TECHNICAL UNIVERSITY OF CRETE

Visual-Feature-based Self-Localization for Robotic Soccer

Eleftherios A. Chatzilaris

A thesis submitted in partial fulfillment for the

Diploma Degree

in the

Department of Electronic and Computer Engineering

Thesis committee:

Michail G. Lagoudakis, Supervisor
Nikolaos Vlassis
Antonios Deligiannakis

Chania, October 2009

‘‘Ουδέν κακόν αμιγές καλού.’’

TECHNICAL UNIVERSITY OF CRETE

Abstract

Department of Electronic and Computer Engineering

by Eleftherios A. Chatzilaris

The ability of Robot Self-Localization is a central problem in the context of RoboCup, the annual international robotic soccer competition. Each robot player needs to know its own current location in the field not only for legal play (preventing leaving the field or entering the own goal area as illegal defender), but also for team play (coordination with teammates, attack/defense formations). Localization in the Standard Platform League (SPL), where all participating teams consist of identical Aldebaran Nao humanoid robots, must rely on a limited number of visual field features (the two goals and the field line markings) and is hindered by the frequent penalizations which are equivalent to the well-known problem of ‘‘robot kidnapping". This thesis describes a localization method for the SPL based on Monte-Carlo Localization with the Auxiliary Particle Filter. The required motion and observation models for the Nao robot are learned using statistical methods and extensive experimentation to adapt to the current locomotion and recognition abilities of the robot and special attention has been given to efficiency issues for on-board execution. The proposed method provides fairly accurate localization estimates for effective play in both the simulated (Webots) and the real SPL fields, and has been used extensively by Kouretes, the RoboCup team of the Technical University of Crete.

Περίληψη

Η ικανότητα του αυτοπροσδιορισμού θέσης ενός ρομπότ είναι ένα κεντρικό ζήτημα στο πλαίσιο του ετήσιου διεθνή διαγωνισμού ρομποτικού ποδοσφαίρου RoboCup. Κάθε παίκτης-ρομπότ πρέπει να γνωρίζει τη δική του τρέχουσα θέση μέσα στο γήπεδο, όχι μόνο για να παίζει σύμφωνα με τους κανόνες (αποφυγή αποχώρησης από τα όρια του γηπέδου ή εισχώρησης στη μικρή περιοχή του δικού του τέρματος ως παράνομος αμυντικός), αλλά και για ομαδικό παιχνίδι (συντονισμός με τους συμπαίκτες, σχηματισμοί επίθεσης/άμυνας). Ο προσδιορισμός θέσης στο πρωτάθλημα Standard Platform League (SPL), όπου όλες οι συμμετέχουσες ομάδες αποτελούνται από όμοια Aldebaran Nao ανθρωποειδή ρομπότ, βασίζεται σε έναν περιορισμένο αριθμό οπτικών χαρακτηριστικών γηπέδου (τα δύο τέρματα και οι γραμμές του γηπέδου) και δυσχεραίνεται από τις συχνές ποινές που είναι ισοδύναμες με το γνωστό πρόβλημα της "απαγωγής του ρομπότ" (robot kidnapping). Η παρούσα διπλωματική εργασία περιγράφει μια μέθοδο για αυτοπροσδιορισμό θέσης στο πρωτάθλημα SPL που βασίζεται στον αλγόριθμο εντοπισμού Monte-Carlo με βοηθη-τικό φίλτρο σωματιδίων (Auxiliary Particle Filter). Τα απαιτούμενα μοντέλα κίνησης και παρατήρησης για το ρομπότ Nao εξάγονται με τη χρήση στατιστικών μεθόδων και εκτεταμένο πειραματισμό για να προσαρμόζονται στις τρέχουσες ικανότητες κίνησης και αναγνώρισης του ρομπότ και ιδιαίτερη προσοχή έχει δοθεί σε θέματα απόδοσης για την εκτέλεση των αλγορίθμων πάνω στο ίδιο το ρομπότ. Η προτεινόμενη μέθοδος παρέχει αρκετά ακριβείς εκτιμήσεις θέσης για αποτελεσματικό παιχνίδι τόσο στην προσομοίωση (Webots), όσο και στο πραγματικό γήπεδο του SPL, και έχει χρησιμοποιηθεί εκτενώς από τους Κουρήτες, την ομάδα ρομποτικού ποδοσφαίρου του Πολυτεχνείου Κρήτης.

Contents

List of Figures

Abbreviations

SPL	**S**tandard **P**latform **L**eague
SIR	**S**ampling **I**mportance **R**esampling
MCL	**M**onte **C**arlo **L**ocalization

Chapter 1

Introduction

This thesis addresses the problem of self-localization in a robotic field. The ability of Robot Self-Localization is a central problem in the context of RoboCup, the annual international robotic soccer competition. Each robot player needs to know its own current location in the field not only for legal play (preventing leaving the field or entering the own goal area as illegal defender), but also for team play (coordination with teammates, attack/defense formations). Localization in the Standard Platform League (SPL), where all participating teams consist of identical Aldebaran Nao humanoid robots, must rely on a limited number of visual field features (the two goals and the field line markings) and is hindered by the frequent penalizations which are equivalent to the well-known problem of ''robot kidnapping". This thesis describes a localization method for the SPL based on Monte-Carlo Localization with the Auxiliary Particle Filter. The required motion and observation models for the Nao robot are learned using statistical methods and extensive experimentation to adapt to the current locomotion and recognition abilities of the robot and special attention has been given to efficiency issues for on-board execution. The proposed method provides fairly accurate localization estimates for effective play in both the simulated (Webots) and the real SPL fields, and has been used extensively by Kouretes, the RoboCup team of the Technical University of Crete.

1.1 Motivation

The lack of a localization system in our team was the first motivation for constructing such a system.

1.2 Thesis Outline

Chapter 2 provides the necessary background of the ideas this thesis builds upon.

Chapter 3 discusses the problem at the RoboCup environment.

Chapter 4 introduces our approach and the algorithms that we chose.

Chapter 5 discusses the implementation and the ideas that where used.

Chapter 6 demonstrates the ability of the system to localize in the simulated field along with the real environment.

Chapter 7 concludes this thesis with a brief summary.

Chapter 2

Background

2.1 RoboCup

The Robocup is an annual international robotics competition. The basic concept is to promote artificial intelligence and robotics research through soccer games between robots. Initial idea introduced back in 1993 and the first official competition was held in Nagoya, Japanm in 1997. The Robocup Competition has a long-term vision:

''By the year 2050, to develop a team of fully autonomous humanoid robots that can win against the human world soccer champions''.

Using robotic soccer as the benchmark domain all robotic problems come to surface and thru this competition newly proposed technologies are being tested and benchmarked potentially for use in other domains as well. In order for a robotic team to play effective robotic soccer, various technologies have to be integrated. Technologies must address the basic problems of robotics which are perception, cognition, action, and coordination. So, researchers around the world develop, examine and, integrate the state-of-the-art technologies of artificial intelligence and robotic research in aiming to solve many robotics problems.

The contest in the latest, 13th, edition (Robocup 2009 in Graz, Austria) had four major divisions, each with a number of leagues:

2.1.1 RoboCup Soccer

The main focus of the RoboCup Soccer activities is competitive soccer. Here robots are built and programmed in order to play autonomously or semi-autonomously, Soccer. This competition had five leagues:

Figure 2.1: Kouretes playing in RoboCup2009

Standard Platform League

The RoboCup Standard Platform League (SPL) is a league, in which all teams compete with identical robots, namely the Aldebaran Nao humanoid robots. Therefore, the teams concentrate on software development only, while still using state-of-the-art robots. The robots operate fully autonomously, i.e. there is no external control, neither by humans nor by computers. The soccer field is built on a total carpet area of length 7.4m and width 5.4m, but the soccer field by itself is approximate 6m by 4m. Together with white field lines like a real soccer field there are 2 goal one in yellow color and one in sky blue color. In latest Robocup 2009 competition only ceiling lights were used, a significant change from previous years were the lighting conditions were changed by spot lights. Each robotic team has it's own robots and for these robots absolutely no modifications or additions to the robot hardware are allowed. No additional hardware is permitted including off-board sensing or processing systems. Only the use of alternative memory sticks in replacement of the supplied memory stick is a allowed. The robots can communicate thru wireless network but only with a bandwidth of up to 500kbps. A game controller agent is also used in order to control the robots state as long as the game state. In the figure 2.1.1 we can see an example of an SPL game where our team Kouretes is playing in the field.

So for a game each team must have 3 robots, one goal keeper and two field players. A game consists of two halfs; each half lasts for 10 minutes. Also a penalty kick shoot-out is used to determine the outcome of a tied game when an outcome is required. There are strict rules applied by referees during the game. If a robot does not ''obey'' these rules and performs and illegal action, is penalized and removed for some seconds from the field. Some of these illegal actions are about pushing other robots and grabbing the ball for an amount of time or enter its own goal area as a defender. These rules are modified and change every year in order to fulfill the special needs of a robotic soccer. So for the competiting teams having already solved many of the hardware problems by the robot

manufacturer, have to deal with software problems and focus on making their algorithms run in real-time on the embedded computer system of the robot.

Small Size League

A Small Size robot soccer game takes place between two teams of five robots each. The robot, must fit within an 180mm diameter circle and must be no higher than 15cm unless they use on-board vision. The robots play soccer on a green carpeted field 2.2 that is 4.9m long by 3.4m wide with an orange golf ball. Robots come in two types, those with local (on-board) vision and those with global vision. Global vision robots, the most common variety, use an overhead camera, which is attached to a camera bar located 4m above the playing surface. Local vision robots have their camera on-board. The vision information is either processed on-board the robot or is transmitted to off-field PC for processing. Typically, the off-field PC also performs most, if not all, of the processing required for coordination and control of the robots.

Figure 2.2: A game at the small size league

Middle Size League

In the Middle Size league, two teams of up to 5 robots play soccer on an 18x12 meter indoor field 2.1.1. Each robot is equipped with sensors and an on-board computer to analyze the current game situation and successfully play soccer. Through wireless communication they can establish inter-team cooperation and receive all referee commands. But no external intervention by humans is allowed, except for substitutions.

Figure 2.3: A game at the Middle Size league from RoboCup2009

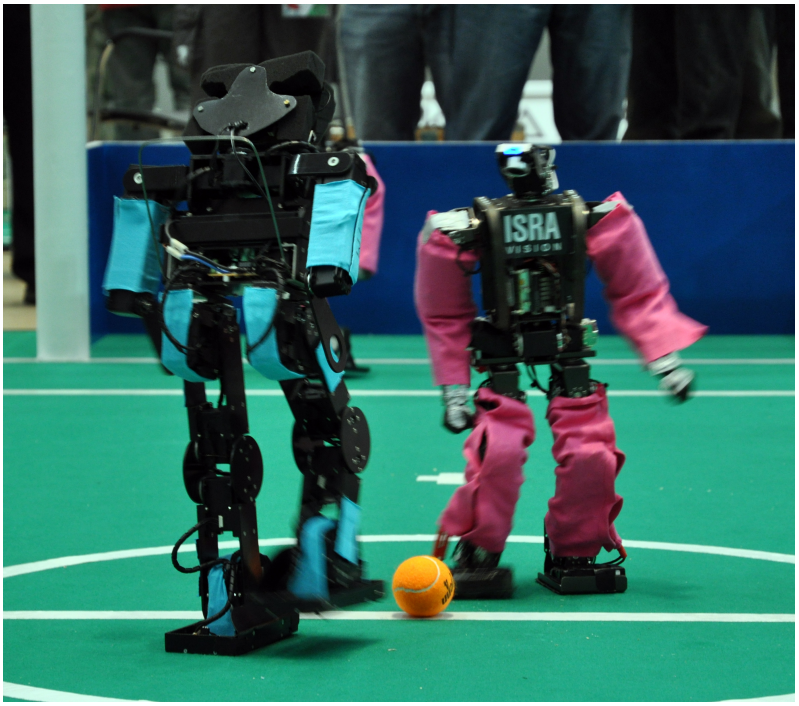

Figure 2.4: KidSize robots playing at the Humanoid league

Humanoid League

The Humanoid League is one of the most dynamically progressing leagues and the one closest to the 2050 goal. In this league, autonomous robots with a human-like body plan and human-like senses play soccer against each other. In addition to soccer competitions technical challenges take place. The robots are divided into two size classes: KidSize 2.1.1 (30-60cm height) and TeenSize (100-160cm height). Dynamic walking, running, and kicking the ball while maintaining balance, visual perception of the ball, other players, and the field, self-localization, and team play are among the many research issues investigated in this league.

Figure 2.5: A soccer game at the simulation.

Simulation League

Without the necessity to maintain any robot hardware, the RoboCup Simulation League focuses mostly on team strategy. In the 2D Simulation League two teams of eleven autonomous software programs (agents) each play soccer in a two-dimensional virtual soccer stadium represented by a central server, called SoccerServer. This server knows everything about the game, i.e. the current position of all players and the ball, the physics and so on. The game further relies on the communication between the server and each agent. On the one hand each player receives relative and noisy input of his virtual sensors (visual, acoustic, and physical) and may on the other hand perform some basic commands (like dashing, turning, or kicking) in order to influence its environment.

The grand challenge in the Simulation League is to determine in all possible world states (derived from the sensor input by calculating a sight on the world as absolute and noise-free as possible) the best possible action to execute. The 3D simulation competition increases the realism of the simulated environment used in other simulation leagues by adding an extra dimension and more complex physics. In 2008, the introduction of a Nao robot model to the simulation gave another perspective to the league, since researchers could test their algorithms and ideas before trying them on the real robots. The interest in the 3D simulation competition is growing fast and research is slowly getting back to the design and implementation of multi-agent higher-level behaviors based on solid low-level behavior architectures for realistic humanoid robot teams.

2.1.2 RoboCup Rescue

Disaster rescue is a challenging domain involving very large numbers of heterogeneous agents in a hostile environment 2.1.2. The intention of the RoboCupRescue division is to promote research and development in this significant domain on topics such as multi-agent team-work coordination, physical robotic agents for search and rescue, information infrastructures, personal digital assistants, standard simulator and decision support systems, evaluation benchmarks for rescue strategies and

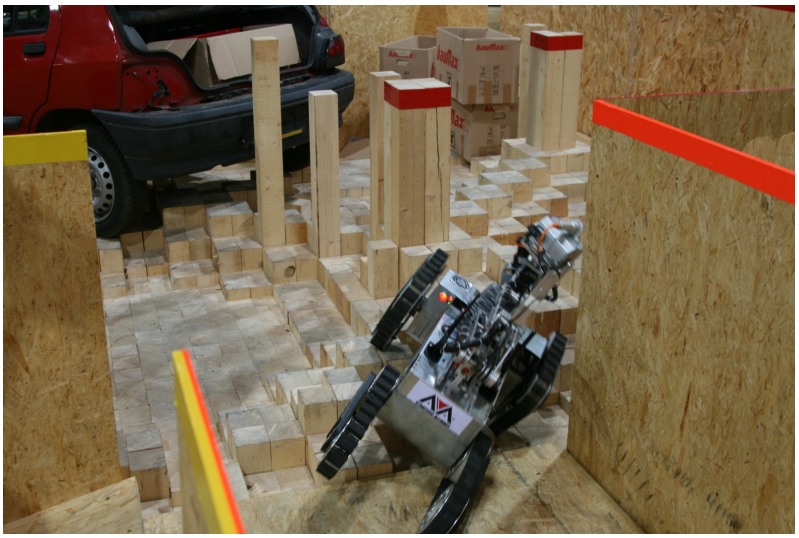

Figure 2.6: A robot in the RoboCup rescue artificial environment.

robotic systems, and integration into comprehensive system. RoboCup Rescue is divided into the robot and simulation leagues.

RoboCup Rescue Robot League

Robots explore a specially constructed disaster site, including mannequins with various signs of life, such as waving hands, shouting noises, and body heat, hidden amongst stairs, platforms and building rubble. The robots, some under human control, must find and approach the victims, identify their signs of life and produce a map of the site showing where the victims are located. The aim is to provide human rescuers with enough information to safely perform a rescue. Each team is scored based on the quality of its maps, the accuracy of the victim information, and the number of victims found.

RoboCup Rescue Simulation League

The league is composed of three competitions: the virtual robot competition, the agent competition, and the infrastructure competition.

In a virtual robot competition run, a team of simulated robots has to explore, map and clear a block-sized disaster area, featuring both carefully modeled indoor / outdoor environments. Robots and sensors used in this competition closely mirror platform and devices currently used in physical robots.

The Agent competition involves scoring competing agent coordination algorithms on different maps of the Robocup Rescue simulation platform. The challenge in this case involves developing coordination algorithms that will enable teams of Ambulances, Police Forces, and Fire Brigades to save as many civilians as possible and extinguish fires in a city where an earthquake has just happened.

The Infrastructure competition involves evaluating tools and simulators developed for the simulation platform and for simulating disaster management problems in general. Here, the intent is to build up realistic simulators and tools that could be used to enhance the basic Robocup Rescue simulator and expand upon it.

2.1.3 Robocup@Home

RoboCup@Home is a new league inside the RoboCup competitions that focuses on real-world applications and human-machine interaction with autonomous robots. The aim is to foster the development of useful robotic applications that can assist humans in everyday life.The ultimate scenario is the real world itself. To build up the required technologies gradually a basic home environment is provided as a general scenario. In Robocup 2009 the world was consisted of a living room and a kitchen.

2.1.4 Robocup Junior

RoboCup Junior or RCJ is a project-oriented educational initiative that supports local, regional, and international robotic events for young students up to the age of 19. It is designed to introduce RoboCup to primary and secondary school children, as well as undergraduates who do not have the resources to get involved in the senior leagues of RoboCup. The focus of RCJ is on education.

RCJ stands apart from other robotics programs for young students for several reasons. First, RCJ is focused more on education than competition. Second, the RCJ challenges remain the same from one year to the next, providing a scaffolded learning environment in which students can develop more sophisticated solutions as they grow and expand their knowledge. Third, the RCJ challenges -- soccer, rescue, and dance -- are familiar; spectators can watch and understand what they are observing, without needing explanations of complicated rules. Fourth, RCJ delves more deeply into computer science and programming due to its emphasis on autonomous robots. Fifth, RCJ sits at the entry-level of the international RoboCup initiative, which is strongly committed to education and involvement of young people in technology.

2.2 Kouretes Team

Kouretes is the first RoboCup team in Greece and was founded at the the Technical University of Crete back in 2006. The team is leaded by prof. Michail G. Lagoudakis and Prof. Nikolaos Vlassis and consist of mainly undergraduate students both from the Electronic and Computer Engineering department, and from department of Production Engineering and Management. The team participates in the SPL league and also in the simulation. The first participation of the team was in Robocup 2006 in Bremen, Germany when still the Four-Legged Sony AIBO robots were used and participated in the Technical Challenges. Next year the team participated also in the Four-Legged league of the RoboCup German Open 2007 competition in Hannover, Germany and ranked in the 7th/8th place.

Figure 2.7: Kouretes at RoboCup 2009, Graz, Austria

Months later the team's participation in the MSRS Simulation Challenge at RoboCup 2007 in Atlanta led to the placement of the team at the 2nd place worldwide. In the RoboCup 2008 in Suzhou, China team participated and won two trophies: 1st place in the SPL-MSRS league and 3rd place in the SPL-Nao league. In 2009 the team participated both at the RoboCup German Open 2009 competition in Hannover and in the RoboCup 2009 in Graz, Austria in the SPL league.

2.3 Nao® the Robot

Nao is an autonomous, programmable, medium-sized humanoid robot, developed by the French company Aldebaran Robotics, a start-up headquartered in Paris, after 3 years of research[1]. The first version ''V1'' of the robot was deployed only for academic use and with special offers for robocup teams in the first quarter of 2008. Some months later a better version of the Robot, named ''V2'' replaced the too fragile first version. In Robocup 2008 that version was used however there were still many weakness to the robot structure leading to many broken robots. Eventually ''V3'' version that was released in first quarter of 2009 and the company replaced the old version, overcame many of the primary problems. The new robots were very solid and strong and were used to all competitions in 2009.

Nao is a unique combination of hardware and software in a modern design. Nao stands tall in all points amongst its robotic brethren. The hardware has been built from the ground up with the latest technologies and features a large number of precise actuators and wide range of sensors. Nao is a

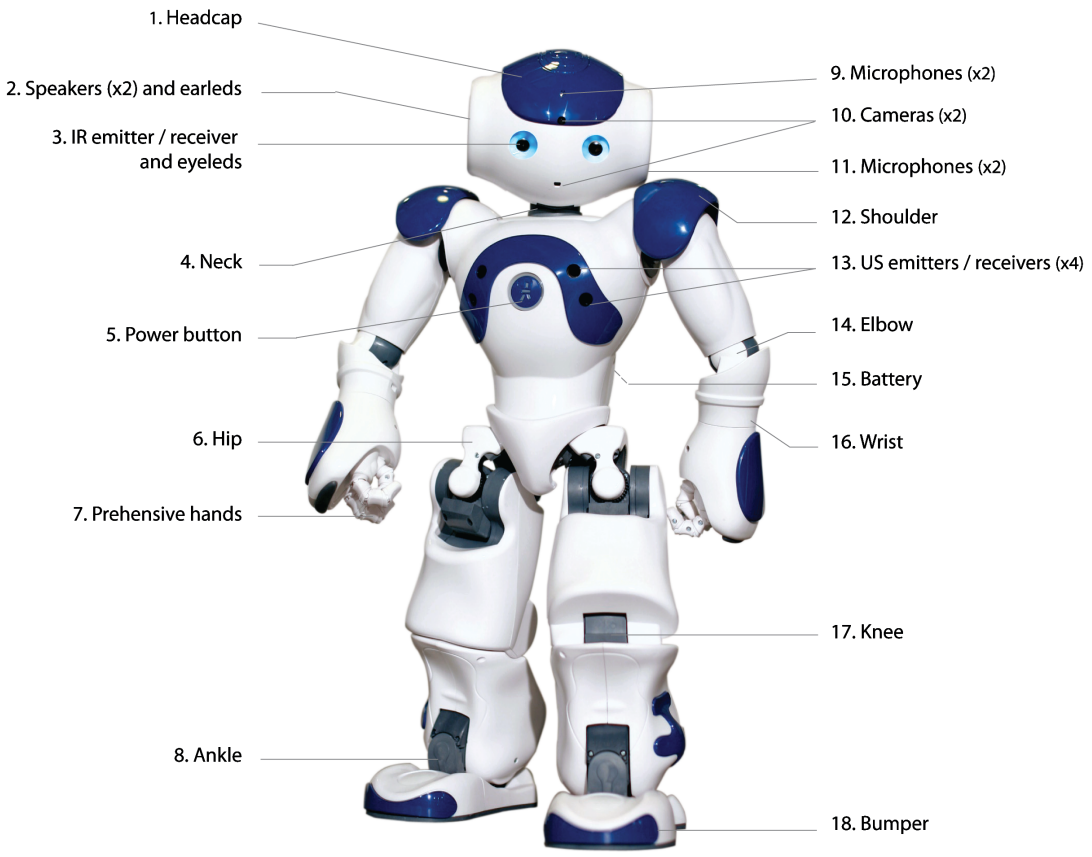

Figure 2.8: Nao V3 Academic Edition

58 cm (23") tall biped robot and weights about 4.3 kgs. The limited Robocup edition has 21 degrees of freedom. It carries a tiny computer using a x86 AMD Geode 500MHz CPU with 256 mb SDRAM and for storage a 2 GB usb flash memory module. The OS that comes with is Embedded Linux (32 bit x86 ELF) OS using a custom OpenEmbedded based distribution. The robot is powered by a 6-cell lithium-Ion battery pack giving about 90 minutes of power. For connectivity both wired and wireless adapters are provided.

Nao is full of sensors. In the head of the nao there are two CMOS digital cameras mounted. Their maximum resolution is 640x480 at a frame rate of 30 fps. Also 2 speakers and 2 microphones are mounted at the ears of the robot. Additionally 4 ultrasound sensors exist in the front side of the chest in order to detect obstacles. There are also a 2 axis gyrometer and a 3 axis accelerometer in the chest of the robot. In each foot there is an array of force sensitive resistors giving information about the pressure that is applied to the feet. Bumper at the end of the feet are provided to detect small collisions. Besides all these there are 21 strong variable force motors provide enough power in order to make the robot move naturally. In each of these joints there are position sensors for tracking the exact position of the joint.

Nao can be programmed and controlled using Linux, Windows or Mac OS and comes with complete

software and documentation. Apart from the robot hardware, Aldebaran robotics provides a programming framework, called NaoQi. This is also a middleware that runs on the robot and controls the low level systems and also making safety checks. Naoqi provides low level API to access the robotic hardware, control the joint and retrieve real time time information from the sensors. In addition there is high level API which allows users to make Nao walk and balance. Naoqi is a distributed environment witch allows parallel, sequential or event driven execution.

2.4 Probabilistic Robotics

In modern robotics probabilistic methods are used to reduce the uncertainty of complex non-linear systems. State estimation methods are used to discover the distribution that describes the position of the robot in the environment. As state we assume the triplet (x, y, θ) where x, y are the coordinates in the field and θ is the orientation of the robot. That state changes as the robot moves and must be updated and corrected. The Bayesean approach provides the mathematical tools needed to address that problem. More specifically, the Bayes filtering can solve the problem of estimating the state of the system using sensor measurements in an optimum way. The concept behind Bayesean filtering is to estimate recursively the probability of the state using date collect by the sensors and information about the actions taken by the robot that change the state on purpose. The filtering is performed at each time step using data about the last actions performed a_t and the most recent observation z_t. The posterior at a time t is called belief and is defined by:

$$Bel(x_t) = p(x_t \mid a_{t-1}, z_t, \ldots . a_0, z_0)$$

We assume that the dynamic system is Markovian, the observations and the actions are independent of past measurements and actions given the current state. Using this assumption the posterior can be found following two rules. A prediction rule and an update rule. When the robot performs an action the state of the system is predicted recursively according to

$$Bel(x_t) = \int p(x_t \mid x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

whereas an observation z_t is made, the state is updated according to

$$Bel(x_t) = a \int p(z_t \mid x_t) Bel(x_t) dx_t$$

where a is a normalizing constant that makes the belief to sum up to one. The Bayesian filter is an abstract concept and provides a probabilistic framework for recursive state estimation. To implement Bayes filters, one has to specify the probability distributions $p(x_t \mid x_{t-1}, a_{t-1})$ and $p(z_t \mid x_t)$ to describe the stochastic transition and observation model of the system.

2.4.1 Particles Filter

A sample-based approach is to represent belief by sets of samples or particles. The key advantage of particle filters is their ability to represent arbitrary probability densities. These types of filters have shown to converge to the true posterior even in non-Gaussian, non-linear dynamic systems. They are very efficcient since they focus their resources, the particles, on regions with high likelihood. In the figure 2.9 is a particle representation used by particle filters [2]. The lower right graph shows samples drawn from a Gaussian random variable, X. These samples are passed through the nonlinear function shown in the upper right graph. The resulting samples (particles) are distributed according to the random variable Y. More specific the belief $Bel(x_t)$ is represented by a set S_t of N weighted samples distributed according to $Bel(x_t)$:

$$S_t = \{\langle x_t^{(i)}, w_t^{(i)} \rangle \mid i = 1, \ldots, N\}$$

Each x_t^i is the state and the w_t^i is the non-negative importance weight, witch sum up to one. The basic form of the particle filter realize the recursive bayes filter according to a sampling procedure refered as Sampling Importance Resampling (SIR)[3].

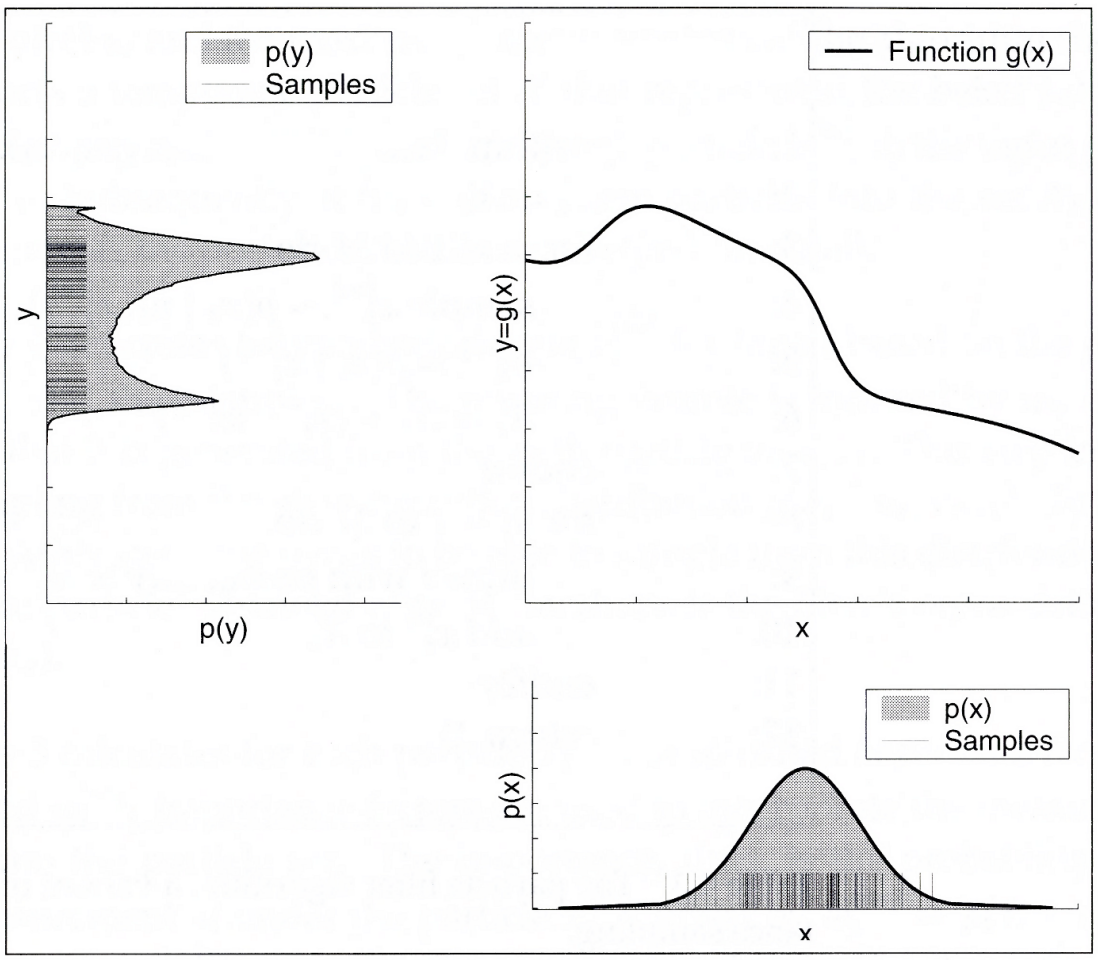

Figure 2.9: The ‘‘particle’’ representation used by particle filters.

Chapter 3

Problem statement

3.1 Localization Types

Perception is the process by which a robot maps sensor measurements into internal representations of the environment. Localization is one of the most pervasive perception problems in robotics. The localization problem can be categorized in three categories of increasing difficulty. First is the tracking problem, where the initial position of the robot is known and the goal is to track the position of the robot as it moves. More difficult is the global localization problem in which the initial robot position is unknown. Finally, the most difficult, but also common in Robocup, is the kidnapping problem, where the robot may be removed and replaced at a random position without any notice and without being informed of the relocation.

The robot has to localize itself using sensor data. In our problem sensor data consist of information about landmarks of the environment. Sensor data represents the relative distance and angle the robots camera observes a landmark. Using this info and the exact position of the landmarks in the field we try to localize the robot.

3.2 The need of Localization in SPL

In the SPL league robots can play without localization, as long as they do not collide with other robots. In the latest Robocup games there were good teams that they were scoring goals easily using the simple rule: *align ball to the opponent's goal and kick*. But, in practice such simple approaches cannot lead to effective play, as there are more problems that must be solved.

First, the robots must not leave the field, because, when they do so, they are penalized for 30 seconds and are removed from the field. In addition, the robots must not enter their own goal areas, except for the goalie defending the team's goal. When other defending robots enter the area, according to the Illegal Defender Rule, they are penalized and removed from the field for 30 seconds. Also, the goalie must not leave the goal area, because he is penalized in the same way. This penalty is

called the ''Standard Removal Penalty'' and after these 30 seconds pass, an assistant referee places the robot back in the field at the halfway line as close to the sideline as possible on the side of the field farthest from the ball with the robot facing the opposite sideline. So, if a goalie is penalized for some reason, he will be placed away from his goal and he has to find a way to return to its goal as soon as possible. It becomes clear that a robot without localization will be penalized most of the time. Furthermore, a penalized goalie will be unable to return to his goal without localization.

Second, there are two ways for initial robot placement in the field before kick-off: autonomous placement and manual placement. In the autonomous placement the robots have a maximum of 30 seconds to reach their initial position on their own. The robots that are autonomously placed in legal positions have the opportunity to be much closer to the ball resulting in a significant advantage over manually placed robots. Apparently, autonomous placement cannot succeed without localization.

Third, having accurate localization gives a team the ability to develop more complex strategies and use formations through comminucation of their current position at all times. Skills, such as passing the ball to a teammate or kicking towards the opponent goal, are greatly facilitated in this case, as there is no need for direct observation of the teammate or the opponent goal. Teams without localization are doomed to using simple, non-sophisticated team strategies.

3.3 Environment Specification

In the robotic soccer, the robots are playing soccer in specialized soccer fields. The SPL field is 4m wide and 6m long. There are two goals in the field, one in yellow color and one in sky blue color with known dimensions (width 1.4m and height 0.8m). In addition, field lines exist in the field and they represent some areas of special interest. Apart from the field limits rectangle, there are two small goal areas one in front of each goal, one center circle, the middle line, and two spot (penalty kick) points, all with known positions and dimensions in mm as shown in Figure 3.3.

A robot is successfuly localized in the SPL field if it knows its coordinates (x, y) and its orientation θ. Therefore, the variables we need to track for localization are (x, y, θ). Looking the field from above we define as the coordinate center point $(0, 0)$ the center of the center circle of the field, as axis x the straight line that passes through the centers of the goal with positive side the side of the yellow goal, as axis y the vertical line that passes longwise the middle line, and as orientation $\theta = 0$ when the robot's chest is facing the yellow goal. This coordinate system is shown in Figure 3.3.

3.4 Probabilistic Localization Essential Ingredients

In order to develop a probabilistic approach to solve the localization problem, we have to derive an *observation model* and a *motion model*. The observation model will provide the distribution $p(z \mid \bar{x})$ of the observation z with respect to the current location of the robot $\bar{x} = (x, y, \theta)$. The motion model will provide the distribution $p(\bar{x}_t \mid \bar{x}_{t-1}, a)$ that describes the resulting location $\bar{x}_t = (x_t, y_t, \theta_t)$ of the robot, with respect to the action a executed in the current location $\bar{x}_{t-1}$.

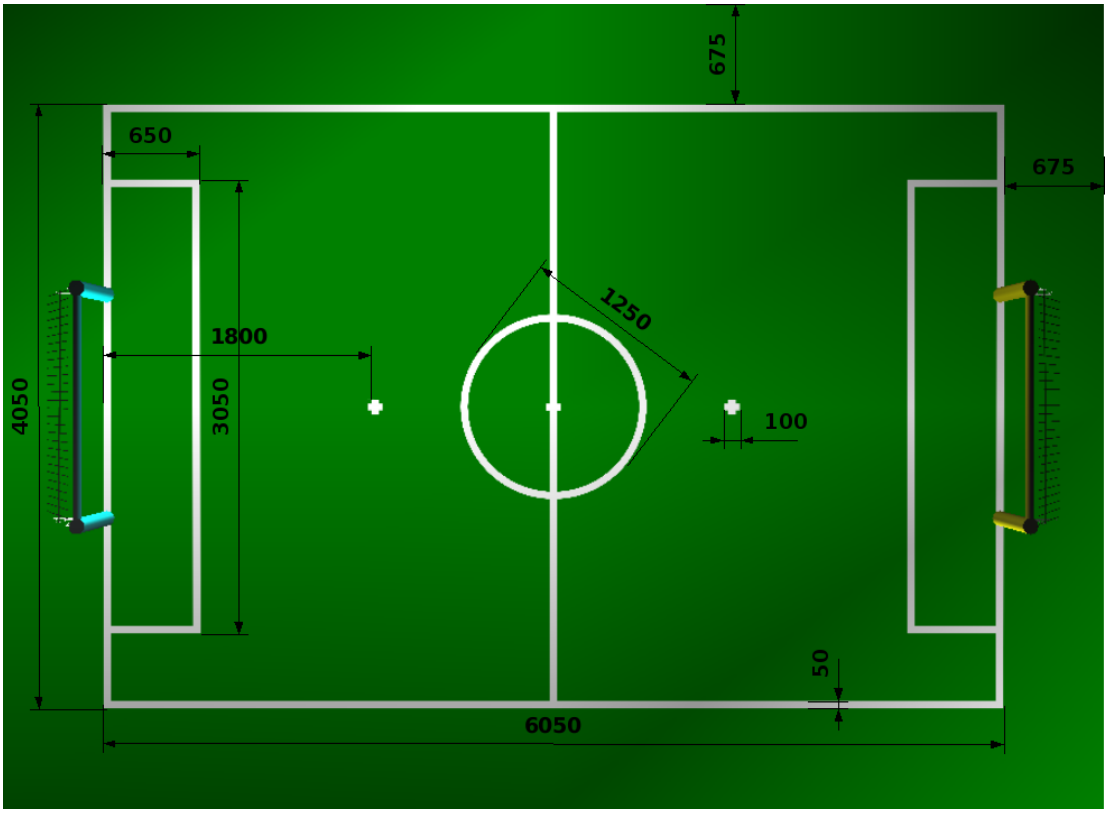

Figure 3.1: Scale diagram of the entire SPL field.

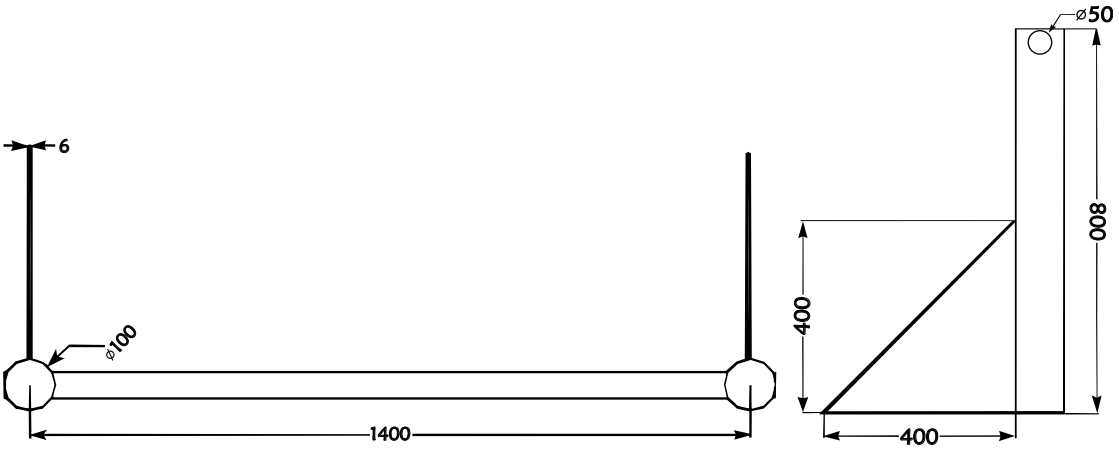

Figure 3.2: Dimensions of the SPL goals (top and side view).

3.5 Our problem

Our goal is to find a sound method for localization in the SPL field. Towards this end, we divide the problem in three stages:

- *Object recognition and motion library*. Even though this is not directly related to localization,

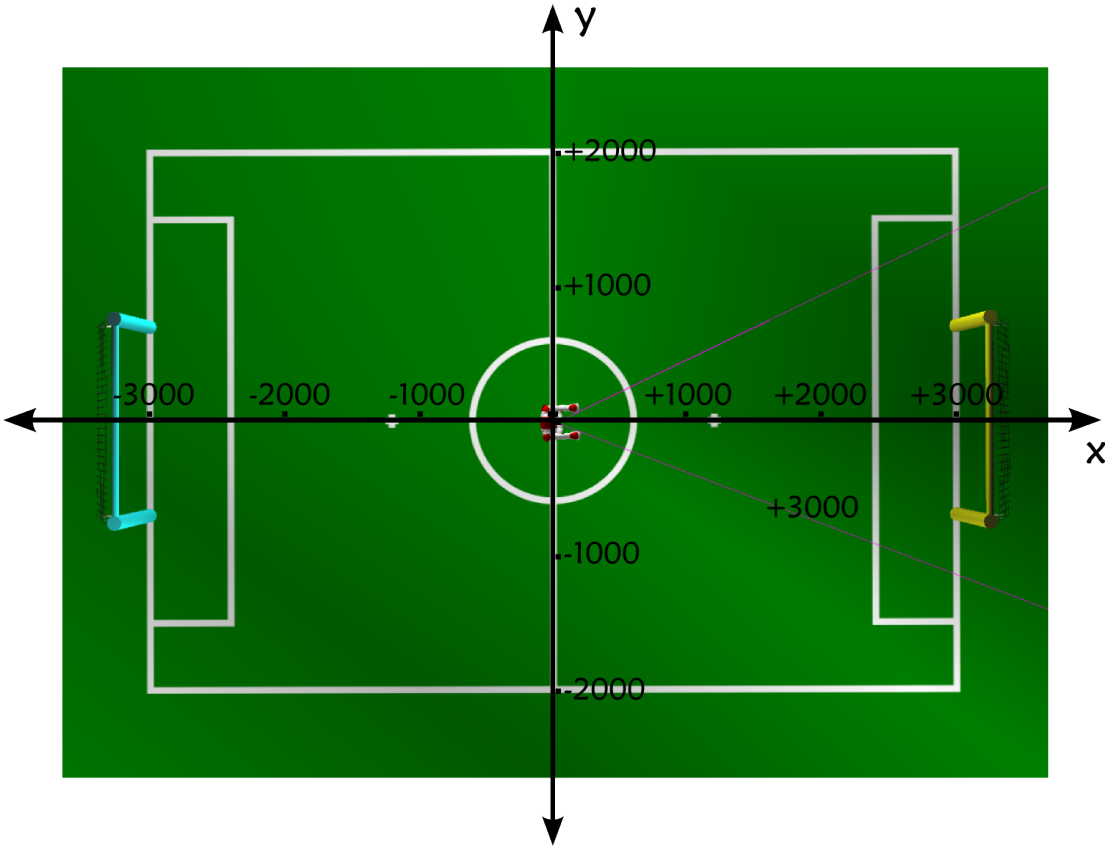

Figure 3.3: Our Field Coordinate system.

we need at least a basic object recognition module and a library of basic locomotion actions. The object recognition module process the camera images and returns the type of landmark recognized and the estimated distance and bearing to that landmark. The motion library needs to include motions for walking forward and for turning left and right.

- *Extraction of observation and motion models*. Probabilistic approaches to localization need these two models, which depend on the abilities and the accuracy of the object recognition model and the motion library, but also on robot and environment characteristics (surface frictions, area illumination, motor strength, etc.). Therefore, we need a systematic way of extracting the observation and motion model in any circumstance.
- *Localization*. This is the core problem of tracking the location of the robot (x, y, θ) at all times given the actions executed and the observations perceived and using the observation and motion models.

3.6 Related Work

All around the world there are many publications about self-localization. We will focus mainly in the Robocup community were the problem that has to be solved is the same as ours. This year in

the SPL league at RoboCup 2009, there were 25 teams participating many of them participating for their first time. Some teams were having localization techniques in their attempt to play effectively soccer. Regarding their team description papers team use many localization techniques and different approaches. Basically most of the teams use particle filters and a smaller group uses kalman filters. Bellow we will see the Localization approaches of some of the teams:

- **TT-UT Austin Villa 2009** [**?**] from the University of Texas at Austin. They have used Monte Carlo Localization technique with Negative Information. Also they can recognize field lines and use them in addition to the other features of the field. The negative information is used to update the particles even if an observation of a landmark is expected but not seen. This is very useful in the field environment were the number of features is small. To improve this this year have developed a line observation technique and incorporated line observations into the algorithm using the distance and heading to the nearest point of the line.
- **Nao Devils**[**?**] from the Robotics Research Institute, TU Dortmund University, Germany. This team took the second place at the Robocup 2009 competition in the SPL league. They have developed a vision-based Monte Carlo localization system and an approach to detect field features without using artificial landmarks. They made use of cooperative world modeling and localization on concepts based on multi-robot SLAM. In addition the use probabilistic physics simulation and estimation to improve the robot's state estimation and odometry information.

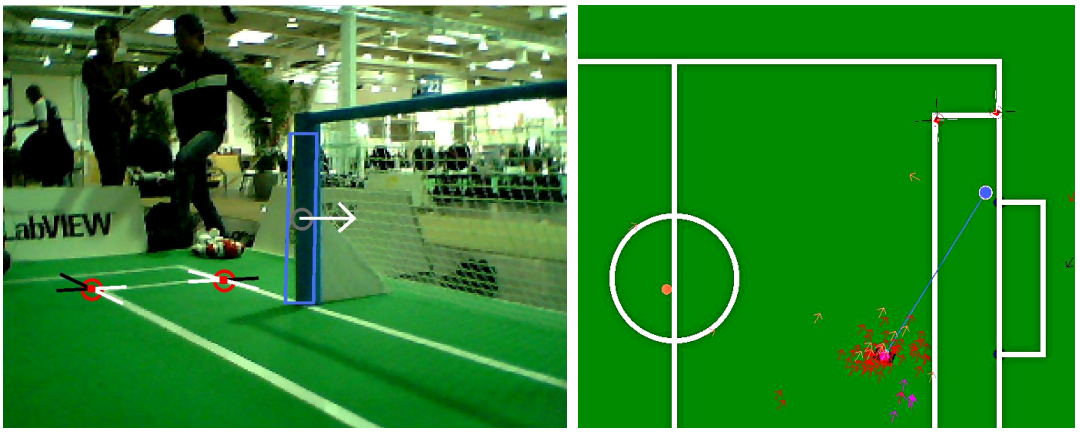

Figure 3.4: Nao Team Devils Observation and Particle filter Localization

- **TJark**[**?**] from the Robot and Intelligent System Laboratory Tongji University. Their localization part is based on Monte Carlo Localization and Kalman filter. The self-locator of robot is based on Monte Carlo Localization, while the ball locator is based on Kalman filter. For MCL method, the support that it's more robust to observation error and systematic error. One of their research on localization is how to adopt lines information into their MCL self locator, including the line crossings and line fragments. However, objects such as line crossings are actually non-unique, and bring a matching problem. They have tried several approaches, and one of them is to apply a Unique Landmark Memory Map(ULMM). They try thru historical observation information to discriminate the non-unique landmarks. The basic idea of this approach is that the robot position and the selected crossing under certain observation should

match with the short-term unique landmarks observation memory. They are implementing this approach through two ways: one is to use objects relationship and select the correct crossings by a decision tree; another is a Bayesian evaluation, selecting the best crossing to maximum a possibility function. Currently they are still working on evaluating which method is better.

- **Cerberus 2009**[**?**] from the Department of Computer Engineering, Turkey. This employs a vision based Monte Carlo Localization (MCL) system. They use MCL with a set of practical extensions (X-MCL). The first extension is to overcome problems and compensate for errors in sensor readings and is using inter-percept distance as a similarity measure in addition to the distances and orientations of individual percepts. Another extension they use is the number of perceived objects to adjust confidences of particles. They calculate the confidence that is reduced when the number of perceived objects is small and increased when the number of percepts is high. Since the overall confidence of a particle is calculated as the multiplication of likelihoods of individual perceptions, this adjustment prevents a particle from being assigned with a smaller confidence value calculated from a cascade of highly confident perceptions where a single perception with lower confidence would have a higher confidence value. In addition to that their third extension is related with the resampling phase where the number of particles in successor sample set is determined proportional to the last calculated confidence of the estimated pose. Finally, their window size in which the particles are spread into is inversely proportional to the confidence of estimated pose 3.6.

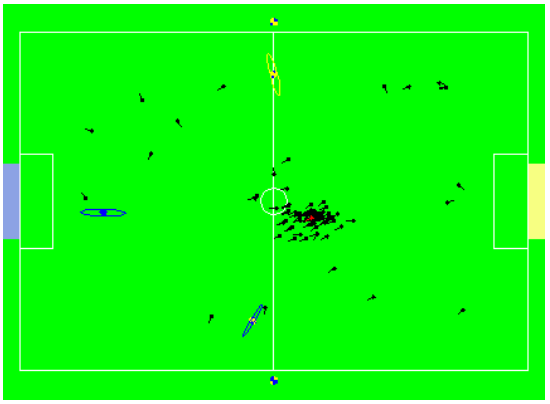

Figure 3.5: Belief state of the robot in Cerberus MCL algorithm

- **B-Human**[**?**] from the University of Bremen, Germany. Thats the winner team in the SPL league at the RoboCup 2009. Comming from the humanoid league and with a lot of experience they have made the most complete system among the teams. For self-localization, B-Human uses a vision-based particle filter based on the Monte Carlo method as it is a proven approach to provide accurate results in such an environment. Additionally, their algorithm is able to deal with the kidnaped robot problem, which often occurs in RoboCup scenarios. For a faster re-establishment of a reasonable position estimate after a kidnaping, they have implemented the Augmented MCL approach. In the figure 3.6 is an illustration of their particle-based representation of the robot pose and the ball position and velocity. Each gray box denotes one possible pose of the robot; light boxes are more likely than dark boxes. The black box shows

the resulting robot pose from the filter. The dots with arrows describe the ball samples (describing position and velocity of a ball). The orange one is the resulting ball estimate.

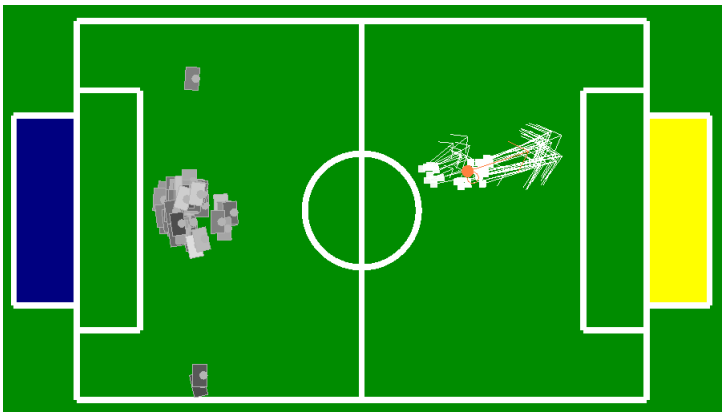

Figure 3.6: B-Human particle filters approach

- **SPQR** [**?**]from Department of Computer and System Sciences University of Rome "La Sapienza'', Italy. Their approach to localization uses a probabilistic technique based on particle filters. They have compared different solutions based on particle filters, investigating the use of two different strategies: the well known Sample Importance Resampling (SIR) filter, and the Auxiliary Variable Particle filter (APF). As a result of the experiments performed, they have detected situations where one strategy is better than the other as well as hints about the use of sensor resetting, that is common in this kind of implementations.

  For the Nao robots, they have integrated in the localization technique information about the game state (or in general about the task state), aiming at choosing the localization strategy and parameter setting that are more suitable for the current situation. Their Localization is based on perception of known landmarks: goal poles, lines and corners, for which different sensor models are used in the particle filter implementation. Moreover they are developing more sophisticated mechanism in proposal selection of Particle Filter algorithm that use the result from a Kalman Filter. The aim of this is to use a more informative distribution rather than approaches from previous years.

- NAO-Team Humboldt 2009 [**?**]the RoboCup NAO Team of Humboldt-University of Berlin. Their work on localization is quite interesting. They have developed two different methods for self localization: monte-carlo localization based on particle filter and a constraint self-localization method. In the first method they have use a random set of particles in the field to describe the robot's position. Then they update the particles based on distance and angle measurements from recognized objects, after they resample the particles population. Next they calculate the largest cluster of particles and for that reason they draw a grid over the field and calculate the number of particles in each grid cell. From the cell with the highest amount of particles the average of the coordinates of particles is calculated resulting in the most possible robot's position estimation. In figure 3.6 we can see in the left image the greed over the particles estimation, and right the estimated robot's position. They noticed that the largest cluster jumps occasionally to adjacent cells. So If there other cells having nearly number of particles as the largest one then they draw again a new grid shifted by 50 percent

regarding the first one. Then they locate the largest cluster in both grids and calculate the average of the particles. This resulted in a dramatic reduction in the possibility of the largest cluster jumping between the grid cells.

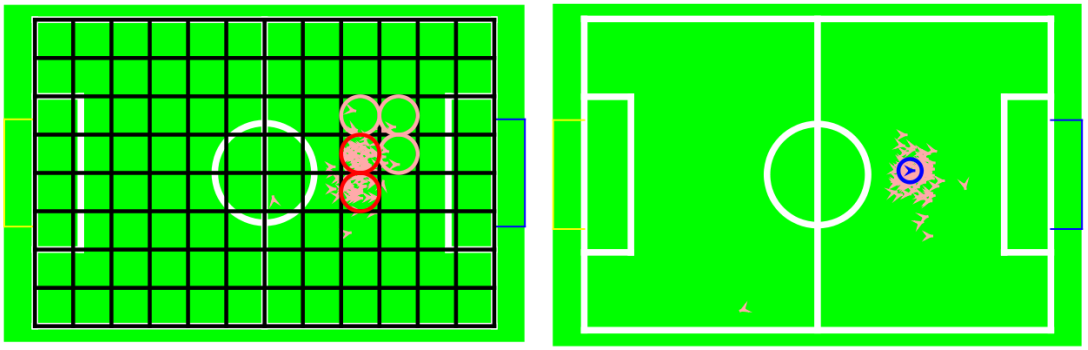

Figure 3.7: Team Humboldt's Particles Filter.

Their second method was a constraint based localization approach using instead of particles, constraint functions. They consider percepts from flags, goals and from lines. The shape of the constraint is determined by the kind of sensor data and expected sensor noise, which is dependent from the percept distance. After having generated all constraints, they propagate the constraints with each other as long as there are no more constraints or the resulting solution space becomes empty. The position belief of the robot is stored in form of constraints as well and propagated with the sensory constraints as well. If, for some steps, the belief doesn't fit to the sensory constraints, or even if no new sensory data are available, the belief constraint boarders are increased at first. If sensory data remain inconsistent, they reset the belief to the sensor data constraints. In figure 3.6 we can see the constraints, generated from percepts (pictures show the algorithm on an Aibo field). In the left a circular constraint as generated from flag percepts and right a line constraint, generated from line percepts.

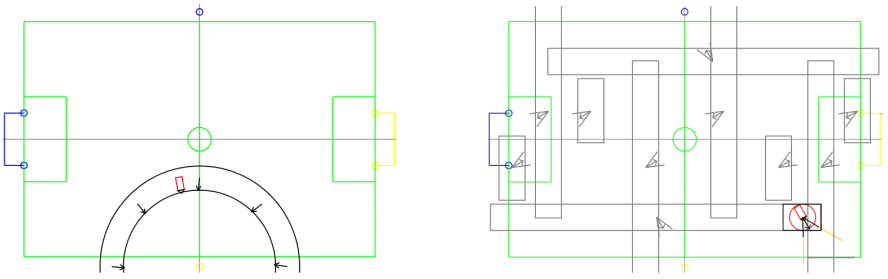

Figure 3.8: Team Humboldt's constraint based localization approach

- **The Northern Bites 2009** [**?**]from the Department of Computer Science Bowdoin College, Brunswick, USA. They are based on the bearings and distances to vision percepts, and their robots localize their position on the field using an Extended Kalman Filter (EKF), which models the robot's x, y position on the field, as well as the robot's heading. Though they experimented with a particle filter to improve our ability to localize from ambiguous data such as unidentified

line intersections, they found that the limited processor of the Nao was not fast enough to achieve the desired number of particles. In addition to modeling the robot's location, the EKF also models the position of the ball on the field, using both locally measured vision percepts, as well as teammate ball reports.

Chapter 4

The proposed approach

4.1 Motion Modeling

As the robot moves in the field, we must find a way to describe it's motion. Because of the uncertainty of the motion in the real world the position that the robot will be after it's motion is unknown. We came up with a probabilistic kinematic model, or motion model that will describe the transition model of the motion. The model describes the posterior distribution over kinematic states that a robot assume when executing a motion action. This distribution is described in our method by samples, the particles. So in simple words this model describes the way the particles will be after a motion, therefore the possible position of the robot.That model includes information about the distance the direction, and the orientation (ρ, θ, ϕ) at which the robot will be after a motion regarding with respect to the previous pose.

Because the robots locomotion actions can be almost omni-directional and wanting to reduce the number of different walking motions and to ease the problem we quantize these motions into steps. So in this problem a walking motion is described by its name (walk straight, turn, side step) and the number of steps that will be executed. Negative number of steps indicate motion in the opposite direction.

Instead of using a velocity motion model we selected the odometry motion model, because our robots locomotion velocity is slow. So in order to extract our odometry motion model several repetitions of a number of the above walking motions were performed. Initially the best way to extract the exact position of the robot before and after the motion is to use the robot in a simulated environment, in our case in Webots Robotic Simulator[4]. From the simulator it was possible to retrieve the exact position and orientation of the robot in the simulated robotic field. As long as we could retrieve initial and final position and orientation, distance and direction of the motion was extracted. The distance ρ was calculated using the Euclidean distance between these points and the direction ϕ using a variation of the arctangent function, the two-argument function $atan2$ to find the direction from one point to the other. The rotation θ was calculated by the minimum difference of the angles, before and after. Because of the uncertainty of the motion the results were noisy.

To model these distributions three basic common distributions were examined. A univariate Gaussian distribution with two parameters of mean and deviation, a mixture of Gaussian using four parameters and a Multivariate Gaussian distribution. Experimental results shown that the simplest univariate Gaussian distribution is enough to represent the robots motion model, quite well simple and fast.

So the motion model describes the distance, the direction, and the orientation after action relative to previous pose as following:

$$\begin{aligned} \hat{\rho} &= d\rho + N(M_\rho, \sigma_\rho) \\ \hat{\phi} &= d\phi + N(M_\phi, \sigma_\phi) \\ \hat{\theta} &= d\theta + N(M_\theta, \sigma_\theta) \end{aligned} \tag{4.1}$$

And a probable pose after that motion using the above is:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} x + \hat{\rho}\cos(\hat{\phi} + \theta) \\ y + \hat{\rho}\sin(\phi + \theta) \\ \hat{\theta} + \theta \end{bmatrix} \tag{4.2}$$

4.2 Observation Modeling

In many robotics applications, features correspond to distinct objects in the physical world. In robotics, those physical objects are also called landmarks. The most common model for processing landmarks assumes that the sensor can measure the range and the bearing of the landmark relative to the robot. The observation model describes the probability of a measurement to be accurate or not. Generally we try to model the error of the measurements.

In the beggining in the simulation we described the error of the measurements only by a deviation factor. The idea was simple. When we are close to a landmark the observation error is small and when we are away is big. The deviation was variable based on the distance from a landmark.

However when finally we had data from real observations the error could not described by this simple model. So in the end we modeled the error by fitting an line equation to the error in order to describe the systematic error and the absolute difference that the error had from that line we considered it as the deviation of the error.

4.3 Localization

We tried to solve the localization problem in the robocup field with the help of a probabilistic method. The Monte Carlo Localization method [2, 5--7] using particles was the one that we examined. Particles are being used to represent the probable position (x, y) and orientation (θ), or else in one word the pose (x, y, θ) of the robot inside the field. The basis of the method is to construct a sample based representation of the probability distributions function using particles. More specifically these

particles are being used as samples of an unknown distribution represent the probability distribution of the robot. This distribution is unknown, is non-Gaussian, basically because of our system's complexity and uncertainties. When an action is taken the pose of the robots is changing according to some model. Moreover when an observation arrives that constrains the possible pose of the robot in the field.

4.3.1 SIR Localization

The basic SIR algorithm [3] that was used is divided in three stages:

- **Prediction Stage**: In this stage we try to predict the probability distribution of the pose of the moving robot after a motion, modeling the effect of noise of the action. The particles are modified according to a model that describes the motion, including the addition of noise in order to simulate the effect of noise in the real world.
- **Update Stage**: Use of observation info to estimate the position of the robot after the motion. Basically weighting the population of the propagated particles using measured data about recognized features of the field extracted from camera images. This will result with particles with high weight in a position that is constraint by the measurements and will minimize the weight of the other particles.
- **Resampling**: In this stage we resample the particles population according to their weight. We draw with replacement particles from the distribution created by weighting the particles at the previous stage. The probability of drawing each particle is given by its importance weight. Resample transforms a particle set of the particles into another set of the same size, but with different distribution. Whereas before the particles were distributed according to a belief, after the resampling they are distributed according to a posterior belief.

Algorithm 1 SIR particle filter algorithm

Input: action a, observation z, a set of particles $S = [x_i, w_i : i = 1 \dots N]$
Output: updated particles $\hat{S}$

For $i = 1 \dots N$ { **Prediction** after motion a }
  $\hat{x_i} = predict(x_i, a)$
End For
For $i = 1 \dots N$ { **Update** using observation z }
  $w_i = Update(x_i, z)$
End For
$a = \frac{1}{\sum_{i=1}^{N} w_i}$ { **Normalize** weights }
For $i = 1 \dots N$
  $\hat{w_i} = a \cdot w_i$
End For
$Index = Resample(w_1, w_2 \dots w_N)$ { **Resample** particles population }
$\hat{S} = S(Index)$ { **Propagate** particles according to the index }

4.3.2 AUXiliary filter

Based on the simple SIR algorithm the auxiliary particle filter [8, 9] uses exactly the same stages but in a different way. The initial particles set is duplicated and is processed thru prediction and update stages. But in the third stage of resampling the algorithm resamples from the initially population using the importance weights of the second weighted population. As a result the initially particles population is filtered and cleaned from particles that would finally end up in regions of low probability, whereas regions of high probability are strengthened with more particles. After that the new particles population continues as input the SIR algorithm.

Algorithm 2 AUXiliary particle filter algorithm

Input: action a, observation z, a set of particles $S = [x_i, w_i : i = 1 \dots N]$
Output: updated particles $\hat{S}$
Local: $S' = [x'_i, w'_i : i = 1 \dots N]$ a second set of particles

For $i = 1 \dots N$ { **Prediction** after motion a }
  $x'_i = predict(x_i, a)$
End For
For $i = 1 \dots N$ { **Update** using observation z }
  $yw_i = Update(x'_i, z)$
End For
$a = \frac{1}{\sum_{i=1}^{N} w'_i}$ { **Normalize** weights }
For $i = 1 \dots N$
  $\hat{w'}_i = a \cdot yw_i$
End For
For $i = 1 \dots N$ { **Normalize** weights }
  $\hat{w'}_i = \frac{w'_i}{\sum_{i=1}^{N} w'_i}$
End For

$Index = Resample(w'_1, w'_2 \dots w'_N)$ { **Resample** weights from AUX particles }
$S_i = S_i(Index)$ { **Propagate** S set particles according to the index }

For $i = 1 \dots N$ { **Prediction** after motion a }
  $\hat{x_i} = predict(x_i, a)$
End For
For $i = 1 \dots N$ { **Update** using observation z }
  $w_i = Update(x_i, z)$
End For
$a = \frac{1}{\sum_{i=1}^{N} w_i}$ { **Normalize** weights }
For $i = 1 \dots N$
  $\hat{w_i} = a \cdot w_i$
End For

$Index = Resample(w_1, w_2 \dots w_N)$ { **Resample** particles population }
$\hat{S} = S(Index)$ { **Propagate** particles according to the index}

Prediction

So having the above motion model it was possible to predict the robots pose after some movements in the field. But as it can be seen it the figure . . . after some movements the uncertainty of the robots position grows unbounded. Even if the starting position could be known the end position will be finally unknown. Here is where the next stage of update comes.

Algorithm 3 Robot translation with noise

Input: action a, observation z, a set of particles $S = [p_i\{x, y, \phi\}, w_i : i = 1 \dots N]$

Motion Model
Translation distance $\rho(a), \rho(a)_{error}, \sigma(a)_{\rho error}$
Translation direction $\theta(a), \theta(a)_{error}, \sigma(a)_{\theta error}$
Rotation $\Phi(a), \Phi(a)_{error}, \sigma(a)_{\Phi error}$
Output: updated particles $\hat{S}$
For $i = 1 \dots N$ { For each particle }
$E_{trs} = \rho + N(\rho_{error}, \sigma_\rho)$
$E_{dir} = \theta + N(\theta error, \sigma_\theta)$
$x_i = x_i + \cos(E_{dir} + \phi_i) * E_{trs}$
$y_i = y_i + \sin(E_{dir} + \phi_i) * E_{trs}$
$\phi_i = \phi_i + \Phi + N(\Phi_{error}, \sigma_\Phi)$
$w_i = \frac{1}{N}$
End For

Update - Particles Weighting

In order to reduce the growing uncertainty of the particles, observations are used to evaluate each of the particles. In our approach observations of known landmarks in the robotic field are being used. As it will be described later the recognized field's landmarks are the blue and yellow goals of the field, whole or only their goal posts. The observations consist of information about the distance from a landmark and the angle, according to the robots chest direction, that was recognized. That angle is also known as bearing angle.

The basic idea behind of the update is that we compare the particles data with the observation data. The most a particle ''fits'' with the observation measurements the more probable it becomes to be the robots position. Even hear because of the noisy feature of the observation measurements we also model them and thats our observation model. To model the measurements noise two parameters were used to describe the deviation of the noise in distance and bearing respectively.

The weighting function consist of the product of three simple probability functions. In the first function the each particle is weighted according to the distance from a landmark. Assuming ρ is the measurement of distance between robot and landmark and σ_ρ is the deviation of the measurement, ρ_i is the distance the particle has from the same landmark. The function is:

$$P(x_i^{t+1} \mid x_i^t, \rho) = \frac{1}{\sqrt{2\pi\sigma_\rho}} e^{\frac{-(\rho_i - \rho)^2}{2\sigma_\rho^2}} \tag{4.3}$$

The second function we compare the bearing of the particles with the angle that was measured. The bearing of the particle θ_i is being calculated using the $atan2$ function and like above the function is:

$$P(x_i^{t+1} \mid x_i^t, \theta) = \frac{1}{\sqrt{2\pi\sigma_\theta}} e^{\frac{-(\theta_i-\theta)^2}{2\sigma_\theta^2}} \tag{4.4}$$

The third and last function that was used weight the particles according to the visibility of each landmark. Even if the field of view of the robot in the horizontal level is about 40 degrees the head of the robot is movable able to scan at a range of 240 degrees in a search for landmarks. But also there is a blank range that cannot be observed in the back of the robot. So for each particle we examine the visibility of a landmark in a position. With these probability we examine the event of being in some positions inside the field were some landmarks are not visible and also we can model the ability of the observation stage in recognizing landmarks.

Wanting to improve our filtering system we also use negative information filtering. We assume that if a observed landmark is not visible by a particle's pose then this particle probably is not correct. So we filter our particles additionally with the next probabilities:

$$\begin{gathered} V = Feature\ Visible \\ O = Feature\ Observed \\ P(O \mid \bar{V}),\ P(\bar{O} \mid \bar{V}) \\ P(O \mid V) = 1 - P(O \mid \bar{V}) \\ P(\bar{O} \mid V) = 1 - P(\bar{O} \mid \bar{V}) \end{gathered} \tag{4.5}$$

The final product know as the importance weight is being calculated using the above:

$$P(x_i^{t+1} \mid x_i^t, \rho, \theta, V) = \frac{1}{\sqrt{2\pi\sigma_\rho}} e^{\frac{-(\rho_i-\rho)^2}{2\sigma_\rho^2}} \frac{1}{\sqrt{2\pi\sigma_\theta}} e^{\frac{-(\theta_i-\theta)^2}{2\sigma_\theta^2}} P(O_{t+1} \mid V_{t+1}) \tag{4.6}$$

Resampling - Propagation

Having weighted the particles population it is necessary to wipe out the particles that have low weight and keep particles that fit better in the distribution, in order to be represent the distribution. In this procedure a new population of particles is being produced using samples from the original. The probability of choosing each particle it is given by it's weight. So the distribution of the particles changes from the original belief to the posterior $bel(x_{t+1}) = P(x_i^{t+1} \mid x_i^t, \rho, \theta, V) bel(x_t)$ after the resampling. The purpose of this procedure is to force the particles limited population to posterior belief. If resampling stage was missing the particles will finally end up in regions of low probability thus their population will not represent either approximate the posterior belief. The resulting particles population possesses many duplicates because the particles are drawn with replacement.

4.3.3 Other Techniques

Besides the basic algorithm that was used some helpfull techniques were also used.

Random Particles

When the algorithm accidentally discard all particles and moves them away from the correct pose then injecting random particles is one solution to overcome that problem. The number of the injected particles can be variable and also the deviation of the particles spread. Also in a kidnaping problem when the position of the robot is changed radically the particles population that was initially focused in a area near the previous position will eventually die because the observation will extirpate the particles weight. Then spreading some of the particles all over the field probable will reach a position when they will survive. So in the next update circle and resample we particles will grow near the correct position. Also with this technique with smaller spread noise is helping the system when the orientation of the particles is not correct. Witch mean that the particles will be propagated in a wrong direction and finally will lose their weight. But spreading some particles near and also putting some noise to the rotation angle solves that problem.

Particles from observation

Another technique to enrich the particles population with particles that most probable are in the correct position is to create particles that fit in the observation data. This is done in a simple way. A grid scan is deployed in the field. In each of the points of the grid for different orientation the weighting procedure is done. Then that particle goes in a ordered set. In the end the best of these particles are inserted into the particles population. This trick can also help in the kidnaping problem meaning that the particles population will grow near a good position.

4.4 Visual Feature Extraction - Observation

The basic sensing system of the Nao robot usable to the Observation is the imaging sensor or more common the head-mounted camera's. After image processing the camera sensing system is used for extracting range and bearing measurement of field's visual features. As visual features of the field are some physical and known object landmarks. The measures include info about the range and the bearing of a recognized landmark relative to the sensor, or in this case the robot itself. The landmarks that were used are the field's goal posts and the goals generally. The goals are painted blue or better sky-blue and yellow and can be easily separated from the environment.

Initially the camera is used to capture images from the environment. The camera sensor has small field of view 58 degrees diagonal and 46.4 degrees horizontal so the visible environment is limited. To overcome that problem and expand the range of the sensing system the robots head motion system has been used, able to turn 240 degrees from one to the other side. Thus the total horizontal

range of sensing became 286.4 degrees including the 46.4 horizontal field of view of the camera. In the head scanning procedure that was followed in each head pose captured images were processed.

Computer vision helped in the processing of camera images. Color segmentation method was used to segment feature colors from the environment. These were also the color segmentation method also used by Kouretes team in latest Robocup events. In order to have the better possible observation a color focusing method was used. At first an image is being captured using the naoqi framework to access the camera. After that image is scanned in a grid searching for a specific color. If the amount of pixels that have been found having that color is above a threshold then the center of these pixels is being calculated and also the divergence of them to the image center. Knowing the divergence of them to the image center horizontally and vertically a tracking method was used trying to turn the image in order to center the observed landmark. This focusing steps were repeated few times and in 3 to 5 repetitions the landmark was focused.

Having basically the goals focused in the camera image the next stage was started. In these stage measurements about the distance from a goal post and also the bearing from it are being extracted. The calculation of these measurement from the captured image is quite simple. The key idea behind the distance calculation method to use the triangle that is shaped beginning from the camera sensor in the head of the robot to the lower point of the goalpost end in the feet of the robot. In that triangle the robots feet are in the same level with the base of the goal posts. The height of the robot can be found in every configuration using naoqi framework. Then the angle between the robot's body and the straight line from the camera sensor to the base of the camera can be calculated using 2 angles. The known pitch angle of the robots head and the angle as a part of the camera's field of view. Because the first angle, the head pitch, is aligned with the center of the camera, it is essential to add the angle that forms in the captured image. This angle is a part of the vertical field of view and is calculated using the number of pixels from the end of the goalpost's base in divergence of image's center and translating this to an angle as saw in figure .... Then using trigonometry and specifically the tangent of the above angle and the robot's height it is straight forward to calculate the distance to the goalpost.

Experimental results initially in the webots simulation environment shown satisfactory results. Although theoretically the results should be accurate the

Chapter 5

Implementation

In theory algorithms seems to work but in practice they have to be implemented and also tested. For this purpose functions and classes were programmed. Being the robot the final destination of this implementation the programming language C++ was chosen being the optimum choice from the available of the Naoqi platforms SDK. In addition python scripts were also used to control the behavior of the robot and make remote calls to module's functions. For extracting statistical data and preliminary algorithms evaluation matlab code had also been written. This mathematic tool gave us the ability to visualize with plots the numerous variables and evaluate procedures.

In the beginning the localization attempt was written in matlab scripts. The algorithm was implemented quite fast and easy and gave the first positive results. The whole project was divided in two main parts. The Localization part and the Vision - features extraction part. For these parts modules were made for the naoqi platform having final goal to be executed inside the robot. For debugging purposes the modules were build in remote mode as separated executables. However the built mode can easily change and instead of executable, libraries can be built.

As modules are considered the Naoqi Modules. These are initially generated by a program called module generator provided by the Naoqi SDK. The module generator generates code structures and base c++ files needed to communicate with the naoqi. Module generator generates projects with code structures and base c++ files. From the generate project two possible binaries can derive: an executable that can connect the naoqi server, or a library that can link to the main executable (add it in autoload.ini). Module generator manages the links and paths, so that you programmer really focuses on the functionalities. In order the modules api to be accessible from the outside method must be bounded because only these methods can be called by another module.

Below will be explained the programming structure that was followed.

5.1 Localization

A basic class was generated to include the methods needed for the localization. That class was named KLocalization. Firstly some helping structures were also made in order to organize the amount of

variable needed by the MCL algorithms.

A structure was made to keep a particles data. It consist of four double variable two for the x, y position the orientation of the robot and on last to store the weight of the particle. A compare operator was made to make possible to compare the particles based on their weight, something that is needed by other methods. But because of the particles are usually processed in vectors or only the weight's of the set is needed, for example in the resample stage, another structure was created having only pointers to double arrays and one integer to keep the size of the arrays. These structures are name partcl and parts respectively. Also a belief structure was made having almost the same data but for other purpose.

For the features of the environment a structure called also feature was made to storage the position of a landmark as two doubles and it's signature as a string.

Furthermore some other structures were made for keeping the models. A basic usefull type name RandVar was made to help the organization of statistical data. It includes three double variable, one to store a value and the other two describing its mean error and the deviation of this error respectively.

For the Motion model a structure was made (KMotionModel) having a sting name an integer about the number of robots steps and three randvar variable named Distance Direction and Rotation. Lastly the Observation model held in a structure (KObservationModel) with one feature type variable for the recognized landmark and two randvar variable for the Distance and the Bearing.

The KLocalization class also has variables. The basic are the Particles population and a second one for the auxiliary particle filter. For keeping all the motion modeling data a map of string to vector of KObservationModel structures was used. Another map for the Features was used with string to KObservationModel structures pairs. Strings were used as a signatures and also as keys to make easier the accessing to models data.

The Whole localization system consists of several functions. We will try to focus more into the localization circle functions but also explain and describe the purpose and the use of the other functions.

Looking from the beginning an initialization function programmed to initialize any Klocalization instance. In that function several initialization where made. Here was where the features of the environment were pushed into the features map. Also the particles were initialized with initial positions uniformly distributed in the field. For extracting random variable the newran library was used.

5.1.1 Motion Model in XML

Furthermore the motion model data were loaded using another function. That function (LoadMotionModelXML) reads from an xml the statistical data and loads them to the map for the motion model. The tinyxml library was used to load the data. Additionally for these loading process a small number of functions were made and they exist in the Kxmlutil.h file. These are basically tinyxml

example ...ref ...functions for reading elements and attributes using tinyxml functions. Some of them were changed to extract randvar structures from the attributes of the xml elements.

The xml file must have the following structure.

```
<?xml version="1.0" ?>
<Walk>Str
    <NumOFSteps_5>
        <Distance Val="30" Emean="4.1245" EDev="0.0180" />
                <Direction Val="0.0" Emean="-0.0086" EDev="0.0006" />
                <Rotation Val="0.0" Emean="-0.0006" EDev="0.0003" />
    </NumOFSteps_5>
</Walk>
<Turn>Turn Motion
    <NumOFSteps_-1>
        <Distance Val="1.0" Emean="-0.1224" EDev="0.6327" />
                <Direction Val="-4.0" Emean="-0.0665" EDev="0.0502" />
                <Rotation Val="0.0" Emean="0.2028" EDev="0.0006" />
    </NumOFSteps_-1>
</Turn>
```

Listing 5.1: KMotionModel stored in XML. In these file only two motions are shown. The first indicates forward motion of five steps and the other right turn of one step.

The basic motions are three, the Walk the Turn and the side steps. For a different number of steps a motion model must be statistically calculated from experiments. Then it should be stored to the above xml file. These xml file give as the ability to store and easily load motion modeling data about different surface the robot can walk.

Ending with initialization phase we will continue with the basic SIR functions and in the meantime other useful functions will be described.

The first stage of the filter is the prediction. A function called predict was made and have as inputs a parts structure reference and a KMotionModel structure. In this function in a for loop every particle is moved according to the motion model. More specific two temporary variables keep the value of distance and direction for the motion. In these variables gaussian noise is added according to the motion model. Then the coordinates of the particle are changed to the old one adding the noise. Also the in the rotation of the particles is added the respective noise.

In the next stage the Update function was implemented to weight the particles according to the observation. This function has as input a parts structure reference and a vector of KObservationModel structures. Here the weight is calculated for its particle in a for loop. In the beginning a check is been done for the existence of observations. If no observation exists then these particle will be only weighted according to the visibility of a landmark or not.

5.1.2 Update particles Weights

If there are observations then the update is done in the following way: First the overall weight of the particle is one. Then a loop iterates in the observation vector and for each iteration the probability is

calculated and multiplied to the overall weight. First the visibility of the examined landmark by the particle is checked and a ratio variable is set. If the landmark is visible by the particle the ratio takes as value the probability of being observable if a landmark is visible $P(O \mid V)$ or else the $P(O \mid \bar{V})$. After the distance of the particle is calculated. This value is used then to calculate the

$$P(x_i^{t+1} \mid x_i^t, \rho) = \frac{1}{\sqrt{2\pi\sigma_\rho}} e^{\frac{-(\rho_i - \rho)^2}{2\sigma_\rho^2}} \tag{5.1}$$

probability of being in that position. A function named normpdf was implemented to calculate that probability. Next the bearing of the particle is calculated and simiraly the probability of the particles bearing to the observation bearing is calculated and multiplied to the overall weight. In the end the overall weight is set to the particle's weight variable.

5.1.3 Importance Sampling

The resampling algorithm need the overall weight of the particles to sum to one. Thus a normalization function was made in witch the overall sum is calculated and then each particle weight is divide to the sum to be normalized. In the case where the particles are having zero weight these function returns false value else true. That event of zero weight in the particles means that after the update stage the particles are having to little probability and they can not represent the distribution. These is usually caused in an kidnapping event were the position of the robot is changed and will be discussed latter.

Continuing the third stage the resample stage was implemented in two parts, two functions. With the first function an index of the particles that should be propagated was produced. To produce that index an multinomiaR name function implemented for that scope. The initially code was written by Arnaud Doucet and Nando de Freitas in matlab script we ported that code to c++. Because the initial script was in matlab some other functions were also made to replace the matlab ones. These included a function (CumSum) that calculated the cumulative sum of a table and another (FlipCumProd) to calculate the cumulative product and return it with reverse order. More specifically the index is an array that keep the index number of the particle that will be copied in each position.

The produced index then is used as input by another function (Propagate) as well as the particles set. Here firstly the particles data are copied to temporary vectors. Then, in a for loop, using the index as guide, values are copied particle's info about the propagated set is replacing the original set.

Besides these function other other function were made: A function to spread the particles was made in order to spread some particles around the distribution because the resampling stage tries to focuses in a spot. This function takes as inputs a reference to partcls structure two double variables with the deviation that we want to spread the particles one for the position spread and another one for the rotation and at last one integer value for the percent of the particles that will be spread.

Another one function was made to generate particles that fit in observation data. Since some times the number of observations is above three witch means that probable the position where the robot could easily and singularly be found. This function looks a like the update function because generally does the same weighting procedure but not in a particles population. The idea of generating new

particles is to put a particle in different positions in the field and weight it. So like a grid, on the field, a double for loop sets a particle is different positions. Then the particle is weighted regarding its positions as in the update stage using the For the bearing another for loop sets the particles orientation to different values. For each of these values the probability using the bearing weighting procedure is checked. The most probable orientation then is being chosen and set to the particles. After that the overall weight calculated and set to the particle weight value. This temporary particle then is pushed to a particles priority queue. These was feasible because a comparator was implemented for the particle structure. Moreover if the queue exceeds a specific size then the particle with the smaller weight is popped out. At the end we have a queue with the best possible particles. These particles later are inserted to the original set replacing randomly particles from the original set.

The Basic function that uses all the above functions directly or indirectly had was the LocalizationStep function. That function had as inputs the number of steps the string name of a motion and a vector with observations. For the selection of the MotionModel that will be chosen another function was made to search and finally select the best existing model for the searching string and number of steps. The search was done first looking for that string as a key to the motion models map and if found then searched into the included vectors for a motion model with the exact number of steps. If the specific number of steps no model can be found then the closest one is chosen and the number of iterations if needed will be calculated.

After the selection of the motion model then all the data for the localization are accessible and the algorithm can start. Firstly the Particles are copied to the Auxiliary Particles structures then these particles are forwarded into the SpreadParticles method to be spread. After the prediction of them is done using the predict function and if needed by the motion model it is repeated. In continue the update function is called with the observations as an input. After the weighting, the particles weights are normalized and then resample. Here the resample function is called with inputs the Auxiliary particles and an array named index for the results to be stored and then the propagation is called with input that index array but for particles the original set. Thus the original particles are chosen and resampled. But these particles have not moved at all, so the SIR algorithm starts from the beginning using the prediction function again, the update and also the normalize. But before the resample stage a belief about the best position estimate is been searched. As best estimate we assume the particle with the maximum weight. After founding the maximum weighted particle the belief of the localization is being set and the resample stage continues. These LocalizationStep function returns a belief structure with the best estimate position and orientation for the robot.

In the top of this KLocalization a control system was needed. That system is the same the kselflocalize module. In that some additional function were made. Some of them where used to communicate with the webots module witch will be explained later. Using a definition in the code we can manage the module to exlude the use of the webots module in order to run exclusively to the real robot.

A function was made to control the robots motion given the name of motion that we want to execute together with the number of steps. This function uses the naoqi motion module to execute the three basic motions, ‘‘walk, turn, side step’’.

Another two useful function for extracting observation data were made. One for the simulation and one from the observation module. The function ExtractFakeObservations as it’s name say was

creating observations using the exact position of the robot. Thus the distance and the bearing was the most accurate that could exist. The distance was calculated using the Euclidean distance from the point were the robot was and the bearing using the atan2 function. Then the fake measurements were packet in KObservationModel structures and then inserted to a vector.

The function ExtractRealObservations used to call the Observe method of the Vision module and the only think that made was to take the results coming from the above call and packet them to KObservationModel structures and then inserted to the observation vector.

The top function witch also was binded to be callable from other modules is the TestLocalizatin function. That function was called using and ALValue input type. That input consists of a string and a number of steps. The function call the DoMotion with these arguments to make the robot move. Then after the motion an Extract Observations function is called to collect observation data. Then the LocalizationStep runs the algorithm of localization to localize the robot and return its belief about the position the robot most probable is in.

In order to visualize the results a gui was made only for displaying the belief and the particles weights. A robocup field was draw using opencv drawing methods. In the top of that image particles were also drawn as long as the robots belief. An example can be shown in the figure .... OpenCv gave us the ability to easily save images of the field the particles on it and the track that the robots belief followed. For the visualization purpose four functions were made to draw the field in the begging the particles the belief and the real position in the simulation testing.

Also when running in the simulation the position error was easily calculate and set to a ALvalue structure which the TestLocalization method was returning.

MoreOver a function was made to evaluate and extract date for modeling the Observation module. The ObservationModeling function was made to take measurements in different positions in a field. The measurement were then saved to a file as well as the exact position of the robot for further statistical processing.

5.2 Measurements extraction

For the observations extraction another module was made. This module has as role to provide measurements about distance and bearing scanning the visible area and processing captured images. Because the feature recognition belongs to the computer vision area was separated from the localization. The module name is Kobserver.

This module includes all the necessary functions needed to export measurements info from the environment the robot sees. Also this module has some testing functions. Three other module are needed from Kobserver. A motion proxy for the head motion a VideoInput proxy for accessing the camera and a log proxy to output so info messages. In the initialization of the module a call to the VideoModule is done to initialize the video input module with resolution, colorspace and frames per second parameters.

We will start to analyze the functions starting from the lower level. As previously said the goals are recognized by their color. That is the reason that we first segment the image to desired colors, in our case skyblue and yellow. The segmentation technique was derived from the Kouretes image processing architecture used by our team ref. So a segmentation class intanse was used in order to segment the image. In the segmentation class there wera also coded the rules needed to seperate the color, witch is also called colortable. These rules were made using the Kouretes Color Classification tool a long with with images captured from the robots camera. Rules were produced from images captured from the webots simulation 3D environment as well as real field environment in our laboratory.

In a function named checkcolor with input the value of the color tha will be checked the process starts. Here is called a method from the VideoInput module witch can be found is in two formats for local and remote call. For this reason definition blocks were made to be compiled for each case using just a define. After the image data is returned, is stored in a IpImage structure witch is a opecv structure for images.

In order to focus the selected color to the center of the image color tracking method was used. Firstly in an execute named function the some parameters about the center of the colored pixel was calculated. More spefiticaly in the execute function a grid scan is perfomed segmenting pixels for the desired color and in paraller calculating the sum of these pixels horizontally and vertically also they are counted. Then the mean of these pixels in horizontal and vertical give as two variable inticates the center of the color's blob. In addition the center of the blob regarding the center of the image is calculated.

These values will are used later to center the blob in the captured image moving the head respectively. These task is done in the track function. Here is calculated the motion that the head joints must have in order to center the blob, and like that focuses probable to a field's goal.

For the extracting measurenments other two functios have implemented. The one finds out the the usefull pixels to calculate the distance and the other uses that info to calculate the distance and the bearing.

The first function is calle ExtractGoalPostsPixels and takes as input the color of the pixels that will be searched. The concept here is to find out the beggining pixels of a goal posts base. So a scan is deployed in starting from left to right and from the bottom of the image to the top pixels. If pixels with the desired color are found and a number of other pixel above of them are also the same color, witch probable is a goal post then the position of the first instance is saved in a array having the lenght as the width of the image and is named points. After that array is scanned for its lower points. That is done calcultating the difference in the values between neightboor values, it is the derivative. If a minimum is found it is inserted in a table its value and the index position it was found. If another minimum is found later is examined if it is closed to the firtst the it replaces it if it is away enough then this low point represents the other goalpost so that is saved in the next to the first maximun. In the end of these procedure a two by two table consist of the values of the lower pixels of the goalposts in the processed image. These four values are then inserted to a ALValue array and returned.

The second function is called GoalPostsDistance it takes as in put the searched color and a boolean value named topcamera witch indicates with witch camera the image was captured. Purpuse of this function is to calculate the distance to the goalpost as well as the bearing to them. Inside the funtion the ExtractGoalPostsPixels is called and the two minimum pixel values are returned. We assume that they represend the lower goalpost pixels in the image. The naoqi api provides a function that calculates the position of a robots parts in regard to a known space. So we chose to find out the position of the head reagarding the foot of the robot. The function return the position of the three coordinates of the robot head. One of them is the height especially needed to calculate the distance. Also another one is needed to calculate the bending angle of the body. Moreover the headpitch and headyaw angles were retrieved using the getposition function for these joints.

For the distance calculation first the angles were calculated. A phi angle is calculated, and represends the angle between the straight line of robot's head to goal post's base and the robot's body. That angle consists of the headpitch anlge and the angle as a part of the field of view of the camera. To calculate the second one the height in pixels witch the color of the goal starts and was previously found is needed. First the ratio of the pixel to the height of the image is found and wanting the divergence form the center of the image the value is substracted from 0.5. That value then is multiplied with the horizontal field of view to finaly result in the angle from the center of captured image. Another one angle is calculated and is named theta that angle is created by the potential forward-backward bending of the robot. This is calculated using the atan of the height and the divergence value from the vertical position. Having all the parameters of the triangle known the distance can be calculated using the tan of the angle that consist of the headpitch the phi and theta multipied by the height of the robot. If the second cammera is used then the above angle differs by 30 degrees because and also is in lower position.

For the bearing equally procedure was followed but hear only an angle was needed. So the using the position as the width were the lower pixel was found the angle regarding the center of the image as a part of the horizontal field of view angle is found. Adding the robot headyaw angle alue we have bearing angle.

These values are pushed in a ALValue data structure and later that structure is returned.

The basic function that executes the functions described above is the Observe function. This is the function tha the kselflocalize module calls in order to retrieve observations. The concept behind these function is to scan all the visible range moving around the head and trying to take measurenments. A table with values of head positions is used as a guide to turn the head around. The head is seted in each position. Then we check for each color respectively. Using the checkcolor function a search for the specific color is done and if the color is found then the track function tryes to focus the robots head to the recognized blob. The check and track function are called 5 times until the robot's head converges to a position having the blob as closest as possible to the center of the image. As the goal is focused the measurenments are take using the GoalPostsDistance function if 4 values are returned the we assume that both goal posts are recognizes. If only two values returned then we assume that the generally the goal is recognized. Then for each of the recognized landmarks three values are pushed to an ALValue structure. The signature (string) of the recognized landmark and the distance and bearing values. Then a check is done for the other goal color and the same

procedure is executed but with input the other goals color. Then the robot's head is moved to the next position and the same procedure is done. In the end the alvalue structure will include all the observations that were extracted and this structure is returned.

Besides the Observe function there is another usefull function for testin perpuses. This function named DisplayImage makes possible for someone to see thru the robot's camera. Besides that a segmended image is also dispayed and the goaldistance fuction is also called. The camera image is saved in the opencv's iplimage and dispayed by opencv functions. Furthermore image saving function was programmed to be called whem the space key be pressed.

5.3 Simulation Agent

Along with the naoqi sdk aldebaran robotics had provided an binary executable for the webots simulation environment. That client makes the simulation run the naoqi platform so what ever runs to the simulation as a part of a naoqi module it can be run theoritacaly and to the real robot. Benefiting from the ability of loading libraries using the naoqi and our knoledge on programming webots clients we made a special module. That module had access to webots simulator API. We had include webots header files about emmiter, receiver, and gps devices and linked with the webots libraries in the compilation. With that trick we could retrive usefull data from theses devices as well as to control the simulation environmet. So in the beggining the most usefull device tha we enables was the gps devices. Using that inside our module we manage to retrieve the exact position of the robot in the simulated field. Besides that we patched a webots supervisor client to be usefull. More specifically using its receiver the supervisor could here to remote commands and the one tha was implemented made possible the placing of the robot in any position and direction inside the field. Then inside our module using the emmitter we could teleport the robot wherever we want.

Although we had all these we also wanted to retrive the robots orientation. But no compass device was existing in the robot. So we added one compass device to the robot patching the nao's vrml model. Then we included the headers needed for the compass device and we enabled the device.

Besides these functions a function to collect statistical data easily was made. The concept is to make the robot move save it's position and oriantation and transfer the robot to its initial position. Setting a value we could set the number of repeatition of the motion we want to execute. So saving the initiall and the finall position of a motion we collected statistical data about the robots motion. These repeative procedure was made for all the neccesary motions. In each case an txt file was produced keeping the initial position and orientation as well as the finall position.

5.3.1 Matlab Scripts

In order to make our life easier matlab scripts were also used. In the beggingng the whole architecture was implemented in matlab. Using fake observation but good motion model the localization algorithm was tested. After starting to work withe the real robot and writing c++ the localization slowly ported to c++

Still some usefull scripts are used. To calculate the mean an the deviation o the robots motions and to be saved in an xml like format a simple scripts was used. Also wanting to find the error function for the observation correction matlab code was writen.

Chapter 6

Results

6.1 Expirenments

We have made a lot of expirenments trying to calibrate our systems parameters. Below will describe some of the expirenments and the results of them.

6.1.1 In the simulation environment

Most of the expirements were taken place in the simulation environment. Here we extracted motion models and observation models.

Motion Model extraction

Using our webots client we run a number of expirements for each available action. After the end of the action the robot was repositioned to the initially position. For each repetition we record the initial and the end pose in a file. In the figure 6.1.1 we can see the initial and end poses of the robot of all the repetitions along with the histograms of the distance the Direction and the Rotation relative to the motion. Then from these files we extracted the motion model of each motion and we save it in an XML file to use in the Localization.

Observation Model extraction

Again using webots and our object recognition module we have try to evaluate our environment measurements. The process was simple. We set the robot in positions several positions of a grid of the field and with several orientations and we are asking the robot to scan the visible area for goals. Then we collect all these data and knowing the position of the robot we can extract the error in the

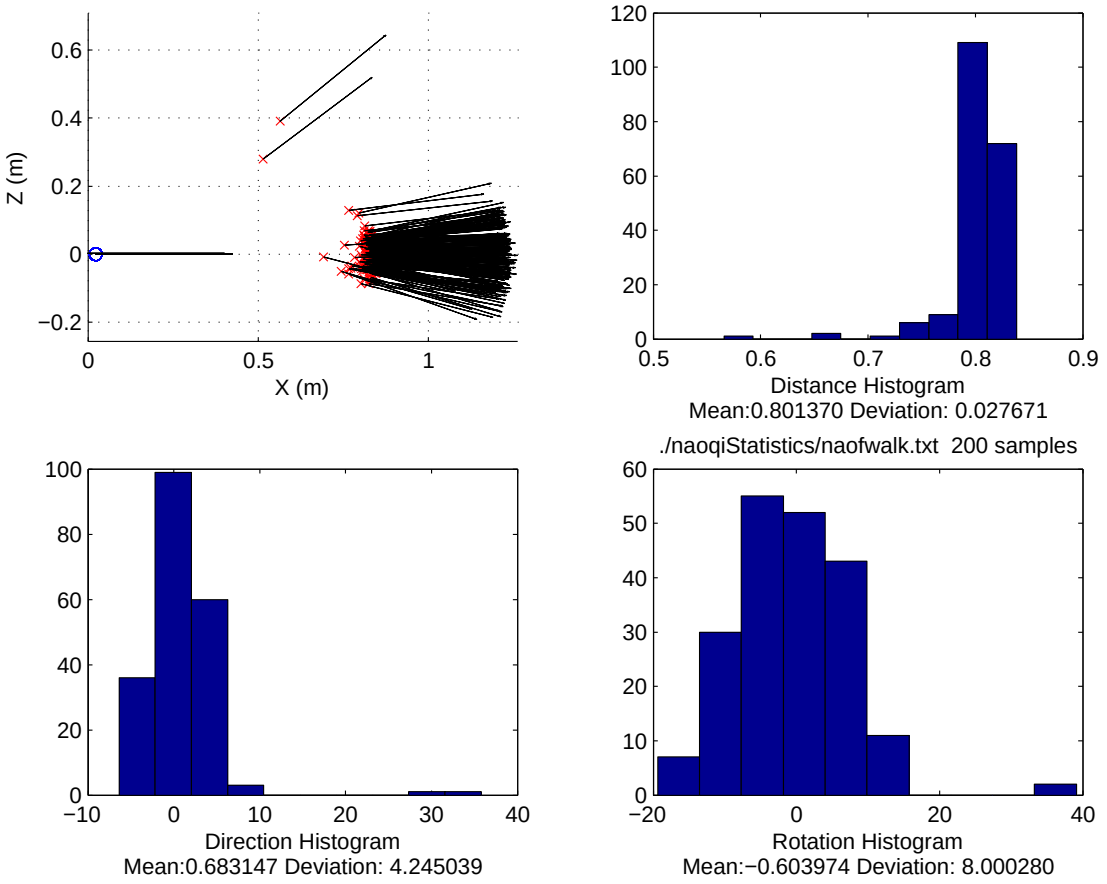

Figure 6.1: 200 repetitions of the action walk 10 steps

measurements. Int the figure 6.1.1 we can see from left to the right: Real Distance to the skyblue goal's left post, Measured distance, Mean Error of measurements, Deviation of the error.

After we must find a way to describe the error relative to the distance. So we take all the measurements from the 2D environment and we put them in a array with 2 collumns. Then we sort the rows by the real distance. Then having that data with distances relative to measured distances we calculate the error relative to the distances. In the figure 6.2.1 we can see the error relative to the distance. Moreover we try to fit a function to that error to describe the mean value of the error regarding the distance. The absolute difference that this function has from the measured data we consider it like the deviation of the error. We also try to fit a quadratic variable to that data. The parameters of these fitted functions both for the mean error and the deviation are saved to a XML file in order to be used from the Localization. Simiraly with the distance error modeling we model the Bearing modeling extracting the paremeters of the functions and saving them to XML.

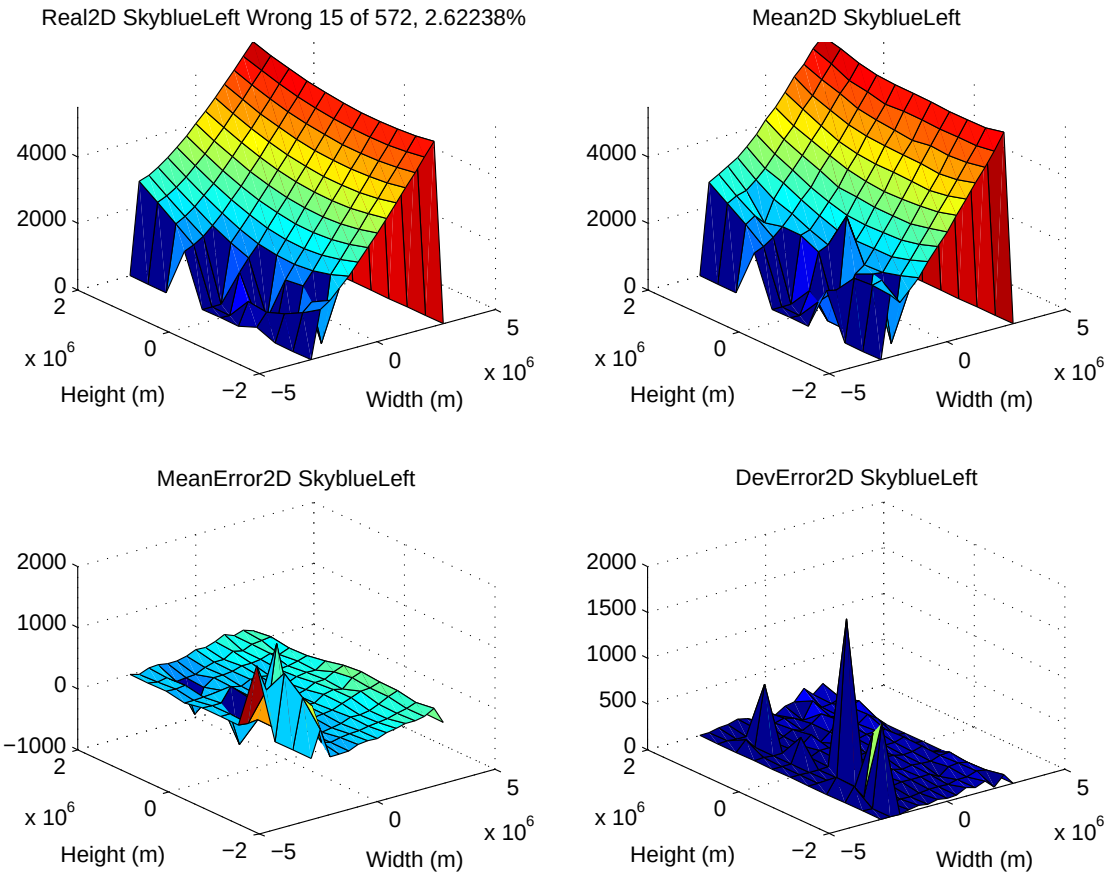

Figure 6.2: Distance measurements in a grid of the field

6.2 Localization Expirenments

6.2.1 In the simulation environment

We have tried different scenarios to test are localization algorithm to the simulation environment. In most of the expirenments we set the robot to an unknown pose. Then we run our algorithm and we try to take observations from the environment. In scenarios when we allow the robot to scan as much as possible observations the algorithm have result in very small error in position estimation. Also when we limit the number of observations to the number visible from the robot without turning the head the results are also good and enough for our purpose.

In the figurefig:error we can see an localization attempt with a cyclic walking scenario. In that expirement we have not enabled all the available extensions and we where trying to estimate the parameters of our particle spreading method. However even not complete and perfect the error 6.2.1 could be consider enough.

After a lot of experimentation and parameters calibration we made some expirements we the best possible results. Again here we have scenarios with the largest observed range along with the limited. Additionally the scenario was so difficult and uncertain that some times the robot crashes in a

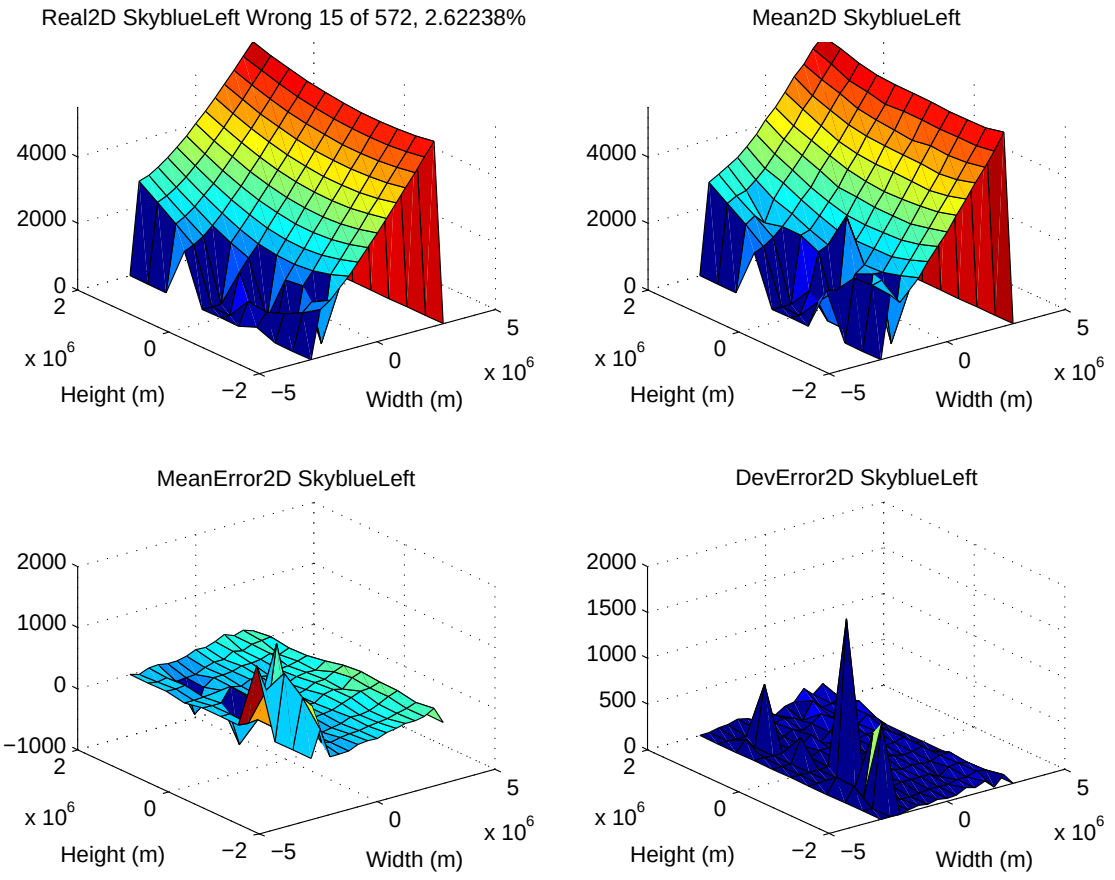

Figure 6.3: Distance measurements in a grid of the field

goalpost.

In the figures **??** we can see the path that the robot followed along with the estimated path derived from the robots beliefs. The mean value of the distance error is very low and that experiment could be fulfill our expectations for robot localization, and practically proves the success of our localization attempt.

Moreover we run expirenments when we limit the robots head scanning range to zero **??** , resulting in recognizing objects only in the range of the field of view of it's camera. We allowed the head to turn only to the point tha could focus in an object. The results were quite interesting and here we see the importance of the motion model when the particles are moving in the right direction, and even if there is no observation the estimation error is low.

	Full range	Limited range
Mean Position Error	71,18 mm	210.78 mm

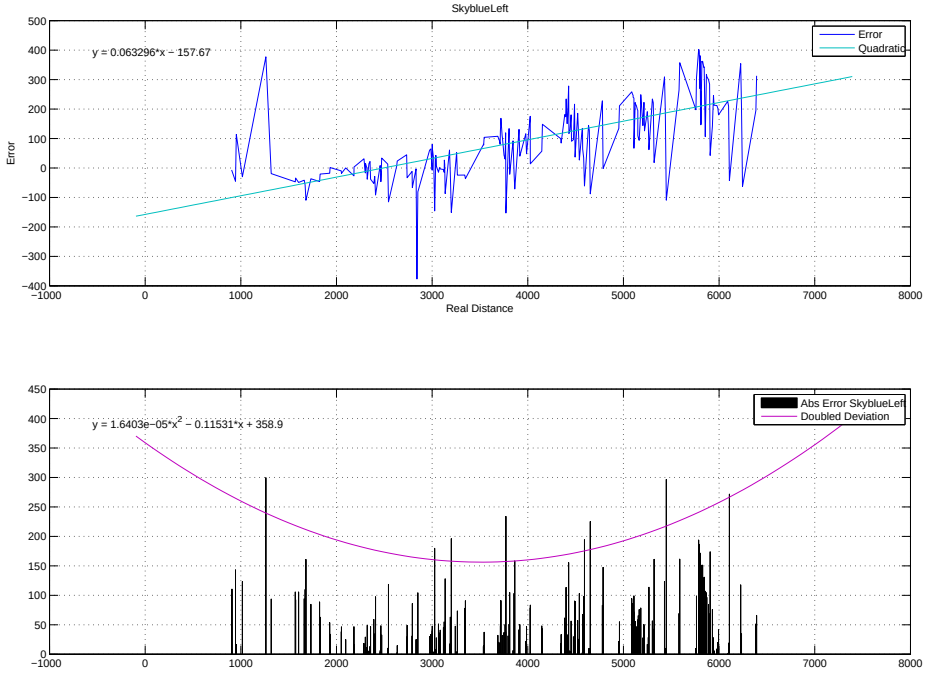

Figure 6.4: Error in Distance measurements, error of the fitted function

6.3 Goal Recognition

In order to have a complete localization system we needed at least a goal recognition system. Using our Kouretes Color Classification tenchique we could extract segmented images. But not all the images were perfect segmented for several reasons. Our recognition algorithm had to be tuned in order to recognize correct as better as possible the goals.

So when the segmentation noise is low and at least a goalpost is full inside the segmented image is easy for use to extract the distance and the bearing to that goalpost. Even if a goal post is too close to the robot and we can not see all it's height if we recognize which goalpost is (left or right), we can measure the relative distance and bearing.

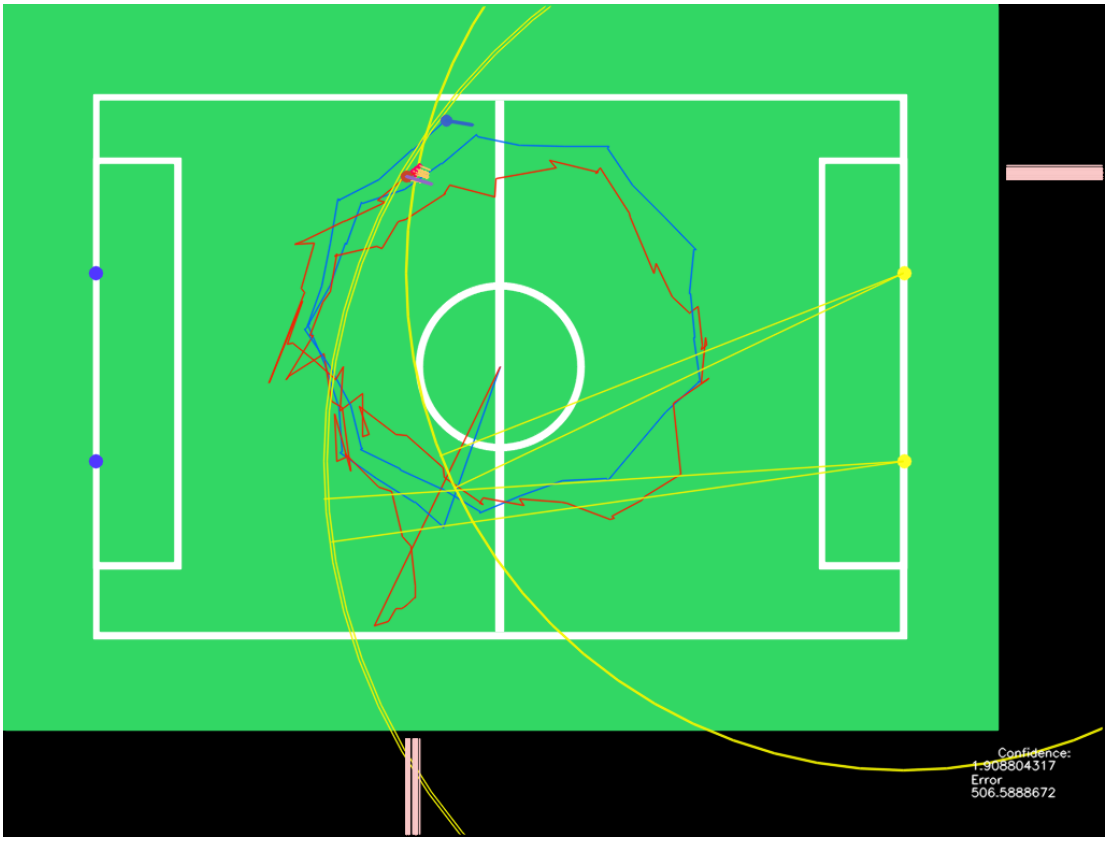

Figure 6.5: Cyclic path, 400 particles, limited observations

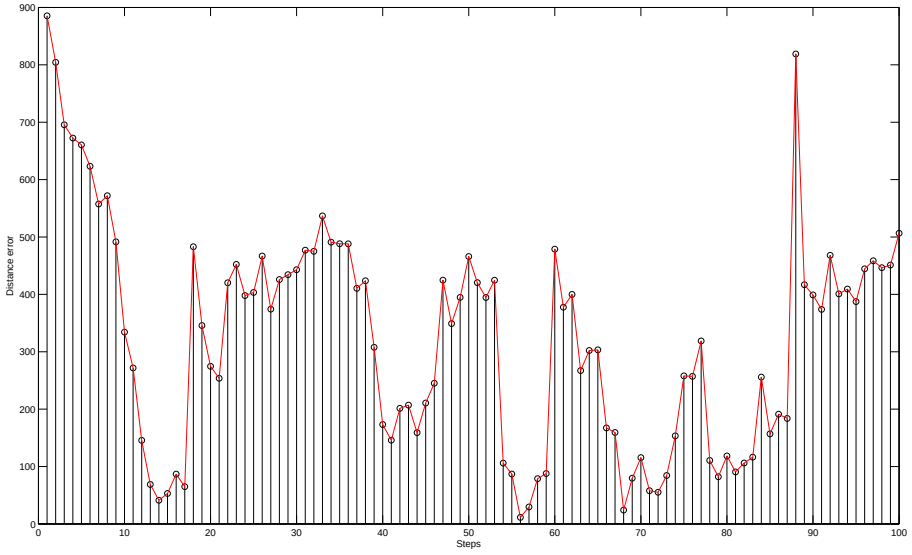

Figure 6.6: Error in the Cyclic path, 400 particles, limited observations

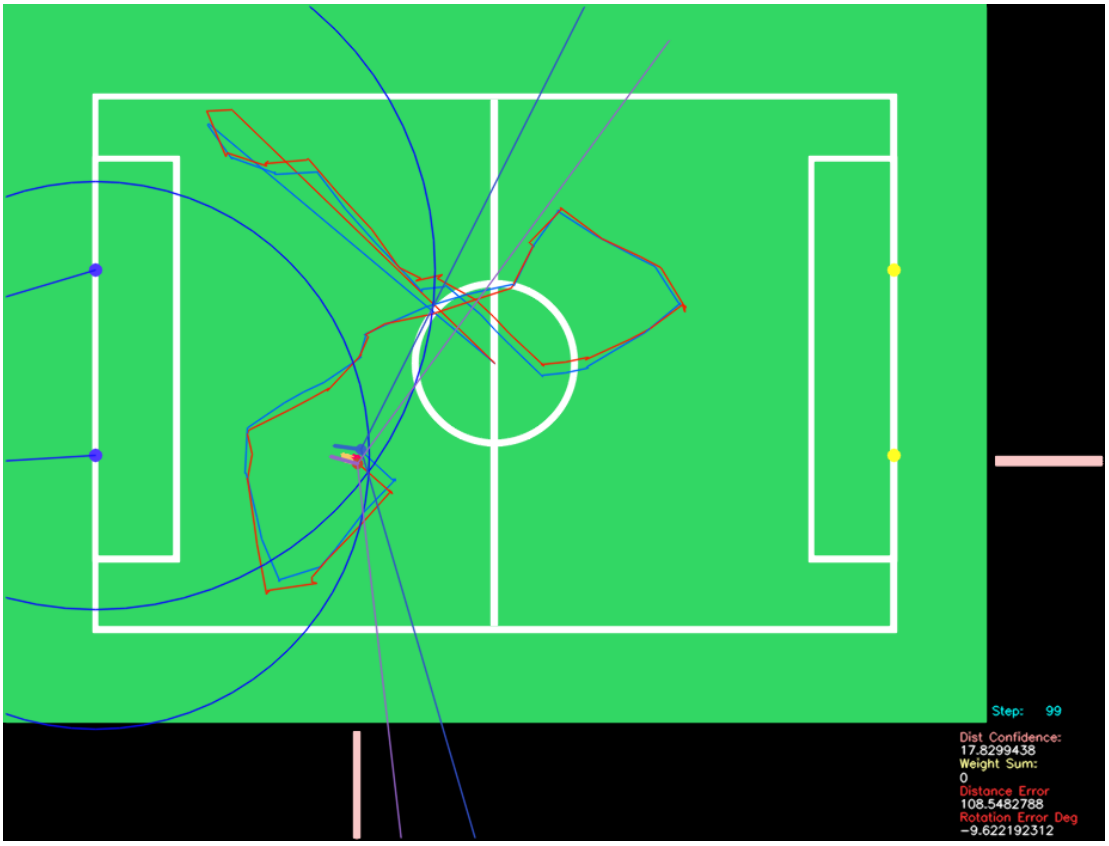

Figure 6.7: Difficult path, 200 particles, full-range scan observations

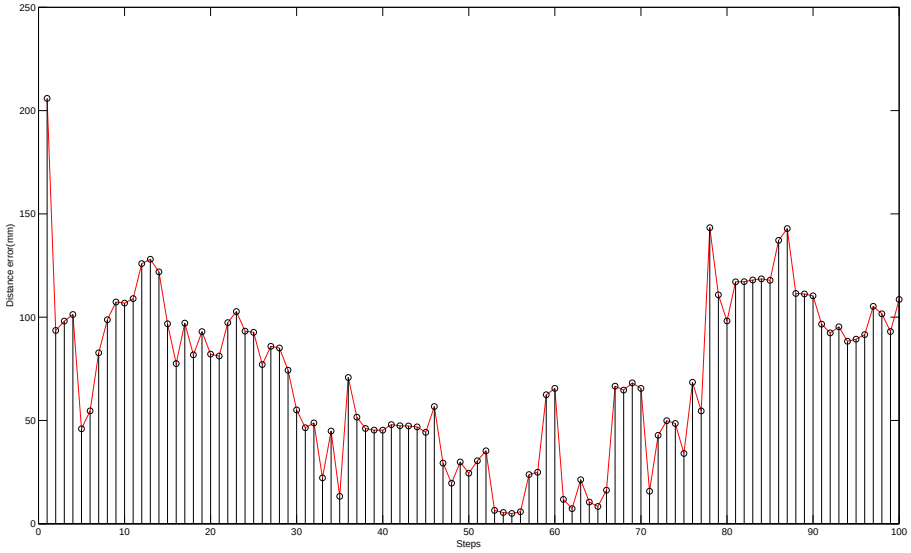

Figure 6.8: Error 200 particles, full-range scan observations

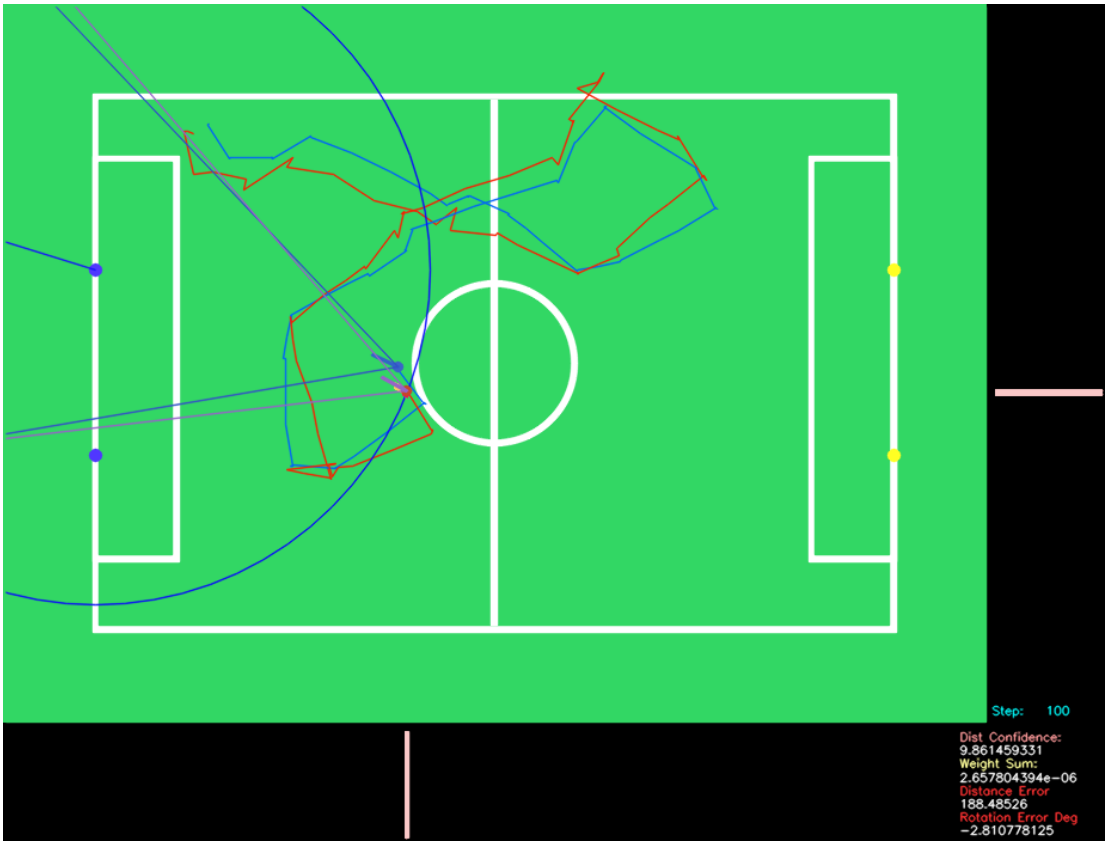

Figure 6.9: Difficult path, 200 particles, observations without scanning

Figure 6.10: Error, 200 particles, observations without scanning

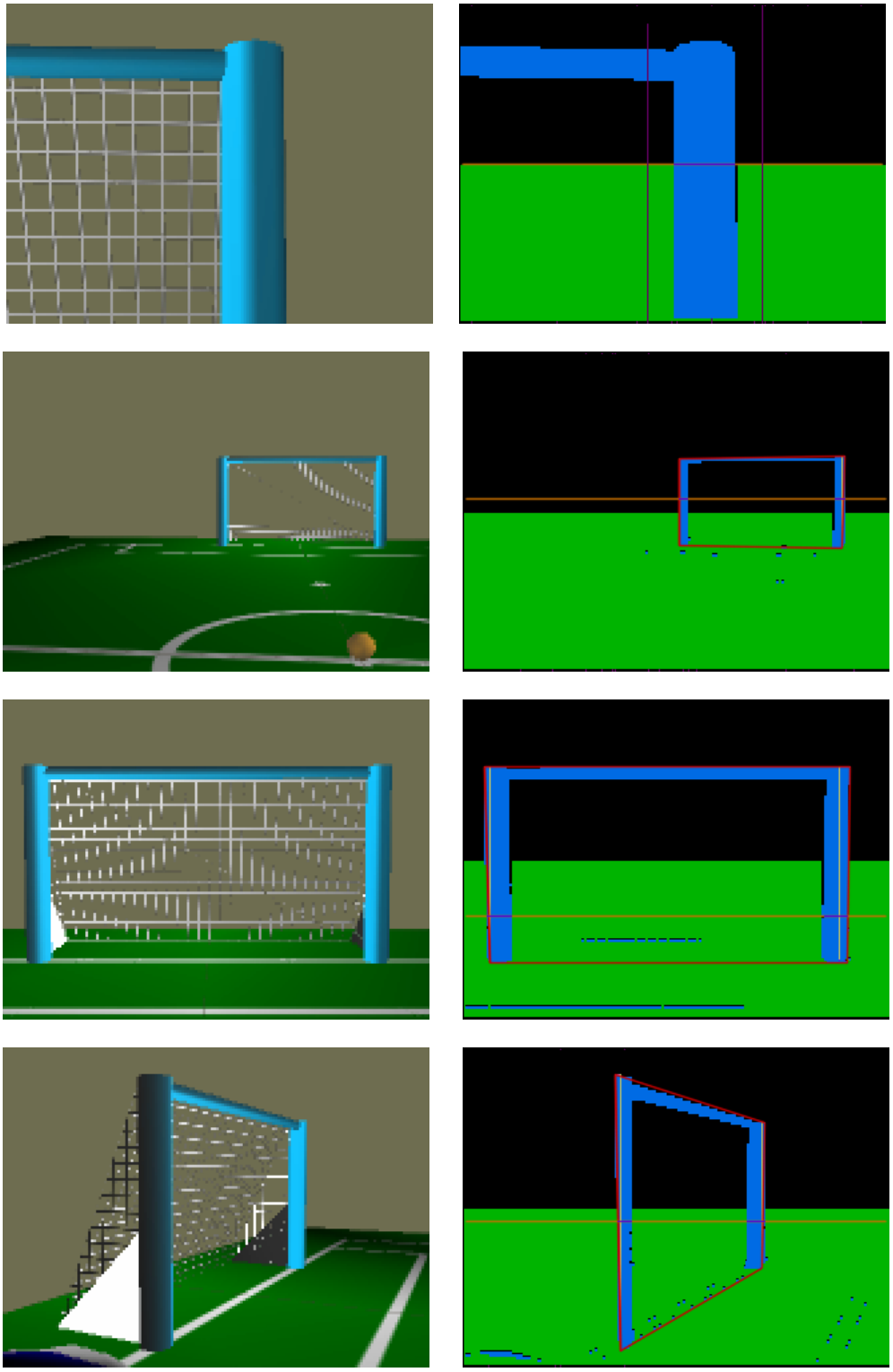

Figure 6.11: Recognizing Goal posts in webots

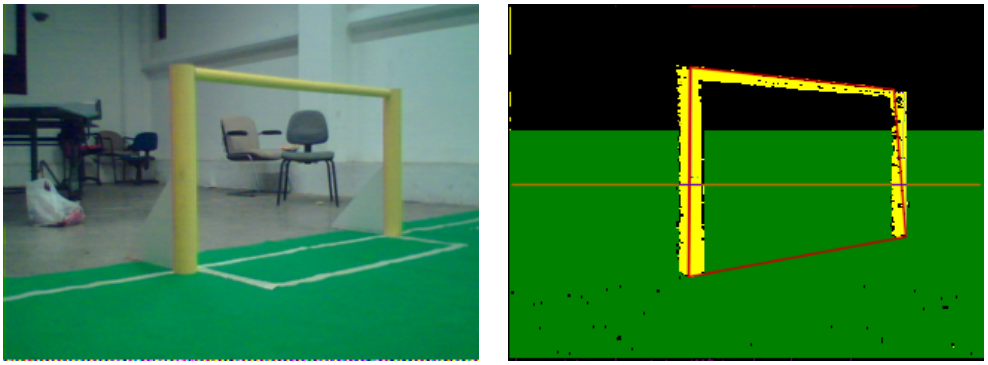

Figure 6.12: Recognizing Goal posts in real environment

Chapter 7

Conclusions

7.1 Conclusions

The method presented in this thesis, has a number of advantages that makes it a good candidate for the real world in a Robocup field. The accuracy of localizing in the field is varying regarding the number of features that are observed. However the error of the estimated position is far enough for our purpose.

7.1.1 The quality of observations

The quality of observations play a serious role in the localization. If we can not trust what the robots sees then is better no to use an observation.

7.2 Lessons learned

The worst think about robotics is that in the real world everything is completely different from the simulation. As for us the lack of a real field in the university in a specialized space in the end was the basic problem.

7.3 Future work

There is a number of features that will be implemented and will make the system more reliable and faster:

- Regarding the robots estimated position, the localization system can ask for observations in a specific range in order find as quickly as possible more observation.

- We can recognize a possible fall using robots sensors and this event could result in more particle spreading in the area where the robot fall.

Bibliography

[1] David Gouaillier and Pierre Blazevic. A mechatronic platform, the Aldebaran robotics humanoid robot. *32nd IEEE Annual Conference on Industrial Electronics, IECON 2006*, pages 4049--4053, November 2006. ISSN 1553-572X. doi: 10.1109/IECON.2006.347816.

[2] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1-2):99--141, 2000.

[3] N. J. Gordon, D. J. Salmond, and A. F. M. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. *Radar and Signal Processing, IEE Proceedings F*, 140(2):107--113, 1993. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=210672.

[4] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39--42, 2004.

[5] "M. Betke and K. Gurvits". "mobile robot localization using landmarks", "1994". URL "citeseer.ist.psu.edu/betke94mobile.html".

[6] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Québec, CANADA H3A 2A7, 2004.

[7] Arnaud Doucet, Nando De Freitas, and Neil Gordon, editors. *Sequential Monte Carlo methods in practice*. 2001. URL http://www.worldcatlibraries.org/wcpa/servlet/WLAHoldingServlet;jsessionid=4C45474A5C0E58167BABE2E351EBBCAA.three?query=no:45024472&sessionid=4C45474A5C0E58167BABE2E351EBBCAA.three&recno=1&zip=50014.

[8] Michael K. Pitt and Neil Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590--599, 1999. URL http://www.jstor.org/stable/2670179.

[9] Nikos Vlassis, Bas Terwijn, and Ben Kröse. Auxiliary particle filter robot localization from high-dimensional sensor observations. In *in Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, pages 7--12, 2002.