Christof Rath, Bakk.techn.

Self-localization of a biped robot in the RoboCup™ Standard Platform League domain

Master's thesis

Institute for Software Technology
Graz University of Technology
Inffeldgasse 16b/II, A – 8010 Graz
Head: Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany

Supervisor: Dipl.-Ing. Dr.techn. Gerald Steinbauer

Evaluator: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Wotawa

Graz, (March, 2010)

Always check the pool before you jump in—you never know!

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am ……. ……..
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

……. ……..
date (signature)

Acknowledgments

In order of appearance—almost.

My thanks to my always supporting family, to my aunt, who sponsored a big part of my studies. To Ruth my love, who was always patient, even when it seemed to never end, and who made with me the sole thing that really matters, our daughter Charlotta. My thoughts to my grandmother, who would have loved to witness my graduation.

I want to thank Gerald Steinbauer, who brought me to the science of robotics and to the RoboCup™, and who gave me the freedom to explore the topics and aspects of this thesis, and to Franz Wotawa who had always an open door and a good advice. And, last but not least, I want to thank my colleagues, for the nights we spent programming, for the nights we spent out, and for the many hours we stood together to keep the spirit alive, I could not have finished this work without your help.

Abstract

The topic of this thesis was the implementation and evaluation of a self-localization algorithm for the biped robot Nao. The robot should be able to evaluate its pose, and consequently its team members pose, on a soccer field of the RoboCup™ Standard Platform League. A computationally efficient approach, based on the work of I.J. Cox, has been implemented, and evaluated by the use of a simulator and the real Nao. The tests with the real Nao were performed in a six-dimensional motion tracker facility, that is, it provides translation and rotation in space, to get accurate reference data. Preliminary results show that the pose could be successfully estimated in 83 % of all evaluated datasets. Furthermore, it could be shown how important proper synchronization of the various datasources (like camera, hardware data, or intermediate results) on the real platform is.

Kurzfassung

Ziel dieser Arbeit war die Implementierung und Evaluierung eines Selbstlokalisationsalgorithmus für den zweibeinigen Roboter Nao. Selbstlokalisation bedeutet in diesem Zusammenhang, dass der Roboter in der Lage ist, seine eigene Position auf einem Fussballfeld der RoboCup™ Standard Plattform Liga zu kennen und folglich auch die Position der anderen Roboter seines Teams. Der implementierte Algorithmus basiert auf der Arbeit von I.J. Cox und zeichnet sich durch den geringen Bedarf an Rechenleistung aus. Zur Evaluierung wurde der Ansatz mittels Simulation und danach am echten Roboter getestet. Die Tests am echten Roboter fanden in einer 6D-Motion Tracker Umgebung statt. Der Motion Tracker liefert die Position und Orientierung der beobachteten Objekte im Raum. Diese Daten dienten als Referenz für die Tests. Bisherige Resultate zeigen, dass die Positionsbestimmung in 83 % der getesteten Datensätze erfolgreich durchgeführt werden konnten. Als weiteres Ergebnis zeigte sich die Wichtigkeit, die unterschiedlichen Datenquellen (Kamera, Hardwaredaten und Zwischenergebnisse) synchron zu halten um schlüssige Ergebnisse zu erhalten.

Contents

List of Figures

Chapter 1

Introduction

The topic of this thesis is the implementation and evaluation of a self-localization algorithm for the biped robot Nao (see Section 2.2), as used in the RoboCup™ Standard Platform League soccer domain. Self-awareness of robot systems is a crucial, but still underdeveloped field in the science of robotics. Today, many systems rely on error-free platforms, that is, in many cases errors are either undetected or ignored, which means that the behaviour of robots is in many cases not dependent on the system status. If such a system is erroneous its behaviour is often unpredictable.

In the soccer domain the minimum requirement, regarding self-localization, is to know where the robot is relative to the ball and relative to the opponents goal. Given that, a robot is able to approach the ball and to turn around the ball until there is a straight line between robot, ball, and opponent goal, the robot will kick the ball somewhere in the direction of the goal. By repetitively executing this procedure, a robot should finally be able to score a goal. In this case the robot has no means to involve its team members into the decision-making process, for example, another robot of the same team might be closer to the ball and therefore faster. In the worst case two robots might be at, more or less, the same distance to the ball. If both of them execute the same task, there is a good chance that they will collide, but for sure, they will leave a big area of the field unguarded. Obviously, without knowledge of the global position, chances are high to leave the field, and to enter restricted areas on the field, which both would lead to penalties.

As mentioned above, there are many reasons for getting knowledge about the global position, but at least when it comes to team-play, the robots of a team have to know their own global positions, and the whereabouts of the other team members on the field to coordinate the behaviours. By knowing the global position on the field, the robots can position themselves on strategic locations (and avoid restricted areas), coordinate passes, or evaluate the risks of the current situation.

1.1 RoboCup™ Initiative

The RoboCup™ was founded as an research and education initiative. It is an attempt to foster artificial intelligens (AI) and intelligent robotics research by providing standard problems. These standard problems have to be easy to understand by a non-technical audience but still complex enough to allow for a wide range of technologies and research topics to be integrated, examined and evaluated on various levels of scientific work. [Web98]

The ultimate goal of the RoboCup™ project is: "By the year 2050, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official FIFA rules, against the winner of the most recent World Cup of Human Soccer."

1.2 History

The idea of robots playing a game of soccer was first mentioned by Prof. Alan Macworth in a paper called "On Seeing Robots" presented in 1992 and later published in a book *Computer Vision: System, Theory, and Applications* [Mac93].

Independently, a group of Japanese researchers organized a Workshop on Grand Challenges in Artificial Intelligence in October 1992 in Tokyo. This workshop led to a serious discussion of using the game of soccer for promoting science and technology. Mainly two reasons take account for that: Almost everyone knows soccer, the goal behind the game, and—at least—the basic roules. Therefore all participants know what the effort is about. And also, soccer is a highly dynamic game and requires a lot of interaction between the players. This offers a broad field of research and many challenges at every topic of robotics. In June 1993, a group of researchers, including Minoru Asada, Yasuo Kuniyoshi, and Hiroaki Kitano, decided to launch a national robotic competition, named Robot J-League. The J-League, Japans top league in professional soccer, was founded in 1992. Due to the overwhelming reactions from researchers outside of Japan, an international joint project was founded named the "Robot World Cup Initiative".

It is to say that concurrent to the discussions held in Japan, several researchers already used the domain of soccer for their work. For example, Itsuki Noda, at the Electrotechnical Laboratory (ETL), a government research center in Japan. He conducted multi-agent research within the soccer domain [Nod94] and started the development of a dedicated simulator for soccer games. This simulator became later the official soccer server for the RoboCup™ 2D soccer simulation league (see Subsection 1.4.1). The version 0 was programmed in LISP and announced to be the first open system simulator for the soccer domain enabling multi-agent systems research. The current version 12.1.3, as of January 2009, is covered under the LGPL [Web07] and can be found at SourceForge [Web09c].

The first public demonstration of this simulator was made at the International Joint Conference on Artificial Intelligence IJCAI-95.

The laboratory of Prof. Minoru Asada at the Osaka University and Prof. Manuela Veloso and her student at that time Prof. Peter Stone of the Carnegie Mellon University had also already been working on soccer playing robots. The participation of these early pioneers in the domain of robot soccer has been very important for the success of the RoboCup™.

The first competition has been the Pre-RoboCup-96 which was held during the International Conference on Intelligence Robotics and Systems (IROS-96) in Osaka, Japan. Eight teams, from Japan the USA and Australia, were competing in a simulation league and there was a demonstration of real robots of the middle size league.

The year 1997 can be seen as a mile stone in the research for AI. In August 1997 the first Robot World Cup Soccer Games and Conferences were held in conjunction with the IJCAI-97 in Nagoya, Japan, with 38 participating teams from eleven countries.

1.3 Timeline

League/Year	'97	'98	'99	'00	'01	'02	'03	'04	'05	'06	'07	'08	'09
Soccer													
2D simulation	●	···	···	···	···	···	···	···	···	···	···	···	···
Small-size	●	···	···	···	···	···	···	···	···	···	···	···	···
Middle-size	●	···	···	···	···	···	···	···	···	···	···	···	···
SPL-4		○	●	···	···	···	···	···	···	···	···	†	
Humanoid				○	○	●	···	···	···	···	···	···	···
3D simulation								●	···	···	···	···	···
SPL-2											○	●	···
Junior													
Soccer		○	●	···	···	···	···	···	···	···	···	···	···
Dance		○	●	···	···	···	···	···	···	···	···	···	···
Rescue			○	●	···	···	···	···	···	···	···	···	···
Rescue													
Simulation		○	●	···	···	···	···	···	···	···	···	···	···
Robot		○	●	···	···	···	···	···	···	···	···	···	···
Service													
@Home										○	●	···	···

Legend	
○	performances
●	competitions
···	competitions continued
†	final year of competitions

1.4 Soccer Leagues

1.4.1 Simulation Leagues

source: robocupgamesprima2009

As one of the first leagues, that held competitions, 2D soccer simulation has a long tradition in the RoboCup™. The games are conducted by a standardized 2D soccer simulation server, the "RoboCup Soccer Simulator" [Web09c]. This server administers the game state and the world model (for example, which player is where and what is insight, where is the ball, etc.) and provides a noisy subset of the world model, as sensory data, to the competing agents via UDP packets.

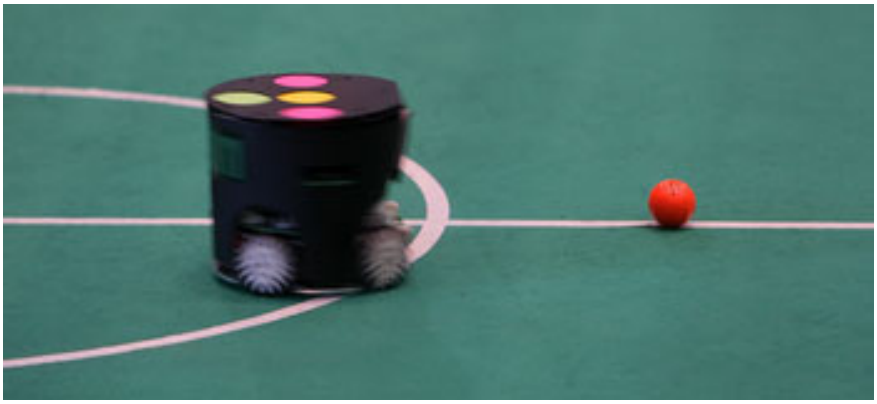
Each competition participant provides an agent, an executable that processes incoming sensory data and sends corresponding actions for the team players. Each player has a number of randomized properties (for example, stamina, regeneration rate, speed, ...). During the initialization phase the server provides a set of randomized team members for each side. The agent has to assign a role according to the fitness of the player (for example, a fast player might become an attacker). The actual game is divided in n time slots. At the beginning of a time slot each player gets a set of sensory data which has to be processed within the time slot to send an action command for that player (for example, goto a position, intercept the ball, ...).

The two-dimensional world model implies severe limitations in simulation accuracy, especially in respect to the position of the position and trajectory of the ball. To overcome these limitations 3D soccer simulation competitions are held since 2004. Currently (as of June 2008) the robots are represented as spheres with equal diameter (0.44 m) and weight (75 kg). The robots possess a kind of omnidrive, which allows them to accelerate into any direction [Rob06].

As there is no actual hardware, one has to cope with. The main research focus in the RoboCup™ soccer simulation domain, is to develop multi-agent behaviours and strategies.

1.4.2 Small-size league

In the small-size, or F180, league the teams focus on the problem of multi-agent cooperation and control in a highly dynamic environment with a hybrid centralized/distributed approach. The acting robots are limited in size: They must fit within a 180 mm diameter circle and must not be higher than 150 mm [LTC09b] (unless they use on-board vision, which is not very common within the league). Usually the teams use a global vision

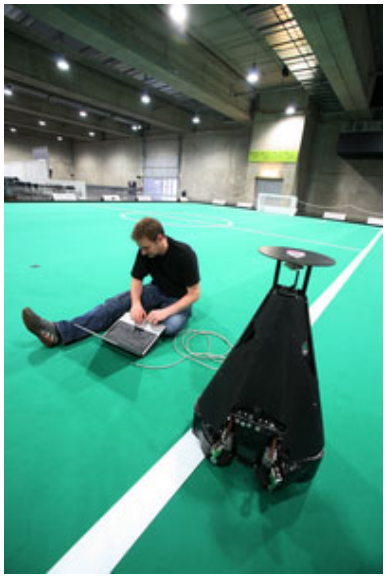


© TU Graz

approach. One or two cameras per team are mounted above the field. The images are processed by an off-field computer to extract the location of the robots and the ball. This computer is often also used to coordinate and to control the robots. For a game a maximum of five robots per team are allowed of which one is the goalkeeper. The ball, used for the games, is a standard orange golf ball.

1.4.3 Middle-size league

The robots of the middle-size league are currently the most sophisticated robots in the RoboCup™ with respect to the on-board computational power and sensory equipment. They have to carry all sensors they want to utilize, and also have to process the sensory data on-board to generate a world model and to act accordingly. The robots may interact with each other via wireless network connection to exchange the world models, and to execute multi-agend behaviours, but it is not allowed to use off-field computers, or a global vision as in the small-size league. Each team may consist of up to five robots, of which one is a designated goalkeeper. The robots may not exceed a 50 cm × 50 cm × 80 cm bounding box (except for the goalkeeper, which may temporarily extent for another 10 cm on any one side to catch the ball). They must not weight more than 40 kg. An orange standard FIFA size 5 ball is used to play the game [LTC08].



© TU Graz

1.4.4 Standard platform league

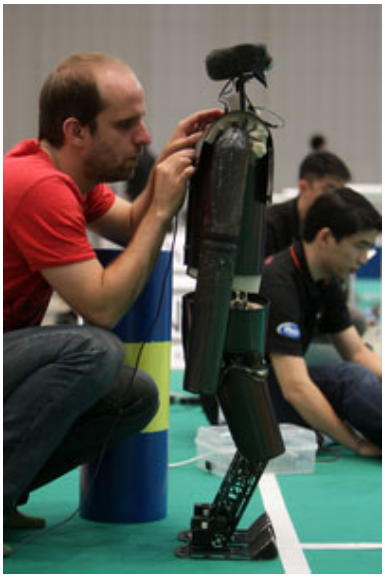
After the introduction of the Aibo™ by Sony, a four-legged robot, the standard platform league evolved. In this league every team operates the exact same hardware. Previously this has been the Sony Aibo™, but since 2008 the Nao, by the french company Aldebaran, a humanoid robot about 50 cm in height, is the new standardized platform. The teams are bound to the resources on the platform, that is, not allowed to modify the hardware of the robots, nor to use centralized data acquisition, or processing capabilities.

© TU Graz

Currently up to three Nao's per team are allowed on the field. The ball, even though not stated in the rules is currently the same as for the four-legged standard platform league, an orange plastic ball, approximately 8 cm in diameter. For further details refer to chapter 2.

1.4.5 Humanoid leagues

Currently the humanoid robots are separated in two groups: Kid-size robots with a height between 30 and 60 cm, and teen-size robots with a height between 100 and 160 cm. The robots must have a human-like body plan, that is, two legs, two arms and a head attached to a torso. The robots must be able to stand upright, and to walk or run upright. The robots should be equipped with sensors that have an equivalent in human senses, like cameras, force sensors, and such. These sensors must be placed at a position roughly equivalent to the location of the human's biological sensors (for example, the camera has to be placed in the head) [LTC09a]. For a soccer game not more than three robots per team are allowed on the field of which one is the dedicated goalkeeper. The kid-size robots play with a standard orange tennis ball, the teen-size robots with a orange beach handball, size 2.



© TU Graz

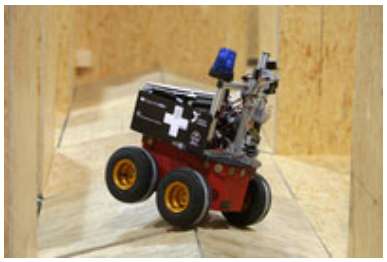
1.5 RoboCup™ Rescue

Disaster management involves very large numbers of heterogeneous agents in a hostile environment. To reduce the risks for rescue teams, tele-operated and/or autonomous robots can be used to navigate through a destroyed environment. The trigger for the RoboCup-Rescue project was the Great Hanshi-Awaji earthquake, that hit Kobe City on January 17, 1995, causing more than 6 500 casualties, destroying more than 80 000 wooden houses.

The main goals in this competitions are to get a map of a destroyed area, and to find hazards and casualties on the way. There are two leagues in the RoboCup™ rescue domain: Rescue robot and rescue simulation.

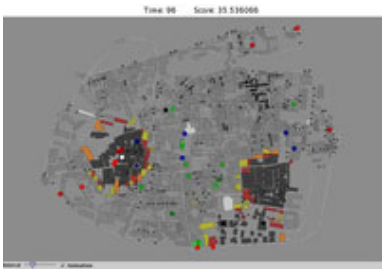
1.5.1 Rescue robot

The goal of the urban search and rescue (USAR) robot competitions is to increase awareness of the challenges involved in search and rescue applications, to provide objective evaluation of robotic implementations in representative environments, and to promote collaboration between researchers. It requires robots to demonstrate their capabilities in mobility, sensory perception, planning, mapping, and practical operator interfaces, while searching for simulated victims in unstructured environments [Web06].



© TU Graz

1.5.2 Rescue simulation



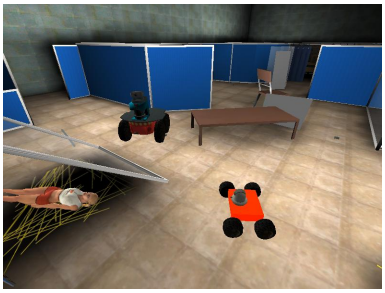
source: robocupgamesprima2009

The purpose of the RoboCup Rescue Simulation league is twofold: First, it aims to develop a simulation system to emulate realistic phenomena predominant in disasters, and second, it aims to develop intelligent agents (simulated robots) that can act in a disaster response scenario, as in the rescue robot league (that is, mapping, observation, etc.).

The RoboCup™ rescue simulation platform runs a kernel which connects various sub-simulators. These are currently a traffic simulator, a fire simulator and a civilian simulator. The traffic simulator enforces the rules of traffic (for example, one-way, multi-lane roads) and simulates the movement of vehicles along roads on a map, plus the resulting hazards like traffic jams and such. The fire simulator simulates the

processes involved in the propagation of fire, considering properties of the objects, wind and water content. The civilian simulator provides the behaviour of civilians, like movement along roads. Within this environment, the intelligent agents are used for search and rescue tasks.

1.5.3 Virtual Robots Competitions



source: www.oxfordrescue.co.uk

As in the Robot League, a devastated area has to be explored for victims with a team of robots controlled by an operator. However, the environment is pure virtual. Compared to the Robot League, the focus of the research is more focused on team work to accurately explore large areas and less on mobility. The major goal of this competition is to encourage intuitive operator interfaces that can be used to monitor and control multiple heterogeneous robots in a challenging environment.

A number of very interesting problems must be solved for a team to be successful in this league. These include navigation, exploration and mapping, victim detection, communication, and cooperation. In order to fulfil these tasks efficiently, practical human-robot interfaces have to be developed.

1.6 RoboCup@Home

Service robotics is the youngest field within the RoboCup™. First performances were held in Bremen, Germany in the year 2006. The league aims to develop service and assistive robot technology with high relevance for future personal domestic applications. [...] A set of benchmark tests is used to evaluate the robots' abilities and performance in a realistic non-standardized home environment setting. [Web10]

The robots have to assist in a household and fulfill typical tasks, like following, or guiding the human, communication by speech and gesture, and fetching items form the environment. The voice command of a test could be: "Grab the remote control and bring it to me". The robot is than forced to autonomously search the environment to detect a pre-defined remote control, grab it, and deliver it to the human who gave the command.

© TU Graz

Imminent problems to solve are safety and reliability. The robots may at no point be a risk for the persons inside the testing area, the spectators, or even the environment, like the furniture. For the competition, extra points are rewarded if the robot completes its tests without touching any object or human.

Furthermore, the rule book [LTC10] states, participating robot have to be autonomous and mobile. The tests for the @Home league aiming for real applications and should produce socially relevant results. The teams are supposed to not only show what can be put into practice today, but should also present new approaches, even if they are not yet fully applicable or demand a very special configuration or setup.

1.7 RoboCup™ juniors

© TU Graz

The RoboCup™ juniors competitions should encourage students, up to age 19, to get in touch with robotics. "The project-oriented education in the field of robotics provides a hands-on, scaffolded environment where learners can grow." [Web09b] The RoboCup™ juniors domain consists of a soccer, a rescue and a dance league.

1.7.1 Junior soccer

The junior soccer tournament, teams of two autonomous mobile robots play in a highly dynamic environment. The ball in use emits infra-red light to ease the task of ball detection and tracking. The field is surrounded by walls, and has colored goals as landmarks. The robots must be constructed exclusively by student members of the team. Mentors, teachers, parents or companies may not be involved in the design, construction, and assembly of robots.

1.7.2 Junior rescue

The robots, in the junior rescue league, identify victims within re-created disaster scenarios. The scenarios vary in complexity from line-following on a flat surface to negotiating paths through obstacles and debris on uneven terrain. The robots must act completely autonomous. Communication between robots is allowed via a bluetooth connection, but no external sensing or computation device is permitted.

© TU Graz

1.7.3 Junior dance

The teams of the junior dance league are encouraged to create a stage performance with which one or more robots performs to or with music. The performances will be classified either as dance or theater performance, and scored accordingly.

Chapter 2

Standard platform league

The standard platform league, also known as SPL, first appeared in 1998 as a performance during the RoboCup™ in Paris with quadruped robots, called Aibo™ (see Figure 2.1a), developed by Sony. In this league all teams are using the very same robots. The teams are not allowed to modify or extend the hardware. Thus the teams can concentrate on software development while, unlike the simulation league, execute their code on real robots.

source: www.cmu.edu

(a) Sony's Aibo™

© TU Graz

(b) Aldebaran's Nao

Figure 2.1: Former and current standard platform

As, in early 2006, Sony announced the discontinuation of its robotic division and therefore stopped the production and development of the Aibo™, the RoboCup™ community had to seek for a new standard platform. In the year 2007, a first beta version of the Nao, developed by the french company Aldebaran Robotics[1], had their performances as the

[1] http://www.aldebaran-robotics.com/en

new hardware platform. In 2008, the Nao's had their first official competitions, and the Aibo's had their last at the RoboCup™ competitions held in Suzhou, China. Even though the teams were supplied with the second generation Nao's, the hardware was still way too fragile to play the games accordingly to the rules. The assistant referees were asked to catch falling robots to prevent them to break and the robots were not allowed to touch each other, and in the end the field was crowded with people that tried to rescue their robots. The stressed hardware led to limited locomotion and thus the majority of the games ended 0:0. The feedback of the RoboCup™ 2008 and the fact that Aldebaran had to maintain a so called "Nao Clinic", an onsite repair and replacement facility, the teams were provided with the current version Nao v3. Figure 2.1b shows a version 3 Nao at a game during the RoboCup™ 2009. This, so announced, last beta version is a real improvement in robustness and reliability, and the RoboCup™ 2009, held in Graz, Austria, showed its potential as the new standard platform.

2.1 Team ZaDeAt

The team ZaDeAt has been founded as an intercontinental research effort between the University of Cape Town, Cape Town, South Africa, the RWTH Aachen University, Aachen, Germany, and the Graz University of Technology, Austria, in the year 2007.

© ZaDeAt

Figure 2.2: Team logo

The *Knowledge-based Systems Group* at the RWTH Aachen University focuses, among other things, on intelligent high-level control of robots and agents acting in dynamic domains. The *Institute for Software Technology* at the Graz University of Technology is engaged in the field of intelligent robust control of autonomous systems. Recently, the *Robotics and Agents Research Lab* at the University of Cape Town has been founded which is run by the Mechanical Engineering Department together with the Agent Lab, which is located in the Computer Science Department and focuses on computational intelligence. [FSMP08]

The ultimate goal is to intertwine the fields *reasoning about actions*, *model-based diagnosis and reconfiguration*, and *emergent behavior modeling* to design self-aware robots. Unresolved challenges are, how complex reasoning and diagnosis techniques can be down-scaled to work on computationally limited systems like the Nao. As for the day-to-day business, the South African part of the team has a strong mechanical background and is, thus, currently assigned to develop a closed loop motion engine. The base framework and the behaviour control are provided by the German members, and the Austrian part develops object detection, self-localization, odometry, and the basic skill set.

2.2 The current platform — Aldebaran's Nao v3

The Nao is a kid-sized, humanoid robot designed and developed by Aldebaran.

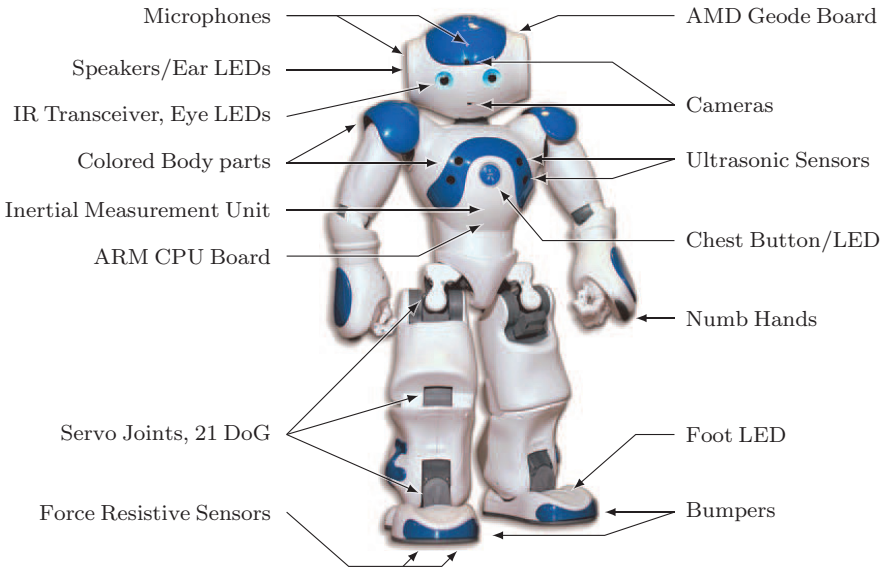


Figure 2.3: Nao overview

Figure 2.3 shows the specialized RoboCup™–version Nao has 21 degrees–of–freedom, that is, there are 21 servos that can be used to move the parts of the body. A block diagram of the robot is shown in Figure 2.4.

Input:

- Current angular position of the joints
- Two cameras in the head
- Two microphones, also in the head
- A chest button
- Two pairs of ultrasound sensors at the chest
- An inertial unit, consisting of a three–way accelerometer and a two–way gyrometer, at the center of the torso
- Two bumpers at the front of the feet

- Four force sensors per foot

Output:

- Desired angular position of the joints
- Stereo loudspeakers, used together with the integrated speech synthesi,s or to play arbitrary sounds
- Various LEDs, around the loudspeakers, the modelled eyes, inside the chest button and on the topside of the feet

Motion:

The teams are free to create their own motion engines. A low level interface is provided to control the joints in real-time. In addition, Aldebaran provides a motion module that enables basic motions based on an open loop control algorithm, that is, the motions are based on timed commands without sensory feedback. The basic motions include:

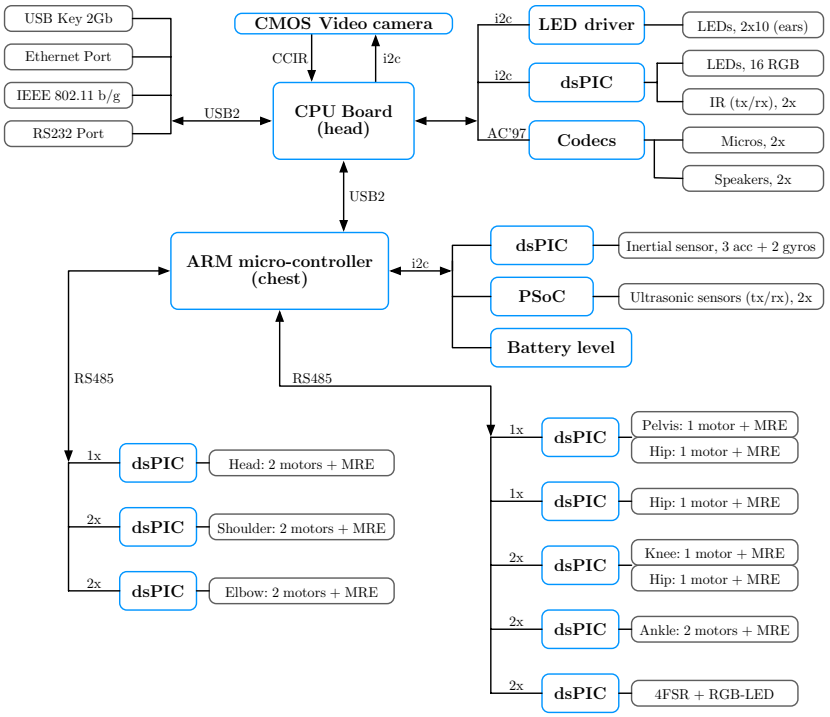
- Walk straight forward and backwards
- Strafe left and right
- Turn left and right
- Walk along an arc of given radius and angle

2.2.1 Main processor (head)

The main processor is a AMD Geode LX-800, working at 500 MHz.

Features:

- 64 kB Instruction / 64 kB Data L1 cache and 128 kB L2 cache
- 400 MHz DDR Memory Controller
- Graphics
- Integrated FPU with MMX and 3DNow! instruction sets
- 128-Bit Advanced Encryption Standard (AES)



after Aldebaran documentation

Figure 2.4: Hardware block diagram

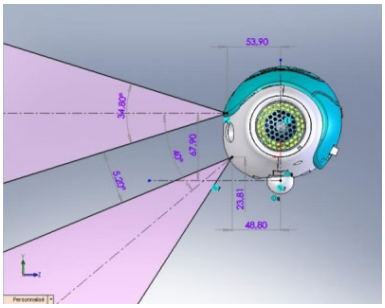
The CPU is extended with 256 MB RAM, and a 2 GB Flash USB drive as permanent storage. To connect the robot to the network, a 100 Mbit ethernet port and a Wi-Fi IEEE 802.11 b/g card are provided.

Apart of the fact that the CPU architecture is quite outdated and the slow cycle rate, compared to nowadays embedded systems, the main drawback is the limited cache size and the poor memory bandwidth. This leads to a severe limitation of algorithms and possible approaches to solve the pending tasks, especially for the image processing.

2.2.2 ARM micro-controller (chest board)

The chest board is equipped with an ARM micro-controller, which supports the communication to and from the distributed hardware units (that is, the motor boards, the inertial unit, and the other sensors and actors). The communication between these two processors is handled via a proprietary format over a USB bus, thus requiring to use the software framework provided by Aldebaran. Currently this framework is executed on the main processor. This leads to a base load of approximately 20 percent, even if the robot is idle.

2.2.3 Video cameras



© Aldebaran

Figure 2.5: Camera locations

Starting with the Nao v3 RoboCup™ version, the Nao integrates two cameras in its head. The upper camera is located at the forehead, as initially designed. The lower camera, located at the mouth, has been integrated after the RoboCup™ 2008 as it became obvious that with only the upper camera, it is not easily possible to see the ball if it lies right before the robot's feet. Thus it was not possible to lineup the robot, in a reasonable amount of time, to kick the ball in a certain direction. As both cameras share the same data bus, only one camera can be used at a time. Switching the cameras is possible but takes about 0.4 seconds, or up to 12 frames. The two identical cameras provide VGA (640×480), YUV422 images at a maximum rate of 30 Hz, with a 58° (diagonal) field-of-view. The placement of the cameras in the robot's head is shown in Figure 2.5.

2.2.4 Inertial unit

The inertial unit, located in the chest, consists of a two-axis gyrometer and a three-axis accelerometer. To determine the torso orientation an Aldebaran algorithm has be implemented in the controller board of the inertial unit. The algorithm uses the accelerometer

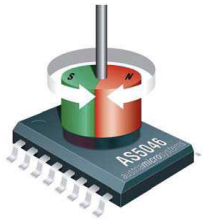
for the static case, when only the gravitational acceleration can be measured. If the unit detects motion, the gyrometer gets used as it has a very good dynamic behaviour. However, the integration of the gyrometer creates a bias of the computed angle, so in the dynamic case, a sensor fusion of computed angle from the accelerometer and the gyrometer is done to reduce this bias.

For the localization task, the torso orientation gets relevant to compute the translation and rotation of the camera (see Subsection 5.2.1).

2.2.5 Joints

The motorized joints are the only means to move the robot. A sequence of motions of several joints is required to perform a single, stable step. The algorithm, currently in use, is developed by Aldebaran and integrated into their middleware. Each joint has at least one magnetic rotary encoder.

These sensors are small chips integrated into the joint, that detects the angle of rotation with a magnet, mounted outside the chip on the joint axis. The relative value provided by this sensor is quite precise and has a small error. However, as the absolute value depends on the orientation of the magnet and therefore on the assembly of the joint axis, calibration is required before shipping, and, if necessary, after severe drops.



© Austria Microsystems

2.2.6 Software framework

Figure 2.6 shows the three basic layers of the software framework. The base is set by the operating system. As second layer acts Aldebaran's NaoQi as middleware, and the third layer builds the team software. The middleware provides only robot specific functionality, like motion, or proprietary hardware access. Hence, the team software has also a tight bound to the operating system.

Operating system

The operating system of the Nao is a modified OpenEmbedded Linux, called OpenNao. Currently (NaoQi v1.3.17), a patched 2.6.22.19-rt kernel is in use. The patches include AMD Geode specific changes, a camera and LED driver, and the realtime preemptive (RT PREEMPT) patch by Ingo Molnar [Mol09]. This patch allows to enforce the realtime policy that is required for motion composition.

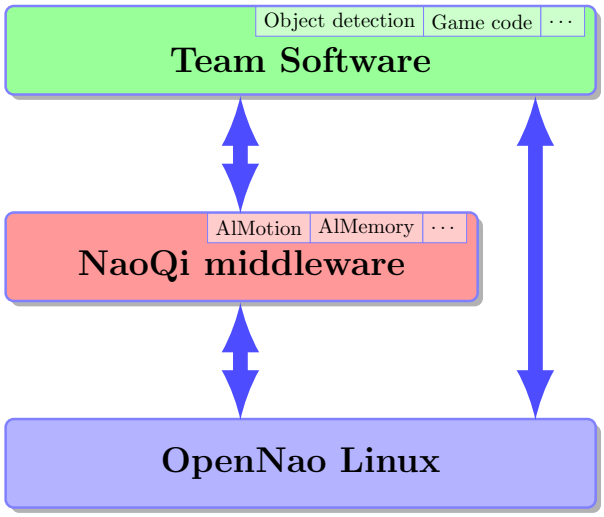


Figure 2.6: Software framework

Aldebaran's middleware — NaoQi

The middleware provided by Aldebaran, called NaoQi, handles the proprietary communication with the ARM micro-controller, and the connected hardware (see Figure 2.4). NaoQi provides a distributed environment, that allows several distributed instances, each containing several software modules to communicate with each other, to share data with each other, and to remotely execute procedures.

To do this, NaoQi provides libraries for several programming languages (C++, Python, Urbi) and operating systems (Linux, Mac OS, Windows). To share information between the instances, the http-based SOAP[2] is used. The SOAP communication is transparent to the user. However, facilitating a distributed environment requires a lot of resources and processing power.

Team software

On top of the NaoQi middleware, a software framework, called FAWKES[3], controls the Nao at the team ZaDeAt. FAWKES is an open source robotics software framework. The initial development was done at the RWTH Aachen [Nie09]. The communication with NaoQi is established via a module that is integrated into the NaoQi framework (see Subsection 5.2.9).

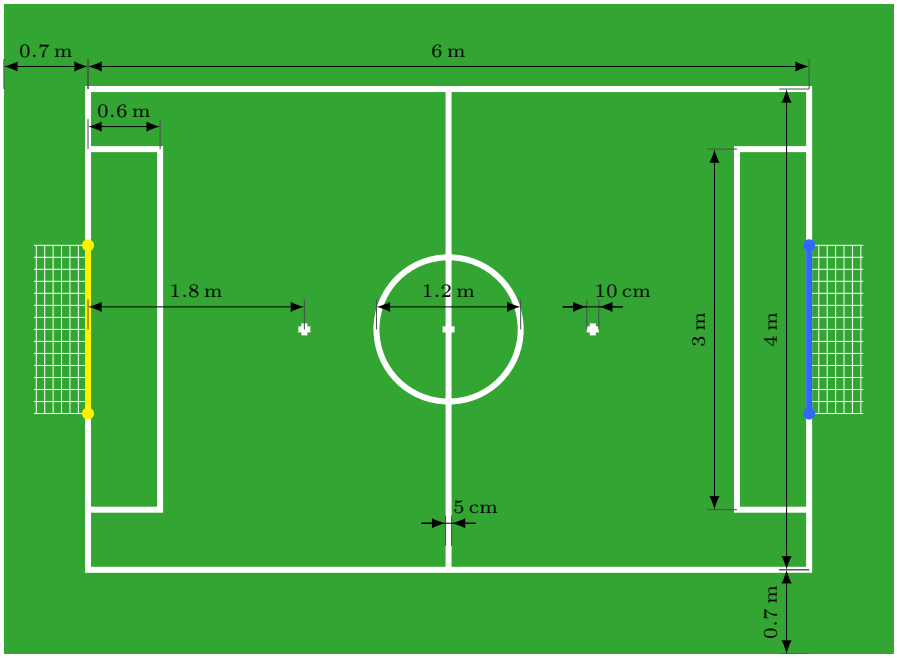
[2]Simple Object Access Protocol – `http://www.w3.org/TR/soap/`
[3]`http://www.fawkesrobotics.org/`

2.3 Gameplay

This section gives an overview of the rules of the standard platform league. The complete rule set can be found in [LTC09c].

2.3.1 Environment

The field is build on a carpet area with a total length of 7.4 m and a total width of 5.4 m. The carpet is a green standard industrial carpet, as used for exhibitions and such, with a thickness of about (2–3) mm. The field lines, as depicted in Figure 2.7, are usually of simple duct tape. The surface of duct tape, however proofed to be rather slippery compared to the field, which sometimes lead to unintended and unpredictable twists, for example, if one foot is on the duct tape and the other is on the carpet.



source: SPL rule book

Figure 2.7: Field layout according to the SPL rules 2009

The goals are either yellow or sky-blue. The side poles have a diameter of 100 mm and a height of 800 mm, the top bar has a diameter of 50 mm. The distance between the side poles is 1.4 m.

The lighting conditions depend on the actual competition site. Only ceiling lights may be used. The competition at the German Opens 2009 showed that constant lighting

conditions are still crucial for a proper soccer games. The objects on the soccer field of the SPL are color coded. The ball is orange, the goals are sky-blue and yellow, and the robots are either red or blue, thus, the object detection rely for a big part on a correct color classification (see Subsection 5.2.2). The exhibition hall in Hanover, the venue of the German Opens 2009, has skylights. On days with scattered clouds, the color calibration proofed useless, as the color temperature changed with the coverage of the sun. In the end the skylights had to be covered, to make a proper gameplay possible.

2.3.2 Game process

Currently a team consists of three robots, of which one is dedicated as goal keeper. The game is played over two half times, each last for ten minutes, with a ten minutes half-time break. In addition, each team can call for one timeout per game. A timeout will interrupt the game for, at most, five minutes, allowing the teams to modify the code, replace robots or adapt the color calibration.

At the kick-off, the robots can take up to 45 seconds to reach their legal positions. If they are unable to reach their positions, they will be placed manually by the assistant referees to pre-defined positions. However, the pre-defined positions are worse than the optimal positions that can be reach by autonomous positioning.

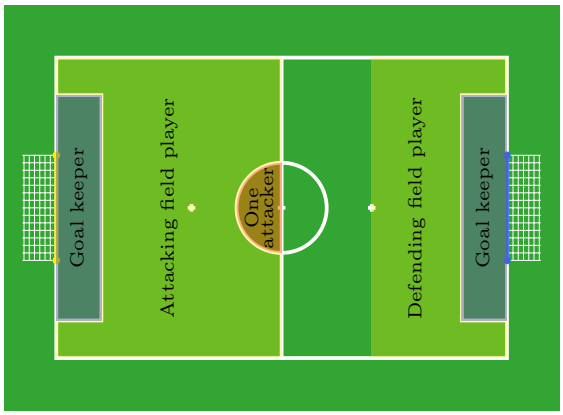
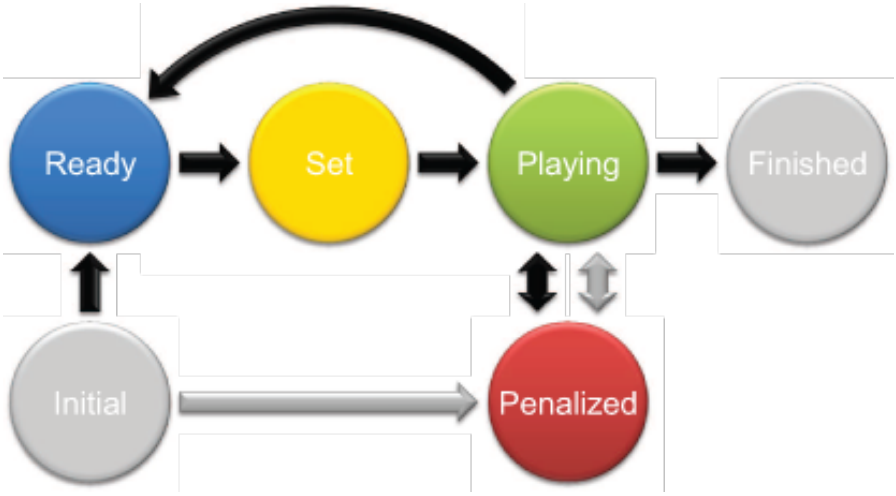


Figure 2.8: Kick-off legal positions

The color differentiation between red robots and the ball, and on the other side, between blue robots and the sky-blue goal is particular tricky. To ensure a fair game, all color parts of the robots are swapped during half time. This means the opponents play one half time as red team, defending the yellow goal, and the other half time as blue team, defending the sky-blue goal. A good part of the half-time break is therefore occupied with the replacement of the color parts.

The robot control software has to model a simple state machine as in Figure 2.9. State transmissions happens either via commands from the GameController[4] over the wireless network (black edges in Figure 2.9), or via short chest button activations (grey edges in Figure 2.9).



source: SPL rule book

Figure 2.9: Basic SPL gamestates

[4] An open source game controller `http://www.tzi.de/spl/bin/view/Website/Downloads`

Chapter 3

Mathematical basics

Self-localization describes the task of finding the own pose, that is, translation and orientation, relative to a known point. In the robotic domain this implies to compute sensory input by applying appropriate algorithms. This also implies that a mathematical description of the problem is required.

This chapter outlines the mathematical background required to tackle the problem of self-localization in a mobile robot environment. Section 3.1 describes the fundamentals of Section 3.2, "Denavit–Hartenberg transformation". Denavit–Hartenberg transformations are used to calculate the camera location based on the geometry of the robot and the current joint sensor readings. Section 3.3 outlines the algorithms used for back-projection, the distance calculation of a point in the world based on the camera location and a point of the camera image. The last Section 3.4 explicates the Kalman filter, an optimal state estimator, used to estimate the pose of the robot based on the output of the self-localization algorithm and odometry readings.

The self-localization algorithms used are described in Chapter 4.

3.1 Homogeneous coordinates and transforms

The description of homogeneous coordinates and transforms given here is an abstract of a report by Alonzo Kelly [Kel06].

3.1.1 Basics

A point in the three-dimensional space can be represented as a column vector:

$$p_1 = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} = (x_1\ y_1\ z_1)^\mathsf{T}$$

An operator is any process, that maps points onto other points. Many operators can be represented as 3×3-matrices:

$$\begin{aligned} p_2 = \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix} = \mathrm{Op} \cdot p_1 &= \begin{pmatrix} op_{xx} & op_{xy} & op_{xz} \\ op_{yx} & op_{yy} & op_{yz} \\ op_{zx} & op_{zy} & op_{zz} \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix} \\ &= \begin{pmatrix} op_{xx}\, x_1 + op_{xy}\, y_1 + op_{xz}\, z_1 \\ op_{yx}\, x_1 + op_{yy}\, y_1 + op_{yz}\, z_1 \\ op_{zx}\, x_1 + op_{zy}\, y_1 + op_{zz}\, z_1 \end{pmatrix} \end{aligned}$$

Simple operators, like scale, or rotation, can be generated with 3×3-matrices by a suitable choice of the entries in the matrices. However, a very simple and often used operator cannot be represented: translation. That is, there is no 3×3-matrix, that adds a constant vector p_{trans} to p_1. Such a translation could be represented as a vector addition: $p_2 = p_1 + p_{trans}$. As p_{trans} is supposed to be independent of p_1, the translation cannot be represented (in general) by a 3×3-matrix.

3.1.2 Homogeneous transforms

To overcome this limitation, the point in the three-dimensional space is projected into a four-dimensional space. The fourth element represents a kind of scale factor: $p_{1,4D} = (x_1\ y_1\ z_1\ w_1)^\mathsf{T}$. The point in the three-dimensional space can be found by dividing the elements by the scale factor:

$$p_{1,3D} = \left(\frac{x_1}{w_1}\ \frac{y_1}{w_1}\ \frac{z_1}{w_1}\right)^\mathsf{T}$$

Points are usually represented with a scale factor of 1: $p_1 = (x_1\ y_1\ z_1\ 1)^\mathsf{T}$. Directions, in terms of points at infinity, can be represented by using a scale factor of 0: $q_1 = (x_1\ y_1\ z_1\ 0)$.

The operators, that can be represented as matrices, for homogeneous points are of the dimension 4×4. A translation can be represented as:

$$p_2 = p_1 + p_{trans} = \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} + \begin{pmatrix} x_{trans} \\ y_{trans} \\ z_{trans} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & x_{trans} \\ 0 & 1 & 0 & y_{trans} \\ 0 & 0 & 1 & z_{trans} \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

3.1.3 Basic operators

The basic operators are translation along, and rotation about any of the three axis of the reference coordinate frame. The operators take a point in the reference coordinate frame, operate on it, and supply the result expressed in the same coordinate frame. A chain of operators is written in right-to-left order, because this it the order in which they are applied. The order is important because matrix multiplication is not commutative.

Translation

$$\mathrm{Trans}(x, y, z) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.1}$$

Rotation about the x-axis

$$\mathrm{Rot_X}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.2}$$

Rotation about the y-axis

$$\mathrm{Rot_Y}(\phi) = \begin{pmatrix} \cos\phi & 0 & \sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.3}$$

Rotation about the z-axis

$$\mathrm{Rot_Z}(\psi) = \begin{pmatrix} \cos\psi & -\sin\psi & 0 & 0 \\ \sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.4}$$

A rotation followed by a translation can be represented in a single matrix.

3.1.4 Interpretation of homogeneous transformations

The interpretation of homogeneous transformations can be threefold:

Operators

As already stated above, the homogeneous transformations can be interpreted as operator, to map a point p to another point p' within the same coordinate frame.

Multiple operator can be joint to a cumulative operator by simple matrix multiplication:

$$\mathrm{Op}' = \mathrm{Op}_n \ \mathrm{Op}_{n-1} \ \cdots \ \mathrm{Op}_1$$

The operators are written in right-to-left order because this is the order in which they are applied to the point p. As the matrix multiplication is not commutative, this order is important.

Coordinate Frames

Homogeneous transformations can also be used to represent coordinate frames relative to a reference frame. Let

$$i = (1\ 0\ 0\ 0)^\mathsf{T} \qquad j = (0\ 1\ 0\ 0)^\mathsf{T} \qquad k = (0\ 0\ 1\ 0)^\mathsf{T}$$

be the directions, and

$$o = (0\ 0\ 0\ 1)^\mathsf{T}$$

be the origin of the reference coordinate frame.

Those vectors could be grouped to the identity matrix

$$(i\ j\ k\ o) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = I$$

By applying an arbitrary number of operators, one can transform $I \rightarrow I'$

$$I' = \left(i'\ j'\ k'\ o'\right) = \mathrm{Op}_n \ \mathrm{Op}_{n-1} \ \cdots \ \mathrm{Op}_1 \ I$$

where i', j', and k' represent the directions, and o' the origin of the new coordinate frame relative to the reference frame.

Coordinate transformations

The third interpretation of homogeneous transformations is the transformation of coordinates relative to one frame into coordinates that are relative to another frame. Given, that a homogeneous transformation can be used to represent a coordinate frame relative to a reference frame, this very same transformation can be used to transform coordinates of the resulting coordinate frame back to coordinates of the reference frame. Let I' be the coordinate frame, named 'b' and I be the reference coordinate frame, named 'a'. Let $p^b = \begin{pmatrix} x^b & y^b & z^b & 1 \end{pmatrix}^\mathsf{T}$ be an arbitrary point represented in the coordinates of the frame 'b'. The same point p^a, represented in coordinates of the frame 'a' can easily be found.

$$p^a = I' \, p^b$$

Sometimes the transformation of coordinates from the reference frame (a) into the derived frame (b) is required. For this case the inverse of a homogeneous transformation is required.

$$p^b = I'^{-1} \, p^a$$

3.1.5 Inverse of a homogeneous transformation

The basic operators described above, and combinations thereof, are assembled by a common pattern:

$$\mathrm{Op} = \left(\begin{array}{ccc|c} & R & & \underline{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \qquad \text{with} \quad \begin{array}{ll} R & \ldots 3 \times 3 \text{ rotation matrix} \\ \underline{t} & \ldots \text{translation vector} \end{array}$$

The inverse of those kind of operators can be found as:

$$\mathrm{Op}^{-1} = \left(\begin{array}{ccc|c} & R^\mathsf{T} & & -R^\mathsf{T}\,\underline{t} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.5}$$

3.2 Denavit–Hartenberg transformation

3.2.1 Motivation

In the field of robotics you often face the question of where a certain point is located relative to a second point, or relative to the origin of a coordinate frame, for example, the position of the camera relative to the ground.

Imagine a simple robot arm with one link of a certain length that is able to turn around one axis. The question is now, what is the position of the end point of the link rel-

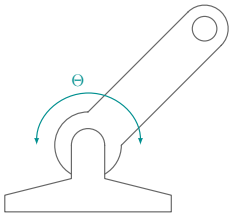


Figure 3.1: A single link

ative to the mount of the arm for a given turning angle θ. The complexity increases dramatically as soon as there are several joints in a chain. So-called Denavit–Hartenberg-Tranformations are often used to address this problem, by transforming one coordinate frame into another. In the simple example from above, the origin of one coordinate frame would sit at the end of the link (the end-effector) and a second coordinate frame would be located in the center of the rotation axis with one axis pointing along this axis. Note that the orientation of the coordinate frame of the end-effector can be chosen freely. The

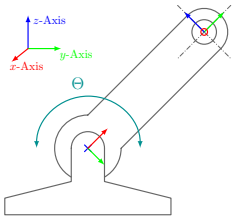


Figure 3.2: A single link with assigned coordinate frames

DH-Transformation can now be used to transform a point (for example, the origin) of the end-effektors coordinate frame into the axis coordinate frame and subsequently into a global coordinate frame.

3.2.2 Minimal line representation

As links between joints or links between joints and end-effectors can be seen as lines, a supporting line representation will be necessary. In general, a line in the Euclidien Space (E^3) has four degrees-of-freedom, therefore at least four parameters are required to represent all possible lines.

A line $L(p, d)$ is completely defined by the ordered set of two vectors [Wik08]. A point vector p, indicating the position of an arbitrary point on L and a direction vector d giving the line a direction as well as a sense. Each point x on the line is given a parameter value t that satisfies: $x = p + td$. The parameter t is unique once p and d are chosen. However, the representation $L(p, d)$ is not minimal, because it uses six parameters while only four would be required.

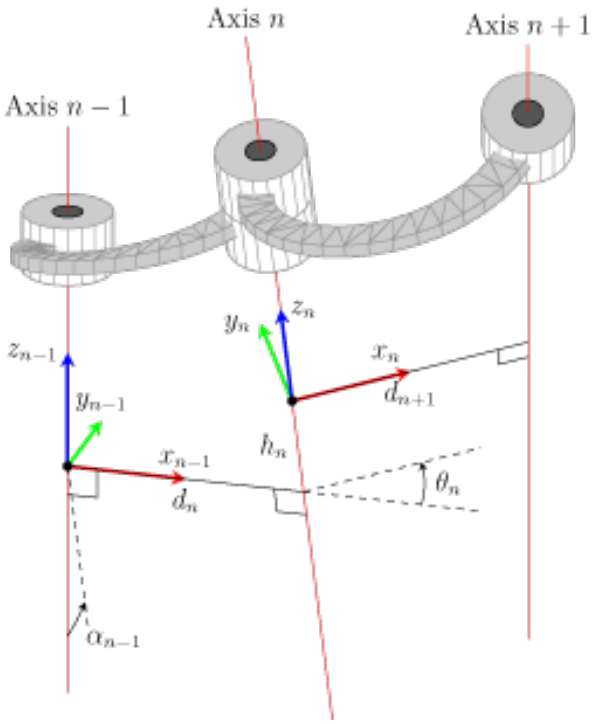
In 1955 Jaques Denavit and Richard S. Hartenberg presented the first minimal line representation [DH55] which is now widely used. The *common normal* between two lines was the main geometric concept that allowed J. Denavit and R.S. Hartenberg to find a minimal line representation. Their reference line is the z-axis of a global coordinate frame. Given that a line L has a direction, the following four parameters are required: The distance d, the azimuth α, the twist θ and the height h. The literature contains alternative formulations, differing mainly in the conventions for signes and reference axes. However all these formulations are equivalent and represent the line L by two translational and two rotational parameters. This property emphasis the use of homogenous transformations (see Section 3.1).

3.2.3 Coordinate frame assignment

To use this line representation, every joint has an attached coordinate frame. By convention the z-axis of every such joint frame is the joint axis. As the end-effector has no joint axis, the direction of the z-axis can be chosen for end-effectors. The x-axis is parallel to the common normal or if there is no common normal, $x_n = z_{n-1} \times z_n$. The y-axis follows the x- and z-axis by defining it to be a right-handed coordinate frame: $y_n = x_n \times z_n$. For simplicity the y-axis are not shown in Figure 3.3 and the frames would actually be at the height of the corresponding links:

Homogenous coordinate frame transformation

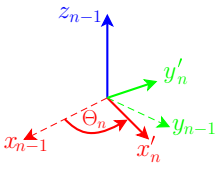
When going from the coordinate system Z_{n-1} to the coordinate system Z_n, one has to turn around the z_{n-1}-axis to bring the x_{n-1}-axis parallel to the x_n-axis:



source: `http://www.fauskes.net/nb/threedill/`

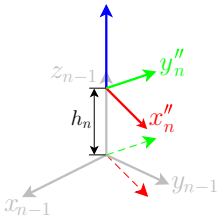
Figure 3.3: Coordinate frame assignment

$$\mathrm{Rot}_{z_{n-1}}(\theta_n) = \begin{pmatrix} \cos\theta_n & -\sin\theta_n & 0 & 0 \\ \sin\theta_n & \cos\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



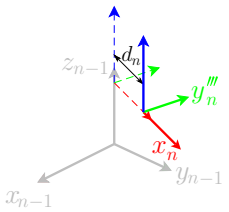
Translate along the old z-axis, to move the old coordinate frame origin to the height of the new frame:

$$\mathrm{Trans}(0,0,h_n) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & h_n \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



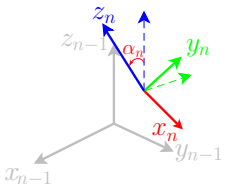
Translate along the new x-axis, to get the distance along the common normal of the two z-axes:

$$\text{Trans}(d_n, 0, 0) = \begin{pmatrix} 1 & 0 & 0 & d_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



And finally rotate around the new x-axis, to align the new z-axis with the center of rotation of the next joint:

$$\text{Rot}_{x_n}(\alpha_n) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



This gives (right-to-left order):

$$\begin{aligned} {}^{n-1}T_n &= \text{Rot}_{x_n}(\alpha_n)\ \text{Trans}(d_n, 0, h_n)\ \text{Rot}_{z_{n-1}}(\theta_n) \\ &= \begin{pmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & d_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & d_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & h_n \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{3.6}$$

As the z-axes are equal to the rotation axes of the joints θ is the parameter that changes, when the position of the joint changes. The other three parameters (d, h and α) are usually defined by the hardware configuration of the robot and fixed, hence the equation 3.6 can be written as:

$${}^{n-1}T_n = \begin{pmatrix} \cos\theta_n & -\sin\theta_n & 0 & 0 \\ \sin\theta_n & \cos\theta_n & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & d_n \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & h_n \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.7}$$

of where the second matrix can be pre-calculated to reduce the computational load.

3.3 Back projection

Back projection is the task of transforming a point on the camera plane, that is, a pixel of the camera image, back to a plane in the environment, for the purpose of this thesis

the ground plane. With this transformation the coordinates of detected objects in the camera image, relative to the robot, for example the ball, or points of the field lines, can be calculated. The exact reference point for objects on the ground plane is the projection of the center of the camera, this is, the principal point *pp*, along the falling line. In Figure 3.4 (right) this would be the origin of the coordinate frame.

3.3.1 Pan/tilt cameras

A very simple case for the back projection is a pan/tilt camera, mounted in parallel to the ground plane at a constant height h_c. A pan/tilt camera has two degrees of freedom. It can turn the camera left and right, and up and down. The coordinates (x_s, y_s) of a

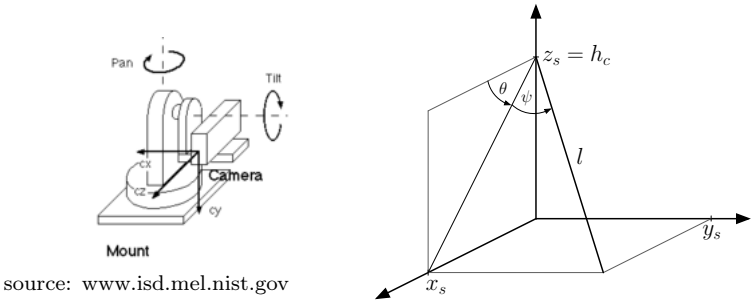


Figure 3.4: Back projection of a pan/tilt camera

point in the world, relative to the camera, can be found as:

$$x_s = l \cos\psi \cos\theta = \frac{h_c}{\tan\theta} \tag{3.8}$$

$$y_s = l \sin\psi = \frac{h_c \tan\psi}{\sin\theta} \tag{3.9}$$

Intrinsic camera parameters In order to calculate ψ and θ a mathematical description of the camera properties is required, the so called intrinsic camera parameters. For the purpose of pan/tilt cameras, they consists of horizontal and vertical field-of-view (fov_h, fov_v), the horizontal and vertical resolution of the camera (res_h, res_v) and the principal point *pp*, that is, the center of the camera plane in pixel (pp_x, pp_y).

The height h_c of the pan/tilt unit is usually fixed by the hardware design and thus constant. To get the variable angles ψ and θ the current setting of the pan/tilt unit

(Ψ, Θ), the intrinsic camera parameters, and a point of interest in the image (i_x, i_y) are required:

$$\psi = (pp_x - i_x)\frac{fov_h}{res_h} + \Psi \tag{3.10}$$

$$\theta = (i_y - pp_y)\frac{fov_v}{res_v} + \Theta \tag{3.11}$$

Note: The assumption of the, per design, fixed height h_c holds only for variable pan/tilt settings if the rotation axis of the pan/tilt unit are aligned with the principal point *pp* of the camera. In Figure 3.4 (left) this assumption would not hold.

3.3.2 Homography

It showed that the pan/tilt model was not sufficient for the use with the Nao, even though the head itself (and therefore the camera) has only two degrees of freedom. When the torso rotates about its x, or y axis, for example when the robot shifts its weight from one foot to the other, the base of the pan/tilt unit is no longer parallel to the ground plane, and the camera therefore rotated about all three axis.

For this case a homography, a matrix operator that generally transforms one plane onto another, is used to transform the image plane back to the ground plane. The composition of the homography is simple, at least if the target plane is the ground plane. First the matrix that transforms a point in the world to the image plane is compiled. It consists of the intrinsic (see above) and the extrinsic camera parameters. The extrinsic camera parameters are the rotation and translation of the camera relative to the ground plane. The intrinsic parameters can be found as [HZ03, pp. 153–157]:

$$K = \begin{pmatrix} \frac{res_h}{2\cdot\tan(0.5\cdot fov_h)} & 0 & pp_x \\ 0 & \frac{res_v}{2\cdot\tan(0.5\cdot fov_v)} & pp_y \\ 0 & 0 & 1 \end{pmatrix} \tag{3.12}$$

For the extrinsic parameters a 3×3 rotation matrix R and the translation vector $T = (x_c, y_c, h_c)^\intercal$ are required, where x_c, and y_c are the offsets of the camera relative to the origin of the robot, for simplicity zero, and h_c the height above ground. The 3×4 projection matrix P is then:

$$P = K\,R \begin{pmatrix} 1 & 0 & 0 & x_c \\ 0 & 1 & 0 & y_c \\ 0 & 0 & 1 & h_c \end{pmatrix} \tag{3.13}$$

With this projection matrix a point (x_w, y_w, z_w) in the world can be transformed to a point (x_i, y_i) in the image:

$$\begin{pmatrix} x_i \\ y_i \\ s_i \end{pmatrix} = P \begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix} \tag{3.14}$$

Note: If $s_i \neq 1$ the result vector has to be scaled accordingly: $\left(\frac{x_i}{s_i}, \frac{y_i}{s_i}\right)$.

Ground plane assumption

The back projection, however, cannot be formulated in general as the projection matrix P cannot be inverted. Under the assumption, that all points of interest, are points on the ground plane, that is, $z_w = 0$, one can see, that the third column of the projection matrix P is of no use, as its elements gets always multiplied with z_w. In this case a reduced projection matrix P' can be compiled from the first, second and fourth column of P:

$$P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix} \rightarrow P' = \begin{bmatrix} p_1 & p_2 & p_4 \end{bmatrix} \tag{3.15}$$

and the world coordinates (with $z_w = 0$) of the image point (x_i, y_i) can be found as:

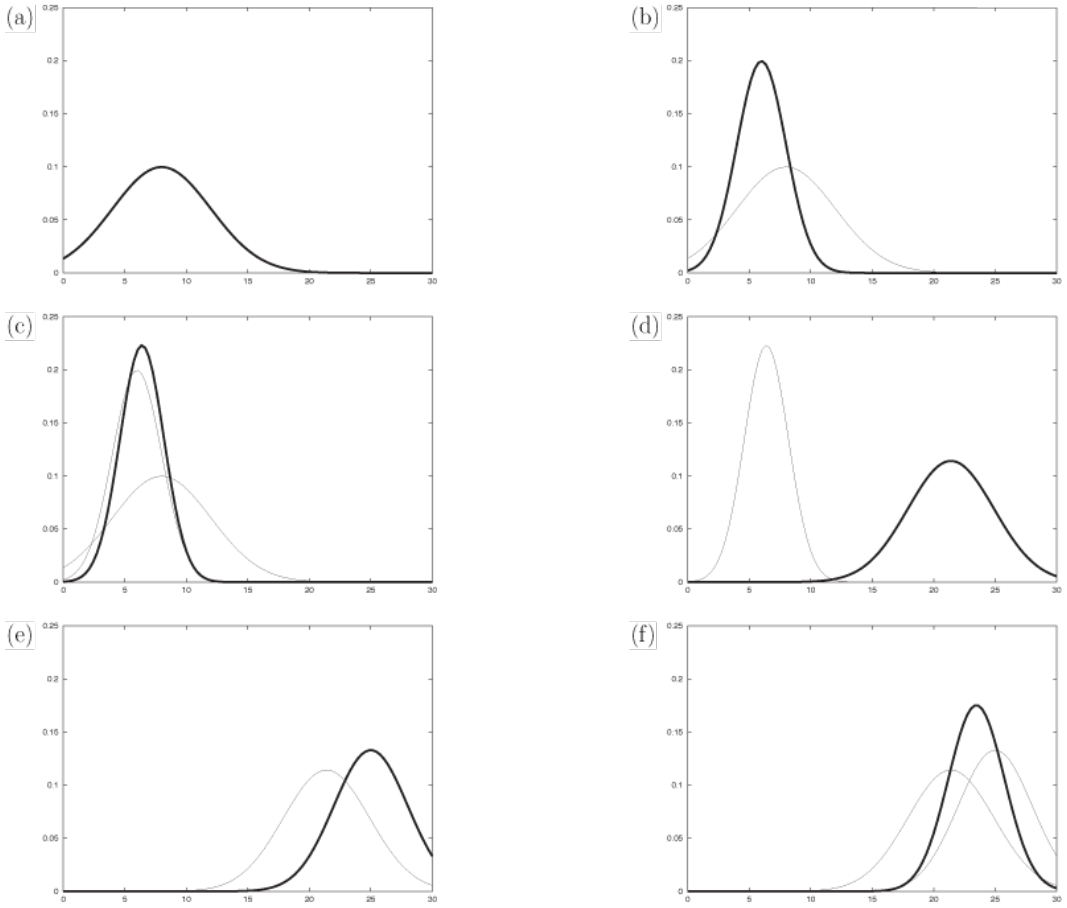
$$\begin{pmatrix} x_w \\ y_w \\ s_w \end{pmatrix} = P'^{-1} \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} \tag{3.16}$$

Note: If $s_w \neq 1$ the result vector has to be scaled accordingly: $\left(\frac{x_w}{s_w}, \frac{y_w}{s_w}\right)$.

3.4 Kalman filter

The Kalman filter [Kal60], [TBF05, pp. 40-53] is an optimal recursive data processing algorithm [May79]. It is a set of mathematical equations that provides an efficient means to estimate the state of a process, based on the previous (or initial) state, a set of control commands, and a set of measurements. The basic idea is to predict the state of a process based on the previous state and the control commands, and, in a different step, to correct those predictions based on the measurements taken.

The following example should be used to describe the functionality of a Kalman filter. Imagine a train that is equipped with a device that can estimate the distance to the next



source: [TBF05]

Figure 3.5: Illustration of Kalman filters: (a) initial belief, (b) a measurement (in bold) with the associated uncertainty, (c) belief after integrating the measurement into the belief using the Kalman filter algorithm, (d) belief after motion to the right (which introduces uncertainty), (e) a new measurement with associated uncertainty, and (f) the resulting belief.

train station, for example by using GPS coordinates. For this example the update of the distance to the next station might be available only every five minutes. At time t_0 the train is at a known location and the uncertainty (the variance σ^2_{pred}) of the distance to the next station is very low. As the train start to travel to the next station, the control commands of the operator can be used to predict the current distance to the next station. The error (σ^2_{cmd}) between control command and execution will increase the uncertainty of the distance to the next station prediction, up to the maximum uncertainty right before the next distance measure update. A distance measurement is of course also error-prone (σ^2_{meas}), hence the measurement should not be taken as granted but merged with the distance prediction based on the variances σ^2_{pred} and σ^2_{meas}, to get the optimal estimate of the current distance to the next station. Figure 3.5 illustrates the function of the Kalman filter.

One attribute, that makes the Kalman filter optimal, is the fact, that any number of measurements and measurement devices can be used to correct the current prediction. In the example above units to measure the distance to the next station might use GPS, markers along the rails, or the round trip time of radio signals. All of them might have different, and changing, update cycles and also variances that change over time (for example based on the number of satellites that can be used to calculate the current position). GPS is unavailable in tunnels, markers might be expensive and therefore seldom, plus they don't provide any updates at all if the train stops, but still, all of those measurements can be merged with the current prediction, based on the current variances, as soon as they are available.

3.4.1 Basic assumptions

The original Kalman filter depends on the following basic assumptions. The process is governed by a linear stochastic difference equation:

$$x_k = A\,x_{k-1} + B\,u_{k-1} + w_{k-1} \tag{3.17}$$

with the measurement:

$$z_k = H\,x_x + v_k \tag{3.18}$$

The process and measurement noises (w_{k-1}, v_k) are independent of each other, white, and with normal probability distributions (Gaussian):

$$p(w) \sim \mathcal{N}(0, Q) \tag{3.19}$$
$$p(v) \sim \mathcal{N}(0, R) \tag{3.20}$$

In (3.17), $x \in \mathbb{R}^n$ represents the process state vector (with n elements), the $n \times n$-matrix A relates the proces state at the previous time step $k-1$ to the current time step k, and the $n \times l$-matrix B relates the control input $u \in \mathbb{R}^l$ to the current state. In (3.18), the

$m \times n$-matrix H relates the process state to the measurement $z \in \mathbb{R}^m$. The process noise covariance matrix Q is of the size $n \times n$, the and measurement noise covariance matrix R is of the size $m \times m$.

3.4.2 Time-discrete Kalman filter algorithm

The algorithm is separated into two parts. The first set of equations calculates the *a priori* estimate (also known as prediction step, or time update), the second set of equation acts as a form of feedback, to incorporate new measurements into to the *a priori* estimate to obtain an improved *a posteriori* estimate (also known as correction step, or measurement update).

Prediction step

$$\hat{x}_k^- = A\,\hat{x}_{k-1} + B\,u_{k-1} \tag{3.21}$$
$$P_k^- = A\,P_{k-1}\,A^\intercal + Q \tag{3.22}$$

In the prediction step the *a priori* estimate is calculated only based on the previous state $\hat{x}_{k-1}$ and the optional control input u_{k-1}. Note that the previous system state may be either another *a priori* estimate (if there were no new measurements), or a *a posteriori* estimate. The uncertainty of the estimate increases, based on the previous uncertainty and the process noise covariance matrix Q. The more prediction steps occur without a correction step, the further the uncertainty of the estimate will increase.

Correction step

$$K_k = P_k^-\,H^\intercal\,\left(H\,P_k^-\,H^\intercal + R\right)^{-1} \tag{3.23}$$
$$\hat{x}_k = \hat{x}_k^- + K_k\,\left(z_k - H\,\hat{x}_k^-\right) \tag{3.24}$$
$$P_k = (I - K_k\,H)\,P_k^- \tag{3.25}$$

The $n \times m$-matrix K_k is called the gain or blending factor, that minimizes the *a posteriori* error covariance P_k. Equation (3.24) computes the *a posteriori* state estimate as linear combination of the *a priori* estimate and the weighted (K_k) difference between the actual measurement z_k and the measurement prediction $H\,\hat{x}_k^-$, also known as the measurement innovation or residual.

3.4.3 Sensor fusion

The Kalman filter can as well be used to fuse the measurements of several sensors into a state estimation with a minimized error [Gut00]. Each sensor provides measurements

with an expected value and a variance, or, in the n-dimensional case an expectation vector and a covariance matrix. The influence of each measurement to the resulting state estimate depends on the respective variances:

$$\sigma = \left(\frac{1}{\sigma_1^2} + \frac{1}{\sigma_2^2} \right)^{-1} \tag{3.26}$$

$$\mu = \sigma \left(\frac{\mu_1}{\sigma_1^2} + \frac{\mu_2}{\sigma_2^2} \right) \tag{3.27}$$

or, in the n-dimensional case, with the measurement vectors m_1 and m_2:

$$p(m_1) \sim \mathcal{N}(0, \Sigma_1)$$
$$p(m_2) \sim \mathcal{N}(0, \Sigma_2)$$

$$\Sigma = \left(\Sigma_1^{-1} + \Sigma_2^{-1} \right)^{-1} \tag{3.28}$$

$$m = \Sigma \left(\Sigma_1^{-1} m_1 + \Sigma_2^{-1} m_2 \right) \tag{3.29}$$

3.4.4 Transformation of density functions

The task of transforming a density function from one domain into another is inherent to the Kalman filter, but it is also relevant for the sensor data processing. Whenever a sensor provides its data in a different domain as the rest of the system, for example a laser scanner provides distance and angle to the closest surfaces, but the rest of the system might use cartesian coordinates, the density function of the sensor data has to be transformed. In general, some unit $u \in \mathbb{R}^n, u \sim \mathcal{N}(\mu_u, \Sigma_u)$, and a transformation $\mathrm{f} : \mathbb{R}^n \to \mathbb{R}^m$ with $\mathrm{f}(u) = x, x \sim \mathcal{N}(\mu_x, \Sigma_x)$ are given, and the expected value μ_x and Σ_x are unknown.

According to [Gut00], this problem can be solved for linear transformations, for which $\mathrm{f}(u)$ can be written as $\mathrm{f}(u) = Au + b$, where A is a constant $m \times n$-matrix:

$$\mu_x = A\,\mu_u + b \tag{3.30}$$

$$\Sigma_x = A\,\Sigma_u\,A^\intercal \tag{3.31}$$

For non-linear transformations the problem can be approximated by the use of Taylor series:

$$\mathrm{f}(u) \approx \mathrm{f}(\hat{u}) + F_{\hat{u}}\,(u - \hat{u})$$
$$F_{\hat{u}} = \frac{\partial \mathrm{f}}{\partial u}\,(\hat{u})$$

The $m \times n$-Jacobian matrix $F_{\hat{u}}$ contains the partial derivatives of f with respect to u, then:

$$\mu_x = \mathrm{f}(\mu_u) \tag{3.32}$$

$$\Sigma_x = F\,\Sigma_u\,F^\intercal \tag{3.33}$$

3.4.5 Extended Kalman filter

The filter was originally defined to calculate state estimates of discrete-time controlled processes that are governed by linear stochastic difference equations. The extended Kalman filter (EKF), however, extends the application of the Kalman filter to processes that require non-linear functions for the prediction and/or the correction step, by linearizing about the current mean and covariance. However, it is to note that the distributions of the random variables (v, w) are no longer normal after their respective non-linear transformations. [TBF05, pp. 54-64]

Let $x_k \in \mathbb{R}^n$ again be the process state vector at the time step k. The process is now governed by the non-linear stochastic difference equation, with a measurement $z_k \in \mathbb{R}^m$:

$$x_k = \mathrm{f}(x_{k-1}, u_{k-1}, w_{k-1}) \tag{3.34}$$
$$z_k = \mathrm{h}(x_k, v_k) \tag{3.35}$$

Prediction step

$$\hat{x}_k^- = \mathrm{f}(\hat{x}_{k-1}, u_{k-1}, 0) \tag{3.36}$$
$$P_k^- = A_k\, P_{k-1}\, A_k^\intercal + W_k\, Q_{k-1}\, W_k^\intercal \tag{3.37}$$

A_k and W_k are Jacobian matrices, that contain the partial derivatives of f with respect to x and w:

$$A_{k_{[i,j]}} = \frac{\partial\, \mathrm{f}_{[i]}}{\partial x_{[j]}}\,(\hat{x}_{k-1}, u_{k-1}, 0)$$
$$W_{k_{[i,j]}} = \frac{\partial\, \mathrm{f}_{[i]}}{\partial w_{[j]}}\,(\hat{x}_{k-1}, u_{k-1}, 0)$$

Correction step

$$K_k = P_k^-\, H_k^\intercal\, \left(H_k\, P_k^-\, H_k^\intercal + V_k\, R\, V_k^\intercal\right)^{-1} \tag{3.38}$$
$$\hat{x}_k = \hat{x}_k^- + K_k\, \left(z_k - \mathrm{h}(\hat{x}_k^-, 0)\right) \tag{3.39}$$
$$P_k = (I - K_k\, H_k)\, P_k^- \tag{3.40}$$

H_k and V_k are again Jacobian matrices, containing the partial derivatives of h with respect to x and v:

$$H_{k_{[i,j]}} = \frac{\partial\, \mathrm{h}_{[i]}}{\partial x_{[j]}}\,\left(\hat{x}_k^-, 0\right)$$
$$V_{k_{[i,j]}} = \frac{\partial\, \mathrm{h}_{[i]}}{\partial v_{[j]}}\,\left(\hat{x}_k^-, 0\right)$$

Chapter 4

Self-localization

This chapter describes the implemented algorithms regarding self-localization. In general, the task of self-localization can be split into two groups. The first group deals with fixed landmark detection, the second group with scan matching, the task of matching a set of detected features on a predefined map of known features.

4.1 Landmark detection

Landmark detection, describes the task of extracting fixed landmarks from the sensor input, and to estimate the own position relative the the known position of those landmarks.

The field of the former standard platform league, had several such landmarks. Color coded beacons were mounted on the sides of the field, and the corner arcs as well as the goals were either blue or yellow. The field of the current standard platform league provides only colored goals as landmarks, that is, one of the goals is yellow and the other one is sky-blue. This limits the chance to see the landmarks and therefore to use it continuously for the localization task.

However, as the field is symmetrically, the color of the goals is the only means to get the absolute orientation on the field. The current design of the goals allows to distinct the left and right side pole of a goal. As the position of the poles is known, the poles can still be used as landmarks, not only to get the global orientation, but also, by applying plain geometry, to estimate the current pose on the field, at least if the robot searches deliberately for either goal.

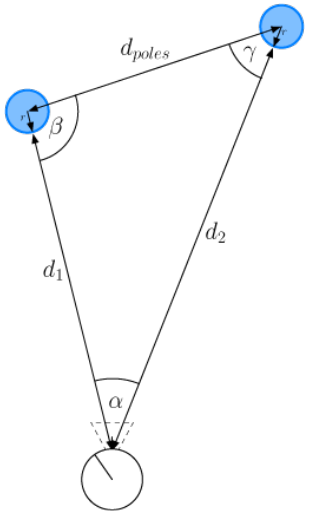


Figure 4.1: Goal poles triangulation

Given, that the lengths d_{poles}, d_1, d_2, and r_{poles}, and the angle α are known, the angles β and γ can be found by applying the law of cosine: $c^2 = a^2 + b^2 - 2\,a\,b\,\cos\gamma$. With the variables in Figure 4.1, β' can be found as:

$$\cos\beta' = \frac{d_1'^2 + d_{poles}^2 - d_2'^2}{2 \cdot d_1' \cdot d_{poles}} \text{ with } d_n' = d_n + r_{poles} \tag{4.1}$$

Experiments have shown that the law of cosine did not hold for the known angle α, or with other words, that $\alpha+\beta+\gamma \neq 2\pi$. This is due to inaccuracies in the object detection, the calculation of the camera location, and the back projection. To limit the resulting error, the angle γ, and thus $\beta'' = 2\pi - \alpha - \gamma$ are calculated as well. The angle β, used for the pose calculation, is then the average of β' and β''.

4.2 Cox algorithm

In 1991, Ingemar J. Cox published [Cox91] an algorithm to estimate the global pose of a robot, based on a predefined map of walls, marked as lines, and the input of a laser range scanner.

The algorithm requires an initial pose $q_0 = (x_{ini}, y_{ini}, \theta_{ini})^\intercal$, a predefined list of lines that map the surrounding walls, and a range scan, that is, a list of relative distances to the closest walls. (1) The initial pose is used to transform the range scan (of relative distances) to the corresponding initial global scan points. (2) For each scan point, the offset, and direction, to the closest line is calculated. (3) Calculate a pose correction

$(\Delta x, \Delta y, \Delta\theta)$, that minimizes the sum of all squared offsets. (4) If the pose correction exceeds some given thresholds $(\epsilon_x, \epsilon_y, \epsilon_\theta)$, use the corrected pose as new initial pose and continue at (1). (5) Calculate the covariance matrix as a means of the quality of the pose estimation.

Initial scan transformation (1)

The initial scan transform is trivial. With the given initial position $l_{ini} = (x_{ini}, y_{ini})^\intercal$ and orientation θ_{ini}, and for a list of scan points $s_i = (x_{rel}, y_{rel})^\intercal$ relative to the current pose do:

$$p_i = \begin{pmatrix} \cos\theta_{ini} & -\sin\theta_{ini} \\ \sin\theta_{ini} & \cos\theta_{ini} \end{pmatrix} s_i + l_{ini} \tag{4.2}$$

With p_i being an initial estimate of the scan point s_i in global coordinates.

Selecting a corresponding line (2)

For each global scan point p_i, find the line in the map that has the shortest euclidian distance d_i. If d_i exceeds some predefined threshold d_{max} the scan point is assumed to be an outlier, and therefore removed from the list of scan points.

For calculating the distances d_i the finite line segments are used, for the next step infinite lines, with $u_i = (u_{ix}, u_{iy})^\intercal$, r_i are used, so that: $z^\intercal u_i = r_i$ holds for arbitrary points z on the line.

Finding the pose correction (3)

Goal of this step is to find a correction $b = (\Delta x, \Delta y, \Delta\theta)^\intercal$ that minimizes the sum of the squared distances of (2). For that, each scan point p_i has to be transformed:

$$\text{trans}(b)(p_i) = \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{pmatrix} (p_i - l_{ini}) + l_{ini} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{4.3}$$

To simplify the task, Cox argued that the angle $\Delta\theta$ should be sufficiently small so that the transformation can be approximated to:

$$\text{trans}(b)(p_i) \approx \begin{pmatrix} 1 & -\Delta\theta \\ \Delta\theta & 1 \end{pmatrix} (p_i - l_{ini}) + l_{ini} + \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \tag{4.4}$$

With this linearization the squared distance of each scan point p_i to the closest line $z^\intercal u_i = r_i$ can be found as:

$$\begin{aligned} err_i = d_i^2 &= (\text{trans}(b)(p_i)^\intercal u_i - r_i)^2 \\ &\approx ((x_{i1}, x_{i2}, x_{i3})\, b - y_i)^2 \end{aligned} \tag{4.5}$$

with:

$$
\begin{aligned}
x_{i1} &= u_{ix} \\
x_{i2} &= u_{iy} \\
x_{i3} &= u_i^\intercal \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} (p_i - l_{ini}) = (p_{ix} - x_{ini}) u_{iy} - (p_{iy} - y_{ini}) u_{ix} \\
y_i &= r_i - p_{ix} u_{ix} - p_{iy} u_{iy}
\end{aligned}
$$

The sum of the squared distances $\mathrm{E}_{\mathrm{fit}}(b)$ for all scan points $p_1, \ldots, p_n$ is defined as:

$$
\begin{aligned}
\mathrm{E}_{\mathrm{fit}}(b) &= \sum_{i=1}^{n} err_i = \sum_{i=1}^{n} \left((x_{i1}, x_{i2}, x_{i3})\, b - y_i \right)^2 \\
&= (Xb - Y)^\intercal (Xb - Y)
\end{aligned} \tag{4.6}
$$

with:

$$
X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & x_{n3} \end{pmatrix} \qquad Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}
$$

To get the correction $\hat{b}$ that minimizes $\mathrm{E}_{\mathrm{fit}}(b)$, one has to solve:

$$
\frac{\partial \mathrm{E}_{\mathrm{fit}}(\hat{b})}{\partial b} = 0
$$

with the solution:

$$
\hat{b} = (X^\intercal X)^{-1} X^\intercal Y \tag{4.7}
$$

The new initial pose is than $q_{it} = q_{it-1} + \hat{b}$.

Estimating matcher accuracy (5)

Cox showed that the covariance matrix, as a means of the matcher accuracy, or quality of the pose estimate can be found as:

$$
\Sigma_{match} = \frac{1}{n-4} (Y - X\hat{b})^\intercal (Y - X\hat{b}) (X^\intercal X)^{-1} \tag{4.8}
$$

4.3 Adaptation of the Cox algorithm

As mentioned above, I.J. Cox used a laser range scanner for his work. One of the properties of those laser scanners is an almost linear error with respect to the distance of a measurement. The sensor device used for this thesis, however, is the camera mounted in the head of the bi-ped robot. Detected line points of the field lines of the soccer field, are projected back to the ground plane (see Section 3.3) to get relative coordinates, used as input for the matcher algorithm. The error of those coordinates is of the order $1/\tan$, that is, the error increases exponentially with respect to the distance. This property

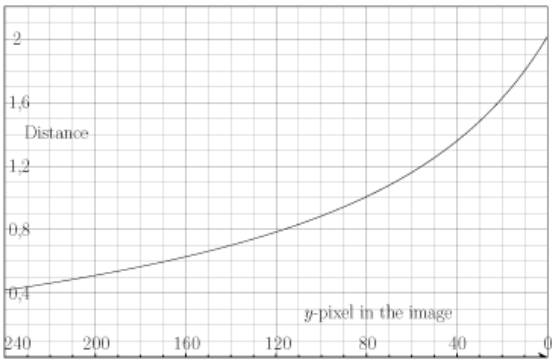


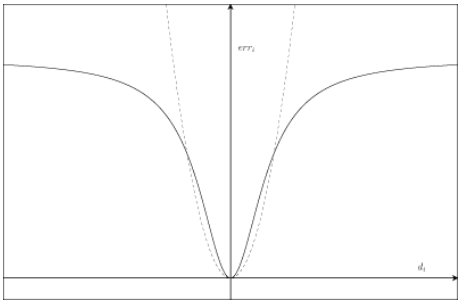
Figure 4.2: Back projection: image coordinate vs. distance ($h_c = 0.45$, $\Theta = 0.52$, $res_v = 240$, $fov_v = 0.6$, $pp_y = res_v/2$)

renders the error function Cox proposed for his matcher algorithm (the squared distance to the closest line) impractical, as distant and therefore uncertain points and especially distant outliers will heavily impact the result, while closer points that are most likely more accurate will be overruled. In addition, I.J. Cox used only straight lines to be able to solve the problem analytically. The center circle or, if available, the corner arcs cannot be modelled, or at best they can only be approximated by line segments.

M. Riedmiller et al. [LLR05] proposed a modified error function and a numerical approach to solve the matching task (3). The proposed error function:

$$err_i = 1 - \frac{c^2}{c^2 + d_i^2} \tag{4.9}$$

maps the distance to the closest line d to a value between 0 and 1 (solid line in Figure 4.3). For values of d between $\pm c$ the shape of the error function is somewhat similar to the squared function (dashed line in Figure 4.3). Outside these boundaries the error function flattens more and more to approximate 1.



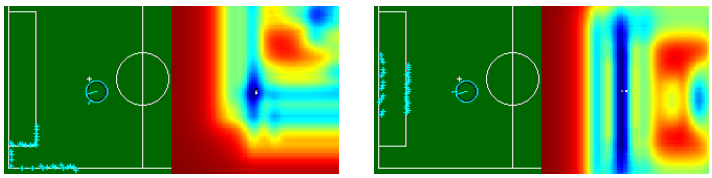
after [LLR05]

Figure 4.3: Traditional squared error function (dashed line) and the more robust M-estimator

The pose correction, that minimizes E_{fit} of step (3), is found numerically, using Resilient Propagation (RPROP) [RB93]. The required gradients, with respect to the translation, have been pre-calculated for a 5×5 cm grid, to minimize the computational load. Hence, only the gradient of the error function with respect to the rotation has to be calculated for each scan point in each iteration, the gradients with respect to the translation can be found in the look-up table.

4.3.1 Aperture problem

The pose estimation cannot always be performed equally sound for all three parameters. Based on amount and structure of the detected line points either parameter of the pose might fail to be estimated. Figure 4.4a depicts a situation where all three parameters



(a) distinct local minima (b) minimum valley

Figure 4.4: Error functions for different line point structures

can reliably estimated as a distinct local minima is to be found. In Figure 4.4b, captured at the very same spot but with a different orientation of the camera, the translation orthogonal to the goal line, and the orientation could be sound estimated, where as it wasn't possible to estimate the translation in parallel to the goal line, in which case the initial pose is the best result that can be provided.

Without proper handling, the parameter that is in parallel to the goal line would skid along the valley with each iteration, as the gradient of the error function is constant, leading to bad results. Hence, the topology of the error function has to be examined for each parameter before the minimization step can be executed. For this examination the second order deviation with respect to each parameter has to be calculated. For results greater than some ϵ_i a local minima can be found and the minimization is performed. The threshold values ϵ_i are found empirically, by logging the second order deviation results for critical situations.

4.3.2 Post processing

The result of the Cox algorithm is a pose estimate and a respective covariance matrix. Even though the covariance matrix is a measure of the quality of the result, it cannot be used to detect misalignments, that is, whether the line points are mapped to the correct field lines.

To lessen the impact of such misalignment and to incorporate the results of the Cox algorithm and the results of the odometry, an extended Kalman filter (see Section 3.4) is used. Starting at a configureable initial pose, the odometry readings are used as prediction step and the pose estimate of the Cox algorithm are used as correction.

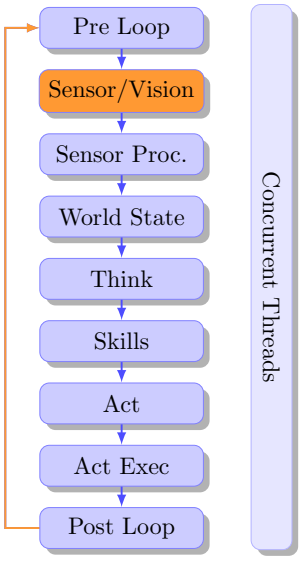
Chapter 5

Implementation

This chapter describes the implementation of the ZaDeAt team software in general and the components required for the localization in detail. The core framework has been developed at the RWTH Aachen as part of the masters thesis [Nie09] by Tim Niemüller. It is now an open source project, to be found at `http://www.fawkesrobotics.org`.

5.1 Fawkes — A robot software framework

Fawkes is a plugin-based, multi-threading, and, per default, single process software framework. It provides basic functionalities, like centralized data storage (called BlackBoard), configuration, logging, timing, and others. The framework functionalities can be accessed via *aspects*. The main application of Fawkes provides the core framework. It initializes and manages the aspects, provides a mechanism to load and unload plugins during runtime, and it establishes the basic network features. Furthermore, it executes a default main loop, where plugin threads can be hooked on to, to enforce synchronized thread execution.



source: [FPS+09]

Figure 5.1: Fawkes synchronization hooks

5.1.1 Plugins

The plugins provide the actual functionality that is required to operate the robot. Each plugin consists of one or more threads. When the plugin gets loaded, all dependent threads are initialized. During this initialization phase, functions that are provided via aspects gets initialized as

well. If, and only if, all threads could be initialized (which implies that all aspects could be provided), and all depending plugins are loaded, the loading of a plugin succeeds. It is guaranteed, by the framework, that all dependencies are fulfilled in order to load a plugin. However, the framework guarantees are only valid during the loading phase of a plugin. In case a resource becomes unavailable, the plugin itself has to take care of a proper handling.

5.1.2 Threads

The threads in Fawkes are based on the POSIX Threads API. Each thread can be executed either in a *continuous* mode, or in a *blocked timing* mode. In the *continuous* mode the thread task gets continuously executed. This is, for example, necessary for high priority tasks, or, on the other end of the scale, long blocking tasks. In the *blocked timing* mode, the thread gets hooked on one of the slots of the main loop (see Figure 5.1). In this case, the thread task gets executed during the respective stage in the main loop, together with all other threads, even of other plugins, that are hooked to the same slot.

5.1.3 Aspects

Framework functionalities, like the BlackBoard, centralized clock, and logging, are provided via *aspects*. An aspect gets initialized during thread creation and it can be guaranteed that either all aspects have been successfully initialized, or the thread creation, and consequently, for example, the loading of a plugin, fails.

5.2 Components

Figure 5.2 gives an overview over the components that are used to operate the Naos of the team ZaDeAt.

5.2.1 naocampos

The exact position of the active camera (relative to the ground plane), is one of the most critical informations when it gets to the calculation of distances, based on camera images. The more accurate the position of the camera is, the more accurate will the resulting distances be. Relevant distances to be measured are the distance to the ball, the distance to the goals (see Subsection 5.2.2), and the distances to the visible lines (see Subsection 5.2.3).

The mandatory variables to be calculated are: the height above ground, and the three rotational parameters roll (the angle when turning around the camera view axis), pitch

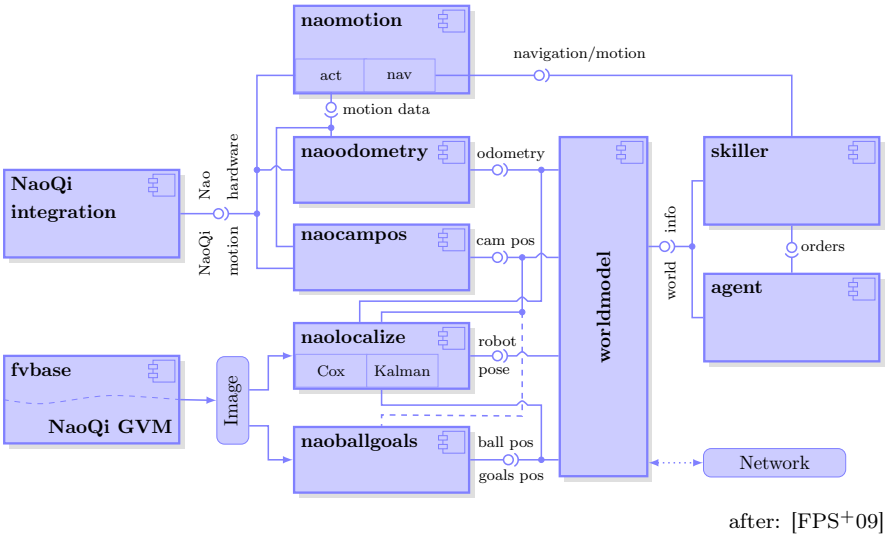


Figure 5.2: Nao software components

(also known as tilt), and yaw (also known as pan). In addition, the translational offset between the reference point for the back projection (see Section 3.3) and the robot's chest is of interest. The reference point for the back projection is the projection of the center of the camera on the ground plane along the fall line.

The joint angles, the currently supporting leg, and the readings from the inertial unit (see Subsection 2.2.4) are the inputs for the camera position calculation. Using the joint angles and applying the Denavid-Hartenberg transformations (see Section 3.2) from the supporting foot, up the supporting leg, through the torso and up the neck to the active camera, would be sufficient in the optimal case. This optimal case would require error-free magnetic rotary encoder (see Subsection 2.2.5) and a hard ground plain. Both requirements are not fulfilled. The magnetic rotary encoders underly the typical variation of technical processes and especially the linearization problem of analog devices. The green carpet of the soccer field is soft enough to impact the rotation of the camera if the center of the weight is not aligned to the center of the supporting foot.

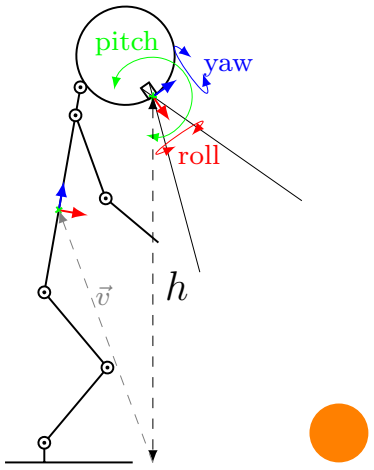


Figure 5.3: Camera location: Hight h above ground, and the orientation in space, roll, pitch, and yaw. The optional vector $\vec{v}$ can be used to improve the results.

As the rotation of the camera is even more critical then the absolute height, especially for distant objects, the readings of the inertial unit can optionally be used to determine the orientation of the torso. However,

the calculation gets even more complex, and as currently only the accelerometer values are used, the result is only valid for the static case. In this case the vector $\vec{v_1}$, from the supporting foot to the center of the torso, and the vector $\vec{v_2}$, from the center of the torso to the principal point of the camera get calculated. As the chains are shorter and the orientation of the torso can be determined with a high accuracy, the cumulative error is less than in the first case.

Data synchronization

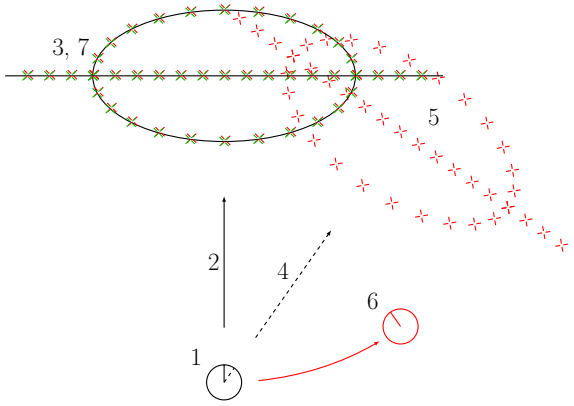


Figure 5.4: Asynchronous camera/hardware data: 1. Actual pose, 2. Camera heading at capture time t_{cap}, 3. Image (scan) at capture time (green), 4. Camera heading at process time $t_{proc} > t_{cap}$, 5. Transformed scan (initial pose at process time), 6. Estimated pose (based on asynchron data), 7. Transformed scan (estimated pose)

A particular important issue is the synchronisation of the camera image and the corresponding camera position. It showed, that ignoring this issue made it impossible to get a stable localization. As the update rate of camera images (10 or 15 Hz) is much lower than the update rate of the joint positions (50 Hz), the position of the joints, as basis of the camera location, could have got updated four to five times. The nominal speed of the head yaw joint, for example, is 7.03° per 20 ms. This leads to a maximum error of 35.15° within five joint position update cycles.

5.2.2 naoballgoals

The object detection is currently color-based. The camera image gets classified using a seeded region-growing algorithm. Due to the limited computational power on the Nao, it is infeasible to classify the whole picture. Consequently, only a subset of the pixels of the camera image can be examined. A grid of so-called scan lines gets applied to the image

and only the pixels along the grid are examined. On the other hand, the examination of the grid pixels only would lead to inaccurate measurements.

The algorithm in use examines the pixels along the scan line grid. If the pixel can be found in a, previously learned, color map, the neighbouring pixels get examined as well and a cluster of matching pixels forms. A drawback of this algorithm is the fact that measures have to be taken to avoid multiple examination of the same pixel as it might be added for evaluation from each direction. Thus, each pixel currently processed has to be tested if it has already been examined. Another drawback is the undefined timing behaviour. The classification time is directly related to the size of classified regions in the image. The classification of close objects takes therefore longer than the classification of remote objects.

The result are bounding boxes around the classified clusters and the exact ratio of matching to non-matching pixels within the bounding boxes.

```
while not end of scanlineGrid do
  if currentPixel already examined then skip it endif
  if currentPixel found in colorMap then
    add currentPixel to examineStack
    while examineStack not empty do
      if firstStackPixel already examined then skip it endif
      if firstStackPixel found in colorMap then
        update boundingBox
        add neighbouring pixels to examineStack
      endif
    done
  endif
done
```

Listing 5.1: Seeded region-growing algorithm

The quality of the object detection relies directly on the quality of the color map. The color map has to be created manually. A training tool allows to select color regions on live images of the robot, and to map the selected colors to an object type, for example, the ball, or the sky-blue goal. Using a Bayesian approach the color map then gets created. Changing lighting conditions, over-trained color maps, and skipped color regions (for example, an unrecognized orange region in the background, while concentrating on the ball) are real hazards when using a color map based approach.

A second tool, developed during the work for this thesis, allows to post-process the learned color map (see Figure 5.5). This allows to close gaps of colors that were not visible during the training, and to extend the learned regions to make the color detection more robust to changing lighting conditions. In the end it showed, that completely hand-crafted color maps worked very well, and that, with a little practice, the time-consuming training step could be eliminated.

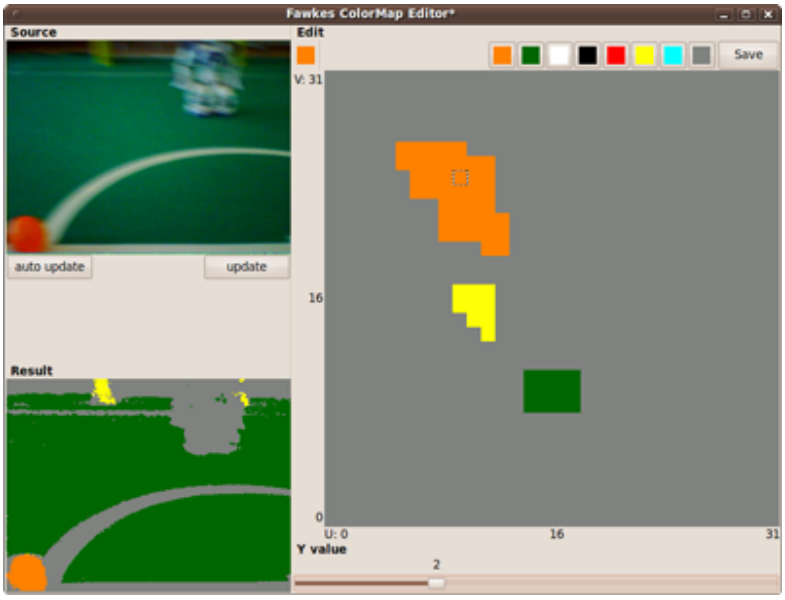


Figure 5.5: Fawkes ColorMap Editor

After the blob detection, which is shared between ball and goal detection, the corresponding bounding boxes (also known as region-of-interest, or ROI) are further processed depending on the object type. This step is necessary to reduce the number of false positives, that is, the number of objects that have been detected as a certain object type, without actually being such (for example, a spectator wearing orange cloths has a good chance to be detected as ball). On the other hand, being too strict during this step may lead to not detecting the objects that are required to play (especially the ball and the goals).

Ball detection – post processing

The post processing for ROIs classified as ball is currently reduced to finding the closed ROI. More sophisticated approaches were tried but could not be evaluated to be stable enough.

One implemented approach used the matching/non-matching pixels ratio within the bounding box of a blob. As the ball is pictured as circle, the ratio between the total area of the bounding box and the area of the blob within it is given as:

$$\frac{A_{circ}}{A_{box}} = \frac{\frac{d^2\,\pi}{4}}{d^2} = \frac{\pi}{4} = 0.785 \tag{5.1}$$

The main problem was the wide field-of-view of the camera (58° diagonal see Subsection 2.2.3). The ball used at the standard platform league has 8 cm in diameter. Given the ball is 3 m away of the robot, the resulting bounding box has an extend of five pixels

in the optimal case (that is, no occlusion, a perfect color map, and good lighting combinations). In this case 20 of the 25 pixels of the blob would have to match to be detected as ball.

Another implemented approach used the ratio of distance to the ball and extend of the bounding box:

$$ext_{blob} = \frac{2\, res_{hor}}{fov_{hor}} \arctan\left(\frac{r_{Ball}}{dist_{Ball}}\right) \tag{5.2}$$

Where ext_{blob} is the expected extend of the blob, for a ball with the radius r_{Ball}, at the given distance $dist_{Ball}$, with the camera properties horizontal resolution res_{hor}, and the horizontal field of view fov_{hor}. Possible occlusion of the ball limits this approach to a "to big to be a ball" criteria. Yet it may help to avoid running of the field in case the ball is not insight at all, as the current solution always takes the closest ROI no matter what. During the RoboCup™ 2009 in Graz the situation occurred a couple of times, that a robot approached a spectator instead of keep looking for the actual ball.

Goal detection – post processing

The goal detection post process aims to extract the two poles of the goal. If both poles are currently visible, the distances to either of them can be used to triangulate, or to be precise trilaterate, the robot's pose (see Section 4.1), and to incorporate these pose estimate to the Kalman filter (see Section 3.4 and Subsection 4.3.2).

The algorithm takes the closest blob found and calculates the distances to all other detected blobs. If the distance between two blobs is approximately 1.4 m, it is assumed that those two blobs are poles of the same goal, and the algorithm terminates. As the poles are connected by a thinner top bar, there is a chance that either a part of the top bar, or even the complete goal is enclosed by a single bounding box. Therefore each ROI gets split at 1/5 from the top and the lower part is classified again to get ROIs that fit the poles as good as possible.

For a correct distance estimation it is required to see the part of an object that touches the ground plane. The back projection will only return correct distances if the bottom of a pole is insight (which depends on the distance, the camera pitch, and possible occlusion). Searching for the green carpet below the pole ROI would be a valid, yet to be included, test.

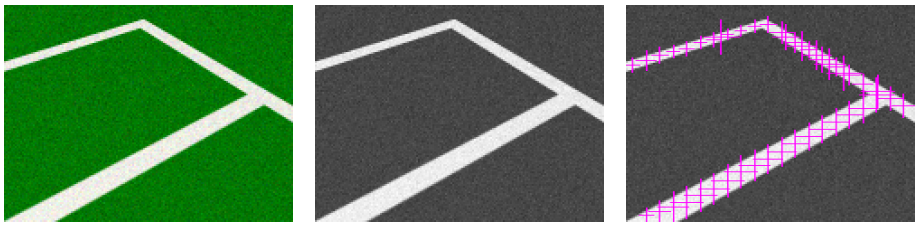
5.2.3 naoloc

The implementation of the naoloc plugin was the main topic of this work. It provides an absolute pose on the field based on odometry readings, feature alignment (detected line points on field lines, see Section 4.2), and goal triangulation. The main parts are the line

detection, the pose estimation, based on the previous pose and the detected line points, and the filtering, where the previous pose, the pose estimation, and the goal triangulation get fused to calculate the optimal new pose estimate.

Line detection

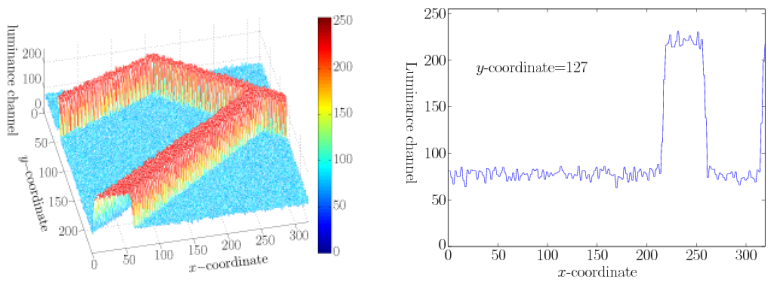
The lines on the soccer field are white lines on a green carpet. Thus, the lines are very ease detectable on the luminance channel, as the contrast between white and the field-green is very good. A line is expected to start with a major rise of the luminance value



(a) Raw image (b) Luminance channel (c) Resulting debug image

Figure 5.6: Line detection in action

and to end with a comparable drop of the luminance value. To detect the rise and drop of the luminance, not the actual value but the gradient over several pixels is examined. By using the gradient, the algorithm gets more robust against changing lighting conditions and pixel noise.



(a) Three-dimensional representation of the luminance channel (b) Single row of the luminance channel

Figure 5.7: Luminance channel analysis

As the robots are white for the biggest part of its surface, they might as well be detected as lines. To avoid this, the maximum line width is defined. If a blob exceeds this width,

it is assumed to be part of a robot and get ignored. In addition, many of the assumed line points, that are actually part of robots, are ignored after back projection. Due to the ground plane assumption, these points would be either to far away, or even above the horizon.

If a line is (almost) horizontal in the image, the line detection fails, as such a line exceeds the maximum line width. To avoid this, the gradient is calculated in x- and y-direction in two consecutive runs. To reduce the computational load a scan line grid is used to limit the number of pixels processed.

Cox algorithm

The Cox algorithm requires an initial pose, a set of line points, represented in world-coordinates relative to the robot, and a set of pre-defined features (the field lines in our case):

```
prevPose:=initialPose
for i:=1 to n do
  transform linePoints to prevPose
  for each linePoint in transformedLinePoints do
    calculate distance to closest fieldLine
    calculate error and gradients
  done
  calculate newPose that minimizes the error
  prevPose:=newPose
donex
calculate covarianceMatrix
return newPose and covarianceMatrix
```

Listing 5.2: Cox algorithm

Details about the Cox algorithm, the error function, or the Resilient Propagation algorithm, used to calculate the new pose can be found in Section 4.2.

Localization post process

In a post process step, the estimated new pose is fed to a Kalman filter [Kal60] (see Section 3.4). Additional inputs for the filter are the odometry readings (see Subsection 5.2.4), and the result from the goal triangulation (see Section 4.1).

In case both poles of either goal are visible, and given, that both poles can be back projected (that is, the ground plane assumption holds), the two poles are used to estimate the current pose. This pose might be less accurate, compared to the result of the Cox algorithm, in case the poles are far away, but it provide a good initial pose in case the pose was lost, and it provide the only true source for a correct orientation as the field, and therefore the pre-defined feature set of the Cox algorithm, is symmetrical.

5.2.4 naoodometry

The naoodometry plugin calculates the relocation of the robot based on the sensor readings of the joint positions. At any given time the transformation of the torso relative to the supporting foot can be calculated by solving the Denavid-Hartenberg transformation chain of the supporting leg (see Section 3.2). By integrating the resulting translation and rotation over time, the locomotion of the robot can be observed.

However, the result of the Denavid-Hartenberg transformation is only accurate if the supporting foot sits flat on the ground, a prerequisite that is not necessarily fulfilled on the soft carpet. Furthermore it is noteworthy that slippage and manual relocation (kidnapped robot) are not detectable. The latter could be solved in the future with the utilization of the inertial unit.

5.2.5 naomotion

This plugin handles the navigation, and the motion execution. The navigator executes `goto(`x, y, θ`)` commands, that, depending on the actual call, are relative either to the current pose of the robot, or to the origin of the global coordinate system, that is, the center of the field. Apparently, a `goto(`x, y, θ`)` call to global coordinates will only succeed if the current pose of the robot is valid. The navigator implements currently only a simple turn-walk-turn path planning algorithm, without any kind of collision avoidance. Future development are proposed to include an incremental Phi* search algorithm [NKL09] for path planning, Bézier curves for the end-point approach, and collision avoidance, at least, by integrating the ultra-sound sensors.

The motion execution handles basic motion calls of the kind `walk(dist)`, `strafe(dist)`, or `turn(angle)`. It currently relies on the motion implementation from Aldebaran, hence the current motion execution is basically a wrapper around Aldebaran's ALMotion calls.

5.2.6 worldmodel

The worldmodel plugin integrates several sources of data (for example, local examinations, team members, or referee decisions) to provide a unified world model for high level components. Integration routines include simple copying of single sources, merging multiple sources based on a set of merging strategies, or filtering the incoming data over time. By integrating other robots beliefs of the world, each robot can refine its own world model [FHL05], thus every robot broadcasts its beliefs with a rate of approximately 10 Hz.

The mapping of data sources to destination, and the integration strategy are defined by configuration values, and as such does not require a modification of the code, which fosters the reusability of the code on different platforms. On the other hand, it is noteworthy that such an generalized approach only allow for very basic integration strategies. Common

tasks like object tracking, that is, updating the pose of objects that are out of sight based on joint, or consequently odometry readings, are currently not implemented.

5.2.7 skiller

The skiller plugin provides an interface to the Lua [IdFF96] scripting language, that is used to implement the basic skills. A skill can be seen as the simplest operation the robot can perform, as seen from a high level abstraction. The list of skills include the basic motion commands (like walk, turn, ...), tasks to kick the ball, to stand up, in case the robot has fallen down, or to search for the ball (or the goals), and more.

The advantages of Lua are the seamless integration into C/C++ frameworks, the small overhead in terms of speed and memory consumption, dynamic structures, no redundancies, and ease of testing and debugging [Ier06]. Lua features a safe execution environment. Errors can be easily detected and will not cause a failure of the whole program. Lua performs automatic memory management. This means that the behavior developer has to worry neither about allocating memory for new objects nor about freeing it when the objects are no longer needed [Web09a].

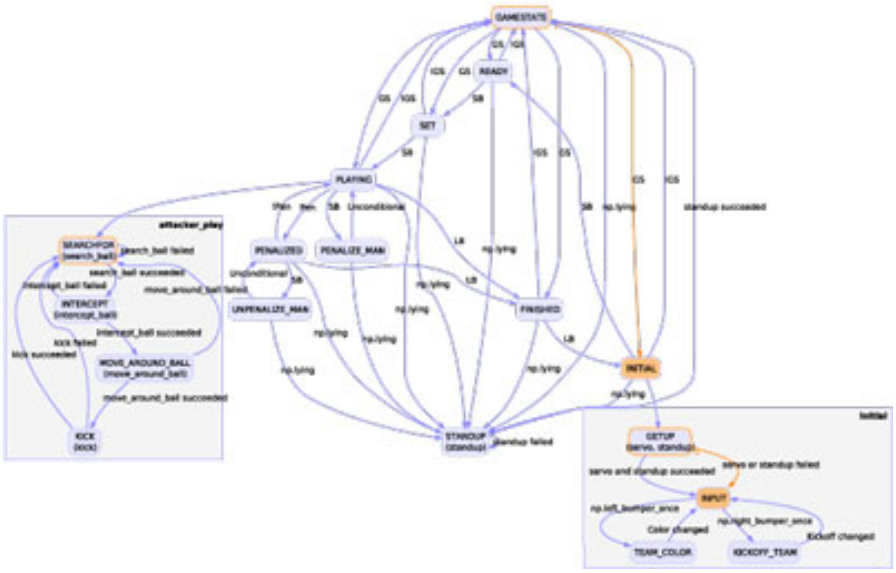
5.2.8 agent

The agent is the top level robot controller. It gets all input data via the world model and is able to execute the skills, that are available to the skiller (see above). The agent can be developed in various ways, either as top level skill executed by the skiller, or as independent component. Currently it is implemented as a FAWKES plugin, similar to the skiller, that executes a Lua defined hierarchical hybrid state machine. The state machine used for the RoboCup™ 2009 competitions is shown in Figure 5.8.

The top level state machine implements the state as required by the SPL rule book [LTC09c] (see Figure 2.9). Each state may execute sub state machines to map the required functionalities.

5.2.9 naoqi_integrator

The naoqi_integrator is developed as a NaoQi module, to integrate the NaoQi middleware into the FAWKES software framework. It provides a data set of the current sensor readings for the BlackBoard, and executes pending commands, sent by various FAWKES plugins.



generated by: Fawkes skillgui

Figure 5.8: ZaDeAt game states for the RoboCup™ 2009 competitions

Chapter 6

Results

6.1 Cox algorithm

The Cox algorithm has been evaluated in simulation and on the real robot. The simulation has become a key factor for debugging and to find improved parameters for the main algorithm and the supporting parts.

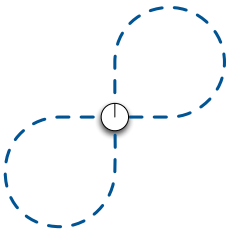
6.1.1 Simulation

Automated test runs provided the biggest part of the evaluation of the Cox algorithm. The bulk execution of test runs with modified parameters led to a final parameter set that could be evaluated on the real Nao in an efficient and non-destructive way. Even during the simulation the robot felt many times, a circumstance that could be easily detected and the simulation restarted. For the real robot this would be an imminent threat. The tests would require the full attention to be able to catch the robot to prevent damage, and thus be very time consuming.

Setup

A test set on the simulator consists of at least 25 runs, each from a random initial pose (translation, and orientation). The self-localization task cannot rely on a particular orientation of the head. The head gets controlled by a high-level agent, which requires to set the orientation of the head based on the current objective (for example search for the ball, or the opponents goal). The self-localization task is limited to examine the camera image that is currently available. To model this behaviour, the head orientation is set independently and randomly during the complete test session. To ensure a broad

field-of-view, the four quadrants, lower-left, upper-left, upper-right, and lower-right, are visited one after the other.



Within each run the robot executes a pre-defined motion pattern. Starting from a random initial pose, the robot walks 0.5 m straight, a clockwise-arc with 0.5 m radius for 270°, 1 m straight, a counterclockwise-arc with 0.5 m radius for 270°, and again, 0.5 m straight. The total distance covered is 6.7 m and it turned for a total of 540°. In the optimal case the robot should have performed a figure-of-eight walk and be at the same spot with the same orientation as it started from. In case the Nao stumbles, the run is cancelled and not counted as one of the 25 runs. However, within the simulation the walk of the Nao is very stable and it only falls down in case it collides with another object.

Results

A single run takes about 3 minutes. The simulator works with a frame rate of 12.5 Hz, this leads to approximately 2 200 frames per run, a test set requires about 90 minutes and produces approximately 55 000 frames. For each frame the actual pose is extracted from the simulator and stored together with the initial and resulting pose of the Cox algorithm.

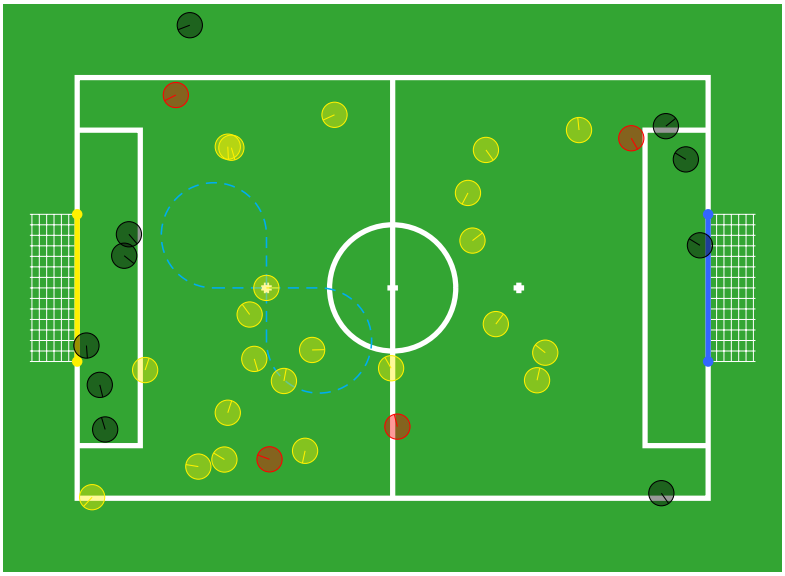


Figure 6.1: Initial poses of the first test set: Black robot icons mark initial poses of test runs where the robot stumbled, red icons mark initial poses where the robot lost its pose, and yellow icons mark test runs where the pose estimation succeeded.

The poses of Figure 6.1, drawn black, mark initial poses of runs where the robot fell down, the red initial poses mark runs where the self-localization failed, and the yellow icons show initial poses where the self-localization succeeded. The pose is considered lost, if either the euclidian distance between ground truth and estimated pose exceeds 0.5 m, or the offset of the orientation between ground truth and estimated pose exceeds 45°. In several cases the pose got lost only for a couple of frames. In case the robot recovered it's pose before finishing the test run, the self-localization task is considered to be successful.

Based on a final parameter set two test series have been performed. The first series was executed on an empty field, that is, the observed robot was the only robot on a, 2009 SPL rules conform, soccer field (see Figure 2.7). The second set took place on the same field but was extended by five static robots. The initial poses of those robots were set according to the rules, that is, one robot of either color was dedicated as goalkeeper and places within the goal area, the remaining two opponent robots were places at the corner points of the penalty area and the remaining robot of the own team was placed within the own half of the field. Sometimes during the run the static robots were pushed around

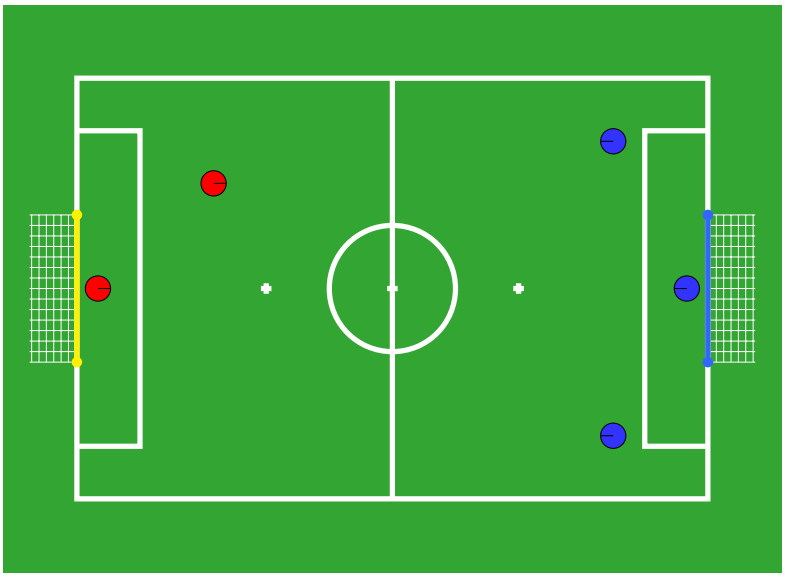


Figure 6.2: Placement of the static robots for the second test set

as the observed robot performed its walk patterns, and, as a result, had a chance to fall down. Thus the poses of the static robots were somewhat random for the single runs of the second test series, but have not been set deliberately.

First test set The first test set required 35 runs to get 25 runs that finished without stumbling. Those 25 runs generated 56 558 frames. For 31 784 frames (56 %), enough line

points could be extracted to perform the Cox algorithm. A total of 52 827 pose estimates were correct, and for 3 731 frames (6 %) the pose was lost.

	Average	Deviation	Median	Maximum
Line points	56.12	25.57	50	173
Distance error* [cm]	15.38	31.26	7.05	322.16
Angular error** [°]	7.13	16.54	2.17	121.49

Table 6.1: Summary of the first test set

	Average	Deviation	Minimum	Median	Maximum
Total frames	2 262.32	n/a	2 261	2 262	2 263
Cox frames***	1 271.36	184.37	864	1 276	1 572
Pose correct	2 113.08	321.99	809	2 247	2 263
Pose lost	149.24	321.67	0	16	1 452

Table 6.2: Overview over the test runs of the first test set

* Eucledian distance between pose estimate and ground truth
** Offset between the orientations of pose estimate and ground truth
*** Number of frames for which enough line points could be detected to perform the Cox algorithm

During the 25 runs the pose got lost 101 times, and was recovered 97 times, that is, four runs ended with a lost pose. In 81 cases the threshold distance of 0.5 m was exceeded, and in 20 cases the pose got lost as the angular offset threshold of 45° has been reached.

Second test set The field for the second test set has been extended by five static robots (see Figure 6.2). Thus, the observed robot had way more encounters with static object, and as a result sumbled more often. It required 50 runs to get 21 runs that have been finished without stumbling of the observed Nao. These 21 runs generated 47 511 frames, for 58 % (27 885 frames) the Cox algorithm could be executed. A total of 33 572 pose estimates were correct, and the poses of 13 939 frames, or 29 %, were lost.

The exceedance of the distance error threshold caused a pose lost 118 times, the exceedance of the angular error threshold for another 55 times. The pose could be recovered 166 times, leaving seven runs finished with a lost pose. Note that the pose lost due to the angular error is only counted in case the distance error was below the threshold

	Average	Deviation	Median	Maximum
Line points	61.06	29.70	54	201
Distance error* [cm]	43.56	73.56	12.93	464.7
Angular error** [°]	24.13	42.43	4.01	180

Table 6.3: Summary of the second test set

	Average	Deviation	Minimum	Median	Maximum
Total frames	2 262.43	3.73	2 255	2 262	2 278
Cox frames***	1 327.86	281.58	799	1 397	1 772
Pose correct	1 598.67	617.76	417	1 765	2 262
Pose lost	663.76	617.72	0	495	1 845

Table 6.4: Overview over the test runs of the second test set

value, that is, if both threshold values were exceeded, only the distance error counter was increased.

The higher average of detected line points of the second run can be explained with the fact that the white body parts of the static robots are also detected as lines. Currently there is no detector for other robots on the field, thus the point sets that are related to robots cannot be distinguished from the point sets of the field lines. A line detector algorithm that used the detected line points to extract straight line segments has been tested, but was removed as it was computationally infeasible on the current platform. The erroneous line points that are detected on other robot's body parts are the main source of why the estimation error, and thus the number of frames where the pose is lost, increases for the second test set.

6.1.2 Real-world test

In order to evaluate the self-localization approach on the real Nao a test environment has been set up within a six-dimensional motion tracker facility. The motion tracker provides ground truth data as reference for the estimates of the Cox algorithm. The experiments were conducted as part of the generation of ground truth datasets for the Nao robot platform [FKN+10].

Setup

The motion tracker is part of the CUBE Laboratory of the Institute for Electronic Music Acoustics of the University of Music and Performing Arts in Graz. The high performance

Figure 6.3: Setup for the tracking session. The left image shows the SPL field within the motion tracker system. The right image shows one of the 15 cameras of the body motion tracking system. The LEDs at the front of the camera provide pulsed infra-red light allowing capturing under day light conditions.

vision-based motion capturing system is usually used for research on innovative forms of arts and music [EPS09]. The tracker system V624, by Vicon, consists of 15 infra-red cameras, that are mounted around the tracked area, and the supporting hard- and software. It is capable to track the three-dimensional positions of reflective markers at a frame rate of 120 Hz. With proper placement of the markers the system is able to calculate the pose of bounding boxes in six-dimensions, that is, three translational and three rotatory components, with a high spacial accuracy of less than one millimeter.

In order to get the reference pose of the Nao, the robot got four reflective markers attached to its chest. In addition, the head got markers as well and was tracked to get ground truth data of the camera location at the time of frame capturing. The latter has been done to verify the current implementation of the camera location calculation. Even though the markers of the head are mounted actually above it, the tracker system is capable to provide the pose that has its origin aligned with the turning axis of the upper head joint (see Figure 6.4 right).

To reduce the number of error sources and to minimize the time required in the tracker lab the tracking session was used for data recording only. The post-processing of the data, the execution of the Cox algorithm, and the statistical evaluation happened in a separate step off-site. One of the key challenges has been the synchronization of the data sets, because there was no way to synchronize the timestamps of the tracker with those of the Nao hardware clock.

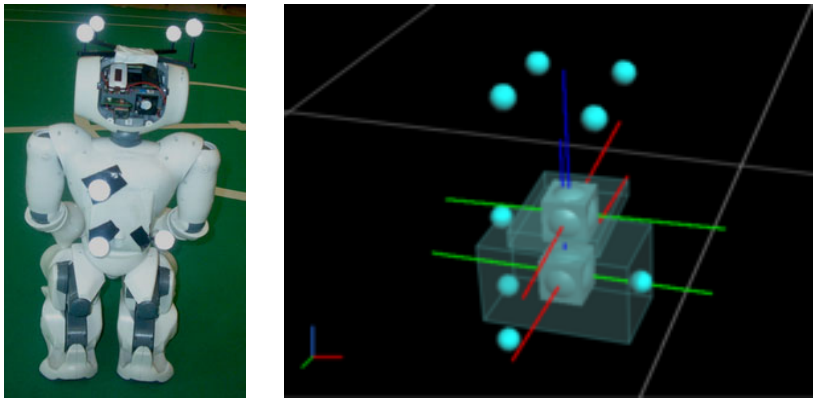


Figure 6.4: Tracking setup of the Nao. The left image shows the Nao with attacked reflective markers. The right image shows the modelled bounding boxes inside the tracking software.

Data synchronization

The default output of the tracker system does not provide timestamps at all.It is a list of coordinates prepended by a frame number. The frame number gets reset with every new run. However, the capturing rate skews between about 118 Hz and 121 Hz, and the timestamp of the start of a run is logged at a accuracy of one second. To get accurate timestamps another feature of the tracker system was exploited. The system sends new measurements as UDP[1] packages over the network. These packages were read by the third-party software *SuperCollider*[2]. SuperCollider is an open-source environment and programming language for real time audio synthesis and algorithmic composition. It has been used to capture the tracker data together with the timestamp of the incoming UDP packages. The result has been one file per run with comma separated values containing the time stamp, and the six-dimensional poses of chest and head. The timestamps of the tracker files, however, are all relative to the program start of the capture software.

As a first post-process step it was necessary to calculate the offset between tracker timestamps and the timestamps captured on the robot. The first and the final run coincidentally began with the same stand up motion. It showed that the z-component of the chest box of the tracker data and the knee pitch of the robot hardware log had a direct relationship during the stand up motion.

In a first attempt the offset o_{first} was calculated for the first run, and for the last run o_{last}. In the optimal case the clocks of the capture system and the robot are equally fast and hence $o_{first} = o_{last}$. In this case the difference between the two offsets was more than four seconds. Consequently the offset between tracker and Nao logs had to

[1] User Datagram Protocol

[2] `http://supercollider.sourceforge.net/`

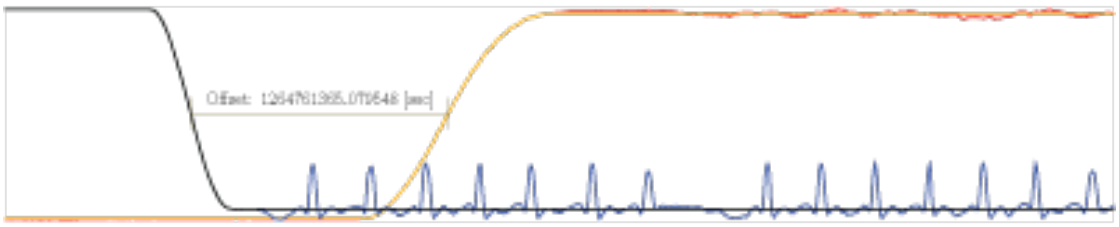


Figure 6.5: Tracker vs. Nao data logs: The red line shows the z-axis of the chest pose of the tracker dataset for the first 1000 frames. The blue line shows the run of the left knee joint. The orange and black lines are the filtered functions used for synchronization.

be calculated separately for each run. To automate this step the approach of the line detection (see Subsection 5.2.3) has been reused to detect the first deliberate movement within the tracker log and the corresponding robot's joint.

Results

The result dataset consists of 19 runs. The runs are compiled from various motion patterns, to cover all available moves of the robot. Details of each run can be found in Table 6.5. The robot covered a distance of 66.7 m and rotated for a total of 6 850°.

A total of 24 762 frames has been captured, of which 12 093 (48 %) provided enough line points to apply the Cox algorithm. The pose estimation succeeded for 20 741 frames, but failed for 3 914 frames (15 %).

The pose was lost for 162 times due to distance error exceedance (of 0.5 m), and 70 time due to angular error exceedance (of 45°). In 230 cases the pose could be recovered, resulting in two runs that finished with a lost pose.

Camera synchronization In order to synchronize a camera frame and the corresponding camera location, the capture timestamp was used to find the dataset within the robot's joints log that had the least offset. This dataset was used to calculate the camera location, as descibed in Subsection 5.2.1. Theoretically, the maximum offset between capture timestamp and corresponding camera location should be less than half the offset between two camera frames. Given the capture rate of 15 Hz, the maximum offset should be 33.3 ms.

The average of all offsets of the tracker dataset frames has been 0.42 ms, with a standard deviation of 55.89 ms. However, the minimum value has been −1 381 ms, that is, the captured frame was almost 1.4 s older than the closest joints dataset, and the maximum

Run	Motion pattern	Ball/Opponent	Head motion
01	walk straight: 1 m	–	–
02	walk straight: 3 m	–	–
03	walk straight: 3 m	✓	–
04	walk straight: 3 m	✓	✓
05	walk straight: 4 m	✓	✓
06	walk arc: 2 m/100°	–	–
07	walk arc: 1 m/180°	–	–
08	walk arc: 1 m/270°	–	–
09	figure-of-eight	✓	✓
10	figure-of-eight	✓	✓
11	turn: +720°, −360°	✓	–
12	turn: +360°, −360°	✓	–
13	turn: +360°, −720°, +360°	✓	✓
14	turn: +360°, −720°, +360°	✓	✓
15	walk sideward: −3 m, +3 m	–	–
16	walk sideward: −4 m, +2 m	✓	–
17	walk sideward: −4 m, +2 m	✓	✓
18	walk sideward: −3 m, +2 m	✓	✓
19	figure-of-eight	–	✓

Table 6.5: Motion patterns of the tracker dataset

		Average	Deviation	Median	Maximum
Line points		35.69	13.56	30	107
Distance error*	[cm]	23.7	18.7	17.99	107.25
Angular error**	[°]	12.8	15.78	7.11	101.1

Table 6.6: Summary of the tracker test set

	Average	Deviation	Minimum	Median	Maximum
Total frames	1 303.26	629.56	392	1 100	2 490
Cox frames***	636.47	394.62	223	465	1 766
Pose correct	1 091.63	448.73	392	947	2 063
Pose lost	206	321.3	0	6	1 071

Table 6.7: Overview over the test runs of the tracker test set

value 1 746 ms. This is explicable for the fact that the flash drive as well as the CPU were at the limit of their capabilities. The fact that several camera frames share the same, outdated, camera location introduces an enormous error. Within 1.7 s the head of the Nao can easily turn from the left to the right limit and back. The impact of this error can be seen in the following section.

Changed parameters for the test run 15 The setup of the fifteenth run proved to be particular challenging with respect to the Cox algorithm and the current state of development. Figure 6.6 shows the result generated with the same parameter set, as used for the other test runs.

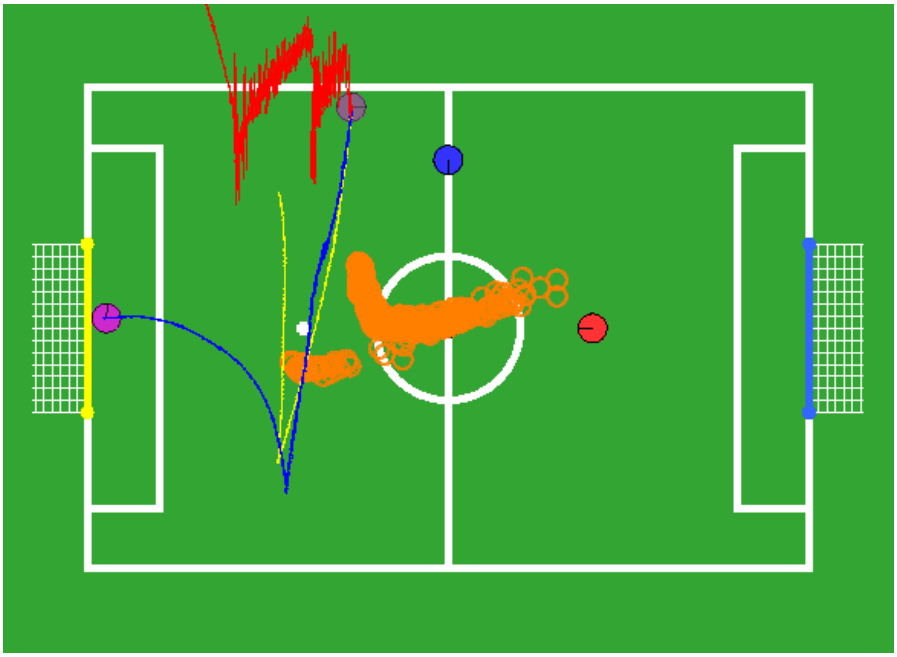


Figure 6.6: Original result of the test run 15, generated with an unmodified parameter set: The magenta colored robot icons shows the initial and final pose of the tested robot, the blue and red icons shows the placement of static robots of the respective color. The blue line depicts the ground truth trajectory, the yellow line the integration of the odometry readings, and the red line the estimated pose. The orange circles illustrates the ball position estimates during the test run. The actual position of the ball ball was at the center of the field.

The robot sticks, more or less, to the side line for the whole time. The trajectory and the placement of the blue robot took account for that behaviour. For the first period, after the start of the run, the Cox algorithm was fed with line points of a T-shape of side line and center line, which is particularly good to minimize the error function with respect to

all three parameters (see Subsection 4.3.1). Thus, the values of the resulting covariance matrix were very small, and consequently the impact on the filter high, compared to the odometry readings. After about 30 cm the blue robot came insight on the right side of the camera image. Without a proper detection algorithm for other robots on the field the white parts of the static robot were also detected as line points resulting, again, in a T-shape geometry that was, again, mapped onto the side line. The same happened with the almost +-shaped line points, resulting from the center line and parts of the center circle. It can be seen that the y-component of the pose was estimated correct while the x-component was tied to the side line. To get the results depicted in Figure A.15 it was, hence, necessary to limit the impact of the Cox algorithm by increasing the values of the covariance matrices of the pose estimates.

6.2 Ball detection

For the test runs containing opponent and ball, the ball detection algorithm (see Subsection 5.2.2) has been applied. The resulting relative position of the ball was transformed, using the ground truth pose provided by the tracker, to get an absolute position on the field. The ball was not moved during a test run, thus, the resulting global position was expected to be constant. Due to the offset between capture timestamp of the camera frame and corresponding camera location and the error introduced by the back projection, the actual resulting position changed quite dramatically. Figures of the tracker test runs, depicting also the ball position estimates, can be found in the Appendix A. For example, Figure A.5 shows the estimated global positions of the ball, as the robot moved along its trajectory of the test run 05. It is to say, that the figure shows only the estimations that were on the field. Several estimates were way out of bound, the estimate that were furthest away had an x-coordinate of over -63 m.

The resulting average estimation of the global position for the test runs, were the ball could be detected, are illustrated in Figure 6.7, together with the corresponding standard deviation ellipses. Note that the extend of the deviation ellipses are scaled down by the factor 10 to be able to draw them within the field boundaries.

6.3 Tournaments

As part of the work for this thesis, it was possible to participate at several tournaments. These were the RoboCup™ 2008 in Suzhou, China, the German Opens 2009 in Hannover, Germany, and the RoboCup™ 2009 in Graz, Austria.

run	Ground truth		Average		Deviation	
	x [m]	y [m]	x [m]	y [m]	x [m]	y [m]
04	0	−0.6	0.157	−0.574	0.615	0.303
05	0	−0.6	−1.440	−1.037	7.915	3.461
09	0	−0.6	0.025	−0.717	0.292	0.621
10	0	−0.6	0.095	−0.700	0.493	1.333
11	0	−0.6	−0.103	−0.301	0.445	0.700
14	0	0	0.125	−0.166	1.153	1.214
16	0	0	−0.156	−0.795	0.153	0.371
17	0	0	−0.271	−0.194	1.736	1.726
18	0	0	−0.378	0.568	3.678	3.064

run	Minimum		Median		Maximum	
04	−1.719	−1.761	0.003	−0.567	1.139	−0.131
05	−63.422	−33.038	0.114	−0.561	1.577	−0.116
09	−1.158	−3.357	0.069	−0.616	0.547	0.341
10	−4.163	−15.215	0.091	−0.533	1.739	0.348
11	−0.490	−3.179	−0.228	−0.053	1.742	0.239
14	−0.728	−5.873	−0.392	0.402	4.788	0.575
16	−0.502	−1.133	−0.136	−0.963	0.125	0.605
17	−16.510	−1.447	−0.060	−0.599	1.568	16.987
18	−2.545	−0.698	−0.693	0.341	43.076	36.917

Table 6.8: Results of the ball detection and global position estimation

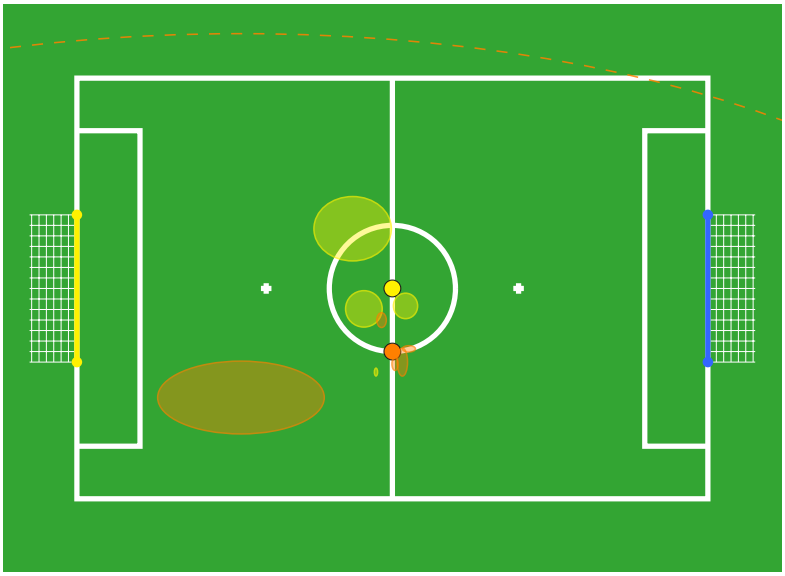


Figure 6.7: Average ball position estimates: The filled circles with the black outline show the actual positions of the ball. The center of the yellow and orange ellipses show the average of all ball estimates for a test run, the extend of the ellipses show the respective standard deviation. The standard deviation has been scaled down by the factor ten to display them within the field boundaries. The dashed line shows the extend of the largest ellipse at true scale.

6.3.1 RoboCup™ 2008, Suzhou, China

The event took place only several weeks after we got our robots for the first time, hence the time in China was used mainly to stabilise the framework on the hardware and to perform experiments required for a stable motion. It showed that the hardware was in a very early state of development and that the reliability of the platform had to be increased tremendously. Many a time the robots required repair after a couple of experiments with modified walking parameters. As the wireless network is always reserved for the tournament games, the robots had to be plugged to the network via cable. The sockets of the network, however were mounted quite fragile and it took only several bends of the cable for the socket to come loose. Also the shell of the robot was not made of some stable plastic, but came from a prototype printer, for which the screws came loose and the parts had a good chance to break in case the robots fell down.

Experiments related to this work showed that the line detection algorithm worked very well, while the calculation of the camera location was still error prone. A part of those errors could be related to hardware malfunctions. In the end it showed that the proposed algorithm, which integrated the accelerometer did not produce results that were reliable and stable enough a the use within competitions. In addition, this approach was also quite costly with respect to the required computational power. As a result the calculation of the camera location is currently only based on the servo readings of the supporting leg, and the head joints. Thus, however, the result is only correct under the assumption that the supporting foot is firmly on the ground and therefore in parallel with the ground plane.

6.3.2 German Opens 2009, Hannover, Germany

The second participation was at the German Opens in 2009. By now the hardware, as well as the middleware provided by Aldebaran, made a big leap towards market ready. The platform became more robust and reliable. Known changes were the replacement of a part of the gears to metal gears, the replacement of the shell, a new head CPU, and a second camera, mounted below the original camera to be able to see the feet of the robot.

At this point the self-localization using the Cox algorithm was implemented but still not competitive. It could be shown that the localization worked in the static case, but failed, more or less instantaneously, as soon as the robot started to move. Later tests revealed that the cause for this behaviour was the offset between a captured camera frame and the related camera location (see Subsection 5.2.1). As the camera location was calculated based on a wrong joint data set, the back projection (see Section 3.3) of the line points fails and consequently also the pose estimation of the Cox algorithm.

The games could be played, to some extend, without the knowledge of the own position. The conducting agent was rather simple: search for the ball, approach the ball, search for

the opponent's goal, turn around the ball until the robot is lined up, and kick. Therefore, color classification and object detection of ball and goals became the main topics during the event. The exhibition hall in Hannover had sky lights and the changing lighting conditions due to scattered clouds was a real problem for the color classification on our robots. The creation of the color tables got changed from a Bayesian learning approach to pure hand-crafted color tables. That way the area of possible colors for a certain object could be made large enough, so that the changing lighting conditions did not impact the result that much anymore.

Improvements could also be seen in other parts of the software. The core framework, which had to deal with teething troubles, like race conditions, became somewhat stable. A fact that finally allowed to pay more and more attention to the actual topics of this thesis.

6.3.3 RoboCup™ 2009, Graz, Austria

For the event in Graz, the object detection was refined. A seeded region growing algorithm (see Subsection 5.2.2) has been implemented to be able to detect the exact boundaries of the detected objects. These information can be used to calculate the plausibility of an object occurrence based on distance and size. It helped furthermore to be able to detect the two goal poles. With the position of the two poles of either goal a rough estimation of the own pose can be calculated based on triangulation (see Section 4.1). The result is quite accurate with respect to the orientation, but may be of least quality with respect to the translation, in case the bottom of the poles are occluded. The estimation can still be useful, either to evaluate the believe of the current pose, or even as a new initial pose for the Cox algorithm. Further work has to be done, as the current system does not support multi-hypotheses tracking [RV09].

It is to say that the RoboCup™ 2009 only got minor attention by the author as his daughter was born during this event.

Chapter 7

Conclusion

Self-localization

During the work for this thesis a self-localization algorithm, based on the work of I.J. Cox [Cox91], has been implemented for the use on the biped robot Nao. This algorithm provides an estimate of the global pose on a soccer field of the RoboCup™ Standard Platform League. The evaluation was done using simulation and a sophisticated six-dimensional motion tracker system, to get ground truth data for the real Nao. A total of 128 831 frames were processed and the pose estimation succeeded for over 83 %, or 107 140 frames. A pose estimation has been defined as successful in case the euclidian distance between estimated position and actual (ground truth) position was less than 0.5 m and the offset between actual and estimated orientation less than 45°.

Frames	Sim 1st	(%)	Sim 2nd	(%)	Tracker	(%)
Total number	56 558	100	47 511	100	24 762	100
Cox processed	31 784	56.2	27 885	58.69	12 093	48.84
Pose correct	52 827	93.4	33 572	70.66	20 741	83.76
Pose lost	3 731	6.6	13 939	29.34	3 914	15.81
Ground truth missing					107	0.43

Table 7.1: Summary of the evaluation of the Cox algorithm

Details about the test sets can be found in Section 6.1. Given the numbers of Table 7.1, it can be said, that the Cox algorithm is suitable for the task of self-localization within the soccer environment of the SPL domain, as long as a stable motion, and thus, reliable odometry readings can be expected. However, the current robots, with the motion engine in use, showed a particularly challenging behaviour. Under odd circumstances, the Nao got caught on the field with the front of its foot, which caused a twist of up to 60° within a single step. These twists were undetected by the odometry, as the odometry calculation

relies only on servo readings. Hence, the expected initial pose was sometimes way off the actual value, and, consequently, the result of the Cox algorithm.

A major drawback of the implemented algorithm is the fact that there are no means to detect that the pose has been lost. The algorithm optimizes the initial pose to find a local minima in the error function, however small this minima might be. To overcome this limitation, the goal detection has been optimized to be able to detect the side poles independently, in order to use the two poles for triangulation. However, the evaluation of the ball detection (see Section 6.2), which showed ball position estimates with an error of more than 60 m, revealed that the absolute numbers of the distance estimation are often of limited use.

The reason why this algorithm still was chosen is the limited computational power required compared to, for example, particle filters. The Cox algorithm calculates the error function n-times (once per iteration) for each pose estimate, with $n = 10$ currently. The particle filter used by other teams in the SPL require 100–150 particles to produce sound results, and the error function has to be evaluated for each particle, for each pose estimate. The vision subsystem of the FAWKES framework has several shortcomings, at least on the Nao platform. Each vision plugin is supposed to perform only a single task, that is, the ball and the goal detection, for example, should be performed in two different plugins. Each plugin, however, gets its own copy of the current camera image. In addition, most of the vision subsystem, depends on one particular colorspace. Therefore a new camera image has to be reordered and copied several times. A task that is very costly with respect to the computation time, as the camera image buffer does not fit in the cache memory, and the connection to the main memory is rather slow on the embedded system.

The evaluation of the time offsets between a camera frame and the corresponding camera location, highlighted that the current framework furthermore needs a mechanism to monitor the current offset in time between image and camera location and to react in case this offset exceeds a given threshold. It should be always better to produce results like there is a ball, I just cannot tell how far away it is, than to approach a ball that is expected to be 60 m away. The same applies for any kind of self-localization approach that uses the camera as sensor. A distance estimation is always only as good as the estimation of the camera location at the capture time, and if this estimation is out-dated, a resulting distance is no good at all.

Other topics

In addition to the self-localization algorithm several loose ends regarding the ball and goal detection have been tied and the overall system stability could be improved by fixing race conditions and eliminating dead locks. A couple of tools were developed to support an efficient preparation for the games during tournaments, and the setup for test sessions. It showed that the on-site preparations often took too long and introduced many errors

that greatly harmed our performance. For example, in the beginning there was no tool support to set the camera parameters, like white balance, contrast, or saturation. The parameters had to be set in a configuration database, the vision subsystem restarted, and the result evaluated using different tools. As this procedure took some time and had to be done several times due to changing lighting conditions, this could be done only for one robot and the resulting parameter set was copied to the other robots. Variances of the different cameras had to be ignored. Several times we forgot to update a robot with the latest settings and ended up with a robot that was not able to detect the ball, only for misconfigured camera parameters.

Appendix A

Results of the tracker test set

The results of the tracker test runs are depicted on the following figures.

Color code: The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. If available, the actual position of the ball is depicted as an orange filled circle with black outline, the estimated ball positions are shown as empty orange circles. The blue and/or red robot icon marks the pose of a static robot of the respective color, if placed on the field for that test run.

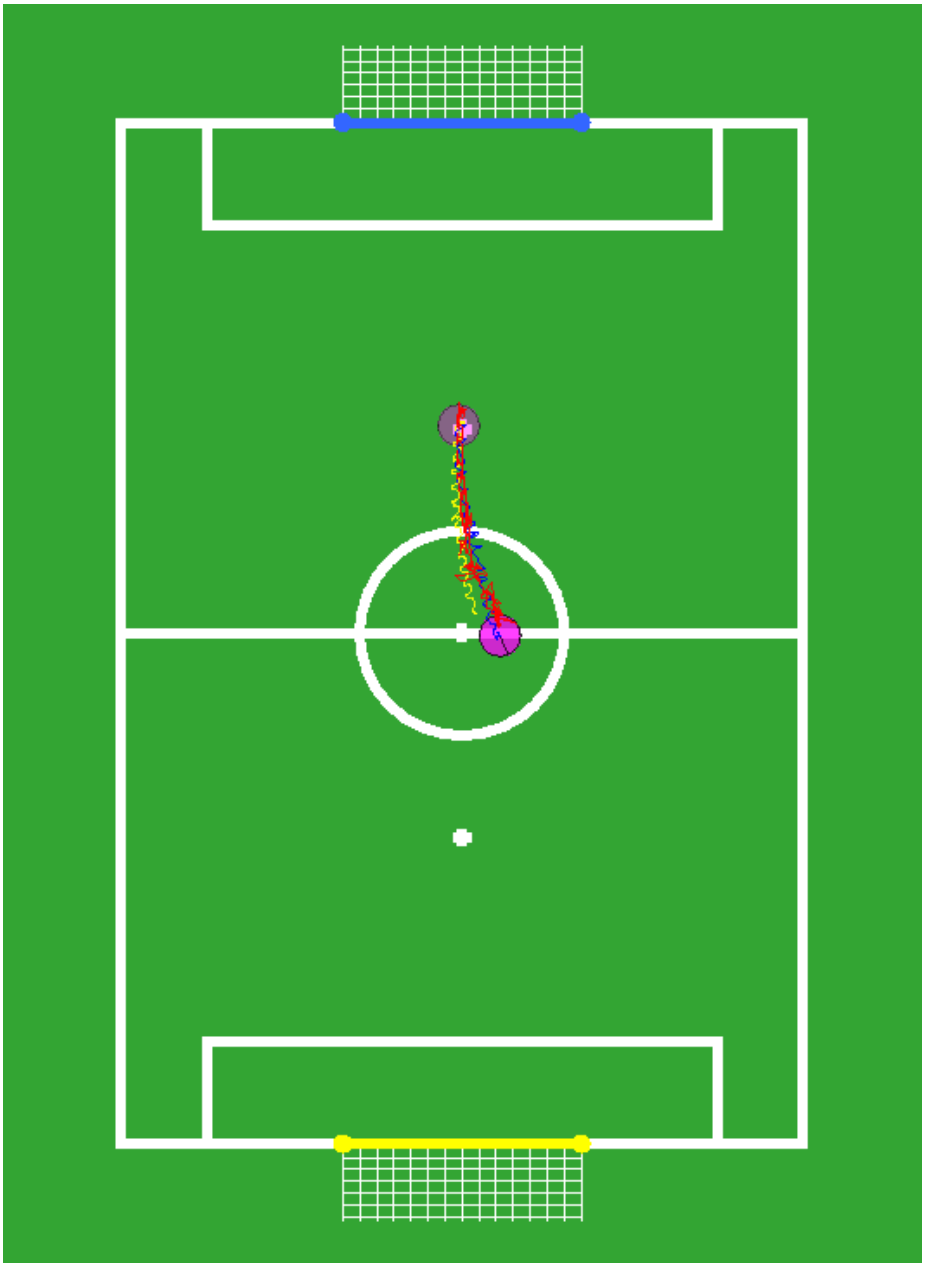


Figure A.1: Run 01: walk straight: 1 m, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter.

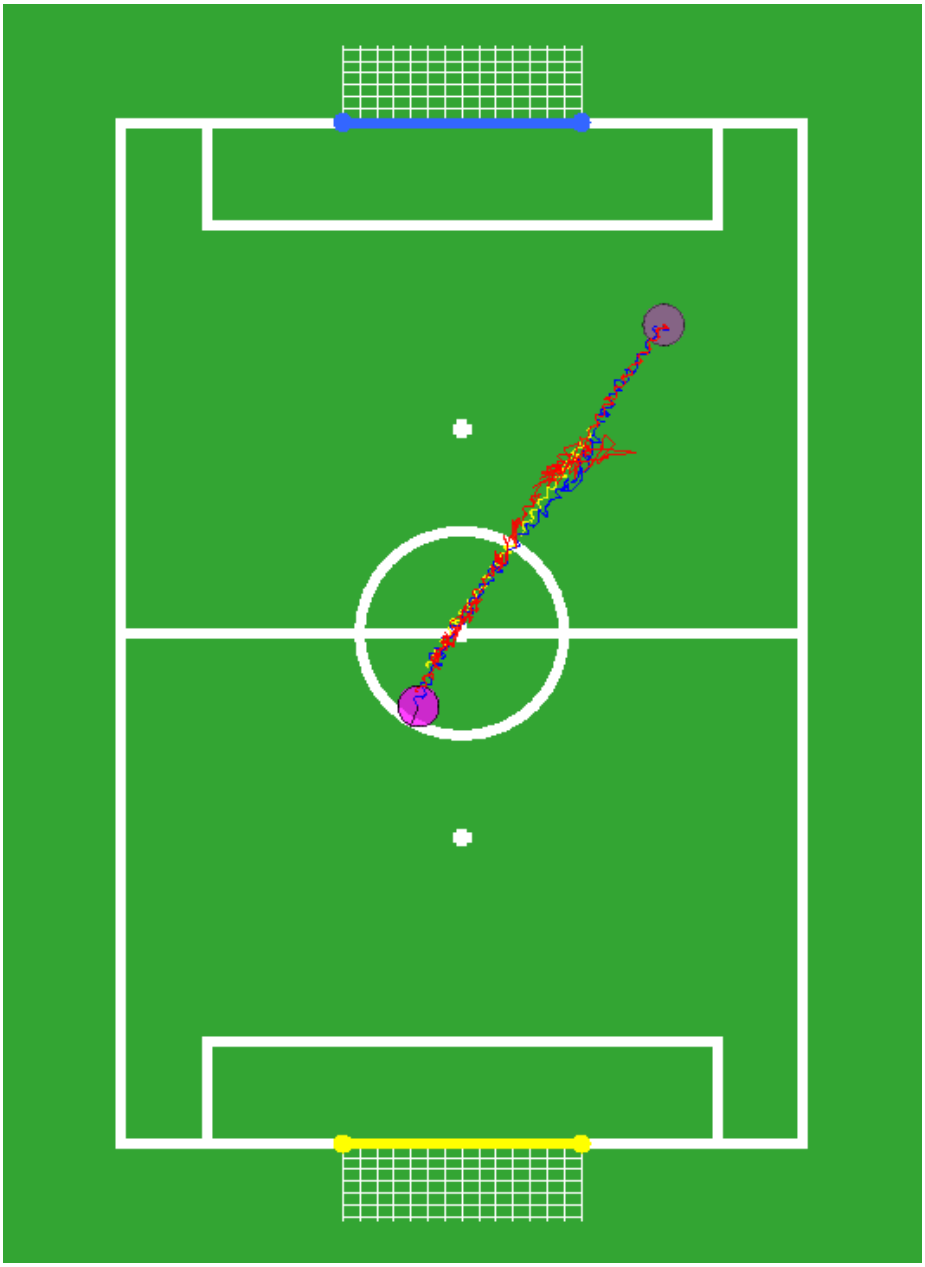


Figure A.2: Run 02: walk straight: 3 m, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter.

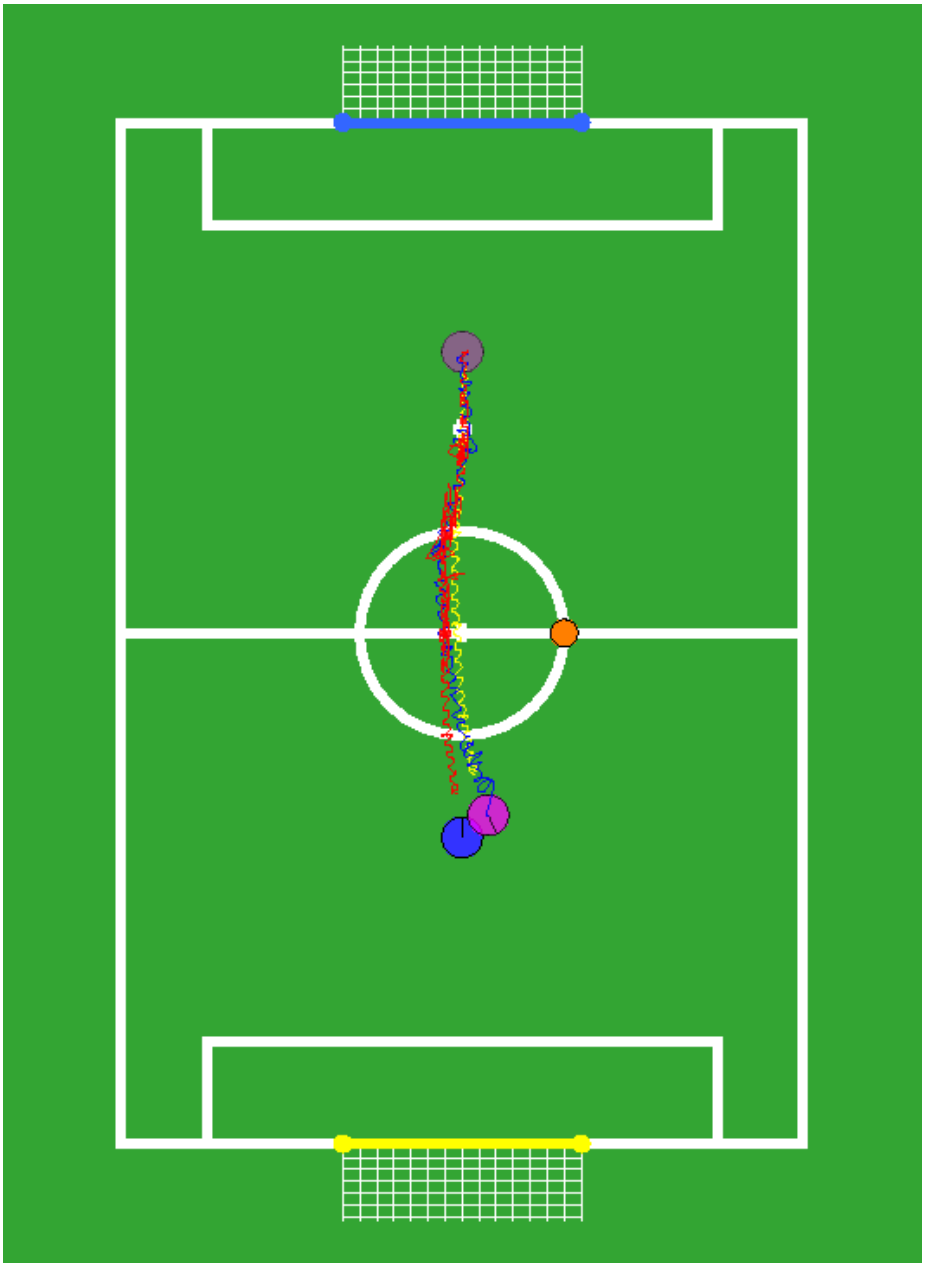


Figure A.3: Run 03: walk straight: 3 m, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The position of the ball is depicted as an orange filled circle with black outline. The blue robot icon marks the pose of a static, blue robot.

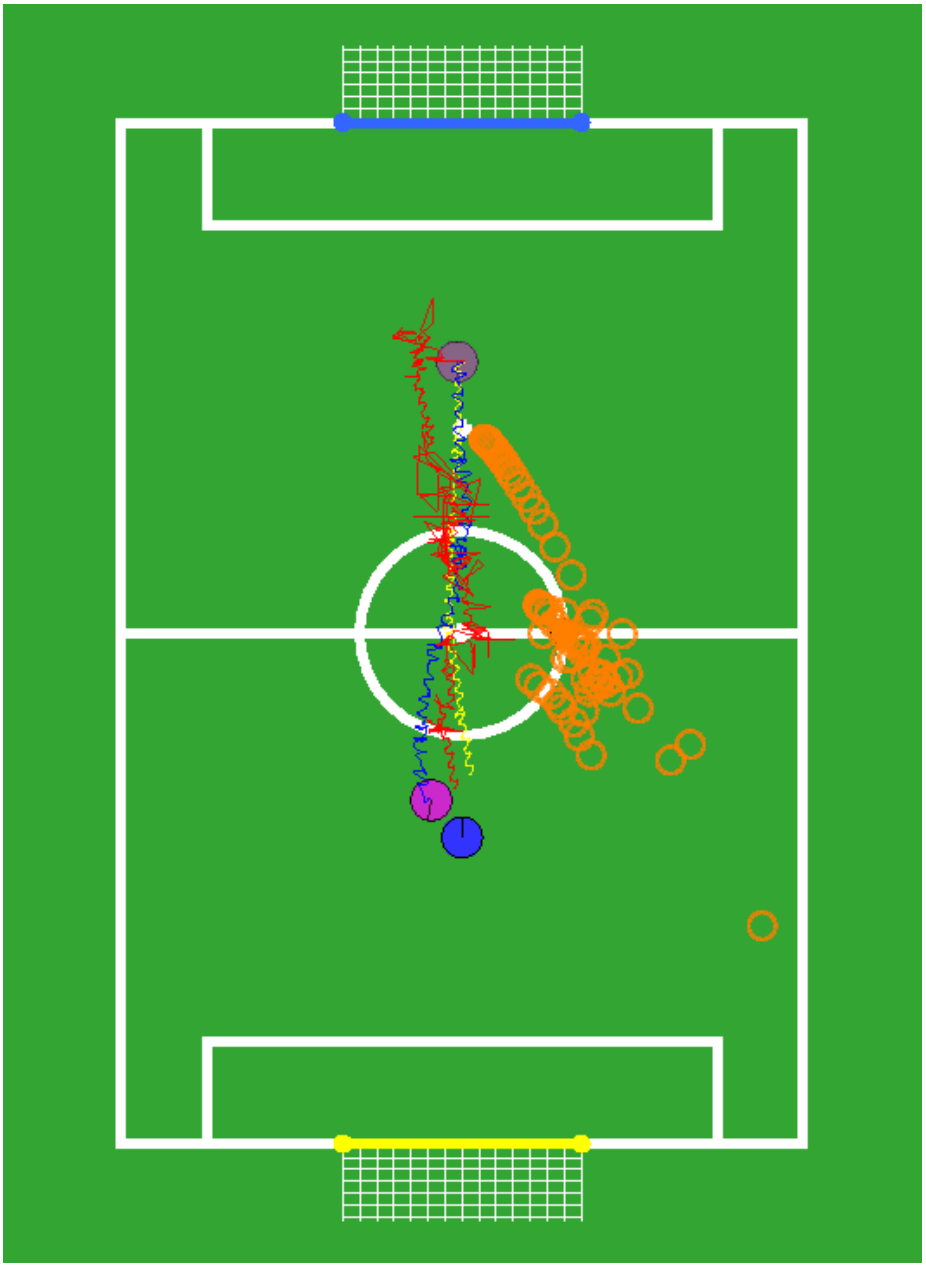


Figure A.4: Run 04: walk straight: 3 m, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The blue robot icon marks the pose of a static, blue robot.

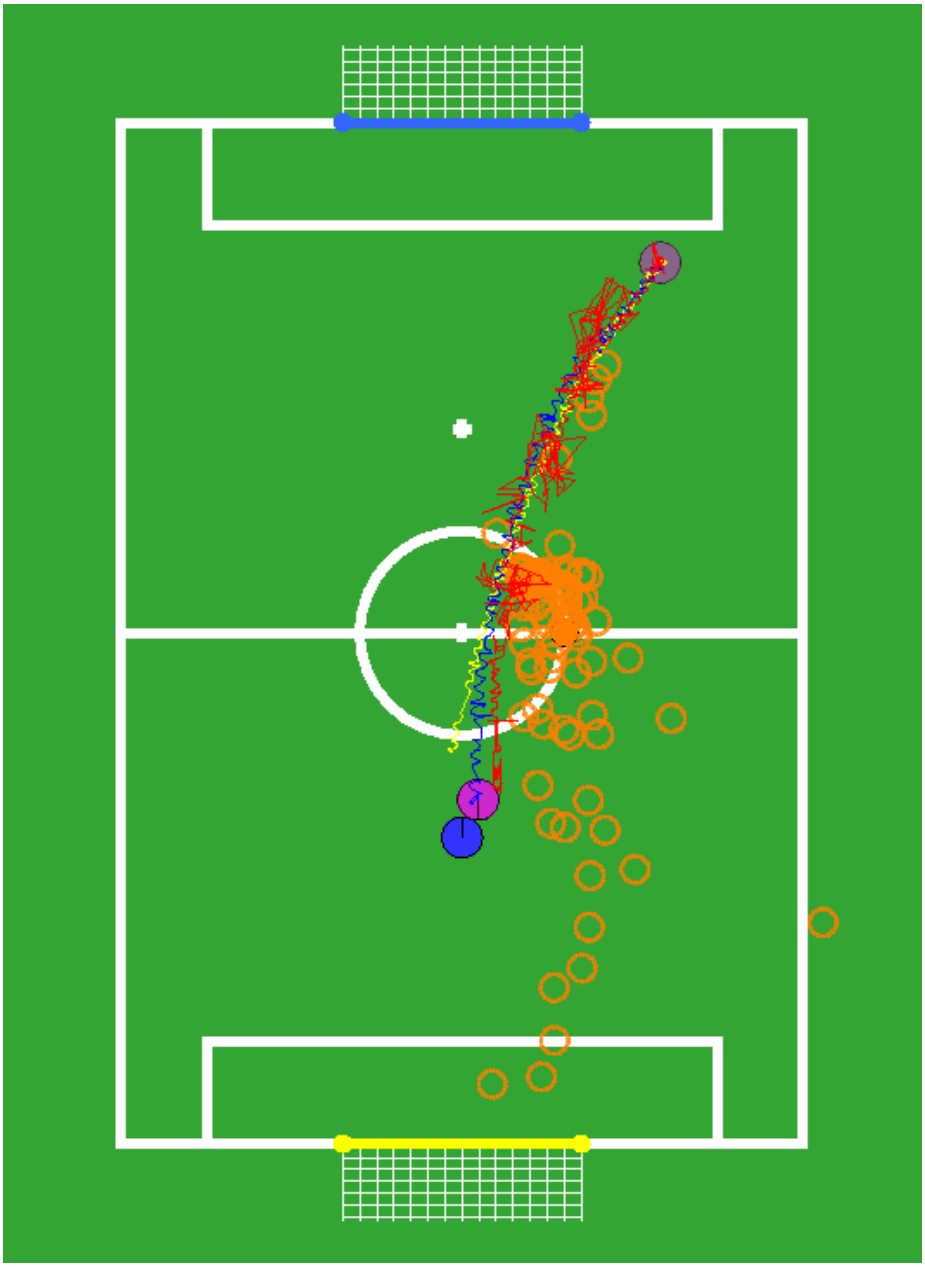


Figure A.5: Run 05: walk straight: 4 m, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The blue robot icon marks the pose of a static, blue robot.

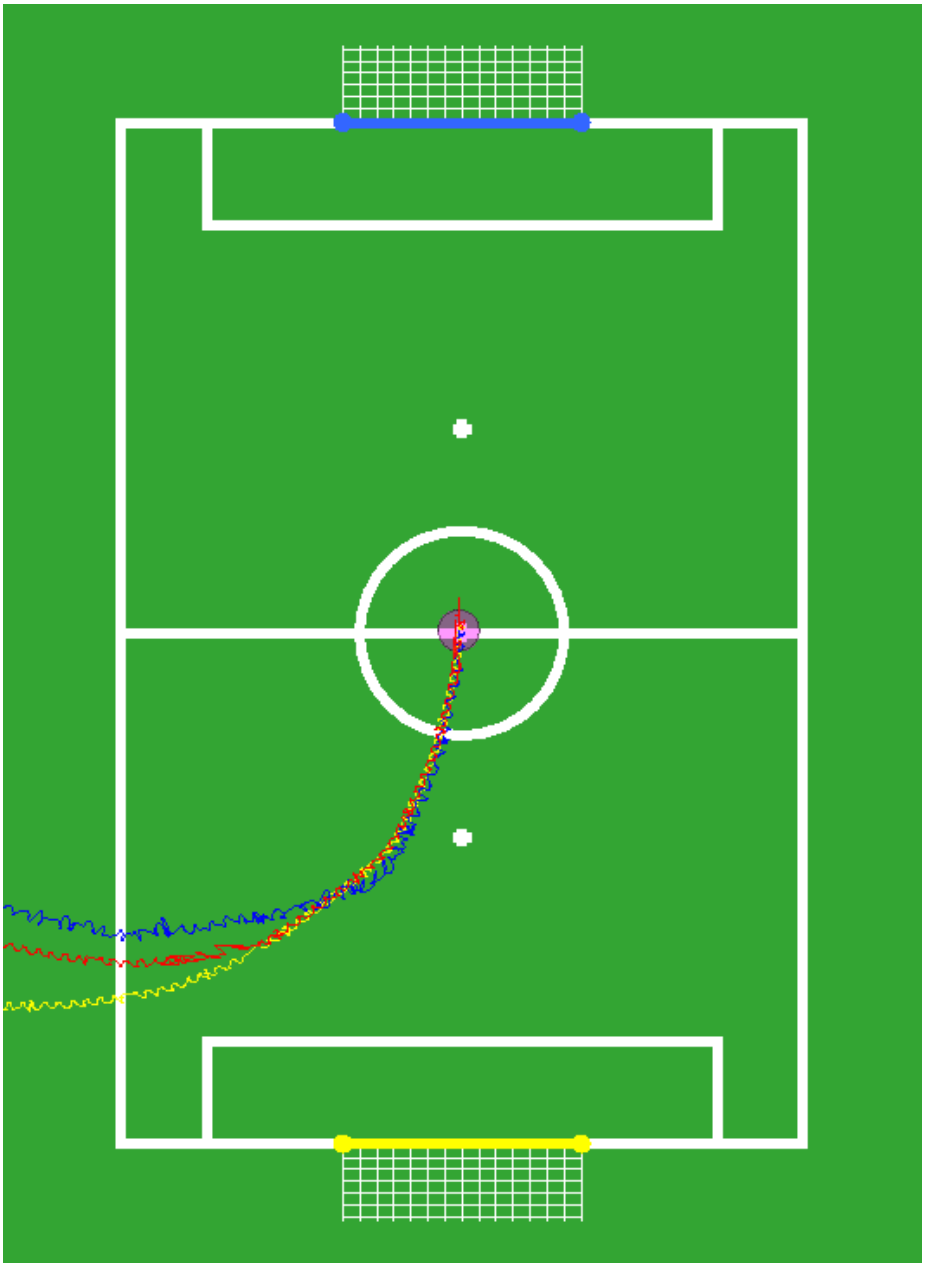


Figure A.6: Run 06: walk arc: 2 m/100°, fixed head. The magenta robot icon shows the initial pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter.

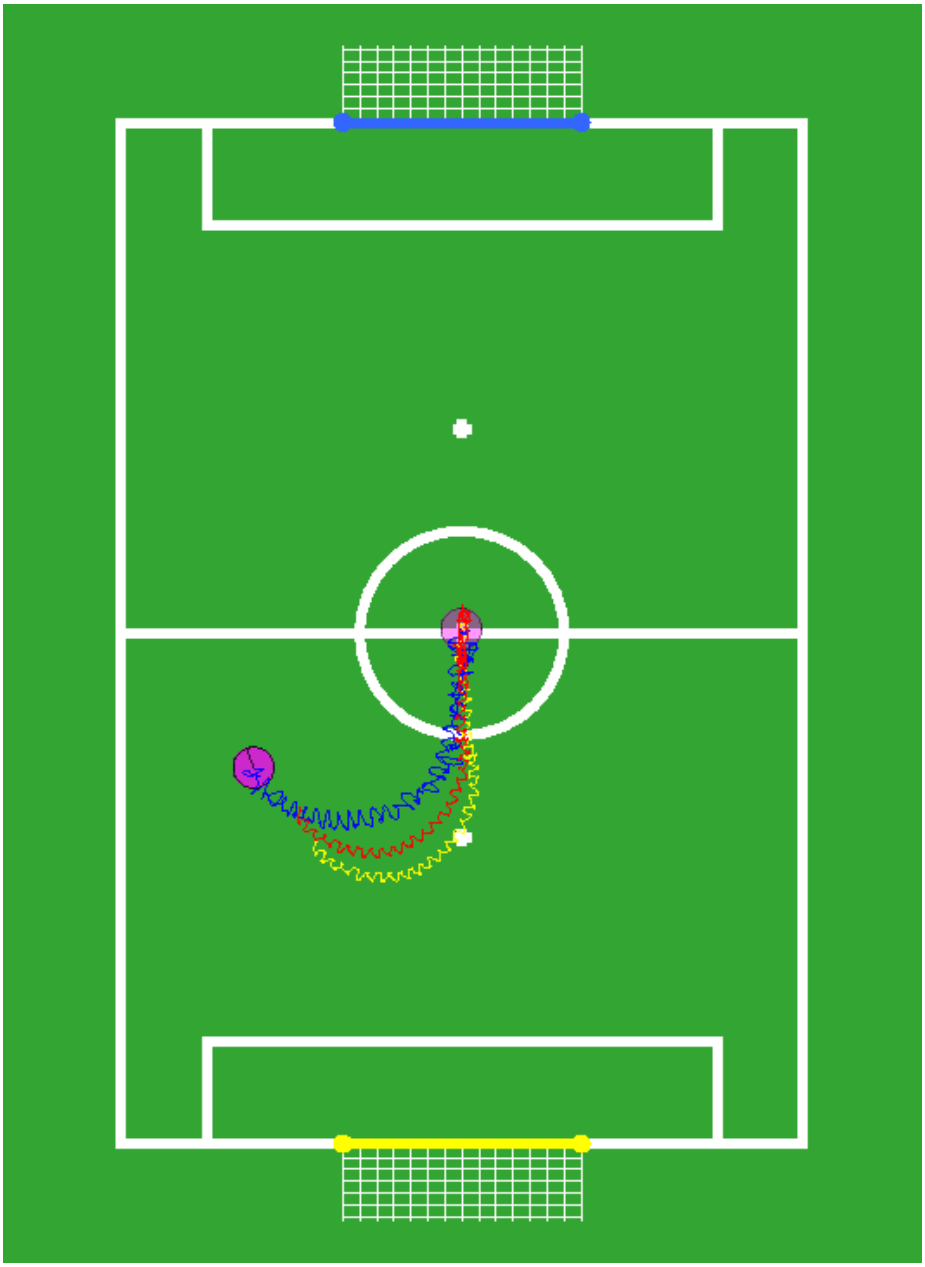


Figure A.7: Run 07: walk arc: 1 m/180°, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter.

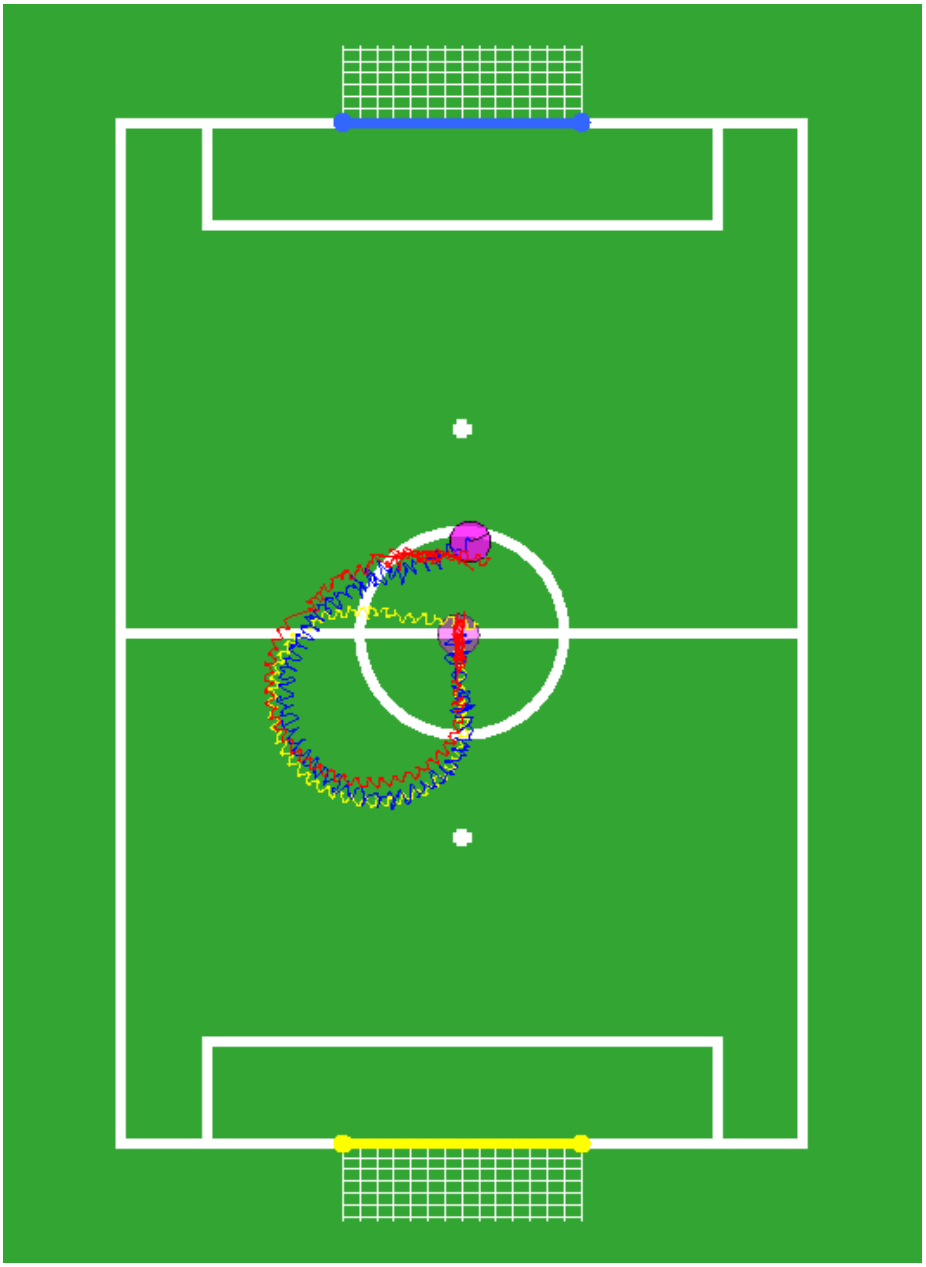


Figure A.8: Run 08: walk arc: 1 m/270°, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter.

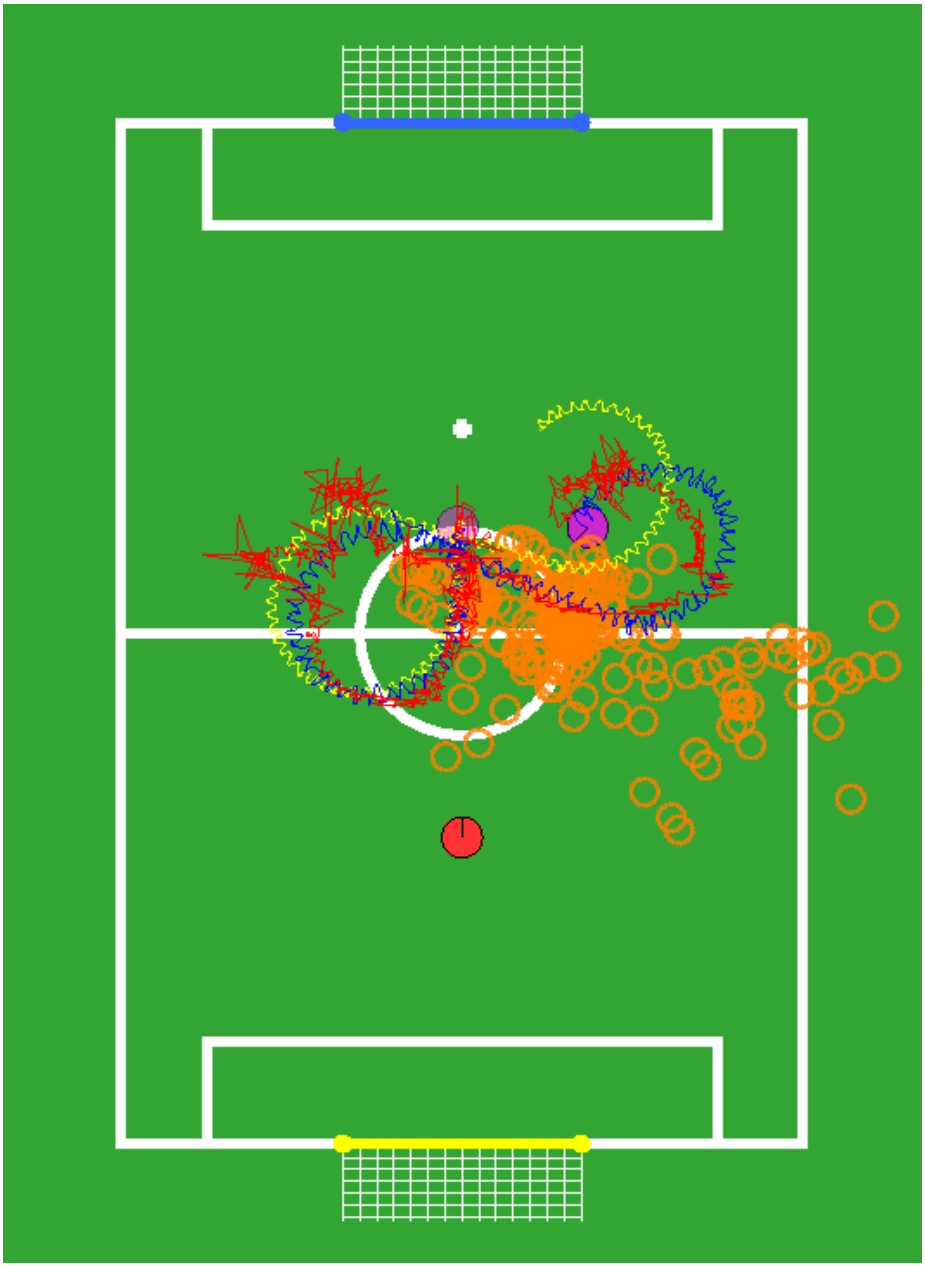


Figure A.9: Run 09: figure-of-eight, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is at the crossing of center line and center circle, the ball position estimates are shown as orange circles. The red robot icon marks the pose of a static, red robot.

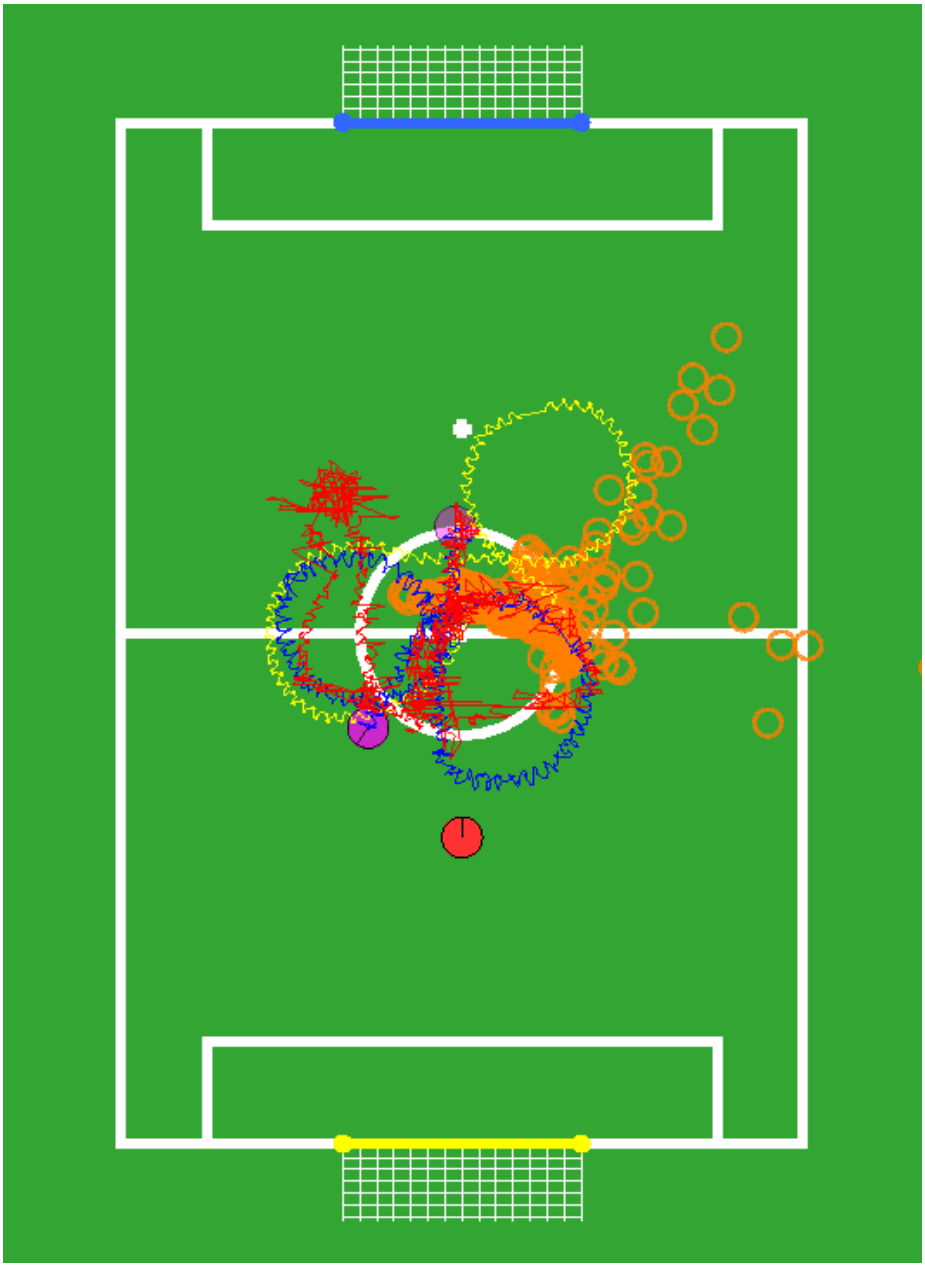


Figure A.10: Run 10: figure-of-eight, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is at the crossing of center line and center circle, the ball position estimates are shown as orange circles. The red robot icon marks the pose of a static, red robot.

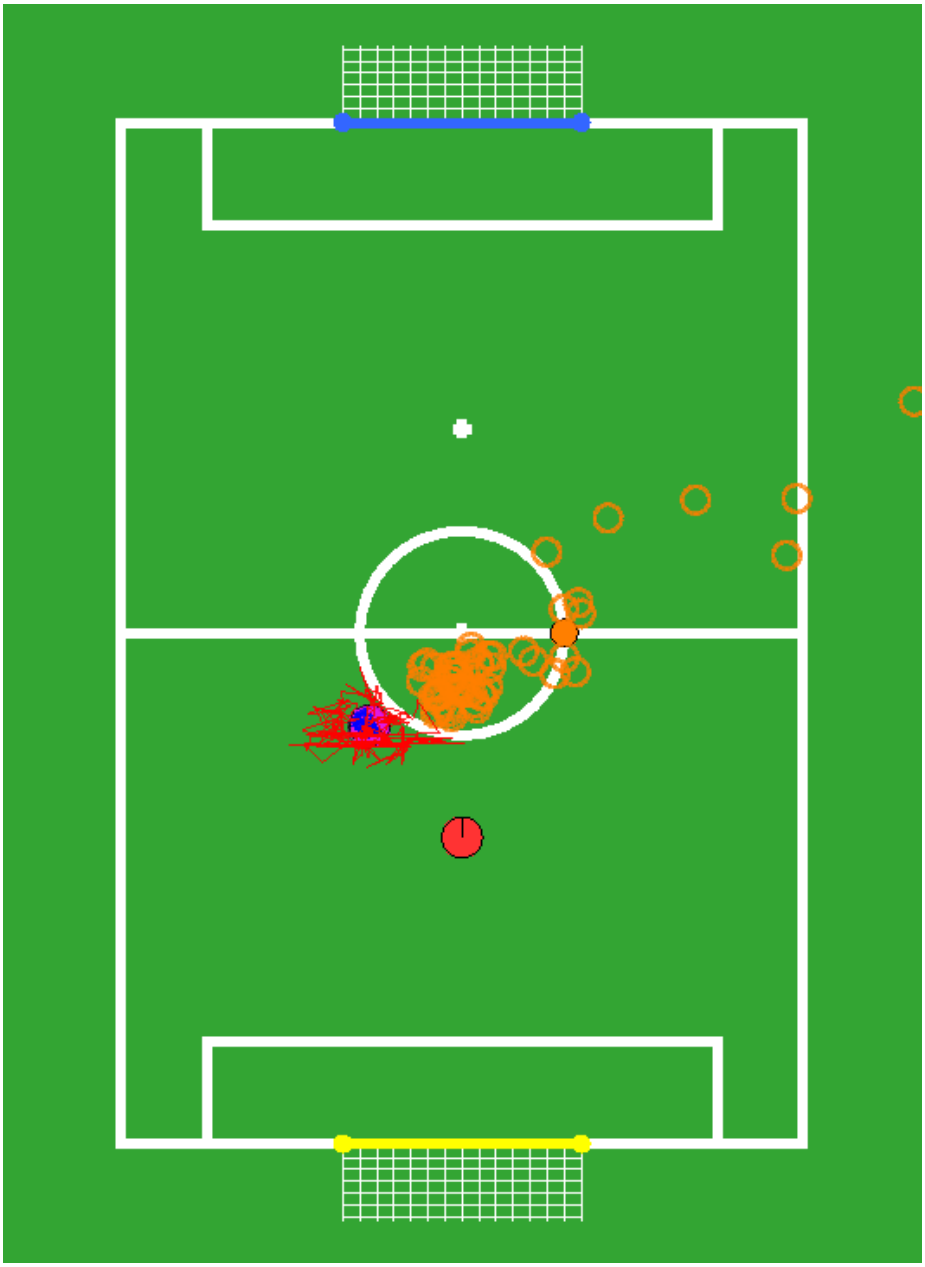


Figure A.11: Run 11: turn: +720°, −360°, fixed head. The magenta robot icon shows the final pose of the tested robot. The blue line, on top of the pose icon depicts the ground truth trajectory. The red line illustrates the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red robot icon marks the pose of a static, red robot.

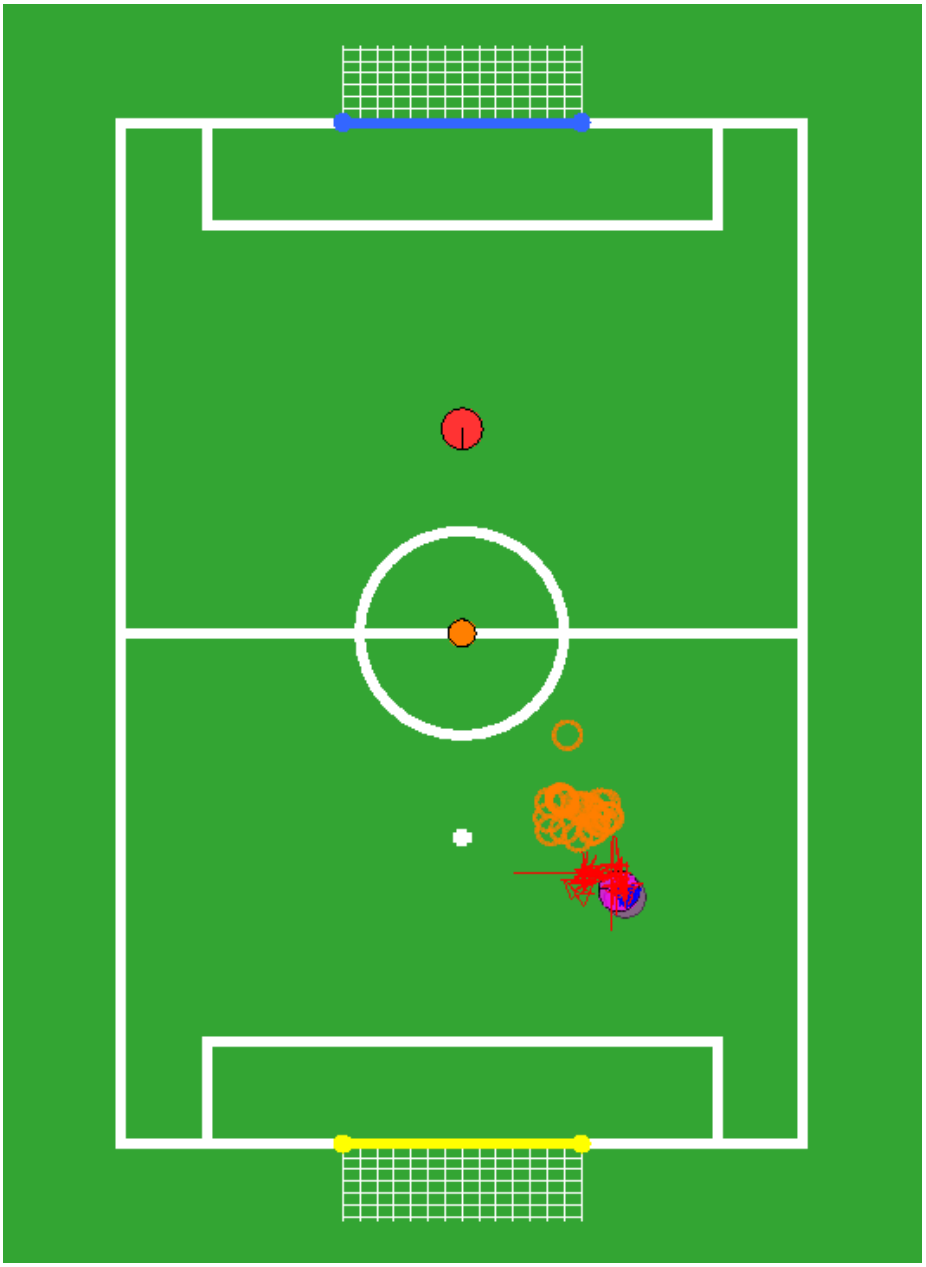


Figure A.12: Run 12: turn: +360°, −360°, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line, on top of the pose icons depicts the ground truth trajectory. The red line illustrates the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red robot icon marks the pose of a static, red robot.

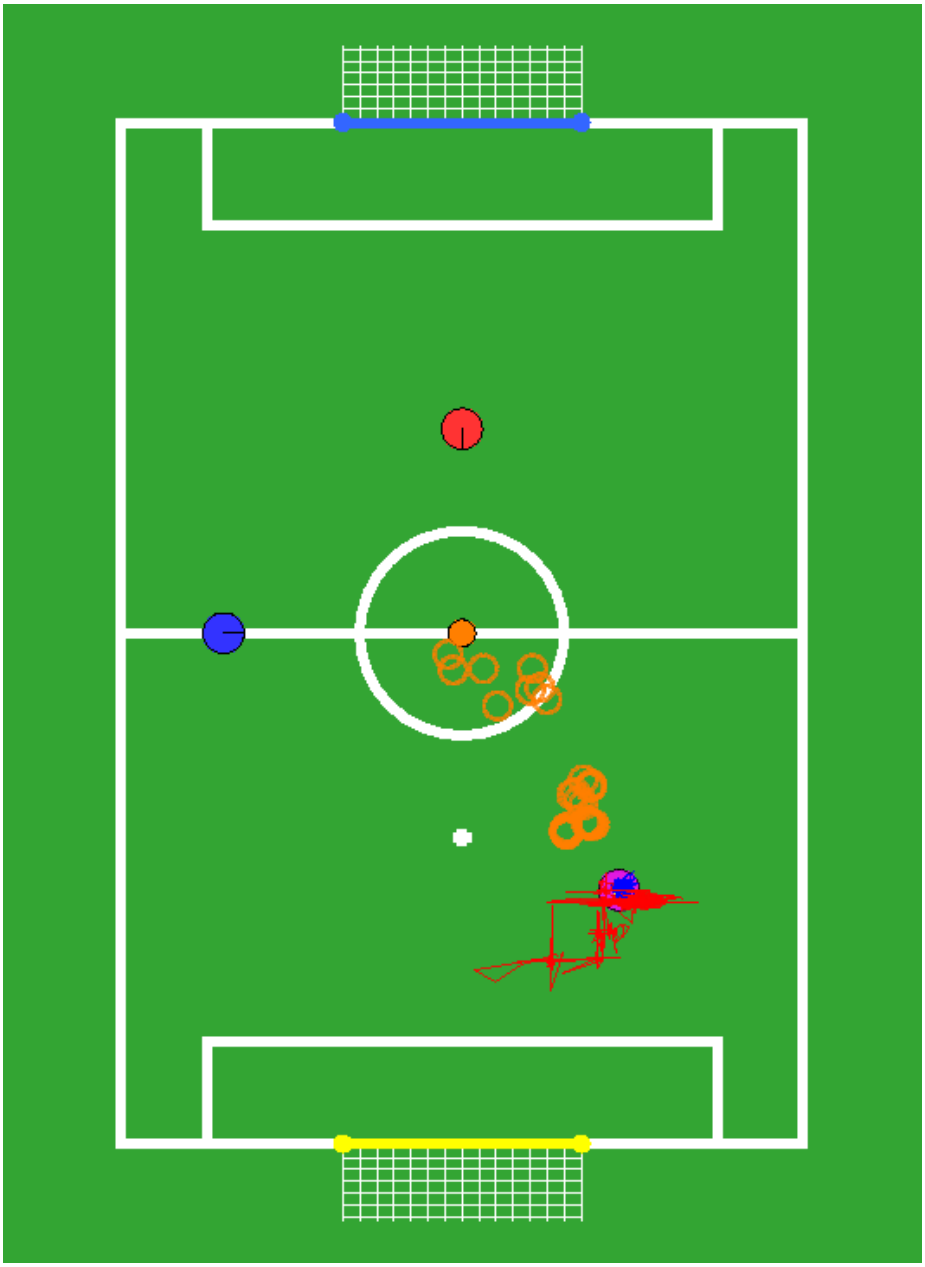


Figure A.13: Run 13: turn: +360°, −720°, +360°, moving head. The magenta robot icon shows the final pose of the tested robot. The blue line, on top of the pose icon depicts the ground truth trajectory. The red line illustrates the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red and blue robot icons mark the poses of static robots of the respective color.

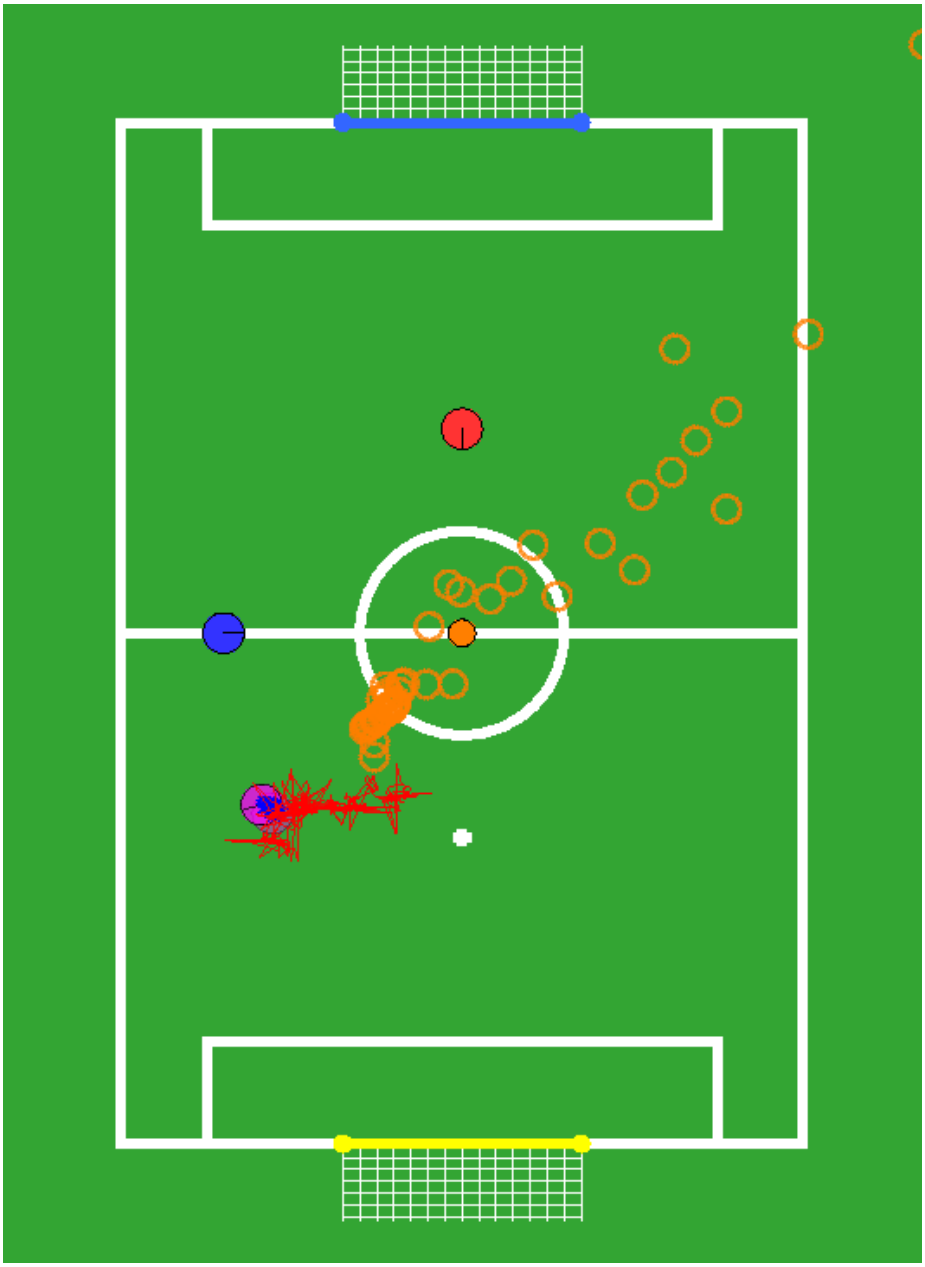


Figure A.14: Run 14: turn: +360°, −720°, +360°, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line, on top of the pose icons depicts the ground truth trajectory. The red line illustrates the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red and blue robot icons mark the poses of static robots of the respective color.

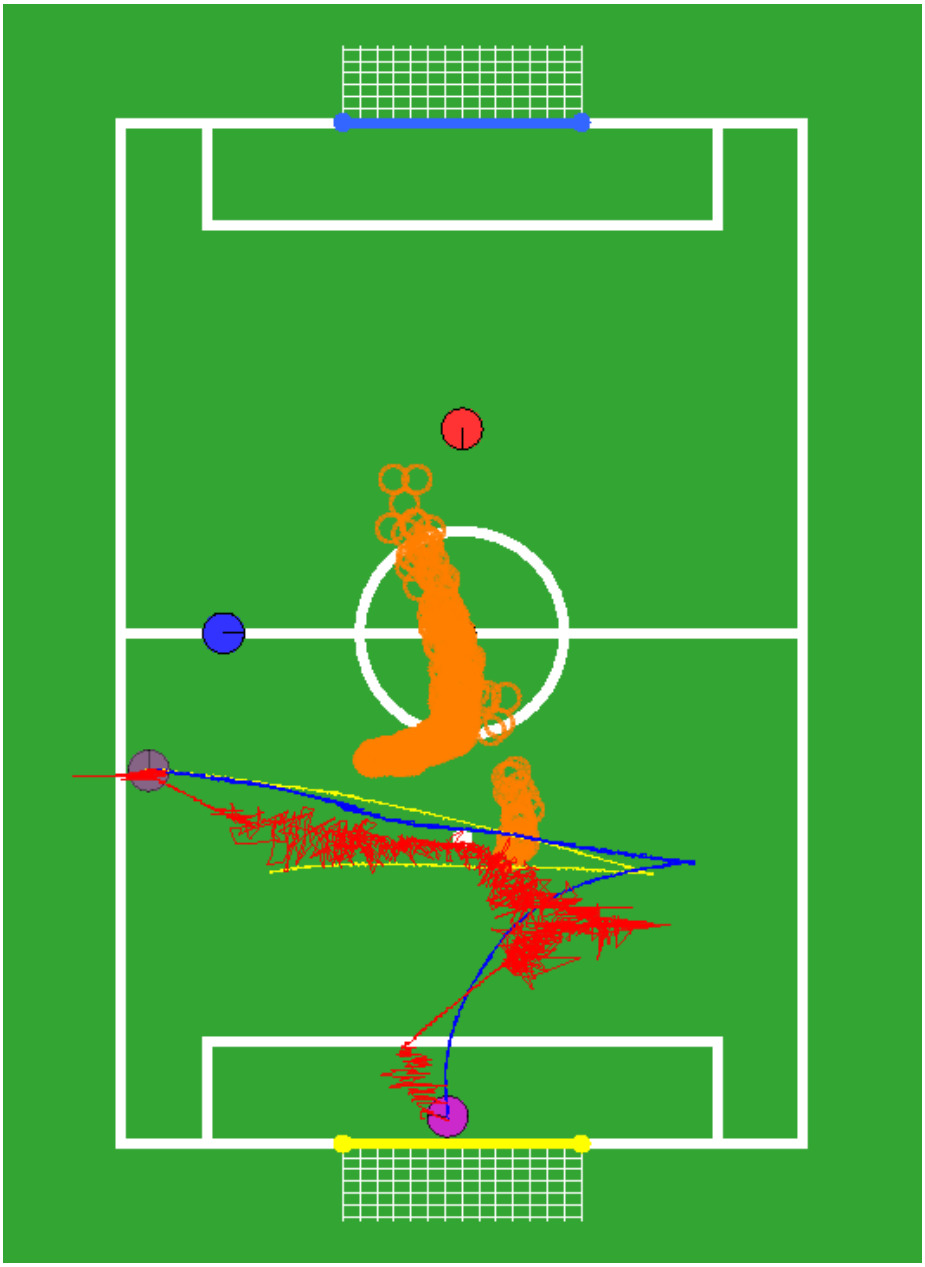


Figure A.15: Run 15: walk sideward: −3 m, +3 m, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, at the center of the field, the ball position estimates are shown as orange circles. The red and blue robot icons mark the poses of static robots of the respective color.

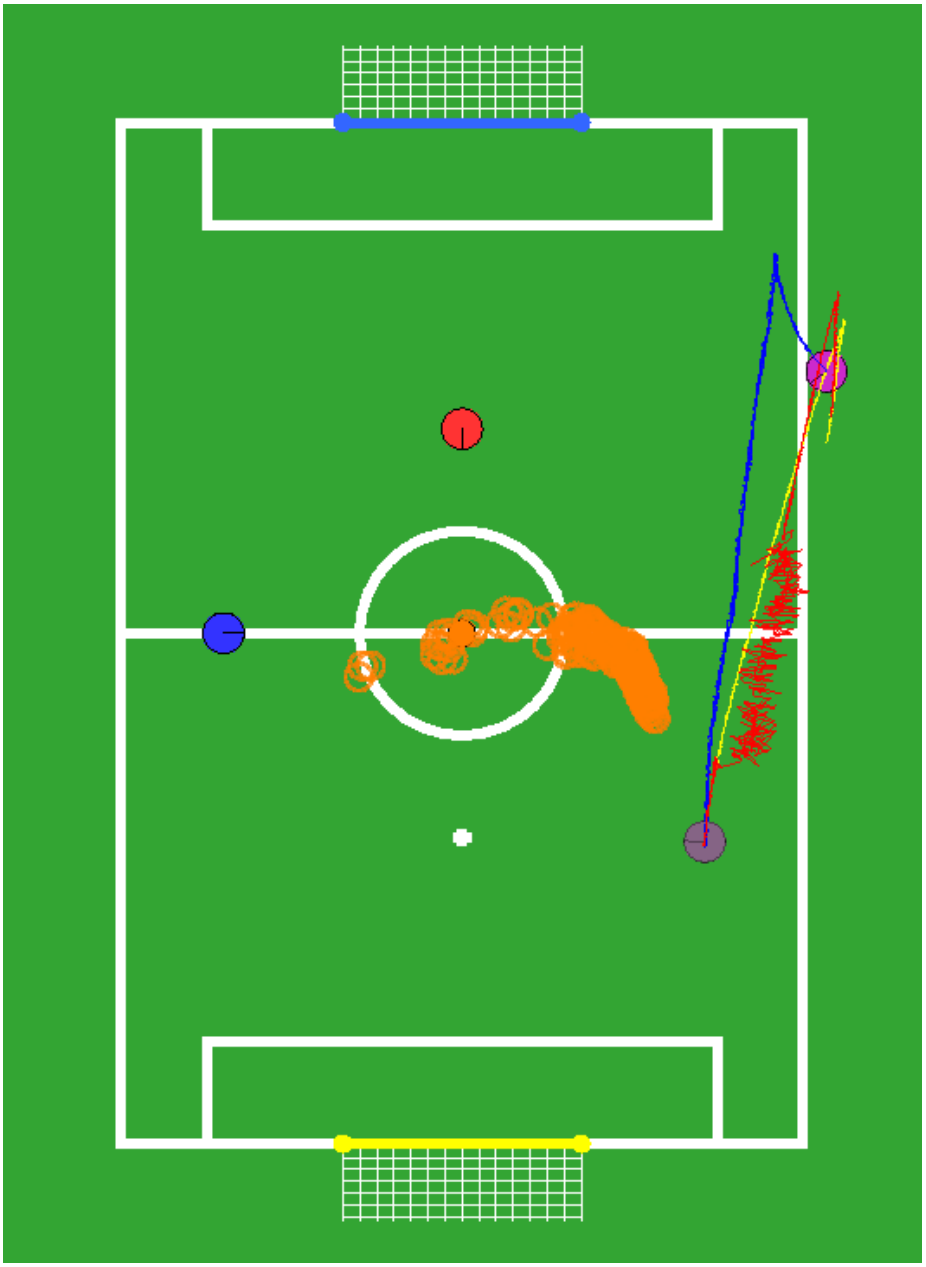


Figure A.16: Run 16: walk sideward: −4 m, +2 m, fixed head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red and blue robot icons mark the poses of static robots of the respective color.

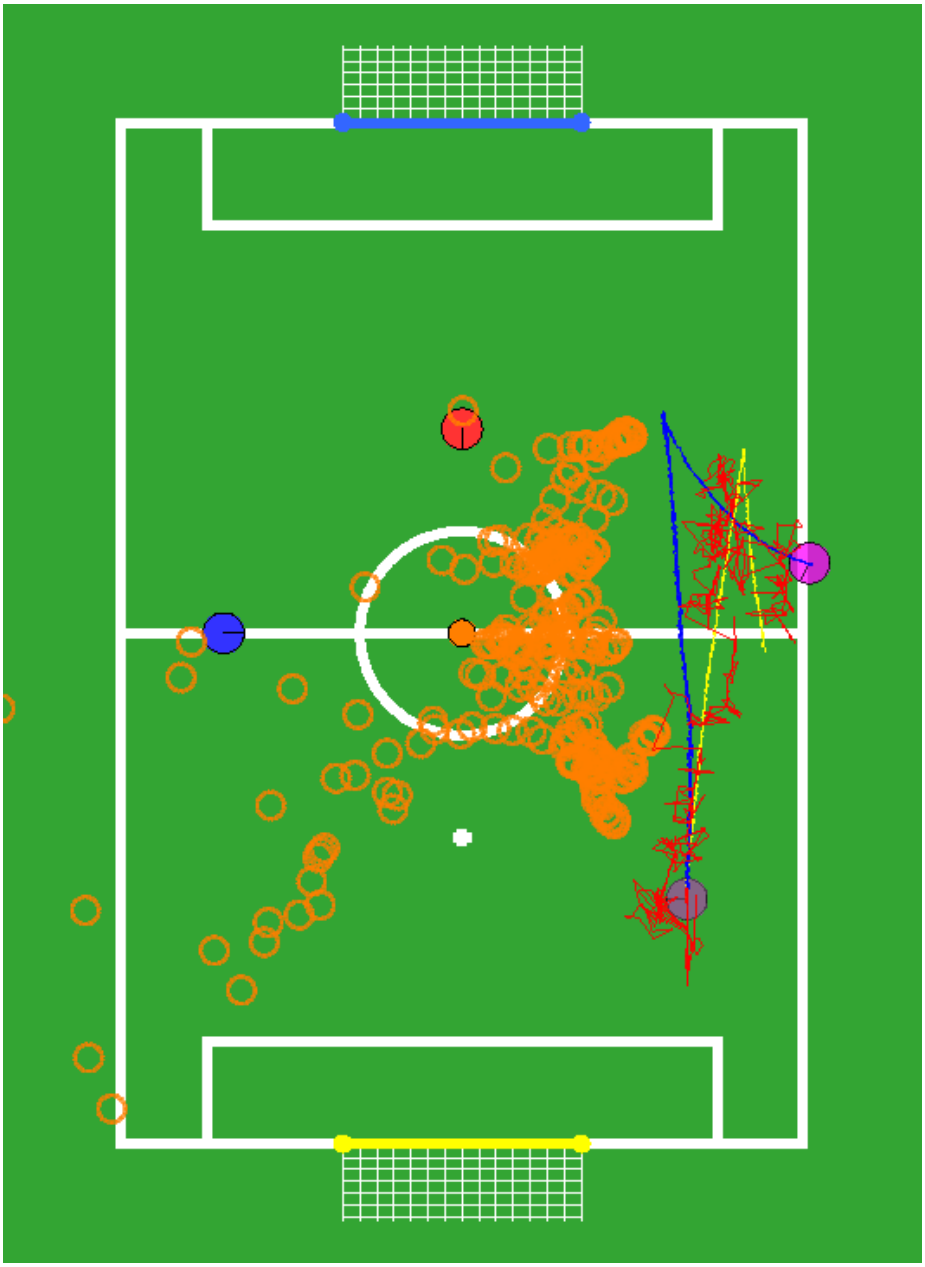


Figure A.17: Run 17: walk sideward: −4 m, +2 m, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red and blue robot icons mark the poses of static robots of the respective color.

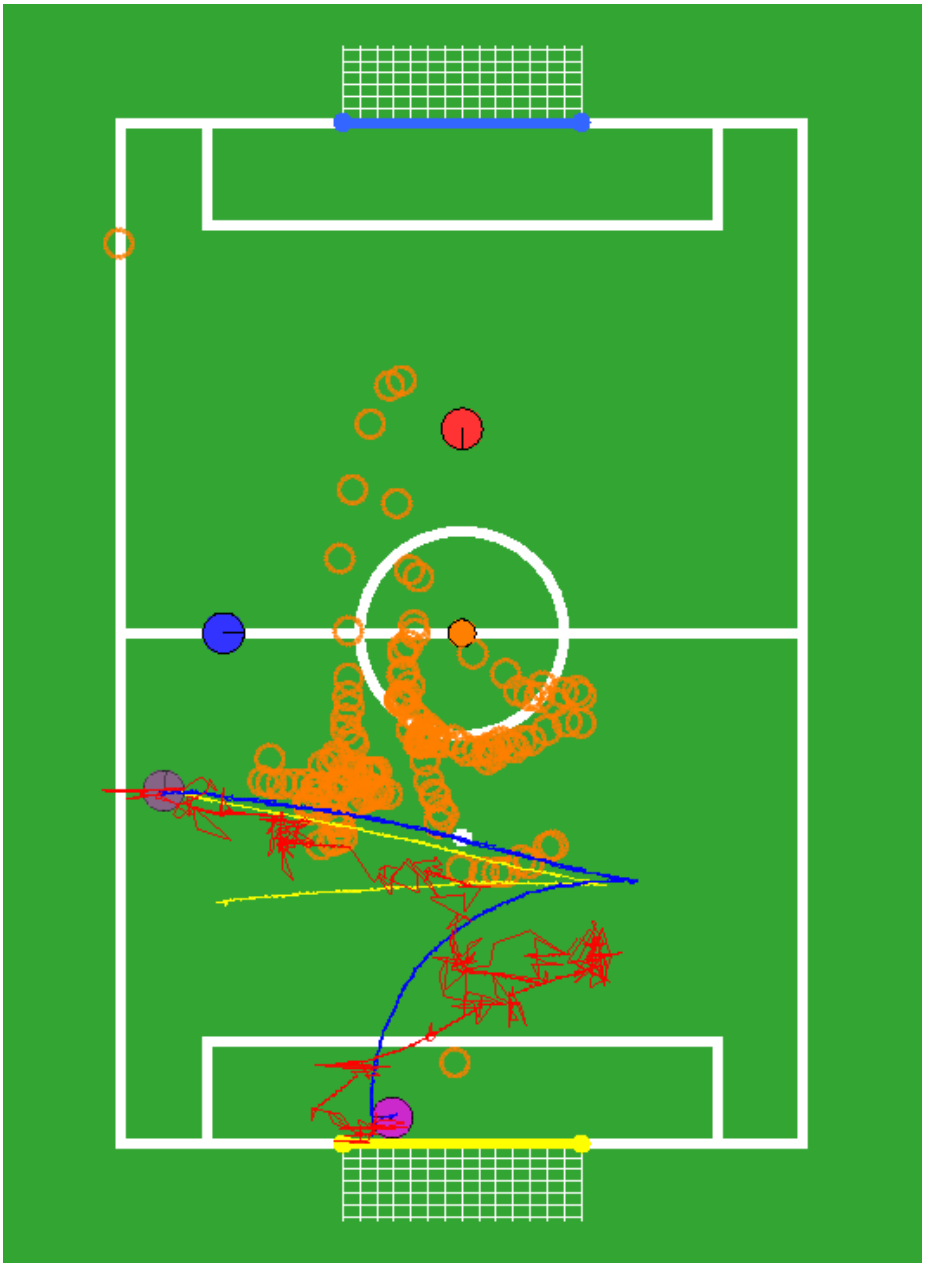


Figure A.18: Run 18: walk sideward: −3 m, +2 m, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter. The actual position of the ball is depicted as an orange filled circle with black outline, the ball position estimates are shown as orange circles. The red and blue robot icons mark the poses of static robots of the respective color.

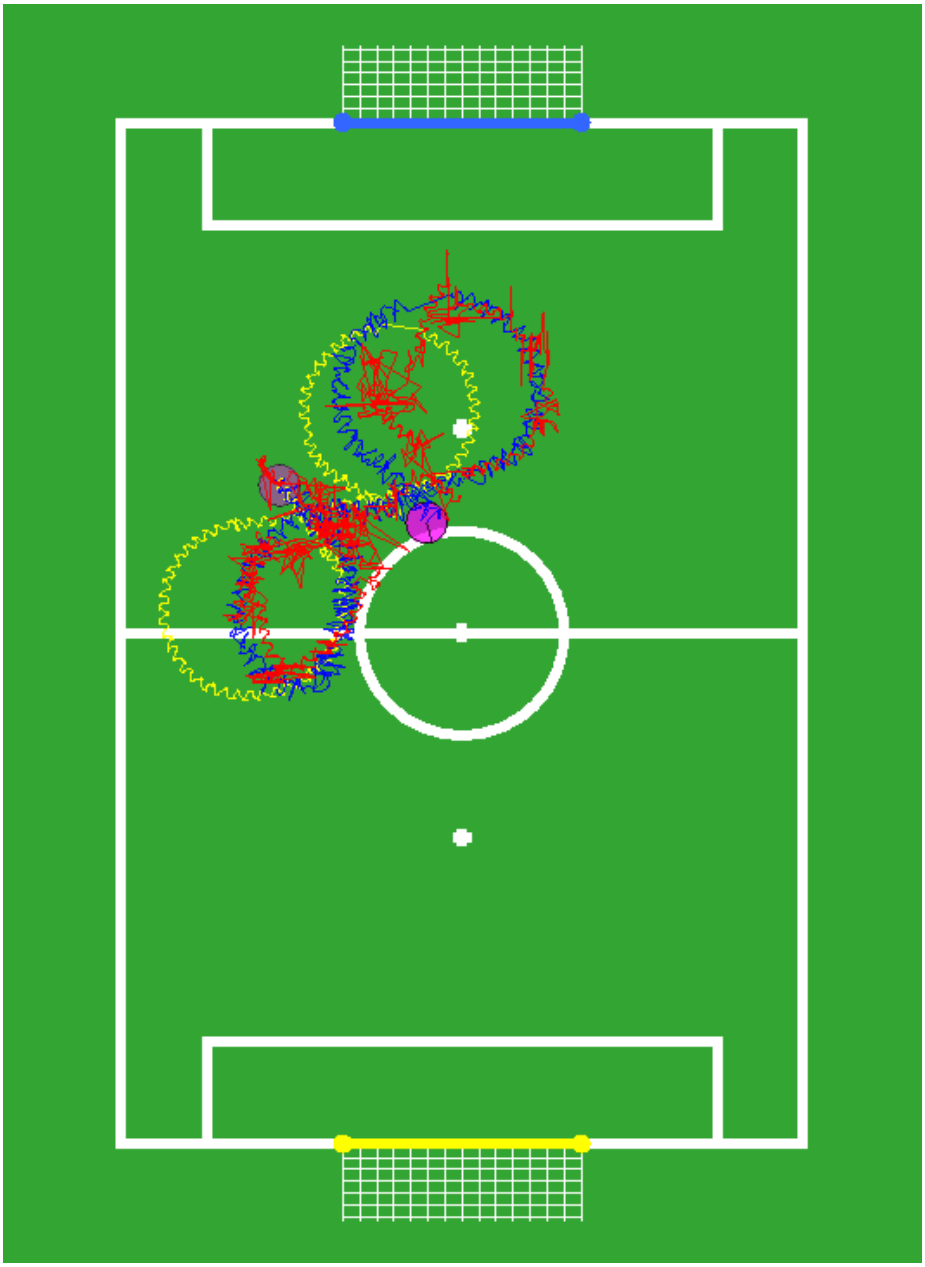


Figure A.19: Run 19: figure-of-eight, moving head. The magenta robot icons show the initial and final pose of the tested robot. The blue line depicts the ground truth trajectory. The yellow line illustrates the integration of the odometry readings, and the red line the estimated pose, as calculated by the Cox algorithm and Kalman filter.

Bibliography

[BJNT06] Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors. *RoboCup 2005: Robot Soccer World Cup IX*, volume 4020 of *Lecture Notes in Computer Science*. Springer, 2006.

[Cox91] Ingemar J. Cox. Blanche – An Experiment in Guidance and Navigation of an Autonomous Robot Vehicle. *IEEE Transactions on Robotics and Automation*, 7(2):193–204, 1991.

[DH55] J. Denavit and R.S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Trans ASME J. Appl. Mech*, 23:215–221, 1955.

[EPS09] Gerhard Eckel, David Pirro, and Gerriet K. Sharma. Motion-Enabled Live Electronics. In *Proceedings of the 6th Sound and Music Computing Conference*, Porto, Portugal, 2009.

[FHL05] Alexander Ferrein, Lutz Hermanns, and Gerhard Lakemeyer. Comparing sensor fusion techniques for ball position estimation. In Bredenfeld et al. [BJNT06], pages 154–165.

[FKN+10] Alexander Ferrein, Tobias Kellner, Tim Niemüller, Patrick Podbregar, Christof Rath, and Gerald Steinbauer. Providing Ground-truth Data for the Nao Robot Platform. Technical report, RWTH Aachen Universitysity, Germany and University of Cape Town, South Africa and Graz University of Technology, Austria, February 2010.

[FPS+09] Alexander Ferrein, Anet Potgieter, Gerald Steinbauer, Tim Niemüller, and Christof Rath. *Fawkes Nao Development by Team ZaDeAt 2009/2010.* University of Cape Town, South Africa and RWTH Aachen University, Germany and Graz University of Technology, Austria, December 2009.

[FSMP08] Alexander Ferrein, Gerald Steinbauer, Graeme McPhillips, and Anet Potgieter. RoboCup Standard Platform League - Team Zadeat - An Intercontinental Research Effort. In *International RoboCup Symposium.*, Suzhou, China, 2008.

[Gut00] Jens-Steffen Gutman. *Robuste Navigation autonomer mobiler Systeme.* PhD thesis, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, 2000.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in computer vision*. Cambridge University Press, second edition, 2003.

[IdFF96] Roberto Ierusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua — An Extensible Extension Language. *Software: Practice and Experience*, 26(6):635–652, 1996.

[Ier06] Roberto Ierusalimschy. *Programming in Lua*. Lua.org, second edition, 2006.

[Kal60] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[Kel06] Alonzo Kelly. Essential Kinematics for Autonomous Vehicles. Technical Report Rev 2.0, The Robotics Institute, CMU, July 2006.

[LLR05] Martin Lauer, Sascha Lange, and Martin Riedmiller. Calculating the Perfect Match: An Efficient and Accurate Approach for Robot Self-Localization. In Bredenfeld et al. [BJNT06], pages 142–153.

[LTC08] Middle-size League Technical Committee. Middle Size Robot League – Rules and Regulations. `http://www.er.ams.eng.osaka-u.ac.jp/robocup-mid/index.cgi?action=ATTACH&page=Rules+and+Regulations&file=msl%2Drules%2D2008%2D12%2D12%2Epdf`, December 2008.

[LTC09a] Humanoid League Technical Comitee. RoboCup Soccer Humanoid League Rules and Setup. `http://www.tzi.de/humanoid/pub/Website/Downloads/HumanoidLeagueRules2009.pdf`, March 2009.

[LTC09b] Small-size League Technical Committee. Laws of the F180 League 2009. `http://small-size.informatik.uni-bremen.de/_media/rules:ssl-rules-2009.pdf`, 2009.

[LTC09c] Standard plattform League Technical Committee. RoboCup Standard Platform League (Nao) Rule Book. `http://www.tzi.de/spl/pub/Website/Downloads/Rules2009.pdf`, January 2009.

[LTC10] RoboCup@Home League Technical Committee. RoboCup@Home Rules & Regulations. `http://www.robocupathome.org/documents/rulebook2010_DRAFT.pdf`, February 2010.

[Mac93] Alan K. Mackworth. On seeing robots. In *Computer Vision: Systems, Theory, and Applications*, pages 1–13. World Scientific Press, 1993.

[May79] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1 of *Mathematics in Science and Engineering*. Academic Press, Inc, 1979.

[Mol09] Ingo Molnar. RT PREEMPT Wiki. `http://rt.wiki.kernel.org`, March 2009.

[Nie09] Tim Niemüller. Developing A Behavior Engine for the Fawkes Robot-Control Software and its Adaptation to the Humanoid Platform Nao. Master's thesis, RWTH Aachen University, Knowledge-Based Systems Group, April 2009.

[NKL09] Alex Nash, Sven Koenig, and Maxim Likhachev. Incremental Phi*: Incremental Any-Angle Path Planning on Grids. In Craig Boutilier, editor, *IJCAI*, pages 1824–1830, October 2009.

[Nod94] Itsuki Noda. Multi-agent soccer game server. In *MACC '94*, October 1994.

[RB93] Martin Riedmiller and Heinrich Braun. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591, 1993.

[Rob06] RoboCup Federation. *RoboCup Soccer Server 3D Manual*, June 2006.

[RV09] Paul E. Rybski and Manuela M. Veloso. Prioritized Multihypothesis Tracking by a Robot with Limited Sensing. *EURASIP J. Adv. Signal Process*, 2009:1–17, 2009.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, September 2005.

[Web98] Website. What is RoboCup. `http://124.146.198.189/overview/21.html`, 1998.

[Web06] Website. Rescue Robots. `http://www.robocuprescue.org/rescuerobots.html`, 2006.

[Web07] Free Software Foundation Website. GNU Lesser General Public Licence. `http://www.gnu.org/licenses/lgpl.html`, June 2007.

[Web09a] Website. Lua Garbage Collection. `http://www.lua.org/manual/5.1/manual.html#2.10`, September 2009.

[Web09b] Website. RoboCup Junior. `http://www.robocupjunior.org`, 2009.

[Web09c] Website. RoboCup Simulation Server. `http://sserver.wiki.sourceforge.net/`, April 2009.

[Web10] Website. RoboCup @Home. `http://www.robocupathome.org/`, 2010.

[Wik08] Wikipedia. Robotics conventions. `http://en.wikipedia.org/wiki/Robotics_conventions`, July 2008.