

**An exemplary
application of decision
theory in robotics:
autonomous calibration
of depth cameras with
non-overlapping fields
of view on a mobile
platform**

Maximilian Denninger



BACHELORARBEIT


AN EXEMPLARY APPLICATION OF DECISION THEORY IN ROBOTICS: AUTONOMOUS CALIBRATION OF DEPTH CAMERAS WITH NON-OVERLAPPING FIELDS OF VIEW ON A MOBILE PLATFORM

Freigabe:

Der Bearbeiter:

Unterschriften

Maximilian Denninger

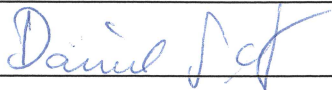


Betreuer:

Christian Rink

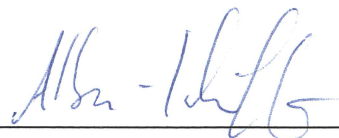


Daniel Seth



Der Institutsdirektor

Dr. Alin Albu-Schäffer



Dieser Bericht enthält 57 Seiten, 20 Abbildungen und 6 Tabellen



Hochschule Kempten
University of Applied Sciences



Deutsches Zentrum
DLR für Luft- und Raumfahrt
Institute for Robotics and Mechatronics

Bachelor Thesis

An exemplary application of decision theory in robotics:
autonomous calibration of depth cameras with non-overlapping
fields of view on a mobile platform

Supervisors, HS Kempten:	Prof. Dr. rer. nat. Jochen Staudacher
Created by:	Maximilian Denninger
Date:	20.03.2015
Department:	Computer Science and Multimedia
Executed at:	German Aerospace Center, Oberpfaffenhofen 82234
Supervisors, DLR:	Dipl.-Math. Christian Rink M. Sc. Dipl.-Math. techn. Daniel Seth
Address of the author:	Eichenstraße 15, Mauerstetten 87665 Email: maxi@maxi-denninger.de , Tel: 0151 15052203

Abstract

This thesis is about the creation of a decider, which is able to evaluate decision trees and utility tables. The decider enables the robot to decide by its own, which action should be performed in the actual situation. Afterwards the decider is used for the calibration process of time of flight cameras with non-overlapping field of views on a mobile platform. A framework is the main objective of this thesis, which can be utilized for known problems as Localization and Exploration in mobile robotics. Therefore, clear interfaces will be designed, to ensure that the framework will be used in future works. After the implementation of the decider a decision tree is designed containing the calibration's process. This tree must have clearly specified actions and states, which can be observed by the robot itself. At the end the combination of the decision process with the calibration will be evaluated.

Zusammenfassung

Diese Arbeit behandelt die Erstellung eines Entscheiders, welcher in der Lage ist, Entscheidungsbäume und Nutzentabellen auszuwerten. Der Entscheider gibt dem Roboter die Möglichkeit, selbstständig auszuwählen, welche Aktion in der aktuellen Situation für ihn am besten ist. Dieser wird anschließend in der Kalibrierung von Tiefenkameras mit nicht überlappenden Sichtbereichen auf einer mobilen Plattform eingesetzt. Diese These zielt darauf ab, ein Framework zu gestalten, welches für bekannte Probleme in der mobilen Robotik eingesetzt werden kann. Das bedeutet, es müssen klare Schnittstellen definiert werden, um zu gewährleisten, dass dieses Framework auch in anderen Teilgebieten verwendet werden kann. Nach der Implementierung des Entscheiders wird ein Entscheidungsbaum ausgearbeitet, welcher den Ablauf des Kalibrierungsprozesses darstellen soll. Dieser muss klar definierte Aktionen und erkennbare Zustände haben, welcher der Roboter eigenständig analysieren kann. Abschließend soll der Entscheidungsprozess und dessen Kombination mit der Kalibrierung evaluiert werden.

Contents

1	Introduction	1
1.1	Motivation and Goal	2
1.2	Related Work	3
1.2.1	Calibration	3
1.2.2	Decision Making	6
2	Fundamentals	8
2.1	Decision Making	8
2.1.1	Bayes Strategy	8
2.1.2	Maximize and Minimize Strategies	9
2.2	Hardware and System	11
2.2.1	OmniRob	11
2.2.2	Sensors	11
2.3	L3D	13
2.4	Mobile Robot Environment	13
3	Concept	15
3.1	Decision Making	15
3.1.1	Decision Tree	15
3.1.2	Strategy	16
3.1.3	Conversion from Decision Tree to Utility Table	21
3.1.4	Roll Back Analysis	24
3.1.5	Decision Criteria	25
3.2	Calibration	28
3.2.1	Calibration Body	29
3.2.2	Laser Measurement	30
3.2.3	ToF Measurement	31
4	Implementation	34
4.1	Decision Maker	34

4.2	Use in Calibration	35
4.2.1	First Iteration	35
4.2.2	Second Iteration	36
4.2.3	Third Iteration	37
5	Test	39
5.1	Evaluation of the Decision Making Process	39
5.2	Combination with the Calibration Process	41
5.2.1	Previous State	41
5.2.2	With the Decision Making Framework	42
6	Conclusion and Future Work	43
	Bibliography	47

List of Figures

1.1	Calibration class tree	3
2.1	The omniRob, with the DLR expansions.	11
2.2	Time of Flight Camera (ToF) camera and laser sensor.	12
2.3	The Fields of View (FoVs) of the ToF cameras are pictured with red pyramids. Green strips show the laser beams, they are clearly visible in the lower right corner.	13
2.4	The omniRob is placed in the simulated mobile laboratory, presented in the viewer.	14
3.1	Two lotteries are pictured as a decision tree. In both a coin toss game is offered. The upper coin toss game has a higher venture and the lower game has a lower venture. An agent can decide between these two lotteries. . . .	16
3.2	The pictured decision tree contains seven decision nodes, no random nodes and eight result nodes. It has 128 strategies based on Equation 3.1.	17
3.3	Purchase decision for a car.	18
3.4	The functionality of the recursive algorithm for the strategies are shown here. The numbers on the arrows indicate the order of the execution. . . .	18
3.5	A decision tree, which is used to explain the strategy generation.	20
3.6	Utility generation in a decision tree.	24
3.7	The simple tree provides a lottery, which has the possibility to play a high or low risk lottery in the case an agent lost the first one.	25
3.8	The blue vector describes ${}_T T^C$ and the red vector contains ${}_B T^T$, with B = robot base, T = TCP point and C = camera frame.	28
3.9	The dimension of the calibration body (CB)	30
3.10	Depth image and point cloud of a ToF camera.	31
3.11	The point cloud used for the Iterative Closest Point (ICP) algorithm. The wall on the left side behind the calibration body was filtered away.	31
3.12	Calibration transformations show the correction of the ToF camera.	32
4.1	First iteration of the decision tree for the calibration process.	35

4.2	Decision tree for the calibration process in the second iteration. The tree has the "bad" case included and can handle this situation.	37
4.3	The decision tree provides the calibration for the ToF cameras.	38

Nomenclature

Abbreviations

CB	Calibration Body
DLR	German Aerospace Center (<i>"Deutsches Zentrum für Luft- und Raumfahrt"</i>)
DoF	Degrees of Freedom
FoV	Field of View
gml	Graph Modelling Language
ICP	Iterative Closest Point
L3D	Library Lib3D
MRE	Mobile Robot Environment
SLAM	Simultaneous Localization and Mapping
TCP	Tool Center Point
ToF	Time of Flight Camera

Rigid Body Transformation

T	Homogeneous 4×4 transformation matrix
${}_{base}T^{TCP}$	Transformation matrix transforms points from the base to the TCP

1 Introduction

In the last decades the use of mobile robots in the industry has grown significantly. In order to keep this process on track the research on this topic is getting more important. This trend will lead to robots, which will be capable to do even more human tasks than robots can do today.

Key elements of mobile robots are Localization, Exploration, Navigation and Simultaneous Localization and Mapping (SLAM) [36]. This thesis introduces an approach in decision theory to improve the execution of these elements. The approach enables the system to decide, which step should be the next. This selection depends on the present situation and enhances the possibility to reach a better result. The result of this thesis is a framework for decision making, which can select an action out of a range of actions. Furthermore, the evaluation of decision trees and utility tables is possible.

All the key elements of mobile robots contain a perception of the environment. Moreover, each key element consists of many steps and in the most steps the perception is fundamental. The environment is perceived by different kinds of sensors and the resulting data can only be used if these sensors are calibrated [33]. In order to achieve an efficient and autonomous calibration, this work combines the framework for decision making with the sensor calibration. The development of this calibration is not a part of the thesis.

In this thesis the phrase camera is used similar to the phrase sensor, because Time of Flight Cameras (ToF) are going to be calibrated. These cameras perceive depth data instead of color information. The data from these cameras is used for the calibration [14].

According to Strobl, "*camera calibration is the process of estimating the parameters of a camera model*" [33]. These parameters can be divided in two main classes. The intrinsic parameters are the first component of a camera being calibrated. These describe the sensor format, the focal length, the principal point and the lens refraction. They are needed to correct the sensor data and to minimize the error of the camera [40]. In this thesis the intrinsic parameters are already given by the manufacturer of the sensors.

The component of the calibration related to this thesis are the extrinsic parameters. These contain the six Degrees of Freedom (DoF) of the camera, which describe the position and the orientation of the camera frame. This transformation is necessary to have the ability

of combining the camera data with an existing world model. This means the depth points, which are generated by the ToF cameras, are converted in points in 3D. Then this transformation is used to guarantee that all data sets have the same coordinate origin. Without this information the relation between these data and a measurement from another camera is difficult.

1.1 Motivation and Goal

The need for robots acting autonomously has increased significantly in the last years, since there is a higher demand for robots. These robots are able to support humans in every imaginable situation. Furthermore, the robots can help humans in a case of emergency. These tasks can only be performed if the robot is able to decide what action should be performed next. Therefore, autonomous decision making is an important part of modern mobile robotics.

The idea of this thesis is to combine the theoretical approaches in decision theory in a framework tested with the calibration of the ToF cameras. In this framework an agent is implemented. Russell and Norvig define an agent with: *"An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actors"* [30]. This means an agent is equal to a decider and able to choose the next action out of a range. This decision process depends on the circumstances. At the end an agent is going to be used in the calibration process of the robot.

This leads to the following subgoals of this thesis:

1. Implementation of an agent, which can evaluate utility tables.
2. Development and implementation of a decision tree to utility table converter.
3. Building up a decision making framework, which is expandable to the use of imprecise probabilities.
4. Developing a decision making model, consisting of a decision tree of the robot and then the decision theory framework is used for the calibration process of a mobile robot.

1.2 Related Work

1.2.1 Calibration

The calibration of the ToF cameras can be divided in some subclasses, which are represented in Figure 1.1. This calibration as mentioned before consists of two main components. The first component are the intrinsic parameters, which describe the intern camera model. They are represented in the left branch of the tree in Figure 1.1. These parameters are not the goal of this thesis and are given by the manufacturer of the cameras.

Many problems in mobile robotics are solved by a combination of several range cameras. Their data can only be used if the transformation describing the position and orientation of the camera is known. This transformation is the second component of the calibration tree and is called extrinsic parameters. They are described in the right branch of the tree in Figure 1.1. This transformation ${}_T T^C$ describes the position and the orientation, in which T is the Tool Center Point (TCP)¹ and C the camera [35].

There are two different common hardware approaches, which can be used to estimate this transformation, refer to the left branch of extrinsic calibration in Figure 1.1. The environment can be perceived by a vision based sensor, which only receives color information. In this case two cameras are used to perform a pattern recognition. This technique is called stereo, for further information see [33].

¹The TCP is the point on the robot, where the camera is mounted. This point could be at the end of a robot arm.

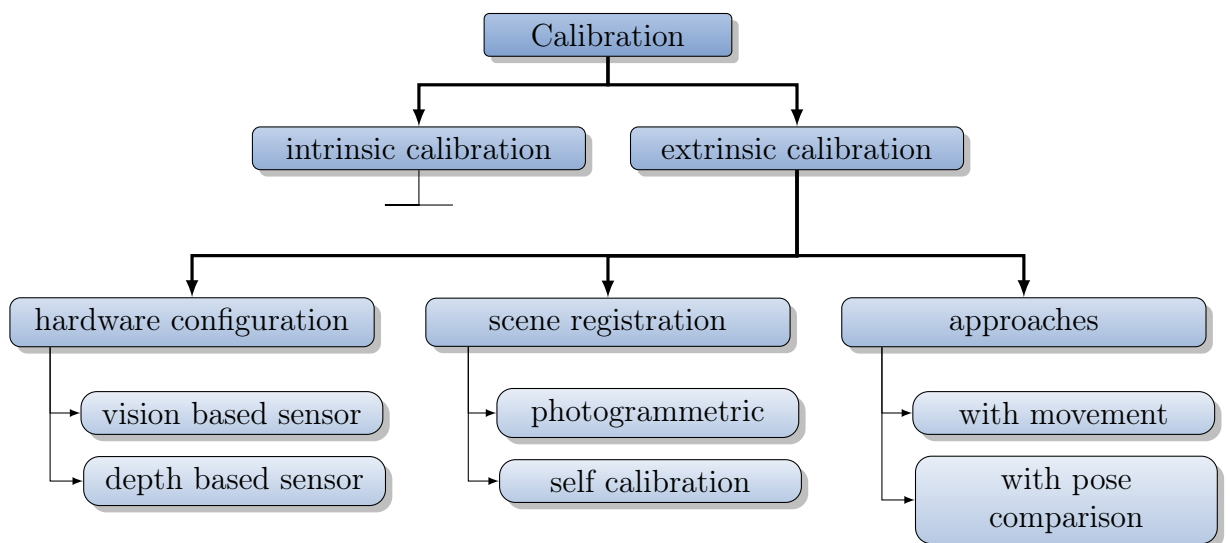


Figure 1.1: Calibration class tree

In the second approach the six DoFs of the camera are determined by a depth based sensor. This geometric information is used to estimate the pose of the camera [14]. This thesis uses ToF cameras and the provided color information of the sensors is not used, because the data is not used for visual representation. Furthermore, the resolution of the sensor is not high enough for a feature detection approach. In this method edges or other elements in a scene are detected in the visual data [15].

The calibration of the ToF camera is made by observing a calibration object and comparing a created point cloud of this object with the depth image of the camera. This process is called photogrammetric calibration [6]. Instead of this, the self calibration uses a static scene. In this scene no object has to be known [40]. The photogrammetric calibration is used in this thesis, because the result of the calibration can be better compared than in an approach where the result depends on the environment [6].

The determination of the transformation can be made with two different techniques, refer to the right branch of the extrinsic calibration:

1. Movement of the hand and observe the motion of the eye:

The underlying function of this technique uses the motion of the TCP from one point to another and compares it with the motion, which is perceived by the camera. These motions are represented by matrices in $\mathbb{R}^{4 \times 4}$. Early solutions of this problem separate the rotation and translation of the transformation, according to [31]. Chen combined 1991 rotation and translation in the screw theory [8]. In 1995 Lu and Chou introduce the eight-space formulation based on quaternions, linearly optimizing using singular value decomposition [26].

2. Simultaneous estimation of the hand-eye transformation and the pose of the robot in the world:

This method compares the pose of the camera in the world with the pose of the TCP mounted on a robot in the world. This formulation was first used by Wang 1992 for hand-eye calibration [38]. With quaternions Zhuang gets a simple linear solution by using singular value decomposition [41].

For more information about these different approaches see [33].

This thesis uses a derivative of the second method, in which the pose in the world is relative to a Calibration Body (CB). The pose is estimated with two laser scanners.

A method which combines the calibration process with the wheel odometry data was introduced by Heng, Li and Pollefeys in 2013 [17]. The pose of the vehicle is estimated

by combining the wheel odometry with the visual odometry. So the pose is more accurate than in previous methods. Moreover, this approach calculates natural feature points on the vision data and only requires the vehicle to be driven around to calculate the extrinsic parameters [17].

In 2014 Fernandez et al. published a new approach for the extrinsic parameters, requiring only to observe one plane from different viewpoints. This method uses the fact that the most scenes consist of large planes and then compare these in each view to estimate the pose of the sensors [12].

But these approaches depend on their surroundings, like the color information of the camera or the planes which should extend over different views. This thesis tries to avoid these dependencies, to ensure that this calibration process works in different environments and additionally without the availability of color information.

1.2.2 Decision Making

In 1713 the model of the expected utility was first mentioned in letters from Nikolaus Bernoulli to Pierre Rémond de Montmort [11]. It describes the benefit an action is gaining during the execution. Afterwards Daniel Bernoulli spent time on the expected utility and formulated the St. Petersburg paradox. In this paradox the expected utility grows without bound. He solved this problem by defining a utility function, which restricts the continued growth of the expected utility. Furthermore, he laid the foundation of risk [3].

Thomas Bayes defined with the bayes theorem the fundamentals of conditional probabilities [2]. Today there are two kinds of statisticians, the subjectivists and the objectivists. The first argue that every personal contribution to the data is a benefit. So the use of probabilities, which are estimated by a person should be used. The objectivists say these probabilities increase the error and the usage of these probabilities should be avoided [13]. This thesis uses conditional probabilities and additionally it is a subjective approach.

Von Neumann and Morgenstern displayed the minimax strategy in their book "Theory of Games and Economic Behavior". Every two-person zero-sum game with finite strategies and the possibility of mixed strategies has one solution vector [37]. It is often mentioned as the ground breaking text, which created the interdisciplinary research field of game theory. Furthermore, they succeeded in expanding the utility theory with the possibility of ranking.

The history of game theory and decision making coincides in many parts. Especially because game theory could be seen as an abstract model of decision making [22]. In the year 1950 the book of E. Lehmann introduced the phrase of "decision theory" [24].

However, today decision making is an academic subject of its own, it is used in many fields like economics, statistics, politics and social science [16].

Decision making can be separated by the question, who is the decider. The first part has an individual person and in the second part a group decides, which action should be performed [23]. This thesis uses the individual person approach.

A separation under the available conditions is possible, too. So a decision can be made under the conditions' certainty, risk or uncertainty. Moreover, there is a combination out of risk and uncertainty [27]. In this thesis the uncertainty approach is used. Hence, all tasks, which are measured by the robot, have always some uncertainty.

Two kinds of uncertainty are distinguishable. The first contains the knowledge about the laws of randomness and the second has no knowledge about them. This means in the

first case are probabilities, which can be determined easily. In the second approach a measurement is necessary to specify the probabilities, because the laws of randomness are not known [10]. The used robot represents the second case, because there is no possibility to determine the probabilities of the used system before the execution.

Moreover, decision making can be distinguished between normative and descriptive decision making. In the first one an agent has full information about all future states. The second approach tries to describe, how humans behave [28]. This thesis uses a weakened normative decision maker, since the robot tries to get as much information as possible.

If the probabilities of a state of nature can not be determined precisely, the algorithm can use imprecise probabilities. This procedure replaces the single values of the probabilities with intervals of them [1, 19]. This thesis does not use them, because of the occurring complexity. But this thesis builds up a base, which is expandable to this approach.

Today there are different approaches to solve the modern problems in mobile robotics. The calibration of the ToF cameras for example could be solved by an expert system, which consists of different rules and a knowledge base [21]. But expert systems are not used in this thesis, because the effort to sustain an expert system is higher than using a decision maker.

This thesis uses sequential decision making. It relies on the statement that the problem must be formulated as a sequence of independent decisions. This means the state of the system defines, which actions are available. Sequential decision making is often used with markov models. But in this thesis decision trees are used, because markov models have no guarantee to determine [25].

Reinforcement learning is an approach to improve the execution of a task and an alternative to decision making. This is not used in this thesis, because of the circumstance that good results will be only achievable if the number of repetitions is high [34].

2 Fundamentals

2.1 Decision Making

A decision of an agent consists of two main components. At first there are the states Θ of nature representing the actual states of the system. Every system has n states of nature θ_i , $i = 1, \dots, n$. An agent has to select between m possible actions. Therefore, the second component are the available actions a_j , $j = 1, \dots, m$. These actions consist of different steps, which will be performed by the robot if the action is selected [27]. Combined these two in a table lead to a so called utility table, refer to Table 2.1. Each element $u(a_j, \theta_i)$ of this table describes how much utility this action gains, a higher value is therefore better. For example has element $u(a_2, \theta_1)$ a higher value than $u(a_1, \theta_1)$. Therefore, the agent can take action a_2 , to maximize its own gain, if the agent should know that the state of nature is θ_1 .

Table 2.1: Utility table

$a \backslash \Theta$	θ_1	θ_2
a_1	1	4
a_2	3	2

2.1.1 Bayes Strategy

A probability is called "a priori", if its value can be estimated before the utility table is calculated. Each state θ_i of nature has a probability p_i of occurrence. The sum of all products of these probabilities p_i and the utilities $u(a_j, \theta_i)$ lead to the expected utility $\mathcal{U}(a_j) = \sum_{i=1}^n p_i \cdot u(a_j, \theta_i)$ for a defined action a_j . The action which maximizes $\mathcal{U}(a_j)$ is the so called Bayes strategy [10].

2.1.2 Maximize and Minimize Strategies

The order of the minimization and maximization specifies the name of the strategy. At the beginning the row is evaluated and as result each row has one value. Afterwards for each of these values the second execution is performed. The first execution defines the first name and the second execution is the last name [10, 27, 29].

Minimax Strategy

The minimax strategy maximizes the minimum utility, this will be useful, if the best action for the worst case is searched.

$$\hat{a}_l = \max_{i=1,\dots,m} \min_{j=1,\dots,n} u(a_i, \theta_j) \quad (2.1)$$

The result of Equation 2.1 used with the Table 2.1 is $u(a_2, \theta_2) = 2$. It is $u(a_2, \theta_2)$, because the lowest value for the first action a_1 is one and for second is two and the maximum of them is the value of the second action a_2 .

Maximax Strategy

The maximum expected utility is maximized with the maximax strategy. The maximax strategy will be used, if the highest possible gain is searched.

$$\hat{a}_l = \max_{i=1,\dots,m} \max_{j=1,\dots,n} u(a_i, \theta_j) \quad (2.2)$$

The maximax strategy shown in Equation 2.2 has as result the utility $u(a_1, \theta_2) = 4$ in the Table 2.1.

Maximin Strategy

In loss tables the maximin strategy is used. The aim of it is to maximize the minimum loss or in a utility table it minimize the maximum utility. This can be useful if the worst action in a utility table is searched.

$$\hat{a}_l = \min_{i=1,\dots,m} \max_{j=1,\dots,n} u(a_i, \theta_j) \quad (2.3)$$

Equation 2.3 chooses out of each row the element with the highest value. The minimum of these elements is the result of the maximin strategy. In Table 2.1 the biggest minimum in the available actions is $u(a_2, \theta_1) = 3$.

2.2 Hardware and System

2.2.1 OmniRob

The used robot is the "omniRob" produced by KUKA, which can be seen in [Figure 2.1](#). It is an omnidirectional mobile robot, this means that it uses meccanum wheels. This type uses several wheels on the brink of each of the four wheels. This small wheels are mounted diagonal to the big wheel and enable the robot to drive in each direction, without turning the big wheels' axis [20]. The omnirob has several modifications from the German Aerospace Center (*"Deutsches Zentrum für Luft- und Raumfahrt"*) ([DLR](#)), this includes for example the [ToF](#) cameras. These eight cameras are mounted around the ominRob.



Figure 2.1: The omniRob, with the [DLR](#) expansions. Source: [DLR](#) internal

2.2.2 Sensors

In this thesis two different types of sensors are used, both receive depth information.

ToF Cameras

This thesis describes a method to calibrate the extrinsic parameters of a Time of Flight Camera ([ToF](#)). [ToF](#) cameras measure the time the light needs to travel from the camera



Figure 2.2: The left picture contains the ToF camera, which is used on the omnirob.
 Source: DLR internal
 The Sick laser scanner is presented in the right picture [32].

to some reflecting object and back to the sensor. The time is then used to calculate the distance. The sensors, which are used in this thesis, are pictured on the left in Figure 2.2. These sensors have a resolution of 64×48 , that results in 3072 points per camera. The robot has eight of these cameras, which generate 24576 points per measurement. To ensure the right position of these 24576 points in the world, the determination of the camera pose must be precise. In Figure 2.3 the cameras' Fields of View (FoV) are represented with red pyramids. The volume of these solids can be seen by the cameras. Furthermore, this figure shows the pose of the cameras at the robot.

Laser Sensors

The ToF cameras are calibrated by using the Sick laser sensors, which are already mounted and calibrated by the manufacturer. They are represented on the right in Figure 2.2. This sensor type only perceives data in a plain. But the precision is usually higher than with ToF devices. The used laser sensor has a resolution of 541. This means it returns 541 distances to surfaces on the same level as the sensor. These laser sensors are mounted horizontally at the used robot and are already calibrated by KUKA. In Figure 2.3 the green lines, which are clearly visible in the lower right corner, are the laser beams of the laser scanner.

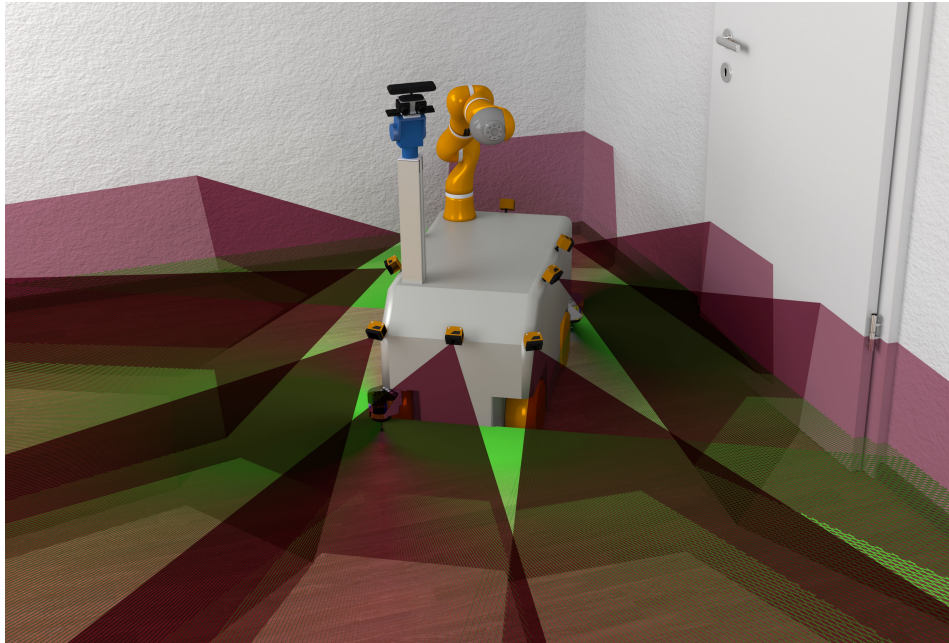


Figure 2.3: The FoVs of the [ToF](#) cameras are pictured with red pyramids. Green strips show the laser beams, they are clearly visible in the lower right corner.

2.3 L3D

The C++ Library Lib3D ([L3D](#)) was used for solving all robot tasks mentioned in this thesis. This library is developed at the [DLR](#) Institute of Robotics and Mechatronics and is still in development. It provides many tasks of a mobile robot as localization, mapping, path planning and execution, sensor simulation and [SLAM](#). Furthermore, it offers methods to process data and contains an [ICP](#) algorithm to align different point clouds, which is used in this thesis [\[4, 9\]](#).

2.4 Mobile Robot Environment

The Mobile Robot Environment ([MRE](#)) is essential for testing the [L3D](#) framework. The result of this thesis was tested in [MRE](#), which simulates the robot. This tool was developed by the institute and is able to simulate the sensors and a mobile platform, which behave like the robot in the real world. Furthermore, this software is used with the same interfaces like the real robot is used, so there is nearly no difference between the real robot and the simulation of it. This simulation was written to enable the developers fast testing of their software without needing to start the whole robot system and build an up specific environment. A part of the simulation is the visualization, which shows the robot in the environment.

The simulation tool consists of different parts, the following three parts are the most important for this thesis:

1. Simulation of the ToF and laser sensors, the simulation of these sensors use a polygon based model of the environment and calculates the distance from the position of the sensor to the next polygon. A gaussian model is used to add noise to this distance. The gaussian model based on the noise model given by the manufacturer.
2. Simulation of the mobile platform. This tool simulates the movement of the robot. The simulated motion depends on measurements, which were made to determine the precision of the robot's translation and rotation. This means motion noise models are used to simulate the robot as real as possible.
3. Visualization of the omniRob in the mobile laboratory in a viewer, represented in [Figure 2.4](#). The viewer enables a quick overview over the actual status of the system.



Figure 2.4: The omniRob is placed in the simulated mobile laboratory, presented in the viewer.

3 Concept

This chapter describes the underlying algorithms for the decision making library and the calibration process.

3.1 Decision Making

This thesis separates decision making in two classes. In both types a decider or a so called agent chooses the action performed next. The main objective of an agent is to pick the best combination of actions to maximize its own gain. In the first type the decider knows exactly all eventualities and what happens next. But this type is incompatible with the real world, because the states in the future only have a probability of occurrence, which can not be estimated definitely. The second type is oriented towards the real world, where the possibility of an entering a state is not 100.0% distinct. This means, a decision is a choice made between alternative courses of action in a situation of uncertainty [10].

3.1.1 Decision Tree

A decider can handle the uncertainty by using a decision tree, as depicted in [Figure 3.1](#). Such a tree consists of three types of nodes:

1. In random node a outcome is selected by chance. Each outcome of a random node has a probability of occurrence. This value can be estimated or measured by many repetitions. A random node is usually represented by a circle.
2. In a decision node the agent chooses a branch of the tree. It is represented by a rhombus. The selection of an action in a decision node can produce costs, which reduce the gain of this walkthrough.
3. The result nodes are the leaves of a decision tree. These nodes contain the result gained by an agent in the case the node is entered. This node type is diagrammed by a rectangle.

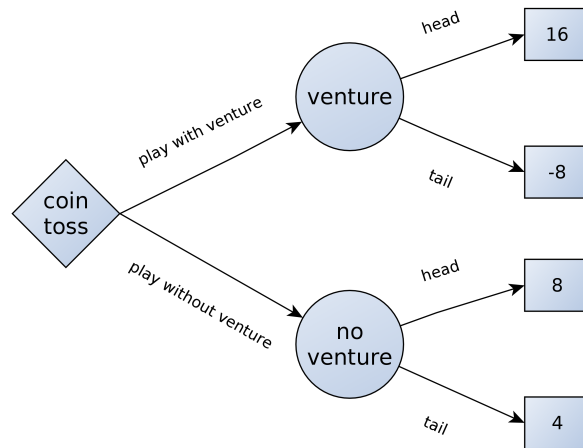


Figure 3.1: Two lotteries are pictured as a decision tree. In both a coin toss game is offered. The upper coin toss game has a higher venture and the lower game has a lower venture. An agent can decide between these two lotteries.

The root is the start point of a tree. The path down to a leaf contains all decision and random nodes, which are visited in this walkthrough.

The root of the tree is a decision node, which means that an agent can choose between the actions playing this game with venture or without venture. In the first lottery, the one with venture, an agent can win more than in the second. But it has the possibility to lose, in contrast to that in the second lottery an agent always win. Therefore, the second lottery has no venture. For instances an agent chooses "playing with venture" and attains the random node "venture". In this node a coin is tossed and the corresponding outcome is selected. In this example the outcome is head and the agent gain 16. If the outcome is tail, the agent will lose 8. For the lottery without venture an agent will gain 8 if the outcome is head else an agent will gain 4. This instance shows the function of a decision tree and where an agent is able to decide which branch he should take [29].

3.1.2 Strategy

An agent tries to maximize its gain. Therefore, it tries to find the best action in each decision node. The agent could use strategies to solve this problem. A strategy specifies for each decision node, which action should be selected. The amount $|s|$ of strategies is calculated by Equation 3.1.

$$[h]|s| = \prod_{k=1}^n |d_k| \quad (3.1)$$

In this equation the amount of actions in a decision node d_k is $|d_k|$. There are n decision nodes in a tree. The product of all these amounts is the quantity of strategies $|s|$, since a strategy is a permutation of all decision nodes' outcomes.

In [Figure 3.1](#) are two strategies, because the decision node has two outputs. The first strategy contains only the decision "play with venture" and the second strategy the action "play without venture".

Algorithm to determine the Strategies

The strategies are determined with an algorithm, which parses the tree and finds all possible strategies. The amount of strategies can be calculated by [Equation 3.1](#). For example a tree has seven decision nodes and each of these nodes has two children, as pictured in [Figure 3.2](#). Based on [Equation 3.1](#) this tree has 128 strategies. Moreover, each strategy contains seven decisions, one for each decision node. This means for this simple tree, all strategies have $128 \cdot 7 = 896$ decisions. The reason for this high number is that a strategy saves all decisions for each decision nodes even if the corresponding node is not entered. In a decision node a strategy decides, which action should be performed and thus which child is the next. Therefore, the quantity of decisions in a strategy can be reduced by removing the decisions of the tree, which are not reached by this strategy. An easy example can show this approach. For instance an agent has to choose between buying and not buying a car, shown in [Figure 3.3](#). If he chooses the second action, he will not have the possibility to choose the colour of the car. In the example explained above with the seven decision nodes, the amount of strategies would be reduced to eight.

In addition to reduce the amount of single decisions in each strategy, the implemented algorithm only saves the leaves of the possible paths through the decision tree. That is

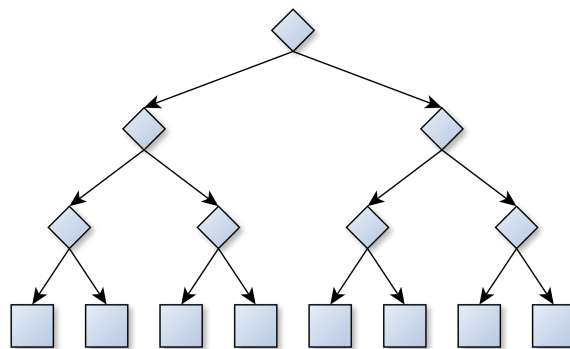


Figure 3.2: The pictured decision tree contains seven decision nodes, no random nodes and eight result nodes. It has 128 strategies based on [Equation 3.1](#).

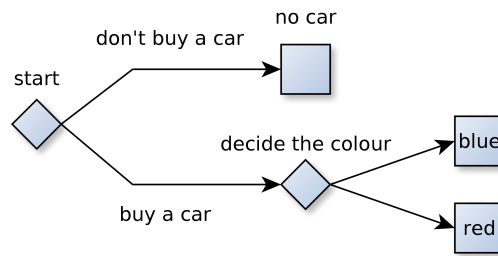


Figure 3.3: The tree represents an agent's process buying a car. An agent can decide between buying and not buying a car. If it decides to buy the car, it will be capable to choose the colour.

possible, because the paths between the root and the leaves are statically defined. Therefore, the rest of the decisions are implicitly given by the leaves. These two approaches save memory and enhance the performance, because in the example with seven decision nodes only eight leaves have to be saved instead of 896 decisions.

It is possible to find all strategies of a tree by parsing the tree and searching for each combination of actions. However, this approach is inefficient, because for each strategy the whole tree has to be parsed. This can be avoided by a recursive algorithm. Such a recursive algorithm starts in the root. The result of each visited node is the combination of the children's results. This means the algorithm walks down to a leaf and returns. After this return the algorithm walks up the tree to the next intersection, which could be the father of the leaf, too. In this node the algorithm start visiting the other children. This is continued till all leaves are visited. Afterwards the algorithm returns to the root. This process is pictured in [Figure 3.4](#). The first entering steps of the algorithm are shown

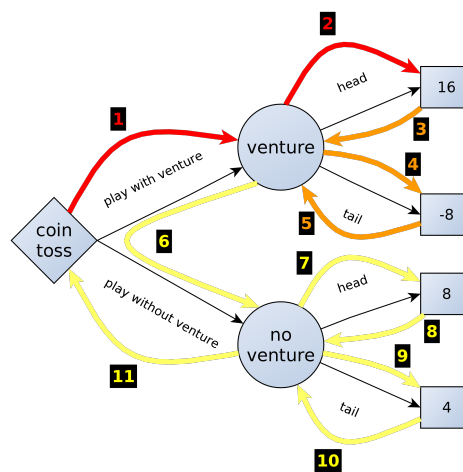


Figure 3.4: The functionality of the recursive algorithm for the strategies are shown here. The numbers on the arrows indicate the order of the execution.

with red arrows, after that the arrow color switches to orange. The last entering steps of the algorithm to find all strategies are shown in yellow. The numbers on the arrows indicate the order.

The combination process, which is performed in the recursive algorithm differs for each node type:

- Random node: The strategies do not contain the outcomes of a random node. Therefore, all children must be visited to guarantee that all strategies are found. This means in a random node a multiplication of the children's results is performed. Two strategies are multiplied, by combining each element of the first strategy with each element of the second strategy. A combination of two strategy elements is the concatenation of these elements, the order is not important. An element of the strategy is a leaf or the last point which is reached with this strategy. In an example with two children the first child has three strategies and the second four. The result would contain twelve strategies. In the case a node has only one child, all the results are copied. If there is more than one child, the results of the children are multiplied consecutively.
- Decision node: The strategies of the children are all copied in a decision node, because the children of this node contain the actions, which lead to them and therefore all children are needed. In an example with two children the first child has two strategies and the second four the father would have six single strategies.
- Result node: A result node, as a leaf of the tree, is the start of a new strategy, because this node specifies a range of actions, which lead to this leaf. This node creates an empty strategy and adds an information to identify this node again. Afterwards this strategy is returned to the father.

This means for one strategy, that the creation of it happens in a result node. New elements can only be added to this strategy in a random node. This will be possible, if the other children have strategies, which can be multiplied with this one. As mentioned before a decision node only copies the whole strategy, without changing the elements.

The algorithm will act different, if the actual node is a random node and the whole branch under this node does not contain any other decision nodes. In this case several new strategies would be generated in this branch and then consecutively multiplied in the random node. But these strategies do not have any new decisions, since a decision node is missing. Therefore, these strategies are not used. In order to achieve this, a random node, returns to the father a message. This message contains the information that the

father should use its own action to build up a new strategy.

In the case the subtrees of this father have no decision nodes either and is no decision node by itself, this father would behave like described above. He returns to its own father the same message, because he has no strategies to add. If this signal is moved upwards to the root, no strategies will be generated, because an agent must always reach a leaf by a selectable action.

Example

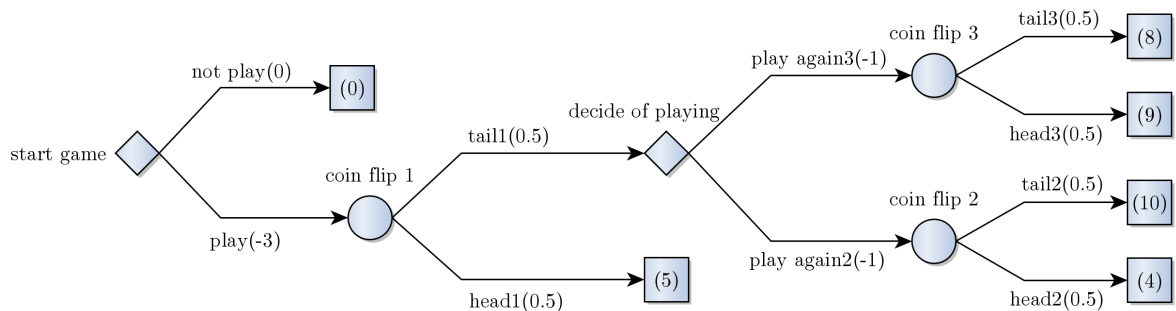


Figure 3.5: A decision tree, which pictures a coin flip game. The game will start with the possibility to select, if an agent wants to play this game or not. If it plays the game, the coin is tossed and the result will define the next node. If the result of the coin flip is head, an agent get a gain with a value of five. In the case the result is tail, an agent can decide between two different coin flip lotteries. If it decides to play the lower lottery, it has the possibility to win ten, if the result is tail, otherwise if it is head, the gain will be four. In the other case the upper lottery is played. If the random node "coin flip 3" has as result tail then the gain will be eight, else the result is head and the gain will be nine.

A decision tree is illustrated in [Figure 3.5](#). It represents a coin flip game, at the start the agent has the possibility to play the game or not. If he is not playing the game, the gain will be zero. In the case the agent decide to play the game, he has to pay three and then the random node "coin flip" is reached. The coin is flipped and the outcome determines which node is performed next. If the outcome is head the agent has no choice to make and he would take the gain of five, which is presented in the rectangle. In the event of tail the agent gets the possibility to decide between two different lotteries. Both actions to this coin flip produce costs with a value of one. In the upper coin flip with the name "coin flip 3" the outcome with tail is eight and with head nine. If an agent takes the lower "play again" action the "coin flip" will be reached and a coin is tossed. The result defines the gain of the agent, if it is tail an agent would gain ten, otherwise he would gain four.

This game has three strategies, to get this result the recursive algorithm from [Section 3.1.2](#) is used. The algorithm starts in the root and enters the upper child in [Figure 3.5](#). This child returns the strategy containing the action "not play". Afterwards the random node with the name "coin flip 1" is visited to calculate the strategies of this node the children are entered. The lower child, which is reached with the event of head, returns to its father that no actions are performed by this node or the children. Two strategies are the result of the upper child, one is "play again2" and the second is "play again3". The strategy "play again3" is created by the return of the random, which does not have any decision nodes. This return of the random node contains the message that the father should use the action which lead to the random node. The other action "play again 2" has the same procedure. These two strategies are return to the first coin flip, where the result is copied. This means the result of this first coin flip is the two strategies with the content ("play again2") and ("play again3"). The result is returned to the father and in this decision node with the name "start game" the strategies of the children are copied together. It means that this tree has three strategies.

3.1.3 Conversion from Decision Tree to Utility Table

As mentioned in [Section 2.1](#) every utility table has states of nature in the columns and actions in the rows. The actions in the rows of the utility table are the strategies of the decision tree. This means, a chosen action contains all decisions needed for this strategy. The columns of the utility table contain the states of nature of a decision tree. They are explained in the next section.

The conversion of a decision tree to a utility table offers the possibility to change the probabilities of the states of nature, during the execution. Furthermore, this technique will be useful if the probabilities for the states of nature are imprecise or are only known lying within a specified interval. For further information of interval probabilities see [\[1, 19\]](#). In addition, it is easier to design a decision tree, because it is possible for the creator to separate the single states and decisions. Therefore, the conversion is useful and needed to guarantee the future use of this software package.

States of Nature of a Decision Tree

The states of nature of a decision tree represent all possible outcomes for all random nodes in the decision tree. This is similar to the strategy, which is a combination of the

Table 3.1: Table for the states of nature for Figure 3.5 based on Equation 3.2

state of nature permutation nr.	c1	c2	c3
1	tail	tail	tail
2	tail	tail	head
3	tail	head	tail
4	tail	head	head
5	head	tail	tail
6	head	tail	head
7	head	head	tail
8	head	head	head

results of the decision nodes. The amount of states $|n|$ of nature can be calculated with Equation 3.2.

$$|n| = \prod_{j=1}^m |r_j| \quad (3.2)$$

A decision tree has m random nodes and in this equation $|r_j|$ is the amount of possible outcomes of each random node r_j . Equation 3.2 has the same rate of growth as the Equation 3.1 of the strategies. For the same reasons as in Section 3.1.2 the amount of these states of nature have to be reduced. Otherwise, the processing time growth is too high.

The states of nature of a decision tree can be calculated with a recursive algorithm. Without the optimization the result would be the permutation of all random nodes' outcomes. The number of states of nature can be reduced by combining all states of nature in which the difference can never happen simultaneously. This means the branches under the outcomes of a random node, which do not happen in this state of nature are not handled in it. For example in Figure 3.5 there are three random nodes, which both perform a coin flip. The outcome of the first coin flip is called $c1$. The second coin flip with the name "coin flip 2" is abbreviated with $c2$ and the random node named "coin flip 3" is $c3$. In this decision tree would be eight states of nature without the optimization. These states of nature are represented by Table 3.1. The optimized algorithm combines all $c1 = head$, because in this case the random nodes $c2$ and $c3$ are not entered anymore and therefore the results of them are not important. In Table 3.1 this would be the rows with the numbers from 5 to 8.

The amount of outcomes in a state of nature, can be reduced by just saving the ends of the possible paths through the decision tree. That is possible, because there is only one path between the root and the path end. Important to note is that these paths always

end in a random node's outcome. This implies that branches under random nodes, which do not contain any other random nodes, change nothing at the states of nature.

As mentioned before the state of nature algorithm is similar to the strategy algorithm. This means the states of nature are searched all at the same time to prevent the save of the visit of a branch. The following recursive algorithm calculates the states of nature for each node. It starts in the root of the decision tree and parses down to the leafs. The states of nature of the decision tree is the result of the root node. The execution of this algorithm differs for each node type:

- In a random node all states of nature are copied from the children, because the outcomes of this node lead to the leaves, which specify one path for each state of nature. In the case all children of a random node have no states of nature to add, the random node will add for each child a new state of nature. This is the only way to create new states of nature. For example with three children, where any has three states of nature to return, the father would have nine states of nature.
- In a decision node the results of the children are multiplied, since the state of nature does not define, which action will be selected by an agent. Therefore, the state of nature have to cover all possible outcomes of this decision node and this is possible by a multiplication. Two states of nature are multiplied by combining each element of the first state with each element of the second state. The combination of two state of nature elements is the concatenation of the elements, the order is not important. The results of the children are multiplied consecutively for the case that there is more than one child. In an example with two children each child has three states of nature. The amount of states of nature in this example would be nine.
- A result node has nothing to add and returns without doing anything.

Probabilities of the States of Nature

A state of nature has a conditional probability and this value is given in the decision tree. It is calculated by starting in the root of the decision tree. For performance reasons only the leaves of possible outcomes for one state of nature are saved. These leaves are now used to generate the path through the tree to them. For each new edge out of a random node the edge's weight is multiplied with the start value 1.0. This means all visited random node outcomes are multiplied together to receive the conditional probability of the state of nature. This is possible, because the random nodes directly under a decision node are independent and all probabilities are conditional.

Utility of a State of Nature with a Strategy

The utility of a strategy is the value of the gain received, by performing the actions, which are possible in this state of nature. Therefore, a state of nature defines for each random node, which outcome will be the next. Moreover, a strategy specifies for each decision node, which action should be performed by an agent. So a combination of both leads to exactly one way down the tree. This way can be seen in Figure 3.6. In the picture the decisions of this path are red and the random node outcomes are blue. As a result, the utility is the sum of all actions' gains on this specified path and the gain of the reached result node. Usually the action produces costs, but in the decision tree only one type is presentable and that's why the actions are gains too and therefore the values are negative. The gain for the path in Figure 3.6 is the sum of the actions "play" and "play again3" with the result node (9). This means the utility for this path is defined with $-3 + (-1) + 9 = 5$.

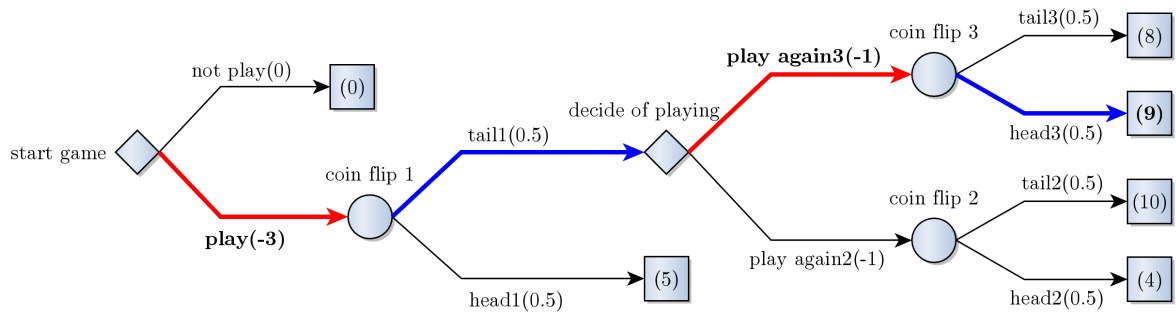


Figure 3.6: The decision tree shows the example that the state of nature is a combination out of "tail1" and "head3" and the strategy is a combination out of "play" and the "play again3". These combinations define the utility. It is reached, with the decisions in red and the random node outcomes in blue.

3.1.4 Roll Back Analysis

The roll-back-analysis is used to find the best action. This is similar to the decision criterion mentioned before. But it works on decision trees and do not need utility tables. The algorithm works recursively and the behaviour differs for each node type:

- In a random node, the probabilities of occurrence for the outcomes are multiplied with the corresponding results of the leaves. The final result of this random node is the sum of these products.
- The result of a decision node is the highest value of the children.
- In a result node the result value is the contained value.

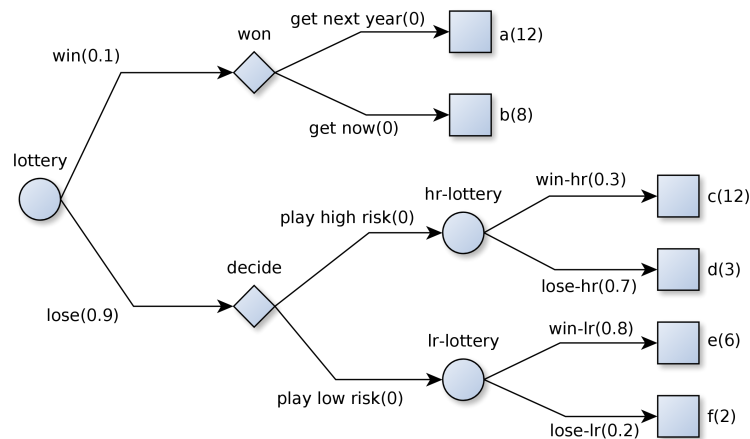


Figure 3.7: The simple tree provides a lottery, which has the possibility to play a high or low risk lottery in the case an agent lost the first one.

With this method the decision tree in Figure 3.1 has for the first action "play with venture" four and for the second action "play without venture" six, in the case the coin flip is fair. This means an agent, who always wants to maximize its own gain would take the second action "play without venture".

This best strategy can be used to walk through a decision tree. In the decision nodes the strategy defines which action should be taken and in the random nodes the nature decide what happens [29].

3.1.5 Decision Criteria

After the conversion of a decision tree into a utility table, the decisions criterion can be used to determine the strategy, which should be performed by an agent. In Section 2.1 different decision criteria are described.

Figure 3.7 presents a decision tree, which contains a lottery. The lottery has a chance to win of 0.1 and to lose the probability is 0.9. In the case an agent wins the gamble, it has to choose between getting now eight gain or in a year twelve. If an agent loses the lottery, it can decide between playing a high or a low risk lottery. If it chooses the high risk lottery, the probability to win twelve will be 0.3 and in case of losing the gain, it will be three. But if it selects the low risk lottery, it will be able to win six with a probability of 0.8. If an agent loses the low risk lottery, the gain will be two. This lottery was designed to show the difference between the various decision criteria.

This lottery pictured as a decision tree in Figure 3.7 can be converted by the algorithm

Table 3.2: The utility table is the conversion of [Figure 3.7](#). It contains the different states of nature and all possible and useful strategies.

state of nature	e(6)c(12)	e(6)d(3)	f(2)c(12)	f(2)d(3)	won
probability	0.216	0.504	0.054	0.126	0.1
play low risk,get now	6	6	2	2	8
play low risk,get next year	6	6	2	2	12
play high risk,get now	12	3	12	3	8
play high risk,get next year	12	3	12	3	12

presented in [Subsection 3.1.3](#). The result of this is shown in [Table 3.2](#). It contains strategies, states of nature, their probabilities and the corresponding utilities between a state of nature and a strategy. The labels of the states of nature are the leaves' labels, which can be reached by this state. The actions' labels are the combination of decisions, which will be performed if the strategy is selected.

Bayes Criterion

The Bayes criterion is exemplified here with the example out of [Table 3.2](#). The algorithm shown in [Subsection 2.1.1](#) is applied on the data of this table. Therefore, each row element is multiplied with the probability of the corresponding state of nature and after that all rows are summed up. Then the row with the highest value is selected and the action of this row shows the strategy, which should be performed. This approach uses the probabilities to get the strategy, which has the best gain depending on the probabilities. In the [Table 3.2](#) the algorithm would select the last action with the name "play high risk, get next year".

Minimax Criterion

The maximization of the minimum utility is called minimax criterion. It selects the minimum of each row, this means after that each action has one value and then the maximum of these values is taken. This method tries to maximize the lowest value. It is a pessimistic approach and in the [Table 3.2](#) the result would be "play high risk, get now". This strategy guarantees that the value is never smaller than three.

Maximax Criterion

The maximax criterion is an optimistic approach, which maximizes each row and then selects the action with the highest value. This can be useful, if the strategy with the highest gain is searched. In the [Table 3.2](#) the maximax criterion would return the action with the name "play low risk, get next year". Usually the result should be "play high risk, get next year". But both strategies have twelve as result. Therefore, the algorithm takes the first of them, which is "play low risk, get next year".

Maximin Criterion

In the case, a cost table is used instead of a utility table, a maximin criterion is used. This criterion is similar to the minimax criterion. The only difference is the temporal order. This criterion selects the highest value in a row and then minimize these values. It can be used to show the action, which should never be selected. For the [Table 3.2](#) the maximin criterion's result is the strategy with the name "play low risk, get now".

3.2 Calibration

The part of the calibration, which is treated in this thesis, are the extrinsic parameters. Explained in [Subsection 1.2.1](#), the extrinsic calibration is the precise determination of the camera frame's pose, relative to a known point. In the used case this known point is the **TCP**. This transformation **TCP** to camera $_T T^C$ is the result of the **ToF** calibration.

The transformation $_T T^C$ is represented by a matrix in $\mathbb{R}^{4 \times 4}$. The **TCP** describes the pose where the camera is mounted. This point could be at the end of a robot arm or in the present case at the center top of the robot, refer to [Figure 3.8](#). This figure shows the transformation represented by a red arrow from the base of the robot to the top of the center $_B T^T$. This point is the **TCP** point of the cameras. The transformation $_T T^C$ is pictured by a blue arrow, which starts at the **TCP** and points towards the camera frame. These two transformations combined in $_B T^C$ enable the robot to merge different data sets from different cameras into one coordinate system. This will be possible, if the camera transformation $_B T^{C_n}$ is known for all n cameras.

The primary idea of the calibration in this thesis is scanning the known calibration object with laser scanners and estimating the pose of this calibration object. Afterwards the

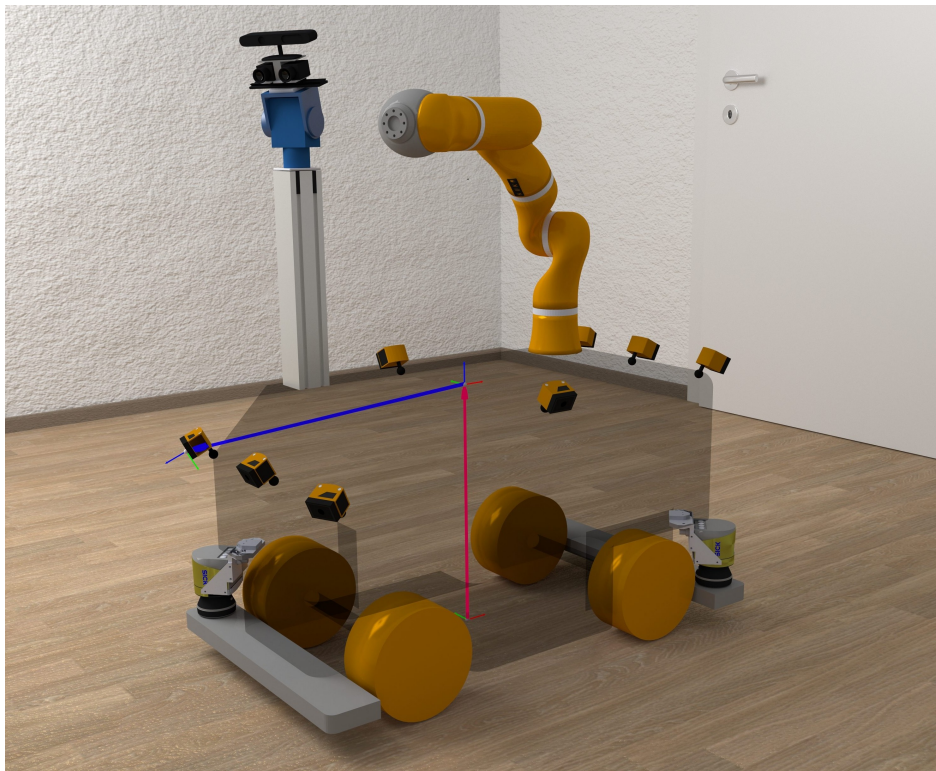


Figure 3.8: The blue vector describes $_T T^C$ and the red vector contains $_B T^T$, with B = robot base, T = TCP point and C = camera frame.

camera perceives the calibration object and estimates the pose of it, too. The difference of this two pose estimations is the transformation, which is added to the old calibration matrix of the camera, to receive the new wanted transformation ${}_T T^C$. The big advantage of this approach is the independence of the odometry. In contrast to that usual methods utilise the odometry to determine the pose of the robot in the world and compare that pose to the estimated pose of the camera [7].

3.2.1 Calibration Body

As described before there are different approaches to calibrate sensors on a mobile platform. In this thesis a photogrammetric calibration is used. Therefore, a new calibration object was created, which should enhance the calibration process.

The calibration body has to meet some criteria, which are described below:

1. Sensors do not perceive all materials equally well. This results in a demand of a good perceptible material.
2. The calibration body must have a wide base to enable the estimation of the pose with the laser sensors.
3. The depth image of a ToF camera must be filled up with the calibration body. This is necessary to guarantee as many data as possible used for the calibration process.
4. In order to find the rotation of the object in three DoFs the faces on top of the calibration body should point in different directions.
5. The weight of the body should be as low as possible.

These requirements result in a body with a wide base and a top body with faces, which point in different directions. The represented calibration body shown in Figure 3.9 is made out of styrofoam, because white styrofoam is well perceptible by the sensors. The base of the body is 15.0 cm high and at the front 1.0 m width. The back of the calibration body has a width of 1.5 m to increase the probability that the laser sensors of the robot see the left and right side of the base.

The cameras only see the top of the calibration body, therefore the body needs three faces which are orthogonal to each other. But this is not suitable, since three faces, which are orthogonal to each other build up a corner. This is a problem, because the ToF measurement can not be performed well in corners, since the influence of bouncing light is too high. Therefore, two faces, the bottom of the top and the main background,

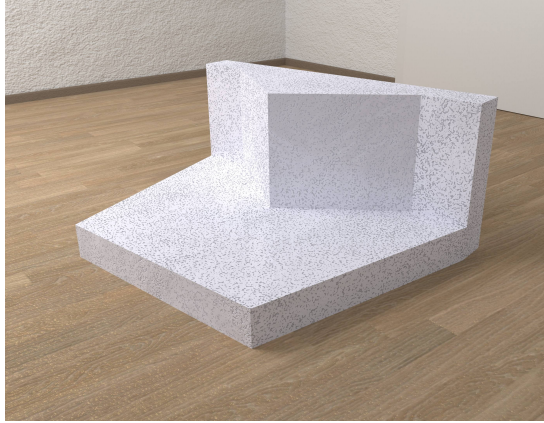
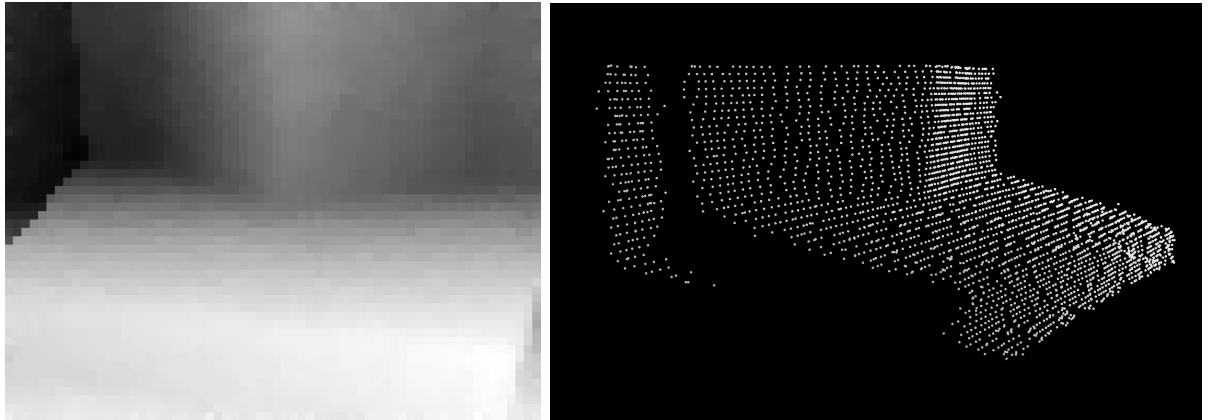


Figure 3.9: The calibration body is made out of styrofoam. The base is at the front 1.0 m width and at the back 1.5 m. The height at the back is 65 cm and at the front the calibration body is 15 cm. In the middle of the calibration body is a prism. This prism enables the detection of three different faces.

are orthogonal to each other. But with two faces the determination of only two DoFs is possible. To enable the detection of the third DoF a prism is added in the center of the background. The depth of the prism is 35.0 cm and the width is 1.0 m. This top layout enables the robot to determine the pose of the calibration object. The distance of the camera to this calibration object is deliberately chosen to ensure that the sensor only perceives the calibration object.

3.2.2 Laser Measurement

The Sick laser sensors, which were described in Section 2.2, are used to find the transformation from the laser sensors to the calibration object $_{laser}T^{CB\ pbl}$ (CB, pbl (pose by laser)). The laser makes a measurement and receives a plain of data points. Afterwards the amount of data is reduced with the knowledge of the estimated transformation $_{laser}T^{CB\ pbu}$ (pbu (pose by user)), which comes from a user. The result points should only contain the calibration object. They should not contain walls or other objects in the environment. The ICP is used to align these data sets. It calculates the distance between the points and tries to minimize this distance [4,9]. After about 100 iterations the result is the transformation $_{CB\ pbu}T^{CB\ pbl}$. This matrix transforms the estimated pose to the location, which is measured by the laser sensors. With this transformation it is possible to calculate the transformation $_{laser}T^{CB\ pbl}$ by multiplying $_{laser}T^{CB\ pbu}$ with $_{CB\ pbu}T^{CB\ pbl}$.



(a) A ToF camera's depth image

(b) The depth data of the ToF camera

Figure 3.10: The (a) picture represents the depth data. A brighter pixel means the point is closer to the camera. A darker pixel is therefore further afar. In the (b) picture the data out of the depth image was converted in a point cloud. In both data sets the wall behind the calibration body is clearly perceptible. The wall is shown on the right side in both pictures. This wall is filtered away to enhance the ICP algorithm.

3.2.3 ToF Measurement

In this step the ToF cameras measure the pose of the calibration object. In order to achieve this, the sensors perform a measurement. The cameras return a depth image shown in Figure 3.10a. Each pixel of this image pictures a depth value. A white pixel is a point, which is near to the camera and a black pixel is far away. This image is converted in a point cloud shown in Figure 3.10b.

In the depth image and in the point cloud are points, which belong to the wall behind the calibration body. These parts are represented by the black area in the depth image

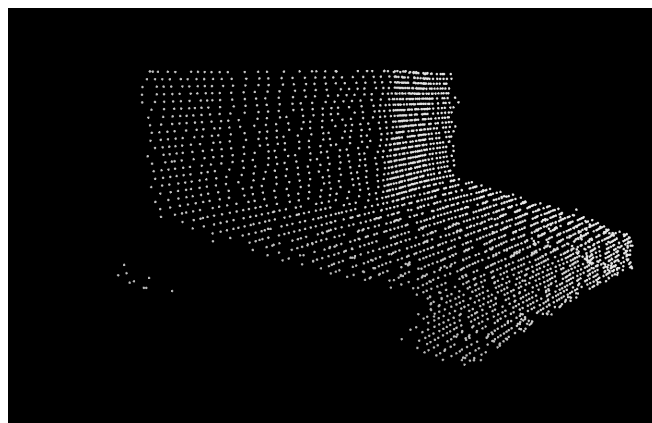


Figure 3.11: The point cloud used for the ICP algorithm. The wall on the left side behind the calibration body was filtered away.

and the points on the left in the point cloud in Figure 3.10. These points are filtered away under the condition that all points, which belong to the calibration body have to be in a radius of 1.0 m. This used transformation ${}_{laser}T^{CB\ pbl}$, which describes the center of this circle, was calculated in the last section. The resulting point cloud is pictured in Figure 3.11.

The ICP is used to align the filtered data from the camera with a point cloud model of the calibration body. The pose is given by the laser scanners ${}_{laser}T^{CB\ pbl}$. This means, the ICP returns a transformation ${}_{CB\ pbToF}T^{CB\ pbl}$ (pbToF (pose by ToF camera)). It is the difference between the pose of the calibration object from the laser ${}_{laser}T^{CB\ pbl}$ and the pose of the calibration body ${}_{ToF}T^{CB\ pbToF}$. This transformation is applied on the transformation ${}_{TCP}T^{ToF}$. The transformations are presented in Figure 3.12. This pictures shows the transformation ${}_{base}T^{TCP}$ in red and the transformation ${}_{TCP}T^{ToF}$ and ${}_{TCP}T^{laser}$ in blue, the lower sensor is the laser sensor and the upper sensor is the ToF camera. The estimated transformation from the laser ${}_{laser}T^{CB\ pbl}$ and from the camera ${}_{ToF}T^{CB\ pbToF}$ to the template are green. The upper arrow starts in the ToF and ends at the point, where

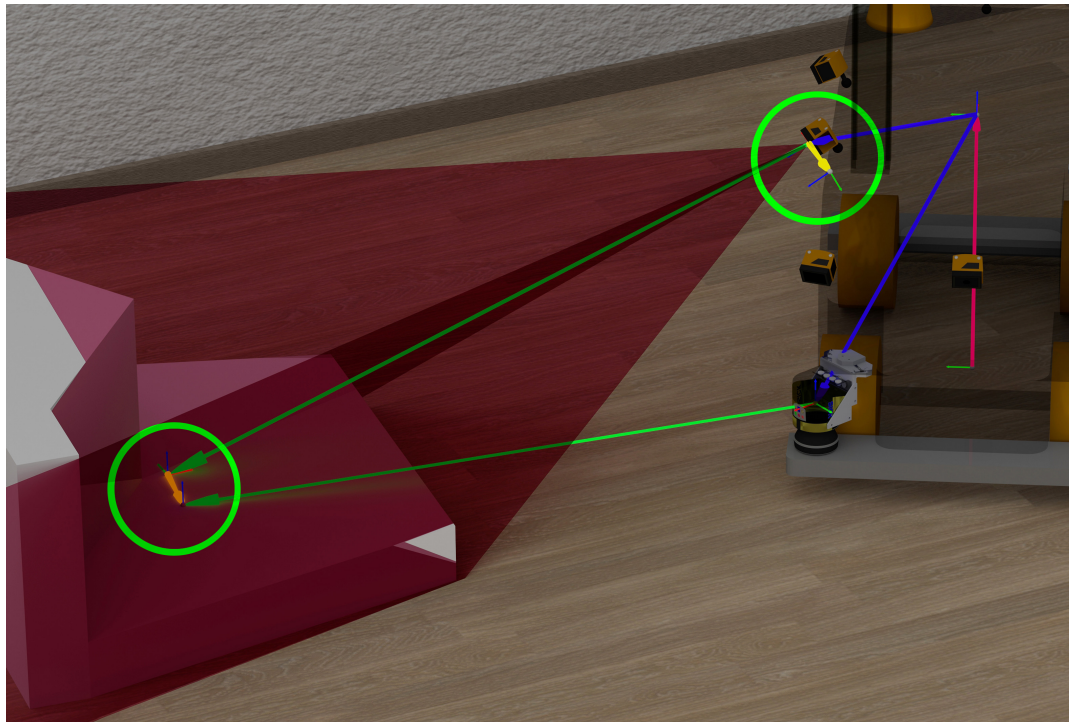


Figure 3.12: The yellow transformation surrounded by a green circle represents ${}_{CB\ pbToF}T^{CB\ pbl}$. It is the difference of the lower green transformation, which shows the point, where the laser scanner believes the CB is and the upper green arrow. This transformation points at the location, where the ToF camera measured the CB. The difference is the correction of the ToF camera's calibration matrix. Both sensors have a blue calibration matrix and the transformation between the robot's base and the TCP is red.

the template will lay if the calibration is right. The lower arrow represents the real pose, which was measured by the lasers before. The difference between these transformations can be used as correction of the ToF camera's calibration matrix. It is represented by a yellow arrow in Figure 3.12. Both yellow arrows are surrounded by a green circle. These arrows correct the pose of the ToF camera.

4 Implementation

The implementation of the algorithms, which were described in [Chapter 3](#), is explained in this chapter. First the decision maker is exemplified and after that the use of the decision maker in the calibration is explained.

4.1 Decision Maker

The decision maker or agent decides, which action should be chosen in a decision node. This concept is implemented by first parsing an input file with the file format Graph Modelling Language ([gml](#)) [[18](#)]. This file format is used, because it is human readable. Additionally, the software "yEd", which is able to change trees easily is available and the complexity is limited to graphs [[39](#)]. For example other formats like GraphML, often used to save graphs, is twice as big as a [gml](#) file. Furthermore, GraphML has a lot of functions, which are not necessary for a decision tree [[5](#)].

After generating the tree out of the [gml](#) file, the states of nature of the tree are found by the algorithm in [Section 3.1.3](#) and are put in the new generated utility table. Afterwards all strategies are searched with the algorithm out of [Section 3.1.2](#). The utility table is completed by the utilities. Each of them consists of a strategy and a state of nature. This utility value is the sum of all action utilities or costs and the value out of the result node. This can be used for finding the best decision with different decision criteria. These criteria describe the behaviour for choosing an action in a utility table, referred to [Section 2.1.2](#) and [Subsection 2.1.1](#).

After the selection of the best strategy an agent performs it. Therefore, the execution starts in the root node and in all occurring decision nodes the strategy defines, which action will be selected. In the random nodes, the implemented agent has to check the state of the system to decide the path the execution should follow. This execution is performed by an agent, which provides a function with the name "startActing()". It parses the tree and handles the results of the random nodes. The implementation of possible future agents is supported by setting up an easy agent interface, which only has one virtual function. This function has the name "performAction()" and takes only the

name of the action or the name of the random node and the type of the actual node. It returns the result of this node, in a random node this is the outcome, which represents the state of nature and in a decision node the actions name is returned.

An implementation of this agent only has to overwrite this one function and define for the names in the random nodes, what the system has to perform. For example the used agent for the calibration process can handle inputs like "isCalibrated" or "init". The first one is the name of a random node. The "performAction()" method returns as result the outcome of this random node. The second input can be an action name and this one could start the init process of the robot, which is needed for the calibration.

4.2 Use in Calibration

The approaches, which were developed above are used to calibrate the [ToF](#) cameras of the omniRob. In order to achieve this a decision tree was constructed, which contains different decisions and observing processes. This decision tree is represented in [Figure 4.3](#). The creation of this tree consists out of three iterations, which will be exemplified in this section.

4.2.1 First Iteration

At the beginning a simple decision tree was designed, which is capable to do a simple calibration. This tree is pictured in [Figure 4.1](#). This tree has no decision nodes with more than one decision, it was designed to test the calibration process and the capability of the framework for decision making. But the actions of the decision nodes can be executed by the robot and the outcomes of the random nodes can be recognized by the system. In

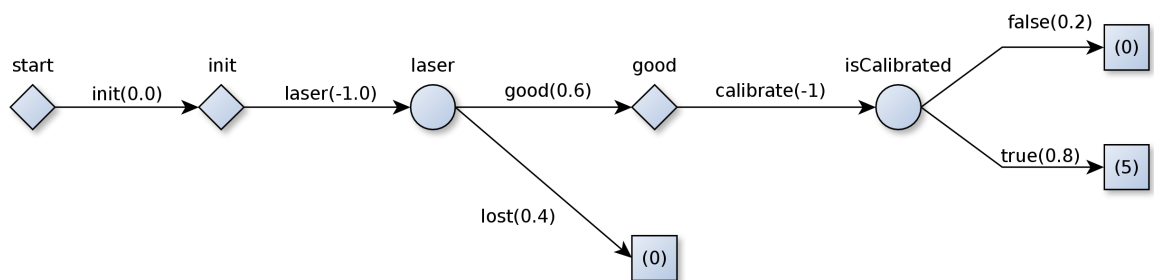


Figure 4.1: First iteration of the decision tree for the calibration process. It contains no decision nodes with more than one child and only two observing processes, which are named "laser" and "isCalibrated".

this figure the decision tree starts with the decision node "start", this is the starting point for the agent and at this point the agent only has the option to choose the init process. This selection will not produce any costs. In this init process the intern state is prepared, which contains the distance between the [ToF](#) camera and the body. This distance is necessary to ensure, that the full advantage of the camera's [FoV](#) is used. In the next step the agent performs a laser measurement and the random node with the name "laser" is reached. In this node the data from the laser measurement is evaluated and as described above the calibration body's pose is tried to estimate with these data. The result of the [ICP](#), which describes the correction of the user's estimation to the laser's estimation, contains additional data. These data allow the algorithm to determine the quality of the measurement. This quality is then used to decide, which outcome of the random node occurred. For example, the quality is "good" and therefore the corresponding decision node is entered. In this decision node the agent has no option at all and can only start the measurement of the [ToF](#) cameras. Afterwards, the same procedure as for the laser is performed. But the result will define if the data is used or not. In this example the output is "true", which means that this camera is now calibrated. In this example the result was "good", but for the case the [ICP](#) results are not good, the calibration would stop, with the result "lost".

The evaluation of the measurement's quality is realized by using the result of the [ICP](#). The implementation of the [ICP](#) algorithm returns some parameters after calculating the needed transformation to correct the data. These parameters contain the number of correspondences, which were found between this two data sets, and the mean error between them. Out of this two parameters the determination of the quality is possible.

4.2.2 Second Iteration

In the first iteration the agent has no choice at all. This can be changed by adding a new branch to the tree as pictured in [Figure 4.2](#). This branch is appended at the point, where the laser measurement is evaluated. This means, it will be possible to distinguish between a good and a bad estimation for the case the object was not found with the [ICP](#). If the [ICP](#) parameters indicate that the alignment did not work well, the bad outcome will be selected. In this node the agent can decide, if he tries to calibrate or to relocate the pose. In the relocation the robot tries to find a better pose than before by evaluating the laser data. After this relocation the laser measurement is performed again. If the result is bad again, the agent can only choose between stopping the calibration process or try to calibrate with the bad starting point.

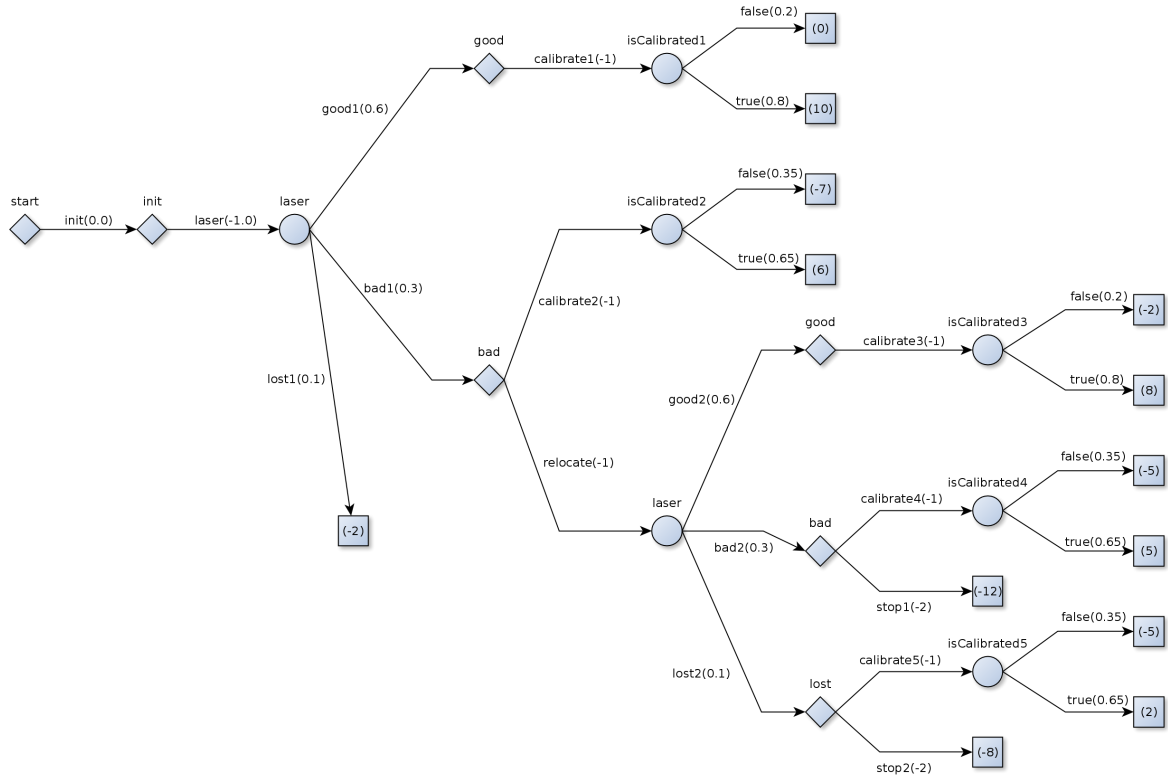


Figure 4.2: Decision tree for the calibration process in the second iteration. The tree has the "bad" case included and can handle this situation.

4.2.3 Third Iteration

At least the laser's result designated "lost" must be processed. In this case the ICP does not find the calibration body. Usually this means that the execution of the calibration must be stopped and a human actor must place the calibration body at a better pose. This can be avoided by performing a random shift. By chance this movement can improve the pose of the robot and enable a calibration. After this motion a laser measurement is performed and the same actions as before in the [Second Iteration](#) are now performed. There is only one difference. In the decision node "lost" an agent has to choose between making a calibration or stopping the execution.

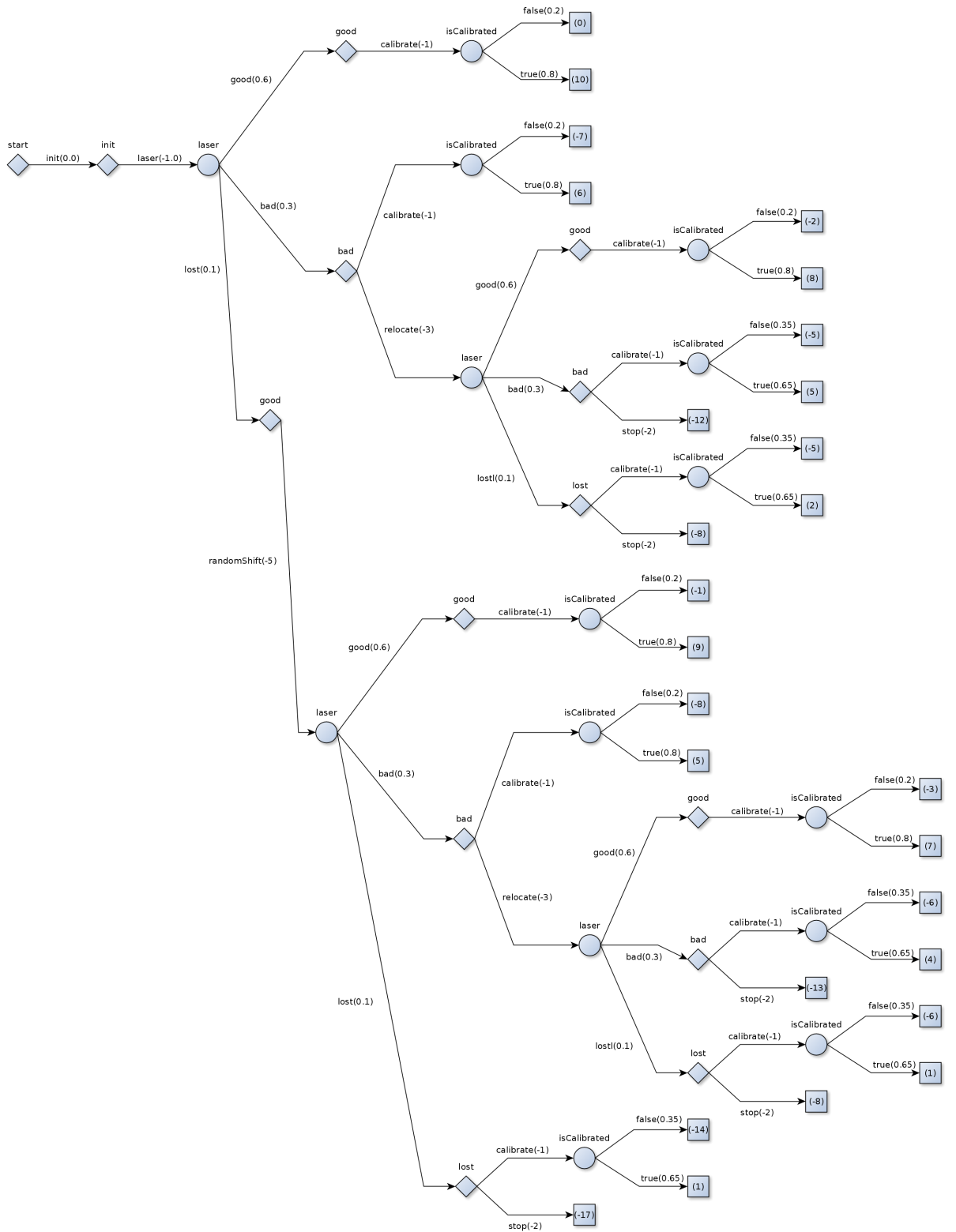


Figure 4.3: The decision tree provides the calibration for the ToF cameras. It is able to handle the "lost" scenario. In this case an agent makes a random shift and then relocates itself.

5 Test

The goal of this thesis is the creation of a framework for decision making. This framework is tested in this chapter. At the end the framework for decision making is combined with the calibration process.

5.1 Evaluation of the Decision Making Process

The evaluation was made with the decision tree out of the [Second Iteration](#), because the tree of the third iteration generates a table with 99 strategies and 30 different states of nature. It would not fit in this thesis. So the tree of the [Second Iteration](#) is converted in a utility table with the algorithm, presented in [Subsection 3.1.3](#). The algorithm's result is presented in [Table 5.1](#). The first rows contain the states of nature, which are generated in [Subsection 3.1.2](#). A presented state of nature, as defined in [Section 3.1.3](#), is the combination of random nodes' outcomes.

The names of these states of nature are a concatenation of the leaves' labels, which can be reached by the combination of these outcomes. For example, the third column of the upper table represents the state of nature with the name "(6)(2)". In the case this state happens, the name shows the labels of the leaves, which will be reachable. In [Figure 4.2](#) it is clearly visible that the leaves "(6)" and "(2)" are reachable over the outcome "bad1" in "laser". Afterwards "(2)" is reached over "isCalibrated2" with the result of "true". The walkthrough will end in leaf (6), if the second "laser" measurement has the result "lost2" and additionally the random node "isCalibrated5" returns true. This example describes the creation of the states' names.

The first columns of the table in [Table 5.1](#) contain the actions' names. They describe the actions, which were generated in [Subsection 3.1.2](#), these actions are the strategies of the decision tree. The names of the strategies are the last decisions, which an agent selects in a walkthrough. For instance the third row of both tables contain the same action named "calib1,calib2". This name contains the last two decisions, which will be selected, if this strategy is taken. For reasons of space the word "calibration" is abbreviated with "calib" in the tables. Refer to [Figure 4.2](#) an agent can reach the actions "calibration1"

and "calibration2" over two paths. Both ways begin in the node "start" and separate in the random node "laser". In the nodes after this separation the last decision is made. This would be "calibration1" in "good" and "calibration2" in "bad". There is no decision necessary for the result node "(-2)".

The utilities are calculated by using the algorithm shown in [Section 3.1.3](#). The decision tree can be parsed to get these values. During this walkthrough an outcome can be selected for each node, because in a random node the state of nature defines the next node and in a decision node the strategy specifies the next node. The utility in the second column and the third row of the upper table has a value of four. For this value the state of nature "(6)(-1)" and the strategy "calib1,calib2" are used.

Table 5.1: The result of the conversion, shown in [Subsection 3.1.3](#). The first rows contain the states of nature and under each state of nature the corresponding probability is labelled. The name of a state of nature consists of the leaves' labels, which can be reached with this state. The actions' names are in the first column under the probabilities, each name represents one path through the tree, because the name is the last actions' names, which can be performed in this walkthrough.

state of nature	(6)(-2)	(6)(5)	(-7)(-5)	(6)(2)	(6)(-5)	(-7)(5)	(6)(-5)	(-7)(2)
probability	0.023	0.038	0.011	0.012	0.02	0.02	0.006	0.006
calib1,calib2	4	4	-9	4	4	-9	4	-9
calib1,calib3,stop1,calib5	-5	-16	-16	-1	-16	-16	-8	-1
calib1,calib3,calib4,calib5	-5	2	-8	-1	-8	2	-8	-1
calib1,calib3,stop1,stop2	-5	-16	-16	-12	-16	-16	-12	-12
calib1,calib3,calib4,stop2	-5	2	-8	-12	-8	2	-12	-12

state of nature	(-7)(-2)	(10)	(0)	(6)(8)	(-2)	(-7)(-5)	(-7)(8)
probability	0.012	0.479	0.119	0.093	0.1	0.003	0.05
calib1,calib2	-9	8	-2	4	-3	-9	-9
calib1,calib3,stop1,calib5	-5	8	-2	5	-3	-8	5
calib1,calib3,calib4,calib5	-5	8	-2	5	-3	-8	5
calib1,calib3,stop1,stop2	-5	8	-2	5	-3	-12	5
calib1,calib3,calib4,stop2	-5	8	-2	5	-3	-12	5

The Bayes criterion is used on this [Table 5.1](#) and the result is "calib1, calib3, calib4, calib5". In contrast to that the other criteria, like maximin and maximax would take "calib1, calib2", because the result node with the label "(10)", has in all strategies the value eight. The criterion minimax would take "calib1, calib3, calib4, calib5" too.

5.2 Combination with the Calibration Process

In this section the calibration process is compared with and without the decision making framework. Both tests are made under the same conditions and show the advantages of the new approach.

5.2.1 Previous State

This work implemented an autonomous calibration process, which enables the robot to react to the results of different measurements. Before this work the robot has no possibility to detect a bad measurement and handle this situation. The table in Table 5.2 shows the translational errors of the calibration process in millimeter before the decision making framework was used. The translational error is the translational difference between the real pose and the calculated pose of the camera. The angular error of a ToF camera is linked to the translational error and therefore it is not shown in this thesis. The columns contain the translational errors for each ToF camera. The rows represent a test walkthrough of a ToF calibration for all eight cameras. The last column in Table 5.2 contains the translational errors' averages of a test. The averages of the translational errors of a camera are presented in the last row. For example the first test shows a good calibration process, where the error's average of all cameras is 69.0 mm. The average of the average of all cameras is shown in the lower right corner of the table. This means the average translational error of all cameras is 184.3 mm.

Table 5.2: This table presents the state before the decision making framework is used. The values are the translational errors in millimeter for each camera and test. A test is a complete calibration walkthrough. The last row contains the translational errors' averages for each ToF camera and in the last column the translational errors' average of each test is shown.

Without Decision Making:									
Test Nr. \ ToF Nr.	1	2	3	4	5	6	7	8	Average per test
1	49.4	90.4	125.1	62.0	93.8	79.2	25.7	26.6	69.0
2	50.7	84.4	100.5	804.1	93.1	28.1	28.7	42.4	154.0
3	49.3	154.5	99.5	654.0	96.9	144.7	34.2	85.7	164.9
4	49.0	83.7	94.3	35.4	88.0	204.0	24.7	1564.0	267.9
5	52.1	59.2	105.2	103.6	183.1	23.3	50.5	90.7	83.5
6	49.5	79.1	134.8	84.9	110.9	118.0	156.8	562.0	162.0
7	50.27	70.9	97.0	1767.0	74.7	1418.0	287.0	96.6	482.7
8	50.7	37.5	93.3	94.7	202.6	91.8	284.0	98.6	119.2
9	50.7	106.1	94.6	71.0	193.4	28.1	326.5	59.7	116.3
10	49.1	43.3	141.3	72.7	91.1	67.9	1272.3	48.6	223.3
Average per ToF	50.1	80.9	108.6	374.9	122.8	220.3	249.0	267.5	184.3

5.2.2 With the Decision Making Framework

The use of the decision making framework enables the robot to decide, which action should be performed next depending on the current circumstances. In Table 5.3 the translational error of the ToF cameras is presented. The error is like before the translational difference between the real and the calculated pose. This table has the same rows and columns layout as the table presented in Subsection 5.2.1.

The difference between the table in Subsection 5.2.1 and Table 5.3 is the absence of high errors. This means due to the decision process the agent is able to detect a bad laser measurement and then repeat this step to improve the precision. For example the translational error in test with number seven and camera four in Table 5.2 is 1767.0 mm, in contrast to that the highest value in Table 5.3 is 404.3 mm for camera six in test number two.

The approach presented in this thesis tries to detect bad laser measurements and the corresponding detection of the pose of the calibration body. If the estimation of the CB's pose is bad, the whole calibration of the actual camera will be wrong. Therefore, the detection of this scenario is useful and the results prove the enhancement of the calibration process.

Table 5.3: The table represents the state with the decision making framework. The translational errors in millimeter are shown for each ToF camera and test. This value shows the difference between the real pose and the pose calculated by the calibration process. A complete calibration walkthrough is called a test. In the last column the translational errors' averages of each test are shown and in the last row are the translational errors' averages for each ToF camera.

With Decision Making:									
Test Nr. \ ToF Nr.	1	2	3	4	5	6	7	8	Average per test
1	52.2	36.4	137.7	75.1	232.9	21.5	160.3	30.0	93.3
2	49.5	94.9	252.6	62.3	223.3	404.3	59.2	42.1	148.5
3	50.7	164.0	88.7	59.4	89.2	16.4	42.7	37.2	68.5
4	48.7	96.9	94.4	70.2	258.2	401.6	147.0	57.8	146.9
5	50.8	44.3	257.0	68.9	88.7	38.1	290.3	36.7	109.4
6	50.3	62.3	256.4	60.9	96.3	23.3	81.5	36.5	83.4
7	48.5	53.8	146.6	118.3	82.0	17.8	80.8	19.3	70.9
8	50.9	91.7	147.8	61.8	90.6	38.6	306.0	70.9	107.3
9	51.4	160.1	229.4	64.0	186.2	27.2	266.6	35.4	127.5
10	54.0	45.0	94.8	101.5	100.5	97.8	37.9	37.3	71.1
Average per ToF	50.7	84.9	170.5	74.2	144.8	108.7	147.2	40.3	102.7

6 Conclusion and Future Work

The goal of this thesis was to create a decision making framework, which contains an agent, which is able to evaluate utility tables and decision trees. This includes the conversion of a decision tree to a utility table. In addition an agent's interface should be provided. The implementation of these aims was shown in this thesis and an evaluation was made.

During the development of the framework several recursive algorithms were evolved, which detect the strategies and the states of nature of a decision tree. Moreover, they determine the states' of nature probabilities and the utilities. The algorithms were used to convert a decision tree in a utility table.

After the implementation and testing of the decision making framework, the combination with the [ToF](#) calibration was made. The evaluation of this combination shows that the calibration's result on its own is not precise enough to improve the pose estimation of the camera. However, the decision making framework improves the stability of the calibration and enables an autonomous detection of bad measurements and defines the behavior for such cases.

In a future work a new calibration process could be designed and combined with the decision making framework. This approach could use better laser sensors to improve the lasers' pose estimation. Furthermore the utility tables could use imprecise probabilities. This technique could describe states, which probabilities can not be expressed with one precise value. Moreover, a reinforcement learning approach could be added to improve the decision trees, which are created by a user. This can be done by randomly trying other strategies to control the utilities of the decision tree. The check can be done by evaluating the results of this random walkthrough.

The decision making framework could be used in different problems in mobile robotics. For example in a case, where a robot should make a coffee and bring it to someone. During the execution of this tasks a person could walk in the way of the robot. Now a programmer could handle all possible situations with several if-clauses or he uses a decision making framework, which provides the tools to enable the robot to decide by itself, what to do.

Bibliography

- [1] Thomas Augustin. On decision making under ambiguous prior and sampling information. In S. Moral G. de Cooman, T. Fine and T. Seidenfeld, editors, *Proceedings of the Second International Symposium on Imprecise Probabilities and Their Applications*, page 9–16, 2001.
- [2] Mr. Bayes and Mr. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs. *Philosophical Transactions (1683–1775)*, 1:370–418, 1763.
- [3] Daniel Bernoulli. Exposition of a new theory on the measurement of risk. *Econometrica: Journal of the Econometric Society*, 1:23–36, 1738.
- [4] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Transactions PAMI*, 14(2):239–256, February 1992.
- [5] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M Scott Marshall. Graphml progress report structural layer proposal. In *Graph Drawing*, pages 501–512. Springer, 2002.
- [6] Claus Brenner, Jan Boehm, and Jens Guehring. Photogrammetric calibration and accuracy evaluation of a cross-pattern stripe projector. In *Electronic Imaging'99*, pages 164–172. International Society for Optics and Photonics, 1998.
- [7] Andrea Censi, Antonio Franchi, Luca Marchionni, and Giuseppe Oriolo. Simultaneous calibration of odometry and sensor parameters for mobile robots. *IEEE Transactions on Robotics*, 29(2):475–492, 2013.
- [8] H.H. Chen. A screw motion approach to uniqueness analysis of head-eye geometry. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 145–151, 1991.
- [9] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.
- [10] Herman Chernoff and Lincoln E Moses. *Elementary decision theory*. Courier Dover Publications, 2012.

- [11] P.R. de Montmort. *Essai d'Analyse sur les jeux de Hazard*. Quillau, 1713.
- [12] Eduardo Fernandez-Moral, Javier González-Jiménez, Patrick Rives, and Vicente Arévalo. Extrinsic calibration of a set of range cameras in 5 seconds without pattern. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14–18, 2014*, Chicago, Illinois, USA, September 2014. IEEE.
- [13] David Freedman. Some issues in the foundation of statistics. In *Topics in the Foundation of Statistics*, pages 19–39. Springer, 1997.
- [14] Stefan Fuchs and Stefan May. Calibration and registration for precise surface reconstruction with time-of-flight cameras. *International Journal of Intelligent Systems Technologies and Applications*, 5(3):274–284, 2008.
- [15] Theo Gevers, Joost Van De Weijer, Harro Stokman, et al. Color feature detection. *Color image processing: methods and applications*, 9:203–226, 2006.
- [16] Sven Ove Hansson. *Decision theory*. Royal Institut of Technology(KTH), 1994.
- [17] Lionel Heng, Bo Li, and Marc Pollefeys. Camodocal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3–7, 2013*, pages 1793–1800, Tokyo, Japan, November 2013. IEEE.
- [18] Michael Himsolt. Gml: A portable graph file format. file format, University Passau, 2010.
- [19] Nathan Huntley. *Sequential decision making for choice functions on gambles*. PhD thesis, Durham University, 2011.
- [20] Bengt Erland Ilon. Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base, April 8 1975. US Patent 3,876,255.
- [21] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1990.
- [22] Anthony Kelly. *Decision making using game theory: an introduction for managers*. Cambridge University Press, 2003.
- [23] Soung Hie Kim and Byeong Seok Ahn. Interactive group decision making procedure under incomplete information. *European Journal of Operational Research*, 116(3):498–507, 1999.

- [24] Erich Leo Lehmann. *Some principles of the theory of testing hypotheses*. Springer, 1950.
- [25] Michael Lederman Littman. *Algorithms for sequential decision making*. PhD thesis, Brown University, 1996.
- [26] Ying-Cherng Lu and J.C.K. Chou. Eight-space quaternion approach for robotic hand-eye calibration. In *Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on*, volume 4, pages 3316–3321, 1995.
- [27] R. Duncan Luce and Howard Raiffa. *Games and decisions: Introduction and critical survey*. Courier Dover Publications, 2012.
- [28] Ariane Motsch. *Entscheidung bei partieller Information*. Gabler Verlag, 1995.
- [29] Howard Raiffa. *Einführung in die Entscheidungstheorie*. Oldenbourg Verlag, 1973.
- [30] Stuart Russell and Peter Norvig. Artificial intelligence a modern approach. *Learning*, 2(3):4, 2005.
- [31] Yiu Cheung Shiu and S. Ahmad. Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form $ax=xb$. *IEEE Transactions on Robotics and Automation*, 5(1):16–29, 1989.
- [32] Sick. *Sick Laser Sensor S300 Standard*, 2010.
- [33] Klaus H. Strobl. *A Flexible Approach to Close-Range 3-D Modeling*. Dissertation, Technische Universität München, München, 2014.
- [34] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.
- [35] Henry F Thorne. Tool center point calibration apparatus and method, October 10 1995. US Patent 5,457,367.
- [36] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, MA, 2005.
- [37] John Von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1947.
- [38] C-C Wang. Extrinsic calibration of a vision sensor mounted on a robot. *IEEE Transactions on Robotics and Automation*, 8(2):161–175, 1992.
- [39] Roland Wiese, Markus Eiglsperger, and Michael Kaufmann. *yfiles-visualization and automatic layout of graphs*. Springer, 2004.

- [40] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.
- [41] Hanqi Zhuang, Zvi S Roth, and Raghavan Sudhakar. Simultaneous robot/world and tool/flange calibration by solving homogeneous transformation equations of the form $ax=yb$. *IEEE Transactions on Robotics and Automation*, 10(4):549–554, 1994.

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Kempton, 20. March 2015

Maximilian Denninger

Authorization

I hereby authorize the university for applied sciences Kempton to publish the abstract of my work on e.g. printed media or a website.

Kempton, 20. March 2015

Maximilian Denninger