Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MASTERTHESIS

# Feature-Based Monte Carlo Localization in the RoboCup Humanoid Soccer League

vorgelegt von

Judith Hartfill

MIN-Fakultät

Fachbereich Informatik

Arbeitsbereich Technische Aspekte Multimodaler Systeme

Studiengang: M.Sc. Informatik

Matrikelnummer: 6417284

Abgabedatum: 30.09.2019

Erstgutachter: Prof. Dr. Jianwei Zhang

Zweitgutachter: M. Sc. Marc Bestmann

Betreuer: M. Sc. Michael Görner

## Abstract

Knowledge about the own pose in the world is essential for an autonomously acting robot. In this work a version of Monte Carlo Localization is developed, that is applicable in the RoboCup Humanoid Soccer League. It integrates several kinds of localization information which are retrieved from a 2D RGB image. The proposed approach was evaluated in terms of precision with four different information sources for localization and their combinations on simulated data. The results show that using the field markings as input usually resulted in the most accurate pose estimation. Using several kinds of localization information could further improve the results. Which information source to add seems to depend on the task: finding the pose (localization) or keeping track of the pose. In the localization experiment including field boundary detections reduced the error, while in the pose tracking experiment field marking features as corners and t-crossings resulted in less erroneous pose estimates.

## Zusammenfassung

Das Wissen um die eigene Position und Orientierung im Raum ist für einen autonom agierenden Roboter unerlässlich. In dieser Arbeit wird eine Version der Monte Carlo-Lokalisierung entwickelt, die in der RoboCup Humanoid Soccer League einsetzbar ist. Verschiedene Arten von Lokalisierungsinformationen werden dafür integriert, die aus einem 2D-RGB-Bild stammen. Der vorgeschlagene Ansatz wurde auf seine Genauigkeit mit vier verschiedenen Informationsquellen für die Lokalisierung und deren Kombinationen auf simulierten Daten evaluiert. Die Ergebnisse zeigen, dass die Verwendung der Feldmarkierungen in der Regel zu der genauesten Posenschätzung führte. Die Verwendung verschiedener Arten von Lokalisierungsinformationen könnte die Ergebnisse weiter verbessern. Welche Informationsquelle hinzugefügt werden soll, scheint von der Aufgabe abhängig zu sein: die Pose zu finden (Lokalisierung) oder die Pose zu tracken. Im Lokalisierungsexperiment mit Feldbegrenzungsdetektionen reduzierte sich der Fehler, während im Pose Tracking-Experiment die Markierungsmerkmale der Felder als Ecken und T-Kreuzungen zu genaueren Lokalisierungsschätzungen führten.

# Contents

# List of Figures

# List of Listings

# 1. Introduction

Knowing the own pose is crucial for a soccer player to act successfully and so it is for an autonomous robot, that plays soccer. Without knowledge on the own position and orientation (pose) on the playing field, it is hard to distinguish the own goal from the opponents one, and scoring an own goal could cause losing the game. However, for soccer-playing robots knowing the own pose is also important for strategic team play: The can communicate their current poses and decide on the best strategy as a team, just like real soccer players.

On a soccer field different things can provide orientation on the current pose: the goals, the field markings or the dimension of the field itself. While the lines are often in the field of view, the goals are not, but when they are, better conclusions on the observer's pose can be drawn. Moreover, differences in the reliability of detection need to be considered for localization: While far away field markings can be hard to detect, goals post are well visible over the whole field. The features of the playing field can also be considered on different levels of abstraction: The center circle can be seen as white color on the field, just like or other lines, or it can be seen as the one circle whereas the other field markings are all straight lines. Just like the difference between lines and goals discussed above, this provides different information for localization.

So using several information sources at the same time can make localization more precise and more stable than relying on one type only. field of play itself. One strategy to overcome that problem is to include information from outside the field. One approach for this is the *visual compass* [Lab04], which estimates the robots pose based on odometry and changes in the detected image. Such an approach bears the risk to fail if the surrounding changes dynamically. Another strategy is to include prior knowledge into the pose estimation. In a RoboCup Soccer game, such information is provided, just like it is in a real soccer game. Before the game, it is decided by coin toss which teams attacks which goal. So at the beginning of the game, it is possible to determine the robot's pose on the field half. Once the correct pose is found, it needs to be tracked.

Robots that participate in the RoboCup Humanoid Soccer League need to be as humanoid as possible, especially in terms of kinematic structure and used sensor types. The field surface is made of artificial turf, making stable biped walking hard. So simple pose estimation techniques like dead reckoning are noisy and are prone to drift over time and thus are not applicable in that context. A suitable localization method is the Monte Carlo Localization, which is based on particle filtering. It is a lightweight approach, that can handle noisy input and works on nonlinear distributions.

## 1.1. Thesis Goals

The goal of this thesis is to develop and evaluate a 3D Monte Carlo Localization in the RoboCup Humanoid Soccer context. It should be easily portable to other codebases and adaptable.It should work on different kinds of localization information simultaneously. Furthermore, the localization should include prior knowledge provided by the game. This thesis aims at evaluating which combinations of features of the playing field are the most valuable information sources for localization and pose tracking.

In Chapter 2, the context of this work is presented, and an overview of the main concepts used for development is given. Chapter 3 presents work in the field of Monte Carlo Localization, with focus on humanoid robots. The approach of this work, including the localization algorithm itself, a module handling the initialization and the developed visualization, are presented. In Chapter 5, the two experiments on the efficiency of different kinds of localization information and their combinations are described. Finally, in Chapter 6, the results are discussed, a conclusion is given, and future work is proposed.

# 2. Fundamentals

The following sections give an overview of the context in which this work is done and the main concepts used. In Section 2.1 the RoboCup is presented shortly, with particular focus on the rules in the Humanoid Soccer Leagues in Section 2.1.1 and the team Hamburg Bit-Bots in Section 2.1.2 with the relevant aspects of software and hardware for this work. Section 2.2 provides a short overview of the components of ROS, the Robot Operating System, that are used in this work. A brief description on the concept of forward kinematics is given in Section 2.3 and odometry is shortly presented in Chapter 2.4. In Section 2.5, the Bayes filter is presented, and the particle filter and Monte Carlo Localization are derived from it. The Robot Motion Model and the Measurement Model used in the Monte Carlo Localization are presented shortly in Section 2.6 and 2.7.

## 2.1. RoboCup

The RoboCup Federation [1] [KAK+97] annually organizes robot competitions to promote research in autonomous robotics. Moreover, it promotes research to a broad public while providing the possibility to compare scientific approaches directly in a competition. In 2018 the RoboCup took place in Montreal, Canada, with around 400 participants and 5000 robots [2]. The competition is divided into four big leagues: Rescue, @Home, Industrial and Soccer, with several sub-leagues each, all with different research priorities. The focus of this work is on the Soccer Leagues, in particular on the humanoid ones, which are explained in more detail in the following.

### 2.1.1. Humanoid Soccer Leagues

In the Humanoid Soccer Leagues, robots compete against each other autonomously in soccer games. The rules are based on the FIFA rules and define the field of play as well as the restrictions of the robots. Each year rule changes are made to advance research on specific problems. Until 2012 the goals were different in color, which made localization easier.

The league is divided into three sub-leagues, the KidSize, TeenSize, and AdultSize. The rules between the leagues differ in player count on a team, ball size, and field size. The AdultSize League has some special rules, like the robots may be saved from falling by a human, to prevent damages caused by impact. In the following, aspects of the rules [Rob19] relevant for self-localization are explained.

### Robots

The robots need to be shaped like a human. They must have two legs, two arms and a head with specific proportions. The restrictions for the robot heights in the leagues can be seen in Table 2.1.

| | Kid Size | Teen Size | Adult Size |
|---|---|---|---|
| Robot Height | 40 - 90 cm | 80 - 140 cm | 130 -180 cm |

Table 2.1.: Robot height restrictions in the Humanoid Soccer Leagues. [Rob19]

Furthermore, they are only allowed to be equipped with human-like sensors, which must be at the corresponding humanoid positions in the robot. Up to two cameras are allowed, with a total field of view of 180°. The maximum pan-tilt motion of the head must be humanoid. The cameras need to be passive, so visual sensors like laser range finders or other depth cameras other than stereo cameras are not allowed. While a game is running, they must act autonomously. The robots may communicate with each other via WiFi.

### Field

In Figure 2.2 the field as it is defined in the laws of the game is visualized and in Table 2.2 the corresponding values for the Humanoid Leagues can be found.

It defines the overall size of the playing field, the size of the field markings, and the position and size of the goals. The line width is specified to be at least 5 cm.

The field surface is artificial turf with a height of about 3 cm. Robots in the KidSize and TeenSize League fall frequently, because of the unevenness of the field's surface.

### Referee, Gamecontroller & Robot Handler

Similar to regular soccer games, there are referees in RoboCup Soccer. Each field of play includes a computer running the *GameController* software, which is operated by the second referee. All referee decisions are fed into the GameController, which communicates this information via WiFi to the robots. Each team needs to provide a *robot handler*. The duty of the robot handler is to remove a robot from play when instructed to do so by the referee or reposition them for reentering the game.

### The Game

A game is played in two halves of 10 minutes. To start the game or after a goal was scored the kick-off procedure is performed. When the referee gives the signal *READY* all robots have to go to their own half. The ball lies on the center mark. Then the referee gives the signal *SET* and illegally positioned robots are removed by the robot handler. A robot is illegally positioned if it is not inside its own half or inside the center circle for the team not having kick-off. The team having kick-off may manually place its goalkeeper robot and a robot for

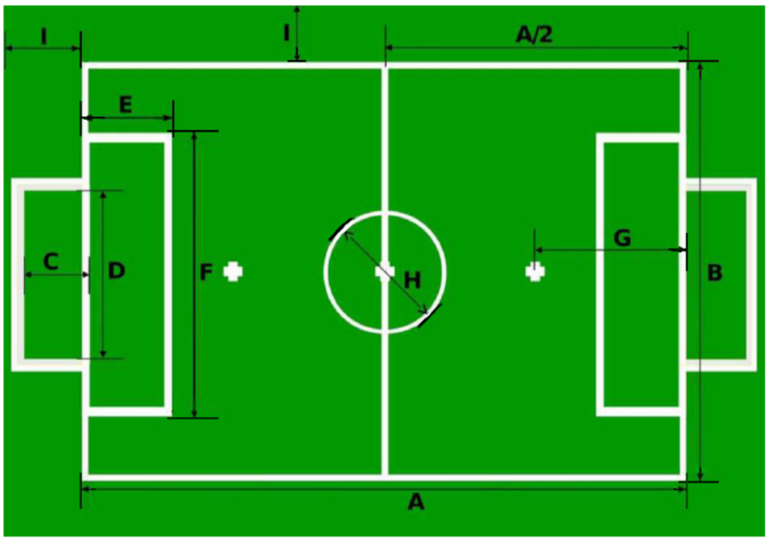


Figure 2.1.: Field of play in the RoboCup Humanoid Soccer Leagues [Rob19].

| | | Kid & Teen Size | Adult Size |
|---|---|---|---|
| A | Field length | 9 m | 14 m |
| B | Field width | 6 m | 9 m |
| C | Goal depth | 0.6 m | 0.6 m |
| D | Goal width | 2.6 m | 2.6 m |
| | Goal height | 1.8 m | 1.8 m |
| E | Goal area length | 1 m | 1 m |
| F | Goal area width | 5 m | 5 m |
| G | Penalty mark distance | 1.5 m | 2.1 m |
| H | Center circle diameter | 1.5 m | 3 m |
| I | Border strip width | 0.7 m | 1 m |

Table 2.2.: Dimensions of the field of play in the Humanoid Soccer Leagues [Rob19].

doing the kick-off. The goalkeeper needs to be somewhere inside the goal area, touching the goal line. The position of the robot doing the kick-off only needs to be somewhere inside the teams half, but it is common in the league to place it close to the center mark, facing the opposite goal. Then the referee gives the signal *PLAY* and the team having kick-off is allowed to play the ball.

The referee can give *removal penalties* to robots. It is a time penalty of 30 seconds. The robot is removed by the robot handler. After the end of the penalty, it needs to reenter the field from the touchline of the team's own half, at the height of the penalty mark. The referee indicates the side.

### 2.1.2. Hamburg Bit-Bots

The Hamburg Bit-Bots have participated in the Humanoid Kid Size League since 2011. A short overview of the relevant hardware and software used in this work is given.

#### The Wolfgang Platform

The *Wolfgang* robot is a humanoid robot with 20 degrees of freedom. The joints are actuated by Robotis Dynamixel MX64 [3] and MX106 [4] motors, which are equipped with rotary encoders. It is equipped with three computation units: An Intel Nuc, which handles most of the software stack, an Nvidia Jetson TX2, handling the image acquisition and computer vision and an Odroid XU4. The camera of *Wolfgang* is the Basler ace acA2040-35gc [5], which is equipped with a Computar M1214-MP2 lens with a diagonal opening angle of 49.2° [6]. Furthermore, the robot is equipped with an Inertial Measurement Unit (IMU) and foot pressure sensors.

#### Vision

The Hamburg Bit-Bots vision pipeline is presented in [FBG+19]. The relevant aspects for this work are the detection of field markings, the field boundary, and the goals posts, which are presented in the following.

In the vision pipeline, different modules handle specific object detection tasks. The *color detector* matches given pixels to the corresponding color space. The color spaces are configured by human operators before the game. The color space used to detect lines is defined by upper and lower bounds for each channel of the hue, saturation, value (HSV) space. For the color space defining the green of the artificial turf, a lookup table is generated using a specific tool. This way, more specific shades of green can be included, which is necessary, as the color highly depends on the viewing angle and the lighting conditions.

The green color space is used in the *field boundary detector*. To detect the outer edge of the field, the field boundary detector scans the image for the uppermost detection of green. The number of scanned rows can be configured to save resources, When the field boundary is blocked by an obstacles, it forms a dent under that obstacle in the image. To smooth the detected field boundary and make it more accurate, the convex hull over it is calculated.

The *line detector* detects points in the image, that are located on the field markings, instead of complete lines, to reduce computational effort. Random pixels in the image are checked against the color space for lines. Three strategies are applied to improve performance. First of all, no points above the highest point of the field boundary are checked, as this could lead to many false-positive detections. Secondly, with the height of the image, the density of randomly chosen pixels is increased. This is done, since a point higher up in the image corresponds to a point that is further away. The increase in density compensates the decreased size of features caused by the perspective transformation. To compensate for that, the density of randomly chosen pixel is increased in higher regions of the image.

For the goalpost detection the Hamburg Bit-Bots changed from the conventional method based on color spaces to using YOLOv3 [RF18], a convolution neural network (CNN). It was

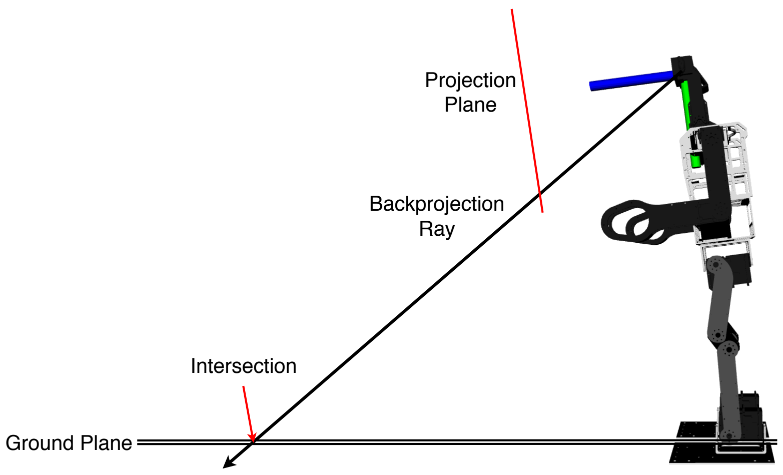


Figure 2.2.: Visualization of transformation from image space to Cartesian space [Gü18]. The projection plane is calculated based on the camera matrix. A ray is cast from the camera frame through the point on the projection plane which corresponds with the detection in the image. The intersection of that ray with the ground plane is the detection's position in Cartesian space.

trained on images taken on RoboCup competitions as well as images taken in the Bit-Bots test laboratory.

### Transformation from Image Space to Cartesian Space

The objects detected in the image need to be transformed into Cartesian space to contain valuable information. As the camera does not provide depth information, the transformation needs to be done based on prior knowledge.

It is assumed that when the robot is standing on the field its feet are parallel to the planar surface of the field. Using forward kinematics (see Section 2.3), the camera's pose relative to the ground plane can be calculated. Furthermore it is assumed, that all detection are on this plane. This is clearly true for planar objects like line points and field boundary points, but also detection of goal posts are provided in a form that represents their lowest point, right above the field. Given the camera matrix the projection plane can be calculated. A ray is cast from the camera through the projection plane at the corresponding point of detection in the image. The intersection of that ray with the ground plane is the point in Cartesian space relative to the robot.

As the playing field consists of soft artificial turf, the feet are not always parallel to the ground plane. This especially affects points that are projected far away from the robot. They underlie larger errors in Cartesian space than points that are projected closer to the robot.

### Hardware Control Manager

The robot's state is controlled by the state machine of the Bit-Bots Hardware Control Manager (HCM). The HCM keeps track of the robot's current state, handles the getting up motion after a fall, and decides which nodes may write motor goals. It provides information that is valuable for localization, such as if the robot is currently falling or already fallen. In these situations, the transformation of data from image space to Cartesian is not working correctly, see Section 2.1.2. So this information should not be used for localization. Furthermore, the robot's pose after falling and getting up is not necessarily the same as before the fall. This needs to be handled by the localization.

## 2.2. ROS

The Robot Operating System (ROS), is an open-source middleware between the actual operating system and the application [QCG+09] [7]. It provides a structure in which software components, called *nodes*, can communicate. The communication is done via *messages*. Nodes can publish data in the format defined by the message to a *topic*. Messages are received by subscribing to a topic. Several Nodes can publish to one topic, and several nodes can subscribe. The ROS master handles the connections between publishers and subscribers. The communication happens directly between the nodes. If the one-way communication provided by messages is not sufficient, *services* can be used. These are particular kinds of messages that have two parts: one for the request and the other one for the response. A node can offer a service, and a client can send the request message and then wait for the reply. The ROS software framework provides tools for debugging and visualization, such as *rviz* [8] for 3D visualization.

## 2.3. Forward Kinematics

In a kinematic chain like a robotic arm, the pose of the end effector relative the robot's base frame can be calculated with forward kinematics [SK16]. In ROS these calculations can be done by the *robot_state_publisher* [9] node, based on given joint angles and the robot model. The resulting coordinate frames can be handled by the transform library tf2 [10]. If the frames are connected in a tree-like structure, tf2 can calculate the transformation between any two frames in that tree. It stores the frames so that the transformation can also be done in retrospect. This is important in two ways. Firstly it possible to get transformations between two coordinate frames from a particular point in time in the past. For this work, this is important for the transformation process from image space to Cartesian space, see Section 2.1.2. Due to the processing time of the vision pipeline, the corresponding pose of the camera for the current visual data lies in the past. Secondly, the change over time between coordinate systems can be tracked. In this work, this is necessary to get the robots last motion.

## 2.4. Odometry

Odometry refers to measuring the changes in the robot's joint angles to estimate its motion [SK16]. Based on this estimated motion, the change of the robot's pose over time can be estimated. Each pose estimate depends on the previous one. So the errors of each prediction step add up, making the estimated pose drift over time. In the Bit-Bots software, according to the ROS Enhancement Proposals (REP) 105 [11] and 120 [12] the odometry data is published as a frame relative to the robots base_link which is located in its torso, and the base_footprint frame is published relative to the base_link.

## 2.5. Bayes Filter, Particle Filter and Monte Carlo Localization

This section is based on [TBF05]. With Bayes filters, a belief distribution, that represents the current state of the robot is calculated. It is a recursive approach, as the belief at time $t_x$ ($bel(t_x)$) depends on the previous belief ($bel(t_{x-1})$). Additionally it takes as input the most recent control ($u_t$) and measurement ($z_t$) data. For all belief states $x_t$ first the *prediction step* is performed, see Listing 2.1, Line 3, in which the control data that represents the robots last movement in the world is applied. On the humanoid robot used in this work, it is the leg's movement, that forms steps and thus changes the robot's state. Then on the resulting prediction ($\overline{bel}(x_t)$) in line 4 the *measurement update* is done. Here the prediction is multiplied with the probability to actually observe the measurement $z_t$ in that state.

```
1  Algorithm bayes_filter($bel(x_{t-1})$, $u_t$, $z_t$):
2    for all $x_t$ do
3      $\overline{bel(x_t)}$ = $\int p(x_t|u_t, x_{t-1})\ bel(x_{t-1})\ dx_{t-1}$
4      $bel(x_t)$ = $\eta\ p(z_t|x_t)\ \overline{bel(x_t)}$
5    endfor
6    return $bel(x_t)$
```

Listing 2.1: Bayes filtering algorithm. [TBF05]

Beliefs usually have to be approximated in robotics, as they could only be calculated precisely for very simple scenarios. There are two prominent ways to do so: Gaussian based filters, like the Kalman filter, that work on unimodal Gaussian distributions. The Kalman filter can be extended to nonlinear problems. In such an extended Kalman filter (EKF), the nonlinear function can be described by its tangent, which is linear.

The unscented Kalman filter (UKF) calculates a linearized approximation of the nonlinear function. Particle filters, on the other hand, approximate the posterior by a finite number of samples. The particle filter algorithm is based on the `bayes_filter` algorithm presented above, but the belief state is represented by so-called *particles*, which are samples drawn from the belief distribution. It takes as input a set of $N$ particles $x_{t-1}$, representing the last belief, the most recent control data $u_t$ and measurements $z_t$ and outputs a new set of particles $x_t$, representing the new belief. In listing 2.2, line 4, a predicted particle ${x_t}^{[n]}$ is sampled given the

the old particle $x_{t-1}^{[n]}$ and the current control data $u_t$. This corresponds to the *prediction step* of the Bayes filter. In line 5 the *measurement update* is performed, which in particle filters calculates the weight of each particle $w_t^{[n]}$. It equals the probability of sensing the current measurement data, given the predicted particle $(p(x_t|u_t, x_{t-1}^{[n]}))$. The distribution over all weighted particles is an approximation of the belief distribution $bel(x_t)$ in the Bayes filter. In particle filters, the resampling step is done additionally. In this step, for each particle, a particle of that set is drawn with replacement according to the weight distribution $w_t$. It focuses the computational resources on the most likely hypotheses by rejecting highly unlikely ones. The resulting set represents the updated belief $x_t$.

```
 1  Algorithm Particle_filter($X_{t-1}$, $u_t$, $z_t$):
 2    $\bar{X}_t$ = $X_t$ = $\emptyset$
 3    for $n = 1$ to $N$ do
 4      sample $x_t^{[n]} \sim p(x_t|u_t, x_{t-1}^{[n]})$
 5      $w_t^{[n]} = p(z_t|x_t^{[n]})$
 6      $\bar{X}_t = \bar{X}_t + \langle x_t^{[n]}, w_t^{[n]} \rangle$
 7    endfor
 8    for $n = 1$ to $N$ do
 9      draw $i$ with probability $\propto w_t^{[i]}$
10      add $x_t^{[n]}$ to $X_t$
11    endfor
12    return $X_t$
```

Listing 2.2: Particle filter algorithm [TBF05]

The Monte Carlo Localization is an algorithm based on particle filters [TBF05]. In addition to the particle filter algorithm presented above, it depends on a map $(m)$, to process the measurement data, see listing 2.3. In the following, the algorithm is presented using an example.

It takes as input a set of $N$ particles $x_{t-1}$, representing the last belief on the robots' pose, the most recent control data $u_t$ and measurements $z_t$ and outputs a new set of particles $x_t$, representing the new belief, see listing. Initially the particles are randomly distributed, as no information on the robots' pose is available, as can be seen in Figure 2.3 for a one-dimensional example. The robot can move left and right and is equipped with a sensor, that can sense doors. For each particle $x_{t-1}^{[n]}$ in the input set, again, first the *prediction step* is performed, by sampling a new particle $x_t^{[n]}$ according to the given control data $u_t$. The different approaches to do this sampling will be discussed below. The particle distribution after this step still is random. In the *prediction* step, for the updated particle $x_t^{[n]}$ a weight value $w_t^{[n]}$ is calculated, given the latest measurement data $z_t$ and the map $m$. An overview of the different measurement model algorithms is provided below. In Figure 2.3 (b) this step is visualized for the measurement of a door. Particles that represent states with a high likelihood to sense a door get a high weight. Finally, the *resampling step* is done, and the updated set of particles is returned. After the resampling, the particle weights are equal again, but the particles now are distributed according to the weight.

Figure 2.3 (c) visualizes the distribution after the next prediction step after the robot has moved to the right. In 2.3 (d) the next measurement update can be seen. Again, particles with high probability to sense a door get high weights. After the next resampling step in Figure 2.3 (e) the particles form regions, that represent the robot's pose estimation.

```
Algorithm MCL($X_{t-1}$, $u_t$, $z_t$, $m$):
  $\bar{X}_t$ = $X_t$ = $\emptyset$
  for $n = 1$ to $N$ do
    $x_t^{[n]} =$ sample_motion_model($u_t, x_{t-1}^{[n]}$)
    $w_t^{[n]} =$ measurement_model($z_t, x_t^{[n]}, m$)
    $\bar{X}_t = \bar{X}_t + \langle x_t^{[n]}, w_t^{[n]} \rangle$
  endfor
  for $n = 1$ to $N$ do
    draw $i$ with probability $\propto w_t^{[i]}$
    add $x_t^{[n]}$ to $X_t$
  endfor
  return $X_t$
```

Listing 2.3: Monte Carlo Localization algorithm [TBF05]

## 2.6. Robot Motion Model

To implement the Monte Carlo localization, a motion model is necessary, which models the state transition in Bayesian filtering. It describes the posterior distribution, that the motion command $u_t$ changes the robot's state from $x_{t-1}$ to $x_t$. According to [TBF05], two typical approaches can be distinguished: those based on velocity and odometry-based models. The velocity motion model can be applied to robots, that are not equipped with rotary encoders It models the motion of the robot based on its velocity, which is obtained from control data. The odometry based approach models the robot's motion based on odometry measurements, which usually is more precise, according to the authors. So as the Wolfgang platform is equipped with rotary encoders in the actuators, an odometry based motion model is used in this work.

## 2.7. Measurement Model

With measurement models, the noise in sensor measurements is modeled, as they underlie uncertainty. There are three prominent ways to approach the measurement model, according to [TBF05], that work on raw data input: the beam model, map matching, and likelihood fields models. The authors focus on range-sensor but state that the principles and equations can be applied to different kinds of sensors. In the beam model, for each particle, the ranges to the next obstacle on the map are calculated by raycasting. The measured laser scan beams are then compared to the calculated one and based on that the particle's probability is calculated. So with the beam model, only the error in the distance can be modeled. When applying this model to data provided by a camera, it could not model the angular error that could occur.

The idea of the map matching model is to build an occupancy map from all measurements and calculate the particle weights on the degree to which the calculated map fits the actual map. This method is independent of the type of sensor, as it only depends on the calculated map from the sensor measurement, not on the measurements directly. It could be easily applied to work on camera data.

The likelihood fields model is more abstract. For each particle, the measured laser range data is projected onto the map. Then, for each laser scan, the distances to the nearest objects are calculated. So in contrast to the beam model, where the probability of the particles is calculated only the ray directly, here the possible surrounding of the ray is included in the probability calculation. Still, it can be adapted to visual observation. In comparison to the map matching approach, it provides more flexibility, as it works on single measurements and not a map generated from them. In this work, the measurement model was implemented according to the likelihood field model, because it provides flexibility while modeling different kinds of errors. It also can work on precomputed probability values and thus saves computational resources.

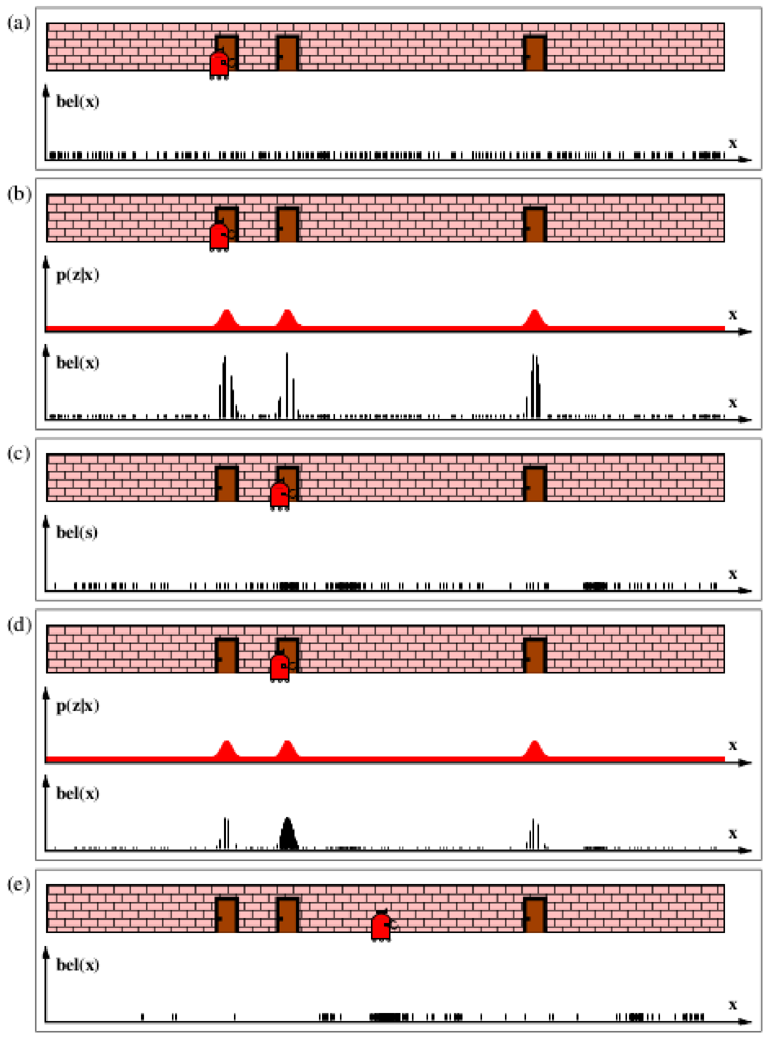


Figure 2.3.: MCL procedure [TBF05]. (a) Initially, the robot's position is unknown, and the particles are distributed uniformly. (b) A measurement is done, and the particle weights are updated according to the measurement. (c) After resampling, the particle's weights are equal, and they are distributed according to their former weights. The robot moves, and the movement is applied to the particle's position (prediction step). (d) Again, a measurement is done, and the particle's weights are updated. (e) After another resampling, many particles are located around the robot's true position.

# 3. Related Work

In the following work in the field of Monte Carlo Localization is presented. Section 3.1 gives an overview of MCL and some relevant variants and applications of it. In Section 3.2, adaptions of MCL in RoboCup Soccer are presented. Two prominent ROS package that implement MCL are presented in Section 3.3.

## 3.1. Monte Carlo Localization

Delleart et al introduced the term Monte Carlo Localization (MCL) in 1999 [DFBT99]. For a detailed description, see Section 2.3. They run two experiments on wheeled robots equipped with laser range finders, one in an office scenario and one in a museum. The algorithm was able to globally estimate the robot's pose correctly as well as track the robot's precisely.

One year later Lenser at all added the resampling step[LV00]. In that step, if the mean probability over all particles is low after the measurement step, some samples are replaced with samples drawn from the probability density of all particles. This way, fewer particles are needed, and the algorithm is more robust against errors, especially for small sample sizes.

In 2002 [Fox02] introduced KLD-Sampling. It adapts the sample size dynamically to increase efficiency, based on the approximation error. In this approach, the approximation error is measured by the Kullback-Leibler distance, giving the KLD-Sampling its name.

The Monte Carlo Localization was successfully applied to a humanoid robot platform by Thompson et al. [TKN06] in 2006. A 3-dimensional Monte Carlo Localization was implemented, working on a 2.5-dimensional map. The robot was 155cm high with 30 degrees of freedom and equipped with a laser range sensor inside its head. The presented a motion model, which includes temporal uncertainty in odometry. An experiment on pose tracking that was conducted on the robot revealed a maximum error of 40 cm, while mostly being close to the ground truth, as measured by a motion capture system.

## 3.2. MCL in RoboCup Soccer

In 2004 Röfer et al. implemented a localization approach in the RoboCup Sony Four-Legged Robot League (SFRL) [RJ04].It was based on MCL and used the lines and the white wall around the playing field as localization information.
In their experiments on pose tracking error, they achieved a mean error of 10.5 cm. They did not report the angular error.
This work was later further developed to also work on color-coded beacons next to the playing field provided for localization and the color-coded goals [RLT06], but no experiments on precision were done. .

Muzio et al. [MAMP16] implemented a version of Monte Carlo Localization based on the goalposts, the corner flags and the lines in the RoboCup 3D Soccer Simulation League. The showed that the localization estimate is more precise when using all three types of localization information instead of only goalposts and corner flags in a position tracking scenario with the robot being kidnapped.

In the RoboCup Standard Platform League, many teams state in their team reports having some localization method, like rUNSWift [BHJ$^+$18], Camellia Dragons [TTK$^+$16] and [Nao18].
The team b-Human provides a detailed description of their localization approach. The robot states are implemented with Unscented Kalman Filters (UKF); each pose hypothesis is represented by one UKF. These UKF are handled by a particle filter. The field lines, the line crossings, the center circle, and the penalty marks are used as localization source.

In the RoboCup Humanoid Kid Size League, the team Rhoban Football Club is one of the few teams reporting to use localization techniques [AGH$^+$19]. They use a particle-filter based approach, with 300 particles. It works on goal posts, and the corners of the playing field, detected by computer vision approaches on the camera image. Moreover, it integrates information from the referee about the state of the game. Furthermore, they improved the quality of odometry by equipping their robots with pressure sensors below the feet.

The team Barelang-FC also uses a particle filter-based approach, which only relies on goalposts as localization information [SJS$^+$19]. It works on only 100 particles.

## 3.3. MCL with ROS

There are several ROS packages available on localization. Two prominent ones are presented here.

### 3.3.1. *amcl*

The ROS package *amcl* [13] provides an implementation of KLD-Sampling, see 3.1, which also known as adaptive Monte Carlo Localization. It is developed to localize a wheeled robot, that is equipped with a laser range finder on a 2D plane.

It offers two measurement models: the `beam_range_finder_model` or
`the likelihood_fields model`, both of them working on laser range finder data, as presented in 2.7.

Two motion models are available: the `sample_motion_model_odometry` algorithm and an extension of it.

The package offers the possibility to initialize the algorithm with a Gaussian as initial belief. The mean and standard distribution one each dimension can be specified. Furthermore, it offers a service for global localization, where all particles are spread over the whole map.

The package is only very limited usable with robots that are equipped with a 2D RGB camera. When converting line point detections into a message of type *sensor_ msgs/LaserScan* [14], much information can get lost, because only the measurements closest to the robot are covered.

Moreover, it is not possible to fuse several information sources into the filter.

Also, the possibilities for initialization are quite limited, as only one pose can be given. This pose would need to cover both sides of one field half, and the given angular value would need to cover two opposite orientations.

The other possibility, global localization, which is available via a service call of *global_localization*, spreads the particles uniformly distributed over the whole map. This is also not very helpful, as due to the symmetry of the field, this will result in two equally good pose estimations, one on each half of the field.

### 3.3.2. *humanoid localization*

The ROS package *humanoid localization* provides an implementation of MCL in 3D space.

As amcl it takes as input a *sensor_msgs/LaserScan* message, but it can also work on point cloud data and can fuse orientation data provided by an IMU (inertial measurement unit) into the filter.

It also offers the possibility to initialize the robots pose with Gaussian distribution or do global localization. Both options are not well suitable in the context of humanoid soccer, as discussed above.

Again, only one kind of information is possible.

# 4. Approach

In this work a version of Monte Carlo Localization is developed, that works on a small humanoid robot and can process several input data sources at once. It is divided into two separate parts: the localization node itself, working on vision data projected into Cartesian space, and the localization handler, which (re-)initializes the filter with information of the robot's state and the state of the game. The integration into the Hamburg Bit-Bots' software stack with the information flow is depicted in Figure 4.1.

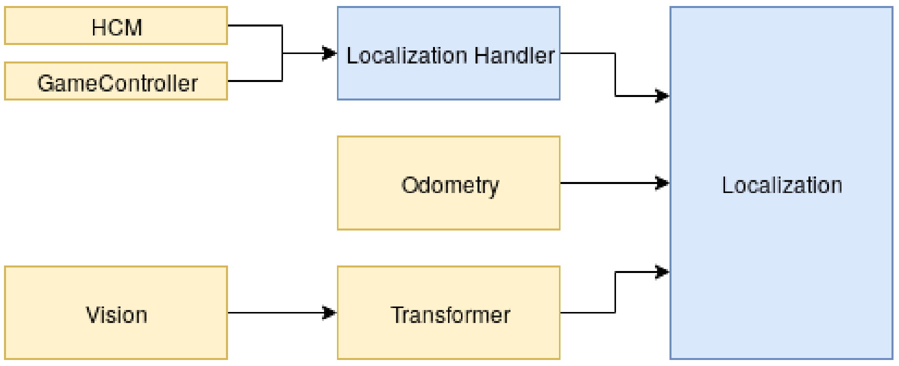


Figure 4.1.: Information flow diagram. The modules from the Hamburg Bit-Bots software stack are marked in yellow. The blue ones were implemented in this work. The Localization Handler initializes and reinitializes the Monte Carlo Localization according to information provided by GameController and Hardware Control Manager. Odometry data is used for the prediction step of MCL. The measurements for the update step are provided by the Bit-Bots vision pipeline and are transformed into Cartesian space.

In Section 4.1 the implementation of the MCL algorithm is presented, including the state definition, the motion model, possible state distributions for initialization, the adapted measurement model, the service for initialization and the way the pose estimate is generated from the particle distribution. The localization handler is presented in Section 4.2 and in Section 4.3 the visualization setup is described.

## 4.1. Localization Node

For the implementation of the Monte Carlo Localization (MCL), the particle filter library [15] was used, which is easily integrable into ROS. It is a lightweight implementation that is highly flexible and thus applicable in different contexts. The particle filter library offers the basic structure in which a custom state model, motion model, state distributions and measurement and map model and can be implemented. The developed version of MCL can integrate different kinds of measurements. A measurement model for laser range finders was adapted to work on measurements from a 2D image. The MCL can be (re-)initialized with different state distributions, based on prior knowledge.

The localization is implemented according to the MCL algorithm [TBF05], see Section 2.5.

In the filter's main loop, first the newest measurements are set, if the corresponding information type should be used in that step and the data is newer than the one used in the last filter step

Then the latest robots latest movement on the field, from the last filter step to now, is looked up in the tf-tree.

After that the prediction step is performed, as described in Section 2.6, by applying *drift* and *diffusion*.

In the measurement step afterward, the observation model, as described in Section 2.7, is applied.

Then the resampling step is done with Importance Resampling, as proposed in [DGA]. Unlike suggested in [LV00], this is not only done if the mean weight over all particles is low but after a fixed number of filter steps, that can be adjusted. This was done with regard to the evaluation done is this work, as is keeps the results among different conditions easier comparable.It is not done after every filter step, because with each resampling hypothesis on the robot's pose are rejected. On a robot with a moving head and a limited field of view, the particle weight always is calculated only on a very limited part of the possible observations. Including more consecutive measurements into the particle weight before resampling makes it more robust against rejecting good particles.

Finally, the pose estimate, as the mean over the best particles, is published as `geometry_msgs/PoseWithCovarianceStamped` message [16] and the markers for visualization of are generated and published.

The localization node is implemented using *dynamic reconfigure* [17]. It allows the user to adjust the parameters of the localization, while it is running.

### 4.1.1. Robot Pose State

The states representing the robot's pose in the world are represented by the vector $\begin{pmatrix} x \\ y \\ \theta \end{pmatrix}$, where $x$ and $y$ refer to the robot's position on the ground plane and $\theta$ to its rotation around its yaw axis, which is orthogonal to that plane. The robots reference frame is the base_footprint frame.

### 4.1.2. Robot Motion Model

Two kinds of motion can be applied to the particles, called *drift* and *diffuse*.

### Drift

The *drift* models the robot's motion relative to the ground plane. It is implemented according to the algorithm `sample_motion_model_odometry` by [TBF05]. It updates a given state with the current odometry data.

The odometry data contains odometry measurements from the last time step and the current one: $u_t = \begin{pmatrix} \bar{x}_{t-1} \\ \bar{x}_t \end{pmatrix}$, retrieved from the tf tree. This way, the step size, over which the robot motion should be retrieved can be easily adjusted to the particle filter frequency.

From these measurements three parameters are retrieved, that model the robots' latest motion: a rotation $\delta_{rot1}$, a translation $\delta_{trans}$ and a second rotation $\delta_{rot2}$, see Figure 4.2.

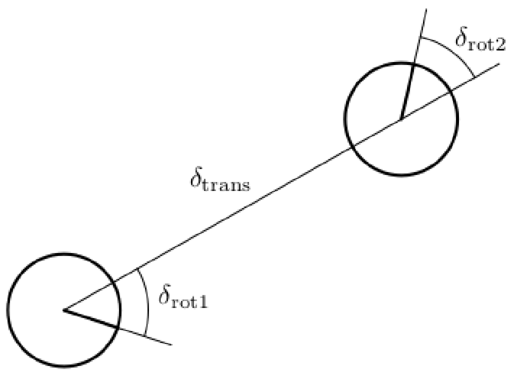


Figure 4.2.: Visualization for the odometry motion model. [TBF05]. The robot's motion can be model by a rotation, followed by a translation and a final rotation.

The first rotation $\delta_{rot1}$ models the direction of motion implicitly given by the translational movement in $x$ and $y$ direction. The translation $\delta_{trans}$ can then be calculated as the distance between the old and the new position. The second rotation $\delta_{rot2}$ models the robots' final orientation, taking into account the old orientation and the first rotation. The calculation is done in lines 3-5 in Listing 4.1.

```
 1  Algorithm sample_motion_model_odometry($u_t$, $x_{t-1}$):
 2
 3      $\delta_{rot1}$ = $atan2(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
 4      $\delta_{trans}$ = $\sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
 5      $\delta_{rot2}$ = $\bar{\theta}' - \bar{\theta} - \delta_{rot1}$
 6
 7      $\hat{\delta}_{rot1} = \delta_{rot1} -$ sample($\alpha_1\delta_{rot1} + \alpha_2\delta_{trans}$)
 8      $\hat{\delta}_{trans} = \delta_{trans} -$ sample($\alpha_3\delta_{trans} + \alpha_4(\delta_{rot1} + \delta_{rot2})$)
 9      $\hat{\delta}_{rot2} = \delta_{rot2} -$ sample($\alpha_1\delta_{rot1} + \alpha_2\delta_{trans}$)
10
11      $x' = x + \hat{\delta}_{trans} cos(\theta + \hat{\delta}_{rot1})$
12      $y' = y + \hat{\delta}_{trans} sin(\theta + \hat{\delta}_{rot1})$
13      $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
14
15      return $x_t = (x', y', \theta')^T$
```

Listing 4.1: Algorithm to sample from the motion model based on odometry [TBF05]. The function `sample(`$s$`)` samples from a Gaussian distribution with zero mean and standard deviation $s$.

The algorithm takes as input the odometry information $u_t$ and an initial pose $x_{t-1}$ and outputs the updated pose $x_t$.

First, the three relative motion parameters discussed above ($\delta_{rot1}$, $\delta_{trans}$, $\delta_{rot2}$) are retrieved from the odometry data (line 3-5).

Then, in line 7-9, to each of these motion parameters, Gaussian noise is applied, depending on the related parameters to account for errors in the robots' odometry. The function `sample(`$s$`)` samples from a Gaussian distribution with zero mean and standard deviation $s$. In this implementation, all of the parameters $\alpha_1$ to $\alpha_4$ were set to $0.001$, as the correct errors were unknown. In Figure 4.3 possible samples for three different sets of the error parameter $\alpha_1 - \alpha_4$ are visualized. Figure 4.3 (a) has moderate parameter values. In Figure 4.3 (b) the results of a high value of $\alpha_3$ can be seen and in (c) $\alpha_3$ is small but $\alpha_1$ and $\alpha_2$ are large. The final orientation of the samples is not visualized, but it varies in the same amount as $\delta_{rot1}$ does, as it also depends on $\alpha_1$ and $\alpha_2$.

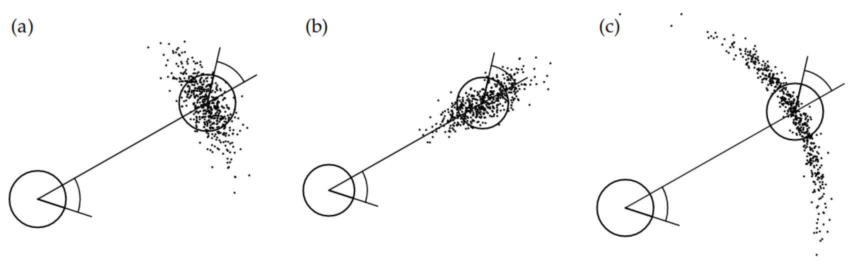


Figure 4.3.: Possible samples from the odometry motion model [TBF05] for three different sets of the error parameters $\alpha_1 - \alpha_4$.

In lines 11-13, a new pose is calculated by applying the updated parameters to the given old pose. Finally, the updated pose is returned.

#### Diffuse

*Diffusion* adds random Gaussian noise to the particles. The amount can be dynamically configured.

The *diffusion* of particles has two purposes. Firstly it was added to compensate for the unknown motion error parameters in the *drift* model. Adding noise afterward scatters the predicted particles and thus might help to reduce the particles to concentrate on faulty states. Secondly, diffusing the particles in each filter step could improve the efficiency of the particle filter, because fewer particles might be needed. The intention behind that is that a limited number of particles can cover only a part of the robot's possible states. If the robot does not move, it is possible that the true pose is not sufficiently covered with particles. So by adding noise to the particles in that case, more states can be covered. The Gaussian noise added in this step is applied to the particle directly, so the distribution of samples, in this case, is centered around each dimension of the particle.

### 4.1.3. State Distributions for Initialization with Prior Knowledge

If some information on the robot's pose it available, it can be used to initialize the particle filter. This is done by distributing the states according to the prior knowledge. For different predefined situations in the game, the localization node can be initialized with different state distributions. They are based on information provided by the laws of the game or help to reinitialize the localization efficiently after the robot fell.

#### Pose known

One of the simplest cases of localization is to keep track of a (roughly) known pose. In a RoboCup Soccer game, this can happen if robots of the team having kickoff were not able to position themselves legally on the field, or positioned themselves strategically disadvantageously. The robot handler then is allowed to position for instance the robot playing the goalkeeper "within the team's own goal area touching the goal line" [Rob19].

So when the game enters the playing phase, the goalkeeper's localization can be initialized inside its own goal, as is needs to be there either by self-positioning or being positioned there manually. The initial state distribution for a goalkeeping robot of a team having kickoff can be modeled with a Gaussian distribution, centered between the own goalposts and oriented towards the field, as can be seen in Figure 4.4.

#### Position known, orientation unknown

A slightly more complicated case is to localize the robot with a more or less known position but unknown orientation. This scenario can occur if a robot has fallen. After getting up its position on the field is approximately the same, but as the exact direction of falling is hard

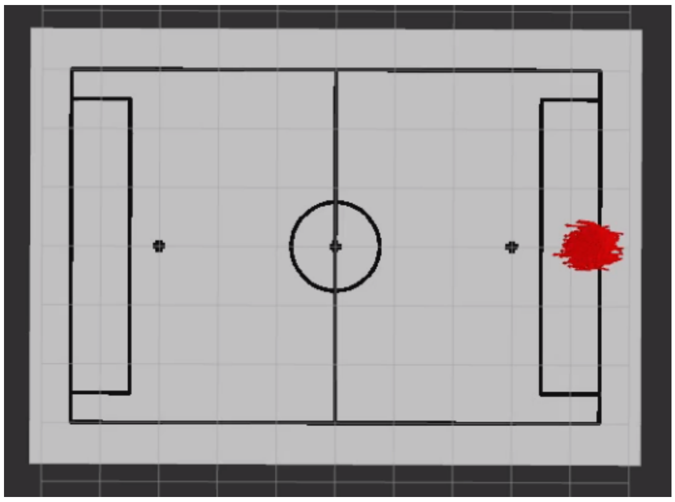


Figure 4.4.: State distribution for a known pose. The particles are distributed at initialization with Gaussian distribution centered around the pose $\begin{pmatrix} -4.5 \\ 0 \\ 0 \end{pmatrix}$ with a standard deviation of 0.1 on each dimension.

to determine, especially if the robot falls onto another robot and rolls over, its orientation is unknown. So the localization node needs to be reinitialized with a roughly known position but unknown orientation. The state distribution for initializing the localization after a fall could be a Gaussian deviation on $x$ and $y$ as for the known pose, but with uniformly distributed $\theta$.

### Multiple specified regions

When a robot re-enters the field, for instance after a removal penalty, it is only allowed to do so "from the team's own half of the field close to the penalty mark facing the opposite touch line" [Rob19]. The referee defines the side of the half and the exact position. In the laws, some information on possible positions and the orientation are provided, but the exact pose needs to be determined. So the initial state distribution for this scenario is two Gaussian distributions, one at each touchline, with the position centered at the height of the penalty mark, and the orientation facing towards the field, as can be seen in Figure 4.5.

### Field half

If no information on the pose is available, for instance, if no information is received from the GameController because of network errors, a more general state distribution is required at initialization. Here, particles are spread with uniform distributed position and orientation over one half of the field. The side can be manually configured.

A complete global localization, with an initial state distribution over the whole field, is only useful if at least one input source provides non-symmetric information. Otherwise, the Monte

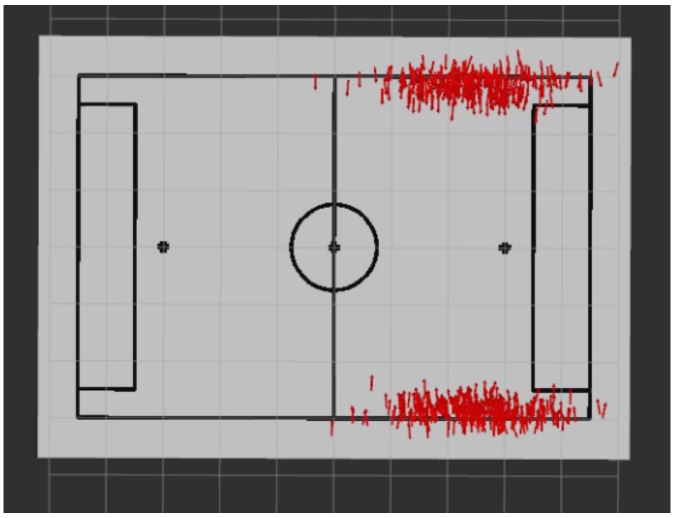


Figure 4.5.: State distribution for multiple specified regions. The particles are distributed at initialization with Gaussian distribution centered around the poses $\begin{pmatrix} 2.5 \\ 3 \\ -1.57 \end{pmatrix}$ and $\begin{pmatrix} 2.5 \\ -3 \\ 1.57 \end{pmatrix}$ with standard deviation of 0.8 on $x$ and 0.1 $y$ and $\theta$.

Carlo Localization will result in two equally good pose estimates on the corresponding poses on the field.

### 4.1.4. Measurement Model and Map Model

The measurement model was implemented following the Likelihood Fields approach by [TBF05], which provides a rating for each particle, depending on the overall likelihood to actually sense the incoming sensor data.

It was adapted to model data from a 2D RGB camera instead of a range finder and to deal with several kinds of inputs simultaneously. The original algorithm is provided in Listing 4.2.

```
Algorithm likelihood_field_range_finder_model($z_t, x_t, m$):
  $q = 1$
  for all $k$ do
    if $z_t^k \neq z_{max}$
      $x_{z_t^k} = x + x_{k,sens} cos\theta - y_{k,sens} sin\theta + z_t^k cos(\theta + \theta_{k,sens})$
      $y_{z_t^k} = y + y_{k,sens} cos\theta - x_{k,sens} sin\theta + z_t^k sin(\theta + \theta_{k,sens})$
      $dist = \min_{x',y'}\{\sqrt{(x_{z_t^k} - x')^2 + (y_{z_t^k} - y')^2} | \langle x', y' \rangle$ occupied in $m\}$
      $q = q \times (z_{hit} \times prob(dist, \sigma_{hit}) + \frac{z_{random}}{z_{max}})$
  return $q$
```

Listing 4.2: Likelihood field range finder model [TBF05]

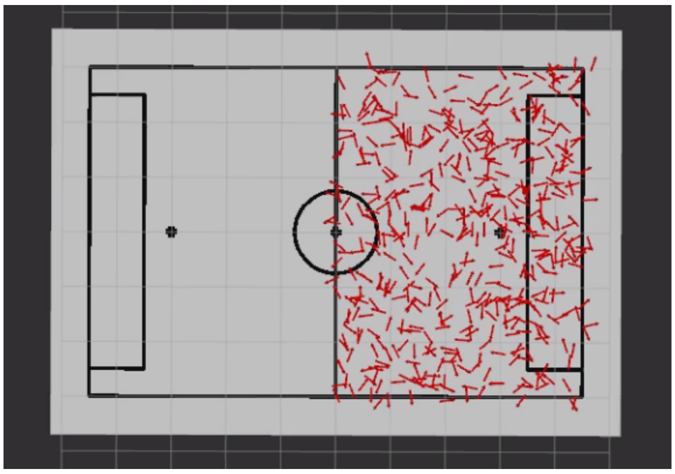


Figure 4.6.: State Distribution for field half. The particles are distributed at initialization with uniform distribution inside the dimension on the field half and uniform distribution on $\theta$ between 3.14 and -3.14.

It takes as input the current laser range reading $z_t = \{z_t^1, ..., z_t^K\}$, a pose $x_t$ and a map $m$. For all values of $z_t$ first, it is checked if the value is a maximal sensor range reading, in which case the data is ignored. Otherwise, the endpoint coordinates of the sensor reading are transformed onto the map, given the robots pose and the heading direction of the laser range finder $\theta_{k,sens}$ (line 5 and 6). Then the minimum distance to the nearest obstacle is calculated, and the probability for the sensor beam is calculated. Finally, the sum of all sensor measurement probabilities is returned.

Two main changes were made, to adapt the algorithm to work on information provided by a 2D RGB camera.

Firstly the measurements from the 2D camera, transformed into the robots base_footprint frame were handled like laser beam endpoints. The distance calculation from the sensor to the detection was skipped, as it only provides information on the three-dimensional space where the measurement was taken. This cannot be applied to the two-dimensional map the localization works on, as the calculated ray does not provide any information on the two dimensional (possible) detection on the field.

Secondly, only the measurement noise $z_{hit}$ and unexplained random measurements $z_{rand}$ were included in the measurement model. The measurement noise models the error of the detection, either caused by inexact detection in the image or due to errors when transforming it into the robot's base_footprint frame, see 2.1.2. Random measurements are false positives detections in the vision algorithms. In the Hamburg Bit-Bots vision pipeline, the are more likely to occur on the field rather than outside the field, as image processing strategies like the field boundary were implemented to search only the image parts that actually are on the field [FBG+19]. Failures like max range readings for range finder do not occur in data provided by object detection algorithms on 2D RGB images.

The pseudo-code of the adapted `likelihood_field_model` can be found in Listing 4.3.

```
1  Algorithm likelihood_field_model_humanoid($Z_t, x_t, m$):
2    $r = []$
3    $w = 0$
4    for all $z_t$ in $Z_t$ do
5      $q = []$
6      for all $k$ do
7        $d_m =$ measuerment_on_map($x_t, z_t^k$)
8        $q$.push_back(get_rating(d_m, m))
9      $q = ((\sum_{i=0}^{q.size()} q[i])/q.size()) * f_z$
10     r.push_back(q)
11   if r.size() =! 0
12     $w = (\sum_{i=0}^{r.size()} r[i])/r.size())$
13   return $w$
```

Listing 4.3: Likelihood field model for a humanoid robot

It takes as input different kinds of localization information $Z_t$. Then for each kind of measurement $z_t$, the measurement is done equivalently to the original algorithm.

Transforming the detection into the map frame could be simplified, because the measurements are already transformed into the base_footprint frame and thus are independent of the camera's orientation relative to the robot. To make the calculation of $x_{z_t^k}$ and $y_{z_t^k}$ better understandable, an approach using polar coordinates was chosen. The measurement $z_t^k$ first was transformed into the polar coordinate representation. This way, the particle's orientation $\theta$ can be added to the angular component of the measurement. The resulting rotated measurement $z_t'^k$ is oriented in a way, that it only needs to be shifted to the particle's position $x, y$. This can easily be done by converting $z_t'^k$ back to Cartesian coordinates and adding them. This is done for each single detection $z_t^k$ in the function `measurement_on_map()` in line 7.

The detection probabilities for each detection type were pre-computed in lookup tables with 1 cm resolution, which is precise enough in the context of robot soccer. Due to an error in the early stages of this work, the probabilities are not represented in the interval $[0, 1]$, but in the range from -10 to 100. Although this lacks a correct mathematical explanation, the results were sufficient. This shows the robustness of the used particle filter library. The probability ratings on these maps range from 0 to 100, with 0 being very unlikely for the particle actually to sense that observation to 100, very likely. The measurement noise $z_{hit}$ is modeled with Gaussians around the true positions of possible detections. An observation outside the map gets a rating of -10, which models the unexpected random noise $z_{rand}$.

The function `get_rating(d, m)` provides the rating of a single measurement on the specified map.

Over all single measurement of one information source, the arithmetic mean is calculated, see line 9, to keep the provided rating per information source independent from the number of measurements used for it. The resulting information source ratings can be weighted by a factor, to adapt the degree in which they are used for the final particle weight. In this work, all information source factors were one.

In line 12, these ratings provided by the different kinds of information again were combined

to a final particle weight $w$ by calculating the mean over all information source ratings.
In the following, the different information sources for the observation model are discussed.

**Lines** One of the primary information sources a soccer field provides for localization is the line-markings. The are detected by the vision pipeline 2.1.2 and published as *humanoid_league_msgs/LineInformationRelative*, as proposed by [Bes17]. The message definition can be found in B.5 and B.6. The data is then applied to the measurement

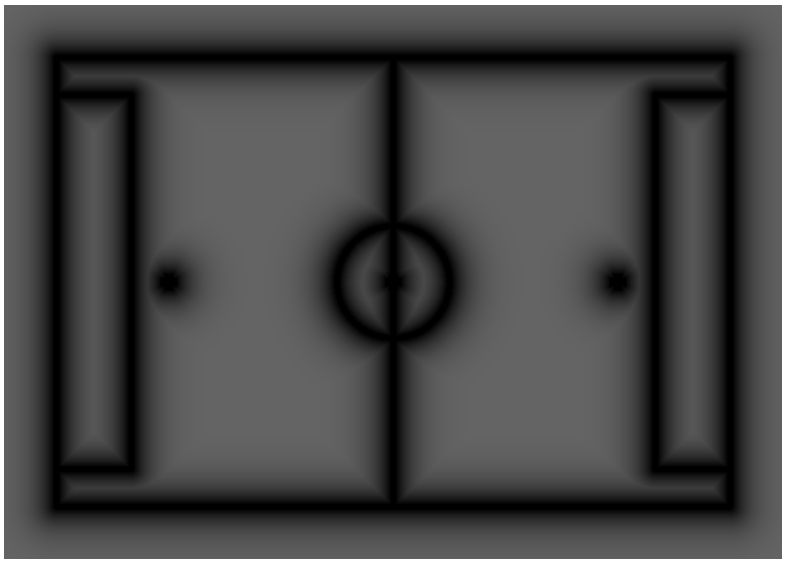


Figure 4.7.: Lookup table for lines. The lighter the pixel the line point is projected on, the lower its rating. Line points outside the map get the lowest rating. The overall information source rating is calculated by adding up all line ratings and dividing by the number of line points detected in that measurement.

model and checked against the probability lookup table shown in Figure 4.7.

**Goal posts** Another essential information for robot self-localization on a soccer field are the goals. Complete goals are hard to detect, as often only parts of the goal are in the field of view. However, even the goalposts without the crossbar provide valuable information for localization. They are published as the *humanoid_league_msgs/GoalRelative* message, see B.4 If only one goalpost is detected, it is a convention at the Hamburg Bit-Bots to fill both goal post fields in the message with that one goal post.

So before applying the measurement to the localization, it is checked if both goalposts are identical. In that case, the goal post is fed into the particle filter only once.

The detected goalpost measurements are rated according to the lookup table in figure A.1a.

**Field boundary** The *fieldBoundaryRelative* message provides information, on where the outside border of the field was detected. Its definition can be found in B.2.

The field boundary points need to be filtered before passing into the Observation model to provide more robust information. The filtering is necessary in two ways. First of all, the algorithm provides wrong data when transformed into Cartesian space in case the complete field of view is covered with green field. As in this case always the uppermost row in the image is classified as field boundary, because it is the first detection of green, the field boundary transformed to Cartesian space will be on the field and moving with the head movement.

The other case is that it is not possible in this field boundary detection for the furthest left and right point to distinguish whether it is a correct detection of the first green point in that column or caused by an obstacle. As mentioned above, the convex hull helps to get rid of most of these dents. However, the outermost points always belong to the convex hull and thus need to be ignored for localization. So for each *FieldBoundaryRelative* message, the corresponding *FieldBoundaryInImage* message needs to be checked for the $y$ coordinates of the points. The definition can be found in B.1. New points are only interpolated starting with the second point to the second last point and only if two corner points have a $y$ value greater then $0$. The probability lookup table for field boundary data can be seen in Figure A.1b

**Field marking features** The field markings can be seen as the line points mentioned above, but they can also be seen on a higher abstraction level. They are defined to form patterns like corners, crossings, or crosses. Depending on the style of the field marking, also points can be such a field marking feature, as they can be used for the penalty marks and the center point. These features consist of at least two lines that meet in a certain way.

In this work, three types of features are distinguished: L-shaped corners, T-shaped crossings, and crosses. Corners appear at the outside of the field marking and at the goal areas, see the blue markings in Figure 4.8. T-crossings are located where the goal areas meet the goal lines, and the middle line meets the touchlines, see green markings. The penalty marks and the center circle can be marked either as points or as crosses. In this work, they are assumed to be crosses. Furthermore, the crossings of the center circle with the middle line are classified as crosses, although they are not perfectly cross-shaped. They are marked in red.

In comparison with the line points, the field marking features provide less information, because the lines between the features are ignored. However, the provided information is more precise. While there is a variety of possibilities to map a piece of line onto the field markings, a detected crossing can only be mapped to one of 8, a T-crossing to one of 6 and a cross to one of 5 different positions. As the features are spread over the whole playing field, they are often in the robot's field of view so that no active head movement is necessary for localization. Depending on the detection method for these field features, different levels of information can be gained from them. If the exact position and orientation of the field features are available, it could be handled similar to the procedure for lines described above, with the advantage that a detected feature could not be fitted onto a line erroneously, but only onto another feature of its

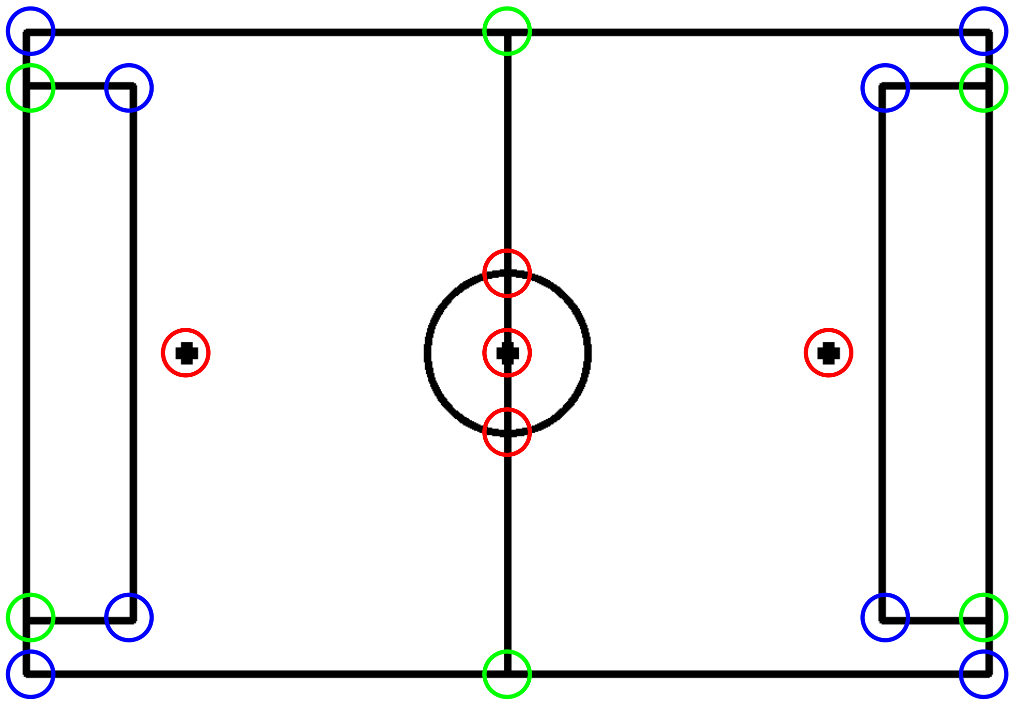


Figure 4.8.: Visualization of field marking features. Corners are marked in blue, t-crossings in green and crosses in red.

kind. However, the features can also be used for localization if no relative orientation is provided by the detection method. The detected field marking feature measurements are rated according to the lookup tables in Figures A.1c, A.1d and A.1e.

### 4.1.5. Initialization Service

The localization node can be re-initialized by a service call of *reset_filter*. It resets the particles according to the distribution specified in the service call. For initialization over one field half and the start distribution with multiple specified regions, on the according mode needs to be set. If the node should be initialized with known position, the position needs to be specified in the request. Due the limited scope of this work, the initialization with a known pose was not included in the service call.

```
int64 init_mode
int64 x
int64 y
---
bool success
```

Listing 4.4: Definition of the *reset_filter* service. A certain initialization mode can be requested and, if necessary, a position can be specified.

### 4.1.6. Calculation of Estimated Pose

The estimated pose is calculated over the mean of the particles with the highest weights. How many particles should be considered can be set by *dynamic reconfigure*.

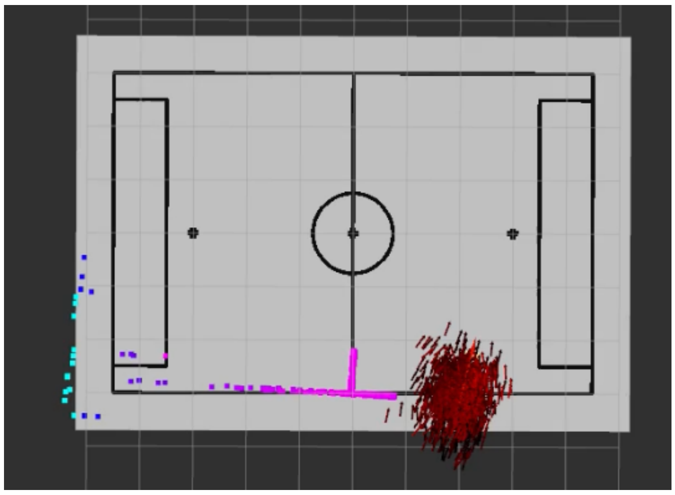
The pose estimate is published in two ways. Firstly a transform from the *map* frame to the *odom* frame is published. Additionally it is published as *PoseWithCovarianceStamped* [16] message. This includes the covariance matrix, which is calculated over all particles, not only the ones used to calculate the estimated pose itself. This way information on the quality of the pose estimation is available.

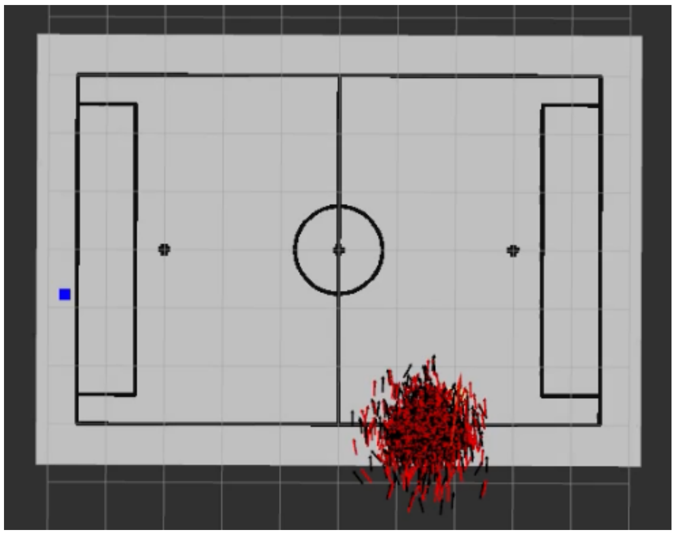
## 4.2. Localization Handler Node

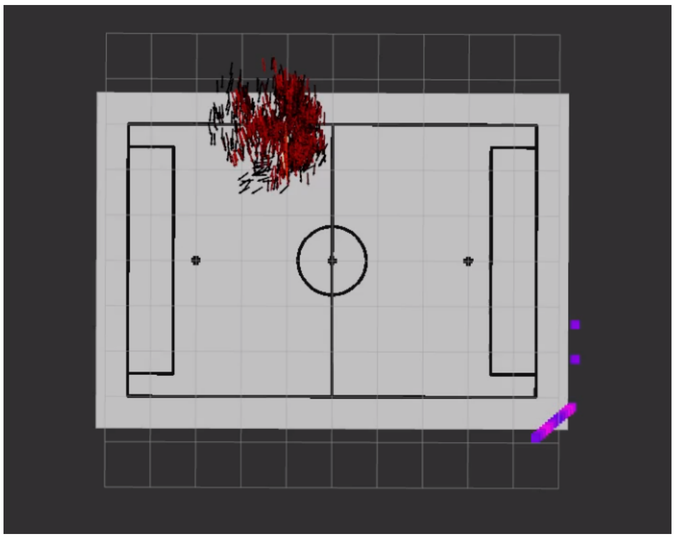
To make the localization more robust, a node that initializes the localization with knowledge from the game state on the one hand and the robot state on the other was implemented. According to the information received from the *GameController* on the state of the game and the robot's role in the game, the best initial state distribution for particles is chosen. The information is received as *humanoid_league/GameState* message, see Listing B.3. Moreover, it tracks the last pose estimate, and in case the robot falls, it reinitializes the particle filter with the known position but unknown pose. The information is published by the HCM as *humanoid_league/RobotControlState*. Its definition can be found in B.7.

## 4.3. Visualization

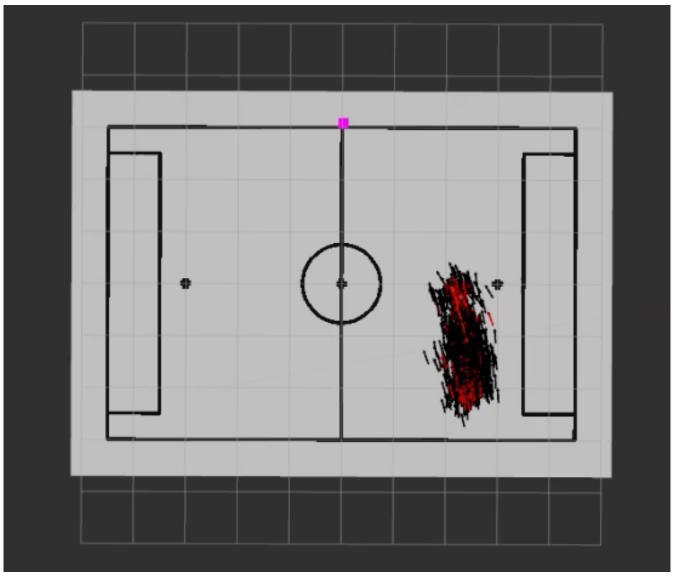
For debugging and parameter adjustment, a visualization setup was built in *rviz* [8]. The particles are published as *visualization_msgs/MarkerArray* [19] of type *arrow*. Each particle weight is indicated by the color of the arrows, with red close to 1 and black close to zero. The incoming measurements were visualized by a marker message of type point. Each line point and each field boundary point was visualized as one point, as well as each goalpost detection and each field marking detection. After one filter step, the measurements used in that step are transformed into the frame of the resulting pose estimate. Then a rating process equivalently to the measurement step was performed, and the resulting measurement rating for the pose estimate was published with colors ranging from cyan to magenta, with magenta for high ratings. Measurements that were not on the map are published in blue.


(a) Visualization of line point data.


(b) Visualization of a detected goalpost with a low rating.



(c) Visualization of field boundary point data.



(d) Visualization of of a detected field marking feature, a t-crossing, with a high rating.

Figure 4.9.: Vizualization setup in rviz. The red to black arrows represent the particles. The more reddish the particle, the higher is its weight. The cyan and violet to pink squares visualize different measurements and their ratings for the pose estimation. The cyan ones are projected outside the map and thus have the smallest weight. The more reddish, the higher the weight.

# 5. Evaluation

To evaluate which kinds of information sources and which combinations of them provide the most accurate pose estimation, two experiments were conducted on localization and one on pose tracking. The information sources used in this evaluation are line point data, field boundary data, goal post data and the field marking features (corners, t-crossings, and crosses).The first experiment evaluates the precision of the robot's estimated pose on one field half. The robot is stationary in this experiment. In the second experiment, the precision to track a robot's known pose while moving is evaluated. The experiments were conducted on simulated data because the robots true pose is provided by the simulation software. The simulation was run using *Gazebo* [20], which is well integrated into ROS.

The line point detections and field boundary detections were provided by the Bit-Bots vision pipeline. The goalpost detections and detections of field features (corners, t-crossings, crosses) were simulated. This was necessary because the Bit-Bots goalpost detection was developed to work on real-world data. So as the net was trained on image sets recorded on competition setups and in the Bit-Bots test laboratory, it does not have a high performance on images recorded in simulation. The field feature detections needed to be simulated, because due to the limited scope of this work is was not possible to implement feature detection. Each feature and goal post was added to the tf-tree on the correct position in the simulated environment. Only those goalposts and field marking features that were currently in the robots field of view were published, containing the position relative to the robot's base_footprint frame. The goals posts were published using the usual *humanoid_league_msgs/GoalRelative* message, while the features were put into the *humanoid_league_msgs/PixelRelative* and *humanoid_league_msgs/PixelsRelative* messages. The message definitions can be found in B.8 and B.9.

So these measurements were simulated without errors, while the lines points and field boundary points were subject to the usual errors that occur in the Bit-Bots vision pipeline. Figure 5.1 shows an example image taken from the robot's camera in gazebo.

All in all seven different realistic scenarios recorded to *rosbags* [21]. In all scenarios the robot was standing on the playing field, moving its camera with a fixed pattern. The head turned from left to right and back, with the direction of gaze being further up in the one direction and further down in the other one. In some scenarios, the robot moved around.

The localization node was configured for the evaluation with the settings in Listing C.1. The publishing rate was set to 25, as the highest frequency of subscribed messages was around 20.

On each scenario, ten trials were run, to evaluate repeatability. Each trial was initialized with a new pseudo-random distribution of particles. This way, independence from a specific distribution could be evaluated. The results were published using the custom *humanoid_league_localization/Evaluation* message type. Its definition can be found in B.10.

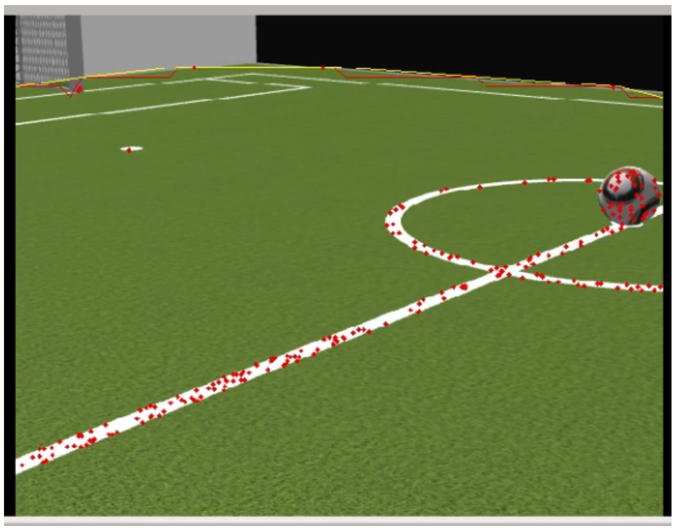


Figure 5.1.: Exemplary image from the robot's camera in Gazebo. The red points visualize line point detection, the yellow line the detected field boundary.

For each filter steps it contains the `header`, the covariance matrices of the best 1 %, 5%, 10%, 20% and over all particles, whether resampling was done in that step and the number of detections for each information source. In Section 5.1, the experiment on localization is described, and the results are presented. The experiment on pose tracking and its results are shown in Section 5.2.

## 5.1. Localization

This experiment was conducted to evaluate the ability of the developed MCL variant to estimate the correct pose with particles initialized over one field half.

### 5.1.1. Setup

The first part of the evaluation was conducted on scenarios where the robot remained stationary and only moved its head. Three different typical poses were chosen, which are visualized in Figure 5.2: a start scenario, marked in red, a striker's pose close to the center circle, marked in green and the goal keeper's pose inside the goal, marked in blue.

In this experiment at initialization, the particles were distributed over the current half, see Chapter 4.1.3. Each of the scenarios was recorded for about a minute.

### 5.1.2. Results

Firstly a summary of all three scenarios is presented. Then each scenario is analyzed separately.

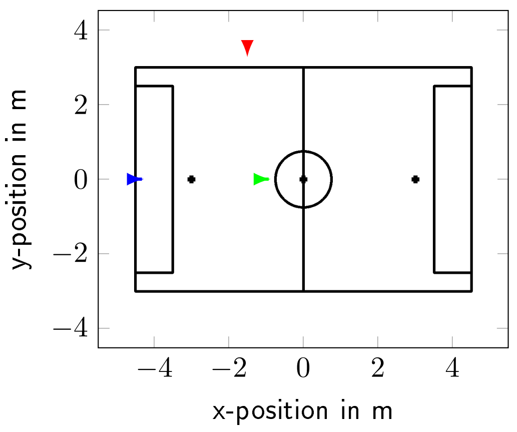


Figure 5.2.: The three arrows show the poses in which the localization experiment was conducted. In red, a starting pose, of a robot about to enter the field is marked. The green arrow is a typical striker's pose, and the blue arrow is the pose of the goalkeeper.

**All scenarios**

Figure 5.3 shows the error distributions between the localization estimates and the true pose over all three localization scenarios. In each of the 15 conditions, 30 trials were run, 10 on each scenario. The box plots show the error distribution over all filter steps.

In both criteria, the euclidean distance error and the angular error, all lower whiskers reach zero as well as most first quartiles. The upper whiskers in most conditions stay below 3 m and 1 rad. The highest ones reach beyond 7 m and up to 3.14 rad, where 3.14 rad is the highest possible value on that scale, as it describes the opposite orientation. The highest possible value for the euclidean distance can be seen to be the highest expansion of the field, which is 12.76 m. Due to the initial distribution of particles over one field half only, this maximal error might be unlikely to reach but still is possible due to the particle movement caused by *diffusion*. Most median values across conditions in the euclidean distance error are below 1 m and below 0.5 rad in the angular error.

In general, it can be said, that broader error distributions on the euclidean error correspond with broader distributions on the angular error, for example in the G, F and G/F conditions. It is not the case for FB/F and L/G/F, but here the medians on the euclidean distance error are relatively low with values below 1m, compared with the other conditions.

Among the four conditions containing only one kind of information source, lines (L) and field boundary (FB) produced the least erroneous results, with small interquartile ranges on both criteria and low medians close to zero.

This result can also be found in the conditions containing two information sources, where the combinations including lines (L/G, L/FB, and L/F) achieved more precise localization estimates than the ones without lines. Among the conditions not containing lines, the two including the field boundary (G/FB and FB/F) achieved the lowest median values, but the third quartile is quite high in the FB/F condition.

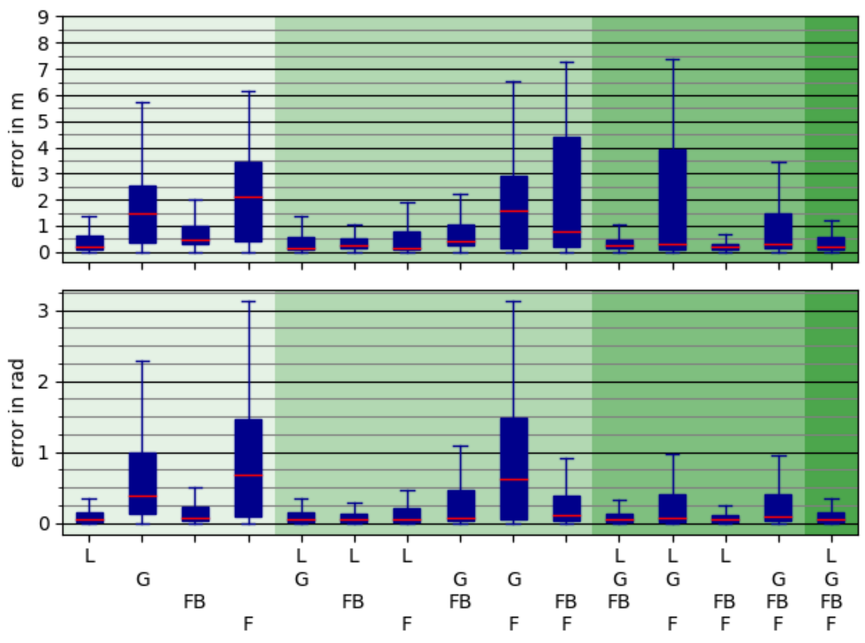


Figure 5.3.: Error distributions over all scenarios in the localization experiment. The absolute error in euclidean distance is given in m and the absolute angular error in rad. The distributions are presented as box plots, where the upper and lower whisker reach to the first data points within 1.5 times the interquartile range. The green tone in the background indicates the number of information sources that were used per condition, from light green for the single information conditions to dark green for the condition containing all four information sources. The absolute error in euclidean distance is given in $m$ and the absolute angular error in $rad$.

The median values among the conditions with three input sources were all close to zero in both criteria. Those including line and field boundary information (L/G/FB and L/FB/F) resulted in narrower distributions with upper whiskers around 1 in the euclidean distance error compared to roughly 7.5 m and 3.5 m for L/G/F and G/FB/F and about 1 rad in the angular error for both.

By including all four information sources for localization (L/G/FB/F), the median value is close to zero, and the deviation is quite narrow, with the upper whisker below 1.5 m for the euclidean distance error and below 0.5 rad for the angular error.

All in all, the best results were achieved in the L/FB/F condition. The conditions L/G/F-B/F, L/G/FB, and L/FB had similar results, and even the error distribution of only lines (L) was not much worse.

Line information seems to be the most reliable information source for localization, as it produces the most precise localization estimates regarding euclidean distance error as well as angular error. The already good precision can be further improved by including field boundary information. Additionally, including the field marking features makes it even more precise.

The second-best information source for localization is the field boundary data, which results in slightly more erroneous pose estimates than line data.

### Analysis of scenarios

To check, whether these findings can be applied across situations, an equivalent analysis is done for each of the three situations.

**Entering the field** In Figure 5.4 the error distributions for all 15 conditions on the start scenario data on 10 trials are visualized. The robot's pose in this scenario is visualized in

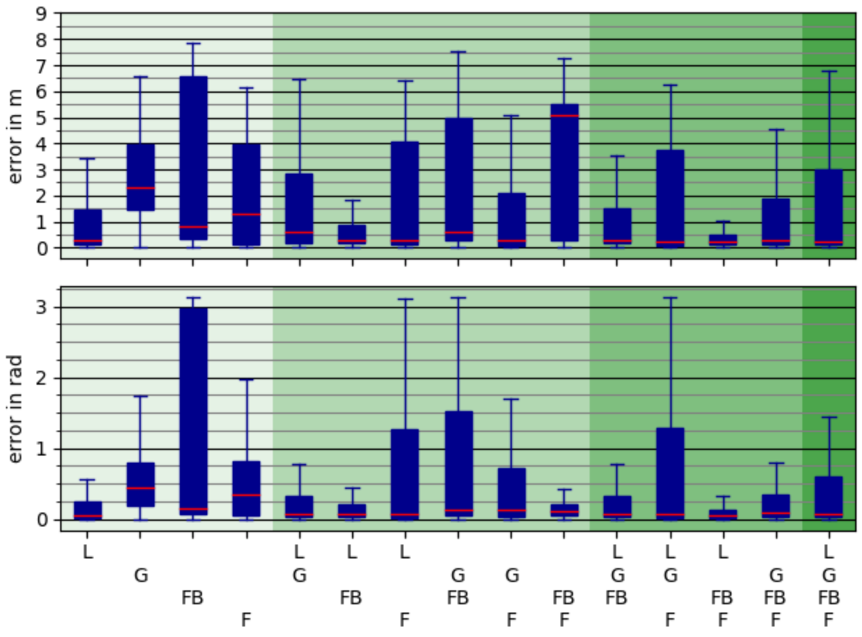


Figure 5.4.: Error distributions in the first scenario of the localization experiment: The robot is about to enter the field from the touchline.

Figure 5.2 with the red mark. Again all lower whiskers are around 0, but the interquartile distances and ranges are larger in both criteria (the euclidean distance error and the angular error) compared to the overall analysis in Figure 5.3. The median values are in a similar range, with the exception of condition, where the median in the euclidean distance error is around 5 m.

Regarding the median, a similar role of lines and field boundary can be found in the one-information-conditions, as they have the lowest median values in both criteria. The distribution of errors in the line condition is similar to the overall analysis, but in the field boundary condition, the interquartile range and upper whisker are a lot higher in both criteria.

Among the conditions with two information sources, the combination of lines and field boundary information (L/FB) bears the smallest risk for errors with a low median value and narrow distribution. While most conditions in this category have low median values in both criteria, the combination of field boundary information and features (FB/F) has a very high one for euclidean distance error, with around 5 m.

The other combinations containing two information sources (L/G, L/F, G/FB, and G/F) all have low median values in euclidean distance error and angular error, but quite high ranges in at least on of them.

For the four conditions with three information sources again, L/FB/F achieved least localization estimate errors. The other conditions (L/G/FB, L/G/F, and G/FB/F) have similarly low median values in both criteria, but broader distributions.

The condition with all four information sources (L/G/FB/F) resulted in low median values, but relatively large interquartile distances and large ranges in both criteria.

The least erroneous results were achieved in this scenario in the L/FB/F condition. Similar results could only be achieved in the L/FB condition.

So all in all, line information seems to be the most precise information source in the situation of a robot entering the playing field. Adding field boundary information (L/FB) helps to reduce localization errors in euclidean distance as well as angle. Including features (F/FB/F) improves the estimation even more. The resulting error distribution in this condition on both criteria is similar to the one achieved in the overall analysis, while many other conditions achieved less precise estimates.

**Striker** In Figure 5.5, the localization errors for the striker scenario is plotted. The robot's pose in this scenario is visualized in Figure 5.2 with the green mark. As before, all first quartiles in both criteria (euclidean distance error and angular error) are close to zero error, as well as most median values. Many conditions have a very narrow error distribution with upper whiskers below 1 m and 0.5 rad.

As in the overall analysis, the best single information conditions are lines (L) and field boundary information (FB) with very small deviations in both criteria in the *lines* condition and only slightly more in the *field boundary* condition.

Of the conditions with 2 information sources, the best estimates were achieved by the ones containing lines (L/G, L/FB, L/F) as well as the combinations containing field boundary information (G/FB and FB/F) information. Among these, only FB/F has the upper whisker over 1 m and 0.5 rad. The other four error distributions are even smaller.

All condition with three information sources achieved very gut results with upper whiskers under 1m and 0.5 rad, as well as the condition containing all four information sources.

In this scenario, again lines and field boundary information provided a good information source for localization, as well as all combinations, including one of them and other sources.

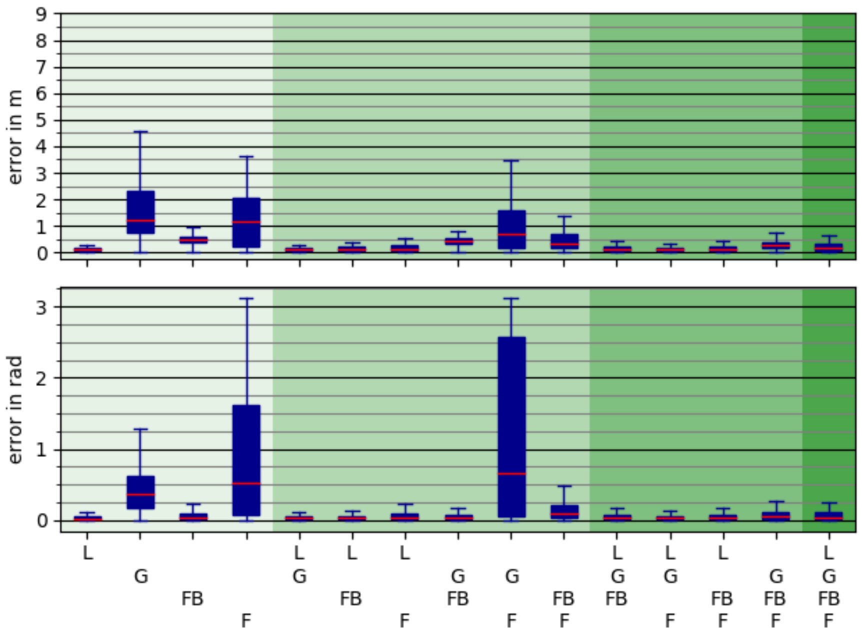


Figure 5.5.: Error distributions in the second scenario of the localization experiment: The robot is standing in front of the center circle.

Many conditions had very narrow error distributions close to zero, including L/G/FB/F, all combinations of three, most combinations of two but also the lines and field boundary information on their own.

This indicates that the localization task on the strikers' pose might be easier than when entering the field.

**Goal keeper** In Figure 5.6, the error distributions for localizing at a goal keeper's pose are visualized. The robot's pose in this scenario is visualized in Figure 5.2 with the blue mark. The data range is very different among conditions, as some conditions have very small ranges close to zero and some have very wide ones, especially on the angular error.

Median values range from close to zero up to 4 m in the euclidean distance error and from close to zero rad up to pi.

It striking, that two error distributions have their lower whisker and first quartile quite far away from zero in the euclidean distance error.

One of them is the field boundary condition (F), which was also not very precise in the other two scenarios. Here, the localization estimate provided by it is not only unprecise but can even be seen to be usually incorrect. The best conditions among the single information conditions, here again, are lines and field boundary information (L, FB), as they have narrow distributions close to zero on both conditions.

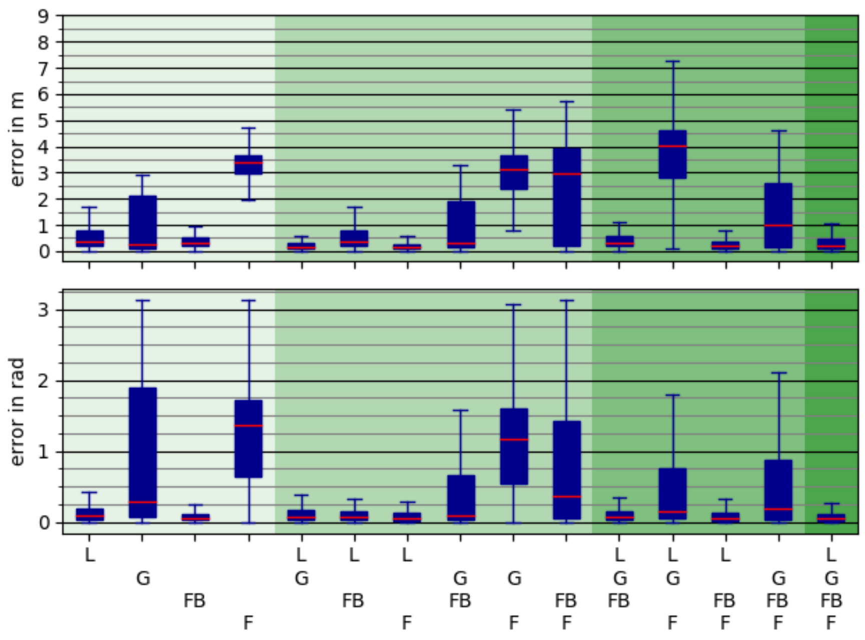


Figure 5.6.: Error distributions in the first scenario of the Localization experiment: The robot is standing inside the goal.

> In this scenario again the conditions with two information sources are most precise if they contain line information (L/G, L/FB, L/F), with very narrow error distributions close to zero.
>
> The combination of line and field boundary information is also most successful among the conditions with three information sources (L/G/FB and L/FB/F). The condition with all four inputs also achieved quite precise results with narrow distributions close to zero on both criteria.

This experiment showed that in many conditions, a good pose estimation could be achieved.
Across all scenarios, it can be seen, that relying on line information only can already result in reasonable localization estimates. This also applies to the field boundary information, except for the first scenario, where the field boundary alone bears a high risk to localize wrongly, with errors of several meters in euclidean distance 3 rad in angle.
Combining line and field boundary information leads to reliable localization estimation across all scenarios. Adding more information sources as goal posts or features does not improve the estimation precision a lot further, as line and field boundary information already result in very small errors. In fact, including goalposts and feature information led to slightly more errors, especially in the first scenarios. This effect might be caused by the small sample size of ten trials each.
The results show that different scenarios can be quite different in difficulty, from relatively high errors over all conditions in scenario one, quite small errors over most conditions in

scenario two, to scenario three, where in some conditions most errors were far away from zero.

## 5.2. Pose Tracking

To evaluate how a well a pose estimation is updated according to the robot's movement, this experiment was done.

### 5.2.1. Setup

The second part of the experiment was done on moving scenarios. In Figure 5.7 the four different realistic scenarios are visualized: two scenarios of the robot moving from a pose outside the field to two different start poses on the field (red and green lines), one scenario of a goalkeeper positioning itself inside its goal (dark blue line) and one of a robot turning around its own axis (blue cross). In each of the scenarios, finally, the robot was oriented towards the other field half.

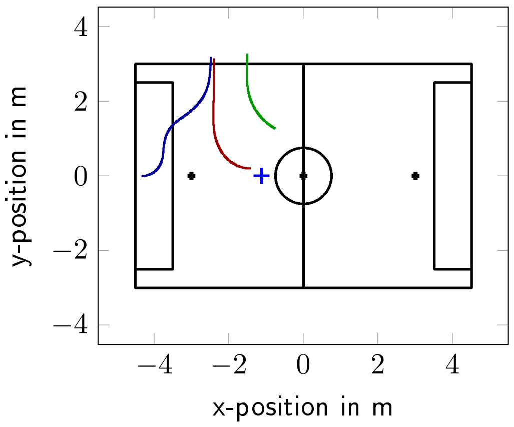


Figure 5.7.: Visualization of the four scenarios in the experiment on pose tracking.

As in the first part of the experiment, ten trials were run per condition in each scenario. The initial distribution, unlike the first part of the experiment, was around the true starting pose.

### 5.2.2. Results

First of all, a summary of all four scenarios is presented, then all scenarios are analyzed separately.

#### All Scenarios

Figure 5.8 shows the absolute error distribution between the pose estimate and the true pose. In each of the 15 conditions, 40 trials were run, 10 on each scenario. The box plots show the error distribution over all filter steps.

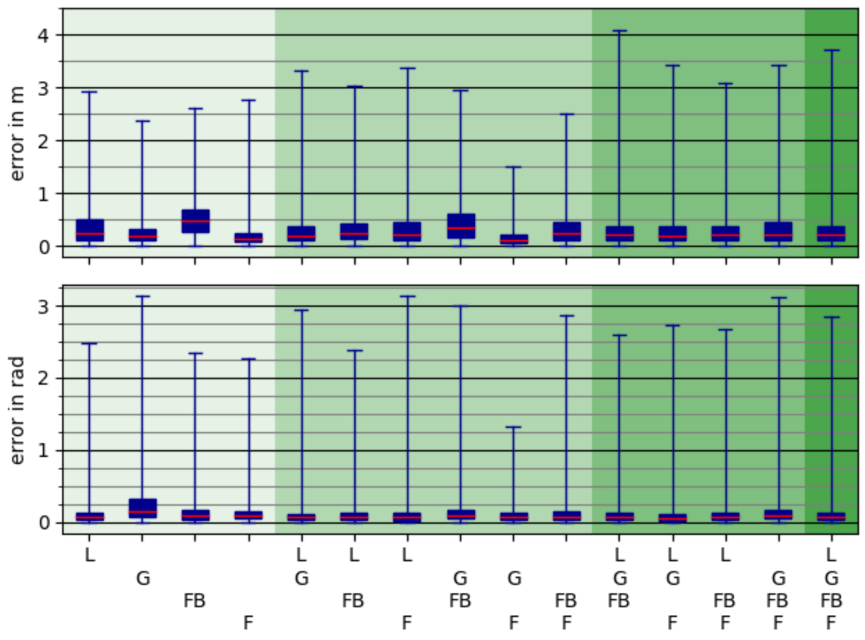


Figure 5.8.: Error distributions over all scenarios in the experiment on pose tracking. The plot is equivalent to Figure 5.3, except that the whiskers in this plot show the maximal and minimal values of the distribution.

In comparison with the equivalent plot from the first part of the experiment, Figure 5.3, fewer differences across conditions can be found. Please note that the range of scale on the euclidean plot has changed from 9 m to 4 m.

As in the localization experiment, the lower whiskers on both the euclidean distance error and the angular error are at 0. In most scenarios, the estimated pose had less than 0.5 m and 0.2 rad error in 75% of the data. As the upper whiskers show the maximal value over all 4 scenarios, they are discussed in the scenario they appeared. All median values are quite low, with most values below 0.25 m and 1 rad.

Although the differences across conditions are small, still the *features* condition (F) has the lowest third quartile in euclidean distance error and the angular error among the conditions with only one information source. Among the conditions with two information sources, the combination of goalposts and features (G/F) has the lowest third quartile in euclidean distance compared to the others, while the angular error distribution is very similar to the other condition with two information sources. There seems to be no difference in the error distributions between the combinations of three information sources. The combination of all for information sources also seems to provide a quite precise pose estimate, but not as good as goalposts and features or features alone.

### Scenarios

To check, whether these findings can be applied across situations, an equivalent analysis is done for each of the four situations.

**Walking** In Figure 5.9 and 5.10 two similar scenarios are visualized. Both are entering the field from a start position, ending inside the own half, facing the opponent's goal.

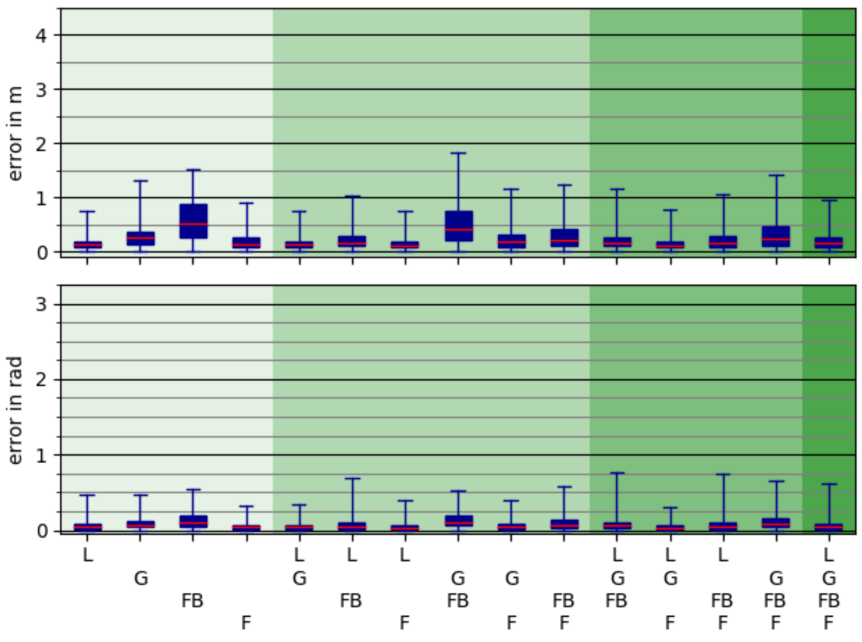


Figure 5.9.: Error distributions in the first scenario of the Pose Tracking experiment: The robot is entering the field from the touchline.

They were chosen to be similar, to draw conclusions on the relation between similarity in situation and the similarity of the resulting error distributions. The similarity can be found in the plots. In many conditions, they have slightly narrower error distributions on both criteria, compared to plot 5.8. The upper whiskers, which mark the maximal value, range between 0.5 and 2 m and an 0.2 rad and 1.2 rad.

The line points resulted in the most precise localization estimates in both scenarios, and the field boundary produced the most erroneous results among the conditions with only one localization information. In the other conditions, it can be seen that the combination of lines either with goalposts or features or both resulted in smaller errors. Adding the field boundary information as fourth information source increases the error.

In Figure 5.11, two exemplary results of the condition including line points, goal posts, and features, which achieved one of the most precise results, can be seen. For each

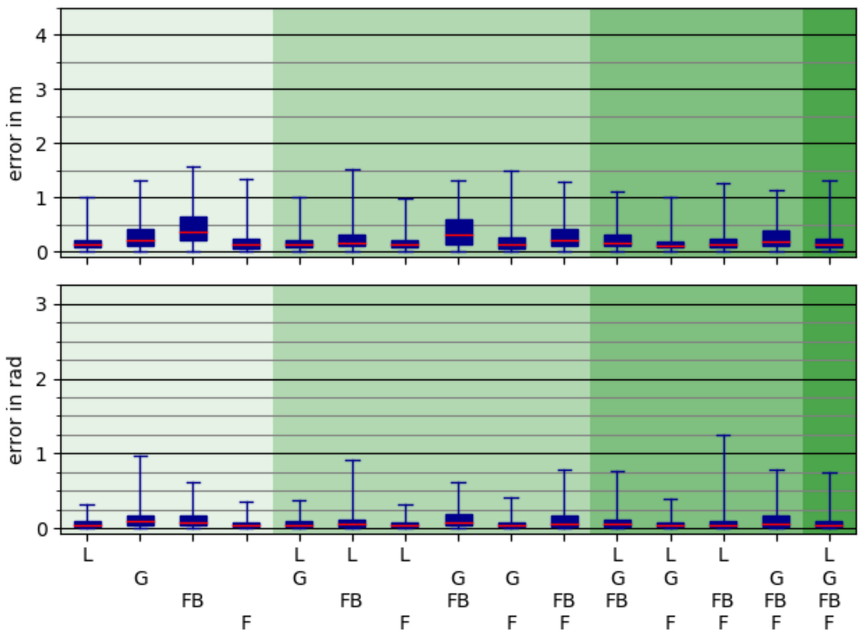


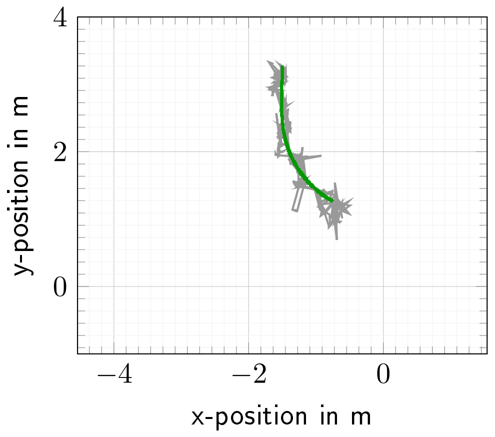
Figure 5.10.: Error distributions in the second scenario of the Pose Tracking experiment: The robot is entering the field and moving to the striker's position.

scenario, one trial was chosen randomly and plotted against the ground truth. The results are subject to the usual noise as it appears in Monte Carlo Localization. It might be reduced with optimized parameter settings. In (b) of that figure, it can be seen, that the initial pose with which the algorithm was initialized was about 50 cm away from the true pose. Still, the localization could recover from that.

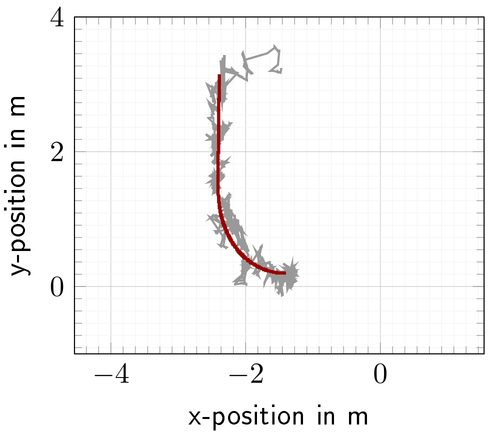
**Turning** In Figure 5.12 the error distributions on a scenario in which the robot is standing in front of the center circle and turning around its own axis is visualized. Similar to the plots of the walking scenarios presented above, the error distributions on both the euclidean distance and the angle are quite narrow and close to zero.

Two things stand out compared to the other scenarios. One is the relatively high upper whisker of over 2 m in the goalposts condition. The other one is the high first quartile in the field boundary condition, as well as in the combination of field boundary information with goalposts (G/FB) and field boundary information with features (FB/F). As before in the walking scenarios, those condition containing line points but not the field boundary information seem to be the most precise ones, but with smaller differences between conditions compared to the walking scenarios.

**Walking and Turning** The plot 5.13 looks quite different from the ones before, with higher third quartiles in both criteria and way higher upper whiskers. In the scenario on which this experiment was executed, the robot enters the field from a start position and



(a) Robot entering the field.



(b) Robot reaching the striker's position. Although the given initial position was about 50 cm away from the true position, the algorithm could recover from that.

Figure 5.11.: Exemplary results of the walking scenarios in the pose tracking experiment. Both plots show randomly chosen trials from the condition including line points, goal posts, and features (L/G/F). The green and red plot are the true position, they gray ones are the estimates.

positions itself inside the goal, facing outside the field. Then it makes a half turn to face inside the field. So this scenario is a combination of the walking and the turning scenarios discussed above. A significant difference compared to the other scenarios are the high upper whiskers in both criteria. Most of them range from around 2.5 to 4 m and from 2.2 to 3.14 rad. The large outliers were caused when the robot reached its position in the goal, facing outside and started turning. In many cases, this movement could not be tracked correctly, and the particle cloud was stuck. So when the robot finished the movement, the angular error half a turn. If this happened often, the particle cloud started to wander around the field randomly, which causes the high outliers in the euclidean error.

In this scenario, the localization information with the smallest errors on both criteria were the features. The most precise condition of the scenario was the combination of features with goalposts (G/F). It is striking here, that the upper whiskers are very small compared to the other conditions in this scenario, with around 1 m and around 1.25 rad. In both conditions combining three kinds of localization information, that contain features and goals (L/G/F and G/FB/F) the error distribution is slightly broader in the euclidean distance and both have quite high upper whiskers.

In Figure 5.14, an exemplary trial from the condition combining goal posts and features is plotted against the robot's true position. In this condition, the angular error did not reach such high values as in many other conditions. So also the error in position stayed

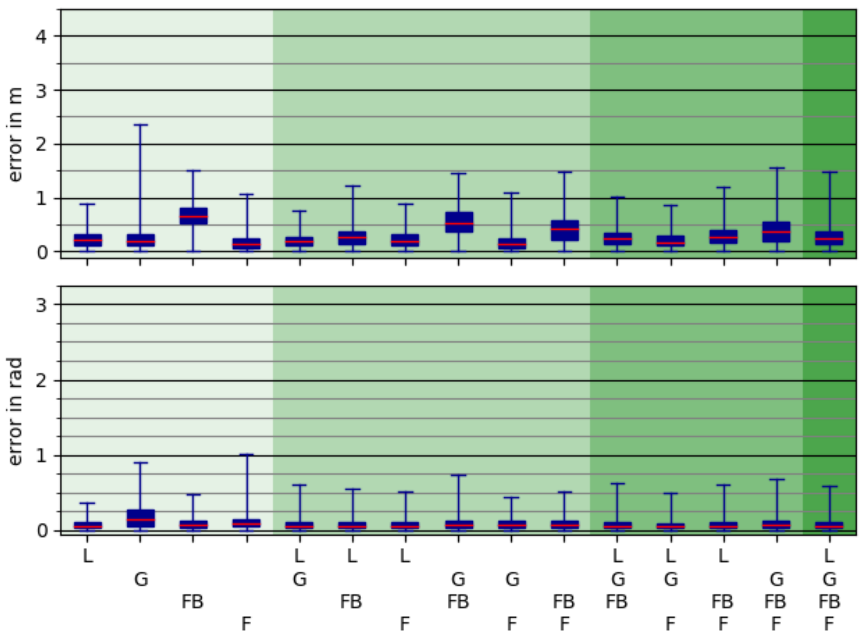


Figure 5.12.: Error distributions in the third scenario of the Pose Tracking experiment: The robot is standing at the striker's position and makes a complete turn.

small, as the particles did not start to wander around. In the exemplary plot, it can be seen that the position stays quite stable when the robot reaches the goal and starts turning.

This experiment has shown several things. As in the localization experiment, different scenarios produced different error distributions. Besides, in this experiment, it could be shown that similar scenarios can result in similar error distributions..

While the overall analysis of all situations indicated that the combination of features and goalposts are the most accurate localization information source, in the walking scenarios as well as in the turning scenario it was the line points. The tendency for features and goalposts is caused by the walking and turning scenario. Facing outside seems to be hard and requires a special localization source.

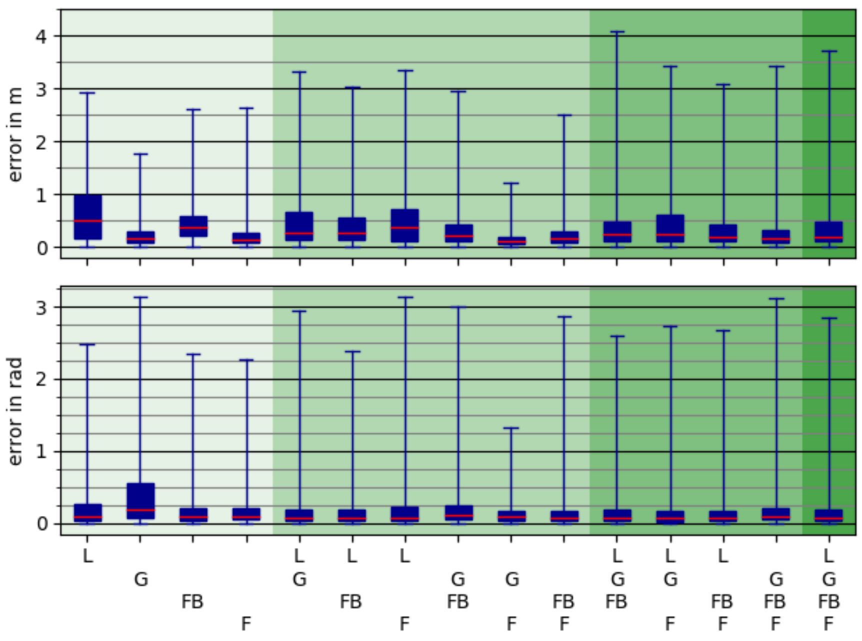


Figure 5.13.: Error distributions in the fourth scenario of the Pose Tracking experiment: The robot is entering the field and moving to the goal keeper's position.

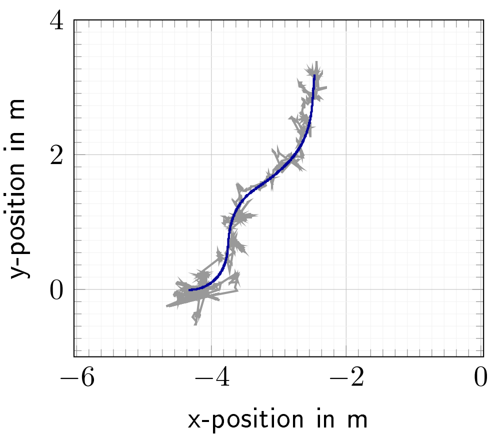


Figure 5.14.: Exemplary results of the Walking-and-Turning scenario in the Pose Tracking experiment. The plot shows one randomly chosen trial from the condition including goalposts and field features (G/F). The true position is plotted in blue and the estimate in gray. Even when the robot started to turn, the position could be tracked precisely.

# 6. Discussion

In the following, the evaluation results are discussed. In Section 6.1, a conclusion on this work is given, and future work is suggested in Section 6.2.

The results of the experiments have shown, that the version of Monte Carlo Localization developed in this work, can estimate the robots pose in terms of localization and pose tracking on specific scenarios with data provided by a simulation environment.

Over all seven tested scenarios, the line points lead to quite accurate localization estimates. Only on the scenario, that included walking and turning, it only scored second. This might be caused by the high number of false-positive line points detection on the goalposts and the net. However, between localization and pose tracking, there were differences in the second-best information for pose estimation. In the stationary localization scenarios, the field boundary information was the next best source for pose estimation.

It often provided a good estimation on its own and improved the estimation when combined with lines. For the pose tracking scenarios, the field marking features were the next best source for localization. In two of the pose tracking scenarios, the *features* condition had the second-best error distribution after the lines, and in one, it was even the best. The reason for this might be the high number of false positives in the line points condition, as mentioned above.

Lines seem to very reliable in general. This might be because they are often inside the field of view. Furthermore, if they are detected precisely, they offer more precise information for localization. For example, the center circle, which is crossed by the centerline, can hardly be mixed up with any other combination of lines on the field. The same applies for the goal areas, which, together with the goal line and the touchlines form uniquely identifiable patterns on the field.

The pose estimation based on field boundary information provided quite accurate results on the stationary scenarios, while in the moving ones, it resulted in more errors. This can be explained by the amount of time it was inside the robots field of view. The head movement pattern had two main perspectives on the field, one with the head lifted more of than the other. The field boundary often was in the robot's field of view, when the head was lifted further up and seldom when the head was more oriented more towards the ground. In the localization scenario, this lack of information over roughly half the time seemed to produce no negative effect on the localization scenarios. An analysis on how much time the conditions needed to find the correct pose might reveal, that it took longer for the field boundary conditions. In the pose tracking scenarios, on the other hand, this absence of information might have led to more erroneous results, as the particles diffuse away from the estimated pose when no information enters the filter.

Field marking features, just as line points, are also quite often in the field of view. They can be seen as a more abstract representation of the same entity. So the low errors they produce on pose tracking scenarios fit the findings from line points. The reason why the field features do not match the results in the localization scenarios needs to be evaluated further. It might also be caused by an implementation error in the script publishing them. The goal post detections, which are published by that script, often occur in 3-5 consecutive filter steps, which seems plausible, as they are in the robot's field for some time. The features though usually appear in only a single filter step. So they provide less information than they actually could. The goals, like the field boundary, are seldom detected and thus were not a very good source for pose estimation over all scenarios. It could be shown the combining information sources, which already provided precise results on their own, the pose estimation accuracy could be further improved. Whether the improvements are significant and justify the increased computational effort required to gain this information needs to be evaluated further. Including information sources, that in the single-information condition led to higher errors, decreased the pose estimation. This shows that only with one good localization source, sufficient results can be achieved, that can be further improved by adding one or two other reliable sources.

The pose estimates errors in the pose tracking experiment, in general, were smaller than in the localization experiment. One reason for that is that in the localization experiment the pose was unknown and so the localization estimate could not be correct. After some time, the particles formed clusters, and the pose estimate became reliable. However, in some trials the estimated pose was incorrect. In these cases, as the robot did not move, only diffusion was applied to the particles and so the MCL could not recover from that. Trials like these increase the estimation error in the localization experiment. In the pose tracking experiment, however, the pose was given and was only lost in some trials of the combined walking and turning experiment. The particles could not track the turning correctly, causing high errors in angle as well as in distance.

While the errors in the localization setup could be reduced by choosing the best combination of localization information according to the current situation, for the pose tracking it can be said, that the source of localization information did not much affect performance. In this experiment setup, the results could be improved if the robot had sensed more information when it started the turning part. This problem could be overcome by adjusting the head movement in such a situation so that more information is available. One strategy might be to choose the direction of gaze according to the turning direction and fixate the corner of the field. Using this approach, the corner of the field, and possibly one goal post would be in sight while the robot is turning. Therefore, at least three kinds of localization information would have been available.

The results seem promising to be used on a real robot. Although the experiment was done simulated data, the quality of that data was not perfect. The line points and field boundary detection was provided by the Bit-Bots vision pipeline, and so contained usual detection errors. Also, the simulated detections seemed to be imperfect due to the implementation error mentioned above.

On the simulated data estimating the robot's pose with an initial particle distribution over one half of the field worked in many cases. With less perfect data from a real-world scenario, more prior knowledge could be used to compensate for that, by using a different initial distribution.

## 6.1. Conclusion

In this work, a version of Monte Carlo Localization was developed for the RoboCup Humanoid League Context. It can integrate several information sources provided by a 2D RGB camera. Furthermore, a node was developed, that handles the initialization of the localization based on prior knowledge and the robot's state. Two experiments on the precision of four different visual information sources for localization and their combinations were run. The data for the experiments was provided by a simulation environment.It could be shown that the line points in many scenarios provide one of the most precise localization estimates. In the localization task also the field boundary information resulted in less erroneous pose estimations and in the pose tracking task, the field marking features produced smaller errors compared to the other kinds of localization information. The results show that the combination of good localization sources can further improve the precision of the pose estimate. While the precision of localization estimates in the localization task depends on the combination of information sources, the performance differences in the pose tracking scenarios were quite small between the combinations. In these scenarios, the performance could be improved by ensuring that any localization information is available.

## 6.2. Future Work

To improve the results of this work, the unknown errors parameter values in the odometry model and in the measurement model should be evaluated. Instead of working on artificial ratings, the algorithm should be changed to work on probabilities. The approach should be evaluated on a real robot with all data provided by the vision pipeline. The number of particles should be adjusted to provide a good tradeoff between computational complexity and performance. Also, a good value for diffusion and resampling rate should be evaluated. The exact weighting between the information sources needs to be evaluated. The results of this work suggest that the weight of the lines should be high. Furthermore, the *distance factor*, that adjusts the reliability of a measurement based on its distance to the robot should be evaluated, as well as a good value for the *resampling interval*.

It could be evaluated, whether other variants of MCL, like KLD sampling, which adapts the number of particles dynamically, improve the results. Another variant of MCL is to include the map into the motion model. By doing so, no particles could be outside the map. That could save computational resources. Although the robot might be standing next to the playing field, in that situation, the transformation from image space to Cartesian space would be corrupted, and so the information would be incorrect.

Moreover, the evaluation could be extended on more information sources for localization, like data provided by a visual compass approach. By doing so, real global localization could be performed, because the information provided by it could solve the problem of symmetry. Also, the abstraction level on the information sources used in this work could be evaluated further. Instead of working on points representing the field markings and the field boundary, it could be evaluated, whether fitting lines through them improves the localization estimates. The corners and t-crossings of the field markings could also be provided, including the orientation.

That might also improve the pose estimation.

Furthermore, the localization handler node could be developed further. If the scattering of particles increases it could request more visual information, either from the vision by adjusting parameters like the number of randomly checked pixel for line points. Alternatively, it could request a different motion, like stopping to walk and look around. This could even be extended to active vision approaches, that control the robots head based on the pose estimate, like [MVY+15].

Moreover, the localization handler could adjust the weighting between the information sources or include and exclude them from the pose estimation.

# Bibliography

[AGH$^+$19] J Allali, L Gondry, L Hofer, P Laborde-Zubieta, O Ly, S N'Guyen, G Passault, A Pirrone, and Q Rouxel. Rhoban football club – team description paper. `https://submission.robocuphumanoid.org/uploads//Rhoban-tdp-5c05011865144.pdf`, 2019.

[Bes17] Marc Bestmann. Towards Using ROS in the RoboCup Humanoid Soccer League, 2017.

[BHJ$^+$18] Kenji Brameld, Fraser Hamersley, Ethan Jones, Tripta Kaur, Liangde Li, Wentao Lu, Maurice Pagnucco, Claude Sammut, Qingbin Sheh, Peter Schmidt, et al. Robocup spl 2018 runswift team paper. `http://cgi.cse.unsw.edu.au/~robocup/2018/TeamPaper2018.pdf`, 2018.

[DFBT99] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, page 1322–1328 vol.2, May 1999.

[DGA] Arnaud Doucet, Simon Godsill, and Christophe Andrieu. On sequential monte carlo sampling methods for bayesian filtering. page 12.

[FBG$^+$19] Niklas Fiedler, Hendrik Brandt, Jan Gutsche, Florian Vahl, Jonas Hagge, and Marc Bestmann. An open source vision pipeline approach for robocup humanoid soccer. In *RoboCup 2019: Robot World Cup XXIII*. Springer, 2019. Accepted.

[Fox02] Dieter Fox. Kld-sampling: Adaptive particle filters. In *Advances in neural information processing systems*, pages 713–720, 2002.

[Gü18] Jasper Güldenstein. Comparison of Measurement Systems for Kinematic Calibration of a Humanoid Robot, 2018.

[KAK$^+$97] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM, 1997.

[Lab04] Frédéric Labrosse. Visual compass. *Proceedings of Towards Autonomous Robotic Systems, University of Essex, Colchester, UK*, pages 85–92, 2004.

[LV00] S. Lenser and M. Veloso. Sensor resetting localization for poorly modelled mobile robots. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 1225–1232 vol.2, 2000.

[MAMP16] Alexandre Muzio, Luis Aguiar, Marcos R.O.A. Maximo, and Samuel C. Pinto. Monte carlo localization with field lines observations for simulated humanoid robotic soccer. In *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, page 334–339. IEEE, Oct 2016.

[MVY+15] Matías Mattamala, Constanza Villegas, José Miguel Yáñez, Pablo Cano, and Javier Ruiz-del Solar. *A Dynamic and Efficient Active Vision System for Humanoid Soccer Robots*, volume 9513, page 316–327. Springer International Publishing, 2015.

[Nao18] Nao-Team HTWK. Team research report nao-team htwk. `https://htwk-robots.de/documents/TRR_2018.pdf`, 2018.

[QCG+09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[RF18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[RJ04] Thomas Röfer and Matthias Jüngel. *Fast and Robust Edge-Based Localization in the Sony Four-Legged Robot League*, volume 3020, page 262–273. Springer Berlin Heidelberg, 2004.

[RLT06] Thomas Röfer, Tim Laue, and Dirk Thomas. *Particle-Filter-Based Self-localization Using Landmarks and Directed Lines*, volume 4020, page 608–615. Springer Berlin Heidelberg, 2006.

[Rob19] RoboCup Humanoid Technical Committee. Laws of the Game 2019. `http://www.robocuphumanoid.org/wp-content/uploads/RCHL-2019-Rules-final.pdf`, 2019.

[SJS+19] Hendawan Soebhakti, Eko Rudiawan Jamzuri, Andrey Karona Sitepu, Alwan Putra, Jony Arif Ricardo, Junito Suroto, and Eko Priono. Barelang fc team description paper humanoid kid size league of robocup 2019. `https://submission.robocuphumanoid.org/uploads//Barelang_FC-tdp-5c4b1cb222c5f.pdf`, 2019.

[SK16] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[TKN06] S. Thompson, S. Kagami, and K. Nishiwaki. Localisation for autonomous humanoid navigation. In *2006 6th IEEE-RAS International Conference on Humanoid Robots*, page 13–19, Dec 2006.

[TTK$^{+}$16] T. Tanaka, T. Tsubakimoto, M. Kawamura, K. Kumagai, H. Matsubara, K. Hidaka, Y. Aizawa, M. Nakagawa, Y. Iwai, T. Suzuki, and K. Kobayashi. Camellia dragons 2016 team description, 2016.

# Online References

[1] “RoboCup Official Website.” https://www.robocup.org. Accessed: 19-09-20.

[2] “RoboCup 2018 Montreal.” http://2018.robocup.org/. Accessed: 19-09-23.

[3] “Robotis Dynamixel MC 64.” http://emanual.robotis.com/docs/en/dxl/mx/mx-64-2/. Accessed: 19-09-24.

[4] “Robotis Dynamixel MC 106.” http://emanual.robotis.com/docs/en/dxl/mx/mx-106/. Accessed: 19-09-24.

[5] “Basler acA2040-35gc Camera.” https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca2040-35gc/. Accessed: 19-09-23.

[6] “Computar M1214-MP2 Lens.” https://www.baslerweb.com/fp-1489067453/media/downloads/documents/accessories_datasheets/lenses/M1214-MP2.pdf. Accessed: 19-09-23.

[7] “ ROS Wiki.” https://wiki.ros.org/. Accessed: 19-09-22.

[8] “rviz - ROS.” https://wiki.ros.org/rviz. Accessed: 19-09-26.

[9] “robot_state_publisher - ROS.” https://wiki.ros.org/robot_state_publisher. Accessed: 19-09-26.

[10] Blaise Gassend, “tf2 - ROS Wiki.” https://wiki.ros.org/tf2. Accessed: 19-09-22.

[11] “REP 105 - Coordinate Frames for Mobile Platforms.” https://www.ros.org/reps/rep-0105.html. Accessed: 19-09-24.

[12] “REP 120 - Coordinate Frames for Humanoid Robots.” https://www.ros.org/reps/rep-0120.html. Accessed: 19-09-24.

[13] Brian P. Gerkey, “amcl - ROS Wiki.” https://wiki.ros.org/amcl. Accessed: 19-08-26.

[14] “sensor_msgs/LaserScan - ROS.” https://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html. Accessed: 19-09-20.

[15] Hamburg Bit-Bots, “particle_filter (GitHub).” https://github.com/bit-bots/particle_filter. Accessed: 19-07-28.

[16] “PoseWithCovarianceStamped - ROS Wiki.” https://docs.ros.org/kinetic/api/geometry_msgs/html/msg/PoseWithCovarianceStamped.html. Accessed: 19-09-29.

[17] Blaise Gassend, “dynamic_reconfigure - ROS Wiki.” https://wiki.ros.org/dynamic_reconfigure. Accessed: 19-09-22.

[18] Hamburg Bit-Bots, “humanoid_league_msgs (GitHub).” https://github.com/bit-bots/humanoid_league_msgs/tree/master/msg. Accessed: 19-08-25.

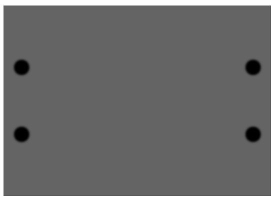
[19] “visualization_msgs/MarkerArray - ROS.” https://docs.ros.org/api/visualization_msgs/html/msg/MarkerArray.html. Accessed: 19-09-29.

[20] “Gazebo.” http://gazebosim.org/. Accessed: 19-09-29.

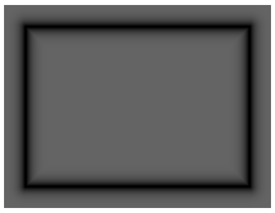
[21] Tim Field, Jeremy Leibs, James Bowman, “rosbag - ROS Wiki.” https://wiki.ros.org/rosbag. Accessed: 19-08-26.

# Appendices

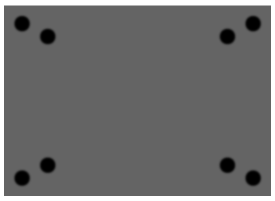
# A. Likelihood Field Lookup Tables



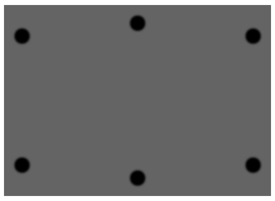
(a) Lookup table for goal posts data.



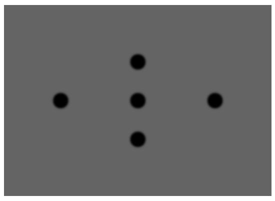
(b) Lookup table for field boundary data.



(c) Lookup table for corners data.



(d) Lookup table for t-crossings data.



(e) Lookup table for crosses data.

Figure A.1.: Likelihood field lookup tables. Points, that are transformed onto black regions get the highest weight, points on the light gray regions a very low on. Points, that do not correspond to a pixel/grid cell get the lowest weight.

# B. Message Definitions

```
# Contains points in the image representing the field boundary.

# The header is included to get the time stamp for later use in tf
std_msgs/Header header

# The points representing the field boundary.
# They are taken in the image coordinate system (x->right, y->down)
geometry_msgs/Point[] field_boundary_points
```

Listing B.1: Definition of the FieldBoundaryInImage Message.

```
# Relative position to a point on the field boundary

# The header is included to get the time stamp for later use in tf
std_msgs/Header header

# The points representing the field boundary.
geometry_msgs/Point[] field_boundary_points
```

Listing B.2: Definition of the FieldBoundaryRelative Message.

```
# This message provides all information from the game controller
# for additional information see documentation of the game controller
# https://github.com/bhuman/GameController


std_msgs/Header header

uint8 GAMESTATE_INITAL=0
uint8 GAMESTATE_READY=1
uint8 GAMESTATE_SET=2
uint8 GAMESTATE_PLAYING=3
uint8 GAMESTATE_FINISHED=4
uint8 gameState

# Secondary state, penaltyshoot is penalty shootout at the end of the
    game,
# penaltykick is a kick during the game
uint8 STATE_NORMAL = 0
uint8 STATE_PENALTYSHOOT = 1
uint8 STATE_OVERTIME = 2
uint8 STATE_TIMEOUT = 3
uint8 STATE_DIRECT_FREEKICK = 4
uint8 STATE_INDIRECT_FREEKICK = 5
uint8 STATE_PENALTYKICK = 6
uint8 secondaryState

# For newest version of game controller
# Tells which team has the free kick or penalty kick
uint8 secondaryStateTeam

bool firstHalf
uint8 ownScore
uint8 rivalScore

# Seconds remaining for the game half
int16 secondsRemaining
# Seconds remaining for things like kickoff
uint16 secondary_seconds_remaining

bool hasKickOff
bool penalized
uint16 secondsTillUnpenalized
# Allowed to move is different from penalized.
# You can for example be not allowed to move due to the current state of
    the game
bool allowedToMove

# Team colors
uint8 BLUE = 0
uint8 RED = 1
uint8 teamColor

bool dropInTeam
```

```
uint16 dropInTime

# The number of the current penalty shot during penalty shootout
uint8 penaltyShot
# a binary pattern indicating the successful penalty shots (1 for
    succesful, 0 fpr unsuccessful)
uint16 singleShots

string coach_message
```

Listing B.3: Definition of the GameState Message.

```
# Relative position to a goal

# The header is included to get the time stamp for later use in tf
std_msgs/Header header

# Position of the left goal post feet (in meter)
geometry_msgs/Point left_post

# Position of the right post, null if only one post was seen
geometry_msgs/Point right_post

# Vector pointing to the (probable) center of the goal (in meters).
# Should only be used if only one goal post is visible. If both are
    visible this should be none.
# This is normally an educated guess, using the goal bar or the position
    of the post on the image
geometry_msgs/Point center_direction

# A certainty rating between 0 and 1, where 1 is the surest.
# 0 means no goal was found
float32 confidence
```

Listing B.4: Definition of the GoalRelative Message.

```
# Contains all relative information about line features on the field

# The header is included to get the time stamp for later use in tf
std_msgs/Header header

LineIntersectionRelative[] intersections
LineSegmentRelative[] segments
LineCircleRelative[] circles
```

Listing B.5: Definition of the LineInformationRelative Message.

```
# A line segment relative to the robot

# Start and end position of the line
# x in front of the robot
# y to the left
# z should be 0
```

```
geometry_msgs/Point start
geometry_msgs/Point end

# A certainty rating between 0 and 1, where 1 is the surest.
float32 confidence
```

Listing B.6: Definition of the LineSegmentRelative Message.

```
# This message provides the current state of the hardware control manager
     (HCM), which is handling falling, standing up and the decision
# between playing animations and walking

# Robot can be controlled from a higher level
uint8 CONTROLABLE=0
# Robot is currently falling
# it can not be controlled and should go to a position that minimizes the
     damage during a fall
uint8 FALLING=1
# Robot is lying on the floor
# maybe reset your world model, as the state should be unsure now
uint8 FALLEN=2
# Robot is currently trying to get up again
uint8 GETTING_UP=3
# An animation is running
# no walking or further animations possible
# Falling detection is deactivated
uint8 ANIMATION_RUNNING=4
# The hardware control manager is booting
uint8 STARTUP=5
# The hardware control manager is shutting down
uint8 SHUTDOWN=6
# The robot is in penalty position
# It can not be controlled
uint8 PENALTY=7
# The robot is getting in or out of penalty position
uint8 PENALTY_ANIMANTION=8
# The robot is used for recording animations
# Reserved all controling to a recording process
# No falling detection is processed and no stand ups will be done
uint8 RECORD=9
# The robot is walking
uint8 WALKING=10
# A state where the motors are turned off, but the hardware control
    manager is still waiting for commandos and turns the motors on,
# if a move commando comes
uint8 MOTOR_OFF=11
# Last status send by the hardware control manager after shutting down
uint8 HCM_OFF=12
# Robot has hardware problems and is not controlable
uint8 HARDWARE_PROBLEM=13
# Robot is currently picked up by a human. Should normally not move
    during this time.
uint8 PICKED_UP=14
# Robot is currently kicking the ball
```

```
uint8 KICKING=15

uint8 state
```

Listing B.7: Definition of the RobotControlState Message.

```
# A pixel with one channel with a position in Cartesian space (for
    transformed heatmaps)

geometry_msgs/Point position

float32 value # between 0 and 1
```

Listing B.8: Definition of the PixelRelative Message.

```
# A list of pixels with one channel with a position in Cartesian space (
    for transformed heatmaps)

std_msgs/Header header

PixelRelative[] pixels
```

Listing B.9: Definition of the PixelsRelative Message.

```
std_msgs/Header header

float64[36] cov_estimate_best
float64[36] cov_estimate_5
float64[36] cov_estimate_10
float64[36] cov_estimate_20
float64[36] cov_mean

int32 resampled

int32 lines
int32 goals
int32 fb_points
int32 corners
int32 tcrossings
int32 crosses
```

Listing B.10: Definition of the Evaluation Message.

# C. Localization Parameters

```

########
# MISC #
########

# the initial pose of the robot
```

```
initial_robot_x1: 2.5
initial_robot_y1: 3
initial_robot_t1: -1.5
initial_robot_x2: 2.5
initial_robot_y2: -3
initial_robot_t2: 1.57

initial_robot_x: -4.5
initial_robot_y: 0
initial_robot_t: 0

init_mode: 4

map_path_lines: '../models/lines_simulator.png'
map_path_goals: '../models/posts.png'
map_path_field_boundary: '../models/fieldboundary.png'

map_path_corners: '../models/corners.png'
map_path_crosses: '../models/crosses_simulator.png'
map_path_tcrossings: '../models/tcrossings.png'

#field size
field_x: 9 #5.45
field_y: 6 #3.88

# for field boundary map
field_padding: 0.7 #0.1

field_boundary_interpolation_steps: 1


#############
# ROS-Stuff #
#############

line_topic: 'line_relative'
non_line_topic: 'non_line_field_points_relative'
goal_topic: 'goals_simulated'
fieldboundary_topic: 'field_boundary_relative'
fieldboundary_in_image_topic: 'field_boundary_in_image'
corners_topic: 'corners'
tcrossings_topic: 'tcrossings'
crosses_topic: 'crosses'

publishing_frame: '/localization_estimate'

pose_publishing_topic: 'pose'

particle_publishing_topic: 'pose_particles'

publishing_frequency: 25

#################
```

```
# Visualization #
#################

debug_visualization: true


###################
# Particle Filter #
###################


particle_number: 500
resampling_interval: 15

use_lines: false
use_non_lines: false
use_goals: false
use_fieldboundary: false
use_corners: false
use_tcrossings : false
use_crosses: false

diffusion_x_std_dev: 0.8
diffusion_y_std_dev: 0.8
diffusion_t_std_dev: 0.5
diffusion_multiplicator: 0.05

min_weight: 0.01
min_resampling_weight: 0.5
percentage_best_particles: 5

distance_factor : 0.5
lines_factor: 1
non_lines_factor: 0
goal_factor: 1
field_boundary_factor: 1
corners_factor : 1
tcrossings_factor: 1
crosses_factor : 1

min_motion_linear: 0.0
min_motion_angular: 0.0
filter_only_with_motion: false
```

Listing C.1: Parameters of the Localization Node.

## Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 30. September 2019

Judith Hartfill

## Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 30. September 2019

Judith Hartfill