DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

DLR

Development and Validation of a Hardware-in-the-Loop System for Robotic Seismic Exploration

Bachelorarbeit

für die Prüfung zum

Bachelor of Science

des Studiengangs Informationstechnik

der Dualen Hochschule Baden-Württemberg Mannheim

von

Luis Wientgens

Abgabe am 30. August 2022

Bearbeitungszeitraum:	07.06.21 – 30.08.22
Matrikelnummer, Kurs:	3135171, TINF19IT1
Abteilung:	Institut für Kommunikation und Navigation
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt e.V.
Betreuer der Ausbildungsfirma:	Dr.-Ing. Ban-Sok Shin
Gutachter der Dualen Hochschule:	Prof. Dr. Holger Gerhards

Erklärung

Ich versichere hiermit, dass ich meine Bachelorarbeit mit dem

Thema

Development and Validation of a Hardware-in-the-Loop System for Robotic Seismic Exploration

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.*

* falls beide Fassungen gefordert sind

Weßling, den 30. August 2022

Abstract

The exploration of the subsurface of planetary bodies in our solar system is a promising approach for future space missions. Until now planetary seismology has been constrained to a single or very few seismic stations. Recent proposals suggest the use of a swarm of robotic rovers carrying seismic sensors for the use in this exploration as they offer higher flexibility due to their ability to relocate. To this end in this thesis a hardware in the loop system for the simulation of seismic surveys is designed and introduced. The system is built on top of the second version of the Robot Operating System and is designed to be distributed and extensible. Verification experiments are conducted in simulation and using real robotic rovers.

Zusammenfassung

Die Erkundung des Inneren von Planeten und planetaren Körpern in unserem Sonnensystem ist ein vielversprechender Ansatz für zukünftige Erkundungsmissionen. Bis jetzt war Seismologie auf fremden Himmelskörpern auf eine oder wenige Messstationen beschränkt. Aktuelle Missionskonzepte schlagen zu diesem Zweck die Verwendung von robotischen Rovern augestattet mit seismischen Sensoren vor, da diese sich repositionieren können. In diesem Sinne wird in dieser Arbeit ein System zur Hardware-in-the-Loop-Simulation von seismischen Kampagnen entworfen und vorgestellt. Das System baut auf der zweiten Version des Robot Operating System auf und ist auf Verteiltheit und Erweiterbarkeit ausgerichtet. Zur Verifikation werden Experimente als Simulation und mit realen Rovern ausgeführt.

Contents

List of Figures

List of Tables

List of Listings and Algorithms

List of Symbols

u	function of $\mathbf{x} \in \mathbb{R}^3$, t
x	first spatial direction in the exploration domain
y	second spatial direction in the exploration domain
z	depth below surface
t	time
∇	nabla operator
c	wave propagation velocity
q	function for the wave propagation velocity at $\mathbf{x} \in \mathbb{R}^3$
D_d	finite difference operator for dimension $d \in (x, y, z)$
T	number of timesteps in the simulation
x_r	reciever offset from source
t_0	offset corrected traveltime
v_{NMO}	normal moveout velocity
v_l	seismic layer velocity
h	seismic layer height

Acronyms

HITL	**H**ardware **I**n **T**he **L**oop
API	**A**pplication **P**rogramming **I**nterface
ROS	**R**obot **O**perating **S**ystem
DDS	**D**ata **D**istribution **S**tandard
PDE	**P**artial **D**ifferential **E**quation
NMO	**N**ormal **Mo**veout
MAPF	**M**ulti **A**gent **P**ath **F**inding
IDL	**I**nterface **D**escription **L**anguage
STA - LTA	**S**hort **T**erm **A**verage - **L**ong **T**erm **A**verage

1 Introduction

One of the most fascinating and prominent aspects of space research is the exploration of foreign planetary bodies. Currently state of the art planetary exploration is mostly focused on investigating physical properties visible from orbit or accessible for exploration by a single lander. Examples include the recent (2021) landing of the Perseverance rover as part of the Mars 2020 mission [1] and the European orbiters MarsExpress [2] and BepiColombo [3]. Therefore missions are mostly constrained to properties above and on the planetary surface while subsurface characteristics of a body remain hidden. One major field of interest in future planetary missions is therefore the seismic exploration of subsurface structures on our nearest panetary neighbor, the red planet Mars [4]. First steps in the direction of planetary seismology were conducted during the moon landings of the Apollo program [5]. A few years later for the first time a seismometer was placed on Mars carried by the Viking lander, however due to problems in deployment the data proved to be of limited usability [6]. More recently in the context of the InSight mission to mars, a single seismometer was deployed and used for passive seismic inferometry [7]. Data from this seismometer was used to gain information about the upper mantle structure of mars [8].

1.1 Motivation

The number of seismometers that will be available for exploration on planetary surfaces is likely to be significantly lower than the number used for surveys on earth, resulting from the challenges imposed by mass constraints and planetary landings.

Furthermore there are no human operators available to manually reposition the seismometers for the concstruction of new recording grids. Therefore the spatial resolution of the data that can be obtained is constituting a limiting factor. One strategy to mitigate this challenge is the use of robotic rovers that can freely move and reposition themselves as a platform for seismometers. Possibilities of this approach include the emulation of bigger recording grids by repositioning of the rovers during a measuremnt while using the same seismic source and adapting the sampling topologie in an irregular fashion to further inspect discovered features of interest. In line with this approach, recently a swarm robotics based system for seismic planetary exploration has been proposed by the German Aerospace Center [9]. As a step towards that vision in this thesis a system for Hardware-in-the-Loop (HITL) simulation of robotic seismic exploration strategies using multiple robotic agents is developed and evaluated.

1.2 Requirements Definition

The system developed over the course of this thesis has multiple design goals, which are stated and explained as follows.

- Accuracy - The simulation components should approximate the real world conditions to a reasonable degree
- Extensibility - The system should scale with varying numbers of agents and should allow for the use of different algorithms over well defined interfaces (APIs).
- Adaptability - The system should provide the possibility of freely adaptable sampling topologies
- Ready for autonomy - The system should allow for the seamless addition of autonomous operation to replace the human input factor

1.3 Chapter Outline

Starting out with Chapter 2 - Robot Operating System and Hardware we give an overview of the robot hardware in use as well as the relevant concepts of the Robot Operating System. The capabilities and constraints of the robots will be introduced.

Chapter 3 - Seismic Exploration and Inversion outlines the basic elements of seismic exploration in general and seismic inversion using normal moveout correction as a modeling technique in particular. Relevant mathematical concepts and notation are described.

The high level design of the presented system is the topic of Chapter 4 - System Design where the substructures and the communication flow as well as the design considerations are described.

In Chapter 5 - Implementation we continue on the software level with implementation details, algorithms and challenges during the implementation process.

Verfication of the system is based on a series of experiments conducted in simulation and with real hardware in the laboratory. Chapter 6 - Verification Exmperiments and Evaluation documents the setting and results of these experiments. The described results are critically evaluated, especially with respect to the design goals formulated in this introduction.

Proceeding in Chapter 7 - Summary we summarize the work that has been done over the course of the thesis and the accompanying results.

Finally the thesis is concluded in Chapter 8 - Outlook where we discuss the results of the thesis in the bigger picture of the surrounding scientific applications and possible future developments.

2 Robot Operating System and Hardware

The Robot Operating System (ROS) is a free and open-source software project that is often described as a meta operating system (meta with respect to conceptually being on a higher abstraction level as and running on top of the regular operating system) for robotic systems [10][11]. More specifically it is a set of libraries built upon a middleware layer that handels communication between a distributed set of nodes that in their entirety comprise the system. Additionally a collection of tools for building and distributing robot software based on ROS is part of its ecosystem. The development of ROS originally started at Stanford Univeristy [12] but is now supervised by the Open Source Robotics Foundation [13].

Development of ROS follows a distribution based scheme that approximately matches the release cycle of Ubuntu Linux. There are however two distinct versions of ROS namely the original ROS and the younger ROS2 project that introduces significant changes and upgrades, especially with respect to real time capability and fault tolerance. In this thesis the `Galactic` release of ROS2 is used and for the remainder of this work the usage of the acronym ROS refers to this version. For further information in regard to the differences between ROS1 and ROS2 the reader is referred to [14].

2.1 Middleware

ROS is built on top of a middleware layer that implements the Data Distribution Service (DDS)[15] standard. DDS has been specified as a framework for realizing transparent communication in distributed systems while beeing robust and reliable enough to be employed in safety critical and real time systems. Because DDS transparently communicates over physically distinct computers, the task separation induced by the nodes can easily be translated to a physically distributed system and in extension swarm systems like the presented application using multiple rovers. DDS is well suited as a middleware layer for ROS as it is assuming the same publish/subscribe principle for message passing that ROS is based on. This concepts are described in the next two sections.

2.2 Nodes

The fundamental building block of every ROS system is the set of its nodes. A node is the ROS concept to describe a self-contained programm unit that can interact with other nodes by publishing and subscribing to ROS messages. A ROS message defines the structure of the data to be exchanged between communicating nodes and can be composited to represent custom data types. Nodes are implemented by extending the *Node* class exposed by the API of the ROS client library [16].

Conceptually each node should have a well defined task in the context of the system and decouple this task from the other nodes. There is no shared state or environment as all relevant data is exchanged by using the DDS middelware for message passing and the communication is entirely asynchronous.

2.3 Communication Methods

Topics, Services and Actions are the concepts that ROS exposes to implement the distributed communication pattern in a system of nodes. They are communication

channels that are used for the transmission of messages. Although they are similar in nature and share concepts they are each used for a particular purpose.

2.3.1 Topics

Topics are the method of communication that is most true to the basic publish/subscribe pattern of the DDS middleware layer. They are used for stateless transmission of messages over the channel identified by the respective topic name. Multiple nodes can publish to or subscribe to the same topic, making it a many-to-many communication method.

2.3.2 Services

In contrast to topics, services are specialized on one to one communication and provide a request/response pattern. A service consists of a request by the client and a response by the server each one being a node. They are distinct from using topics as a service client only recieves a response when specifically requested opposed to whenever a publisher decides to send data.

2.3.3 Actions

Actions are a communication method for issuing long running tasks with the option for cancelation and intermediate progress updates. Conceptually they are intended to implement functionalities of a ROS system that influence real world actions, especially actuator commands, that require the possibility to be interupted during execution.

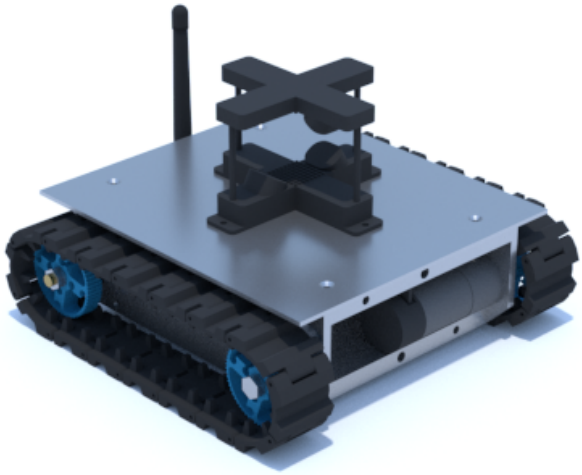

Figure 2.1: 3D model of a *dwarf* rover (modeled by swarm exploration group at DLR)

2.4 Robots

The robots used in this study are track-based rovers that have been custom-built by the Swarm Exploration Group of the Institute of Communication and Navigation at the German Aerospace Center. They are equipped with a Raspberry Pi running nodes in a ROS environment and are connected to the network over onboard WIFI. The rover can be manipulated by sending velocity commands to the respective *cmd_velocity* topic to control the speed of the two onboard motors that each drive one of the tracks. Nicknamed *dwarf*, a rendering of the rover can be seen in Figure 2.1. Apart from monitoring of battery voltage the rovers itself do not provide sensor data and do not have the capability of positioning themselfes or recognize nearby objects. Positioning is instead provided globally by a Vicon Tracking System [17] that identifies each rover by a unique pattern of reflectors and publishes the positional data at a rate of 100 Hz using a ROS node as a bridge between the Vicon software and the system. Vicon systems are commonly used to provide ground truth position information in robotics research and can provide positioning with a mean error lower than 2 mm for moving objects [18]. Each rover is equipped with a platform that can be used to optionally mount external sensors. As in this study the data recording is numerically simulated no sensors are mounted on the robots.

3 Seismic Exploration and Inversion

Seismic exploration as a field of research is concerned with the goal of discovering and modelling underground structures by using acoustic waves. Mathematical models of the physics of wave propagation are applied in order to gain information on properties of subsurface structures by using data recorded by sensors on the surface, hence it can be seen as comparable to other remote sensing methods like ground penetrating radar based on electromagnetic waves [19].

3.1 Overview of Seismic Reflection

As the first step of any seismic exploration survey measurements must be taken to record the relevant data in the process of data aquisition. Here the method of active seismic reflection surveying is fundamental for gathering the required seismic data. In such a seismic survey receivers in the form of geophones or seismometers record the ground motion or acceleration using either a spring damped mass in a magnetic field to translate movement into electric signals or microelectromechanical systems that function as accelerometers [19]. They are placed on the surface over the exploration domain in a formation that is called the recording or sampling topology. In reflection surveys the topologies are often regular with the stations placed along a line or in a grid formation. An active seismic source (e.g. explosives) is used to generate a seismic shock and cause the released energy to propagate trough the subsurface. The resulting data that represents the ground movement or acceleration at each receivers position over time constitutes a seismogram. Another term for the recording of the seismic wave at a specific position is trace with a survey recording multiple traces per seismic shock which is also called shot [20].

Similar to optics seismology uses both the model of waves and the model of rays to describe propagation of acoustic energy in planetary bodies. While both models describe the same underlying physical phenomenon, they allow for different approaches of imaging the physical parameters of the subsurface. Rays model the propagation path of energy while waves model the displacement or pressure as a function of location and time. Seismic rays travel orthogonal to the corresponding wavefront of the propagating acoustic wave [20]. While seismic reflection uses the ray model to interpret the propagation in the subsurface, the resulting recorded seismograms are representations of acoustic waves and therefore the wave model is relevant for their interpretation and also their simulation as part of the HITL system. Additionally there is a further distinction between longitudinal and transversal waves which are also called pressure (P-) and shear (S-) waves respectively [19].

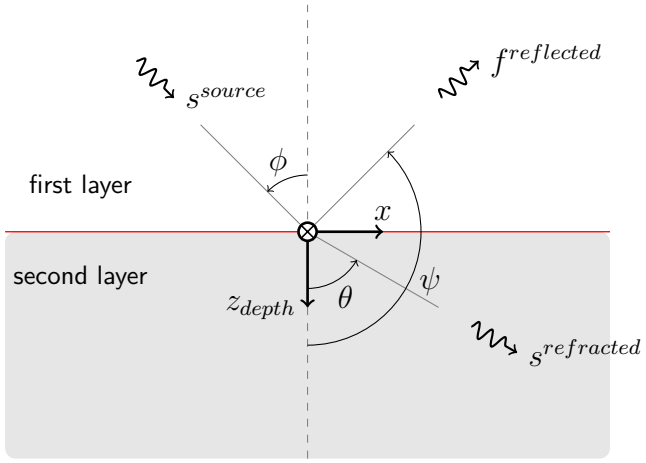

Figure 3.1: Reflection and refraction of a seismic wave at an interface between two layers of differing velocity

The underlying assumption of seismic surveys is that different materials have distinct physical properties that affect the propagation of seismic waves. In this context the most important property is the seismic velocity of the material, meaning the speed of acoustic waves propagating trough it. By gathering information about the time it takes for a signal to travel from the source to a receiver, information about the seismic velocity and therefore the corresponding material can be gained. Planetary subsurfaces are approximately comprised of layers of geological material

with their differing seismic velocities and anomalies like gas enclosures, faults or caves where the seismic velocities additionally differs from the surrounding material [19]. The boundaries between this layers are called interfaces. The velocity change that correspond to a change in the acoustic impedance of the material causes seismic waves to be reflected and refracted at this boundaries which leads to a change in direction of the corresponding raypaths. Figure 3.1 illustrates the reflection and refraction of a seismic wave at an interface. While the refraction angle θ is dependent on the ratio of velocities of the two layers, the reflection angle ψ can be expressed as $180° - \phi$ and is therefore dependent on the angle of incidence relative to the layer normal [19]. For each interface the reflection causes a subseqent arrival of seismic energy at a receiver after the arrival of the direct wave from the source to the respective receiver. In conducting seismic exploration by using reflection the aim is now to use the traveltime for each of the multiple arrivals at the receivers to estimate the depth of layers as well as their velocity in the interpretation of the recorded data.

3.2 Numerical Forward Modeling

Measurements in this thesis are simulated by an existing python program from [21]. The propagation of seismic waves is modeled mathematically by the acoustic wave equation that takes the form of a partial differential equation (PDE). In most general cases it is not possible to give an analytical solution to PDEs which requires approximate solving by numeric computation [22][23]. The general wave equation for a three dimensional simulation domain Ω takes the form of Equation (3.1) [22]. In this general form u is a function of x, y, z and t giving the amplitude of the wave which in the context of the thesis denotes the acoustic pressure at a given position and time and caused by the seismic wave. The squared nabla operator ∇ indicates the partial differentiation of u in each spatial dimension resulting in Equation (3.2). The dimensions x and y are assigned to correspond to the dimensions of the surface plane with z denoting the depth. There is also a dependency on the propagation velocity of the wave c.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u \quad t \in (0, T] \tag{3.1}$$

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \quad | \quad x, y, z \in \Omega \tag{3.2}$$

The general equation as given above assumes a static propagation velocity over the whole domain. For simulation of subsurfaces with seismic velocitys that vary w.r.t. to the position in space this model is to simple and the equation needs to be extended to include variable coefficients in the form of a function q for the propagation velocity at each position in the domain [22].

For the numerical computation the equation needs to be brought in a discrete form using finite differences to approximate the differentials. Therefore the partial differential is replaced by a finite difference operator D_d for the corresponding dimension d. The calculation of the finite differences in the spatial dimensions require the consideration of the neighbouring values in each direction. Therefore the computation requires a discretized seismic velocity model as input. A regular grid with fixed offsets $\Delta x = \Delta y = \Delta z$ between the grid points is used where the velocity function q is defined for each grid point. The discrete notation is shown in Figure 3.2. A similar model discretization has already been used in a previous work that implemented a parallel solver for a high frequency approximation of the wave equation [24].

Now the wave equation can be given in discrete form for the cases of uniform velocity $q = c \quad \forall \mathbf{x} \in \Omega$ as Equation (3.3) and for varying velocities as Equation (3.4) [22].

$$[D_t D_t u = c^2 (D_x D_x u + D_y D_y u + D_z D_z u) + f]^n_{i,j,k} \tag{3.3}$$

$$[\rho D_t D_t u = (D_x q D_x u + D_y q D_y u + D_z q D_z u) + f]^n_{i,j,k} \tag{3.4}$$

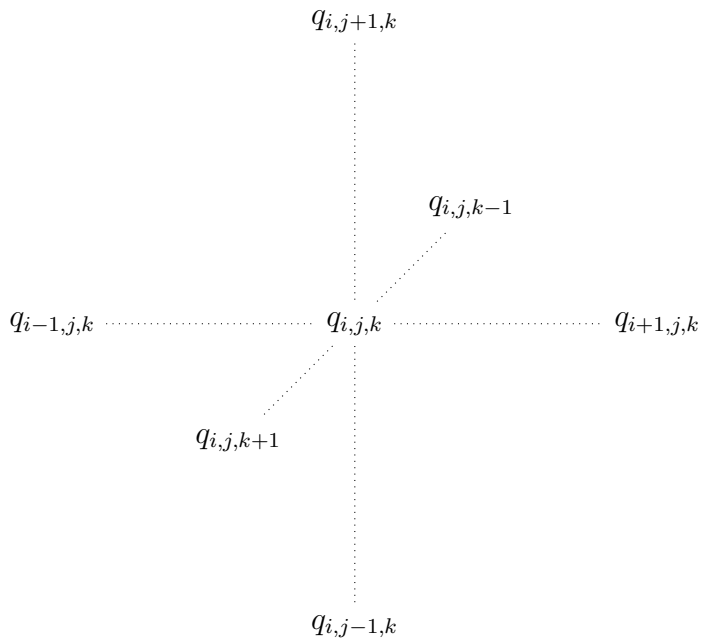

Figure 3.2: Notation used in the finite difference grid model for numerical solution of the wave equation

The discrete equation is solved iteratively at each point for each time step n resulting in the wavefield. For each iteration the solutions for the preceeding timesteps need to be known which means the wavefield for $t = 0$ needs to be instantiated with initial conditions that set the values for the solution at timestep $t_n = 0$. This initial condition includes setting a source term at the location of the source. Also on the edges of the domain damping boundary conditions need to be defined to specify the value of neighbours for grid points on the boundary and to prevent the boundaries from introducing additional reflections [25].

The resulting wavefield takes the form of a four-dimensional array of length T that holds a field of values at each spatial position corresponding to the result of the wave equation at this position over time for each timestep t_n. For each position on the surface this gives the simulated seismogram for a receiver at that position.

It is important to be aware of the limits of this numerical simulation. Multiple sorts of errors that are well known in numerical analysis are introduced, including the

discretization error and the error due to machine precision [23]. Further the resulting simulated seismograms are noise free which is a severe simplifictation compared to real seismograms that always contain noise added to the signal [26].

3.3 Inversion by Normal Moveout Correction

Seismic inversion is a set of methods that has the goal of estimating the physical parameters of the subsurface by using the information obtained in a seismic survey. In this thesis the data gathered by the simulated seismic reflection survey is inverted by using the method of normal moveout correction (NMO) [27] for estimation of layer depth and velocity.

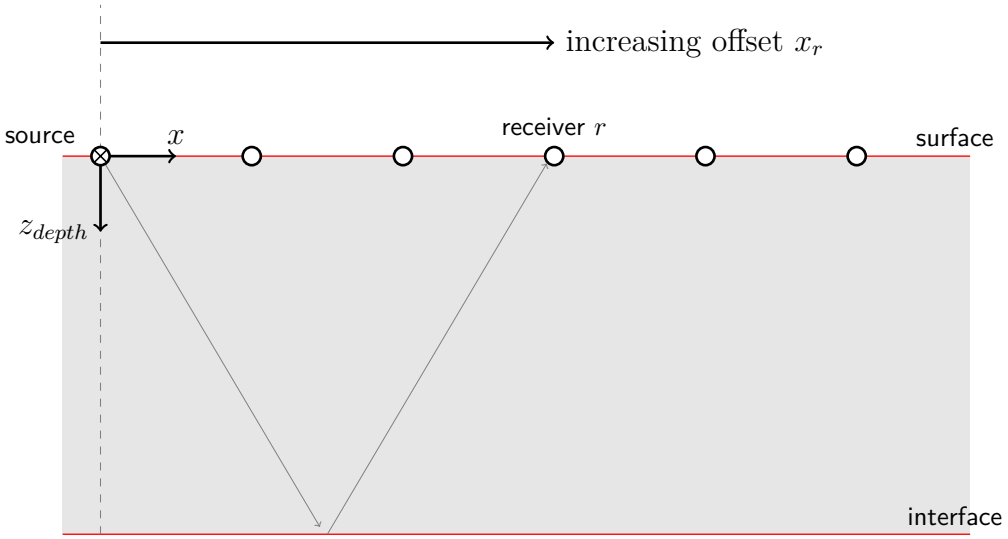

Figure 3.3: Schematic illustration of receiver placements and resulting offsets in a seismic reflection survey

In a seismic reflection survey as described in Section 3.1 the receivers are placed in a line which results in each receiver r having an increasing offset x_r from the source position. Seismic waves need to travel an increasingly long distance to reach the receivers placed further out as is illustrated in Figure 3.3. To use the travel time to estimate depth and velocity however the path of the seismic ray should be reflected

by the interfaces at normal incidence. Normal moveout correction mitigates the spatial offset of the receivers to calculate t_0 according to Equation (3.5)

$$t_0^2 = t^2 - \frac{x_r^2}{v_{NMO}^2} \tag{3.5}$$

removing the part of the traveltime attributed to sideways propagation [28]. Here t_0 is the offset corrected arrival time for a reflection at a receiver r at offset x_r with t being the measured arrival time. However in the presented case the equation has a second unknown, namely the velocity v_{NMO}. Therefore one variable has to be estimated for the inversion. As the recording is in a line, for the same reflection the traveltimes get longer with increasing distance from the source. This results in a parabolic curve of the traveltimes that can be used to estimate t_0 using polynomial regression. With t_0 obtained the layer velocity v_l can be estimated by using Equation (3.6)

$$v_l = \frac{x_r}{\sqrt{2t_0\Delta t}} \tag{3.6}$$

with $\Delta t = t - t_0$ for t at x_r [27]. The depth h of the corresponding layer can be calculated by using Equation (3.7).

$$h = v\frac{t_0}{2} \tag{3.7}$$

Information can also be found for multiple layers by considering additional later reflections and taking the root mean square average of the velocity for all preceeding layers and applying Dix's Formula as shown in [19]. The height of the deeper layers can then be found by using the calculated layer velocity and applying Equation 3.7 with t_0 replaced by the traveltime acutally spent traversing the layer. This interval time can be calculated by taking the difference of the t_0 for the top and the base interface of the layer respectively.

4 System Design

The system design is oriented at the distributed nature of the ROS node architecture. The components are not only logically separated but also are able to run on distinct physical computers. On a higher level it can be subdivided into two functional parts, namely the system under test and the simulation components. We first present the general simulation loop before going into more detail on the system under test.

4.1 Hardware-in-the-Loop Testing

Hardware-in-the-Loop (HITL) testing is usually employed in the case that testing a system under real conditions is not possible or feasible for economic or other reasons [29]. System in this context is used to describe a process that depends on external sensory input and acts back on the environment based on this information. Examples range from simple control loops in embedded devices up to more complex and safety critical systems like avionics hardware in aerospace applications [30][31] or control systems in medical devices [32]. To conduct the necessary testing all aspects of the system that are infeasible for real testing need to be physically simulated. Therefore the input data to the hardware system needs to be obtained from the simulation by considering the effects previous output data would have under real conditions. All input and output data is recorded to evaluate the behavior of the system after tests. Applied to this work the hardware ascpect is comprised of the rovers that position themselves at locations where simulated measurements are taken and new positions are fed back to the rovers.

4.2 Simulation Loop

The highest level of the system design is the HITL simulation loop of the aquistion system. Its conceptual function is to perform all steps of a seismic survey. The participating agents start from their inital positions and measurements for this positions are taken from the simulated wavefield. Following the data is processed with an inversion method. Then the system can either exit or reposition the agents in a new configuration for a new iteration of the measurement and processing steps. This high level behaviour is illustrated by Figure 4.1.

4.3 System Components

On a lower level of abstraction the system is divided into three functional components. The measurment system responsible for the simulation of seismic measurements, the inversion system responsible for processing of the measured data and the movement system that interfaces with the rovers and therefore controls the actual hardware. The single components communicate using messages and have no shared state. They can therefore be distributed in a transparent fashion even over multiple computers as long as they can resolve the names of their communication peers which is a characteristic of a distributed system [33]. Additionally this alows for easy refactoring and implementation of new algorithms internal to the components, as the only guarantees they give are their defined message interfaces. Following the presentation of the concept and functionality of each compomnent, the whole architecture is shown in Figure 4.2.

4.3.1 Measurement Component

The measurement components responsibility is to provide seismic measurements for the current positions of the sensors. These measurements can be either simulated or using real sensors. The public facing interface of the measurement component is

therefore a message that includes all relevant information for representing the seismic data. Table 4.1 shows the information that should be included in the communication.

Attribute Name	Explanation
position	sensor position
source	position of the seismic source
duration	length of the duration (s)
data	measurement data

Table 4.1: Interface of measurement component

Additionally the measurement component should be provided with all parameters that are necessary to provide simulated measurements. A numerical simulation of seismic wave propagation is therefore an integral part of this component. Table 4.2 is listing the required parameters.

Parameter Name	Explanation
x	real domain size in x direction (m)
y	real domain size in y direction (m)
z	real domain size in z direction (m)
d	discretization factor (scalar, controls model scaling)
source	position of the seismic source
model	velocity distribution
solver	algorithm choice

Table 4.2: Parameters of measurement component

This component needs to be continously supplied with the current position of the sensors by a localization source to set the position attribute.

4.3.2 Inversion Component

The inversion component is responsible for processing of the aquired measurement data. Its interface is therefore the recieving end of the previously defined measurement message send by the measurement component. The inversion component keeps its own model of the seismic properties of the simulation domain, therefore it requires

some of the same parameters as the measurement component. Table 4.3 shows all the relevant parameters. At the current stage of the system concept the inversion component is only a receiver that writes out its processed results but does not feed back into the system. It is however trivial to feed back messages as it does not affect its internal state.

Parameter Name	Explanation
x	real domain size in x direction (m)
y	real domain size in y direction (m)
z	real domain size in z direction (m)
d	discretization factor (scalar, controls model scaling)
scheme	algorithm choice

Table 4.3: Parameters of inversion component

4.3.3 Movement Component

The movement component controls the robotic agents participating in the system. This includes path planning and communicating with the agents. This component needs to be continously supplied with the current position of the agents by a localization source to be aware of the positions as start points for path planning. Also information about the extend of the domain is required for path planning. Table 4.4 shows the relevant parameters.

Parameter Name	Explanation
x	real domain size in x direction (m)
y	real domain size in y direction (m)
d	discretization factor (scalar, controls resolution of positions)
planner	algorithm choice

Table 4.4: Parameters of movement component

The movement component provides the ability to specify the next positions and to execute the planned paths by supplying them to the robotic agents.

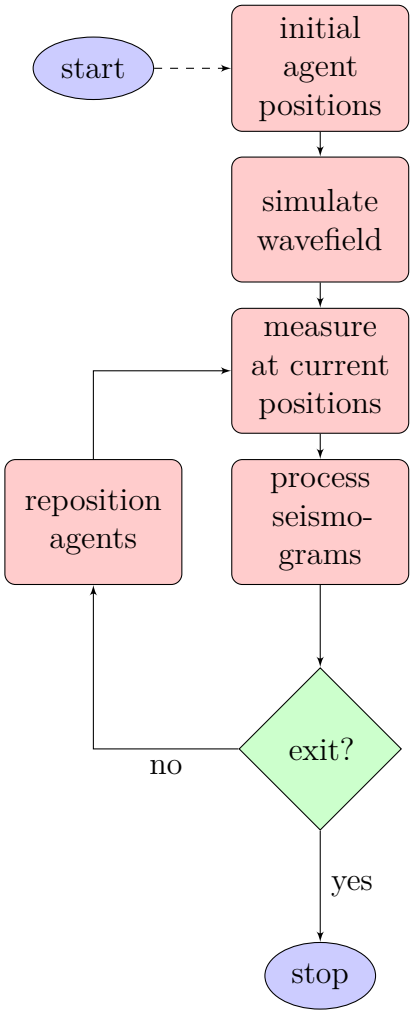

Figure 4.1: High level system flowchart illustrating the logical steps of the HITL-Simulation

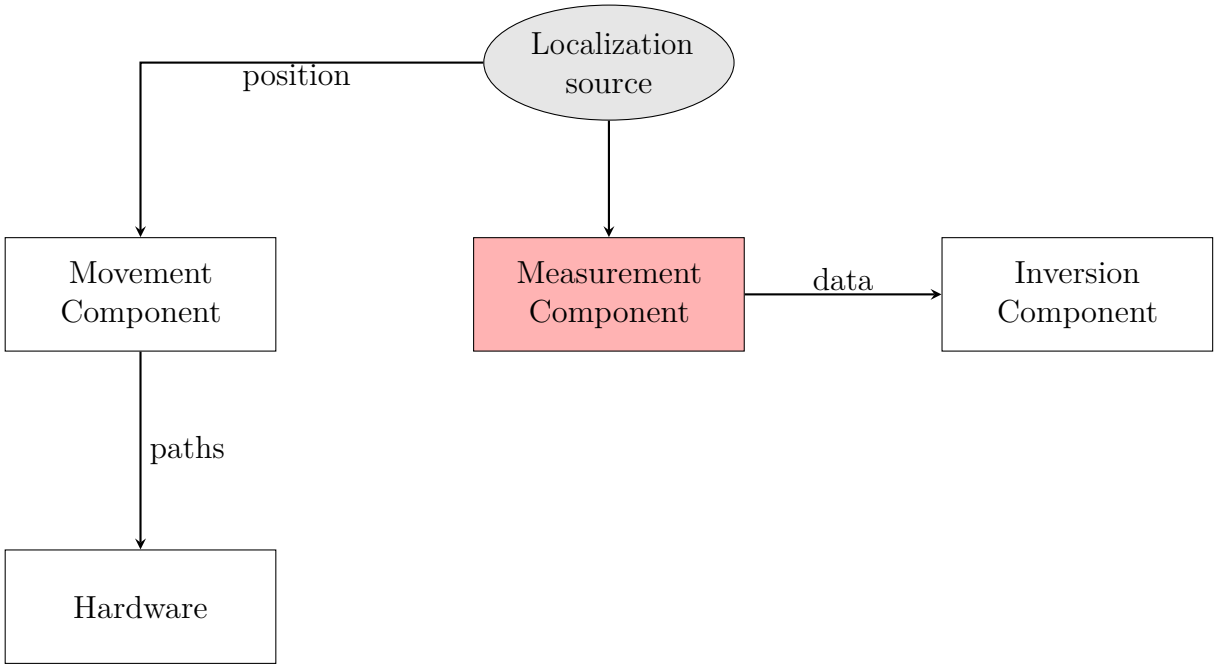

Figure 4.2: Component architecture of the system

5 Implementation

The following chapter will describe how the system design introduced in the preceding chapter is implemented for usage in the experiments. Furthermore the algorithmic implementation for path planning and the normal moveout correction is presented in more detail.

5.1 ROS Nodes

To implement the system design, the component architecture needs to be mapped to the concepts of the ROS architecture. Each component is implemented as one or multiple ROS nodes. Additionally a managment interface is implemented to allow control of the system. For visualization of the generated data this interface also contains a node responsible for plotting the results. The full system is then constructed from all running nodes. The combination of running nodes and their communication channels is called rosgraph in the context of ROS. A visualization off the full rosgraph [1] is shown in Figure 5.1. Each node is an executable inside a ROS package that can be used to further organize the node hierachy. Packages live inside a workspace that bundles a specific functionality or logical entity like the system in this thesis. Packages can also contain additional material like configuration files and additional librarys and modules. Following is a presentation of all packages and their nodes detailing their implementation and features. Prior to that the simple plug-in system that is used to allow for the integration of different algorithms for relevant functionality is introduced.

[1] For a system with 3 simulated rovers, hence all rovers are embedded in a simulated vicon namespace.

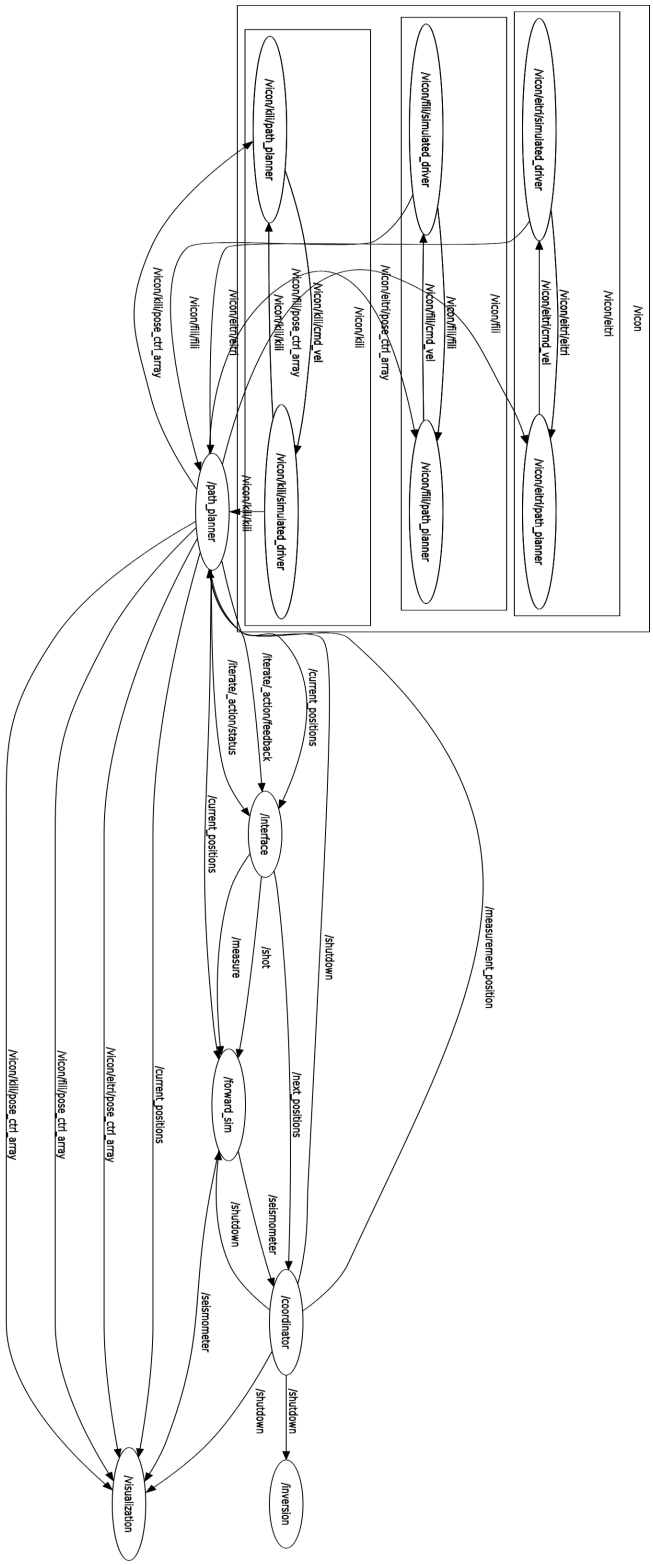

Figure 5.1: Full rosgraph of the running system showing the nodes and their communication channels

5.1.1 Plugin Architecture

One of the formulated goals of the system is the extensibility w.r.t. the integration of new algorithms. To this end a simple plug-in architecture has been designed that is used for the algorithms used for path planning, inversion and forward modelling. The approach uses path names to select the appropriate algorithms without using a registration scheme. To include a new algorithm a new folder needs to be created in the plugin directory with a file using a specific name and containing a function having a specific signature as defined by the respective use case.

Inside each package containing a node that implements the plugin architecture, a plugin folder holds all algorithm implementations. Algorithm selection is implemented using the Python importlib library to import the implementation based on its name that can be set as a parameter during system configuration as will be presented in Section 5.2. Figure 5.2 shows the conceptual organisation of the file hierachy as relevant for the plugin architecture. The two consecutive folders using the name of the respective package are thereby a consequence of the ROS package structure.

The interface defined for each plugin use case will be presented in the description of the respective node.

5.1.2 Interface and Coordinator

The interface is comprised of all nodes and scripts that are not part of the system itself but deal with control and data input/output. The coordinator is an additional ROS node injected into the communication flow between the components for control and synchronization purposes.

Interface For manual control of the system a ROS node is implemented that exposes a text based terminal interface. The interface communicates with the relevant nodes using the ROS communication channels to relay the command information. All command options that are exposed to the user are the following:

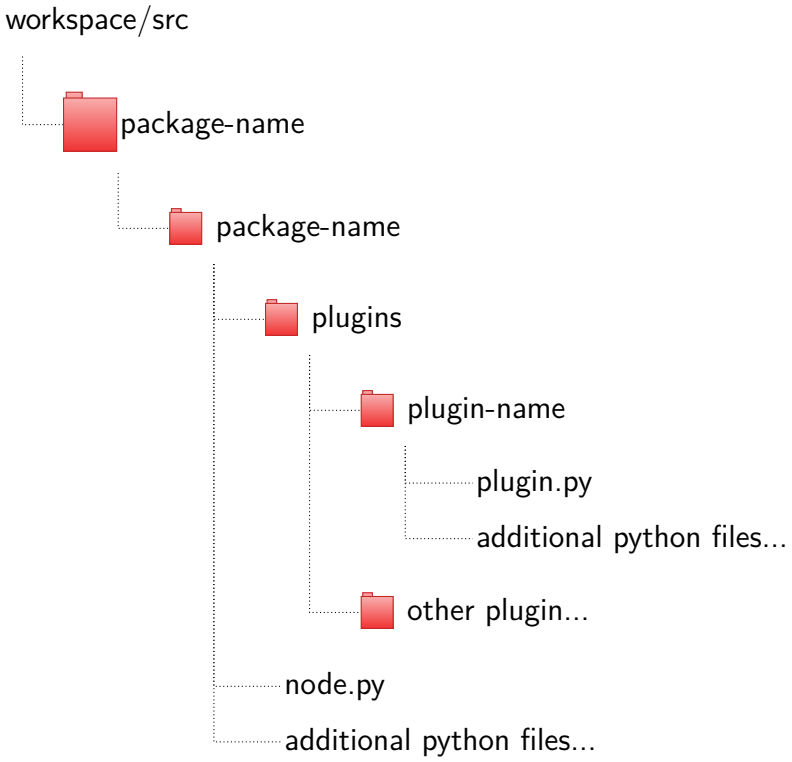

Figure 5.2: File organisation for the plugin architecture

- p - input next positions: input a string of integers separated by "/" representing the coordinates of the next measurement position
- i - iterate: execute the repositioning of the agents
- s - input new shot point: set a new position of the seismic source and regenerate simulated wavefield for this source position
- m - measure: take measurements and execute inversion for current agent positions
- e - exit: save generated data and exit

Visualizer The visualizer is a node implemented for visualization of measurement and path data. It collects the relevant data by subscribing to the topics publishing the agent paths, positions and measuremnt data. Upon recieving the data is plotted and saved at a user specified location.

Coordinator The coordinator node does not have any simulation related functionality but instead relays the requested new positions from the interface to the path planner nodes and keeps track of the number of measurements taken to request the execution of the inversion service of the inversion node. This design decision has been made to decouple the decision of when to execute inversion from the actual implementation of the inversion node.

5.1.3 Path Planner

The implementation of movement control is composed of two levels of path planning and execution. The rovers themselves only take direct velocity commands for control of the two motors. However they also come with a path planner that takes waypoints or list of waypoints in the domain and translates them into the velocity commands required for movement to this waypoints. A waypoint is a message of the ROS message type *Pose*. A Pose is a combination of position and orientation in space. For the purposes of this work we only need the x and y components of the position while

we do not care about rover orientation when the rover has reached its goal position. This path planner for translation of waypoints into motor control commands is from now on reffered to as the low level path planner.

To determine the waypoints the rovers need to pass, a high level path planner is implemented that translates the rover positions and the goals in form of the requested next positions in a list of waypoints for each rover that comprise the individual paths. Each time the agents reposition themselves, they have to move in an environment that has multiple other dynamic agents they could potentially cross paths with. Therefore a strategy is needed to find and execute paths for all agents without agents colliding on their respective path. This problem falls into the category of planning problems and is central to and known as multi agent path finding (MAPF)[34]. A multitude of approaches to this problem have been proposed [35]. One big challenge of MAPF is the size of the state space of the planning problem. The individual state spaces of the agents are combined combinatorically which results in an exponential size increase. Therefore in general a complete and optimal solution to the MAPF problem is an NP-hard problem [35]. Practically usable approaches therefore compromise on either completeness or opitmality of the solution to reduce the complexity. In this work optimality is not the most important constraint as yet there is no critical time or path length limit for agent repositioning. Completeness is a more important metric, if a solution exists it should be found.

The approaches presented in the literature can be classified in different categories based on their fundamental idea in the categories *Reduction-Based*, *Rule-Based* and *Search-Based* [35]. Reduction-based solvers focus on optimality but are less efficient and not always complete. They have not been considered in this work. Rule based solvers focus on completeness and plan paths by applying a set of defined movement rules at each planning step [35]. The last categorie puts emphasis on nearly optimal solution sacrificing completeness and are based on graph search. A recently proposed search-based approach that has become highly popular [35] for MAPF in general is the CBS-MAPF (Conflict Based Search) algorithm [36]. It plans paths for each individual agents on a low level and avoids path collisions by using a data structure called *Constraint Tree*. While it is optimal, it is not complete and inefficient on

open space graphs [36]. A python implementation of CBS ha s been tried as a path planning algorithm, however the runtime and rate of successfully found paths proved to be poorly for our setup and we moved on to a new approach. Furthermore in most of the MAPF approaches as presented above assumptions are made regarding the capability of the agents that are not applicable for the conditions in this work. Specifically those are the existence of a wait step i.e. a path step where the rovers maintain their position and a fixed length for each movement step. The low level path planners of the rovers themself do not support a fixed length wait state in a path. Also they aren't synchronized or tick based for fixed length movement steps. As a consequence such time slot based solutions to the MAPF problem are not applicable to the situation at hand.

To solve the path planning issue therefore a different approach is taken to the general MAPF solutions. The strategies named above assume that all agents execute their movements concurrently which is the root cause for the problem of dynamic obstacles. If instead the agents move sequentially, the other agents can be treated as static obstacles in the navigation graph with their start position as well as their goal positions. This way the order in which the paths are executed is arbitrary. As the individual path planner for each agent the A* (a-star) graph search algorithm is used. A* was already proposed in 1968 in a seminal paper by Hart et. al. [37] in the context of robotics research at Stanford. A* is complete and optimal for the construction of shortest paths through graphs. For each agent a start and a goal position is used and A* expands trough the graph while considering an addtional admissible heuristic when choosing the next position in the graph [37]. For path planning we discretize the domain in a regular grid while including a discretization factor to increase the graph size which translates to increased resolution in the real world. Each grid point has 8 considered neighbors, in the cardinal and diagonal directions respectively. When requesting new agent positions, which agent ends up at which position is arbitrary and not defined. Therefore at first a task assignment has to be done that assigns each start position a goal position to determine a (start, goal) tuple as input to the path planner. For the task assignment a greedy approach is chosen that loops trough the remaining goals for each start position and selects the nearest remaining goal. For both the A* heuristic and the task assignment the

Algorithm 5.1 Greedy Goal Assignment

Require: start positions *start*, goal positions *goal*,
  // Assign goal positions to start positions
  assignments ← []
  if ∃*s* ∈ *start* == *g* ∈ *goal* **then**
    add (s,g) to assignments
    remove s from start
    remove g from goal
  end if
  for s in start **do**
    index ← 0
    best ← 0
    best_value ← 0
    for g in goal **do**
      distance ← *euclidean_distance*(*g* − *s*)
      if *best* == 0 **then**
        best_value ← *distance*
      else
        if *distance* < *best_value* **then**
          best_value ← *index*
          best ← *index*
        end if
      end if
      index ← *index* + 1
    end for
    add (s, goal[best]) to assignments
    remove goal[best] from goal
  end for
  return assignments

Parameter	Name	Explanation
Main file name	planner.py	name of the file containing plugin entry function
Function name	plan	name of the plugin entry function
Argument	starts	list of start positions
Argument	goals	list of goal positions
Argument	grid	navigation graph
Return value	paths	list of lists of positions
Return value	concurrent	flag to indicate wether concurrent execution is possible

Table 5.1: Plugin interface of path planner node

euclidean distance on the grid is chosen. The goal assignment algorithm implemented is shown in Algorithm 5.1.

The path planner node is one of the nodes implementing the plugin architecture presented in 5.1.1. Consequently the planning approach presented above has been implemented as a plugin while the node itself only recieves the requested next positions, keeps track of the current positions and communicates the planned paths to the low level planners per agent. The interface definition for the plugin architecture is as illustrated in Table 5.1.

The nodes and communication channels participating in the movement system are visualized as a rosgraph [2] in Figure 5.3. This illustration also shows the low level path planner node for each agent and the topic it uses to send movement commands to the motor driver node.

5.1.4 Measurement Simulation

Simulation of the seismic measurements is implemented by the simulation (*forward_sim*) node. Its two responsibilities are the simulation of the seismic wavefield for a given subsurface model and the creation and publishing of measurements for the current rover positions.

[2] Again for the system with 3 simulated rovers as in the full graph above.

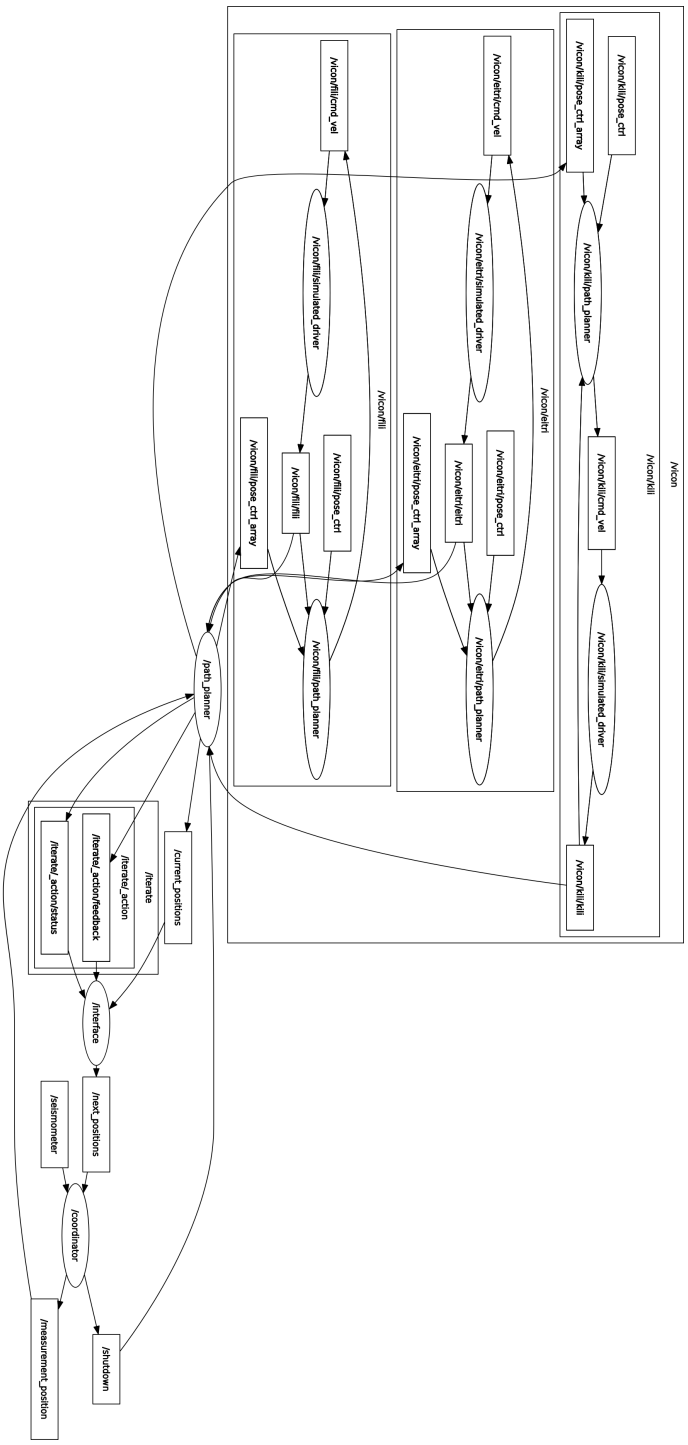

Figure 5.3: Rosgraph of the running system showing only movement related nodes and their communication channels

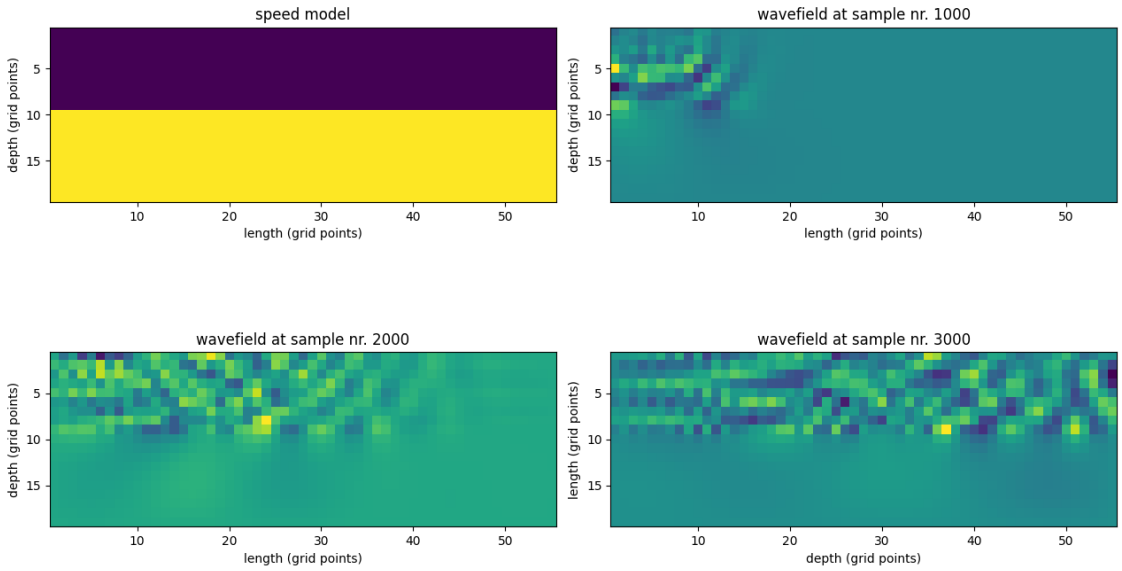

Figure 5.4: Speed model and corresponding simulated wavefield at different time steps

The wavefield is simulated in its entirety at node startup using the specified solver plugin and the simulation is rerun upon setting a new source point so the retrieval of measurements is only a matter of taking the timeseries representing the seismic signal at the requested position from the wavefield without computation for each measurement individually. Figure 5.4 shows a visualization of the simulated wavefield for a velocity model with one interface using snapshots at different time steps.

The simulation node also implements the plugin architecture so different numerical solvers can be used. The interface is defined as according to Table 5.2. This interface definition also includes optional arguments that may or may not be used by the algorithm implemented by the plugin depending on its needs.

The interface definition of a measurement is implemented by defining a ROS message type called *Measurement*. Custom ROS messages are defined using an Interface

Parameter	Name	Explanation
Main file name	solver.py	name of the file containing plugin entry function
Function name	get_wavefield	name of the plugin entry function
Argument	source	tuple, position of seismic source
Argument	model	three dimensional array defining the speed model
Argument	x	physical model height (m)
Argument	y	physical model width (m)
Argument	z	physical model depth
Argument	d	discretization (points/m)
Optional	duration	simulation time (s)
Optional	frequency	center frequency of source signal
Optional	time shift	shift of source signal start time
Return value	wavefield	four dimensional array holding the wavefield

Table 5.2: Plugin interface of measurement node

Despriction Language (IDL) that is translated to the relevant languages by a code generator during the build process. The required attributes of a measurement are elabortated in Section 4.3.1. Implementation using the ROS IDL looks like shown in Listing 5.1.4 where *Point* is a predefined ROS message denoting a position with cartesian coordinates. This also demonstrates how ROS messages can be nested to create more complex and abstract message types.

Listing 5.2: Measurement Message Definition

```
1 geometry_msgs/Point position
2 geometry_msgs/Point shot_position
3 float32 duration
4 float32[] data
```

5.1.5 Inversion

The inversion node implements data processing for inversion of the seismic measurements. Therefore this node also implements the plugin architecture so different inversion schemes can be integrated. Table 5.3 specifies the interface definition. Again optional arguments are used. The directory argument has the intent to specify

Parameter	Name	Explanation
Main file name	inversion_scheme.py	name of the file containing plugin entry function
Function name	invert	name of the plugin entry function
Argument	measurements	list of measurement messages
Optional	directory	directory to save internal plugin data
Optional	initial_model	existing model estimate
Return value	layers	list containing layer depths and corresponding velocities

Table 5.3: Plugin interface of inversion node

a directory to save diagnostic data and visualizations about the inversion process internal to the plugin. Some schemes could make use of the existing version of the model estimate, so this is also provided as an optional argument. Therefore internally the inversion node keeps its own subsurface model that represents the estimated subsurface based on the inversion results. This model is updated with each inversion to include the information obtained by new measurement data.

For testing purposes the normal moveout correction as explained in Section 3.3 is implemented. A flowchart of the implemented algorithm is shown in Figure 5.5. The first step in the algorithm is the formatting and sorting of the measurement data. As input the algorithm takes a list of seismic trace data in the form of the *Measurement* message introduced in the previous section. Those messages arrive at the inversion node in a random order, so the measurement data first needs to be sorted by ascending offset from the seismic source so they are in the correct order for the polynomial refression step. Using Algorithm 5.3 a new list of tuples containing the source to receiver distance and the corresponding raw measurement data is constructed.

For the next step the travel times require the knowledge of the arrival time of the reflection under consideration. Arrival times are determined using seismic event pickers like those based on the STA/LTA ratio [38] that puts into relation the short term average of the signal amplitude to the long term average and thereby enhances short term increases in the signal amplitude. In a previous work [39] the usage and

Algorithm 5.3 Measurement Data Formatting

Require: seismic traces tr
  // Format and sort seismic measurement data
  $shot_x \leftarrow tr.shot\_position.x$
  $shot_y \leftarrow tr.shot\_position.y$
  $offset \leftarrow []$
  for $t \in tr$ **do**
    $distance \leftarrow 0$
    $pos_x \leftarrow tr.position.x$
    $pos_y \leftarrow tr.position.y$
    if $pos_x == shot_x$ **then**
      $distance \leftarrow |pos_y - shot_y|$
    else
      if $pos_y == shot_y$ **then**
        $distance \leftarrow |pos_x - shot_x|$
      end if
    else
      $distance \leftarrow euclidean\_distance(shot - pos)$
    end if
    add (distance, tr.data) to offset
  end for
  sort offset by shortest distance first
  return offset

configuration of this trigger algorithms has been studied using the implementations in the Python library Obspy, a package for computational seismology [40]. Picker algorithms need to be parameterized and parameter setting is highly dependent on the characteristics of the seismic signal and survey condtions as well as the noise and resulting from this there is no general "silver bullet" solution [41].

When the individual travel times are known they need to be used as a basis to deduce the travel time without any source offset of the receiver. With enough individual travel times gathered as data points, regression can be applied to this problem. To this end the zero offset travel time is estimated using second order polynomial regression [42] based on prior knowledge of the approximately parabolic form of the function describing the arrival time dependent on the receiver offset. For this estimation of t_0 the Numpy library [43] provides an implementation of polynomic regression. As the picking of the onset times is not guranteed to deliver the correct arrival time for the reflection in question the regression will give false estimates for t_0 in some cases. In the worst case scenario t_0 will be estimated as being negative and would therefore violate causality. From a mathematical perspective it will lead to an undefined square root in the velocity esitmation. Therefore a negative zero offset travel time is caught as an error condition that hints to a wrong picking of arrival times.

5.2 System Configuration

For configuring and launching multiple Nodes, ROS provides a tool called roslaunch [44] that allows the definition of system startup actions in a programmatic way using Python scripts that in this context are called launch files. This launch files contain a *Launch Description* that contains all nodes that should be launched including their configuration int the form of ROS parameters and topic name remappings.

To allow for the easy configuration of the whole system, launch files have been written that start up all nodes and set the parameters mapping to the component parameters defined in Chapter 4. Some of the nodes have semantically equivalent parameters like the model parameters for the simulation node and the inversion node

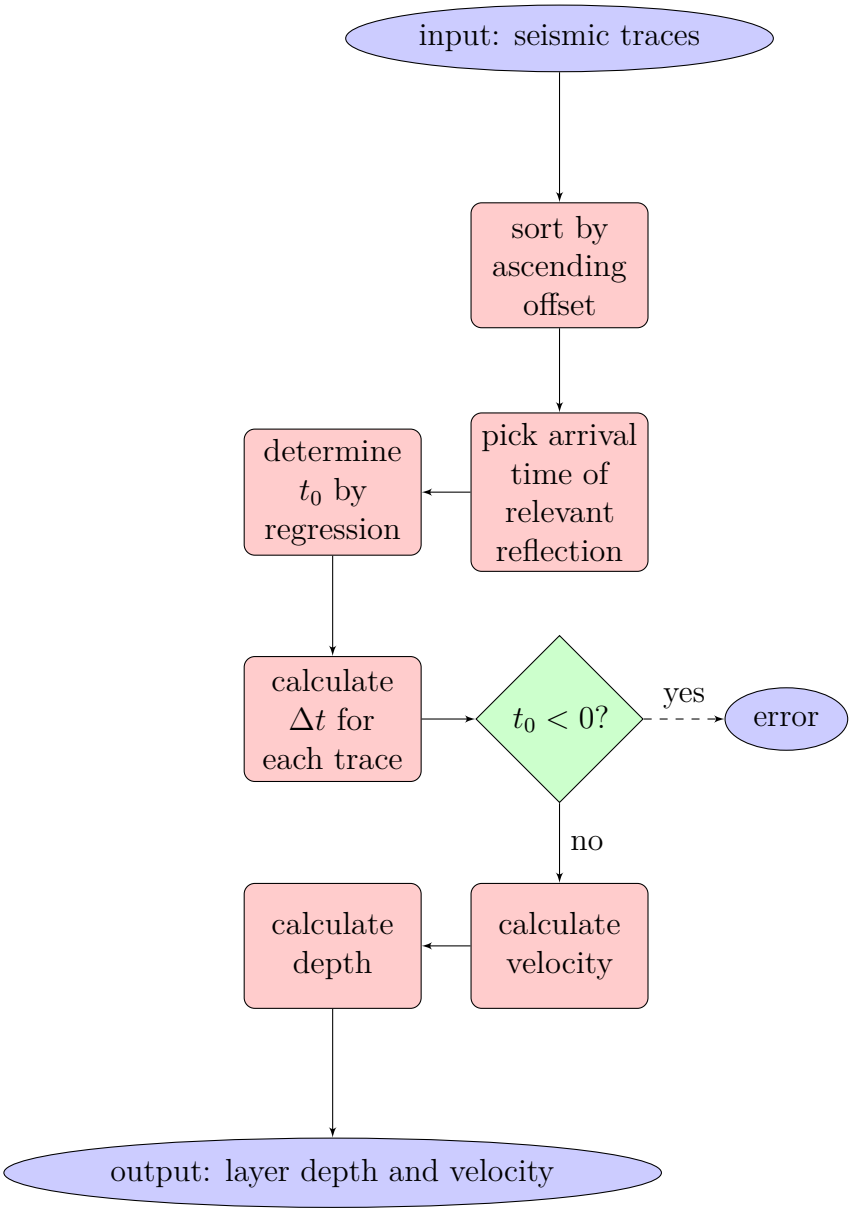

Figure 5.5: Flowchart of NMO algorithm

that should have the same dimensions and discretization. Those nodes are grouped in a single launch file where the parameters can be set at a central point to prevent inconsitencies. The launch file for the movement components also starts up the path planners for the individual agents and in the case of a fully simulated setup like in the rosgraph illustrations above also a node for agent and vicon simulation.

6 Verification Experiments and Evaluation

Experiments have been conducted in simulation as well as using the real robots to verify the integration with hardware. The laboratory environment called *Holodeck* has a space for robot experiments that is fully covered by the Vicon System. A picture of rovers in a measurement line can be seein in Figure 6.1. The picture also shows the reflector balls for identification by the Vicon system. In the following experiments an area of 14x5 meters is used as the exploration domain. The scaling factor for the seismic modeling is chosen to be 4 giving a 56x20x20 points domain which includes a depth of 5 meters. The scaling factor for the path planner is set to 2 giving a resolution of 0.5 meters in the x and y directions for possible waypoints. Additionally the origin of the world coordinate system is shifted for internal use to position it in the upper left corner of the experimentation area. All parameters of the experiment setup are summarized in Table 6.1.

Parameter	x	y	z
Experimentation Area	14 m	5 m	-
Seismic Model Scaling	x4	x4	-
Seismic Model Dimensions	14 m (56p)	5 m (20p)	5 m (20p)
Seismic Model Resolution	0.25 m	0.25 m	0.25 m
Path Planner Scaling	x2	x2	-
Path Planner Dimensions	28	10	-
Waypoint Resolution	0.5 m	0.5 m	-
Origin Offset	+11.3 m	+3 m	-

Table 6.1: Experiment setup parameters

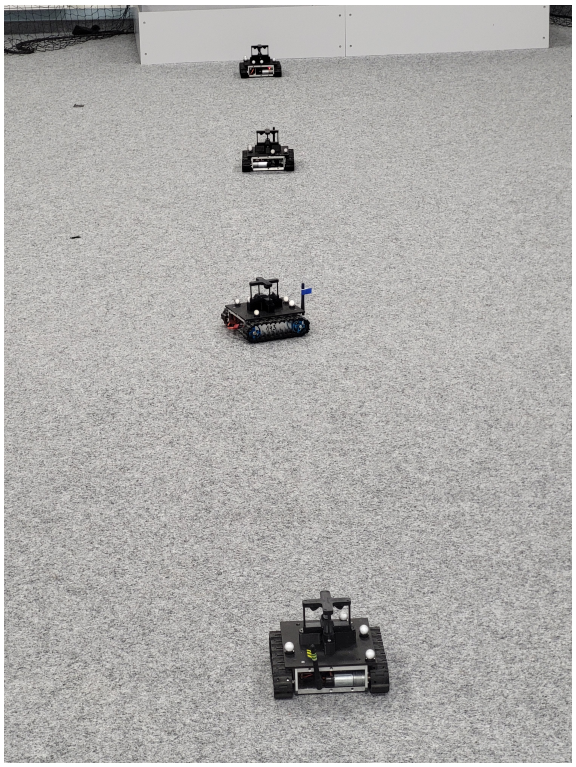

Figure 6.1: *Dwarf* rovers in measurement formation in the *Holodeck*

6.1 Movement Experiments

The first category of experiments tests the path planning and movement aspects of the system. To this end the movement components are tested with different numbers of participating agents using the parameters defined in Table 6.1 for the environment.

The first test is using one rover to confirm that a path is planned and executed in a simple setting. As start and goal respecitvely the positions (11.3, 1) and (5, 2) are chosen. This ensures that the agent has to translate in both spatial dimensions. The resulting path is shown in Figure 6.2. This also shows the individual waypoints and the waypoint resolution of 0.5 meters.

Next path planning and movement is tested with three participating agents. In the first scenario the agents relocate their measurement line formation from a vertical line to a horizontal line. The resulting paths are shown in Figure 6.3. Again the

individual waypoints are illustrated by the line markers. In this scenario it can be seen how the planned paths for two of the agents conflict which could cause a collision if executed concurrently. It can also be seen how the path planner circumvents the goal position of the agent belonging to the blue path with the orange path, so that no collision occurs in the case the blue path is executed before the orange one. In a next scenario the horizontal measurement line is relocated to a parallel horizontal line. The resulting paths are shown in 6.4. It can be seen that the goal assignment strategy correctly assigns each agent to the respective new position that has the same coordinate in the x dimension. The resulting paths have no common waypoint positions so this is a case where parallel execution would be possible even with the used non conflict free path planning approach.

In a final test the number of agents is further increased to five. The first scenario is again the relocation from a vertical to a horizontal measurement line. All the resulting paths are shown in Figure 6.5. Here there are multiple points of potential collision. Again it can be seen how the path planner is planning around the goal positions of other agents in the case of the blue and purple paths to allow for arbitrary path execution order. The next scenario adapts the parallel relocation of the measurement line by including a shift by one meter in horizontal direction for each goal position. Resulting paths are visualized in Figure 6.6. Here a pathological case can be seen where the greedy goal assignment strategy breaks down. Each agent gets assigned the nearest position in the order they are considered. Resulting from the ordering however for the last agent the only position left is actually the farthest away from its current position resulting in the very long purple path. To mitigate such cases the task assignment would need to be changed to a globally optimal assignment approach. As a last scenario the agents relocate back to a vertical line. This scenario is constructed to include the special case that one of the start positions equals one of the goal positions. This case is treated as special before the task assignment to make sure no other agent than that already at this positions gets the goal assigned. All resulting paths can be seen in Figure 6.7. This is yet another case where no path conflicts occur and a parallel execution would be possible.

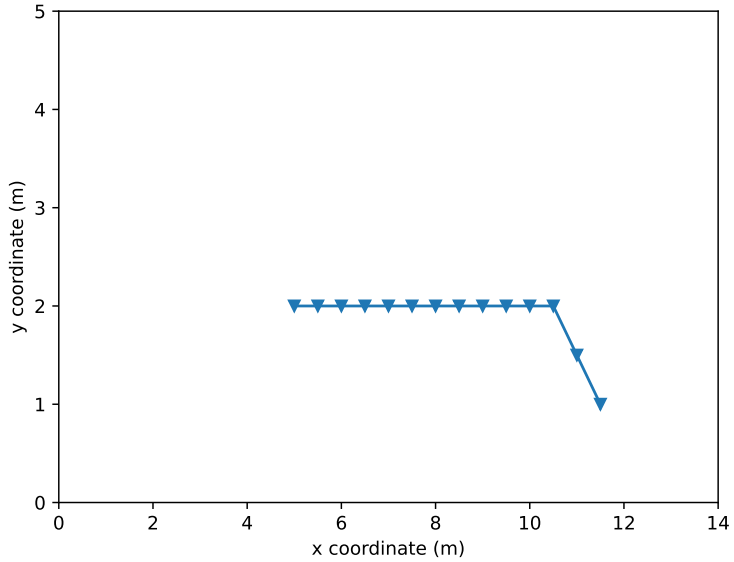

Figure 6.2: Path of one rover from position (11.3, 1) to position (5, 2). The triangle markers illustrate the waypoints.

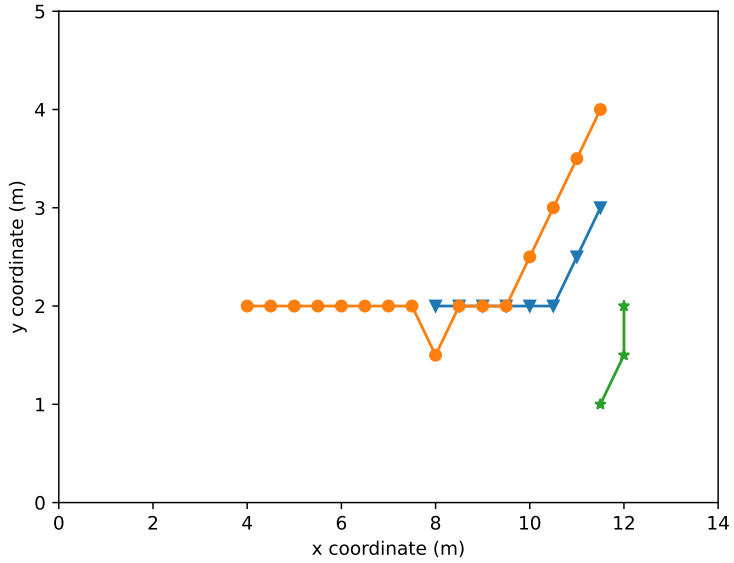

Figure 6.3: Paths of three rovers relocating from a vertical to a horizontal line formation.

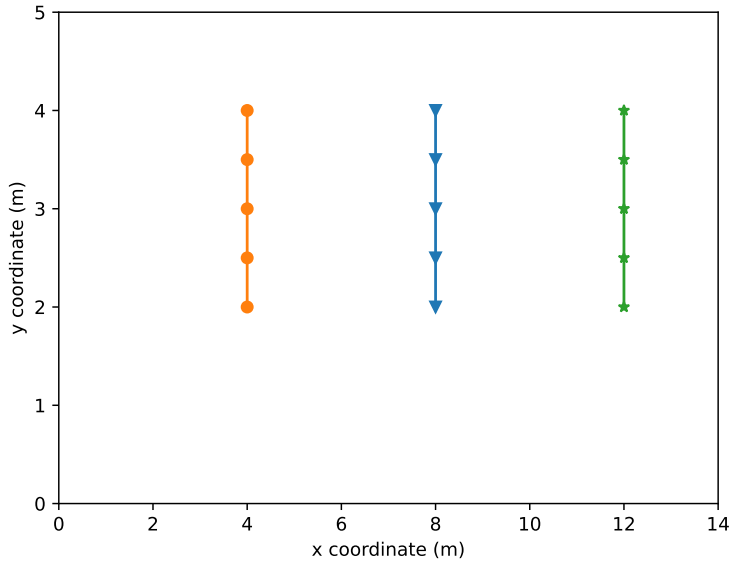

Figure 6.4: Paths of three rovers relocating to a parallel line formation.

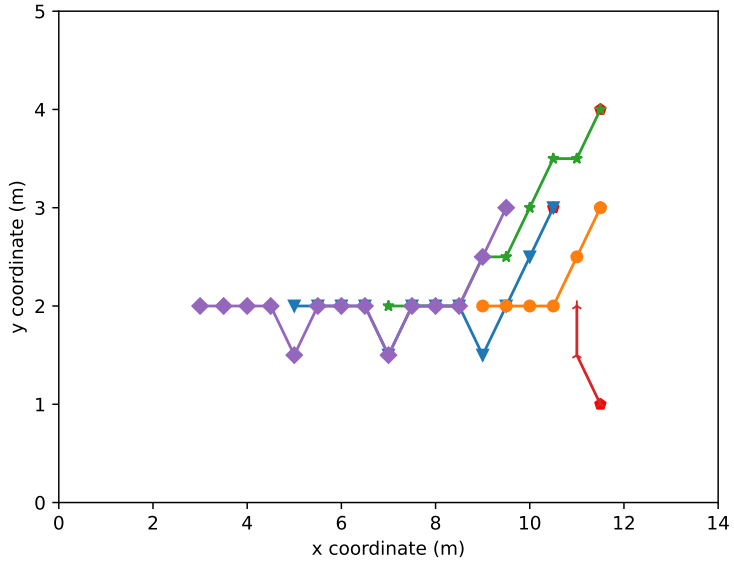

Figure 6.5: Paths of five rovers relocating from a vertical to a horizontal line formation.

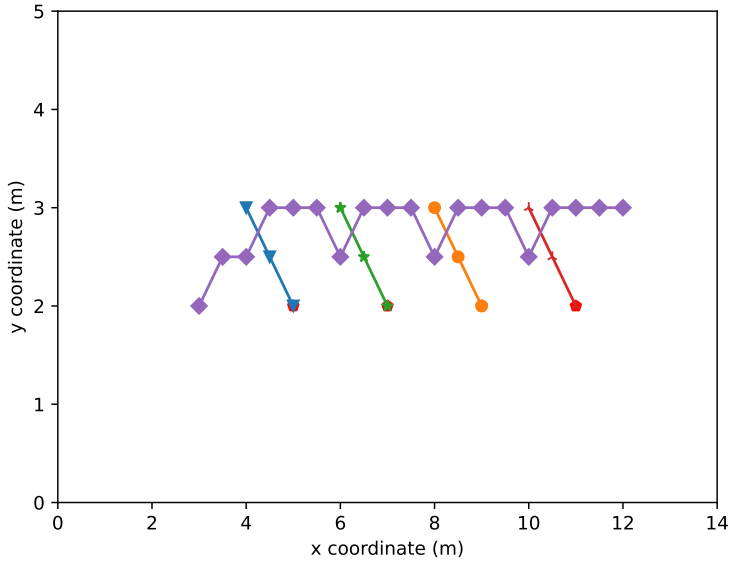

Figure 6.6: Paths of five rovers relocating a horizontal line formation while shifting position by one meter.

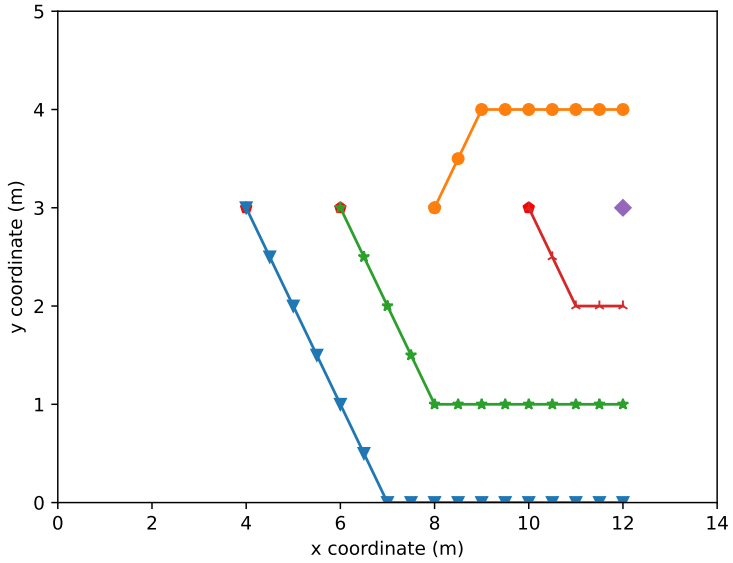

Figure 6.7: Paths of five rovers relocating from a horizontal to a vertical line formation.

6.2 Seismic Experiments

In the seismic experiments the measurement and inversion components of the system are tested. The rovers are placed in a line with the position of the seismic source. We use a velocity model that has a seismic velocity in the first layer of 200 m/s and a depth to the first interface of 5 meters. Two examples of the simulated signal measured at different offsets from the source can be seen in Figure 6.8 and Figure 6.9. The source position is marked in the plots with a star and a line extending out to the position of the agent which equals the sensor position. The signals are normalized to the intervall $[-1, 1]$. It can be seen that the signal for the sensor with the larger source offset has a later onset time than the signal for the sensor closer to the source which indicates that the source offset is reflected in the travel times.

For the inversion experiments a configuration file for the picker parameters of the NMO plugin is implemented to allow for experimentation in order to find acceptable picker and trigger settings. The configurable parameters are as summarized in Table 6.2.

Parameter	Explanation	Units
STA	Length of the STA window	int, nr. of samples
LTA	Length of the LTA window	int, nr of samples
High threshold	Threshold for reflection arrival onset	int, abs. value
Low threshold	Threhold for end of reflection arrival	int, abs. value
Cutoff frequency	Optional lowpass filter	int, Hz or 0 to toggle off

Table 6.2: Parameters for the STA/LTA picker.

The STA/LTA picker returns a characteristic function that describes the ratio of the short term average over the long term average for each sample. The thresholds are used to determine the moment of a reflection arrival when the value of the characteristic function rises above the threshold. Figure 6.10 shows the characteristic function of the measurement shown in Figure 6.11. It can be seen how the characteristic function has two peaks that coincide with the peaks that are visible in the signal below. In this case a value between 1.5 and 2 would be sensible for the high threshold in the parameter settings.

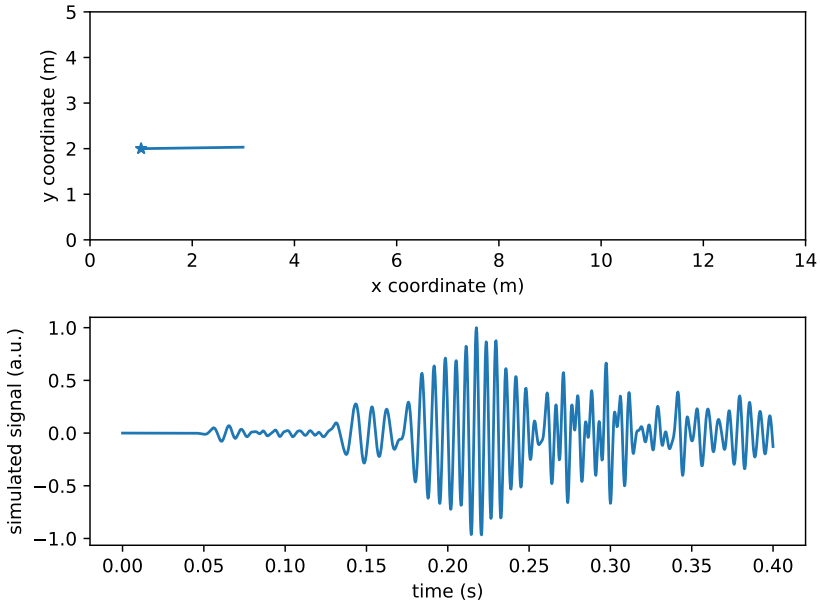

Figure 6.8: Measurement at position (3, 2) with source at (1, 2).

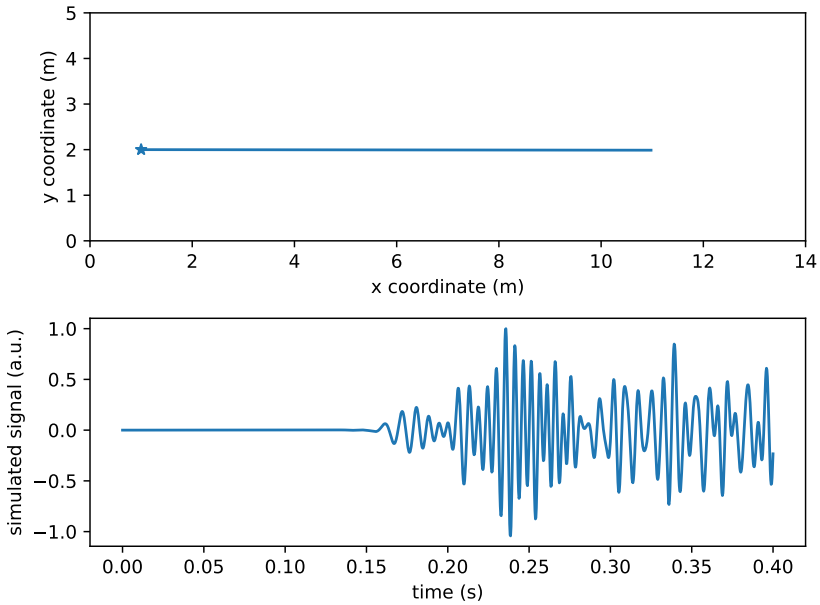

Figure 6.9: Measurement at position (11, 2) with source at (1, 2).

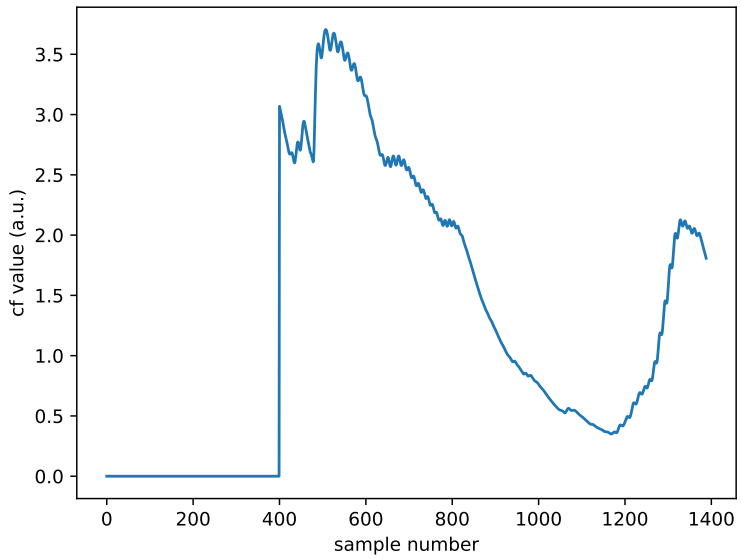

Figure 6.10: Characteristic function of a seismic signal.

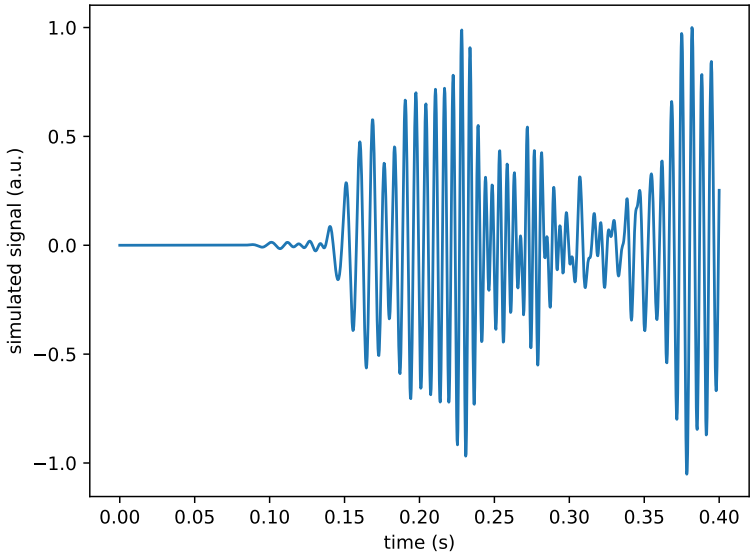

Figure 6.11: Seismic signal with two arrivals visible.

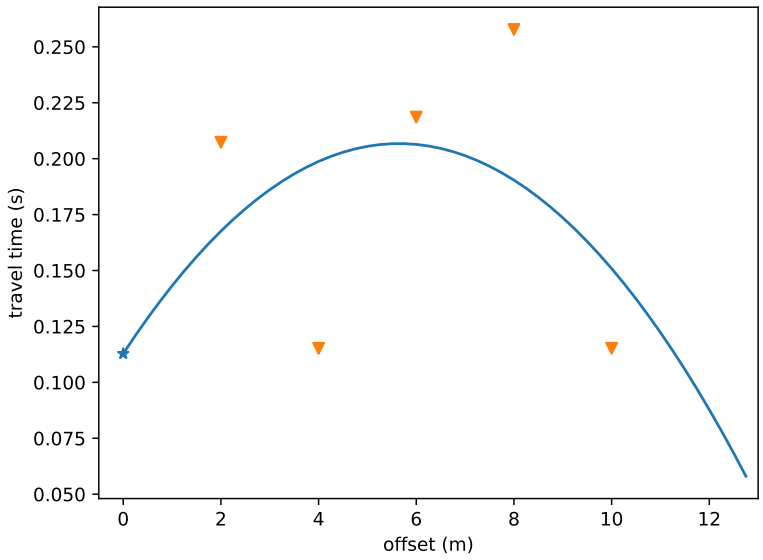

Figure 6.12: Bad estimation of t_0 caused by bad travel time picks for regression.

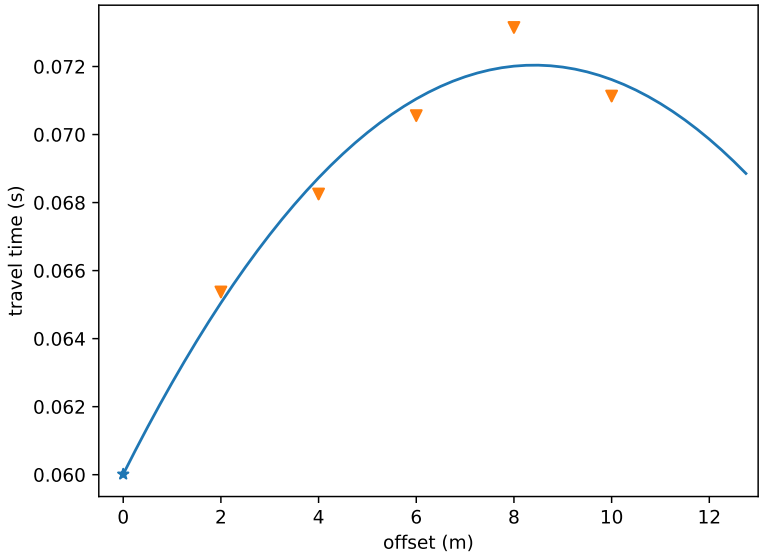

Figure 6.13: Acceptable estimation of t_0 based on the picked travel times.

In practice however the consistent picking of valid travel times for all measurements presented a challenge. Often a set of picker parameters only provided useful travel time picks for one or two measurements while picking wrong for the others. An example of this is shown in Figure 6.12 where the picked traveltimes, which are marked by the orange triangles, fluctuate up and down. The visualized regression parabola therefore is a bad estimation of the true travel time curve and the estimated t_0 marked by the blue star is not usable. In another measurement in Figure 6.13 the picked travel times follow the principal form of increasing in a parabola with one outlier at an offset of eight meters. Using the estimated t_0 for inversion delivers an estimated layer velocity of 170 m/s (True velocity of 200 m/s) and a depth of 5 meters rounded to the next integer which corresponds to the true depth of 5 meters. Therefore the implemented NMO algorithms works when provided with consistent and valid travel times, but those cases are rare and the ususal results have been inconsistent picks, so travel time picking needs significant improvement to make the NMO inversion robust enough for practical use.

6.3 Evaluation

The experiments show the already implemented capabilities and also the remaining limitations of the system. It has been shown that the components of the system are able to interact in the intended way to execute a seismic survey. Looking back at the initially defined requirements leads to the following conclusions.

- Accuracy - This aspect is relevant for the seismic simulation and the inversion. When presented with acceptable arrival time picks on the basis of the simulated seismograms the inversion delivered a result that was accurate to a few meters/second in velocity. This present a reasonable baseline to improve upon with better time picking.
- Extensibility - The system was able to operate with different numbers of agents, hence it is extensible in the number of sensors. Further a plugin architecture

was implemented that allows to integrate new algorithms for seismic simulation, inversion and path planning.

- Adaptibility - The system itself makes no assumptions about the position of agents hence the agents can be placed in arbitrary formation. This does not mean however that certain inversion schemes like NMO do not have their own requirements regarding measurement formation.
- Ready for autonomy - The path planner can get new positions from any source that publishes on the relevant ROS topic. Therefore it is straigthforward to implement a node that automatically sends new position and measurement commands. This is a task that could be delegated to the coordinator node.

All in all the implemented system is providing a working base layer that still has some major flaws that need to be worked on. Besides the issue of travel time determination the inversion node model update is still not flexible enough. The current implementation assumes that layer depths are constant over the whole domain. Here improvement is necessary to allow layer depth updates on a per axis or even per position basis to invert more complex models.

7 Summary

This thesis presented the development and validation of a Hardware-in-the-Loop system with the goal of simulating seismic data gathering and processing for subsurface models using multiple robotic rovers. To this end a distributed multi agent exploration system was implemented using the ROS robot middelware framework. General interfaces for seismic data gathering as well as data processing are provided that allow for the use of different data sources and inversion algorithms and contribute to the extensibility of the system. The system can work with different numbers of agents which influences the time it takes to change measurements position and the amount of information gathered with each measurement. Emphasis was laid on the extensibility and adaptibility of the system so that new algorithms for inversion, forward modeling and path planning can easily be integrated and tested. The ROS node architecture also hides the implementation details of the components so that the system is adaptable to different combinations of real and simulated aspects. An algorithm for path planning and for inversion using normal moveout have been implemented and presented. Using these algorithms validation experiments have been conducted in simulation and with real hardware to validate the workflow. Agents are able to reposition themselves without collision using sequential path execution and simple velocity profiles can be detected using NMO if the reflection arrival time estimation is giving valid results which still is a challenging problem. There are still constraints left concerning the efficiency and flexibility of the path planning and the limited capabilities of the implemented inversion scheme and the uncertainty introduced by the arrival time picking. Also manual definition of the measurement configuration is necessary. To conclude the thesis the following Chapter 8 will give an outlook and suggestions on possible future extensions and improvements.

8 Outlook

With the basic framework of the Hardware-in-the-Loop system developed, future work will focus on extending and improving its capabilities. This includes addressing the constraints mentioned in the evaluation in Chapter 6. Path planning algorithms should be adapted to the system and included that allow for concurrent path execution without active control or fixed time steps. Even without other algorithms a simple first improvement could be to check the paths planned by the current algorithm for collisions and mark the paths as concurrently exectuable if none are found. Alternatively a promising approach is to extend the path planning with an optimization step to reduce the overall execution time of the repositioning even for the case of non concurrent path execution. More work is also necessary to improve the inversion logic. The general model update needs to be extended to include multiple layers and local layer depths to allow for more complicated velocity models. Additional improvements could regard the forward modeling. The execution time of the numerical simulator should be optimized, either by using multiprocessing or threading or by implementing a wave equation solver on a parallel hardware architecture like a GPU. Of vital importance if to find a more robust solution for the estimation of travel times as this step presented the main challenge in the experiments.

As the motivation behind the system is to serve as a testbed for algorithms in a multi-agent seismic survey, a major goal should be to develop, integrate and test this new algorithms. This includes the mentioned path planners but also the inversion schemes. More advanced inversion techniques such as seismic tomography and full waveform inversion should be studied using the system. As with the wave equation solver again the distributed nature of the system would allow for this algorithms to

be implemented and to run on computers containing specialized hardware to increase performance. Another approach in this regard would be the training and usage of physics informed neural networks on dedicated GPU accelerators for use in inversion. Those experiments could also look at the automatic determination of new positions based on the inversion results which is a topic that hasn't seen much attention yet in the context of seimology.

Finally the hardware aspect can recieve a further look. The simple sending and recieving of waypoints while being a limiting factor for path planning, makes few assumptions about the used robots and their capabilities which allows for the testing and evaluation of new platforms. Of special interest here is the replacement of the external localization provider by the capability of self localization for the agents as will be required in real environments without an external localization provider.

Bibliography

[1] National Aeornautics and Space Administration. *Mars 2020*. 2022. URL: https://mars.nasa.gov/mars2020/ (visited on 08/01/2022).

[2] European Space Agency. *MarsExpress*. 2022. URL: https://www.esa.int/Science_Exploration/Space_Science/Mars_Express (visited on 08/01/2022).

[3] European Space Agency. *BepiColombo*. 2022. URL: https://sci.esa.int/web/bepicolombo (visited on 08/01/2022).

[4] W. Sun et al. "Detection of seismic events on Mars: a lunar perspective". In: *Earth and Planetary Physics* (2019). DOI: 10.26464/epp2019030.

[5] Y. Nakamura, G. Latham, and H. Dorman. "Apollo Lunar Seismic Experiment - Final summary". In: *Journal of Geophysical Research* (1982). DOI: 10.1029/JB087iS01p0A117.

[6] D. L. Anderson et al. "Seismology on Mars". In: *Journal of Geophysical Research* (1977).

[7] G. K. Becker. "Crustal thickness from seismic noise correlations in preparation for the InSight mission to Mars". Dissertation. Georg-August Universität, 2018.

[8] A. Khan et al. "Upper mantle structure of Mars from InSight seismic data". In: *Science* (2021). DOI: 10.1126/science.abf2966.

[9] B.-S. Shin and D. Shutin. "Subsurface exploration on Mars and Moon with a robotic swarm". In: *Global Space Exploration Conference (GLEX 2021)* (2021).

[10] S. Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[11] Open Robotics. *ROS 2 Documentation: Galactic*. 2022. URL: https://docs.ros.org/en/galactic/.

[12] K. Wyrobek. *The Origin Story of ROS, the Linux of Robotics*. 2017. URL: https://spectrum.ieee.org/the-origin-story-of-ros-the-linux-of-robotics (visited on 06/27/2022).

[13] OSRF. *Open platforms for robotics*. 2022. URL: https://www.openrobotics.org/ (visited on 06/27/2022).

[14] G. Stavrinos. "ROS2 for ROS1 Users". In: *Robot Operating System (ROS). The Complete Reference (Volume 5)*. 2021. DOI: 10.1007/978-3-030-45956-7_2.

[15] OMG. *DDS 1.4 Specification*. 2015. URL: http://www.omg.org/spec/DDS/1.4 (visited on 06/27/2022).

[16] OSRF. *rclpy API documentation*. 2019. URL: https://docs.ros2.org/latest/api/rclpy/.

[17] Vicon Motion Systems Ltd. *Vicon Tracker*. 2022. URL: https://www.vicon.com/software/tracker/ (visited on 06/27/2022).

[18] P. Merriaux et al. "A Study of Vicon System Positioning Performance". In: *MDPI Sensors* (2017). DOI: 10.3390/s17071591.

[19] A. E. Mussett and M. A. Khan. *Looking into the Earth*. Cambridge University Press, 2002.

[20] E. Robinson and S. Treitel. *Digital Imaging and Deconvolution: The ABCs of Seismic Exploration and Processing*. Society of Exploration Geophysicists, 2008. DOI: 10.1190/1.9781560801610.

[21] B.-S. Shin and D. Shutin. "Adapt-then combine full waveform inversion for distributed subsurface imaging in seismic networks". In: *IEEE International Conference on Acoustics, Speech and Signal Processing* (2021).

[22] H. P. Langtangen and S. Linge. *Finite Difference Computing with PDEs*. Springer Open, 2017. DOI: 10.1007/978-3-319-55456-3.

[23] A. Meister and T. Sonar. *Numerik*. German. 2019. DOI: 10.1007/978-3-662-58358-6.

[24] L. Wientgens. *Implementation and Analysis of an Eikonal Equation Solver for Seismic Tomography on a Graphics Processing Unit*. Study Report. DHBW Mannheim, 2021.

[25] D. Komatitsch and J. Tromp. "A perfectly matched layer absorbing boundary condition for the second-order seismic wave equation". In: *Geophyscial Journal International* (2003).

[26] P. Bormann and E. Wielandt. "Seismic Signals and Noise". In: *New Manual of Seismological Observatory Practice*. Ed. by P. Bormann. 2nd ed. GFZ German Research Centre for Geosciences, 2013. Chap. 4.

[27] Ö. Yilmaz. *Seismic Data Analysis*. Society of Exploration Geophysicists, 2001. DOI: 10.1190/1.9781560801580.

[28] L. Uieda. "Step-by-step NMO correction". In: *The Leading Edge* (2017). DOI: 10.1190/tle36020179.1.

[29] J. A. Ledin. "Hardware-in-the-Loop Simulation". In: *Embedded Systems Programming* (1999).

[30] M. Ghorbani et al. "Real-time hardware-in-the-loop test for a small upper stage embedded control system". In: *2018 Real-Time and Embedded Systems and Technologies (RTEST)* (2018). DOI: 10.1109/RTEST.2018.8397164.

[31] A. Kaden, B. Boche, and R. Luckner. "Hardware-in-the-loop Flight Simulator - An Essential Part in the Development Process for the Automatic Flight Control System of a Utility Aircraft". In: *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control* (2013).

[32] C. di Mascio and G. Gruosso. "Hardware in the Loop Implementation of the Oscillator-based Heart Model: A Framework for Testing Medical Devices". In: *MDPI Electronics* (2020). DOI: 10.3390/electronics9040571.

[33] G. Bengel et al. *Masterkurs Parallele und Verteilte Systeme*. German. 2015. DOI: 10.1007/978-3-8348-2151-5.

[34] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[35] H. Ma. "Graph-Based Multi-Robot Path Finding and Planning". In: *Current Robotics Reports* (2022). DOI: 10.1007/s43154-022-00083-8.

[36] G. Sharon et al. "Conflict-based search for optimal multi-agent pathfinding". In: *Artifical Intelligence* (2015). DOI: 10.1016/j.artint.2014.11.006.

[37] P. Hart, N. Nilsson, and B. Raphael. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE Transactions on Systems Science and Cybernetics* (1968).

[38] L. Küperkoch, T. Meier, and T. Diehl. "Automated Event and Phase Identification". In: *New Manual of Seismological Observatory Practice*. Ed. by P. Bormann. 2nd ed. GFZ German Research Centre for Geosciences, 2013. Chap. 16.

[39] L. Wientgens. *Seismic data acquisition and processing with Raspberry Shake modules*. Study Report. DHBW Mannheim, 2020.

[40] M. Beyreuther et al. "ObsPy: A Python Toolbox for Seismology". In: *Seismological Research Letters* 81.3 (05/2010), pp. 530–533. DOI: 10.1785/gssrl.81.3.530.

[41] A. Trnkoczy. *Understanding and parameter setting of STA/LTA trigger algorithm*. Information Sheet. GFZ German Research Centre for Geosciences, 1999. DOI: 10.2312/GFZ.NMSOP-2_IS_8.1.

[42] L. Massaron and A. Boschetti. *Regression Analysis with Python.* Packt Publishing, 2016.

[43] C. R. Harris et al. "Array programming with NumPy". In: *Nature* (2020). DOI: 10.1038/s41586-020-2649-2.

[44] A. Böckenkamp. "roslaunch2: Versatile, Flexible and Dynamic Launch Configurations for the Robot Operating System". In: *Robot Operating System (ROS). The Complete Reference (Volume 5).* 2020. DOI: 10.1007/978-3-030-20190-6_7.