Technische Universität Berlin

Reactive Full-Body Behaviour for Humanoid Robot
- ball blocking behaviour for robot field player

Bachelor Thesis

Fakultät IV -Eletrotechnik und Informatik

Fachgebiet Agententechnologien in betrieblichen Anwendungen und der Telekommunikation (AOT)

Prof. Dr.-Ing. habil. Sahin Albayrak

Fakultät IV Elektrotechnik und Informatik

Technische Universität Berlin

vorgelegt von

Johannes Schneider

Betreuer: Dr. Yuan Xu

Johannes Schneider
Matrikelnummer: 350423
Waldeyerstraße 9
10247 Berlin

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

Summary

This work approaches a full body reaction on the example of a ball blocking behaviour for a robot in SPL league. To find an interception between possible blocking locations the ball velocity will be calculated. The created motions are able to block along blocking lines and create blocking locations on these lines. A block behaviour which utilizes this feature was created.

Zusammenfassung

Diese Arbeit befasst sich mit mit einer vollkörper Bewegung am Besispiel eines NAO roboters für die SPL League. Dabei wird versucht den Abfangpunkt des Balls mit Hilfe der Ball Geschwindigkeit zu bestimmen. Di e erstellten Block Bewegungen blocken entlang von block Linien und erstellen Abfangpunkte auf diesen. Dieses Verhalten wurde in das Verhalten des Roboters eingefügt.

Contents

Chapter 1

Introduction

robot:
"A machine capable of carrying out a complex series of actions automatically, especially one programmable by a computer."
oxford dictionary [5]

"By 2050, a team of fully autonomous humanoid robot soccer players shall win a soccer game, played according to official rules of FIFA, against the winner of the most recent FIFA World Cup."
RoboCup grand challenge, 1997, [6]

1.1 Motivation

Since 1954 when the first automated robot was created, robots are used for dangerous or repetitive actions.
But their usefulness is not limited to the industry and production. The field of application grows and, depending on the field of work, robots have different forms and sizes. The number of robots used in the industry is increasing. But the use is not limited to industrial robots. More and more robots are also used in normal households. With the transition of robots out of factories the requirements rose, namely the number of environmental conditions and outside influence. The robot must be able to react fast to changes in its environment, the perception and the evaluation must be quick and with a

low error rate. To ensure this and to minimize interference factors, clearly organized and defined surroundings are preferred. The restricted rule-set of a soccer game, limiting the allowed moves, the obstacles on the field and the clear defined field itself is a good example for this. The soccer world cup for robots is the RoboCup and this work will be centred around the standard platform league (SPL) of the RoboCup. The RoboCup itself is an soccer orientated game of two teams playing against each other. In the SPL the players of both teams are NAO robots from Aldebaran-Robotics and act autonomously during the game. The goal is, to score as many goals with the ball.

All the outside variables, the robots itself, the field, goals, ball are clearly defined in size, shape and colour.

In the SPL all modifications are only in the code implementation, as in the name suggested all code runs on a standardized platform, the NAO robot.

In the RoboCup SPL final of 2016 the B-human [3] from Bremen and UT Austin Villa [2] has shown great soccer. For example the block movements of this two teams where exemplary and effective. Both teams were able to deflect the ball with normal field players.[1] With these they were able to reduce the amount of shots reaching the goal area and disturb the enemy play itself. Also with these the players could increase the ball possession time.

The SPL team of the TU-Berlin is called DAInamite. The DAInamite implementation is lacking a blocking behaviour, except for the keeper of course and this could greatly increase their performance on the field. This block movement is a way to get into possession of the ball by interrupting actively the path of the ball or at least interrupt the path of the ball that it poses no danger to score. Once a behaviour in the restricted and clearly defined environment is created and working, the results and the behaviour itself can be moved out of the confined game. New extensions, scenarios can be added to adapt better this the real life application. Blocking the ball might not have many real life applications outside of playing soccer, but once the behaviour is established, it can be conveyed to any reactive full body behaviour.

1.2 Approach and Goals

The goal of this work is the creation of a working block motion and the implementation into the play behaviour of the robot. With the block the robot will be able to stop the ball from various directions, interrupting the ball movement and stop the ball. Thanks to this behaviour the robots and team will be able to defend more successfully and increase the ball possession itself. This addition will ultimately help scoring and winning.

1.2.1 Block Motion

There is no block motion for field players at the moment. The motions of the keeper uses in great extend the arms to increase the block, a practice not desired for the field player. The new block motion need to be created and they need to be fast and stable. Also it is necessary to create different block movements for different situation for example a far reaching motion to block a distant ball.

1.2.2 Ball Model

The Ball Model consists of all information and calculations concerning the ball itself, for example the position of the ball relative to the robot. The Python class is the ball tracker class. The ball tracker class calculates with an implementation of a Kalman filter the position of the ball. A Kalman filter is "a technique for filtering and prediction in linear Gaussian systems [...]. The Kalman filter implements belief computation for continuous states."[7]

- Vision: the ball needs to be detected and the trajectory needs to be calculate
- the error in detecting the ball position stationary is sufficient, but needs an error handling and precision in movement
- the error handling depends on the distance to the ball, by inaccuracies in height and in the angle

1.2.3 Ball Blocking Behaviour

The goal of the ball behaviour is a smooth transition from the original state into the block behaviour. This only occurs if the ball is in reach and it is realized in the ball model. Then the robot executes the ball block via the ball motion.

- bring the block motion and ball model together

1.3 Structure of the Thesis

This thesis is structured as follows. In Chapter 2, I discuss essential background related to the thesis topic. This will include detailed information about the NAO robot used in the SPL and which is the model in the SimSpark simulation environment. The SimSpark will be discussed also as well basis for Motions for the NAO robot and Kalman filter

used for the Perception.This chapter also represents a detailed analysis of the problem that will be addressed. In particular, the problems of the motion and ball model. In Chapter 3, my planned solution is presented. This solution covers the block motion itself, the ball model and how it comes together in the ball blocking behaviour. Also I will discuss how the quality of the new component is tested in SimSpark. Chapter 4 explains the Implementation for the SimSpark Simulation. Chapter 5 evaluates our solution basing on our specified goals and I conclude and revisit the points above under the finished implementation.

Chapter 2

Background

In the following chapter are the details about the environment, robot and functions I used in this work. It consists of the simulation environment and all necessary components. The robot has a motion class for movement and operation of all the joints. The perception class handles the sensor data and a Kalman filter is used to refine this data.

2.1 Nao robot

This work is using the NAO robot by Aldebaran-Robotics. The NAO robot is a humanoid robot. It has a small height of 57.4 cm, different sensors, like camera and ultrasonic, and 25 joints. [4] The NAO robot supports different languages like C++ and Python, which is used in this work. The robot has a ATOM Z530 1.6 GHz CPU, 1 GB RAM, 2 GB Flash memory and a 8 GB Micro SDHC. The sensors of the NAO robot include force sensitive resistors, inertial units, sonars, joint position sensors and contact and tactile sensors. Also the robot has microphones, infra-red and two cameras. Both cameras are mounted in the middle of the head, on top and bottom. They cover both a 47.64° angle high and 60.97° angle wide. The Top Camera goes forward (with a 1.2° offset downwards) and the bottom camera is located with a 39.7° offset downwards. The bottom camera is in this position for a better visual feedback of the area in front of the feed while walking, mostly for the ball.

2.2 Motion

The NAO robot owns 25 independently moveable unique joints. The hip joint is only one motor though listed as two separated joints. A motion consists out of the names of

the joints which will be changed, the new angle of the joint and the time when the angle should be reached. Through angle interpolation can the movement be calculated at any given moment.

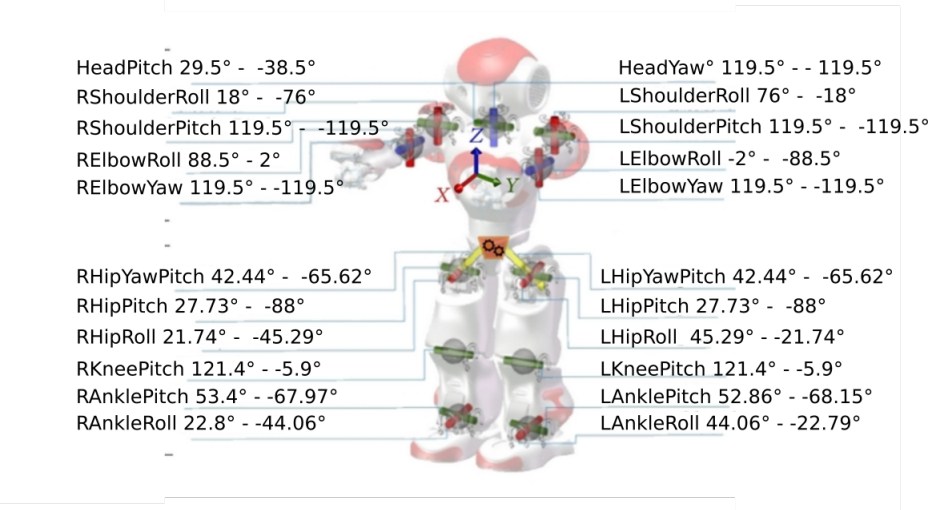

Figure 2.1: fig: joints and maximum angles of the NAO

The existing block implementation are for the goal keeper. These motions are mainly restricted to this player. Also the ball model is only reliable in this stationary position.

2.2.1 Key Frames

A motion consists of the names of the joints which will be manipulated, the joint angle, either the total value the angle assumes or a relative value and the time when the angle should be reached.

$$j_{names} = (j_{HeadPtch}, ..., j_{LAnklePitch})$$
$$k_{keys} = ((k_{HeadPitch_1}, ..., k_{HeadPitch_n}), ..., ((k_{HeadPitch_1}, ..., k_{HeadPitch_l}),)$$
$$t_{times} = ((t_{k_{HeadPitch_1}}, t_{k_{HeadPitchn}}), ..., (t_{k_{HeadPitch_1}}, ..., t_{k_{HeadPitch_l}})$$
$$motion = (j_{name}, k_{keys}, t_{times})$$

Each of the keys needs a time. Each joint can only move in one direction of possible three. (yaw, pitch, roll) To execute the motion an interpolation of the angles is necessary. This is done in the AngleInterpolation function in the motion class.

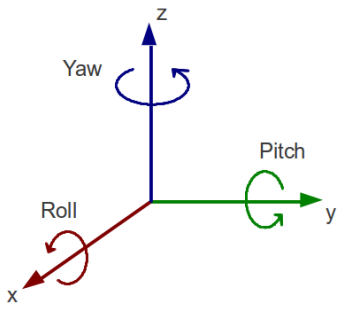

Figure 2.2: Roll Pitch Yaw

2.2.2 Choreographe

Choregraphe is a multi-platform desktop application, which is able to create animations. Also Choreographe is able to create behaviours and dialogues, tests, control and write scripts with Python or create applications. The possibility of creating animations was used here to create the blocking motions. The motions where exported and converted to Python functions.

2.3 SimSpark

The simulation environment is SimSpark. SimSpark is an open-source generic simulation system. It supports a multi agent simulations up to 22 agents (11 vs 11) and is used since 2004. [8] It is based on the Spark, a generic simulation environment, SimSpark features "realistic motor, heterogeneous robots and agent proxies" [8] The SimSpark simulation provides all the sensor input. These inputs are processed by the same implementations as on the robot in the real world. Also all the orders and commands by the code are realized by the simulation. "Some of the sensors delivers information from physics engine,such as joint position, gyro, accelerometer, and force resistance. Furthermore,lines can be sensed by virtual vision. Additionally, a more realistic camera which delivers images rendered via OpenGL hardware accelerated offscreen buffers is implemented." [8]

Test Scenario

Simspark: This is a simulation environment. It is created by DAInamite for extended testing of new components before it is transferred to the real NAO robots.

2.4 Vision of the NAO robot

The robot is able to recognize its environment through a number of sensors, but the perception of the ball is done with the two cameras in the head. The ball attributes can be found in the SPL rules. Through these attributes like radius, colour definition and pattern, the robot can distinguish the pattern from the other robots and other obstacles. Through the strict border structure of the play field and a number of assumptions the number of white and black patterns next to each other is minimal.Black and white patterns in the surroundings, outside the field can be ignored.

The interception point between robot and ball can be calculated by knowing the ball position. The speed of the ball is calculated though the time of perception of the ball. This can be used to calculate when the block needs to happen.

There are moments when the robot should not block. The robot should only register and react when the ball is passing by the robot. Also he should not block balls from himself or his team-mates, only when the ball is going into the robots own goal direction.

2.4.1 Kalman Filter

In order to deal with the uncertainty of perception and optimize the localization of the ball a Kalman filter is used. The Kalman filter is a method of prediction in a linear Gaussian system. The filter takes a series of measurements over a period of time as input and is able to estimate an accurate prediction about future measurements. This is done by using a joint probability distribution over each of the measurements. in this way the noise and inaccuracies of the data are flatten and reduced.
Assumed a moment in time t, the following moment in time is $t+1$, the prediction is the mean μ_t, the probability of the moment is defined by $p(x_t \mid u_t, x_{t-1})$, u_t is the control data, the informations got by sensors, $t-1$ is the moment before the moment t.

The Kalman filter algorithm for linear Gaussian state transitions:[7]

1: input: $\mu_{t-1}, \Sigma_{t-1}, \mu_t, z_t$

2: $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$

3: $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$

4: $K_t = \bar{\Sigma}_t C_t^T (C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$

6: $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$

7: output: $\mu_t\Sigma_t$

z_t is the the Gaussian noise vector,Σ_{t-1} is the covariant , A_t is a matrix in dimension of the vector of the state x_t, B_t is the matrix in dimension of the control vector u_t, C_t is a matrix in the dimension of the vector z_t,

2.5 Ball Blocking Behaviour

The game state transition system is a finite state machine.

- Include Ball Model and Block into behaviour:
  Determinate the best way to interrupt the actual behaviour and change into the ball block behaviour if a ball block is needed/recommended.
  The change into the ball behaviour is triggered by a ball moving into range with an intercept able path.
  Afterwards the situation needs to be evaluated and the best behaviour must be activated.

- Test Behaviour in Simulation: implementation in Simspark and testing in Simspark

2.6 Problem Summary

As in the sections before explained, the robot uses the implementation of the RoboCup Team of the DAI-Labor, DAInamite. These Implementation will be extended by the ball block. For the motion, ball model and the behaviour itself will created an own python class. The motion class will calculate the best block depending on the situation. The criteria will be the stability of the block vs. time and extend needed to intercept the ball.

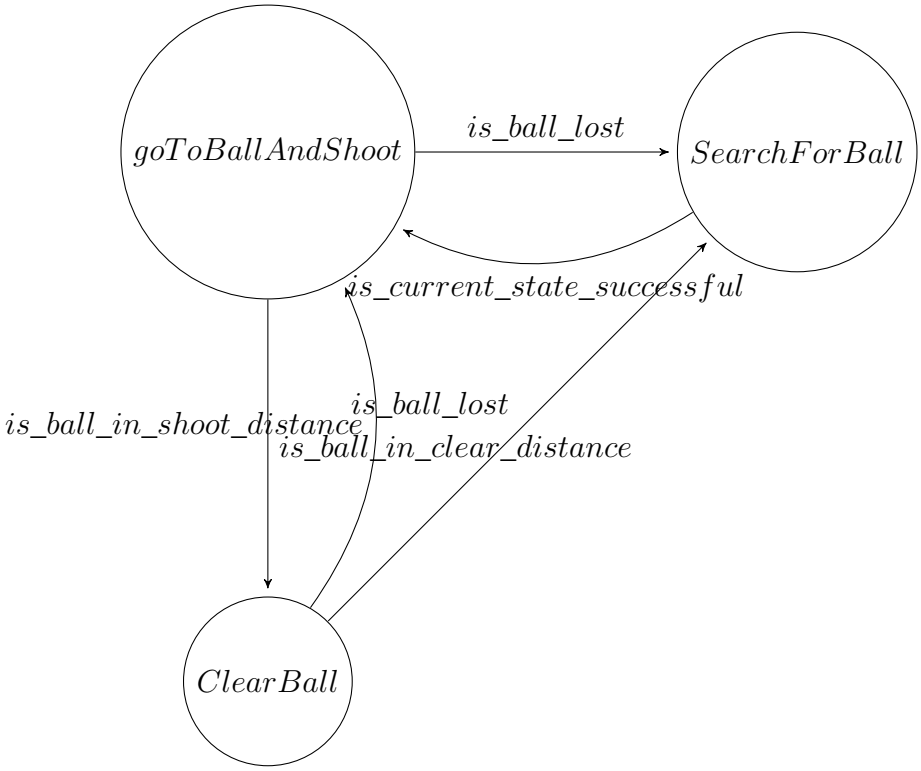

Figure 2.3: The behaviour of the striker

Chapter 3

Solution

The solution consists of three steps. First the robot needs to be reactive, perceive the environment and act by this perception. This will be realized in the ball model. The action needs to work towards the goal, in this case block the ball. The blocking is realized with the block motion. Both components will come together in the integration into the ball behaviour.

3.1 Block Motion

The key frames for the block motion can be created in Choreographe. The keys and the names of the motion are exported in .xml format in a .xap file. These can be imported in Python and then converted in the needed form.

The motions will assume the left leg as the leg the robot is standing on and execute a block to the left side (from the robots point of view). For blocks to the right side the keys will be converted. The goal of the new implementation is to create a standard block motion for a basic use case, for example a simple block of a straight ball which would pass by the player..

- Block in Simulation: implementation in Simspark.
  Testing the quality of motion, in this case the block motion, will be done by a series of movements. These will determine how stable the motion ends. Falling and loosing balance will give negative test result.
- Test block in Simulation: testing in Simspark

Different position will need different motion. The further the leg is extended, the longer it takes and the motion becomes less stable. Blocking the ball should always use

the closest block. This reduces the time needed to get into the stance and the chance of interaction with obstacles (goal post and other robots) Also there might be the need of an "emergency" block when the ball comes fast, there is no time to prepare or it is to late for a proper block.

The robot should do a block motion which includes the following points:

- rules: the motion needs to follow the rules and spirit of the SPL Rule book. This means especially that no hands may used to block the ball, unless it is the keeper in the penalty area.
- wide range: the block motion should be able to reach and block as many positions as possible in the action radius of the robot.
- success: the motion should be able to block the ball, interrupt its movement and reflect its path to a better position then the ball would move to without the block
- stability: the robot should not fall over by the motion itself, even better, the motion should be stable enough to survive collision with obstacles(ball, robot or goal).
- speed: the robot should only spend as less time as possible blocking and safely return to its normal behaviour.
- stress: the stress to the joints need to be minimal. Extremely fast motions as well as high weight on single joints should be prevented

The ball might move through the action radius of the robot without crossing the x-axis in the action radius of the robot. To increase the chance of blocking these balls, additional motions for blocking along the $g(x) = x$ line and the x axis are created. This should increase the possibilities of blocking the ball.

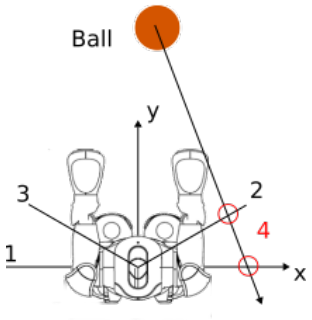

Figure 3.1: Block lines, 1: g_0, 2: g_1, 3: g_2, 4: Interception Location

The block lines above are defined as following, with the local robot coordinates (0, 0) in the middle of the robot. From the ball position the route of the ball defined by its velocity will intercept with the block lines.

$$g_0(x) = 0$$
$$g_1(x) = x$$
$$g_2(x) = -x$$

The maximal length of these block lines are defined as the possible extension of the leg without loosing stability.

3.1.1 Testing the block motion

Each of the block motions will be tested in the SimSpark simulation. For that the Agent class will be customized. Testing the quality of the block motion will be done by a series of movements. These will determine how stable the motion ends. Falling and loosing balance will give negative test result. The motion will consist of 10 iterations of the block and return to the stand in short sequence.

3.2 Ball Model

Each change of a joint changes the posture of the NAO robot and with that the perception of the ball. The ball model needs the ball position from the perception. This is received as a ball hypothesis.

$$ball\_hypothesis = (position(x, y), velocity(vel\_x), vel\_y), error, ball\_last\_seen$$

The ball velocity given by the ball hypothesis has a big error margin, an own implementation is needed. Given a good ball position the ball velocity can simply be calculated.

3.2.1 Interception Point

The interception point is where the balls route will intercept with the positions the robot is able to block. The interception point class will have the coordinate where the ball meets the possible block lines. The interception point will be added to the agent. Each time the perception is updated, so will the interception point.

The interception point is calculated from the ball position and the ball. The direction of the ball movement will be one of the following.

1. the velocity be (0,0)
2. the velocity direction is parallel to the ground interception line
3. the velocity is too little, it will become (0,0) (by damping)
4. the interception point is out of range
5. blocking possible

3.3 Ball Blocking Behaviour

The goal is to include the ball model and ball blocking into the existing behaviour. The ball behaviour consists of three states, between which the robot may change.

- goalie: only one goalie is allowed per team, it will assume this role if he is given the order. It already keeps the ball off the goal and has a blocking, parry behaviour.
- striker: the striker is the player which is closest to ball and is supposed to interact with the ball.
- supporter: if the player is neither goalie nor striker.

The ball block will be only added to the striker behaviour, because it is supposed to be the only one interacting with the ball. The goalie behaviour will be used as an example. The condition function will be using the interception point from the agent and decide by this calculation if it will block or continue with the originally go to ball and shoot behaviour. The block behaviour consists of a distance and interception function which calculates the condition for changing the state. The block will be able to transit from the goToBallAndShoot state into the block state. The State goToBallAndShoot needs to see the ball (it will change into the SearchForBall state if the robot looses the ball) and will already try to shoot the ball if the ball is in range. Otherwise it will make the robot walk towards the perceived ball and face the ball. This is a good position for blocking the ball. The ball block itself will only execute the block motion and then return into a neutral position, from where the robot can continue with ease.Then the robot returns into the goToBallAndShoot state automatically. There are no other transitions in the striker behaviour needed:

- If the ball is lost, the robot can not block.
- If the robot shoots, the robot does not need to block the ball.

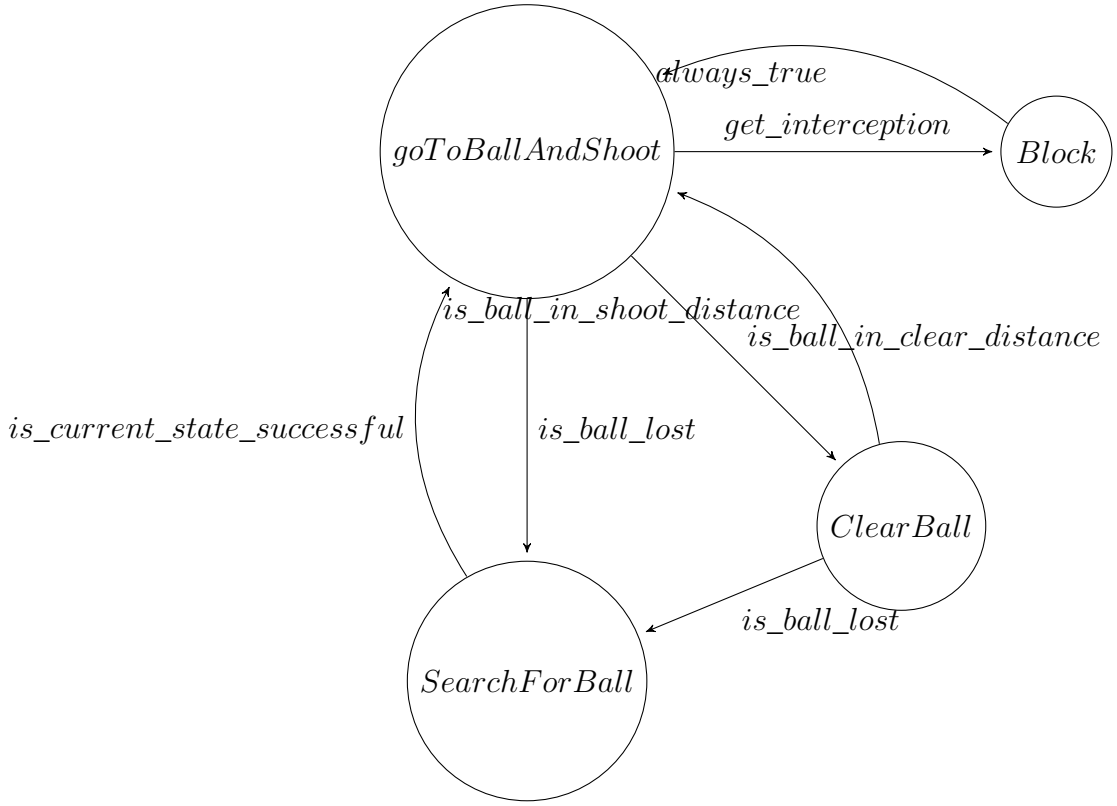

Figure 3.2: The striker behaviour with block extension

Determinate the best way to interrupt the actual behaviour and change into the ball block behaviour if a ball block is needed/recommended. The change into the ball behaviour is triggered by a ball moving into range with a possible interception location. Afterwards the ball the situation needs to evaluated and the best behaviour must be activated.

Chapter 4

Implementation

The implementation was done in Python and uses the DAInamite code from GitHub. The code consists of the possibility in the agent (spark_agent) to add the block_control, which uses the new interception_point class.
To ensure this, a block control, an interception point and a ball blocking behaviour is created. The block_control is feed with the ball_hypothesis from the Kalman filter of perception of the agent. The block control also uses a Kalman filter to clear the ball position given by the ball hypothesis. The block control updates the interception point with the ball position and ball velocity from the Kalman filter.

If the interception point has possible interception locations, the interception time is calculated, when the block should happen. Also the best block motion out of the created block motions is chosen. The chosen motion then is executed with the angle interpolation function of the motion class of the agent.

Also there is a emergency block motion, a wide spreading of the legs when the ball is coming close to the robot and no other block motion of the robot is possible.

In the following I will use different coordinate systems. The global robot coordinate system is centred on the starting point as (0 , 0) and every movement is tracked inside. The local ball position is relative to this robot position. The global ball position is the calculated position of the ball in the global robot coordinate.

All calculations about the position are repeated in every update of the block control.

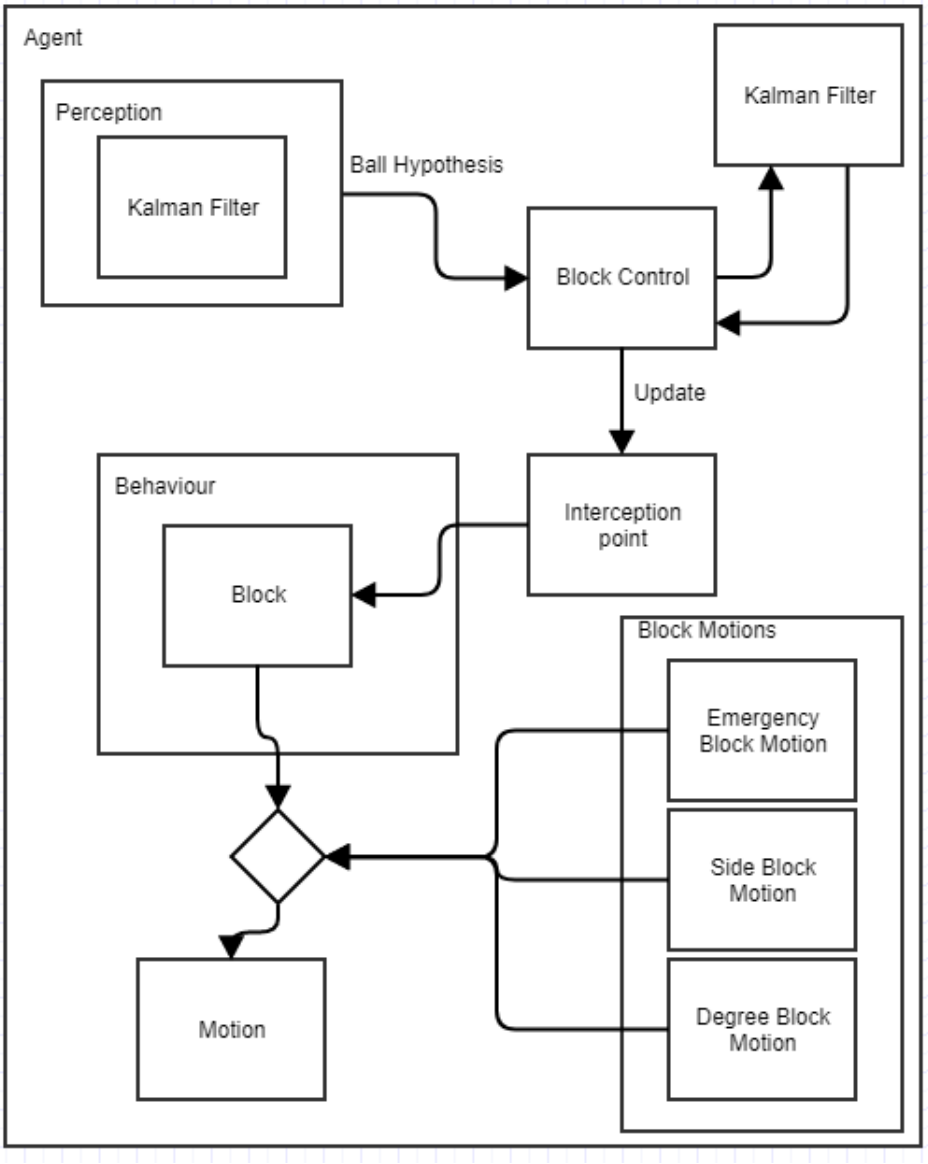

Figure 4.1: Structure of the implementation

4.1 Implementing Block Control

The block control is a hub, where the information about the robot, position, rotation, joint angles and the ball position and velocity come together, before the ball is evaluated in the interception point class. The ball position is relative to the NAO robot in local coordinates. These are transformed into the global position of the robot. The block control is updated every time sense is called. Sense updates the perception including all the vision, time, odometry and posture in every iteration. Following data is updated in block control:

- best ball hypothesis in field frame
- time (from perception)
- pose of the robot (x, y, rotation angle)

The block control calculated the velocity and creates, if possible, the interception point. The velocity is tracked via a Kalman filter. This is an additional filter for tracking the velocity while the first Kalman filter tracks the multiple ball hypothesis. The Kalman filter is used to reduce the noise and minimize the impact of false positives of the ball position further.

4.2 Ball Hypothesis in Field Frame

The ball hypothesis is the hypothesis of the Kalman Filter. Only the best one is used. it consists of the ball position, ball velocities, the error and the time when the ball was last seen. The position from the best ball hypothesis is translated in the field coordinates.

$$ball\_hypothesis = (x, y, vel_x, vel_y, error, ball\_last\_seen)$$

4.2.1 Ball Velocity

The ball velocity included in the ball hypothesis does not bring good result in its own error margin.

The most important part of the whole ball model is the direction and the velocity of the ball. With the velocity it is possible to determine the point of interception and the time. Sadly the velocity given by the best_ball_hypothesis_in_field_frame has a big area of error while the observing robot is moving.

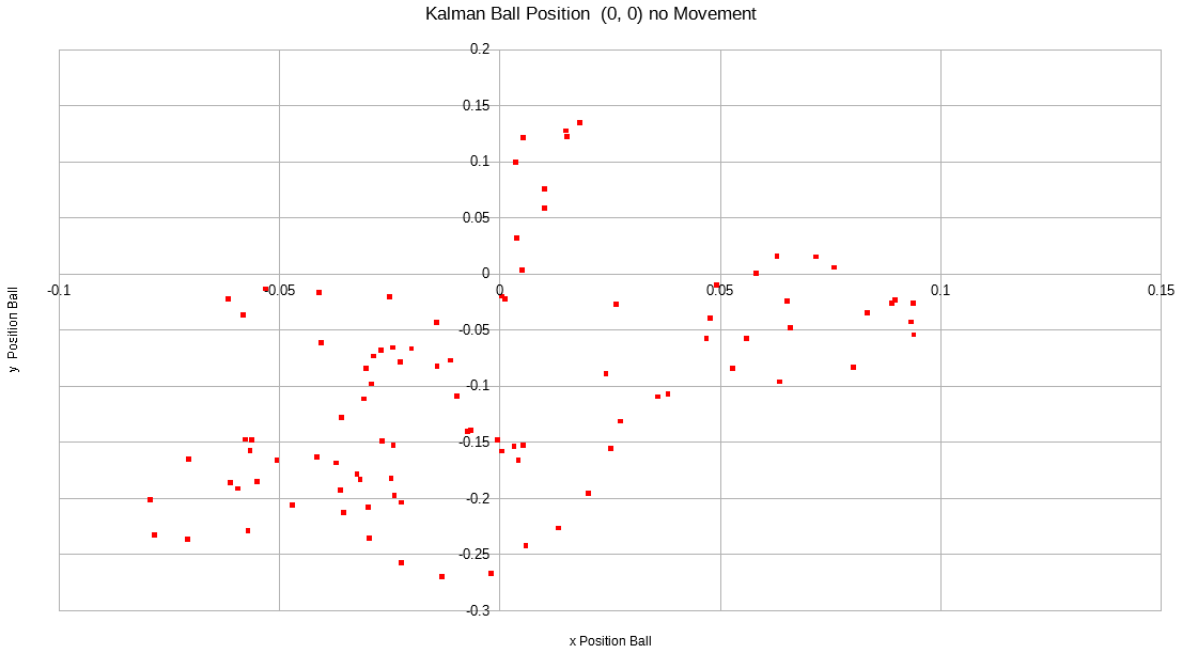

Figure 4.2: Positions of the Ball Position with no movement (0,0)

$$v = \frac{s}{t}$$
$$v_x = \frac{pos_x}{t}$$
$$v_y = \frac{pos_y}{t}$$

The graphic above shows the result of the filtered Ball position. One unit in game is app. 1.15m. The ball position is roughly between (0.161m - -0.3m , 0.1m - -0.08m).

The noise is reduced, but the calculation of the velocity produces a spread. The ball position is updated in every iteration. That makes the time <0.1s. This increases the minimal changes in position to a larger velocity. A additional Kalman filter was used to reduce the noise. To calculate the Interception point from here produces errors.

A correct localization of the robot is needed. With a correct localization, no movement of robot and ball, the perceived ball position changes around 0.1 units.

4.3 Implementing Block Motion

The block motion was done in Choreographe. It was exported and is an extra .xap file. The needed motion for the block is loaded in the beginning and locally stored. The file is formatted in the usual names, keys, times format.

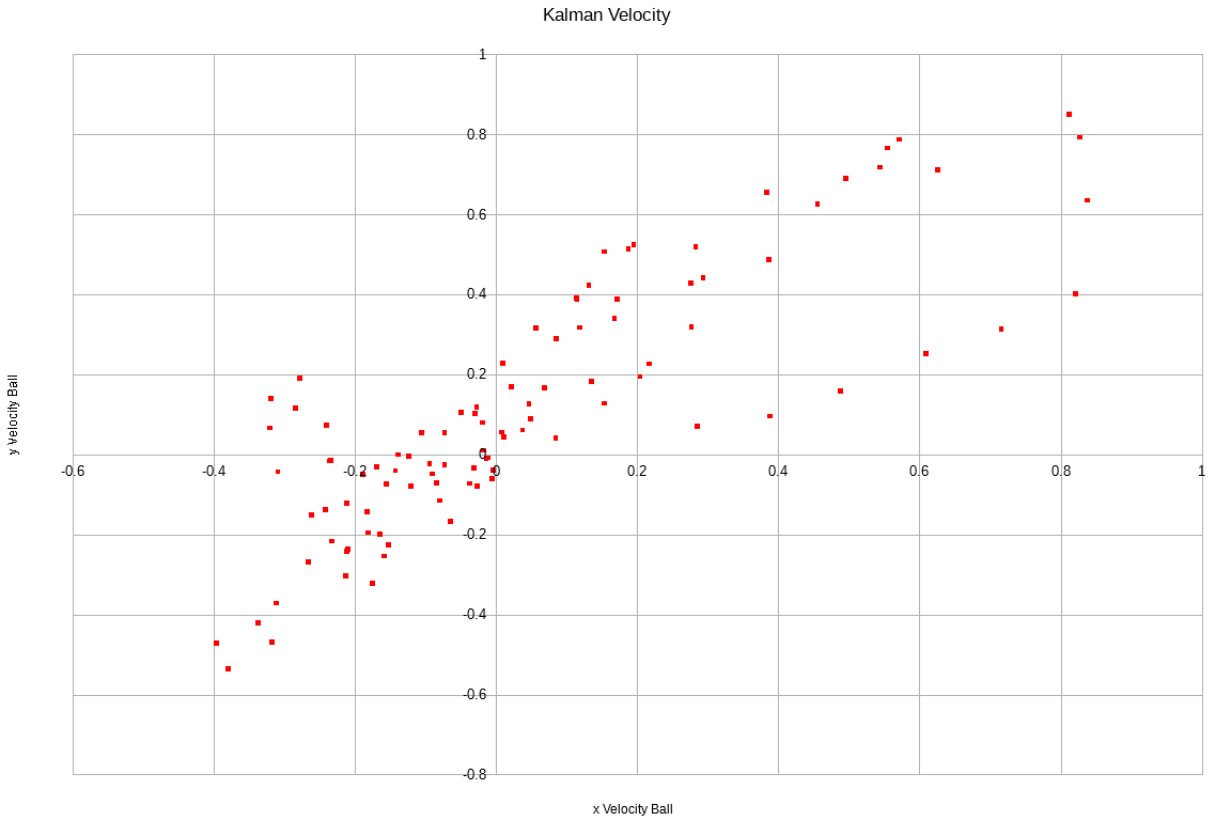

Figure 4.3: Velocity of the Ball Position with no movement (0,0)

4.3.1 Poses

As in Figure 4.4 seen, there are multiple poses available. The right block motions are identical as the left block except that the pitch and yaw joints in side specific joints are inverted. This was done in the conversion from Choreographe.

In Figure 4.6 the different foot position are shown. Without moving the supporting foot it was not possible to put the foot down on the g_0 base line. Turning the robot in position might make it collide and fall.

All the motions are created with the left leg remaining on the floor and extending the right leg to the block. To inverse this, blocking with left leg, a function was created. In this function all occurrence of left and right are reversed.

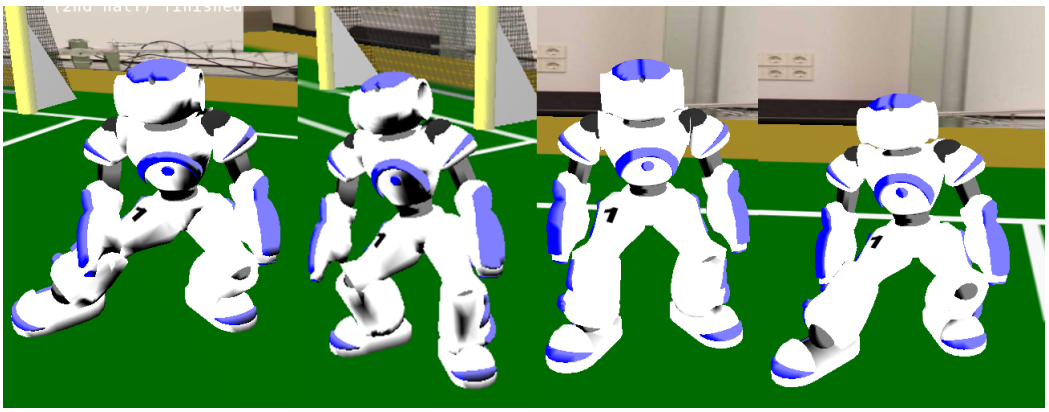

Figure 4.4: Side Long, Side Short, Degree Short, Degree Long

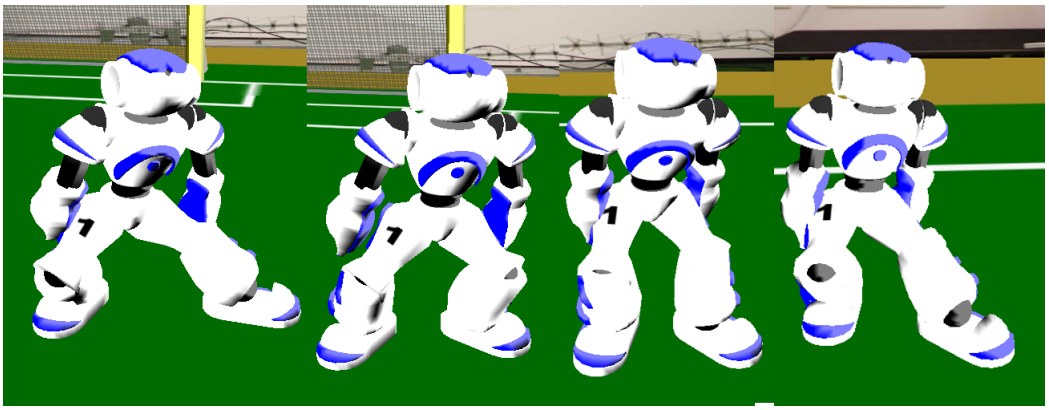

Figure 4.5: Side Long Right, Side Short Right, Degree Short Right, Degree Long Right

4.3.2 Emergency Block

The Emergency block is a last resort block. It covers unspecified a large area on the cost of stability and a high chance of falling or losing stability. It is only used if there is a high chance of a goal and that might stop the ball. It is only used in closed proximity of the own goal.

4.4 Implementing Interception Point

The interception point saves the ball positions and times when the position are perceived and uses these to get the vector the ball is moving. The interception point uses the ball position, player position, ball velocity and time from the block control.

The ball line was calculated from the ball position and the ball velocity.

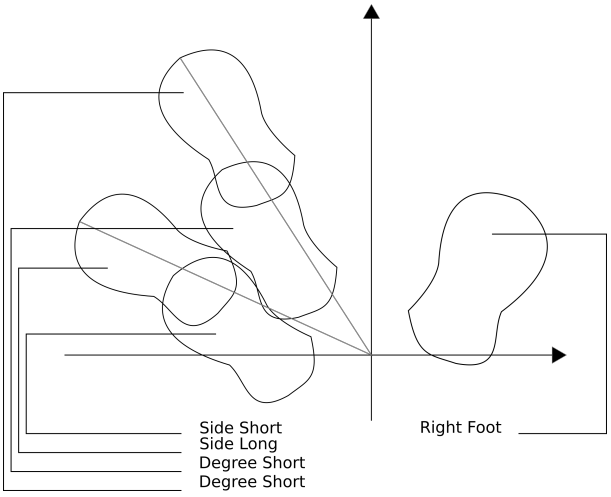

Figure 4.6: Room Coverages of the poses (Left)

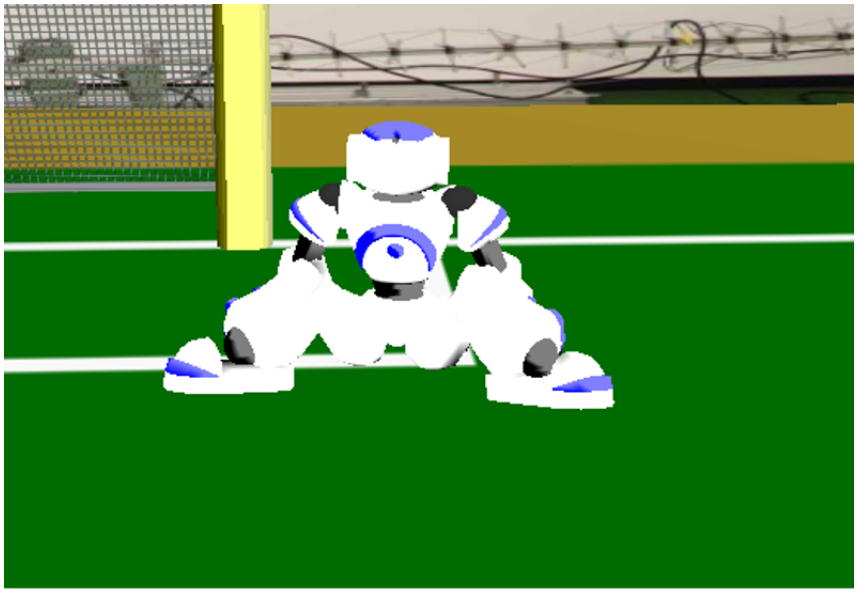

Figure 4.7: Emergency Block

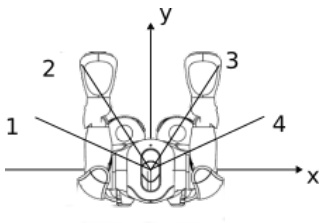

Figure 4.8: Block lines, 1: h_0, 2: h_1, 3: h_2, 5: h_3

ball position: $ball_pos = (x, y)$
ball velocity: $ball_vel = (vel_x, vel_y)$

ball line: ball(x) := $\frac{y_{ball\_vel}}{x_{ball\_vel}} x + x_{ball\_pos} * \frac{y_{ball\_vel}}{x_{ball\_vel}} + y_{ball\_pos}$

The interception location of the movement vector of the ball and the original block lines of the robot were calculated as following.

$$g_0(x) := a_0 x + c \text{ with } a_0 = 0 \text{ and c = 0}$$
$$g_1(x) := a_1 x + c \text{ with } a_1 = 1 \text{ and c = 0}$$
$$g_2(x) := a_2 x + c \text{ with } a_2 = -1 \text{ and c = 0}$$
$$ball(x) := bx + d$$
$$P(\tfrac{d-c}{a-b}, a\tfrac{d-c}{a-b} + c)$$

The ball will intersect with the base line of the robot, but it might be out of range of the robot.

After creating the motions it was needed to change the angles of the block lines:
The new block lines above are defined as following, with the local robot coordinates (0, 0) in the middle of the robot.

$$h_0(x) := \tfrac{2}{3}x \text{ with } a_0 = \tfrac{2}{3}$$
$$h_1(x) := \tfrac{10}{6}x \text{ with } a_1 = \tfrac{5}{3}$$
$$h_2(x) := -\tfrac{10}{6}x \text{ with } a_2 = -\tfrac{5}{3}$$
$$h_3(x) := -\tfrac{2}{3}x \text{ with } a_3 = -\tfrac{2}{3}$$
$$\text{ball line: } ball(x) := bx + d$$
$$P_{all} = P_0(\tfrac{d}{a_0-b}, a_0\tfrac{d}{a_0-b}), ..., P_3(\tfrac{d}{a_3-b}, a_3\tfrac{d}{a_3-b})$$

With the new block lines the interception point class is able to create possible block locations and times.

With the Interception locations P_{all} now the best location needs to be in front of the robot and less than the maximum extension of the pose away from it. If multiple

poses could block the ball, the pose with the smallest extension of the leg is chosen. The longer blocks extend 0.28 units and the shorter 0.15. If the interception location is closer than 0.15 to the robot, a short block is used, is it bigger a long block. Is it wider than the 0.28 the robot can not block the ball with this motion.

The robot should only block balls going into his side. To insure that the robot only blocks if he is facing the enemy goal or the ball is directed into the global area between (-4.3, 0.75), (-4.3, -0.75) for the left side and (4.3, 0.75),(4.3, -0.75).

Is no ball position perceived, no interception point is calculated.

4.5 Implementing Ball Blocking Behaviour

The Block behaviour was added to the Striker class. To change the state the interception point needs to fulfil one of these conditions:

- There must be an interception location and the moment for beginning a block has come (motion duration)
- The ball is too close to the robot, it performs the emergency block.

Chapter 5

Conclusion

5.1 Summary

In chapter 2 I discussed the backgrounds of this work, the robot, the simulation environment and the perception.
Afterwards, in chapter 3, the problem was described. The motion, the ball model and the ball behaviour were introduced.
The fourth chapter showed the solutions and the fifth the implementation of the motion, model and behaviour.

5.2 Evaluation

5.2.1 Evaluation Block Motion

The proposed block motions works in the test environment and blocks the ball successfully. The block motions were tested with a modified simspark_motion agent with the IPpthon enabled. To test and evaluate the motion it was checked if the robot is able to perform the motion fast (under 1s execution time from start to finish), if the robot was able to stop the ball in this position and go back into a neutral standing position.
The creation in Choreographe and the converting of the ball motions into functions worked very good.

name	successful repetitions	stability, comments
short block 45° left	50/50	Stable
long block 45° left	50/50	Stable, but turns the robot by app. 7°
long block side left	48/50	mostly Stable, but turns the robot by app. 30°
short block side left	50/50	Stable, but turns the robot by app. 35°
short block 45° right	50/50	Stable
long block 45° right	50/50	Stable, but turns the robot by app. 7°
long block side right	50/50	mostly Stable, but turns the robot by app. 30°
short block side right	50/50	Stable, but turns the robot by app. 35°

Table 5.1: Block motion test in SimSpark

5.2.2 Evaluation Block Behaviour

The block behaviour performed as proposed works successfully. It changes into the block state when the robot is supposed to block. To test the block behaviour, the ball positions where directly given to the block control and the perception of the robot were bypassed. With this modification the block performed successfully. The right block is chosen according to the situation.

5.2.3 Evaluation Ball Model

The proposed block model does not fulfil the requirements. The error around the real ball position is too big to distinguish correctly between the needed and the proposed block motions. The ball model creates wrongly block attempts. This is caused by spikes in the velocity. These spikes are created by a relatively high error of the position multiplied by the short time of measurement. This creates way higher velocities then the ball really has and triggers the interception location with only short time to react (because of the high velocity).

5.3 Conclusion

As described before, the approach to the blocking with the velocity of the ball brings problems with it. The perceived position of the ball has a great uncertainty and in the calculation of the speed in small time windows this error is multiplied. To gain a reliable interception point and interception time out of this is unlikely and I was not able to do so. The approach might need to change. Either have a better error correction to calculate the right velocity and time or change the whole approach. It seems to make sense to try to improve the perception in order to be able to calculate the right velocity of the ball. This would open new possibilities of interaction between the robot and the ball, with the possibility of improving not only the blocking, but also the passing and shooting of the ball. All the components worked alone in simple test environments without noise and without relying on the perception of the robot by feeding correct positions.

5.4 Future Work

While the approach over the velocity seemed promising, I was not able to find a solution to the problems at hand. Because of the unreliable ball position, it was impossible to distinct the real ball motions from the error of the perception. Once the problem with the velocity is solved or an other approach is found, which gives reliable interception locations, the block motions itself can be improved. One way could be to have a threshold of numbers of velocity which needs to be reached, before it recognises it as a real change in position. This might lead to a slower reaction time, meaning only longer balls shot could be blocked. Another approach could be to have a threshold barrier which triggers the block no matter how fast the ball is, could lead to a successful block like with the emergency block. For example the motions can be expanded. Another approach could be to create a dynamic foot positing to stop the ball totally, to evolve the blocking of the ball to a stopping.

Bibliography

[1] [RoboCup 2016] SPL final: B-human - UT austin villa - YouTube, 2016. https://www.youtube.com/watch?v=XgRw42oHN-YS#t=25m06s , last checked 04.05.2017.

[2] The austin villa robot soccer team: Home, 2017. https://www.cs.utexas.edu/ AustinVilla/ , last checked 04.05.2017.

[3] B-human, 2017. https://www.b-human.de/ , last checked 04.05.2017.

[4] Discover nao, the little humanoid robot from SoftBank robotics | SoftBank robotics, 2017. https://www.ald.softbankrobotics.com/en/cool-robots/nao, last checked 04.05.2017.

[5] Oxfor dictionary, 2017. https://en.oxforddictionaries.com/definition/robot 26.10.2017, last checked 29.10.2017.

[6] Regele, Levi, and Bott. Prorobot predicting the future of humanoid robots, 2004.

[7] Burgard Thrun and Fox. *Probabilistic Robotics*. MIT Press, 2006.

[8] Yuan Xu and Hedayat Vatankhah. Simspark: An open source robot simulator developed by the robocup community. *RoboCup 2013: RobotWorld Cup XVII*, pages 632 – 639, 2014.

List of Figures

List of Tables

Definitions

DAInamite	RoboCup team of the TU-Berlin
keeper/goalie	Goalkeeper, always with jersey number 1
Nao	Nao robot from Aldebaran-Robotics
Python	An interpreted programming language with high focus on readability
RoboCup	Robot soccer world cup with different leagues (see SPL)
SimSpark	Simulation environment for the Standard Platform League
SPL	standard platform league, all robots in the league are the same
.xap	file format and is a package managment system