. On the comparison of uncertainty criteria for active SLAM. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 2080–2087. IEEE, 2012.

David W Casbeer, Randal W Beard, Timothy W McLain, Sai-Ming Li, and Raman K Mehra. Forest fire monitoring with multiple small UAVs. In *American Control Conference, 2005. Proceedings of the 2005*, pages 3530–3535. IEEE, 2005.

Benjamin Charrow, Sikang Liu, Vijay Kumar, and Nathan Michael. Information-theoretic mapping using cauchy-schwarz quadratic mutual information. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 2015.

Jie Chen, Kian H Low, Colin Tan, Ali Oran, Patrick Jaillet, John M Dolan, and Gaurav S Sukhatme. Decentralized data fusion and active sensing with mobile sensors for modeling and predicting spatiotemporal traffic phenomena. *arXiv preprint arXiv:1206.6230*, 2012.

Jie Chen, Kian H Low, Yujian Yao, and Patrick Jaillet. Gaussian process decentralized data fusion and active sensing for spatiotemporal traffic modeling and prediction in mobility-on-demand systems. *IEEE Transactions on Automation Science and Engineering*, 12(3):901–921, 2015.

H L Choi and S J Lee. A potential-game approach for information-maximizing cooperative planning of sensor networks. *IEEE Transactions on Control Systems Technology*, 23(6):2326–2335, Nov 2015. ISSN 1063-6536.

Han L Choi and Jonathan P How. Continuous trajectory planning of mobile sensors for informative forecasting. *Automatica*, 46(8):1266–1275, 2010. ISSN 00051098. URL `http://dx.doi.org/10.1016/j.automatica.2010.05.004`.

Jen Jen Chung, Nicholas R J Lawrance, and Salah Sukkarieh. Learning to soar: Resource-constrained exploration in reinforcement learning. *The International Journal of Robotics Research*, 34(2):158–172, 2014. ISSN 0278-3649.

Oliver M Cliff, Robert Fitch, Salah Sukkarieh, Debra L Saunders, and Robert Heinsohn. Online localization of radio-tagged wildlife with an autonomous aerial robot system. *Proceedings of Robotics Science and Systems XI*, pages 13–17, 2015.

Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

Noel Cressie. Statistics for spatial data. *Terra Nova*, 4(5):613–617, 1992.

Philip Dames, Mac Schwager, Daniela Rus, and Vijay Kumar. Active magnetic anomaly detection using multiple micro aerial vehicles. *Robotics and Automation Letters, IEEE*, 2015.

Jory Denny, Miguel Morales, Saul Rodriguez, and Nancy M Amato. Adapting RRT growth for heterogeneous environments. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1772–1778. IEEE, 2013a.

Jory Denny, Kensen Shi, and Nancy M Amato. Lazy toggle PRM: a single-query approach to motion planning. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2407–2414. IEEE, 2013b.

Vishnu R Desaraju, Jonathan P How, V R Desaraju, and J P How. Decentralized path planning for multi-agent teams with complex constraints. *Auton Robot*, 32:385–403, 2012. ISSN 0929-5593.

Rosen Diankov and James Kuffner. Randomized statistical path planning. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 1–6. IEEE, 2007.

Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

Cory Dixon and Eric W Frew. Maintaining optimal communication chains in robotic sensor networks using mobility control. *Mobile Networks and Applications*, 14(3):281–291, 2009.

Cho Doo-Hyun, Ha Jung-Su, Lee Su-Jin, Moon Sunghyun, and Choi Han-Lim. Informative path planning and mapping with multiple UAVs in wind fields. In *Proceedings of the 13th International Symposium on Distributed Autonomous Robotic Systems, DARS*, 2016. URL `http://arxiv.org/abs/1610.01303`.

Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: Optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.

Alessandro Farinelli, Alex Rogers, Adrian Petcu, and Nicholas R Jennings. Decentralised coordination of low-power embedded devices using the max-sum algorithm. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 639–646. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Dave Ferguson and Anthony Stentz. Anytime RRTs. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5369–5375. IEEE, 2006.

Jonathan Fink and Vijay Kumar. Online methods for radio signal mapping with mobile robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1940–1945. IEEE, 2010.

Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. BIT*: Batch informed trees for optimal sampling-based planning via dynamic programming on implicit random geometric graphs. Technical report, Tech. Report TR-2014-JDG006, ASRL, University of Toronto, 2014a.

Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. *arXiv preprint arXiv:1404.2334*, 2014b.

Seng K Gan, Robert Fitch, and Salah Sukkarieh. Online decentralized information gathering with spatial–temporal constraints. *Autonomous Robots*, 37(1): 1–25, 2014.

Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2013.

S Grime and H F Durrant-Whyte. Data fusion in decentralized sensor networks. *Control Engineering Practice*, 2(5):849–863, Oct. 1994.

B Grocholsky, J Keller, V Kumar, and G Pappas. Cooperative air and ground surveillance. *IEEE Robotics Automation Magazine*, 13(3):16–25, Sept 2006. ISSN 1070-9932.

Carlos Guestrin, Andreas Krause, and Ajit P Singh. Near-optimal sensor placements in Gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272. ACM, 2005.

Leonidas J Guibas, Christopher Holleman, and Lydia E Kavraki. A probabilistic roadmap planner for flexible objects with a workspace medial-axis-based sampling approach. In *Intelligent Robots and Systems, 1999. IROS'99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 1, pages 254–259. IEEE, 1999.

Durdana Habib, Habibullah Jamal, and Shoab A Khan. Employing multiple unmanned aerial vehicles for co-operative path planning. *International Journal of Advanced Robotic Systems*, 10:1–10, 2013. ISSN 17298806.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

G M Hoffmann and C J Tomlin. Mobile sensor network control using mutual information methods and particle filters. *IEEE Transactions on Automatic Control*, 55(1):32–47, Jan 2010. ISSN 0018-9286.

Geoffrey Hollinger and Sanjiv Singh. Multi-robot coordination with periodic connectivity. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4457–4462. IEEE, 2010.

Geoffrey A Hollinger and Gaurav S Sukhatme. Sampling-based robotic information gathering algorithms. *The International Journal of Robotics Research*, 33 (9):1271–1287, 2014.

Geoffrey A Hollinger, Sunav Choudhary, Parastoo Qarabaqi, Christopher Murphy, Urbashi Mitra, Gaurav S Sukhatme, Milica Stojanovic, Hanumant Singh, and Franz Hover. Underwater data collection using robotic sensor networks. *IEEE Journal on Selected Areas in Communications*, 30(5):899–911, 2012.

David Hsu, Tingting Jiang, John Reif, and Zheng Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 4420–4426. IEEE, 2003.

Yong K Hwang and Narendra Ahuja. A potential field approach to path planning. *IEEE Transactions on Robotics and Automation*, 8(1):23–32, 1992.

Maani G Jadidi, Jaime V Miro, Rafael Valencia, and Juan Andrade-Cetto. Exploration on continuous Gaussian process frontier maps. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6077–6082. IEEE, 2014.

Maani G Jadidi, Jaime V Miro, and Gamini Dissanayake. Mutual information-based exploration on continuous occupancy maps. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 6086–6092. IEEE, 2015.

Léonard Jaillet, Anna Yershova, Steven M La Valle, and Thierry Siméon. Adaptive tuning of the sampling domain for dynamic-domain RRTs. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2851–2856. IEEE, 2005.

Léonard Jaillet, Juan Cortés, and Thierry Siméon. Transition-based RRT for path planning in continuous cost spaces. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2145–2150. IEEE, 2008.

Léonard Jaillet, Juan Cortés, and Thierry Siméon. Sampling-based path planning on configuration-space costmaps. *Robotics, IEEE Transactions on*, 26(4): 635–646, 2010.

Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, page 0278364915577958, 2015.

Brian J Julian, Sertac Karaman, and Daniela Rus. On mutual information-based control of range sensing robots for mapping applications. *The International Journal of Robotics Research*, 33(10):1375–1392, 2014.

Ma Kai-Chieh, Ma Zhibei, Liu Lantao, and Gaurav S Sukhatme. Multi-robot informative and adaptive planning for persistent environmental monitoring. In *Proceedings of the 13th International Symposium on Distributed Autonomous Robotic Systems, DARS*, 2016.

Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the RRT*. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 1478–1483. IEEE, 2011.

Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

Anssi Kemppainen, Janne Haverinen, Ilari Vallivaara, and Juha Röning. Near-optimal SLAM exploration in Gaussian processes. In *Multisensor Fusion and Integration for Intelligent Systems (MFI), 2010 IEEE Conference on*, pages 7–13. IEEE, 2010.

Donghyuk Kim, Junghwan Lee, and Sung-eui Yoon. Cloud RRT*: Sampling cloud based RRT*. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 2519–2526. IEEE, 2014.

Kangjin Kim, Joe Campbell, William Duong, Yu Zhang, and Georgios Fainekos. DisCoF + : Asynchronous DisCoF with flexible decoupling for cooperative pathfinding in distributed systems. In *IEEE International Conference on Automation Science and Engineering (CASE)*, pages 369—-376. IEEE, 2015. ISBN 9781467381833.

Marin Kobilarov. Cross-entropy motion planning. *The International Journal of Robotics Research*, 31(7):855–871, 2012.

Andreas Krause and Carlos Guestrin. Near-optimal observation selection using submodular functions. In *AAAI*, volume 7, pages 1650–1654, 2007a.

Andreas Krause and Carlos Guestrin. Nonmyopic active learning of Gaussian processes: an exploration-exploitation approach. In *Proceedings of the 24th international conference on Machine learning*, pages 449–456. ACM, 2007b.

Andreas Krause and Carlos Guestrin. Submodularity and its applications in optimized information gathering. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(4):32, 2011.

Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *The Journal of Machine Learning Research*, 9:235–284, 2008.

Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2): 498–519, 2001.

James J Kuffner and Steven M LaValle. RRT-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

Yoshiaki Kuwata and Jonathan P How. Cooperative distributed robust trajectory optimization using receding horizon MILP. *IEEE Transactions on Control Systems Technology*, 19(2):423–431, 2011. ISSN 10636536.

Yoshiaki Kuwata, Justin Teo, Sertac Karaman, Gaston Fiore, Emilio Frazzoli, and Jonathan P How. Motion planning in complex environments using closed-loop prediction. In *Proc. AIAA Guidance, Navigation, and Control Conf. and Exhibit*, 2008.

Xiaodong Lan and Mac Schwager. Planning periodic persistent monitoring trajectories for sensing robots in Gaussian random fields. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2415–2420. IEEE, 2013.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Steven M LaValle and James J Kuffner. Rapidly-exploring random trees: Progress and prospects, 2000.

Steven M LaValle and James J Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

Nicholas RJ Lawrance and Salah Sukkarieh. Autonomous exploration of a wind field with a gliding aircraft. *Journal of guidance, control, and dynamics*, 34 (3):719–733, 2011.

Allan R Leite, Fabricio Enembreck, and Jean-Paul A Barthes. Distributed constraint optimization problems: Review and perspectives. *Expert Systems with Applications*, 41(11):5139–5157, 2014.

Daniel Levine, Brandon Luders, and Jonathan How. Information-theoretic motion planning for constrained sensor networks. *Journal of Aerospace Information Systems*, 10(10):476—-496, 2013. ISSN 2327-3097.

Daniel S Levine. Information-rich path planning under general constraints using rapidly-exploring random trees. Master's thesis, Citeseer, 2010.

Stuart Lloyd. Least squares quantization in PCM. *IEEE transactions on information theory*, 28(2):129–137, 1982.

Kian H Low, John M Dolan, and Pradeep Khosla. Adaptive multi-robot wide-area exploration and mapping. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1*, pages 23–30. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

Roman Marchant and Fabio Ramos. Bayesian optimisation for intelligent environmental monitoring. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2242–2249. IEEE, 2012.

Alexandra Meliou, Andreas Krause, Carlos Guestrin, and Joseph M Hellerstein. Nonmyopic informative path planning in spatio-temporal models. In *AAAI*, volume 10, pages 16–7, 2007.

Luis Merino, Fernando Caballero, J Ramiro Martínez-de Dios, Joaquin Ferruz, and Aníbal Ollero. A cooperative perception system for multiple UAVs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23 (3-4):165–184, 2006.

Luis Merino, Fernando Caballero, and Anibal Ollero. Active sensing for range-only mapping using multiple hypothesis. In *Proceedings of the 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 37–42, Taipei (Taiwan), October 2010.

Nathan Michael, Michael M Zavlanos, Vijay Kumar, and George J Pappas. Maintaining connectivity in mobile robot networks. In *Experimental Robotics*, pages 117–126. Springer, 2009.

Lauren M Miller and Todd D Murphey. Optimal planning for target localization and coverage using range sensing. In *Automation Science and Engineering (CASE), 2015 IEEE International Conference on*, pages 501–508. IEEE, 2015.

Pratap Misra and Per Enge. *Global Positioning System: Signals, Measurements and Performance Second Edition.* Lincoln, MA: Ganga-Jamuna Press, 2006.

Pragnesh J Modi, Wei-Min Shen, Milind Tambe, and Makoto Yokoo. ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1):149–180, 2005.

Mohd Murtadha Mohamad, Nicholas K Taylor, and Matthew W Dunnigan. Articulated robot motion planning using ant colony optimisation. In *Intelligent Systems, 2006 3rd International IEEE Conference on*, pages 690–695. IEEE, 2006.

Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, 2004.

L Srikar Muppirisetty, Tommy Svensson, and Henk Wymeersch. Spatial wireless channel prediction under location uncertainty. *IEEE Transactions on Wireless Communications*, 15(2):1031–1044, 2016.

Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan S Muhammad. RRT*-SMART: A rapid convergence implementation of RRT*. *International Journal of Advanced Robotic Systems*, 10, 2013.

Joseph L Nguyen, Nicholas RJ Lawrance, Robert Fitch, and Salah Sukkarieh. Real-time path planning for long-term information gathering with an aerial glider. *Autonomous Robots*, pages 1–23, 2015.

Cynthia Nikolai and Gregory Madey. Tools of the trade: A survey of various agent based modeling platforms. *Journal of Artificial Societies and Social Simulation*, 12(2):2, 2009. ISSN 1460-7425. URL `http://jasss.soc.surrey.ac.uk/12/2/2.html`.

nytimes. Six years after Fukushima, robots finally find reactors melted uranium fuel, 2017. URL `goo.gl/ebSRB3`.

Ruofei Ouyang, Kian H Low, Jie Chen, and Patrick Jaillet. Multi-robot active sensing of non-stationary Gaussian process-based environmental phenomena. In *Proceedings of the 2014 international conference on Autonomous agents and* *multi-agent systems*, pages 573–580. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

Mark Owen, Randal W Beard, and Timothy W McLain. Implementing Dubins airplane paths on fixed-wing UAVs. In *Handbook of Unmanned Aerial Vehicles*, pages 1677–1701. Springer, 2015.

Timothy Patten, Robert Fitch, and Salah Sukkarieh. Large-scale near-optimal decentralised information gathering with multiple mobile robots. In *Proceedings of the Australasian Conference on Robotics and Automation*, 2013.

Sven M Persson and Inna Sharf. Sampling-based A* algorithm for robot path-planning. *The International Journal of Robotics Research*, 33(13):1683–1708, 2014.

Adrian Petcu and Boi Faltings. A scalable method for multiagent constraint optimization. In *Internatioanl Joint Conference on Artificial Intelligence.* Citeseer, 2005.

Mihail Pivtoraiko, Ross A Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26 (3):308–333, 2009.

Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

Joaquin Quiñonero-Candela and Carl E Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959, 2005.

J-P Ramirez-Paredes, Emily A Doucette, J Willard Curtis, and Nicholas R Gans. Optimal placement for a limited-support binary sensor. *IEEE Robotics and Automation Letters*, 1(1):439–446, 2016.

Carl E Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2005.

Alessandro Renzaglia, Christophe Reymann, and Simon Lacroix. Monitoring the evolution of clouds with UAVs. In *IEEE International Conference on Robotics and Automation*, 2016.

Markusand Rickert, Arne Sieverling, and Oliver Brock. Balancing exploration and exploitation in sampling-based motion planning. *IEEE Transactions on Robotics*, 30(6):1305–1317, December 2014.

NJ Robinson, PC Rampant, APL Callinan, MA Rab, and PD Fisher. Advances in precision agriculture in south-eastern Australia. ii. spatio-temporal prediction of crop yield using terrain derivatives and proximally sensed data. *Crop and Pasture Science*, 60(9):859–869, 2009.

Havard Rue and Leonhard Held. *Gaussian Markov random fields: theory and applications*. CRC press, 2005.

Lorenzo Sabattini, Nikhil Chopra, and Cristian Secchi. Decentralized connectivity maintenance for cooperative control of mobile robotic systems. *The International Journal of Robotics Research*, 32(12):1411–1423, 2013.

Oren Salzman and Dan Halperin. Asymptotically near-optimal RRT for fast, high-quality, motion planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 4680–4685. IEEE, 2014.

Oren Salzman and Dan Halperin. Asymptotically-optimal motion planning using lower bounds on cost. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 4167–4172. IEEE, 2015.

Nicola Santoro. *Design and analysis of distributed algorithms*, volume 56. John Wiley & Sons, 2006.

Zeynep G Saribatur, Esra Erdem, and Volkan Patoglu. Cognitive factories with multiple teams of heterogeneous robots: Hybrid reasoning for optimal feasible global plans. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2923–2930, 2014. ISBN 9781479969340.

Michael C Shewry and Henry P Wynn. Maximum entropy sampling. *Journal of applied statistics*, 14(2):165–170, 1987.

Thierry Siméon, J-P Laumond, and Carole Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

Thierry Siméon, Stéphane Leroy, and J-P Lauumond. Path coordination for multiple mobile robots: A resolution-complete algorithm. *IEEE Transactions on Robotics and Automation*, 18(1):42–49, 2002.

Amarjeet Singh, Andreas Krause, Carlos Guestrin, and William J Kaiser. Efficient informative sensing using multiple robots. *Journal of Artificial Intelligence Research*, pages 707–755, 2009.

Amarjeet Singh, Fabio Ramos, Hugh D Whyte, and William J Kaiser. Modeling and decision making in spatio-temporal processes for environmental surveillance. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 5490–5497. IEEE, 2010.

Krzysztof Socha and Marco Dorigo. Ant colony optimization for continuous domains. *European journal of operational research*, 185(3):1155–1173, 2008.

Matthijs TJ Spaan, Tiago S Veiga, and Pedro U Lima. Decision-theoretic planning under uncertainty with information rewards for active cooperative perception. *Autonomous Agents and Multi-Agent Systems*, 29(6):1157–1185, 2015.

Michael L Stein. *Interpolation of spatial data: some theory for kriging.* Springer Verlag, 1999.

Ruben Stranders, Alex Rogers, and Nicholas R Jennings. A decentralised online coordination mechanism for monitoring spatial phenomena with mobile sensors. In *Workshop 15: Agent Technology for*, page 9. Citeseer, 2008.

Ruben Stranders, Alessandro Farinelli, Alex Rogers, and Nicholas R Jennings. Decentralised coordination of mobile sensors using the max-sum algorithm. In *Proceedings of the 21st international jont conference on Artifical intelligence*, pages 299–304. Morgan Kaufmann Publishers Inc., 2009.

Ruben Stranders, Francesco M D Fave, Alex Rogers, and Nicholas R Jennings. A decentralised coordination algorithm for mobile sensors. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 874–880. AAAI Press, 2010.

Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. `http://ompl.kavrakilab.org`.

Ildiko Suveg and George Vosselman. Reconstruction of 3D building models from aerial images and maps. *ISPRS Journal of Photogrammetry and remote sensing*, 58(3):202–224, 2004.

J Swigart and S Lall. An explicit state space solution for a decentralized two-player optimal linear-quadratic regulator. In *Proc. of American Control Conference (ACC.2010)*, volume 1, pages 6385–6390, 2010.

Onur Tekdas, Deepak Bhadauria, and Volkan Isler. Efficient data collection from wireless nodes under the two-ring communication model. *The International Journal of Robotics Research*, 31(6):774–784, 2012.

theguardian. Fukushima nuclear plant blast puts Japan on high alert, 2011. URL `goo.gl/gHrZuL`.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics.* MIT press, 2005.

Chris Urmson and Reid G Simmons. Approaches for heuristically biasing RRT growth. In *IROS*, volume 2, pages 1178–1183, 2003.

Alberto Viseras and Calin Olariu. A general algorithm for exploration with Gaussian processes in complex, unknown environments. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3388–3393. IEEE, 2015.

Alberto Viseras, Zhe Xu, and Luis Merino. Distributed multi-robot cooperation for information gathering under communication constraints. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*, (accepted) 2017.

Alberto Viseras, Dmitriy Shutin, and Luis Merino. Online information gathering using RRT-based planners and GPs. *Robotics and Autonomous Systems*, (under review) 2017a.

Alberto Viseras, Zhe Xu, and Luis Merino. Distributed multi-robot information gathering under complex constraints. *Autonomous Robots*, (under review) 2017b.

Alberto Viseras, Michael Angermann, Iris Wieser, Martin Frassl, and Joachim Mueller. Efficient multi-agent exploration with Gaussian processes. In *Robotics and Automation (ACRA), 2014 Australasian Conference on*, 2014.

Alberto Viseras, Rafael Ortiz Losada, and Luis Merino. Planning with ants: Efficient path planning with rapidly exploring random trees and ant colony optimization. *International Journal of Advanced Robotic Systems*, 13(5): 1729881416664078, 2016a.

Alberto Viseras, Thomas Wiedemann, Christoph Manss, Lukas Magel, Joachim Mueller, Dmitriy Shutin, and Luis Merino. Decentralized multi-agent exploration with online-learning of Gaussian processes. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 4222–4229. IEEE, 2016b.

Alberto Viseras, Valentina Karolj, and Luis Merino. An asynchronous distributed constraint optimization approach to multi-robot path planning with complex constraints. In *Proceedings of the Symposium on Applied Computing*, pages 268–275. ACM, 2017a.

Alberto Viseras, Dmitriy Shutin, and Luis Merino. Online information gathering using sampling-based planners and GPs: An information theoretic approach. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 123–130, Sept 2017b.

Glenn Wagner and Howie Choset. Subdimensional expansion for multirobot path planning. *Artificial Intelligence*, 219:1–24, 2015. ISSN 00043702. URL `http://dx.doi.org/10.1016/j.artint.2014.11.001`.

Shangxing Wang, Andrea Gasparri, and Bhaskar Krishnamachari. Robotic message ferrying for wireless networks using coarse-grained backpressure control. *IEEE Transactions on Mobile Computing*, 16(2):498–510, 2017.

Changyun Wei, Koen V Hindriks, and Catholijn M Jonker. Multi-robot cooperative pathfinding: A decentralized approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 8481 LNAI(PART 1):21–31, 2014. ISSN 16113349.

Wikipedia, the free encyclopedia. Venn diagram, 2017. URL `goo.gl/svyMzK`. [Online; accessed August 21, 2017].

Ryan K Williams and Gaurav S Sukhatme. Probabilistic spatial mapping and curve tracking in distributed multi-agent systems. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1125–1130. IEEE, 2012.

Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 146–151. IEEE, 1997.

Kwangjin Yang, Seng Keat Gan, and Salah Sukkarieh. A Gaussian process-based RRT planner for the exploration of an unknown and cluttered environment with a UAV. *Advanced Robotics*, 27(6):431–443, 2013.

Peng Yang, Randy A Freeman, Geoffrey J Gordon, Kevin M Lynch, Siddhartha S Srinivasa, and Rahul Sukthankar. Decentralized estimation and control of graph connectivity for mobile sensor networks. *Automatica*, 46(2):390–396, 2010.

Yiqun Dong. What's the difference between RRT and RRT* and which one should we use, 2015. URL `goo.gl/6nIkL7`. [Online; accessed August 22, 2017].

Michael M Zavlanos, Magnus B Egerstedt, and George J Pappas. Graph-theoretic connectivity control of mobile robot networks. *Proceedings of the IEEE*, 99(9):1525–1540, 2011.

Weixiong Zhang, Zhao Xing, Guandong Wang, and Lars Wittenburg. An analysis and application of distributed constraint satisfaction and optimization algorithms in sensor networks. In *AAMAS*, volume 3, pages 185–192, 2003.

Yu Zhang, Kangjin Kim, and Georgios Fainekos. *Discof: Cooperative pathfinding in distributed systems with limited sensing and communication range*. Springer, 2016. ISBN 978-3-642-32722-3. URL `goo.gl/4Bm4vZ`.

Zhen Ziyang, Gao Chen, Zhao Qiannan, and Ding Ruyi. Cooperative path planning for multiple UAVs formation. In *The 4th Annual IEEE International Conference on Cyber Technology in Automation, Control and Intelligent Systems*, volume 210016, pages 469–473, 2014. ISBN 9781479936694.