SocRob Goalkeeper

Group 02gr934

September 1st 2002 – January 20th 2003

Institute for Systems and Robotics
Instituto Superior Técnico
Lisbon, Portugal

Department of Control Engineering
Institute of Electronic Systems
Aalborg University
Denmark

The Faculty of Engineering and Science

Aalborg University

Institute of Electronic Systems

TITLE:
SocRob Goalkeeper

PROJECT PERIOD:
IAS9
September 1st - January 20th

PROJECT GROUP:
02gr934

GROUP MEMBERS:
Hans H. Lausen
Jakob Bro Nielsen
Michael Schioldan Nielsen

SUPERVISORS:
Thomas Bak (AAU)
Pedro M. U. A. Lima (IST)
Roozbeh Izadi-Zamanabadi (AAU)

Issues: 7

Pages: 92

Abstract:
This project, which was written at ”Instituto Superior Técnico” (ISR) in Lisbon, Portugal, addresses the improvement of a goalkeeper for the ”RoboCup Middle Size League”. The goalkeeper is one of four robots in the team at ISR. Each of the robots are controlled using a set of behaviours according to the role of each robot in the team. The set of behaviours and their relations is defined in a finite state machine.
The first step of the project is to define such a state machine. The state machine enables the goalkeeper to track an arc in front of the goal and thereby minimize the exposed goal area. The state machine is partly implemented, and furthermore an algorithm to determine the position and orientation of the goalkeeper in the field is implemented. Since the group wasn’t able to do tests playing against other teams, the state machine is simulated to determine parameters for the implementation. The simulations are developed using the freeware software CheckMate.
The tests reveals that the implementation is working as expected.

Preface

This project is written by a project group from Aalborg University (AAU), Denmark during a stay at ”Instituto de Sistemas e Robotica” (ISR) located at the Lisbon Campus of ”Instituto Superior Técnico" (IST). The work was done under a Socrates grant from October 7^{th} 2002 to January 20^{th} 2003.

The project is written to satisfy the needs of two parties: The SocRob-group at ISR (specific technical details making future improvements easier) and the project supervisors at AAU (general, methodical aspects used for evaluation of the work).

The project group wishes to offer their gratitude to the SocRob-group for all the help, suggestions and feed-back, and to assistant professor Pedro Lima (ISR) and associated professor Thomas Bak (AAU) for arranging the contact making the stay possible.

A CD is enclosed to this project and references to the files are done like this: [CD, \dir].

Hans H. Lausen   Jakob Bro Nielsen   Michael S. Nielsen

Contents

1 Introduction 1

1.1 Problem Specification . . . 2
1.2 About the SocRob Project . . . 2
1.3 About the RoboCup: Rules and Regulations . . . 2
1.4 The Present Implementation . . . 3
1.5 The Proposed Improvement of the Goalkeeper . . . 4
1.6 Reading Instructions . . . 5

2 Configuration of Goalkeeper 7

2.1 Hardware . . . 7
2.2 Software . . . 9

3 State Machine 13

3.1 Overall State Machine . . . 13
3.2 Behaviours . . . 14
3.3 Modifications . . . 16

4 Motion Control 19

4.1 Kinematics . . . 19
4.2 FollowBall . . . 20
4.3 InterceptBall . . . 32

5 Self Localization 41

5.1 Introduction . . . 41
5.2 The Algorithm . . . 43
5.3 Implementation Issues . . . 47
5.4 Results . . . 50
5.5 Conclusion . . . 56

6 State Machine Simulations 59

6.1 Introduction to Hybrid Automata . . . 59
6.2 Design of Hybrid Automata . . . 60
6.3 Implementation - CheckMate . . . 63
6.4 Results . . . 64
6.5 Conclusion . . . 68

7 Joint Test of Behaviours 69

8 Conclusion 71

APPENDIX 74

A Control of Motors 75

B The Hough Transform 77

B.1 Example 78

C Geometric calculations 79

C.1 Line and Circle Intersection 79

C.2 Robot Position in Goal Coordinates 80

D CheckMate 83

D.1 Introduction 83

D.2 Switched Continuous System Block (SCSB) 83

D.3 Polyhedral Threshold Block (PTHB) 85

D.4 Finite State Machine Block (FSMB) 85

D.5 Userdefined Files 87

D.6 Simulating the Model 88

Bibliography 92

CHAPTER 1

Introduction

In the recent years, as the hardware for mobile robots has been less expensive, the case of having multiple robots cooperating to accomplish tasks has become increasingly interesting for research and industry, with applications to areas such as deep water exploration, building surveillance, transportation of large objects and rescue-operations after large-scale disasters. In short it is called cooperative robotics, and is defined as a population of robots that are behaving as one distributed robot to accomplish tasks that would be difficult (or impossible) for one single robot.

The area of cooperative robotics is of special interest to the project group since the group is following the masters program in "Intelligent Autonomous Systems" at Aalborg University in Denmark.

The coordinated execution of a robotic task is one of the key features of cooperative robotics. The resources of each robot (e.g., sensors, actuators, shared memory, CPU) required for a given task to be completed must be properly managed and articulated with the different behaviours composing the task at each of the robots.

Soccer Robots participating in the RoboCup-tournaments is an example of cooperative robotics. In the game soccer robots in various leagues according to their various sizes compete according to rules that are overall similar to humanoid soccer rules. The soccer robots are facing extensive challenges in the coordinated execution of robotic tasks as mentioned above, and in fact pose additional challenges, as most of their tasks must be performed within dynamic environments.

In the RoboCup Middle-Size League one of the four allowed players is dedicated as goalkeeper. The goalkeeper switches between different behaviours to fulfil its role in the team. Examples of these behaviours are

- Cover the goal when the ball is not dangerously close to the goal or approaching it fast.
- Intercept the ball when the ball is headed towards the goal.
- If the ball is under control, kick it away.
- Leave the goal to tackle an opposed robot in control of the ball.

At ”Instituto de Sistemas e Robotica” (ISR) located at the Lisbon Campus of ”Instituto Superior Técnico" (IST) a team of robots competing in the RoboCup Middle-Size League has been under development since 1997 in the project called SocRob. The main work of development over the years has been concentrated on the attacking players and the defender instead of on the goalkeeper.

1.1 Problem Specification

The fact that the goalkeeper is less developed compared to the more sophisticated attackers and defender, makes the aim of this project:

To improve performance of the goalkeeper in the team at ISR, enabling it to defend the goal better (decreasing the number of goals scored) than the previous implementation.

1.2 About the SocRob Project

The SocRob project is a project on Cooperative Robotics and Multi-Agent Systems carried out by the Intelligent Systems Laboratory at ISR/IST since 1997. SocRob is an abbreviation for both ”Society of Robots” and ”Soccer Robots”. The project is considered as a case study testing a population of four robots playing football, [Lima, 2002b].

The case study consist of the challenging problem of having robots not only shoot a ball towards the goal, but also detect and avoid static (e.g. stopped robots or the goals) and dynamic (e.g. moving robots) obstacles. In addition to that the robots must cooperate to defeat an opposing team of robots.

The project started out with the development of home made robots called SocRob. These robots were used for two years. In 1999 a robot from the company Nomadic (the model ”Super Scout II” - Scout for short) was bought. The robot was very successful compared to the homemade ones, so at present all four robots are Scouts. The specifications of the scouts and the equipment added to them will be described in details later.

The behaviours of each of the robots in the SocRob-project are implemented using state machines written in ANSI C. Each robot has a role at the top level (e.g., goalkeeper), and each role consists of a number of behaviours, and each behaviour of one or more primitive tasks.

1.3 About the RoboCup: Rules and Regulations

The SocRob project participates in tournaments in the RoboCup ”Middle Size League (F2000)” organized by the ”The RoboCup Foundation”. Besides the ”Middle Size League (F2000) ” there are four other leagues:

- ”Simulation League” where the are no robots, instead it is all simulated in computers.
- ”Small-Size League (F-180)” with robots that can fit into a cylinder 18 cm. wide and with a ball 43 mm. in diameter.

- ”Sony Four Legged Robot League” where the robots are the well known ”dogs” from Sony.
- ”Humanoid League” with legged robots.

The rules and regulation of ”Middle Size League” can be found in [Robocup.org, 2002], and the most important rules will be stated in the following.

The game is played in a green field up to 10 m long and up to 5 m wide. There are two goals (one with yellow background and one with blue) 2 m wide, 0.4 m deep and 0.9 m tall. The width of internal lines are 5 cm and 12.5 cm for border lines. The field is limited by poles (50 cm high, 10 cm diameter) fixed to the ground one meter from the field border. The poles are spaced 40 cm from each other, and the poles are white on the upper 30 cm and black on the lower 20 cm. An illustration of the field can be seen in figure 1.1.

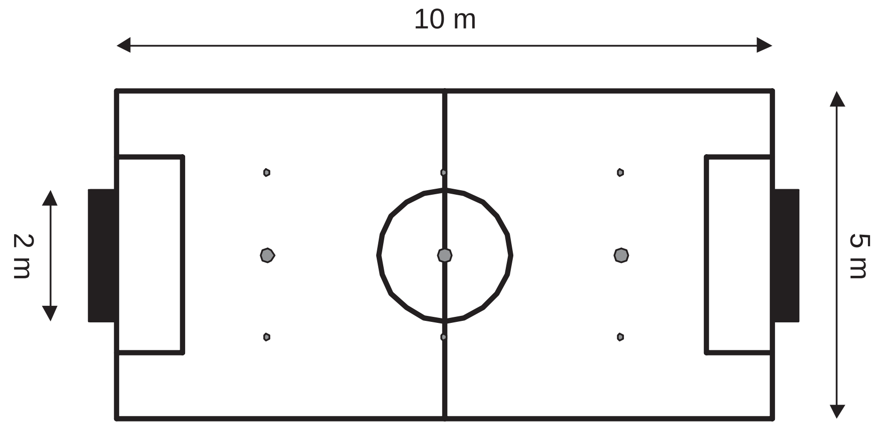


Figure 1.1: The field used in ”Middle Size League”. The six small dots are used as starting points for the ball if the robots keep pushing it outside the field.

1.4 The Present Implementation

At the current stage of development for the goalkeeper, the basic behaviours is specified below, and is also illustrated in figure 1.2:

- The goalkeeper moves sideways on a line in front of the goal according to the position of the ball and should never leave this line.
- The goalkeeper intercepts the ball at the point where it is estimated to cross the goal line.
- If the goalkeeper saves a shot on goal and gains control of the ball, it is unable to kick the ball away. So a defender has to pick up the ball at the goalkeepers position and remove it.
- The goalkeeper is unable to determine its own position in the field in case it gets "lost".

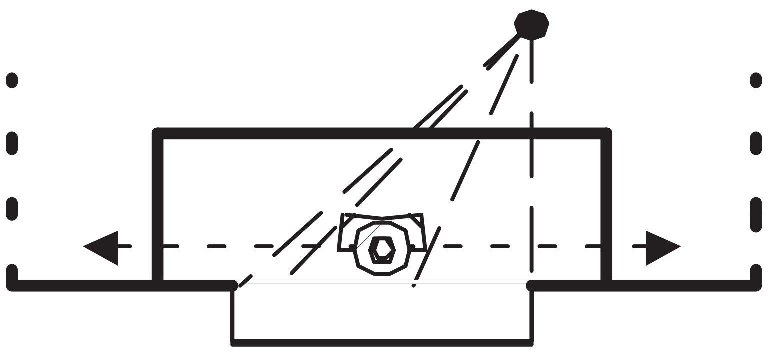


Figure 1.2: The present solution of the goalkeeper. The goalkeeper is moving on a line in front of the goal and is intercepting the ball at the goal line. The dashed lines indicates the exposed goal area for the opponent.

The goalkeeper is unable to kick away the ball simply because it has no mechanical kicking-device as the other robots do. So in the case the goalkeeper intercepts a ball, one of the other robots are required to pick it up at the goalkeeper and remove it from the goal area.

The robots in the SocRob-team are tracking their own position in the field using odometry from the wheels. This only works as long as the robot doesn't get any bumps (e.g, from collisions with other robots), as long as the wheels aren't spinning because of lack of friction or as long as both wheels are in contact with the field at all times. It sometimes happens during games that a robot ”hangs” on the side of another robot thereby making the odometry wrong. From experienced human team-members we learned that this is a problem for the goalkeeper in particular, since the area in front of the goal is often crowded with friendly and opponent robots.

1.5 The Proposed Improvement of the Goalkeeper

To improve the performance of the goalkeeper it is the intention of this project to implement solutions that will make the goalkeeper in the SocRob-project able to:

- Move on an arc with center in the middle of the goal to cover the goal better and thereby reducing the exposed goal area for the opponent.
- Intercept balls moving towards goal on a straight line in front of the goal.
- Determine its own position in the field.

discussed to implement a kicking device on the robot i the near future. The first solution is illustrated i figure 1.3. The ideas of the improved solution is based on the work of Giovanni Indiveri in [Indiveri, 2001] and [Indiveri, 2002].

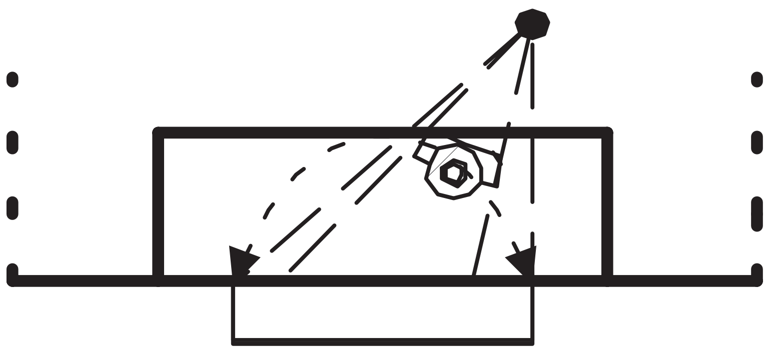


Figure 1.3: The first part of the proposed solution. The goalkeeper is guarding the goal driving on an arc and thereby reducing the exposed goal area for the opponent.

1.6 Reading Instructions

In order to make the further works of this project more obvious to the reader, the outline of the project is presented in the following. First in order to get the foundation of further works the configurations of the goalkeeper hardware and software are given in chapter two. In chapter three a state machine capable of fulfilling the intended tasks are designed, and the behaviours of the state machine are designed and implemented in chapter four. In chapter five the goalkeepers ability to determine its own position in the field (called selflocalization) is designed and implemented. The state machine designed in chapter two is simulated in chapter six, and a joint test of all the implementations are performed in chapter seven. The conclusions of the project are offered in chapter eight.

CHAPTER 2

Configuration of Goalkeeper

The team of robots in the SocRob project consists of four Nomadic Super Scout II. Three of the scouts are used for playing in the field, and the fourth is used as goalkeeper. In general the four scouts are configured identically but differences exist between the three field players and the goalkeeper. In this chapter an introduction to the goalkeeper will be given. First a description of the hardware will be given, followed by an introduction to the embedded software. The chapter is in part based on [Lima, 2002a].

2.1 Hardware

The goalkeeper is seen in figure 2.1. It is propulsed by two wheels. Normally the goal-

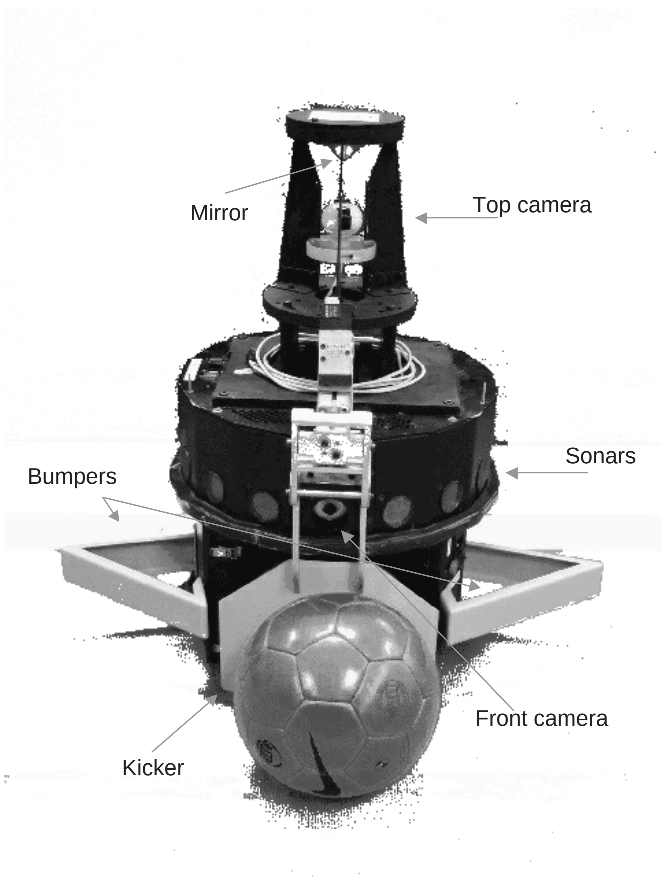


Figure 2.1: The goalkeeper.

keeper is driven from side to side in front of the goal, while facing it's front camera straight ahead, pointing away from the goal. The front camera is placed in the middle of the robot. To be able to cover the goal well, bumpers are attached to the side of the robot. All the way around and equally spaced, fifteen sonars are mounted. On the top of the goal-keeper the omni-directional catadioptric system is installed. In short, this is a camera placed underneath a cone-shaped mirror. This gives the robot a bird's eye view of the field. In figure 2.2 a more detailed image of the omni-directional catadioptric system is seen. In the front of the goalkeeper a kicking device to kick the ball was recently installed.

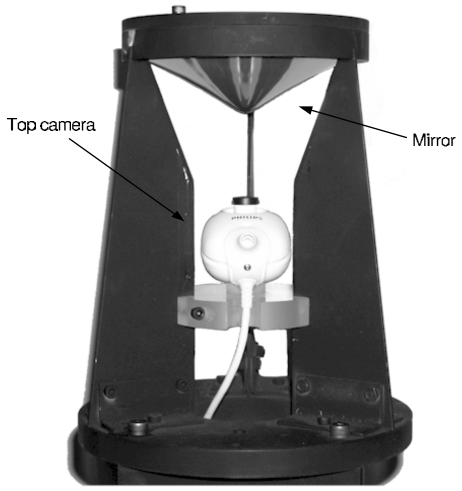


Figure 2.2: The omni-directional catadioptric system.

2.1.1 Generel Characteristics

The goalkeeper has the following characteristics:

- Two-wheel (driven by high torque Pittmann GM14902E032 DC motors, with 512 ppr encoders) differential drive kinematics.
- 15 sonar sensors equally distributed around the robot.
- Pentium 233 MHz based motherboard, 64 MB RAM, 8 GB hard drive (2,5"), one PCI and one PC-104 bus connector.
- M68k-based daughter board with three-axis motor controller, sonar, bumper interface (currently not in use) and battery level meters.
- Two 12V batteries, 18Ah capacity.
- Phillips 740K pro, USB digital color camera, with a 1/4" CCD and a 3 mm. lens.
- Omni-directional catadioptric system including a Philips 740K Pro camera installed under a mirror (11 cm. diameter), manufactured to capture a birds eye view of the soccer field.

- Pneumatic kicking device, based on Festo components, plus one bottle for pressurized air storage.
- Lucent WaveLAN/IEEE Turbo 11Mbps (Silver) wireless Ethernet PCMCIA boards (IEEE 802.11b standard) connected through a PC104/PCMCIA bridge.

2.2 Software

The software structure is the same in all the robots, and the same code is implemented in all four robots. To maintain the same source code in all four robots, the source code is version controlled using CVS. The scouts are running Linux, and because the scouts doesn't have a display nor a keyboard, the graphical interface X is forwarded to other PC's using ssh for development. Since all the four robots should not perform the same tasks, each robot is assigned a role at startup of a game as stated in the introduction. Each role consists of one or more behaviours, and the behaviours are based on primitive tasks. The details of this hierarchy will be addressed in section 2.2.1. The functional architecture is implemented using micro-agents described in section 2.2.2 and a memory shared between the scouts described in 2.2.3.

2.2.1 Functional Architecture

The functional architecture is based on the concepts of roles, behaviours, primitive tasks and guidance primitive functions. Figure 2.3 shows how the four layers interact with each other. At the top level, a role is assigned to each robot. In this case, the role as goal-

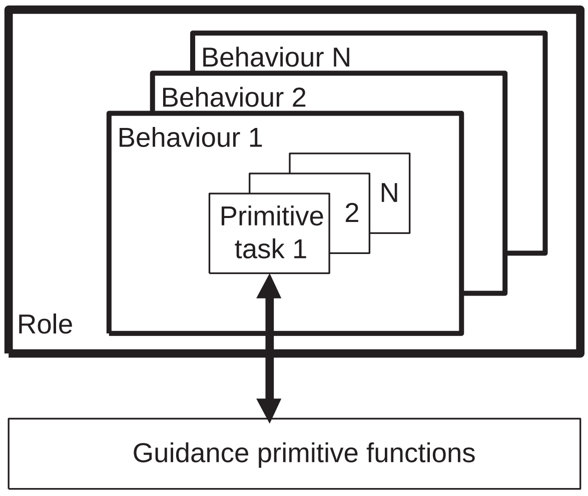


Figure 2.3: The functional hierarchy of the software. Each robot is assigned a role, which consists of one or more behaviours. Behaviours are implemented using one or more primitive tasks, and the primitive task uses the Guidance primitive functions to interact with the lowest level of hardware on the robot.

keeper is assigned to the robot in question. Other roles are attacker and defender. The role is filled by executing a behaviour according to the current situation, and is executed by running a number of primitive tasks one at the time. In case of the goalkeeper a behaviour could be to intercept the ball, and that behaviour could consist of primitive tasks to estimate a point of intersection with the ball and moving to that point. The primitive tasks are implemented using functions from the guidance primitive functions. These are considered as a function library containing functions to e.g. set motor velocity and the like. In the SocRob-project each behaviour is implemented as a finite state automaton whose states are the primitive tasks. The implemented behaviour is called a plugin.

2.2.2 Micro-agents

The software architecture of the SocRob-project is based on micro-agents (a Linux thread running continuously) and a distributed blackboard (BB). There are eight micro agents available in the source code: `Machine`, `Control`, `Vision`, `Halt`, `Kicker`, `Proxy`, `Relay`, `X11`. These microagents can also be found in the robots in terms of directories. In the following a brief description will be given to each of these agents.

Machine Coordinates the different available behaviours (in the `Control` micro-agent) by selecting them one at a time. The chosen behaviour is communicated to the `Control` micro-agent.

Control Receives the behaviour selection message from the micro-agent `machine` and runs the selected behaviour. This micro-agent can also select the vision modes (e.g. up, self, front, emptyspot) by communicating this information to the `Vision` micro-agent.

Vision Reads images from one of two devices. This micro-agent can have different modes and currently four modes are available: `up` - to detect the ball with the up cam, `front` - to detect the ball and the goals with the front cam, `self` - for self-localization with the up cam, and `emptyspot` - with the up cam to determine the region around the robot with the largest amount of green.

Kicker Sends open and close signals to the electronics of the kicker electro-pneumatic valve.

Halt This micro-agent was formerly used to read the device `psaux`, where an optical mouse was connected. The mouse used to detect weather the robot was in motion or not.

Proxy Handles the communications of a robot with it's teammates using TCP/IP sockets. This connection is wireless and is typically used to send shared variables to the blackboard (BB).

Relay Relays the BB information on the state of each robot to the game interface running on an external computer, using TCP/IP sockets. The game interface is a GUI-program running in realtime.

X11 This micro-agent handles the X11-specific information sent by each robot to the external computer, using TCP/IP sockets. It is typically used to send shared variables for text display in an X-window.

2.2.3 Distributed Blackboard

The distributed blackboard is a collection of shared variables accessible to several agents. SocRob's distributed blackboard consists within each individual robot of memory shared among different micro-agents. Some of the variables are local, meaning that the associated information is only relevant for the robot, where the information was acquired and/or processed. Other variables are global, and when updated, these must be broadcasted to the other robots in the team.

2.2.4 Example

Having described the subsystems of the robot, the intention of this section is to clarify how the subsystems are working together. With all the code in the robot compiled and ready to run, there is basically not more to it, than to execute `main`. When `main` is executed, all the micro-agents are started and initialized. The finite state automaton describing the behaviours of the role is started in the micro-agent `machine`. `machine` then determines, using the state machine, which behaviour should be run. It puts this information on the blackboard as a variable and waits for the micro-agent `control` to change the behaviour. The behaviour is then executed by `control` in a loop. The loop runs until `machine` puts a new variable on the blackboard indicating a change of behaviours. The behaviour running in `control` puts information about success and failure on the blackboard. This information is read by `machine`, which uses it to determine if the behaviour should keep running or a change of behaviours is needed. If the behaviour isn't changed nothing is written on the blackboard, and thus `control` continues to run. If it is changed the procedure starts over again with `machine` putting the information of which behaviour should be run on the blackboard. So to sum up:

1. `main` is run, the robot is assigned a role, and all micro-agents are started and initialized.
2. The state machine describing the role is started in `machine`.
3. The behaviour that should be run is put on the blackboard by `machine`.
4. This information is read by `control`, which changes behaviour accordingly.
5. The behaviour is run in a loop.
6. The behaviour (running in `control`) supplies `machine` with information about success and failure through the blackboard.

7. Based on these informations, `machine` may or may not decide to change behaviour.
8. If it is decided to change behaviour, steps 3-8 are repeated.

CHAPTER 3

State Machine

In the following a description of the new state machine is given. There are five behaviours in the new state machine. The five behaviours are described separately in section 3.2. Based on the information of some predicates, transitions between the behaviours can occur. Each behaviour can be divided into one or more primitive tasks. Each behaviour has its own state machine where each state correspond to a primitive task and each of these primitive tasks make use of the guidance functions.

3.1 Overall State Machine

Figure 3.1 on the following page shows the five states in the overall state machine. These states are the behaviours: `Go2Area`, `InterceptBall`, `Wait4Ball`, `KickBall` and `FollowBall`. Transitions between these behaviours happens only when the predicates between becomes true. There are a number of predicates and these are explained in section 3.1.1.

3.1.1 Used Predicates

`ball_lost` The goalkeeper doesn't know the position of the ball.

`ball_in_dangerzone` The ball is positioned inside the predefined danger zone.

`ball_in_front` The ball is right in front of the goalkeeper.

`ball_moving_away_goal` The ball is moving away from the defending goal.

`ball_moving_towards_goal` Opposite of `ball_moving_away_goal`: The ball is moving towards the goal defended by the goalkeeper.

`gk_far_goal` The goalkeeper is located outside the minimum defensive arc.

`gk_near_goal` Opposite of `gk_far_goal`. The goalkeeper is located inside the minimum defensive arc.

`success` Indicates the success of an event.

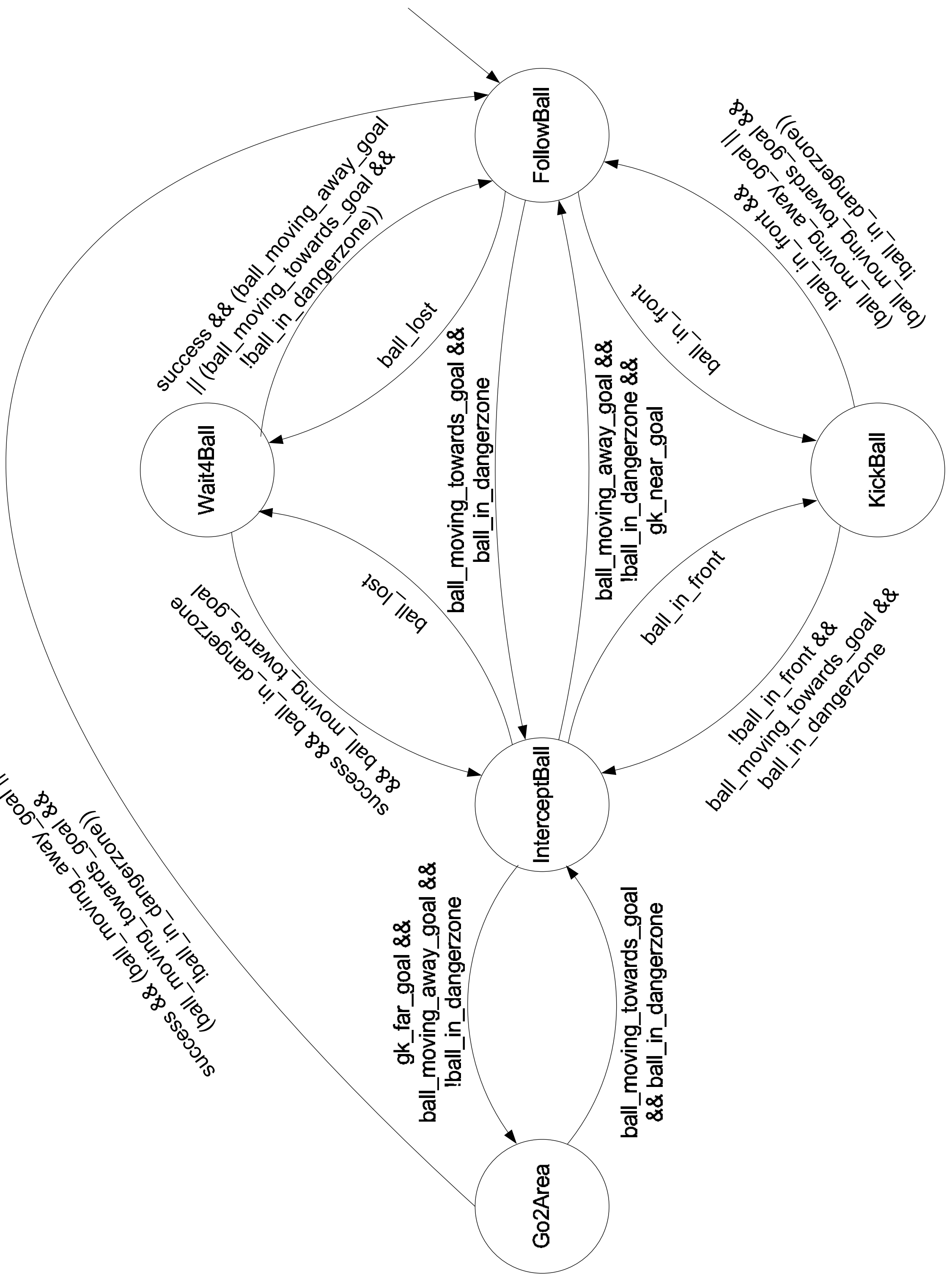

FollowBall
Wait4Ball
KickBall
InterceptBall
Go2Area
success && (ball_moving_away_goal || (ball_moving_towards_goal && !ball_in_dangerzone))
ball_lost
ball_moving_towards_goal && ball_in_dangerzone
ball_moving_away_goal && !ball_in_dangerzone && gk_near_goal
ball_in_front
!ball_in_front && (ball_moving_away_goal || ball_moving_towards_goal && !ball_in_dangerzone))
ball_lost
success && ball_moving_towards_goal && ball_in_dangerzone
ball_in_front
!ball_in_front && ball_moving_towards_goal && ball_in_dangerzone
success && (ball_moving_away_goal || (ball_moving_towards_goal && !ball_in_dangerzone))
gk_far_goal && ball_moving_away_goal && !ball_in_dangerzone
ball_moving_towards_goal && ball_in_dangerzone

Figure 3.1: The goalkeeper state machine.

3.2 Behaviours

In this section each of the five behaviours are described in further detail.

3.2.1 FollowBall

In `FollowBall` the ball is in the danger zone specified in figure 3.2. The ball is not considered as a big threat, but the goalkeeper has to be able to handle a shot from the distance. The goalkeeper follows the ball while tracking an arc with adjustable radius and center in the goal. Also in figure 3.2 the principle of the defensive arc is illustrated. The `r_min` and `r_max` corresponds to the minimum and maximum radius of this arc.

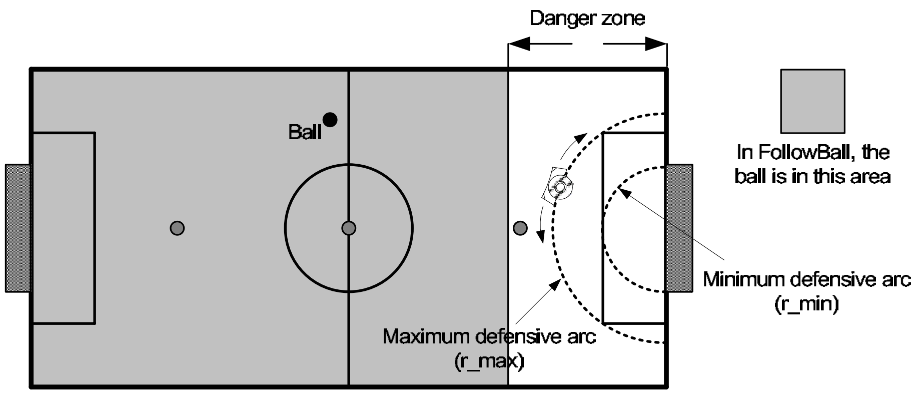

Danger zone
Ball
In FollowBall, the ball is in this area
Minimum defensive arc (r_min)
Maximum defensive arc (r_max)

Figure 3.2: `FollowBall`: Principle of the goalkeeper following the defensive arc in front of the goal.

This adjustment of the radius is dependent of the distance between the ball and the goal. When the ball is near the border of the danger zone, the radius of the arc is at its minimum. When the ball is at the center line or further away, the radius is at its maximum. The goalkeeper never moves away from the arc in `FollowBall`, wether it is at the `r_min`, `r_max` or in between.

3.2.2 InterceptBall

In the behaviour `InterceptBall` the ball is in the danger zone and moving towards the goal as seen in figure 3.3. The goalkeeper is not following the arc as in `FollowBall` but moving on a line in front of the goal. On the line the goalkeeper will try to intercept an incoming ball.

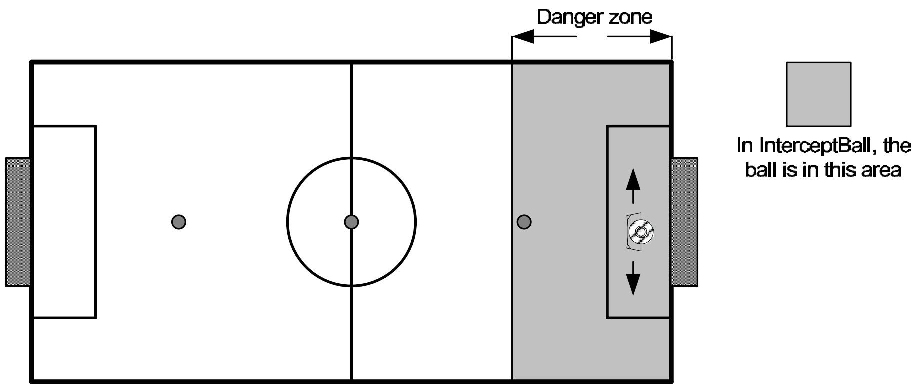


Figure 3.3: `InterceptBall`: The ball is moving towards goal and is in the dangerzone

3.2.3 KickBall

If the goalkeeper gets control of the ball it should try to kick it away. The goakeeper should direct the shot towards an unoccupied area so it won't bounce back.

3.2.4 Go2Area

The purpose of this behaviour is to bring the goalkeeper straight back if it has moved far away from the goal. Go2Area should bring the goalkeeper back to inner point within the smallest arc `r_min`. From here the goalkeeper can return to a defending behaviour.

3.2.5 Wait4Ball

A shift from either `InterceptBall` or `FollowBall` to the behaviour `Wait4Ball` can occur if the goalkeeper can not see the ball. Only when the ball is spotted again a shift back into the former behaviours can be made. When the `Wait4Ball` is executed the goalkeeper will try to spot the ball, but it will also return to a safe location in front of the goal.

3.3 Modifications

In the previous sections a state machine for the behaviours and their primitive tasks was designed. This state machine was designed from ideal conditions and with limited knowledge of the software framework in which it was to be implemented. As the implementation of the first behaviour started to take shape, it became obvious that the state machine would have to be modified if the goalkeeper was to have a decent performance. This because of implementation constraints, which will become more clear in the following chapter, where the control algorithms for the behaviours are designed and implemented. Though the reasons behind a modified design of the state machine yet has to be clarified, the modified design is presented in the following section, so a coherent design of the behaviours can be given in the next chapter.

In spite of changes in the designed state machine the modified version should conserve the described behaviours from section 3.2. The changes can be divided into the two implementation layers: Behaviours and primitive tasks.

3.3.1 State Machine of Behaviours

In the level of behaviours `Wait4Ball` has been removed and a new behaviour `SelfLocalize` has been introduced. Furthermore some of the predicates have been removed. The modifications can be seen in figure 3.4, where the modified state machine is presented. Most of the changes has to do with time consumption executing the code. In the original design (figure 3.1) it was intended that the self localization algorithm should be integrated in the control loop, so it isn't in the state machine as a behaviour. This wasn't

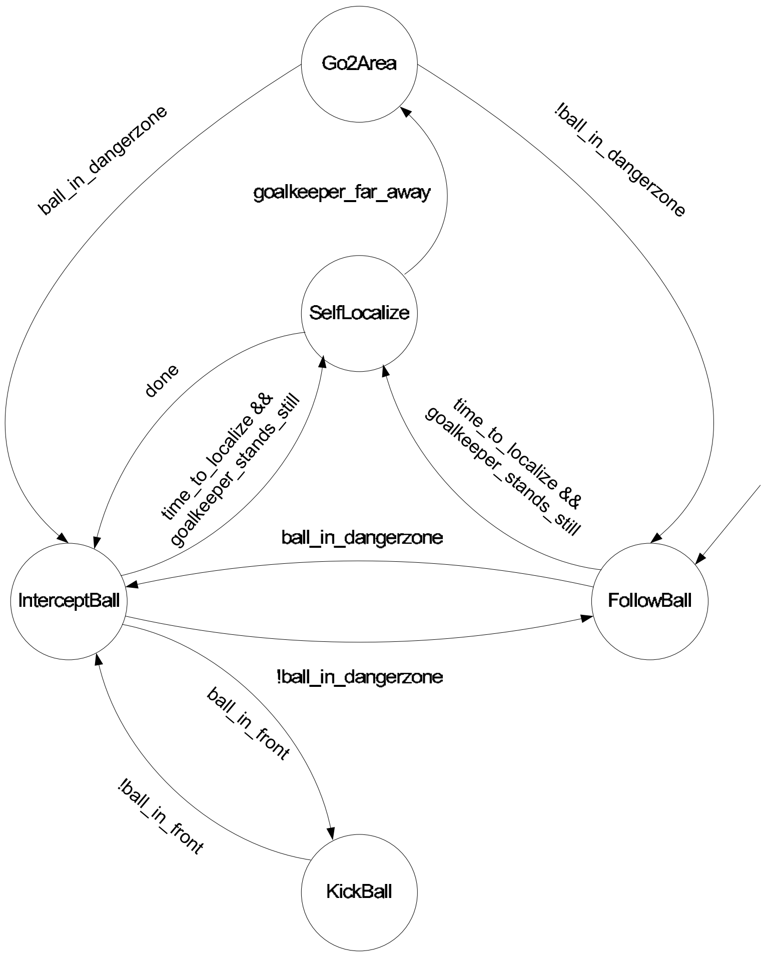


Figure 3.4: Modified state machine for the goalkeeper.

feasible partially because of the framework with the micro-agents and partially because of the time needed to make a self localization. The behaviour Wait4Ball was removed and implemented as a primitive task in the behaviours FollowBall and InterceptBall because this saved the time needed to change behaviour.

In figure 3.4 the state machine never returns to FollowBall after a run of SelfLocalization. The reason for this is that the software framework doesn't allow the machine micro-agent to use the functions needed to evaluate a predicate of the ball position. The state machine is without memory of the state it was in prior to the self localization. Therefore the state machine always returns to InterceptBall to obtain a quick reaction from the goalkeeper in the return to a situation where the ball is close to goal.

3.3.2 State Machines of Primitive Tasks

The state machine for FollowBall is showed in figure 3.5 and the state machine for InterceptBall in 3.6. The function of each of the primitive tasks are explained in the next chapter. The implemented state machine can be found in [CD, \source\machine\goalkeeper.c].

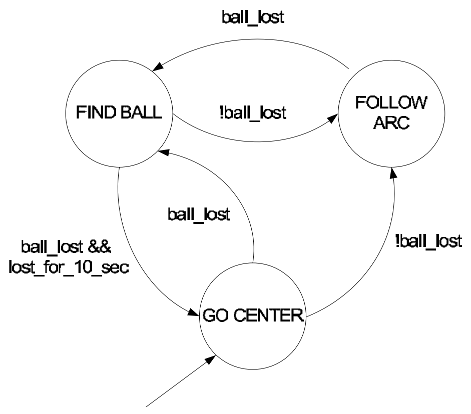


Figure 3.5: State diagram of primitive tasks for FollowBall.

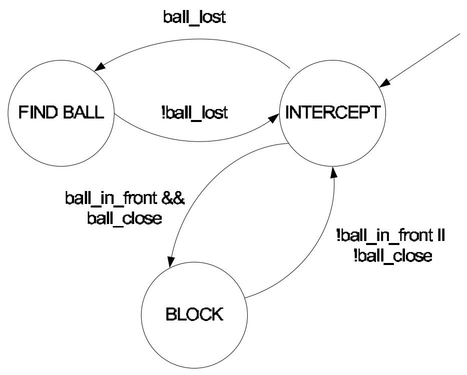


Figure 3.6: Sate diagram of primitive tasks for InterceptBall.

CHAPTER 4

Motion Control

The subject of this chapter is the control algorithm used for motion control of the goal-keeper. Because of the different behaviours of the goalkeeper both trajectory tracking and posture stabilization problems are posed. Thus each behaviour will have its own control algorithm, which will be derived in following sections. In order to restrict the problems the control algorithms will not take obstacle avoidance into account. This limitation can be somewhat justified because none of the designed behaviours involve dribling where obstacle avoidance truly is a necessity. Before any of the control algorithms can be derived, it is necessary to look at the kinematics of the goalkeeper.

4.1 Kinematics

The two-wheeled differentially driven robots used in the SocRob can be described by the same kinematic equations as the unicycle. The unicycle is a nonholonomic system which has some constraints. For the unicycle these constraints arise from the no slipping condition and no movement in the lateral direction. Let $q = (x, y, \phi)^T$ be the vector that describes the position and the orientation of the goalkeeper in the plane Q, see figure 4.1. That is $q \in Q = R^2$. The first-order kinematic model for the unicycle and hence the

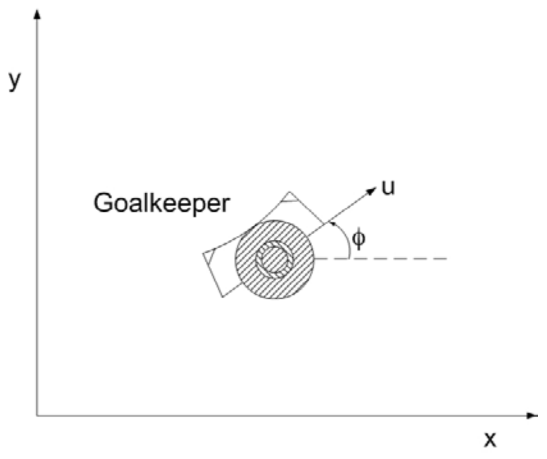


Figure 4.1: The posture of the goalkeeper is described by the three states x, y and ϕ.

goalkeeper is [Giuseppe Oriolo and Vendittelli, 2002]:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} u\ \cos(\phi) \\ u\ sin(\phi) \\ \omega \end{bmatrix} \tag{4.1}$$

Where u is the linear velocity and ω is the angular velocity. Furthermore u and ω are the control inputs.

4.2 FollowBall

The movement on an arc in front of the goal is a trajectory tracking problem. Different control algorithms such as dynamic feedback linearization, linear feedback design or non-linear feedback design can be used to solve the problem of tracking an arc. Based on the results from [Giuseppe Oriolo and Vendittelli, 2002] linear feedback design doesn't seem to give as good performance as the two others. By advice from associated professor Pedro Lima it was decided to choose a control algorithm based on nonlinear feedback control which was proposed by Giovanni Indiveri at RoboCamp, 2002 [Indiveri, 2002]. The idea of the control design is to make the arc in front of the goal to an equilibrium point for the goalkeeper and achieve asymptotic stability.

4.2.1 Design

Consider the kinematic model from equation 4.1. If the position is given in polar-coordinates an equivalent kinematic model would be:

$$\begin{bmatrix} \dot{r} \\ \dot{\alpha} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} u\ \cos(\alpha) \\ \omega\ -\ \frac{u}{r}\ sin(\alpha) \\ \frac{u}{r}\ sin(\alpha) \end{bmatrix} \tag{4.2}$$

Where the angle α is given as shown on figure 4.2. From figure 4.2 the goalkeeper is on the arc when $r = d$. Furthermore the angle α should be 90° when the goalkeeper is moving on the arc in order to keep the front towards the ball. These two requirement are expressed in the following error vector:

$$e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} d - r \\ \frac{\pi}{2} - \alpha \end{bmatrix} \tag{4.3}$$

The next step is to introduce an Lyapunov function candidate. Since asymptotic stability is a requirement, the following equations must be satisfied [Khalil, 2002]:

$$V(0) \ = \ 0 \quad and \quad V(x) > 0\ in\ Q - \{0\} \tag{4.4}$$

$$\dot{V}(x) \ < \ 0\ in\ Q - \{0\} \tag{4.5}$$

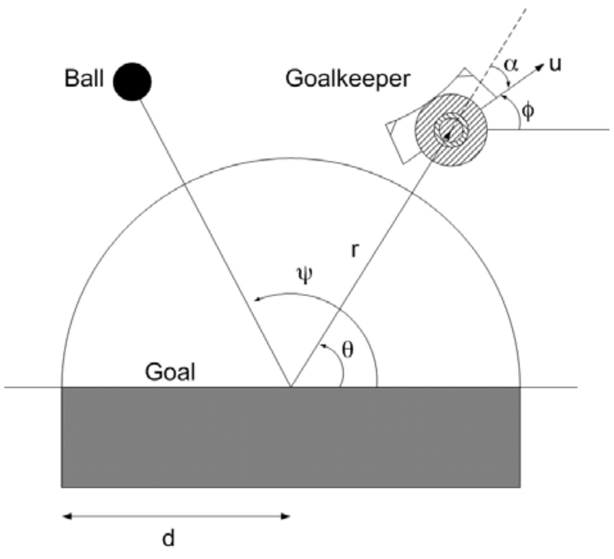


Figure 4.2: The posture of the goalkeeper given in polar coordinates.

Since there are no systematic method for finding an appropriate Lyapunov function [Khalil, 2002], it is chosen to use the same as given in [Indiveri, 2002].

$$V(e) \;=\; \frac{1}{2}\left(h\,e_1^2 \;+\; e_2^2\right) \quad , h > 0 \tag{4.6}$$

$$V(e) \;=\; \frac{1}{2}\left(h\left(d - r\right)^2 \;+\; \left(\frac{\pi}{2} - \alpha\right)^2\right) \quad , h > 0 \tag{4.7}$$

With h as a positive constant, it is clear that equation 4.7 is positive semidefinite. Furthermore it can be seen that $V(0) = 0$ and $V(e) > 0 \; in \; Q - \{0\}$ which is concordant with equation 4.4. The derivative of $V(e)$ is examined to establish, if asymptotic stability is feasible. The derivative $\dot{V}(e)$ is given by:

$$\dot{V}(e) = h\left(d - r\right)\,\dot{r} \;+\; \left(\frac{\pi}{2} - \alpha\right)\,\dot{\alpha} \quad , h > 0 \tag{4.8}$$

Insertion of equation 4.2 in equation 4.8 yields:

$$\dot{V}(e) = h\left(d - r\right)\,u\,\cos(\alpha) \;+\; \left(\frac{\pi}{2} - \alpha\right)\left(\omega \;-\; \frac{u}{r}\,\sin(\alpha)\right) \quad , h > 0 \tag{4.9}$$

To achieve asymptotic stability a control law for u and ω must be derived in such a way that equation 4.9 becomes negative definite.

The approach for the choice of ω is to cancel some of the undesirable terms which makes it difficult to determine the nature of $\dot{V}(e)$. Consider the the following control law for ω:

$$\omega = \frac{u}{r}\,\sin(\alpha) \;-\; \gamma\left(\frac{\pi}{2} - \alpha\right) \;-\; h\,u\left(d - r\right)\frac{\cos(\alpha)}{\frac{\pi}{2} - \alpha} \quad , \; h > 0\,, \gamma > 0 \tag{4.10}$$

Insertion in equation 4.9 yields:

$$\dot{V}(e) = -\,\gamma\left(\frac{\pi}{2} - \alpha\right)^2 \quad , \gamma > 0 \tag{4.11}$$

From equation 4.11 it is clear that $\dot{V}(e)$ becomes negative semidefinite. Combining this with $V(e)$ being positive semidefinit, $V(e)$ tends to a finite positive limit. That $V(e)$ in fact tends to zero and hence guarantees asymptotic stability is proven in the following.

Proof

It is known that $\alpha \to \frac{\pi}{2}$ because $V(e)$ tends to a finite limit and that $\dot{\alpha}$ is a uniformly continuous function. From Barbalats lemma it is known that $\dot{\alpha} \to 0$, [Khalil, 2002]. Inserting this in equation 4.2 yields:

$$\dot{\alpha} = \omega - \frac{u}{r}\sin(\alpha) \tag{4.12}$$

$$\Downarrow$$

$$0 = \omega - \frac{u}{r}\sin(\frac{\pi}{2}) \tag{4.13}$$

$$\Downarrow$$

$$\omega \to \frac{u}{r} \tag{4.14}$$

From equation 4.10 ω is given as:

$$\omega = \frac{u}{r}\sin(\alpha) - \gamma\left(\frac{\pi}{2} - \alpha\right) - h\,u\left(d - r\right)\frac{\cos(\alpha)}{\frac{\pi}{2} - \alpha} \quad , \ h > 0\ , \gamma > 0 \tag{4.15}$$

Since $\alpha \to \frac{\pi}{2}$

$$\lim_{\alpha \to \frac{\pi}{2}} \omega(t) = \frac{u}{r} + h\,u(r - d) \tag{4.16}$$

With $\omega \to \frac{u}{r}$ it is known that $hu(r - d) \to 0$ as $t \to 0$. The following assumption is made about u:

$$\lim_{t \to \infty} u(t) \neq 0 \tag{4.17}$$

Then

$$\lim_{t \to \infty} r(t) = d \tag{4.18}$$

Thus asymptotic stability is achieved since $V(e) \to 0$ as $t \to \infty$.

With the control law from equation 4.10 for ω, the goalkeeper will independently of a control law for u tend to move towards a circle with center in the middle of the goal. Only the constraint $u \neq 0$ is present to ensure asymptotic stability. The desirable motion for the goalkeeper is to move on the arc and stop when $\psi = \theta$ thereby minimizing the exposed goal area. From the designed state machine for the behaviours the goalkeeper is close to or on the circle in the behaviour followball. Therefore it is assumed that a control law for u which drives $\theta \to \psi$ and then stops will be sufficient. A P-controller is chosen for u:

$$u = K_p * (\psi - \theta) \tag{4.19}$$

To summarize the control design: A nonlinear control law for the angular velocity has been derived. The control law guarantees asymptotic stability towards a circle with radius

d if $u \neq 0$. Furthermore the angle α tends to $\frac{\pi}{2}$. The linear velocity is controlled with a p-controller. The two control laws are:

$$\omega = \frac{u}{r}\sin(\alpha) - \gamma\left(\frac{\pi}{2}-\alpha\right) - h\,u\left(d-r\right)\frac{\cos(\alpha)}{\frac{\pi}{2}-\alpha} \quad , \; h>0\;,\gamma>0 \qquad (4.20)$$

$$u = K_p*(\psi-\theta) \qquad (4.21)$$

4.2.2 Simulation

The nonlinear control design is tested in a MATLAB simulation. So far the control design has been made in continuous time without regard to the sample frequency. Because of the implemented structure with the micro-agents the sample frequency varies. Through tests where a variable is incremented for each run of the control mirco-agent it was found that the sample frequency was approximately 10 Hz. With the simulation the control design was tested for stability at this frequency.

The proportion between the constants h and γ from equation 4.20 decides how the proportion between the two errors e_1 and e_2 should converge to zero. If γ is large compared to h, e_2 will converge faster than e_1 and the goalkeeper will have to move back and forth several times before reaching the radius of the arc. This situation is shown in figure 4.3, where the goalkeeper moves from the center of the goal to the arc. Is h large

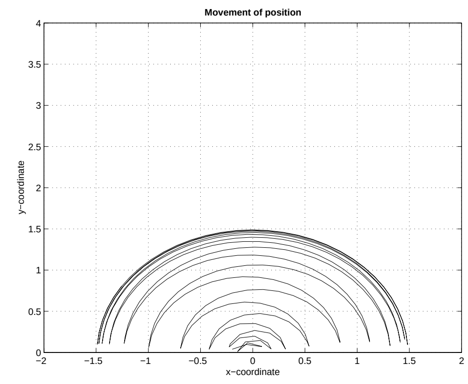


Figure 4.3: Movement from the center of the goal (0,0) to the arc for a goalkeeper with a large γ value compared to h.

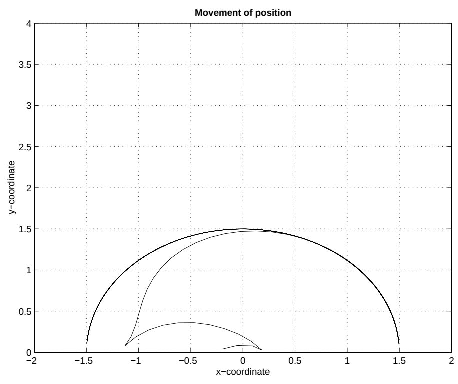


Figure 4.4: Same situation as in figure 4.3 but with a large h value compared to γ.

compared to γ the goalkeeper will move almost directly to the arc and then start to turn as shown on figure 4.4. The purpose of the simulations shown on figure 4.3 and 4.4 is only to show the influence of h and γ and dosn't illustrate the true motion for the control design. The simulations doesn't show the true motion because the used linear velocity u is a constant that changes sign when θ reaches 0 or π. Appropriate values are through trial and error found to be $h = 4$ and $\gamma = 2$.

The rules of the SocRob game prohibit the goalkeeper to have an initial start position in front of the goal [Robocup.org, 2002].The starting position of the goalkeeper is therefore next to the goal on the left side as shown in figure 4.5. From the start position the

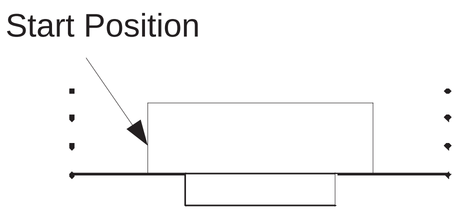


Figure 4.5: The initial starting position for the goalkeeper in a SocRob game.

goalkeeper should converge to the arc and stop in front of the goal as shown in figure 4.6. The lower right plot of figure 4.6 shows that it takes approximately 4 seconds for the

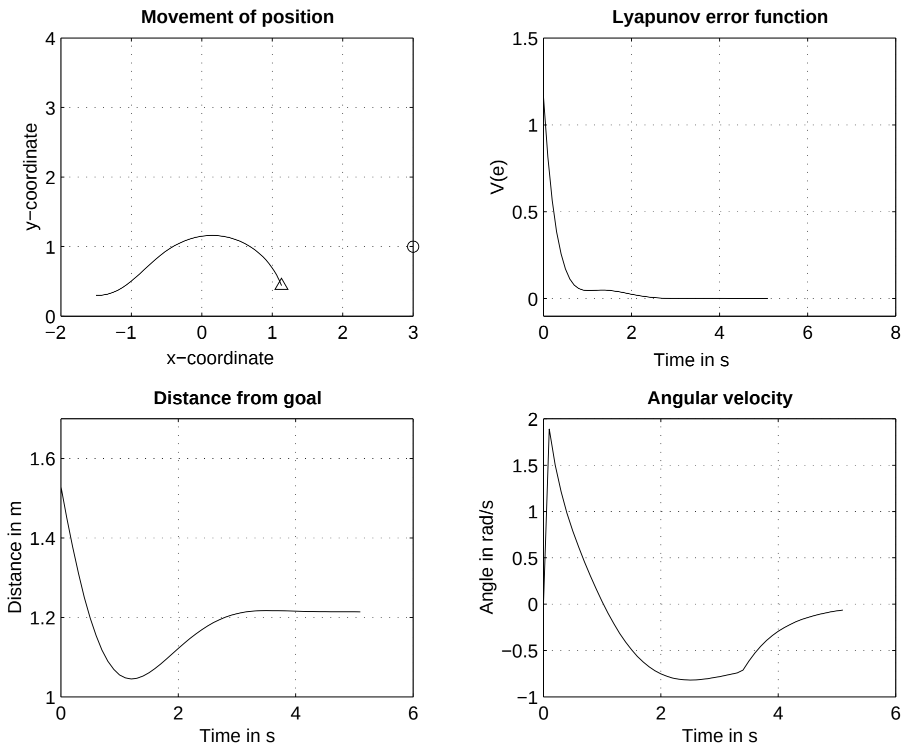


Figure 4.6: Results from a simulation where the goalkeeper moves from its start position on to the arc and stops in front of the goal. In the top left picture the circle is the ball and the triangle is the final position of the goalkeeper. The parameters for the simulation is: $d = 1.2$, $h = 4$, $\gamma = 2$, $K_p = 1.5$, $u_{max} = 0.8\frac{m}{s}$

goalkeeper to reach its final posture. This is seen from the angular velocity which starts to converge to zero as the linear velocity decreases. In the last second the goalkeeper comes to a halt. Whether four seconds is a reasonably time for the movement will be evaluated during tests of the goalkeeper. From figure 4.6 it is clear that the goalkeeper will follow the arc. Figure 4.7 shows the results from a simulation where the goalkeepers path converges towards the circle from a starting position further away.

With the results from the simulations it is concluded that the control design is working as intended. Furthermore it is concluded that the control design is stable at a sample frequency of 10 Hz. The conclusion of stability is however entirely based on kinematic and can therefore only serve as an indication for stability after implementation in the

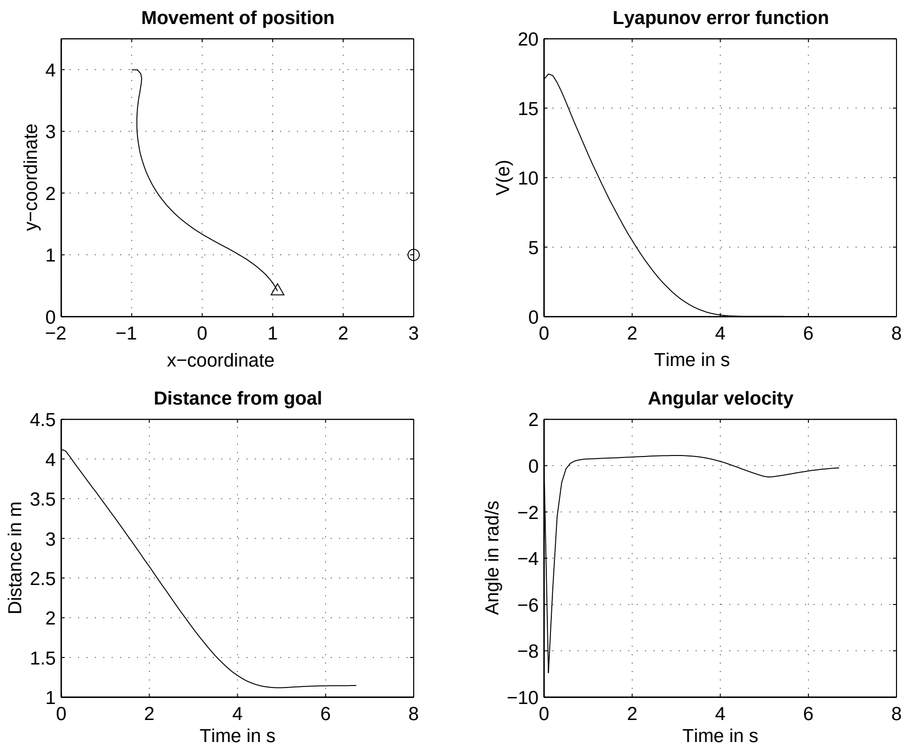


Figure 4.7: From a starting position in (-1,4) the goalkeeper moves to the arc and stop in front of the ball.

goalkeeper.

4.2.3 Implementation

Each behaviour is implemented as a plugin to the control micro-agent. For each run of the control micro-agent the function `method()` in the current plugin will be executed. The control algorithm is therefore implemented in the function `method()`. In the given framework each run of `method()` can be thought of as a sample. Here the sample frequency is not limited by the speed which measurements can be retrieved, but by the time used by other micro-agents. Which behaviour that is to be executed in the `control` micro-agent is decided in the `machine` micro-agent where the state machine of the behaviour is implemented, see section 3.3.1. In the following some problems and functions encountered during a run of `method()` is described. The code can be found at [CD, \source\control\followball.c].

Change of Coordinates

Inputs to the control algorithm is the posture of the goalkeeper and the position of the ball. The position of the ball is needed to determine the angle ψ from equation 4.21. The posture of the goalkeeper can be read from the blackboard. The posture from the blackboard is automatically updated with measurements from odometry. Problems with odometry that becomes inaccurate as time goes by is posed in the following chapter and for now it is therefore assumed that the position from odometry is precise.

On the blackboard the posture of a robot is described in the world coordinate system, which has origin in the center of the field and a positive x-axis in the direction of the opponents goal. In the design of the control law this coordinate system wasn't suitable and another was used. A change of coordinate systems was therefore necessary. The coordinate system used in the design of the control laws has origin in the middle of the goalkeepers goal and a positive y-axis towards the center of the field, see figure 4.8. This coordinate system will be referenced to as the goal coordinate system. Furthermore the angle ϕ which describes the orientation of the goalkeeper in world coordinates is zero when the kicker for the goalkeeper is pointing in the positive direction of the y-axis and is measured in degrees between 0 and 360 counter clock wise. In the goal coordinate system ϕ is given between $\pm\pi$. The change of coordinates is implemented as a function

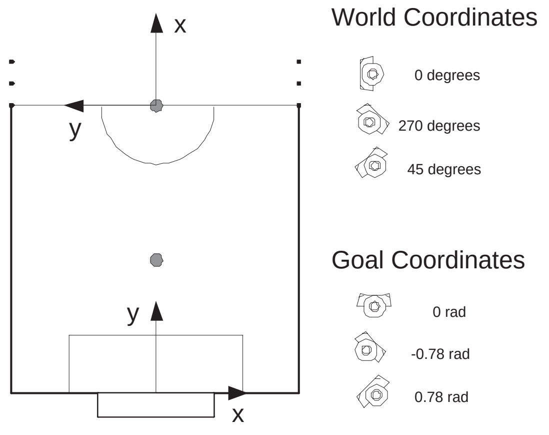


Figure 4.8: The two used coordinate systems. The world coordinate system must be used when a posture of a robot is placed on the blackboard. The goal coordinate system is introduced through the control design.

called `changeCoordinateSystem()` which changes both posture of the goalkeeper and position of the ball from world coordinates to goal coordinates. Since the control algorithm needs polar coordinates as input a function `cartesianToPolar()` changes the goalkeepers posture from cartesian to polar coordinates. Unless otherwise is stated, coordinates described in the following will be in goal coordinates.

Ball Movement

As mentioned one input to the control algorithm is the position of the ball. The position is determined in the `vision` micro-agent and put on the blackboard. Even if the ball lies still the position of the ball will vary. The variations depends on consistency of the light throughout the field and the quality of the camera calibration. When the `vision` micro-agent determines a position of the ball it does so from determining the center of mass for the largest orange contour. This causes a problem if the goalkeeper is unable to see the ball which can easily happen since the goalkeeper only has vision of half the field. If no ball is present, there is a good chance that the `vision` micro-agent will spot something different which is orange, and believe this is the ball. With wrong measurements of the ball position the goalkeeper moves around in unwanted ways instead of standing still as intended with the primitive task `FIND_BALL`. Tests show that the orange spots which the `vision` micro-agent mistakes for the ball isn't found for every sample and only rarely found twice in the same position. The problem is solved by implementation of the function `checkBallMovement`. `checkBallMovement` takes advantage of the fact that two subsequent measurements of the orange spots rarely is in the same position. If the ball is out of sight and the `vision` micro-agent is unable to find it, then FollowBall will be in the primitive task `FIND_BALL`. In `FIND_BALL` the function `checkBallMovement` will examine two subsequent measurements of the ball position and compare them to a threshold around the first position. If the second position is within the threshold, the position of the ball is accepted, and it is concluded that the ball has been found. If not the first measurement is discarded and a third is retrieved which is compared to the second and so forth.

There are however some drawbacks to the solution with `checkBallMovement`:

- Imagine the ball is on the opposite site of the field and FollowBall is in the primitive task `FIND_BALL`. From the opposite site of the field a shot towards the defending goal is made. Once the ball enters the defending half of the field, the goalkeeper will see the ball. But if the ball has a large velocity no two measurement will be within the threshold and the goalkeeper will remain in `FIND_BALL` instead of repositioning itself or changing behaviour to InterceptBall.

- If the ball moves from in sight to out of sight the `vision` micro-agent might still find a wrong position which is accepted. This because `checkBallMovement` only is used once the `vision` micro-agent declares that it can't find the ball. Or in other words the problem of securing a rightfully transition to the primitive task `FIND_BALL` still hasn't been solved.

Variation of Radius

In order to minimize the exposed goal area the radius of the defending arc should increase as the ball moved further away from the goal, see section 3.2. To implement this feature a function `newRadius()` is used which calculates a new radius for the defending arc based on the ball position for each run of `method()`. For this function a new part of the field is defined as the rZone, see figure 4.9. In the rZone the radius of the defending arc will change with the change of the ball position. When the ball is in the danger zone the radius of the definding arc is in minimum and if the ball is beyond the rZone the radius is in its maximum.

Control Algorithm

The control algorithm is implemented in a function called `followArcControl`. Before a new linear and angular velocity is calculated, the angle between the center of the goal

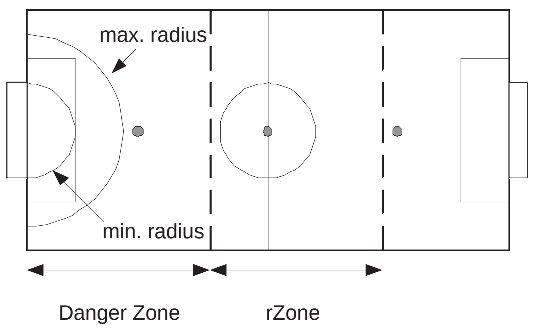


Figure 4.9: The field divided into the danger zone and the rZone.

and the ball ψ used in equation 4.21 is examined, to ensure that the goalkeeper won't move behind the goal. Once ψ has been examined, a new linear velocity u is calculated. The new velocity is tested to see if it exceeds a maximum limit. If it does, it is set to its maximum value. Based on this linear velocity the angular velocity is calculated.

The output from the control algorithm is the linear and the angular velocity. What the goalkeeper needs however is the angular velocity for each wheel. In appendix A it is shown how to calculate the angular velocity for each wheel based on the wanted linear and angular velocity of the goalkeeper. In the given framework a function `motorsSetVW()` is available which is used to make the calculation.

Flow of Method()

Once the `machine` micro-agent has changed to a given behaviour, the control agent will execute the function `method()` for each run of the behaviour. In `method()` the current state of the behaviour is fetched. The state of FollowBall is one of the three primitive tasks: `FOLLOW_ARC`, `GO_CENTER` and `FIND_BALL`, see section 3.3.2. The only implementational difference between the tasks `FOLLOW_ARC` and `GO_CENTER` is that a virtual ball is placed in middle of the field in `GO_CENTER`. Doing this makes the goalkeeper move to the middle of the arc in order to cover the goal from the virtual ball. `GO_CENTER` is the initial state of followball when a SocRob game is started. The reason for this, is the vision of the goalkeeper, which approximately gives sight of half of the field. When a game is begun the goalkeeper is unable to see a ball in the middle of the field from its initial starting position. It is therefore desirable to move the robot to the center of the arc, so it covers the goal and has better chance of seeing the ball. In figure 4.10 a flow chart of `method()` is given.

4.2.4 Test

To examine if the goalkeeper was behaving as intended, identical test runs to those shown simulated in section 4.2.2 was logged. In the fist test the goalkeeper moves from its initial start position to the defending arc, and stops when it reaches the line between the ball and the center of the goal. This run is similar to the one shown on figure 4.6. The purpose of

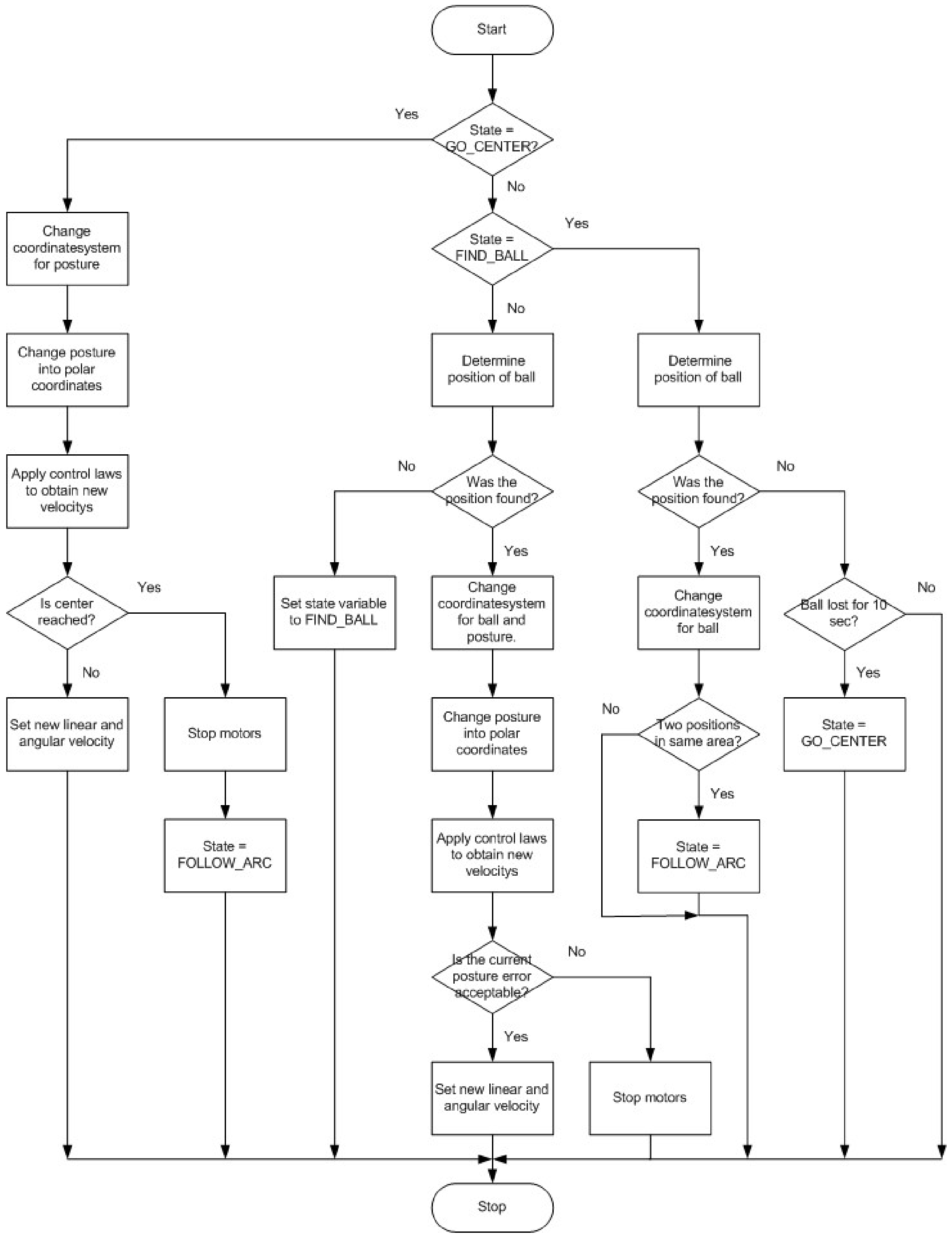

Start
State = GO_CENTER?
Yes
No
State = FIND_BALL
Yes
No
Change coordinatesystem for posture
Change posture into polar coordinates
Apply control laws to obtain new velocitys
Is center reached?
Yes
No
Set new linear and angular velocity
Stop motors
State = FOLLOW_ARC
Determine position of ball
Was the position found?
No
Yes
Set state variable to FIND_BALL
Change coordinatesystem for ball and posture.
Change posture into polar coordinates
Apply control laws to obtain new velocitys
Is the current posture error acceptable?
No
Yes
Set new linear and angular velocity
Stop motors
Determine position of ball
Was the position found?
No
Yes
Change coordinatesystem for ball
Two positions in same area?
No
Yes
State = FOLLOW_ARC
Ball lost for 10 sec?
No
Yes
State = GO_CENTER
Stop

Figure 4.10: Flowchart that shows a run of the function `method()` in the control plugin followball.c.

the test is to ensure proper movement of the goalkeeper. Since the movement is the focus of the test, a virtual ball is used instead of a real one. Otherwise there may be some errors in the measurements of the ball position causing the goalkeeper to move different from the simulation in figure 4.6. As seen from the top left plot in figure 4.11 the goalkeeper moves to the arc but has troubles staying on it. This is because of the dynamics which wasn't included in the simulation. Because of the dynamics the goalkeeper is limited in

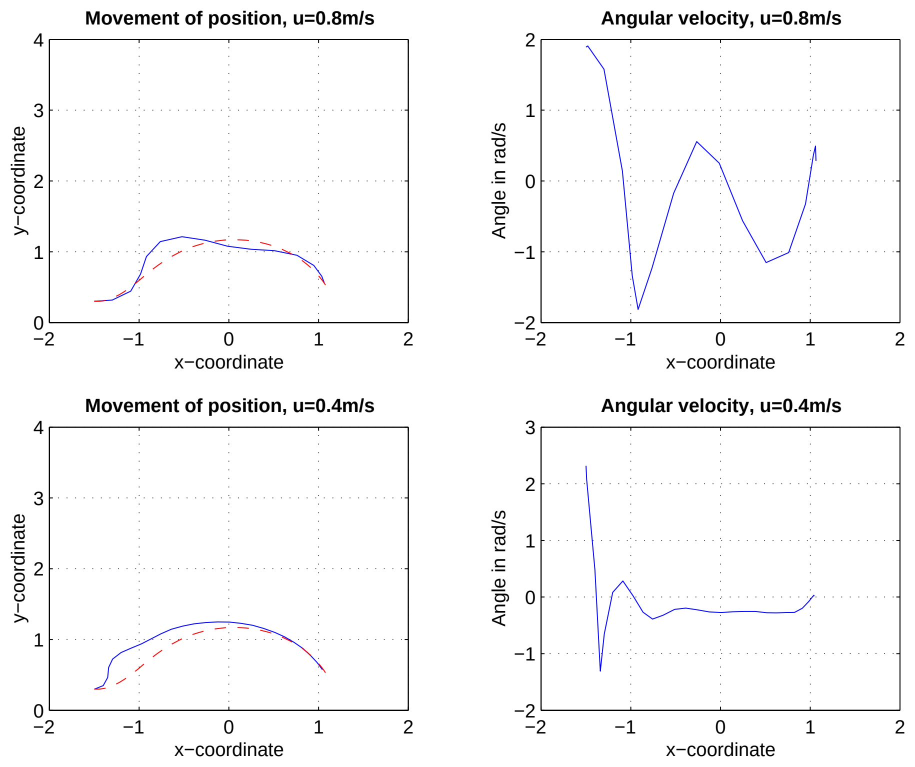


Figure 4.11: The goalkeeper moves from its starting position to the arc and stops as it reaches the line between the ball and the center of the goal. The solid line shows the movement in goal coordinates logged from odometry and the dashed line is the movement from the simulation. The radius for the defending arc is 1.2 m. The angular velocity is the one used as input to the control algorithm and not the actual angular velocity of the goalkeeper.

the acceleration in the angular velocity. From start of the test, the linear velocity will go to its maximum value because the difference in the angles θ and ψ in equation 4.21. The calculation of the angular velocity is based on the linear velocity and takes a high positive value, as seen in the top right plot of figure 4.11. Because of the dynamics it takes time to reverse this angular velocity causing the goalkeeper to make an overshoot in the trajectory between -1 and -0.25 in the movement on the x-axis. The problem disappears as the linear velocity decreases, which happens as the goalkeeper reaches $x = 0.5$. In other words the dynamics causes a trade off between a high velocity and good tracking. If the maximum linear velocity is reduced the goalkeeper will move in a path more similar to the arc as shown in the left bottom plot in figure 4.11. The reason why the goalkeeper doesn't find the arc from the start with $u_{max} = 0.4\frac{m}{s}$ is because the constants h and γ was tuned to $u_{max} = 0.8\frac{m}{s}$ in the test. Though the movement in the top left plot in figure 4.11 doesn't seem pretty it is acceptable. It is important to remember that the simulation is a worst case scenario, where the starting position of the goalkeeper is utterly wrong compared to the position of the ball. Nevertheless should such a situation occur during a SocRob game, a high velocity is preferable compared to a good trajectory tracking.

The second test examines the goalkeepers ability to move from a position away from the arc, to the arc. According to the simulation shown in figure 4.7 the goalkeeper should be able to converge to the arc from a starting position in $(-1,\ 4)$ in goal coordinates. This wasn't feasible in test with the goalkeeper. Several test was made where the distance between the starting position and the arc gradually was increased. Some results are shown in figure 4.12. From figure 4.12 it is clear that FollowBall isn't a suitable behaviour

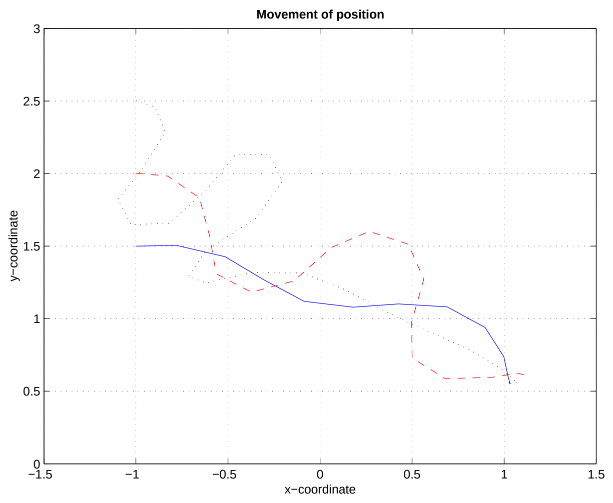


Figure 4.12: The goalkeeper moves from different start positions to the arc and stops on the line between the ball in $(3,\ 1)$ and the center of the goal in $(0,\ 0)$. The test was made with a maximum linear velocity $u_{max} = 0.8\frac{m}{s}$.

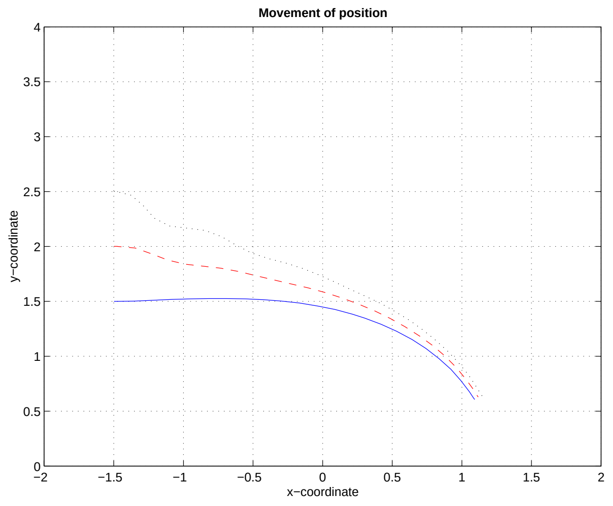


Figure 4.13: Same test as in figure 4.12 but with a maximum linear velocity $u_{max} = 0.4\frac{m}{s}$.

to use, if the goalkeeper should be brought back to the arc from a position further away than approximately 0.75 m. from the arc. The goalkeeper moves with an acceptable trajectory to the arc with a starting position in $(1,\ 1.5)$. If the y-coordinate is larger, the goalkeeper comes very close to, or unstable as seen in figure 4.12 where the goalkeeper starts in $(1,\ 2)$ and $(1,\ 2.5)$. The problems experienced here is once again caused by the dynamics. Right from the start, the goalkeeper starts turning, so it moves towards the arc. Again the linear velocity is at maximum, and the angular velocity takes a value of such a size that the goalkeeper is unable to reverse it in the given time. This results in the s-alike movements towards the arc. With a reduced linear velocity the goalkeepers ability to reach the arc increases as shown i figure 4.13. From this plot it can also be seen that the trajectory gets worse as the starting position gets further away. This indicates that the problem isn't entirely solved by decreasing the linear velocity. Since the ability to converge to the arc from a starting position isn't intended for the behaviour FollowBall, further effort to solve the problem wasn't done.

In the test above the feature of changing the radius according to the position of the ball in the rZone wasn't used. Tests were successfully made where the goalkeeper changed its radius. It was however concluded that the change of radius didn't work as well as intended. One reason for this is that the goalkeeper is unable to move directly to an arc with a new radius. The position of the ball has to move from one side of the field to another before the change of radius takes place. Furthermore the chance of the undesirable situation, where the ball is placed between the goalkeeper and the goal is increased as the goalkeeper moves further away from the goal. Since none of the tests indicated an increase in performance, it was chosen not to use this feature.

Several tests has been made to examine the goalkeepers ability to follow the ball as it moves around the field. The results showed that the goalkeeper was able to follow the ball, moved on the arc as desired and stopped when the exposed goal area had been minimized. During these test it was observed that even though the goalkeeper acted as intended, it was unable to save a precise shot even if it passed very close to the goalkeeper. There are two reasons for this. One is that once the goalkeeper has moved into position and minimized the exposed goal area, a precise shot only makes a small change between the angles θ and ψ causing the goalkeeper to move with a slow linear speed. The second is the acceleration of the goalkeeper is limited to a max value of 1 $\frac{m}{s}$. The limit of the acceleration and the small linear velocity with shots towards the goal makes the goalkeeper unsuitable for interception of shots. It should be mentioned that tests where the goalkeeper played against another robot showed that the attacking robot couldn't score any goals because it was unable to make a precise shot.

Summarizing the results of the tests, it can be concluded that the behaviour FollowBall has been implemented and works as intended except for the feature of changing radius. Based on the results it was decided not to use that feature. The goalkeeper may experience problems returning to the defending arc, if it is placed more than approximately 0.75 m. from the arc. Furthermore it can be concluded that the behaviour FollowBall is unsuitable for interception of shots.

4.3 InterceptBall

If the ball moves inside the danger zone the goalkeeper switches behaviour to InterceptBall. In InterceptBall the goalkeeper should defend the goal on a line in front of the goal. It is most likely that the posture of the goalkeeper is on the arc, when the `control` micro-agent switches to InterceptBall. The control algorithm should therefore not only be able to move the goalkeeper back and forth on a line, but also make it able to move the goalkeeper to the line from any position on the arc. This problem resemblances the tracking problem in FollowBall. It is therefore chosen to rederive the control algorithm used to control the angular velocity in FollowBall so a line in front of the goal becomes the equilibrium point instead of the arc. Since the reasoning for the design already has been given in section 4.2.1 the design in the following section will not be as detailed as previously.

4.3.1 Design

Consider again the kinematic model from equation 4.22:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} u\ \cos(\phi) \\ u\ \sin(\phi) \\ \omega \end{bmatrix} \tag{4.22}$$

Where x, y and ϕ are given as shown on figure 4.14. In InterceptBall the goalkeeper

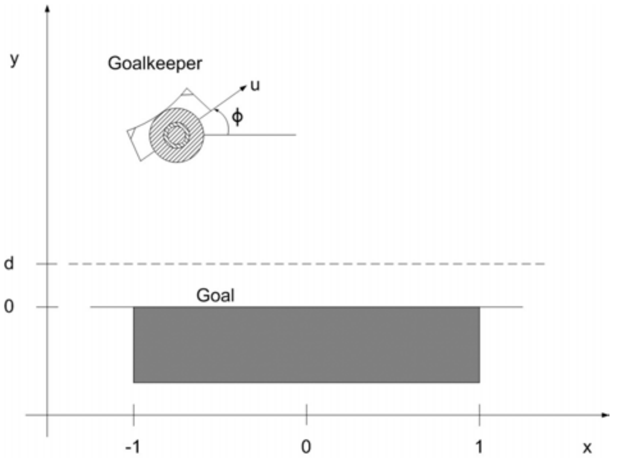


Figure 4.14: The value of d sets the y-coordinate for the line the goalkeeper should move on in InterceptBall.

moves on a line in front of the goal. When the goalkeeper moves on the line, the two requirements $y = d$ and $\phi = 0$ are satisfied. The requirements are expressed in the following error vector:

$$e = \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} = \begin{bmatrix} d - y \\ \phi \end{bmatrix} \tag{4.23}$$

A Lyapunov candidate function is introduced:

$$V(e) = \frac{1}{2}\left(h\ e_1^2 + e_2^2\right) \quad , h > 0 \tag{4.24}$$

$$V(e) = \frac{1}{2}\left(h\left(d-y\right)^2 + \phi^2\right) \quad , h > 0 \tag{4.25}$$

In order to establish, if asymptotic stability is feasible the derivative of $V(e)$ is found:

$$\dot{V}(e) = h\left(d-y\right)\dot{y} + \phi\,\dot{\phi} \quad , h > 0 \tag{4.26}$$

Inserting equation 4.22 in 4.26:

$$\dot{V}(e) = h \left(d - y\right) u \ \sin(\phi) \ + \ \phi\omega \quad , h > 0 \tag{4.27}$$

Consider the control law:

$$\omega = -\gamma \ \phi \ - \ h \left(d - y\right) u \ \frac{\sin(\phi)}{\phi} \quad , \ h > 0 \ , \gamma > 0 \tag{4.28}$$

Applying the control law yields:

$$\dot{V}(e) = - \ \gamma \ \phi^2 \quad , \gamma > 0 \tag{4.29}$$

Which is a negative definite function. Since the control law is derived on the same reasoning, as the one in the control algorithm from FollowBall, a proof for asymptotic stability will not be given here.

A control law for the linear velocity u can now be designed independent of the control law for ω. Based on the good results from FollowBall, it was chosen to use a P-controller for the linear velocity. The goalkeeper should place itself on the line according to one of two situations. One is similar to FollowBall, where the ball is inside the danger zone but not moving fast towards the goal. In this situation the goalkeeper should minimize the exposed goal area by placing himself on the line between the ball and the middle of the goal. The other situation is when a ball is moving fast towards the goal. Here the goalkeeper should predict the interception point between the trajectory of the ball and the defending line in front of the goal. Under the assumption that the velocity components are accessible for the ball, the following equations are used to predict the x-coordinate for the interception point. First the predicted time for the ball to reach the defending line is calculated:

$$T = \frac{(y_{ball} \ - \ d)}{v_y} \tag{4.30}$$

Where y_{ball} is the y-coordinate for the ball position. d is the y-coordinate for the defending line and v_y is velocity component in the y-direction. The predicted x-coordinate is then:

$$x_{intercept} = (x_{ball} \ + \ v_x \, T) \tag{4.31}$$

With the ability to predict the interception point of the ball, the goalkeeper has more time to move to a given posture and hence the problem experienced in FollowBall where goalkeeper was to slow to intercept a shot should be avoided. The control law for the linear velocity u is given as:

$$u = K_p * (\psi - \theta) \tag{4.32}$$

Where ψ is either the angle to the ball or the angle to the predicted interception point $x_{intercept}$. θ is as in FollowBall the angle between the center of the goal and the goalkeeper.

The results of the design for InterceptBall are a control law for the angular velocity ω and the linear velocity u given by equation 4.33 and 4.34

$$\omega = -\gamma\,\phi - h\left(d-y\right)u\,\frac{\sin(\phi)}{\phi} \quad , \; h>0\,,\gamma>0 \tag{4.33}$$
$$u = K_p * (\psi - \theta) \tag{4.34}$$

Depending on the direction and velocity of the ball, an interception point between the trajectory of the ball and the defending line may be used to determine the angle ψ.

4.3.2 Simulation

A simulation is made in MATLAB to verify that the rederived control design is working as intended. The interesting thing to see is the ability for the goalkeeper to converge to the defending line. This is interesting because when a transition from FollowBall to InterceptBall occurs, the goalkeepers path to the line has influence on its ability to intercept a shot from outside the danger zone.

Different situations are simulated where the goalkeeper starts from positions away from the defending line. It was found that an appropriate proportion between the values h and γ in equation 4.33 is given with $h = 5$ and $\gamma = 2$. With these values the goalkeeper converges towards the defending line in a path resembling a straight line. This is desired since the goalkeeper must reach the interception point as quick as possible. In figure 4.15 four of the simulations are shown. From the bottom right plot in figure 4.15 it is seen that

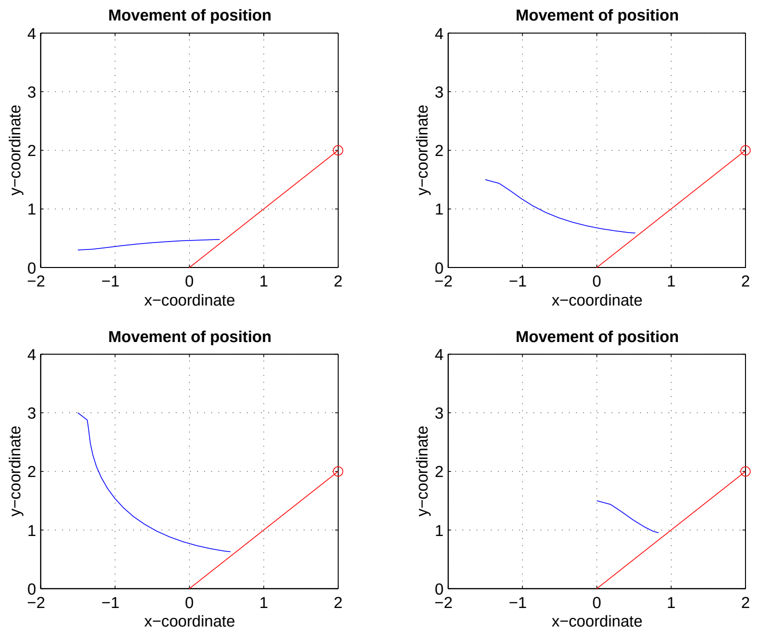


Figure 4.15: Four situations where the goakeeper has different start positions. The circle in (2, 1) symbolizes the ball. The line from the ball to the origin illustrates a shot towards the goal. The defending line is placed in $d = 0.5$. Starting position for the goalkeeper in the figures from the left to right is (-1.5, 0.3), (-1.5, 1.5), (-1.5, 3) and (0, 1.5).

the goalkeeper isn't always able to reach the defending line in $d = 0.5$ before it stops to intercept. This isn't considered to be a problem since the primary objective is to intercept the ball and not necessarily reaching the line. If the goalkeeper hasn't reached the line before intercepting it can move the remaining part of the way to the line as it subsequent seeks to minimize the exposed goal area.

4.3.3 Implementation

Because of the resemblance between FollowBall and InterceptBall large parts of code from FollowBall was reused in the implementation of Interceptball. The primitive task `FIND_BALL` was implemented in the exact same way with the use of the function `checkBallMovement`. The primitive task `INTERCEPT` was implemented in a simular way to the primitive task `FOLLOW_ARC` in FollowBall. The difference in implementation is the feature of predicting the trajectory of the ball in `INTERCEPT`. The code can be found at [CD, \source\control\interceptball.c].

Prediction

Determination of the ball movement and calculation of ψ is implemented in a function called `Predict()`. From the blackboard it is possible to read the velocity components of the ball. The velocity is placed on the blackboard by the `vision` micro-agent which needs at least three measurements to make the calculations. The prediction should only be made when a ball is approaching fast, which is determined from the y-component of the velocity. If the y-component is less then the value $-0.8\frac{m}{s}$ in world coordinates, the movement of the ball is considerd to be a shot, and an interception point is predicted with equation 4.31. The value of $-0.8\frac{m}{s}$ was found to be appropriate through trial and error. If the ball isn't approaching fast, the new position of the ball is compared with the former position to see if the ball has moved. This is implemented to remove fluctuations in the angle ψ which causes the goalkeeper to continuously correct its own position, though the ball may lie still. Correction shouldn't be a problem but it is experienced that these corrections can cause the goalkeeper to lose odometry. Since no two measurements of the ball position never is the same, the function `checkBallMovement()` is used to decide if a ball is moving or lies still. If the ball lies still, the old value of ψ is used in the control algorithm. If the ball neither approaches fast or lies still a new value of ψ is calculated and used in the control algorithm to minimize the exposed goal area.

Block

The thought of the primitive task `BLOCK` was that when the ball is close to the goalkeeper, the front camera should be used due to superior performance over the top camera in this situation. As it turned out, there were some problems with the front camera. When used, the `vision` micro-agent doesn't calculate the velocity of the ball. Therefore it becomes obvious that the front camera only should be used, when the ball is so close to the goalkeeper that a prediction wouldn't be helpful. A prediction isn't helpful if the ball is so close that the ball would have passed the goalkeeper in the three samples that is needed to make a prediction. The `vision` micro-agent doesn't calculate the position of the ball on the field either. It returns the distance from the goalkeeper to the ball, and the angle between the front of the goalkeeper and the ball. Based on these two measurement the position of the ball can be calculated relative to the posture of the goalkeeper. The problem with this is that the angle returned by the `vision` micro-agent has a bug when the ball is close to the goalkeeper causing it to vary with a factor of ± 1. This bug makes the measurements from the front camera useless as input to the control algorithm. Because of the bug, the goalkeeper doesn't correct its position in `BLOCK`. The goalkeeper simply stands still as long as the ball is within vision of the front camera and within a distance of $0.7m$. It could be argued that the front camera shouldn't be used at all, until the bug has been fixed. There are however also problems concerning the use of the top camera when the ball is close to the goalkeeper. Because of the hardware implementation of the kicker, the top camera is unable to detect the ball if it is closer than $0.7m$ and right in front of the goalkeeper.

In figure 4.16 a flow chart is given for the function `method()` where the three primitive tasks `FIND_BALL`, `INTERCEPT` and `BLOCK` are implemented.

4.3.4 Test

Tests of situations equal to those simulated in section 4.3.2 was made to see if the behaviour worked as intended. In these tests the goalkeeper has different starting positions away from the defending line placed in $d = 0.5$. The results are shown in figure 4.17. From figure 4.17 the same problem experienced in FollowBall is observed. The results does seem better, since the goalkeeper is able to converge to the line from a starting position further away than in FollowBall. The reason for this is the maximum linear velocity, which in InterceptBall is decreased to $u_{max} = 0.6$. This was done to ensure that when a transition from FollowBall would occur, the goalkeeper wouldn't move in a trajectory which has an overshoot in the defending line. An overshoot here might result in a trajectory close to the origin in world coordinates where the goalkeeper could stop because the angle ψ and θ is equal. Once stopped in a position close to the origin the goalkeeper will have to move back and forth several times before the defending line is reached again. Based on the results from 4.17 it was concluded that the goalkeeper moved in acceptable trajectories.

Test where performed to see if the goalkeeper was able to predict a collision point and thereby intercept the ball. It was found that in most cases the goalkeeper did intercept the ball, if the velocity of the ball wasn't too high. A constraint that has great importance is the fact that three measurements is needed before the velocity of the ball is available. During the design in section 4.2.2 some simple tests where made to determine the sample frequency. The sample frequency was found to be 10 Hz. As it turned out the sample

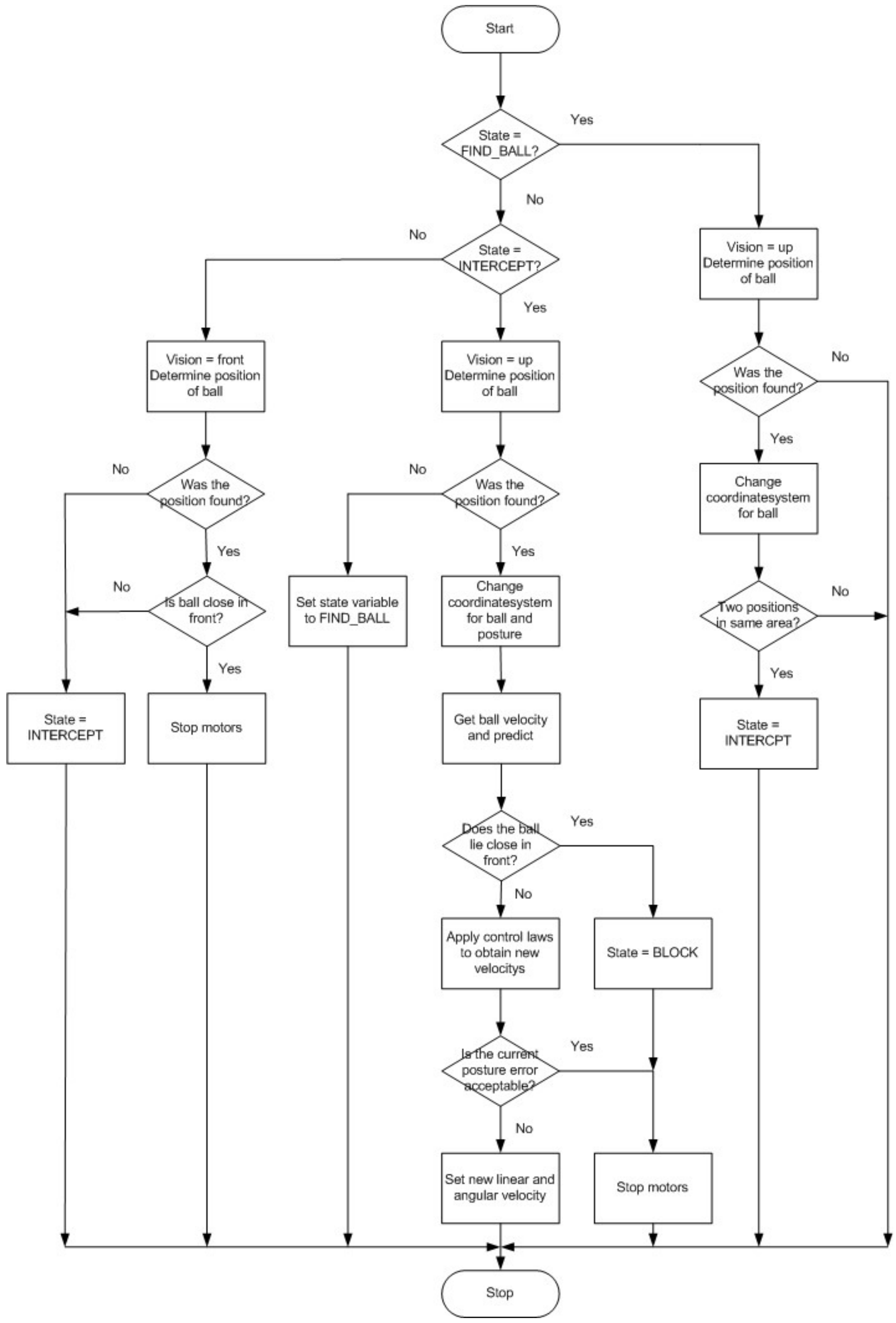

Start
State = FIND_BALL?
Yes
No
State = INTERCEPT?
No
Yes
Vision = up Determine position of ball
Vision = front Determine position of ball
Vision = up Determine position of ball
Was the position found?
No
Yes
Was the position found?
Was the position found?
Is ball close in front?
Set state variable to FIND_BALL
Change coordinatesystem for ball and posture
Change coordinatesystem for ball
Two positions in same area?
State = INTERCEPT
Stop motors
Get ball velocity and predict
State = INTERCPT
Does the ball lie close in front?
Apply control laws to obtain new velocitys
State = BLOCK
Is the current posture error acceptable?
Set new linear and angular velocity
Stop motors
Stop

Figure 4.16: Flowchart that shows a run of the function `method()` in the control plugin interceptball.c.

frequency had changed during implementation to 5 Hz causing the delay from shot to reaction to be 0.8 sec. The change of sample frequency was assumed to be slow functions in `method()`. The test showed that the function `method()` was executed in less than `0.00007` sec. It was therefore concluded that the problem wasn't in the behaviours FollowBall or InterceptBall but somewhere in the micro-agent structure. Because of the

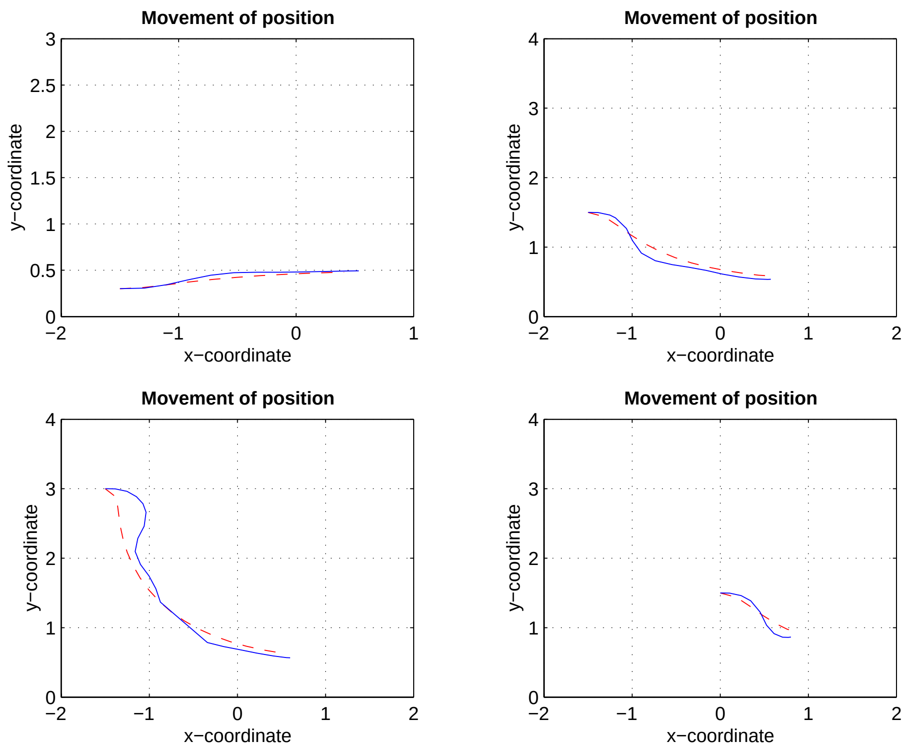


Figure 4.17: Test of situations equal to those simulated. The dashed line is the simulated trajectory and solid is the logged trajectory from odometry.

limited time available for the project, further tests wasn't made. The delay combined with the acceleration is the dominating constraint of performance.

In other tests another problem became obvious. It was observed that every now and then the goalkeeper would predict a wrong collision point causing it to move to the wrong side. The problem is caused by a wrong sign in the measurement of the velocity in the x-component. The fault seems only to be present in the first measurement of the velocity. This indicates that reliability could be increased in the measurements from the `vision` micro-agent, if the minimum limit for prediction was four samples instead of three. It is however not considered to be advisable since this would increase the delay with another 0.2 sec.

Based on the test results it is concluded that the behaviour InterceptBall was implemented and works as designed. The primitive task `BLOCK` has no control algorithm because of a bug in the determination of the angle between the goalkeeper and the ball with the front camera. The goalkeeper is able to intercept as intended, if the velocity of the ball isn't to high. The dominating constraint of performance is a combination of a delay from shot to reaction of 0.8 sec. plus the acceleration.

CHAPTER 5

Self Localization

As stated in the introduction, one of the problems with the robots is that they lose track of their own posture (position + orientation) in the field over time. So in order to make the goalkeeper able to determine its posture in the field, a self localization algorithm is needed. The design, implementation and testing of such algorithm will be the subject of this chapter.

5.1 Introduction

In this section a description of the problem, and a possible solution is given. Furthermore the operating conditions and specifications for the algorithm will be addressed.

5.1.1 The Problem

All the robots in the team are relying on odometry from the two wheels to determine their posture. This works fine unless one of three situations occur:

1. Either one or both of the wheels of the robot are not touching the ground.
2. Both wheels are touching the ground, but one or both are spinning due to lack of friction.
3. The robot is moving without the wheels turning.

The first is the case if the robot is ”caught” on another robot, e.g. hanging on the side of it. The second case happens if e.g. the ball is between two robots, both trying to push the ball in opposite directions. In both cases a wheel can turn without having the robot move accordingly, and thereby make the odometry wrong. In the third case, the posture (mainly the orientation) of the goalkeeper changes without the odometry changing, if the robot bumps into other robots (or they bump into it).

5.1.2 A Solution

To solve this problem an algorithm in the goalkeeper that is capable of determining the posture using existing sensors should be implemented. Since there are no long-range sonars or other sensors to determine distances from the goalkeeper to other objects, the only possible solution is to use the omni-directional catadioptric system. With images from this camera the area around the robot is visible from a top view as seen in figure 5.1. Since it is a property of the omni-directional catadioptric system that distances are

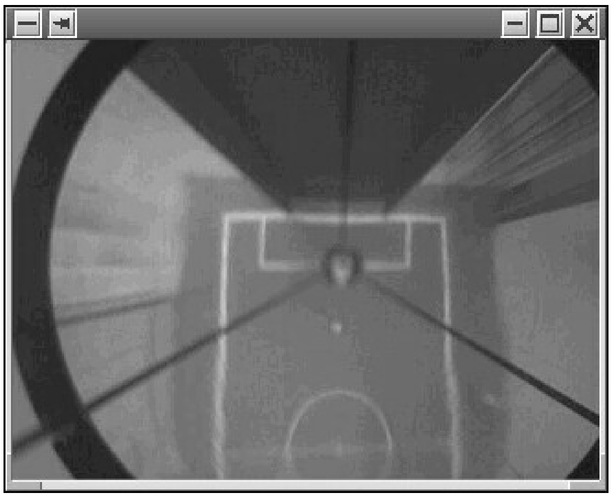


Figure 5.1: An image acquired from the omni-directional catadioptric system. In the center of the image the robot is seen, and in the right hand side of the image one of the goals is visible. Although the mirror has some errors, the distances in general are preserved.

preserved, it is possible to determine the distance from the goalkeeper to surrounding objects by evaluating images. Furthermore the goal posts will always point towards the position of the robot in the image.

5.1.3 Operating Conditions and Specifications

Under normal operation the goalkeeper should be near its own goal at all times. So a good object to detect from the image will be the goal. Extracting the goal posts as a feature from the image, will reveal the position of the robot relative to them. Then the posture of the robot can be estimated by geometry (except in one special case as it will be seen later).

From experienced human team members some rough specifications for the self localization algorithm was formed. It is expected that estimates of position within 20 cm. from the accurate position and estimates of orientation within 10° from the correct orientation will be sufficient for successful operation of the goalkeeper.

Ideally the execution time of the algorithm should be zero, but since this is not possible, it is evaluated in the following, what the maximum execution time should be. Basically there are two options: Running the algorithm while the robot is moving or not moving. If the algorithm is executed while the robot is moving, the posture estimated from the image will be inaccurate (since the robot will be moving away from the posture while the posture is calculated). The top speed of the scouts is $1\frac{m}{s}$, so in 0.2 seconds the scout will be 20 cm. away from the posture that will be output from the self localization algorithm assuming it is accurate. Taking inaccuracies in the algorithm into account, the execution time would have to be less than 0.2 seconds. This doesn't seem realistic.

If the robot is standing still in the field while self localizing, the movement of the ball in the time the algorithm is running is of interest. By rough estimation it is accepted that the ball moves two meters while running the algorithm. Assuming that the ball in general will not move faster than $4\frac{m}{s}$ it is decided that the maximum execution time should not exceed 0.5 seconds.

To sum up: The aim of the following is to design, implement and test an algorithm capable of determining the posture of the goalkeeper in the field within the tolerances defined previously. The posture will be determined from an image of the omni-directional catadioptric system by extracting the positions of the goal posts. The algorithm should not take longer than half a second to execute, and will be run as a behaviour as defined in the state machine in section 3.3.

In the following the algorithm will be presented. Later in section 5.3 implementation issues will be addressed, and in section 5.4 the results will be presented.

5.2 The Algorithm

To get an overview the algorithm is separated into four stages, as illustrated in figure 5.2. In the first stage the raw frame from the camera is processed to retrieve the contour of

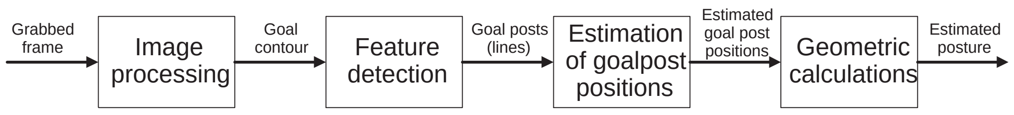


Figure 5.2: The four stages of the algorithm. Input is a grabbed frame from the camera and output is the estimated posture of the robot.

the goal. The contour is used as input for stage 2, where the goal posts are detected giving two sets of lines (one for each post) as output. For each of the two sets of lines the position of each goal post relative to the robot is estimated in stage 3. In stage 4 these positions are used in geometry to determine an estimate of the posture of the robot.

5.2.1 Stage 1: Image Processing

In order to retrieve the contour of the goal from the image two steps are necessary:

1. Segmentation of the goal in the image (identifying the pixels with the same color as the defending goal and disregarding all other pixels).

2. Edge-detection revealing the contour of the goal in the image.

Segmentation of the Goal

Having the grabbed image from the omni-directional catadioptric system the segmentation will be a question of taking each pixel in the image, and copy it to a new image, if and only if it has the color of the defending goal. Furthermore the images acquired from the camera are mirrored (because of the mirror in the omni-directional catadioptric system). This is inverted in the segmentation to make image inspection easier when debugging. The segmented goal from the image in figure 5.1 is illustrated in figure 5.3.

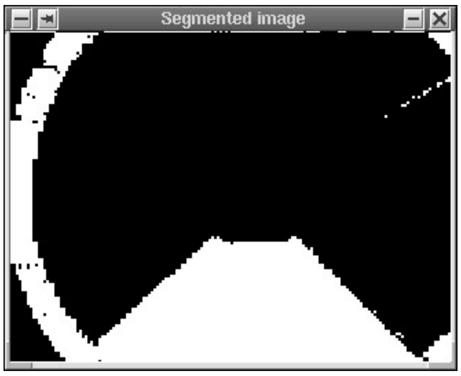


Figure 5.3: The segmented goal. The ring in the image is from the omni-directional catadioptric system.

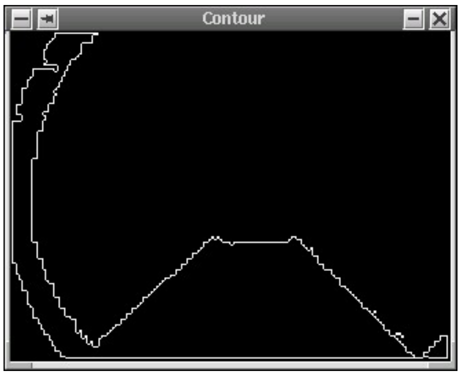


Figure 5.4: The contour of the goal. The ring from omni-directional catadioptric system will be disregarded later.

Edge-detection

Using the image containing nothing but the goal (see figure 5.3), the edges of the goal are extracted revealing the goal posts and the back of the goal (see figure 5.1). The extracted contour is seen in figure 5.4.

5.2.2 Stage 2: Feature Detection Using the Hough Transform

To detect the goal posts in the image containing the contour of the goal the following three steps are performed:

1. Hough Transform of the image.
2. Line filtering.
3. Goal post detection.

Applying the Hough Transform

To extract the goal posts from the image containing the contour of the goal, it is needed to detect lines in the image. For that purpose the Hough Transform is used. The Hough Transform is a popular method of extracting geometric primitives from images. The simplest version of the algorithm detects lines, but it can be generalized to find more complex features, [Intel, 2001]. The Hough Transform is described in more details in appendix B. The lines detected by the Hough Transform in the image containing the contour is seen in figure 5.5.

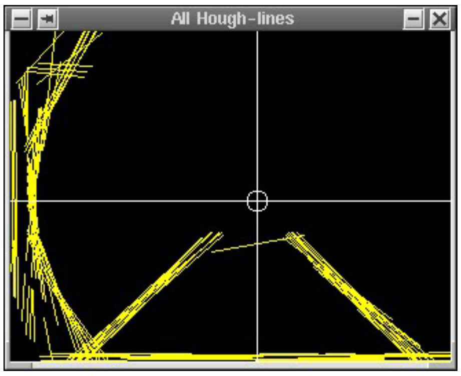


Figure 5.5: The output from the Hough Transform on the filtered image.

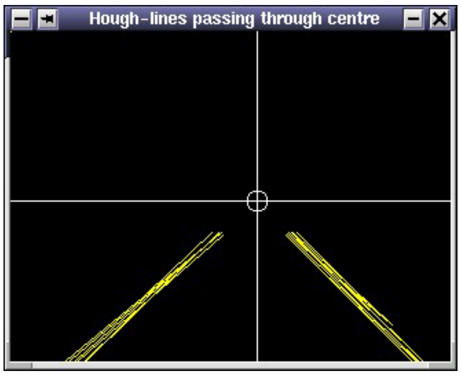


Figure 5.6: The lines passing through the robot (the center of the image). These lines are considered as goal post candidates.

Line Filtering

Applying the Hough Transform to the image containing the contour of the goal will result in lines belonging to either one of the goal posts and some lines not belonging to either of the posts. As mentioned earlier, it is a property of the camera that the robot is in the center of the image, and that the goal posts will point to the center. So all lines not fulfilling this requirement are disregarded, as seen in figure 5.6. The geometric approach used is described in appendix C.1.

Goal Post Detection

The lines that are regarded as goal post candidates are then evaluated to detect the goal posts. This is done by comparing the angles of all the lines in a histogram, and detecting the two angles most likely to be the angles of the goal posts. Afterwards two arrays of one or more lines are created containing the line(s) of each goal post. The goal post candidates are showed in figure 5.7.

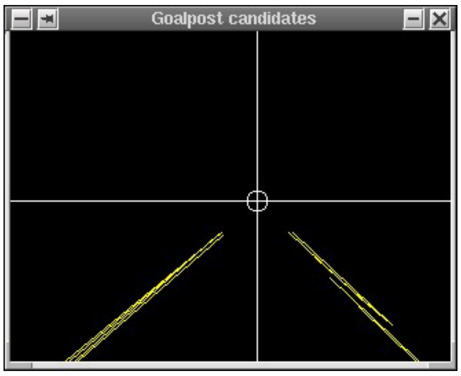


Figure 5.7: The goal post candidates.

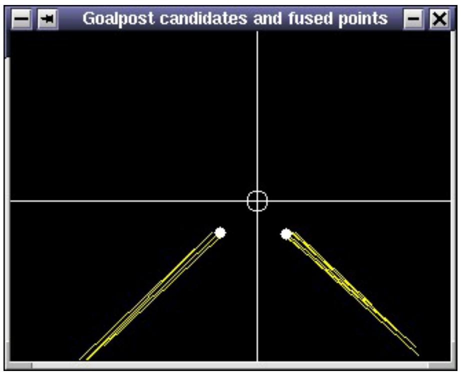


Figure 5.8: The goal post candidates and the estimated positions of the goal posts relative to the robot.

5.2.3 Stage 3: Estimation of Goal Post Positions

Having two arrays, each of one or more lines, the positions of each goal post are estimated using existing code in the robot. In the case where only one line describes the goal post, the position of the goal post is obviously the endpoint of the line closest to the robot. In the case there are more lines, the endpoints of the lines closest to the robot are used to estimate the position of the goal post. So, to estimate the two positions the following are executed:

1. Sorting line endpoints.
2. Estimating post positions.

The code is based on the methods presented in [Nunes, 2002], and will not be described further in this project.

5.2.4 Stage 4: Geometric Calculations

With the positions of the two goal posts relative to the robot, and the knowledge of the positions of the goal posts in the field, it is possible to determine the posture of the robot using geometry. For the calculations of the posture the same coordinate system as earlier is used, see section 4.2.3.

Since the positions of each of the goal posts in the field are known, determining the position of the robot relative to one of the posts will reveal the position, assuming the goal posts P_0 and P_1 are placed relative to each other as illustrated in figure 5.9. .

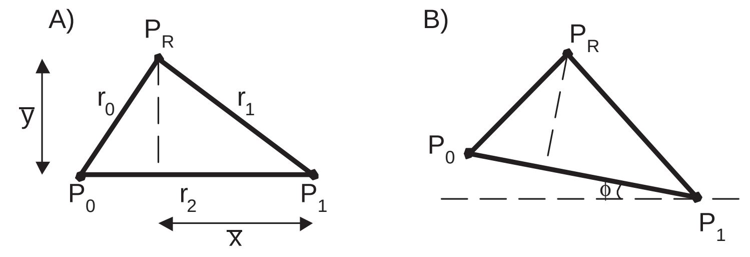


Figure 5.9: A) The robot position P_R relative to goal post P_1. B) Robot orientation is the same as the orientation of the line between the two posts in the image (taking signs of the angle into account). The illustrated situation corresponds to situation 5 in figure 5.10

Knowing the position makes it possible to calculate the orientation - in most cases. With the setup of the omni-directional catadioptric system on the robot, the location of the goal posts relative to the robot can be classified into ten different situations as illustrated i figure 5.10.

Problems arise in situations 9 and 10, that is when the goalkeeper is standing on or close to the goal line. Due to the geometry of the problem, it is not possible to tell one goal post from the other. Thus it is not possible to tell, if the robot is facing the goal or the field. In other words, the algorithm has an ambiguity. Telling if the orientation is e.g.

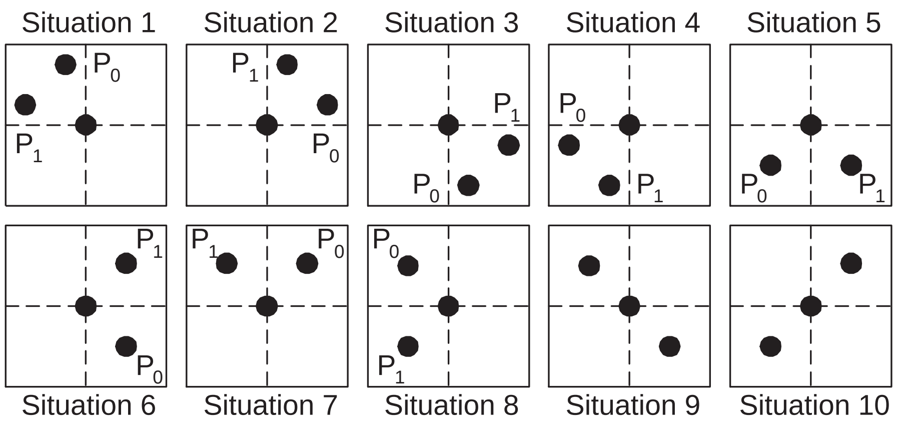


Figure 5.10: The 10 situations of goal post positions relative to robot.

$0°$ or $180°$ is not possible. One way of solving this would be to check in what side of the image the color of the goal is found, but due to lacking time this is not done in this project. Another disadvantage is that the calculation of orientation depends on the position. So if the position is wrong, the orientation will also be wrong. Further details of the geometry can be found in appendix C.2.

5.3 Implementation Issues

In this section specific technical details about the implementation are provided. The C-code for the algorithm can be found in [CD, \source\vision\gkself.c].

5.3.1 OpenCV

Since the algorithm is based on image processing, it was decided to use the "Open Source Computer Vision Library (OpenCV)" developed by Intel. The C-library has been used in other parts of the software for the robots. So some knowledge of using the library was available in the SocRob-group, thereby cancelling the negative fact that OpenCV currently is in beta version 0.3, and thus lacks some documentation.

The library among many things offers implementations of the Hough Transform, edge-detection algorithms, filtering functions and functions for showing images on screen for debugging. See [Intel, 2001], [Intel, 2003] and [Yahoo, 2000] for further details.

5.3.2 Error Handling

It turned out impossible to get a correct posture from the algorithm at every run. So it was decided to make the implementation some what conservative, in the sense ”rather not give a posture than give a wrong one”. To accomplish this the algorithm was stopped if one of six functions did not execute successfully as illustrated in figure 5.11. The first five of the functions returns failed if something goes wrong processing the image, e.g. if only

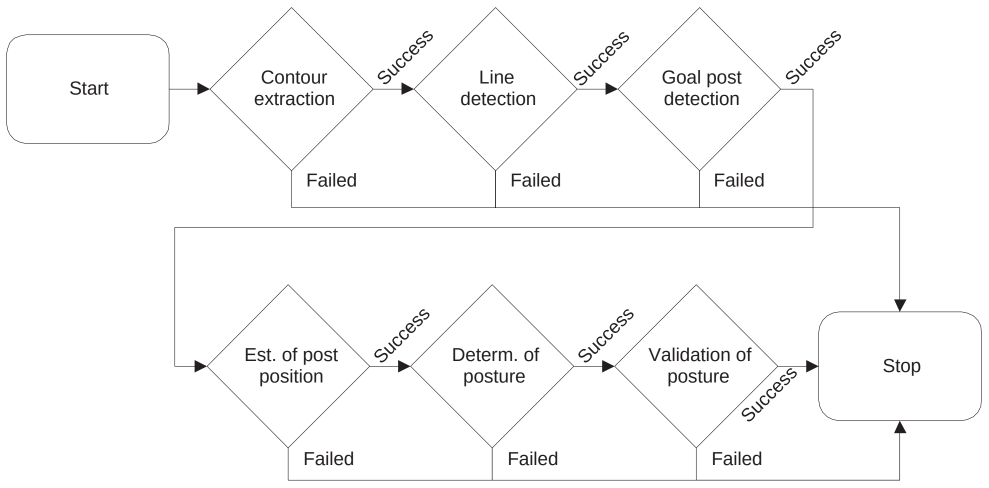


Figure 5.11: The error handling of the algorithm. If one of the six functions is not executed successfully the algorithm is stopped - and no posture is estimated.

one goal post is detected in the image or if the estimation of the position of one of the goal posts fails. In the sixth and final function the posture found is validated, so a posture more than half a meter outside the field is not accepted. The conditions for each of the functions to fail are given in table 5.1.

Function	Failes if
Contour extraction	No contour found in image.
Line detection	Hough Transform returns one line or none.
Goal post detection	Less than two goal posts detected in image.
Est. of post positions	Points used for estimation are to far apart.
Determ. of posture	No solutions found in geometric calculations.
Posture validation	Position of robot more than 0.5 m outside the field.

Table 5.1: Conditions for failure of the six functions used for error handling.

5.3.3 Filtering Noise From the Image

The estimation of goal post positions improves when each goal post is described by more lines. So it is desired, to have the Hough Transform return as many lines of relevant length and positions as possible. Thus, in order to get more lines as output from the Hough Transform, the image containing the contour (see figure 5.4) is filtered using OpenCV-functions. The filtering of the images ”smoothens” the lines, improving performance of the Hough Transform on the contour as illustrated in figure 5.12.

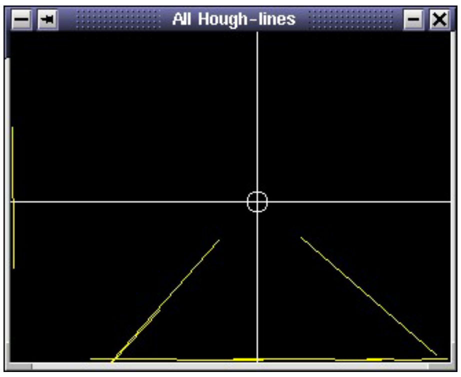


Figure 5.12: The result from the Hough Transform applied to the image in figure 5.4 without filtering.

5.3.4 Determining Goal Post Candidates

Having the lines from the Hough Transform as illustrated in figure 5.5 filtering of the lines is applied to detect goal post candidates. The filtering policy is as stated before to disregard all lines not passing through the center of the image (calibrated so the center of the image is considered the position of the robot in the image).

Having all the lines satisfying this condition, the next step is to detect the goal posts. This is done by comparing the angles of the lines and identifying the angles of the two goal posts using this procedure:

1. Sorting of line endpoints. To prevent confusion of angles, the array of line endpoints are sorted, so the point closest to the center of the image is regarded as the first point of the line and the other as the second.
2. The angle of each line is calculated.
3. A histogram of the angles is constructed.
4. The angle of each of the goal posts are determined.
5. Two arrays (one for each goal post) containing lines are created.

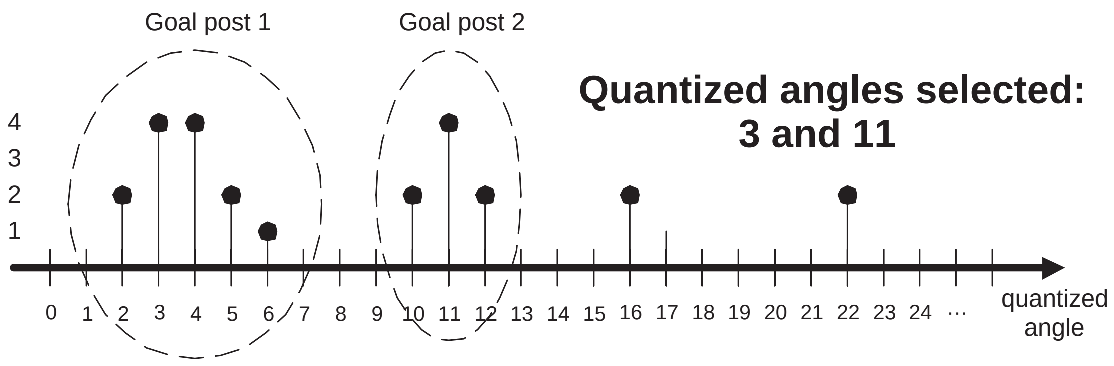


Figure 5.13: The histogram of the quantized angles.

The challenging part of the procedure is the determination of the angle of the goal posts. Here it is done by evaluating a histogram of quantized angles and thereby determine the two angles with the most lines. Those angles are regarded as the angles of the goal posts. The histogram is created by considering quantized versions of the angles found in step 2 as illustrated in figure 5.13. The policy for selecting angles of the goal posts is also illustrated in the figure. Occasionally it happens that very few lines are detected for one goal post and many lines are detected for the other goal post. To prevent selection of wrong angles in this situation, it is required that there should be at least one quantized angle with no lines (corresponding to quantized angles 7, 8 and 9 in figure 5.13) between two angles in the histogram.

5.4 Results

To test the algorithm two tests were performed. First a general test, to determine whether it is possible to estimate the posture of the robot. Afterwards the performance is tested, by considering the quality of the results in various positions in the field. Both tests were performed under ideal conditions: No other robots were present in the area visible to the robot. In the following the results of the two tests will be presented.

5.4.1 General Test

The robot was placed in a position centered one meter in front of the goal. The robot was then rotated in that position to eight different angles ($0°$, $45°$, $90°$, $135°$, $180°$, $-135°$, $-90°$ and $-45°$). The algorithm was run 20 times in each angle, totaling 160 measurements of posture.

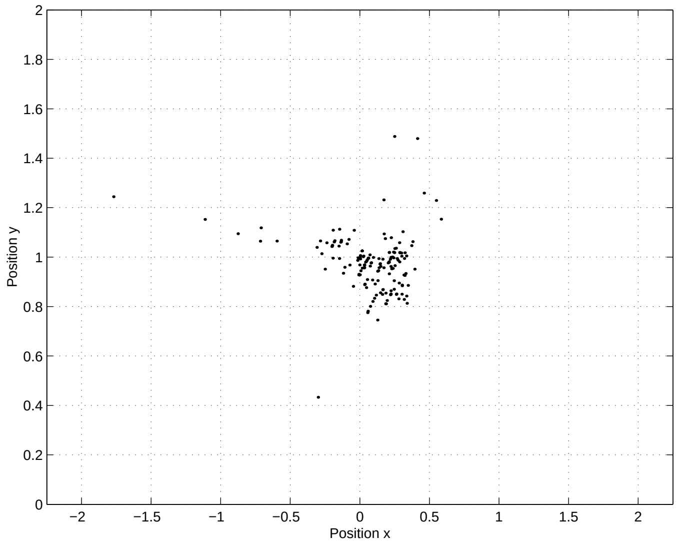


Figure 5.14: The estimated positions.

In figure 5.14 the estimated positions is presented, and the estimated angles are plotted in figure 5.15. In the estimation of the position some outliers are detected. And

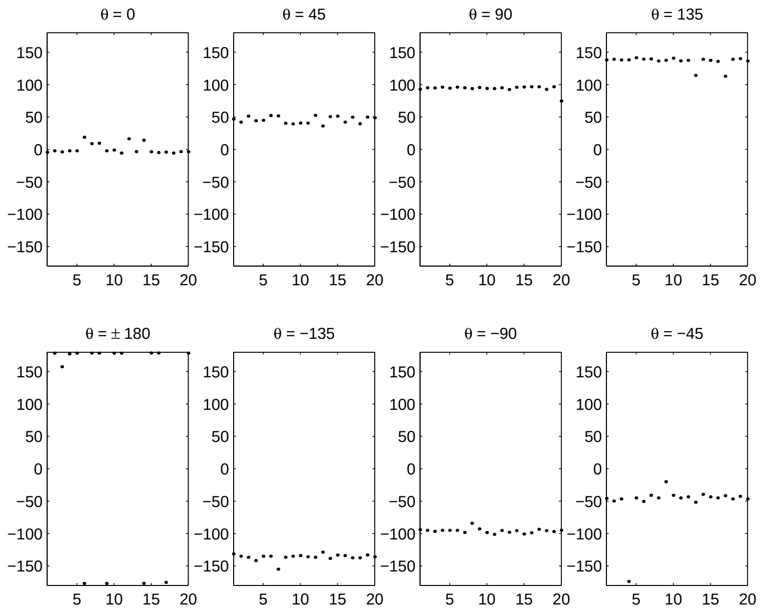


Figure 5.15: The estimated angles of orientation. 20 estimations were made of each angle.

since the calculation of the orientation is dependent on the position the outliers propagate and introduces outliers in the orientation as well. Based on the results, it can be verified that the algorithm is working, since the position and orientation is estimated with satisfying results.

5.4.2 Performance Test

To test the performance of the algorithm, two issues were considered: The success rate of the algorithm and the error of the estimated postures. The robot was placed with the same orientation in 60 different positions in front of the goal, as illustrated i figure 5.16. The positions were numbered from 1 to 60, starting from the corner of the field in the left of the goal. In each position the algorithm was run until 20 estimates of the posture was found (in some it wasn't possible to get 20 estimates though).

Rate of Success

As explained earlier the algorithm doesn't return any posture if one of several conditions is not met (e.g. if only one goal post is found in the image). Having this in mind, the rate of success for the algorithm is defined as the ratio between algorithm runs which are successful (a posture is found) and the total algorithm runs. The rate of success was evaluated in each of the 60 positions in the field. The results are plotted in figure 5.17. As expected the rate of success on the goal line outside the goal is zero, since the goal color is not visible from here. Also it is seen that the rate of success is very low in positions 2 m. and further away from the goal. The algorithm has the best rate of success in the

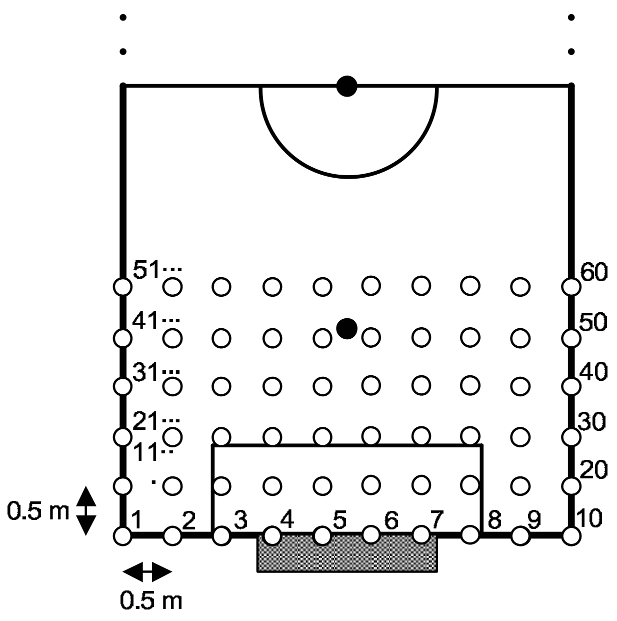


Figure 5.16: The 60 positions in which postures were estimated.

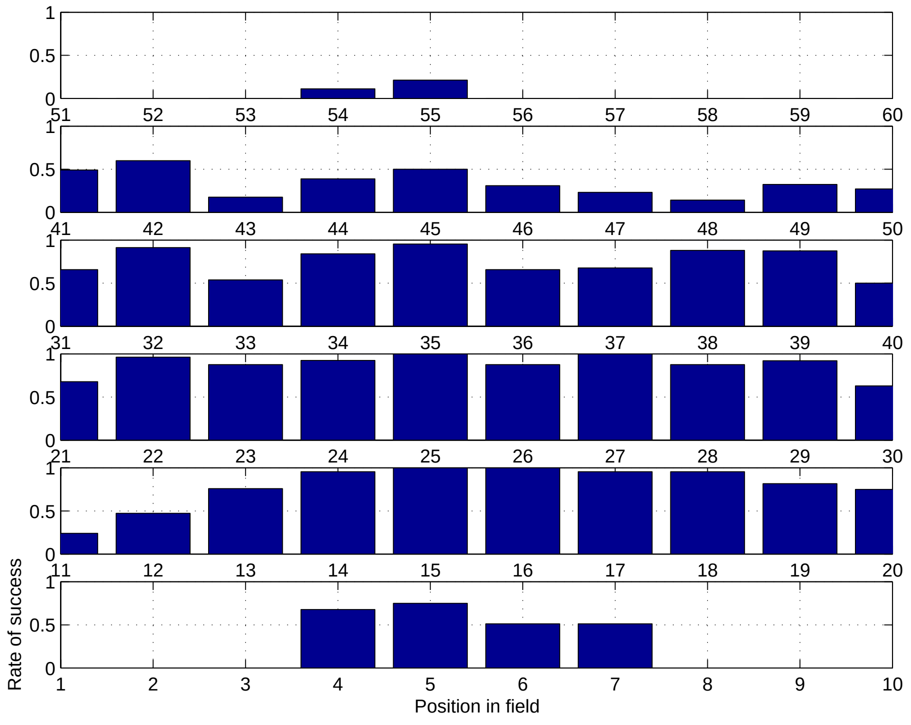


Figure 5.17: Rate of success in the positions in front of the goal.

positions 1 and 1.5 meters away from the field end line, and in general there are more successes in positions in front of the goal, than in the positions not in front of the goal.

5.4.3 Error of Estimated Positions

Having considered the rate of success, the next area of interest is the error of the position estimate. In figure 5.18 the estimated positions in four positions are illustrated. In the

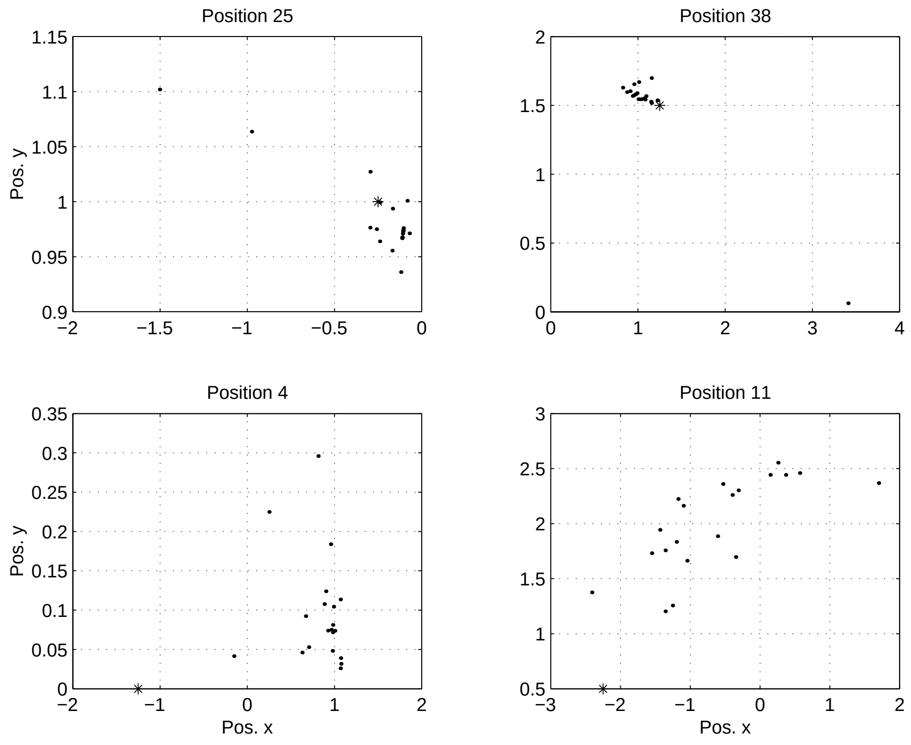


Figure 5.18: Estimated positions in position 25, 38, 4 and 11. The asterisks indicates the actual position. Note: The plots has different scales.

top row the estimated positions in position 25 and 38 are shown. For position 25 the estimates are close to the actual position. In the y direction all estimates are within 10 cm., but in the x direction some outliers about 75 cm. and 1.5 m. are detected. Also in position 38, which is not in front of the goal, good results are obtained. There is one outlier (with dramatic error), but all other estimates are close to the actual position. In the lower row the estimates in position 4 and position 11 are shown. Position 4 is on the goal line, and the results are poor. Almost every estimate is very poor. This is due to poor detection of goal post as illustrated in figure 5.19. To prevent major errors to be introduced, a threshold was applied to the x-position. The calculated x position was compared to value of the x position from odometry. If the difference was larger than 1 m., the posture was not used. The threshold avoids large errors on the posture, but it also makes it impossible to run the algorithm successfully if the posture is really wrong. In position 11, which is on the side line, the results are bad as well, but better than on the goal line. In the y direction the error is within half a meter, but in the x direction the error is much greater.

To evaluate the overall quality of the position estimates, a mean of the estimates (of both x and y directions) in all the positions were made, and the error of the mean

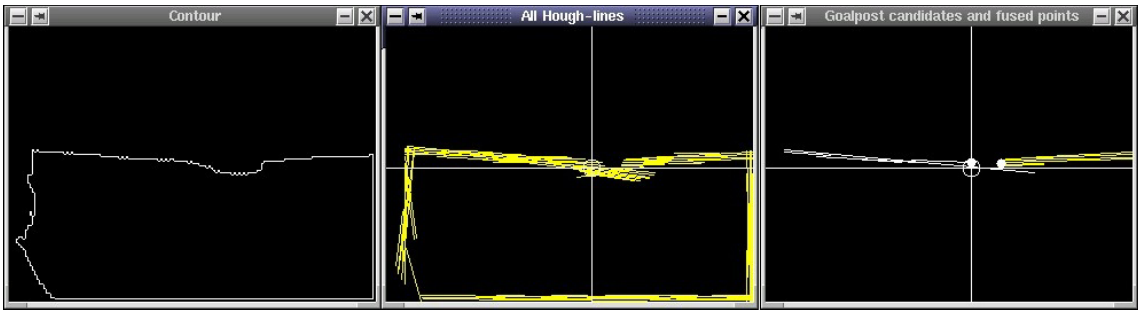


Figure 5.19: Situation from the goal line. The detection of the posts are failing, because the back of the goal is mistaken for goal post. To prevent major errors to be introduced, a threshold was applied to the x-position.

calculated. The errors are plotted in figure 5.20. The errors of the means are in general

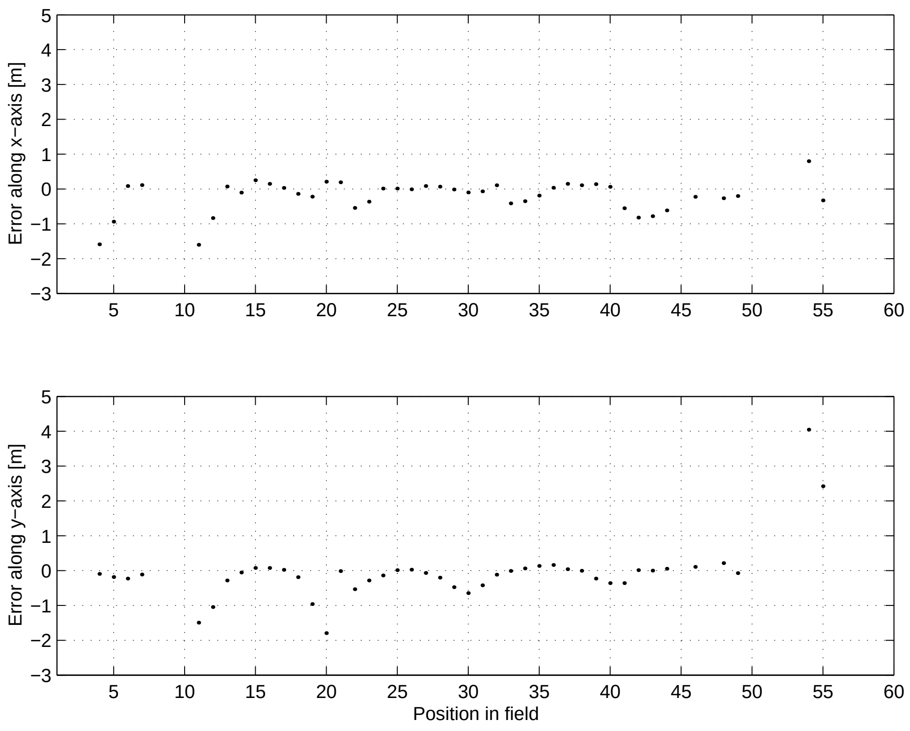


Figure 5.20: The error of the mean in x and y directions in the positions measurements were performed in.

close to zero with some exceptions. The error of the mean i the x direction is acceptable in most cases in positions 13 to 40. In the positions not in front of the goal, the error of the mean in the y direction increases, but in general the error is small until position 50. The error of the estimates in the positions not in front of the goal, is due to an unforeseen property of the goals. It turns out, that when the robot is placed outside the goal posts (meaning not in front of the goal), it is not possible to detect the nearest goal post. The situation is illustrated in figure 5.21. Instead of detecting the goal post, the back of the goal is mistakenly considered as a goal post. Since this is 40 cm. behind the goal line, an error is introduced. The error arises from the shape of the goal. It only has the goal color on the inside, and for the closest post, the color is lost of sight in the positions mentioned here.

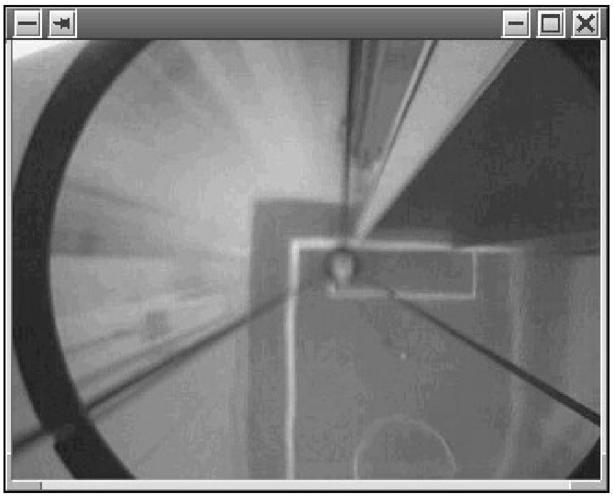


Figure 5.21: If the robot is placed outside the goal posts, the closest goal post is not detected.

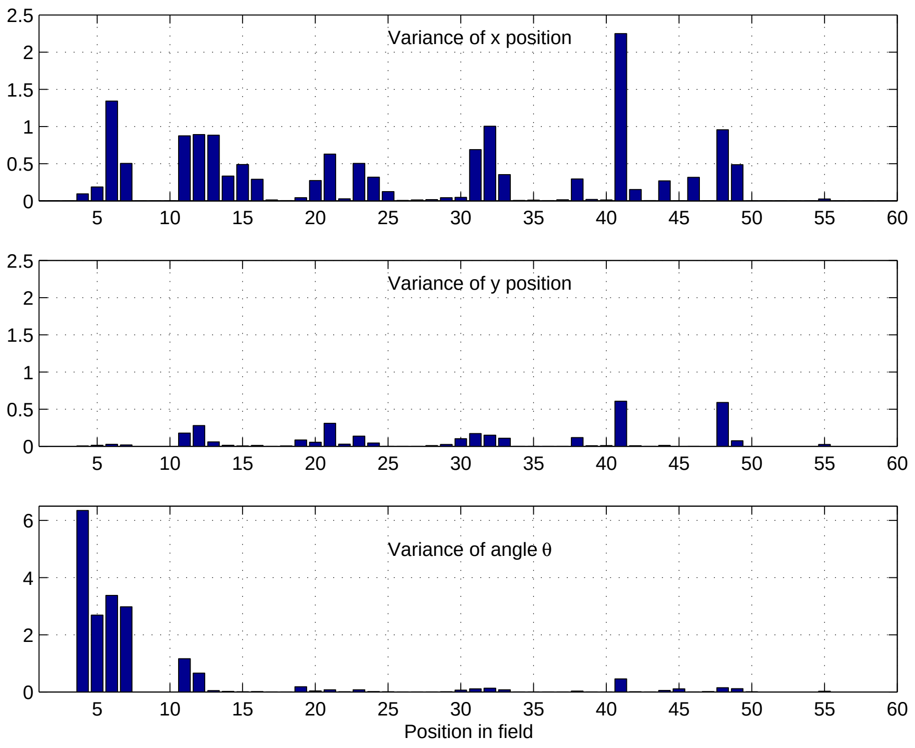


Figure 5.22: Variance of position in x and y, and angle θ.

Having considered the error of the mean estimated position, the variances of estimates in each positions are considered. The variances of the orientation and x and y directions are plotted in figure 5.22. As expected the variance of the orientation is high in the positions on the goal line. This is expected, since the orientation is dependent of the position, and the estimates of the positions on the goal line are poor. The variance of the x position is in general higher than the variance of the y position. At this time the group cannot offer any reasonable explanation to this.

5.4.4 Time Consumption

The time consumption of the algorithm was evaluated by adding timestamps to the code. The time consumption of each of the four stages are listed in table 5.2. The time consumption of the algorithm is not constant. The time used in stage 2 is varying because of the Hough Transform. The further away from the goal the robot is, the fewer lines are detected. And thus the implementation of the Hough Transform executes faster. In order to be able to estimate postures further away from the goal than 1 m., it was decided to accept the higher time consumption in positions close to the goal.

Stage	(0,1)	(1, 1.5)	(0,2)
1	0.1531 s	0.1521 s	0.1447 s
2	0.6517 s	0.3002 s	0.2577 s
3	0.0004 s	0.0003 s	0.0002 s
4	0.0018 s	0.0005 s	0.0009 s
Total	0.8070 s	0.4531 s	0.4035 s

Table 5.2: The time consumption in each of the four stages at three different positions, (0,1), (1, 1.5) and (0,2).

5.5 Conclusion

In this chapter, a self localization algorithm for the goalkeeper was designed, implemented and tested. The algorithm is able to estimate the posture of the goalkeeper with some limitations leaving room for improvement. The tests performed on the algorithm were run under idealized conditions, since no other robots were present. This is not guaranteed to be the situation when playing against other teams, so tests with other robots around would be relevant. It is certain though, that the performance will not improve, because the line of sight to the goal will be disturbed making goal post detection harder.

Clearly the results are best in the area in front of the goal. The results on the goal line and on the side lines are not as good. For the goal line this was expected, since the orientation would be wrong due to the ambiguity, but not that the position would have such a large error.

One way of improving the algorithm would be to detect more features. If the position could be estimated based on e.g. the posts in the corner of the field, the two postures found could be used to estimate a posture. For this purpose, the same code as used for estimating goal post positions from goal post candidates, could be used.

The algorithm is currently limited to be used in the goalkeeper, but it could be easily used in other robot team members for self localization. If e.g. one of the attackers is close to the opponent goal, the algorithm could be used to determine its position relative to this goal instead of its own goal.

CHAPTER 6

State Machine Simulations

In this chapter a simulation of the state machine will be performed. The simulation is limited to contain the two behaviours FollowBall and InterceptBall. These behaviours will be tested in a number of different situations - shots directed from different places of the field against the goalkeeper, that is. Two models, one for the movement of the ball and one for the movement of the goalkeeper will be combined to perform the simulation. The results from this simulation will be used in determining the optimum size of the danger zone and the radius of the defensive arc in the behaviour FollowBall. The purpose of these tests is to find the configuration that will lead to the highest number of saves by the goalkeeper. The tests are necessary due to the fact that the project-group has not been able to determine the optimum configuration in real games. The optimum configuration found in the simulations can be used by the SocRob-team in future tournaments.

All simulations will be performed using CheckMate [Carnegie Mellon, 2003], a toolbox developed to the MATLAB/Simulink environment. CheckMate is free to download and furthermore it was recommended by a project supervisor. First a general introduction to hybrid automata is given followed by the design of the two automata needed here: Ball and goalkeeper. These hybrid automata will then be implemented in CheckMate, and the results will be presented. Finally a conclusion will be drawn.

6.1 Introduction to Hybrid Automata

A hybrid automaton, shown in figure 6.1, is a generalization of the finite automaton that includes continuous dynamics within each discrete state. Each discrete state in the hybrid automaton is called a location. Associated with each location is the invariant, the condition which the continuous state must satisfy while the hybrid automaton resides in that location, and differential equations representing continuous dynamics in that location. Transitions between locations are called events. There are two types of events: Internal and external. An internal event happens, when the invariant is no longer satisfied. An external event happens when a transition in the hybrid automaton is forced by another system, that does not belong to the hybrid automaton [van der Schaft; Hans Schumacher, 1999]. Each event is labeled with the guard and the reset conditions on the continuous state. The location transition is enabled, i.e. allowed to be taken, when the guard condition is satisfied. Upon the location transition, the values of the continuous state before and after the transition must satisfy the reset condition.

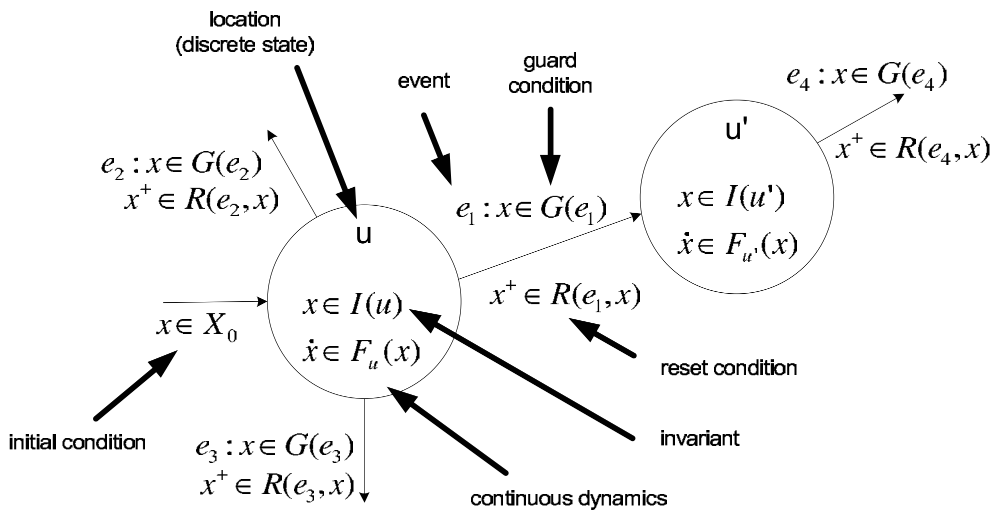


Figure 6.1: A hybrid automaton.

6.2 Design of Hybrid Automata

In the previous section an introduction to hybrid automata was given. In this section automata for the ball and goalkeeper will be designed. The designed automata will be used later in the CheckMate models.

6.2.1 Ball

In this section the aim is to develope a hybrid automata that can simulate the movement of the ball in the field. The ball should be able to move freely within the specified area shown in figure 6.2 A. Given an initial position and velocity the ball should be able to bounce off the sides of the field, as illustrated in figure 6.2 A. When the ball bounces the angle of incidence θ should be equal to the angle of reflection θ. The kinematics of the ball is simplified to move along a straight line with constant velocity and without acceleration. With these specifications it will be possible to simulate shots fired from every spot in the field. First the kinematics of the ball is considered, a differential equation of the form $\dot{x} = Ax + b$ is found. In future simulations the position and velocity of the ball is always needed. These four states can be written as follows:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \Rightarrow \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \tag{6.1}$$

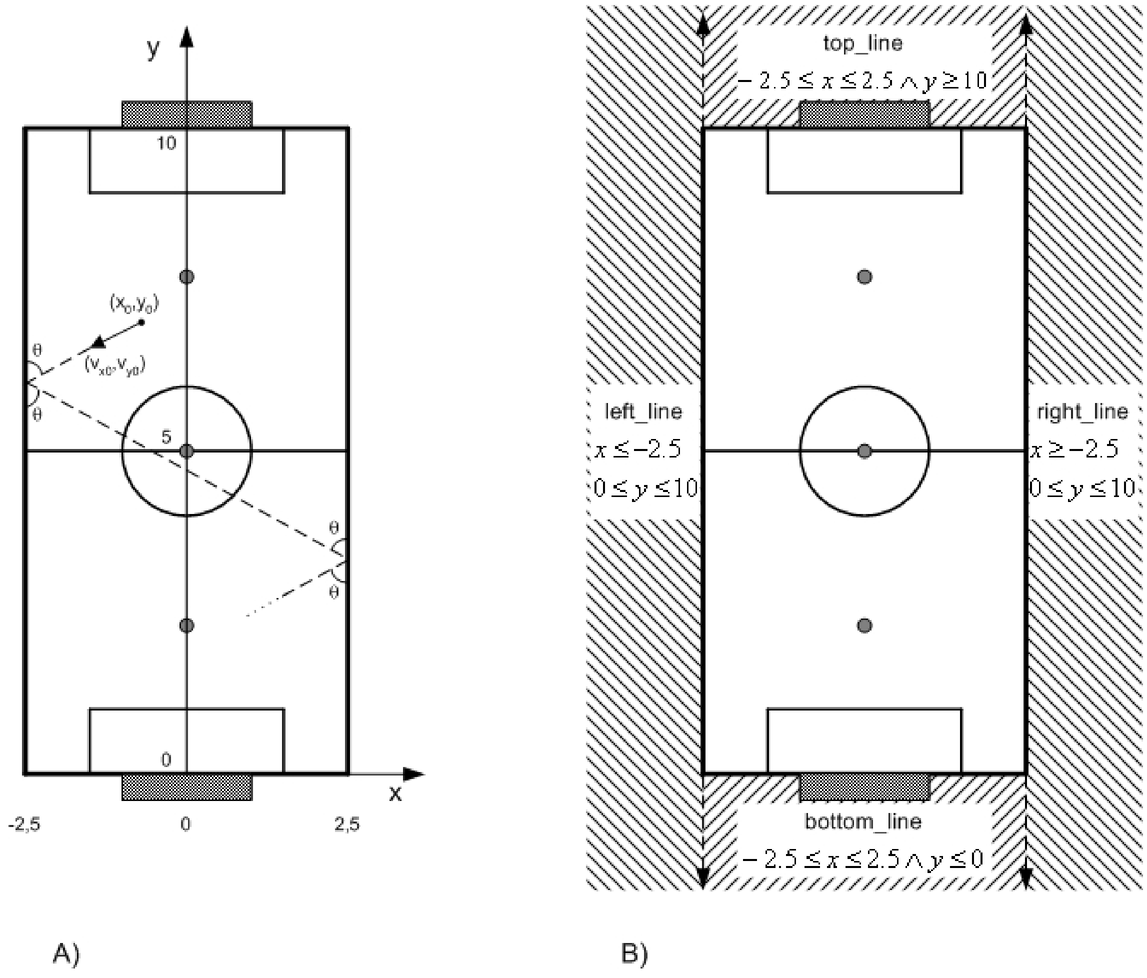


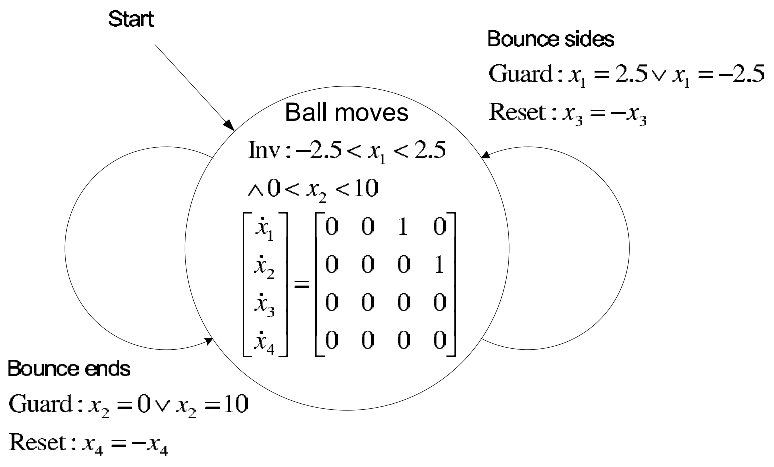
Figure 6.2: A) The principle of the moving ball. B) The polyhedral regions specified in CheckMate.

Acceleration is not wanted, so a_x and a_y can be set to zero. v_x and v_y is given by the states x_3 and x_4. The differential equation can be written as:

$$\dot{x} = Ax + b \Rightarrow \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{6.2}$$

For the ball to bounce, the velocity has to be reset. There are two different situations. When the ball hits one of the sides, the sign of v_x is changed. When the ball hits one of the ends, the sign of v_y is changed. This change corresponds to a jump in the continuous states of a hybrid automaton.

The movement of the ball can be modeled by a hybrid automaton with only one discrete state. This hybrid automaton is shown in figure 6.3. In section 6.3 this hybrid automaton is implemented in Simulink using CheckMate blocks.

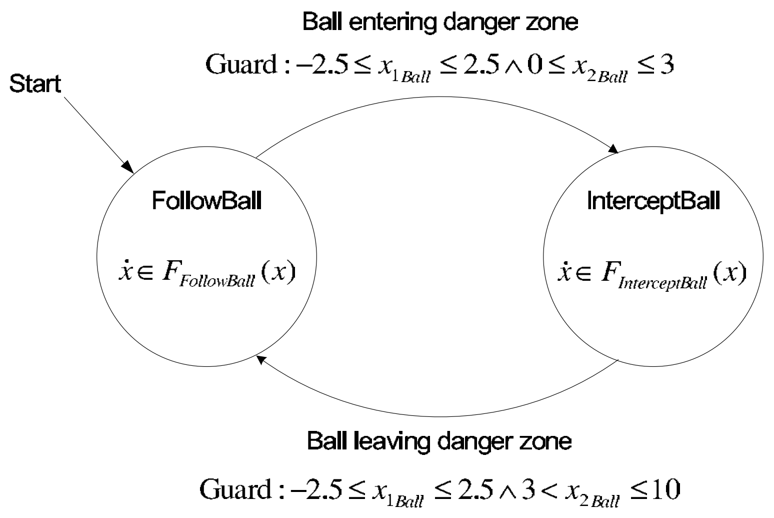


Start
Bounce sides
Guard : $x_1 = 2.5 \vee x_1 = -2.5$
Reset : $x_3 = -x_3$
Ball moves
Inv : $-2.5 < x_1 < 2.5$
$\wedge 0 < x_2 < 10$
$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
Bounce ends
Guard : $x_2 = 0 \vee x_2 = 10$
Reset : $x_4 = -x_4$

Figure 6.3: The ball movement modeled as a hybrid automaton.

6.2.2 Goalkeeper

The hybrid automaton for the goalkeeper will contain two locations, FollowBall and InterceptBall. The movement will be described by two states, x_1 and x_2, the x- and y-components of the goalkeepers position in the field. Transitions between these two locations will be made according to the position of the ball relatively to the danger zone. Thus, the event that causes a transition in the hybrid automaton is an external event. The principle of the danger zone is illustrated in figure 3.3 on page 15. In this section the danger zone is defined as the area: $-2.5 \leq x_1 \leq 2.5$ and $0 \leq x_2 \leq 3$. The two locations of the hybrid automaton will keep the kinematics derived in section 4.2.1 on page 20 and 4.3.1 on page 33 respectively. Thus, the dynamics for the motors that force the goalkeeper to move, and the dynamics for the goalkeeper itself is not considered. Figure 6.4 shows the hybrid automaton for the goalkeeper. To keep the figure simple, the kinematics within each location is not entirely specified, though named $F_{FollowBall}(x)$ and $F_{InterceptBall}(x)$. In the next section the hybrid automaton for the ball and goalkeeper is



Ball entering danger zone
Guard : $-2.5 \leq x_{1Ball} \leq 2.5 \wedge 0 \leq x_{2Ball} \leq 3$
Start
FollowBall
$\dot{x} \in F_{FollowBall}(x)$
InterceptBall
$\dot{x} \in F_{InterceptBall}(x)$
Ball leaving danger zone
Guard : $-2.5 \leq x_{1Ball} \leq 2.5 \wedge 3 < x_{2Ball} \leq 10$

Figure 6.4: Hybrid automaton for the goalkeeper.

implemented in Simulink using CheckMate blocks.

6.3 Implementation - CheckMate

The hybrid automata derived in section 6.2 was implemented in CheckMate, and in appendix D on page 83 a further description of implementation issues is given. This section will present the model, and the files used can be found in [CD, \checkmate]. Figure 6.5 shows the CheckMate model. The model is used in the following tests. These tests will

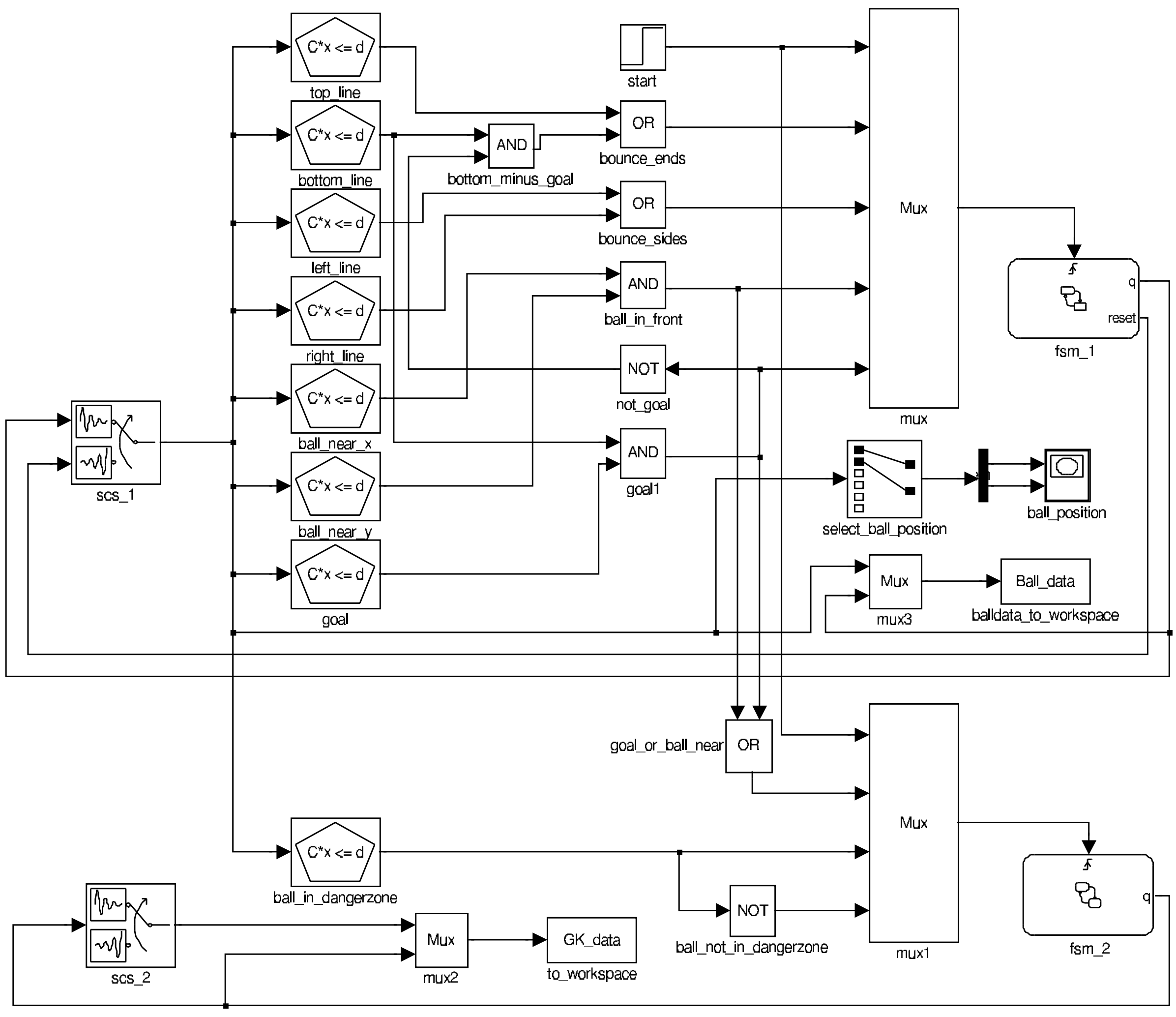


Figure 6.5: CheckMate model for the movement of the ball and the goalkeeper, with behaviours FollowBall and InterceptBall.

be described and the results presented in the section below.

6.4 Results

To find the optimum size and shape of the danger zone and the optimum length of the radius of the defensive arc, three tests were performed. The tests will reveal the goalkeepers ability to save shots fired from a distance or up close to the goal. The tests can not reveal how the goalkeeper will be able to defend the goal, when the opponent is dribbling towards the goal. All results shown below will be presented as a percentage of saves by the goalkeeper.

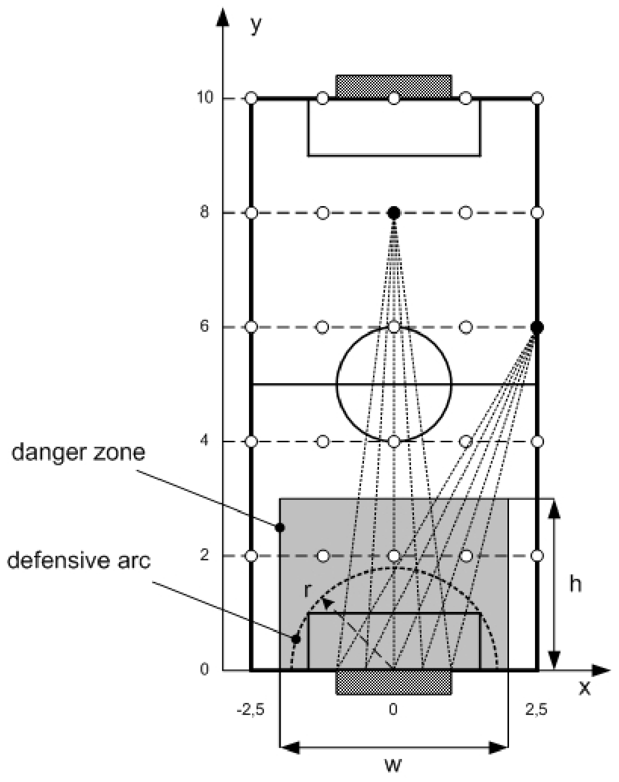


Figure 6.6: Shots fired from different positions in the field, aimed at the goal.

In each of the three tests, shots were fired from different locations in the field, each shot aimed towards the goal. In every different setup for the danger zone or the radius of the defensive arc, 625 shots were fired against the goal. The speed of the shots differed. In five groups of 125 shots each, the speed was set to 2m/s, 2.5m/s, 3m/s, 3.5m/s, 4m/s respectively. Figure 6.6 gives an overview of the field, and how a group of 125 shots was fired from different locations. The white dots placed at five horizontal lines are all initial positions for the ball. It is seen that for every white dot, five shots are fired, aimed at different spots within the goal. E.g., shots from two different locations is shown in figure 6.6, five shots fired from the coordinates (0,8) and other five from (2.5,6) and so on. The initial position of the goalkeeper before each shot is fired, is at the defensive arc in front of the goal.

This section is divided in two parts. The first part will deal with tests concerning the danger zone and the last part will concern the radius of the defensive arc.

6.4.1 Danger Zone

Two tests to find the optimum size and shape of the danger zone were performed.

Test One

First, to find the optimum height h (see figure 6.6) of the danger zone, h was varied. The following configuration was used in the test:

$$h = 2, 3, 3.5, 4.5, 5, 6, 7$$

$$r = 1.2$$

$$w = 5$$

Figure 6.7 shows the results. It is seen that the highest percentage of saves by the goal-

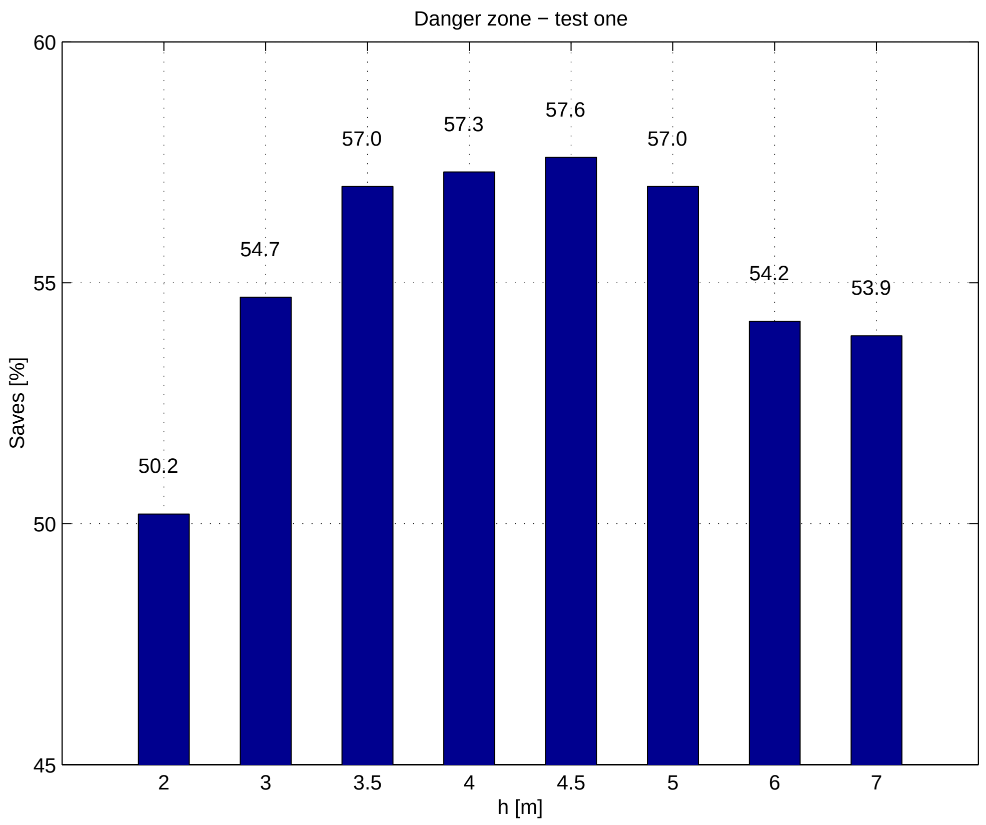


Figure 6.7: The results of test one concerning the danger zone. w is fixed to $5m$, while h is varied. r is set to $1.2m$.

keeper is carried out with $h = 4.5$ (57.6%). Is h increased further the percentage of saves is decreasing. Likewise if h is decreased from 4.5 the amount of saves is decreased. The lowest percentage of saves is obtained with $h = 2$. Three values of h is chosen for further investigation in the following tests. Due to the fact that $h = 4.5$ has the highest percentage of saves, $h = 4, 4.5, 5$ is chosen.

Test Two

Three values of h were selected in test one. The purpose of test two is to examine whether varying w will increase the percentage of saves. The following configurations were used in the test:

$$h = 4, 4.5, 5$$

$$r = 1.2$$

$$w = 3, 4$$

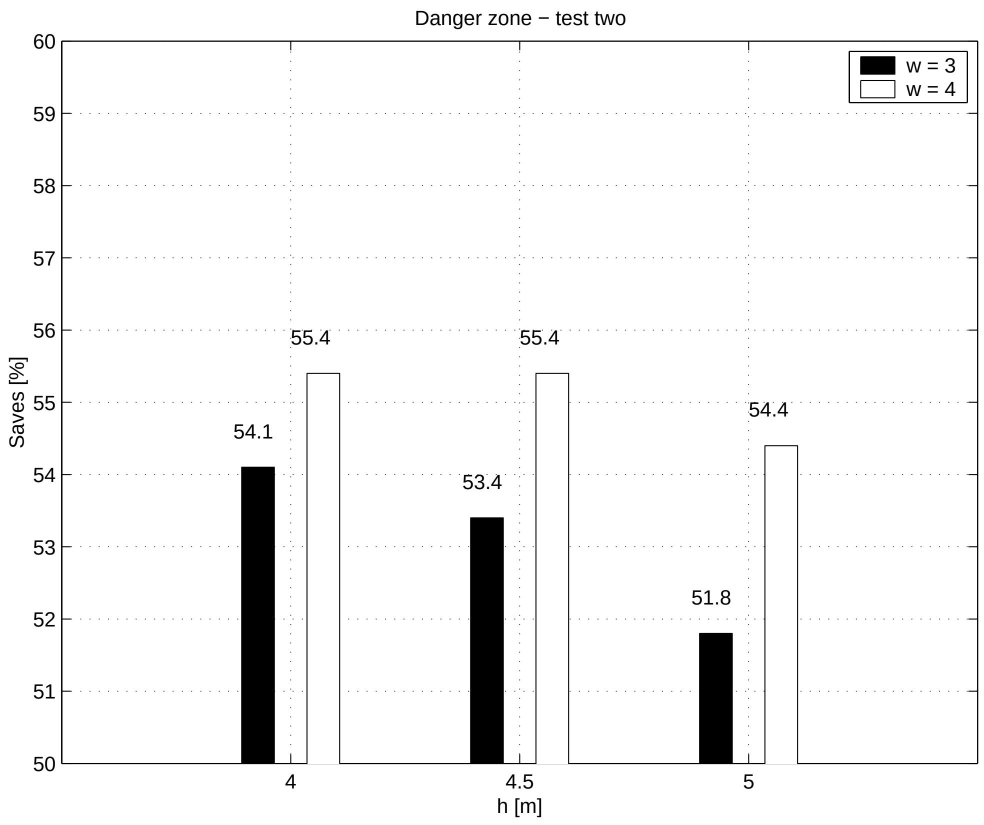


Figure 6.8: The results of test two concerning the danger zone. w and h is varied, while r is set to $1.2m$.

Figure 6.8 shows the results. The black bars shows the results for $w = 3$ and the white bars the results for $w = 4$. The highest percentage of saves are shared by two values of h, namely $h = 4$ and $h = 4.5$. Both has $w = 4$. But still these percentages are not as high as the results achieved in test one. Test two does not reveal a better result, when w is varied.

The test only shows how the goalkeeper reacts to shots. It gives no idea of how the goalkeeper will react, if for example an opponent dribbles along one of the sides all the way to the bottom line. Maybe here it would be best for the goalkeeper to stay in FollowBall, instead of intercepting. This could happen only if $w \neq 5$.

In spite of the reflections written above, the best result of the model is still obtained with $w = 5$, $h = 4.5$ (section 6.4.1). Therefore this configuration is used in the last test, to find the optimum length of the radius in the defensive arc.

6.4.2 Radius of Defensive Arc

In the last test it is investigated whether a variation in the radius of the defensive arc is improving the percentage of saves by the goalkeeper.

Test Three

For test three, five different values of r was chosen. w and h was kept at $5m$ and $4.5m$ respectively. Therefore the following configuration was used in the test:

$$r = 0.6, 1, 1.2, 1.5, 2$$

$$w = 5$$

$$h = 4.5$$

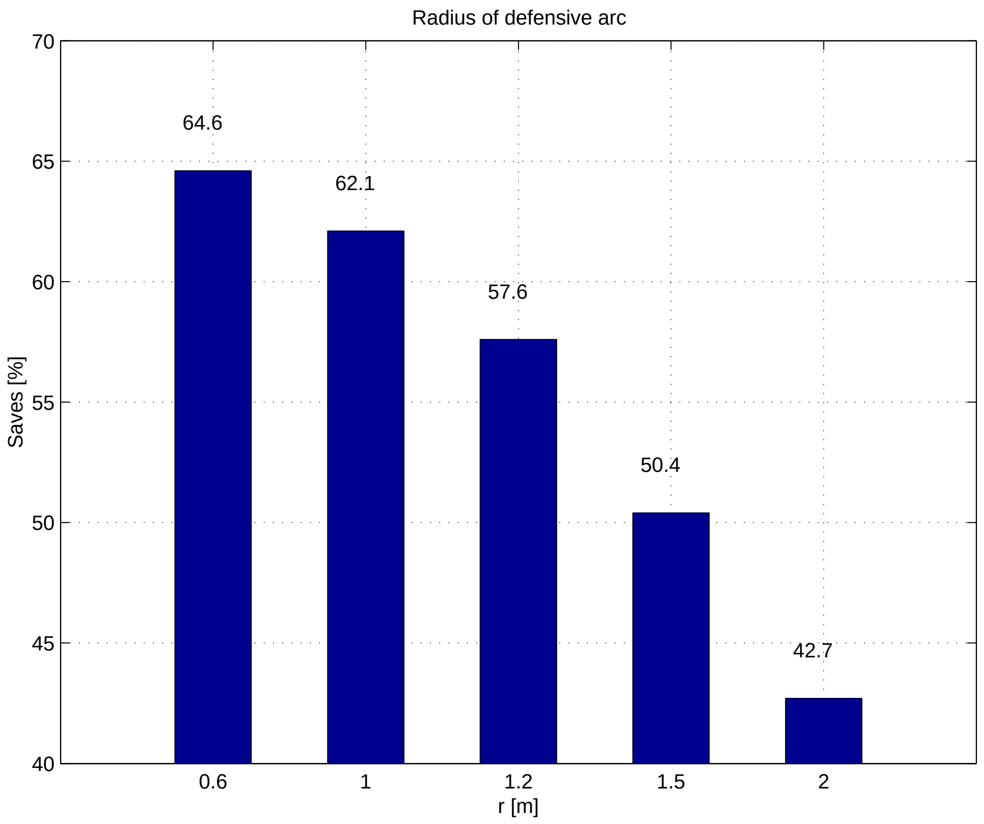


Figure 6.9: The results of test three concerning the radius of the defensive arc. r is varied, while $h = 4.5m$ and $x = 5m$.

Figure 6.9 shows the results. As expected, we get the same result with $r = 1.2m$, as test one showed (section 6.4.1). With $r = 0.6m$, the highest percentage (64.6%) of saves is achieved. The lowest percentage (42.7%) is achieved with $r = 2m$.

At first $r = 0.6m$ seems as a good result, and by far the best result of the three tests in all. But it seems strange to have a radius of the defensive arc set to $r = 0.6$. At this length, the goalkeeper is not able to cover the goal entirely when defending in the FollowBall behaviour. On the other hand the result of test three is not that surprising. When the goalkeeper is making a transition from the FollowBall to the InterceptBall behaviour, the goalkeeper is always intercepting the ball at a straight line $y = 0.5m$. Due to this, when $r = 0.6m$, the distance from the goalkeeper to the line $y = 0.5$, is shorter than when $r = 2m$. This is most likely the reason for the high amount of saves when $r = 0.6$.

6.5 Conclusion

Test one showed that the configuration $h = 4.5m; r = 1.2; w = 5$ achieved the highest percentage (57.6%) of saves. This configuration and the two adjacent configurations to the best result, was chosen to be investigated further in test two. Test two revealed that no increasement in the percentage of saves was achieved by setting $w = 3$ or $w = 4$. It was mentioned that the result of the test does not give the reason to completely reject the possibility of having the danger zone differ from $x = 5$.

The best results of test one ($h = 4.5m; r = 1.2; w = 5$) was used as a basis for test three. Here the radius of the defensive arc was varied. Test three showed the best result with $r = 0.6m$. Here the percentage of saves was 64.6%. This result was the best of all tests. The reason for the high success was given to the fact that the goalkeeper is intercepting balls on a straight line ($y = 0.5$). The closer the goalkeeper is to this line when running in FollowBall, the higher rate of success is achieved.

The experiences obtained in test three, indicates that the behaviour FollowBall is used in a wrong way. The advantages of letting the goalkeeper follow the defensive arc seems few, when the goalkeeper is always seeking to the line $y = 0.5m$, when intercepting the ball. A more advanced InterceptBall should be implemented to prevent this, so the motion towards the point would be a straight line starting in the position of the goalkeeper. Thus choose the shortest way to the ball, instead of calculating the point of interception at the same horizontal line.

Overall the tests performed in this chapter reveals a tendency of how the goalkeeper reacts to shots. Though, the results, and especially the margin of results, indicates that a final conclusion for the danger zone and radius of defensive arc, can not be drawn here. To make a final decision of this matter, a refinement of the model has to be done. Especially the dynamics for the goalkeeper and the delays in the software has to be included.

CHAPTER 7

Joint Test of Behaviours

Because of the limited time available in the project the behaviours Go2Area and KickBall was never implemented or designed. Thoughts of implementing a simple version of KickBall as a primitive task was made. If the goalkeeper was in InterceptBall and in the primitve task BLOCK the goalkeeper might as well try to kick the ball away if it is within reach of the kicker.

A joint test between the two behaviours FollowBall and InterceptBall was made. In the tests the goalkeeper remained stable and was able to move back and forth between the arc and the line as transitions between the two behaviours occurred. A result of the test is the conclusion that the FollowBall loses its original thought when put together with InterceptBall. The original thought was that FollowBall should minimize the exposed goal area and thereby increasing the performance in interception. The test however showed that the goalkeeper actually becomes slower because it first has to find its way back to line. Doing so it often turns $90°$, in which angle its width is significantly smaller, thus causing a large area of the goal to be exposed. The situations is illustrated in figure 7.1. Better performance would be expected if the defending line in InterceptBall was

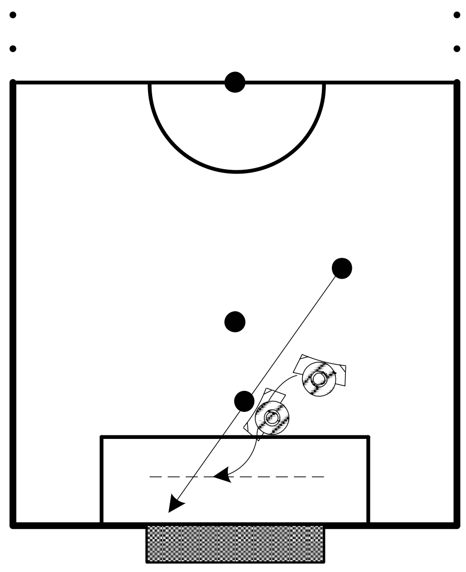


Figure 7.1: A dangerous situation where the goalkeeper changes behaviour from FollowBall to InterceptBall.

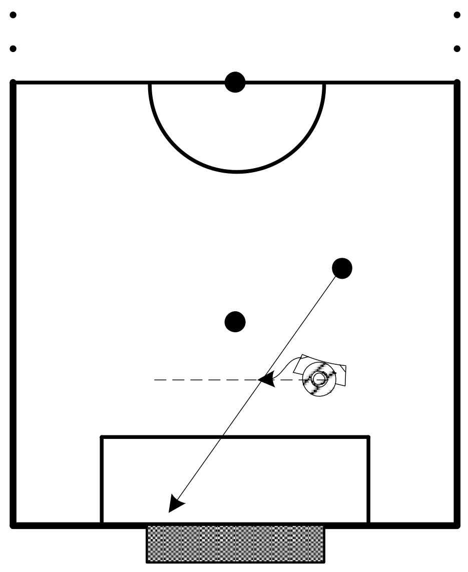


Figure 7.2: Same situation as in figure 7.1 but with a defending line placed through the position where the transition to InterceptBall occurred.

placed through the position of the goalkeeper as the transition occurred. If done so, the purpose of FollowBall would still make sense and ought to increase performance. A similar situation as given in figure 7.1 would then look as showed on figure 7.2. The drawback of this solution is the increasing chance of a situation where the ball ends up between the goalkeeper and the goal.

Furthermore the tests showed that an implementation issue in the switching between the behaviours which should be considered. In the given framework it takes two samples for the `machine` micro-agent to change behaviour after a signal has been given to change behaviour. This currently causes the goalkeeper to stand still in FollowBall for 0.4 sec before the goalkeeper starts moving to the line. The problem could be resolved if the entire state machine for the goalkeeper was implemented in a single control plugin. Video's showing the test where the goalkeeper switches between the two behaviours can be found in [CD, \video].

Having combined the two behaviours FollowBall and InterceptBall the test was expanded with the behaviour SelfLocalize. Two predicates must be satisfied before a transition to SelfLocalize can occur, see figure 3.4. One is that the goalkeeper stands still. The other is that the variable `countLocalize` which is incremented for each run of a control plugin has reached the value of the black board constant `goLocalize`. The value of `goLocalize` hereby determines the minimum time passed between execution of SelfLocalize. In the tests `goLocalize` was set to 25. With this value time between execution of SelfLocalize was approximately 5 sec. Several tests were made. In general the tests showed that the goalkeeper remained stable and corrected its postures according to the self localization algorithm. Best results were achieved when transitions to SelfLocalize was made from FollowBall. Here it was possible to lift up the goalkeeper, move it away and then observe it move back to its original position. A video of this test can be found in [CD, \video\SelfSave.avi].

CHAPTER 8

Conclusion

In this project improvements of the goalkeeper in the SocRob-team at ISR/IST has been designed, implemented and tested. The SocRob-team consists of four robots, one of which is dedicated as goalkeeper. Since the goalkeeper of the team is less developed than the other robots, the problem specification of the project group was:

To improve performance of the goalkeeper in the team at ISR, enabling it to defend the goal better (decreasing the number of goals scored) than the previous implementation.

In order for the project group to get better acquainted with the robot a description of its configuration was made, involving the functional architecture, use of micro-agents and the blackboard used as shared memory between the four robots. Each robot is assigned a role at game start, and to fulfill this role a number of behaviours are executed accordingly. The behaviours of a role and their relations are implemented using a finite state machine.

A finite state machine of the goalkeeper was designed. The state machine contained five states: InterceptBall, FollowBall, Wait4Ball, Go2Area and KickBall. The state machine enables the robot to cover the goal running on an arc in front of it. In the previous implementation the goalkeeper was running on the goal line. A danger zone was defined to determine when the goalkeeper should change between the states FollowBall and InterceptBall. Unfortunately it became obvious during the implementation of the solutions in the goalkeeper that it was not possible to implement the full state machine designed. Instead a modified version of the state machine was designed, and the rest of the project was based on this version of the state machine.

The design of the behaviour FollowBall included a nonlinear control law for the angular velocity and a P-controller for the linear velocity. The design made the defending arc in front of the goal to a equilibrium point and asymptotic stability was achieved. FollowBall was implemented and tested. The test showed that the goalkeeper was able to track the defending arc in front of the goal as intended. If the goalkeeper was placed more than approximately 0.75 m. from the defending arc, it had troubles finding its way back. The troubles are caused by the dynamics which wasn't included in the design of the behaviour. The feature of changing radius according to the position of the ball was implemented in FollowBall but chosen not to be used, because it didn't increase performance. In FollowBall the goalkeeper minimized the exposed goal area, but was unable to intercept shots.

The control algorithm for InterceptBall was based on the same design as FollowBall. Because of a lower maximum linear velocity in InterceptBall, the goalkeeper showed better results when returning from a point away from its equilibrium point. In InterceptBall the goalkeeper was able to predict an interception point on a line when a shot against goal was made. The goalkeeper was able to save shots and it was therefore concluded that InterceptBall worked as intended. It was furthermore concluded that the goalkeepers ability to save these shots was limited by the time needed to retrieve three measurements for the prediction. With sample frequences at 5 Hz. a delay of approximately 0.8 sec. is introduced before the goalkeeper reacts.

From experienced team members the project group learned that the robots lose track of their posture (position and orientation) over time when playing games. The robots are relying on odometry from the two wheels, and this malfunctions in a number of situation, e.g. when the wheels are spinning due to lack of friction. To solve the problem a self localization algorithm is implemented. The algorithm uses an image from the omni-directional catadioptric system to determine the position of the goal posts relative to the goalkeeper. From this information the posture of the goalkeeper is found using the prior knowledge of the position of the posts in the field. The results from the self localization algorithm proves that it is able to determine the posture of the goalkeeper, leaving room for improvement though. The algorithm performs well in the area in front of the goal, but not so well on or near the goal line and further away from the goal. Furthermore the tests of the algorithm were performed in ideal conditions with no other robots around, so the results should be considered according to that.

Since it wasn't possible for the project group to test the implementation in a real life simulation playing against another team, simulations of the state machine were needed to determine optimal values for the size and shape of the danger zone and radius of the defensive arc. A hybrid automaton consisting of the ball and goalkeeper movement was implemented in CheckMate. Three tests were performed. The results revealed that the optimum shape and size of the danger zone is the whole width of the field up til 4.5 m. away from the goal (in a field 10 by 5 m). The optimum radius of the defensive arc is 0.6 m. according to the simulations. The results should be considered with caution though, since the model of the system (the hybrid automata) doesn't contain any dynamics of the goalkeeper nor any of the delays in the software of the robots. The simulations indicated though that the use of the behaviours FollowBall and InterceptBall could be improved.

From a joint test of the three implemented behaviours it was concluded that the goalkeeper switched between the three behaviours according to the state machine and remained stable. It was concluded that FollowBall lost its purpose because the goalkeeper always moved back to the line in front of goal when the ball entered the danger zone. The goalkeeper corrected its position according to the coordinates given by SelfLocalize and was able to recover from a wrong position.

Since the full implemented goalkeeper was not tested in games against other teams, it is not possible for the project group to state weather the new implementation is an improvement of the previous one. From indications of experienced team members, the project group feels confident that this is the case.

In future works on the goalkeeper, the project group suggests the following possible improvements to be considered:

Finish state machine Design and implementation of the last two behaviours in the state machine.

Improve InterceptBall Change the position of the defending line in InterceptBall to the position the goalkeeper is in as it enters InterceptBall.

Remove dangerous ball Introduce a new behaviour which makes the goalkeeper able to remove a dangerous ball close to the goal.

More features in self localization To improve performance of the self localization algorithm, more features should be detected in the image.

More detailed model If the model of the ball and goalkeeper movement used in the simulations is improved, the results would be more liable.

APPENDIX A

Control of Motors

From the control algorithm derived in chapter 4 the desired linear and angular velocity is obtained. What is needed to control the goalkeeper is the reference for the angular velocity to each of the two wheels. The following coherence between the output from the control algorithm and angular velocities for the wheels is known, [Giuseppe Oriolo and Vendittelli, 2002]:

$$u = \frac{r\,(\omega_R + \omega_L)}{2} \tag{A.1}$$

$$\omega = \frac{r\,(\omega_R - \omega_L)}{d} \tag{A.2}$$

Where ω_R and ω_L is the angular velocity for the right and the left wheel. r is the radius of the wheel and d is the distance between the wheels. Isolation of ω_R from equation A.1 yields:

$$\omega_R = \frac{2\,u \;-\; r\,\omega_L}{r} \tag{A.3}$$

Inserting equation A.3 in equation A.2 yields:

$$\omega = r\frac{\left(\left(\frac{2\,u\,-\,r\,\omega_L}{r}\right) - \omega_L\right)}{d} \tag{A.4}$$

$$\Downarrow$$

$$\omega = \frac{2\,u}{d} - \frac{2\,r\,\omega_L}{d} \tag{A.5}$$

$$\Downarrow$$

$$2\,r\,\omega_L = 2\,u \;-\; \omega\,d \tag{A.6}$$

$$\Downarrow$$

$$\omega_L = \frac{2\,u\,-\,\omega\,d}{2\,d} \tag{A.7}$$

ω_R is found through inserting of equation A.7 in equation A.3 which yields:

$$\omega_R = \frac{2\,u \;+\; \omega\,d}{r} \tag{A.8}$$

With equation A.7 and equation A.8 the references for the motors are found.

Appendix B

The Hough Transform

The Hough Transform is a popular method of extracting geometric primitives from images. The simplest version of the algorithm detects lines, but it can be generalized to find more complex features, [Intel, 2001]. The Hough transform works by considering a discrete set of single primitive parameters, e.g. if lines should be detected, then the parameters are ρ and θ. The line equation is

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{B.9}$$

ρ is the distance from the origin to the line, and θ is the angle between the x-axis and the line vector that points from the origin to the line, and is perpendicular to the line, see figure B.1.

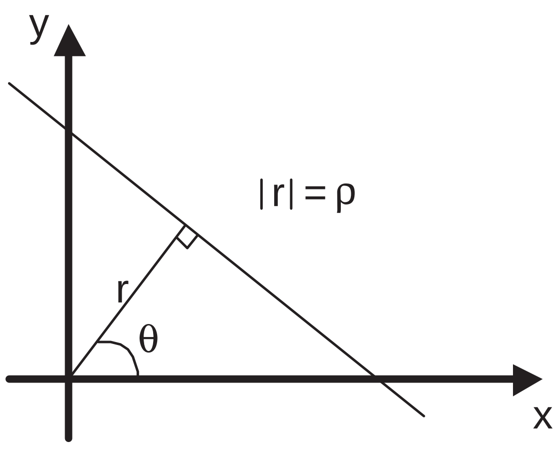
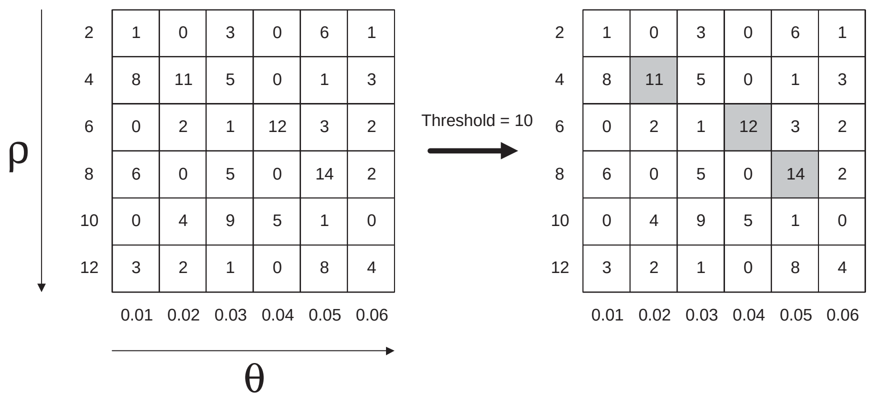


Figure B.1: The line described by equation B.9.

Every pixel in the image in question may belong to many lines described by a set of parameters. So an accumulator is defined, which is an integer array A(ρ, θ) containing only zeros initially. For each non-zero pixel in the image all accumulator elements corresponding to lines that contain the pixel are incremented by 1. After running through all pixels in the image, a threshold is applied to distinguish lines from noise features. That is select all pairs (ρ, θ) for which A(ρ, θ) is greater than the threshold value. All such pairs characterize detected lines.

B.1 Example

The Hough transform is applied to an image with the resolution 2 pixels for ρ and 0.01 rad for θ. The accumulator is shown in figure B.2. The threshold applied to the accumulator



ρ \ θ	0.01	0.02	0.03	0.04	0.05	0.06
2	1	0	3	0	6	1
4	8	11	5	0	1	3
6	0	2	1	12	3	2
8	6	0	5	0	14	2
10	0	4	9	5	1	0
12	3	2	1	0	8	4

ρ \ θ	0.01	0.02	0.03	0.04	0.05	0.06
2	1	0	3	0	6	1
4	8	11	5	0	1	3
6	0	2	1	12	3	2
8	6	0	5	0	14	2
10	0	4	9	5	1	0
12	3	2	1	0	8	4

Figure B.2: The accumulator from the example given. The applied threshold is 10.

is 10, so the detected lines in this example are:

$$4 = x\cos(0.02) + y\sin(0.02)$$
$$6 = x\cos(0.04) + y\sin(0.04)$$
$$8 = x\cos(0.05) + y\sin(0.05)$$

Appendix C

Geometric calculations

C.1 Line and Circle Intersection

In order to filter out the lines that are not passing through the robot, geometric calculations to determine whether the lines are intersecting with a circle are performed. Points $P(x, y)$

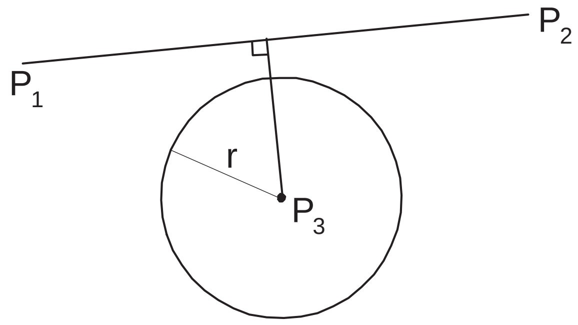


Figure C.1: The line and the circle.

on a line defined by two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ is described by

$$P = P_1 + u(P_2 - P_1) \tag{C.10}$$

or in each coordinate

$$x = x_1 + u(x_2 - x_1) \tag{C.11}$$

$$y = y_1 + u(y_2 - y_1) \tag{C.12}$$

A circle centered at $P_3 = (x_3, y_3)$ with radius r is described by

$$(x - x_3)^2 + (y - y_3)^2 = r^2 \tag{C.13}$$

Substituting the equation of the line into the circle gives a quadratic equation of the form

$$au^2 + bu + c = 0 \tag{C.14}$$

In the equation a,b and c equals:

$$a = (x_2 - x_1)^2 + (y_2 - y_1)^2 \tag{C.15}$$

$$b = 2\left[(x_2 - x_1)(x_1 - x_3) + (y_2 - y_1)(y_1 - y_3)\right] \tag{C.16}$$

$$c = x_3^2 + y_3^2 + x_1^2 + y_1^2 - 2\left[x_3x_1 + y_3y_1\right] - r^2 \tag{C.17}$$

The solutions to this quadratic are described by

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{C.18}$$

The exact behaviour is determined by the determinant $d = b^2 - 4ac$. If d is less than 0 then the line does not intersect the circle. If it equals 0 then the line is a tangent to the circle intersecting it at one point, namely at u = -b/2a. If it is greater than 0 the line intersects the sphere at two points.

Thus, to determine, if a line is passing through the robot equation C.18 is evaluated. If $d > 0$ the line is accepted. Otherwise it is discarded as goal post candidate.

C.2 Robot Position in Goal Coordinates

Knowing the position of the robot in the image P_3, and the two goal posts P_0 and P_1, the position of the robot relative to goal post P_1 is found. The situation is illustrated in figure C.2. Using the cosine relation yields

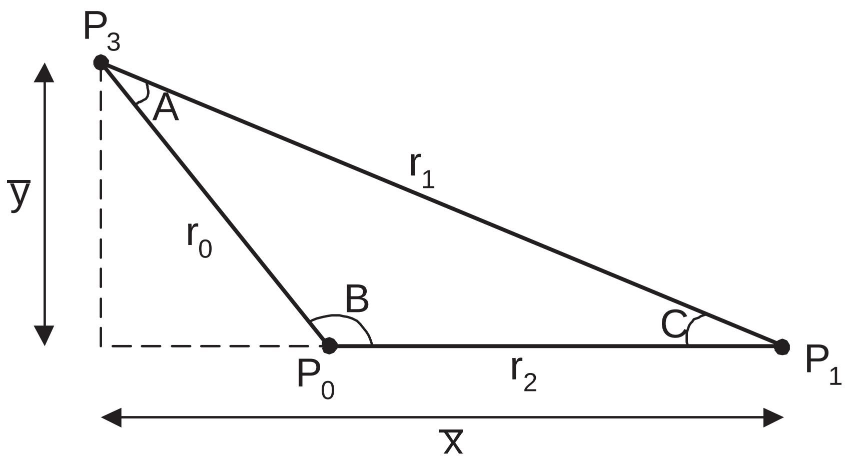


Figure C.2: Determining the position after P_3 is known.

$$r_0^2 = r_2^2 + r_1^2 - 2r_1r_2cos(C) \tag{C.19}$$

$$\Downarrow$$

$$cos(C) = -\frac{r_0^2 - (r_2^2 + r_1^2)}{2r_1r_2} \tag{C.20}$$

$$\Downarrow$$

$$C = acos\left[-\frac{r_0^2 - (r_2^2 + r_1^2)}{2r_1r_2}\right] \tag{C.21}$$

(C.22)

The distance $\bar{y}$ is

$$\begin{aligned}\bar{y} &= sin(C)r_1 && \text{(C.23)}\\ &= r_1 sin\left[acos\left[-\frac{r_0^2-(r_2^2+r_1^2)}{2r_1r_2}\right]\right] && \text{(C.24)}\end{aligned}$$

And the distance $\bar{x}$ is

$$\bar{x} = cos(C)r_1 \tag{C.25}$$

The orientation is found by evaluating the angle of the line P_0P_1, $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$:

$$\phi = \frac{x_0 - x_1}{y_0 - y_1} \tag{C.26}$$

APPENDIX D

CheckMate

In this appendix an introduction to CheckMate [Carnegie Mellon, 2003] is given. When describing the features in CheckMate, the hybrid automaton for the ball (see section 6.2 on page 60) is used as an example. The appendix is divided in six sections. First a general introduction is given, then each of the three basic elements of a CheckMate model is described. Next, the user-defined m-files is described, followed by a short description, how to simulate CheckMate models.

D.1 Introduction

CheckMate is a verification tool for hybrid dynamic systems, that is, dynamic systems with both discrete and continuous state variables. CheckMate is a free MATLAB-based tool for modeling, simulating and investigation of properties of hybrid dynamic systems.

Hybrid systems are modeled using the Simulink graphical user interface (GUI). Parameters and specifications are entered using both the Simulink GUI and user-defined m-files. CheckMate commands are entered in the MATLAB command window. The class of hybrid systems that can be modeled in CheckMate is called threshold-event-driven hybrid systems (TEDHS). A TEDHS consists of three types of subsystems, the switched continuous system, the threshold event generator, and the finite state machine, [Chutinan, 1999]. Figure D.1 shows the blocks available in the CheckMate catalog. Three blocks in figure D.1 corresponds to the three subsystems in a TEDHS. These blocks are the switched continuous system block (SCSB), the polyhedral threshold block (PTHB) and the finite state machine (FSM). These blocks are described further in the sections below.

D.2 Switched Continuous System Block (SCSB)

A switched continuous system block (SCSB), see figure D.3, represents a continuous system whose dynamics for the continuous variable x depends on the value of the discrete input u. The continuous dynamics is selected by the value of u according to the form:

$$\dot{x} = f_u(x)) \tag{D.27}$$

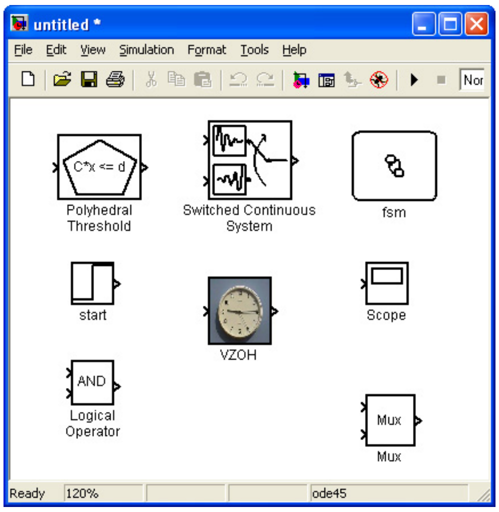
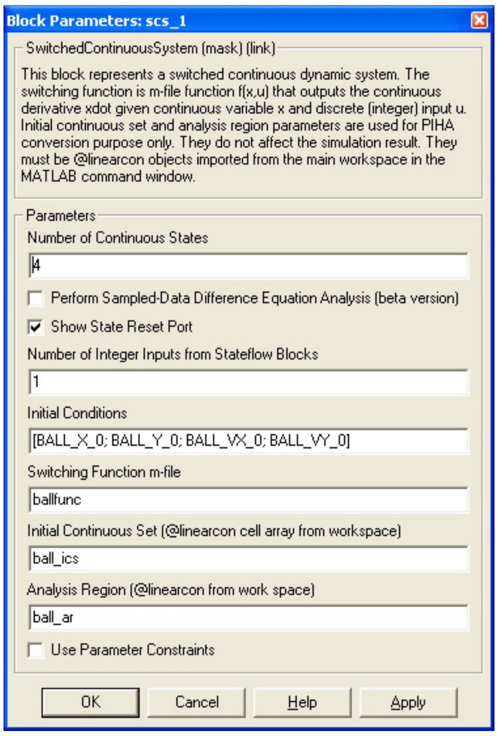


Figure D.1: The available blocks in the CheckMate catalog.



Block Parameters: scs_1

SwitchedContinuousSystem (mask) (link)

This block represents a switched continuous dynamic system. The switching function is m-file function f(x,u) that outputs the continuous derivative xdot given continuous variable x and discrete (integer) input u. Initial continuous set and analysis region parameters are used for PIHA conversion purpose only. They do not affect the simulation result. They must be @linearcon objects imported from the main workspace in the MATLAB command window.

Parameters

Number of Continuous States: 4

[ ] Perform Sampled-Data Difference Equation Analysis (beta version)

[✓] Show State Reset Port

Number of Integer Inputs from Stateflow Blocks: 1

Initial Conditions: [BALL_X_0; BALL_Y_0; BALL_VX_0; BALL_VY_0]

Switching Function m-file: ballfunc

Initial Continuous Set (@linearcon cell array from workspace): ball_ics

Analysis Region (@linearcon from work space): ball_ar

[ ] Use Parameter Constraints

OK Cancel Help Apply

Figure D.2: The properties of the switched continuous system block.

The discrete input u to an SCSB can only be connected to finite state machine blocks (described below). The switching function which returns the state derivatives for each discrete value of u is specified in an m-function file (see section D.5 for more information). Along with the state derivatives, the m-function file also returns the type of the differential equations for each value of u. Three types of ordinary differential equations are available:

’clock’ A differential equation of the form $\dot{x} = c$, where c is a constant vector. In this case, the m-file returns the state derivatives which is the constant c.

’linear’ A differential equation of the form $\dot{x} = Ax + b$, where A and b are constants $n \times n$ matrix and $n \times 1$ vector, respectively. In this case, the m-file returns the matrix A and matrix b instead of the derivaties.

’nonlinear’ A general differential equation $\dot{x} = f(x)$. In this case, the m-file returns the state derivatives given the state x.

Figure D.2 shows the properties of the SCSB. The settings shown are the settings from the ball model (see section 6.2.1 on page 60). Here the number of continuous states, the number of integer inputs and the initial conditions are given. Furthermore the name of the m-file which contains the switching function (i.e. ballfunc.m) is written. The names typed in the boxes for the initial continuous set and the analysis region points to some boundaries in the MATLAB workspace used by CheckMate when running simulations (see D.5).

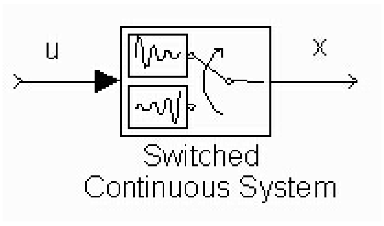


Figure D.3: Switching continuous system block.

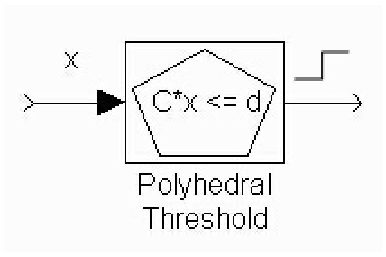


Figure D.4: Polyhedral threshold block.

D.3 Polyhedral Threshold Block (PTHB)

A polyhedral threshold block (PTHB) (figure D.4) represents a polyhedral region $Cx \leqq d$ in the continuous space of the input variable x. The output is a binary signal indicating whether x is inside the region or not, i.e. whether the condition $Cx \leqq d$ is true. The input must be connected to the outputs of SCSB's only. Outputs from multiple SCSB's must be vectorized before feeding into a PTHB. The PTHB's point to boundaries that represent polyhedra that are used to build guard regions. A boundary is a linear constraint object defined in MATLAB. In section D.5, an example is given on defining a boundary.

D.4 Finite State Machine Block (FSMB)

A finite state machine block (figure D.5) represents a finite state transition system driven by the event and data input (in this project only event inputs are used). State transitions are triggered by events, each of which is associated with an event input. An event occurs when there is a change in the value of the corresponding event input, which is binary. The trigger type of each event can be rising edge, falling edge, or either edge. Once an event

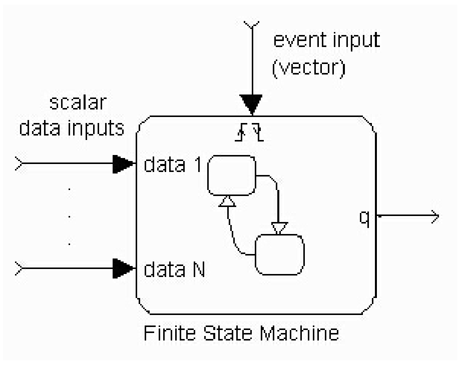


Figure D.5: Finite state machine block.

occurs, the decision whether to take a transition is also determined by conditions on the data inputs. The FSMB is implemented by the regular Stateflow block with the following restrictions:

- No hierachy is allowed in the Stateflow diagram.

- Data inputs must be boolean functions of the PTHB and FSMB outputs only.
- Event inputs must be boolean functions of the PTHB outputs only, i.e. events can only be generated by the continuous trajectory leaving or entering polyhedral regions.
- Only one data output is allowed. Every state in the Stateflow diagram is required to have an entry action that sets the data output to a unique value for that state.
- No action other than the entry action discussed above is allowed in the Stateflow diagram.

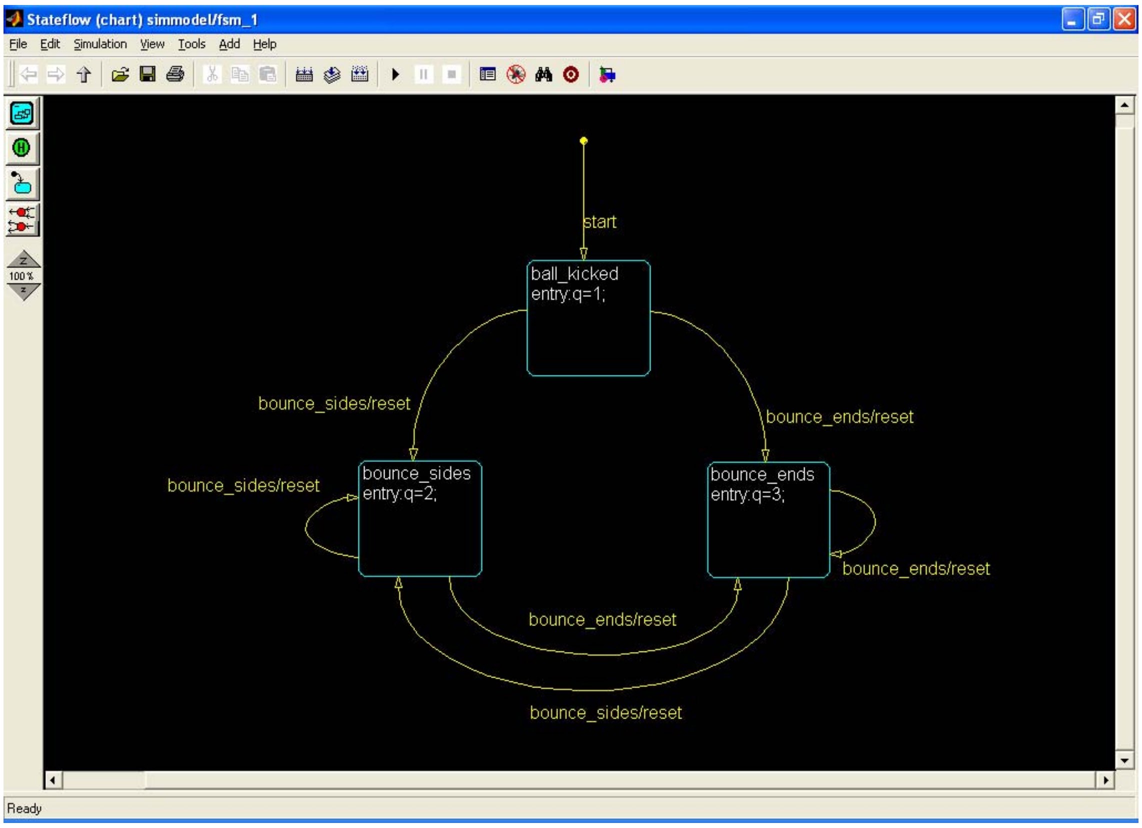
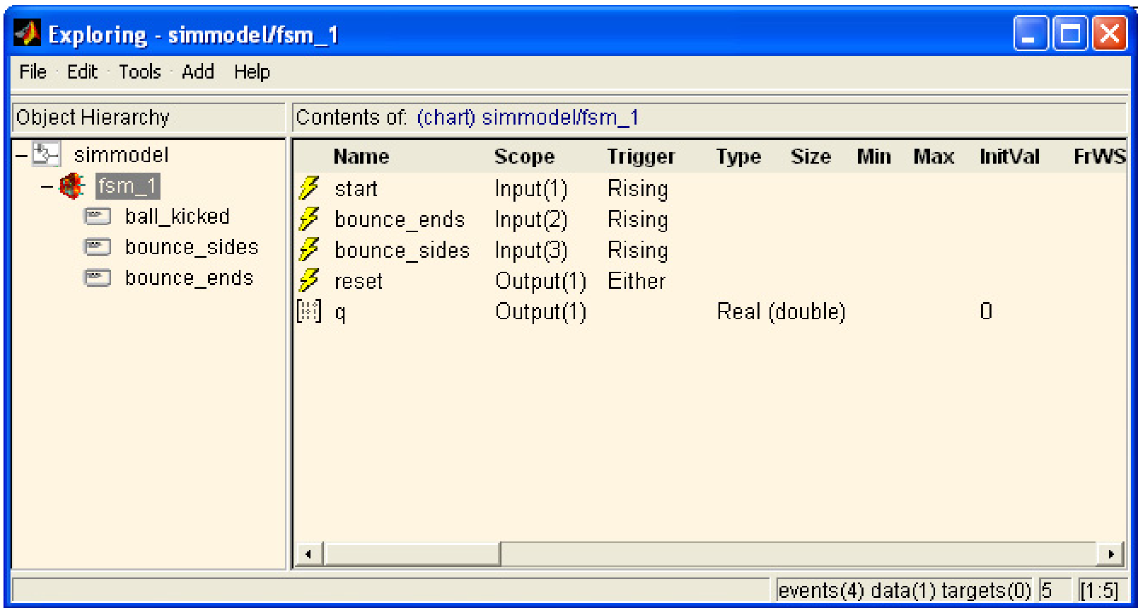


Figure D.6: The finite state machine for the ball model.

When double clicking at the FSMB, a window appears. This is where the discrete state machine is implemented. Figure D.6 shows the corresponding window from the ball model. The attentive reader is noticing that the state machine is not quite identical to that derived in section 6.2. Due to implementation reasons it was modified, so it contains three states: `Ball_kicked`, `Bounce_sides`, `Bounce_ends`. In each block, information to CheckMate is given. The names of each state (i.e. `Bounce_sides`) is only treated as labels, and are not used by CheckMate. Below the labels, a unique number q is allocated to each state. This number is passed to the SCSB through the output of the FSMB, and corresponds to the value obtained through the u-input (see section D.2).


Exploring - simmodel/fsm_1
File Edit Tools Add Help
Object Hierarchy
Contents of: (chart) simmodel/fsm_1
simmodel
fsm_1
ball_kicked
bounce_sides
bounce_ends
events(4) data(1) targets(0) 5 [1:5]

Name	Scope	Trigger	Type	Size	Min	Max	InitVal	FrWS
start	Input(1)	Rising						
bounce_ends	Input(2)	Rising						
bounce_sides	Input(3)	Rising						
reset	Output(1)	Either						
q	Output(1)		Real (double)				0	

Figure D.7: The explore window.

The FSMB shifts between its states according to the event inputs. In the ball example, the event input is a vector with three binary elements. According to the changes in these elements transitions is made. In the explore window (see figure D.7) the three vector elements is evaluated according to the statements in the first three lines (with lightnings). For example a transition from the state `Ball_kicked` to `Bounce_sides` is made, when the second element in the input vector goes high. The line named `reset` makes it possible to use a reset condition in the state machine for the ball. The reset is activated in every transition and a signal is given to the SCSB through the reset output port of the FSMB. The bottom line in the explorer window sets up the output q of the FSMB. This output tells the SCSB the current state (i.e. value 2 for the state `Bounce_sides`).

D.5 Userdefined Files

setup.m This file contains the definitions of the boundaries that the PTHB's point to (i.e. regions used to create guards for the system) and boundaries used for the initial continuous set as well as the analysis region. For example, the boundary that represents the left line in figure 6.2 B) is defined as:

```
left_line=linearcon([],[], [1 0 0 0 0 0], [-2.5]);
```

Also the setup.m file contains the system specification. This expression is used by the `explore` (read more about `explore` in section D.6) routines when analyzing the system.

constants.m This file contains constants used in the simulation (i.e. dimensions of the field, simulation time etc.)

ballfunc.m This file contains the description of the continuous-time dynamics. The function basically returns the right-hand-side of the differential equation $\dot{x} = f(x)$. The dynamics change depending on the discrete state that the system is in. For example, according to the ballfunc.m file, when the system is in discrete location 2, the ball has reached one of the sides and has bounced. The following peace of code is chosen from ballfunc.m.:

```
function [sys,type,reset]=ballfunc(x,u)

global_ball_x = x(1);
global_ball_y = x(2);
global_ball_v_x = x(3);
global_ball_v_y = x(4);
GK2ball_x = x(5);
GK2ball_y = x(6);

switch u,
case 1,
.
.
case 2,
    ball_x_dot = global_ball_v_x;
    ball_y_dot = global_ball_v_y;
    ball_v_x_dot = 0;
    ball_v_y_dot = 0;
    GK2ball_x_dot = (global_GK_v_x - global_ball_v_x);
    GK2ball_y_dot = (global_GK_v_y - global_ball_v_y);

    sys = [ball_x_dot; ball_y_dot; ball_v_x_dot; ball_v_y_dot; GK2ball_x_dot; GK2ball_y_dot];
    reset.A = [1 0 0 0 0 0; 0 1 0 0 0 0; 0 0 -1 0 0 0; 0 0 0 1 0 0; 0 0 0 0 1 0; 0 0 0 0 0 1];
    reset.B = zeros(6,1);

case 3,
.
.
.
```

The first line sets up the function `ballfunc`. The function receives two arguments, x and u. x is the current state vector and u is the current location received from the FSMB. The state vector is assigned to different variables in the 6 lines below the function declaration. u is used in the switch-case sentence, and in the example shown above only the tasks for location 2 is shown. The `ballfunc` returns three arguments: `sys, type, reset`. In the first 6 lines of case 2, new values is written to the state vector, and this new vector is returned to sys. The two last lines in case 2 assigns the reset condition, when entering discrete location 2.

D.6 Simulating the Model

CheckMate models can be simulated like any other Simulink model. When the play button is hit in the Simulink window, the system is simulated using the initial conditions specified in the SCSB initial conditions field inside the SCSB dialog. Furthermore a set of commands is available from the MATLAB command prompt. Two main commands, `explore` and `verify`, are available from the MATLAB command prompt. Only `explore` is used in this rapport. Therefore a short description of this command is given below.

D.6.1 Explore

The `explore` tool is invoked from the MATLAB command window by typing `explore` at the prompt. In this form, the tool simulates the point entered as the initial condition in the SCSB parameters for the time specified in the Simulink simulation parameters. After the simulation is complete, the `explore` routine checks the simulated trajectory against the specification and informs the user whether or not the trajectory violates the specification. An example of the exploring is showed in the following:

```
For initial condition x = [0 5 1 1]: The system never enters the state "avoid"
/ specification satisfied
>>
```

This tells the user the initial state of the system, and that the specification was satisfied.
Also, the user can specify a set of initial condition points. At the command line type:

```
a=[0 5 0 1; 0 5 1 0];
explore -points a
```

This will cause the analysis to `explore` the trajectories starting from the points $x = [0\,5\,0\,1]$ and $x = [0\,5\,1\,0]$. In the rapport the commands above was used. The matrix a contained the different shots.

Bibliography

[Carnegie Mellon, 2003] Carnegie Mellon University, http://www.ece.cmu.edu/ webk/checkmate/. *CheckMate Official Site*, January 2003.

[Chutinan, 1999] Alongkrit Chutinan. *Hybrid System Verification Using Discrete Model Approximations*. PhD thesis, Carnegie Mellon University, May 1999.

[Giuseppe Oriolo and Vendittelli, 2002] Alessandro De Luca Giuseppe Oriolo and Marilena Vendittelli. *WMR Control via Dynamic Feedback Linearization: Design, Implementation and Experimental Validation*. *IEEE Transactions on Control Systems Technology*, 2002.

[Indiveri, 2001] Giovanni Indiveri. *On the Motion Control of a Nonholonomic Soccer Playing Robot*. August 2001.

[Indiveri, 2002] Giovanni Indiveri. *An Introduction to Wheeled Mobile Robot Kinematics and Dynamics*. Slides for lecture given at RoboCamp, Paderborn (Germany), April 2002.

[Intel, 2001] Intel Corporation, http://developer.intel.com. *Open Source Computer Vision Library - Reference Manual*, 2001.

[Intel, 2003] Intel Corporation, http://www.intel.com/research/mrl/research/opencv/index.htm. *OpenCV main page*, Jan 2003.

[Khalil, 2002] Hassan K. Khalil. *Nonlinear Systems*. Prentice-Hall, 2002.

[Lima, 2002a] Pedro Lima. *Current Status of the Socrob Project*. Technical report, ISR, Instituto Superior Tecnico, Lisboa, Portugal, 2002.

[Lima, 2002b] Pedro Lima. *Homepage of ISocRob project*. http://socrob.isr.ist.utl.pt, October 2002.

[Nunes, 2002] Pedro Marcelino; Pedro Nunes. *Improving Object Localization through Sensor Fusion applied to the RoboCup Simulation League*. July 2002.

[Robocup.org, 2002] Robocup.org. *Robocup Official Site*. The RoboCup Federation, http://www.robocup.org/02.html, October 2002.

[van der Schaft; Hans Schumacher, 1999] Arjan van der Schaft; Hans Schumacher. *An Introduction to Hybrid Dynamical Systems*. Springer, 1999.

[Yahoo, 2000] *Open Source Computer Vision Library Community*. http://groups.yahoo.com/groups/opencv, 2000-.