MASTER

Pushing further
alternative ball-handling by intercepting and pushing with an omni-directional soccer robot

Hendriks, O.

Award date:
2013

Pushing Further

Alternative ball-handling by intercepting and pushing with an omni-directional soccer robot

Embedded Systems Master Thesis

February 2013 - August 2013

Eindhoven University of Technology

Control Systems Technology Group, prof.dr.ir. M.Steinbuch

July 2013

Author

ing. Okke Hendriks

Student number: 0758134

OkkeHendriks@gmail.com

Tutor

ir. Rob Hoogendijk

r.hoogendijk@tue.nl

Supervisors

prof.dr.ir. Twan Basten

a.a.basten@tue.nl

dr.ir. Pieter Cuijpers

p.j.l.cuijpers@tue.nl

dr.ir. René van de Molgengraft

m.j.g.v.d.molengraft@tue.nl

Abstract

Tech United Eindhoven has been participating in the RoboCup mid-size soccer league since 2005 using its omni-directional soccer robots called Turtles. The team has been searching for a method to increase the speed at which the Turtles can receive the ball and subsequently score or pass the ball. Currently only a ball handling mechanism is used for this purpose. However, it requires the Turtle to be oriented towards the ball to intercept it, after which it has to turn before it can drive or shoot to the preferred target. To increase the speed of play, a new type of ball handling is developed in the form of the push action. During the push a Turtle will intercept and control the ball using its housing. An impact model, which describes the trajectories of the ball before and after impact has been implemented. In order to intercept the ball at the impact location, having the correct velocity and orientation, a trajectory planner has also been developed and implemented. The push action has been integrated into the general strategy of the Turtles such that the push can be used at corner situations. During the RoboCup World Championship 2013 a goal has been scored using the push. Future work includes the development of a more sophisticated trajectory planner and using the push action for passing.

Acknowledgements

First of all, I want to thank all people at Tech United Eindhoven for their spirit and commitment keeping Tech United the (2^{nd};) best team of the world! The software of the Turtles has a steep learning curve but with your help, support and patience I was able to master the necessary parts and do the work that is presented in this thesis.

I would especially like to thank my daily tutor Rob Hoogendijk for brainstorming together about the endless possibilities of the push while still ensuring I was working in a structured manner. Also, I would like to thank my supervisor from the Electronic Systems group Twan Basten for supporting me during this project. Even while he was on vacation, emails are usually answered within the hour. I also really appreciate the time which the graduation committee has invested in me.

This project, and the rest of my studies, have been enabled and for a large part funded by my parents, which I cannot thank enough for motivating and providing me with everything I could wish for.

Contents

1 Introduction **6**
1.1 Background . . . 6
1.2 Motivation . . . 6
1.3 Challenges . . . 7
1.4 Outline . . . 9

2 State of the Art **10**
2.1 Turtle Hardware . . . 10
2.1.1 Housing . . . 10
2.1.2 Processing . . . 10
2.1.3 Omni-wheels . . . 10
2.1.4 Ball Handling . . . 11
2.1.5 Sensors . . . 11
2.1.6 Actuators . . . 12
2.1.7 Communications . . . 12
2.2 Turtle Software . . . 12
2.2.1 Software architecture . . . 12
2.2.2 Motion software modules . . . 13
2.2.2.1 Strategy . . . 13
2.2.2.2 Actions . . . 13
2.2.2.3 Control . . . 15
2.2.3 Trajectory Planner . . . 15
2.2.4 Mu-Fields . . . 16

3 Push **24**
3.1 Goals . . . 24
3.2 Types . . . 24
3.3 Push software flow . . . 26
3.3.1 Shooting turtle . . . 26
3.3.1.1 Determining point of intercept . . . 26
3.3.1.2 Push ranking . . . 28
3.3.2 Pushing Turtle . . . 30
3.3.2.1 Finished condition . . . 30
3.3.2.2 Predicting ball movement . . . 30
3.3.2.3 Push physics . . . 33

4 Turtle movement **47**
4.1 Trajectory planner requirements . . . 47
4.2 Trajectory generation . . . 47
4.3 Cubic polynomials . . . 48
4.3.1 1D polynomial trajectories . . . 48

4.3.2 Trajectory feasibility check . . . 49
4.3.3 Implementation problems . . . 51
4.4 Velocity limiting . . . 52

5 Experiments 53
5.1 Push accuracy . . . 53

6 Conclusion 56
6.1 Reflection on requirements . . . 56
6.2 Future work . . . 57

Bibliography 58

Nomenclature

MS Motion State, the motion state is defined as the combination of location, rotation and velocity of for example a ball MS_b or robot MS_r. The subscript indicates which object the motion state belongs to. A target motion state can, besides a location and velocity vector, also contain an absolute velocity instead of a vector. When one of the components of a motion state has to be addressed, the following notations are used.

- Location: $MS.loc.x$ & $MS.loc.y$
- Velocity: $MS.vel$ gives the vector and $MS.vel.x$ & $MS.vel.y$ its components.
- Rotation: $MS.rot$
- Absolute velocity: $MS.velabs$

BDF Ball Deceleration Factor

COR Coefficient of Restitution

DOF Degree(s) of Freedom

GCS Global Coordinate System, the global coordinate system forms the basis of the coordinates that are used throughout the Turtle software. The origin is placed at the middle of the playing field with the positive y-axis always directed at the opponent goal.

LUT Lookup Table

PCS Push Coordinate System, the push coordinate system is a coordinate system with its origin on the point of impact of the ball and a push surface. The positive y-axis is placed perpendicular to the push surface, on the side that will be impacted by the ball.

POI Point Of Intercept

Chapter 1

Introduction

The work presented in this thesis has been carried out for the Tech United robot soccer team which participates in the international RoboCup Challenge. In Section 1.1 the background information of Tech United and the RoboCup challenge is given. For the world championship robot soccer 2013, Tech United wanted to implement a new type of action for its robots to perform, the push action. The motivation for this new action is given in Section 1.2. In order to implement the push, a number of challenges have to be tackled. These challenges can be found in Section 1.3.

1.1 Background

RoboCup [11] is an international research and education initiative. It fosters artificial intelligence (AI) and intelligent robotics research by providing a standard problem at which a wide range of technologies can be integrated and examined. For this purpose, RoboCup chose to use soccer as a primary domain. In a highly dynamic environment such as soccer, cooperation between multiple fast-moving robots requires a lot of new technologies. This leads to major innovations that society can benefit from in the future.

The RoboCup challenge consists of several different competitions, ranging from simulation leagues to humanoid robot leagues. Tech United Eindhoven [13], a team of Eindhoven University of Technology (TU/e), participates in three leagues; the @Home, Humanoid, and Middle-Size League. The push action is implemented in the Middle Size-League (MSL) team of Tech United Eindhoven, in which two teams of five robots autonomously play soccer.

Tech United is developing three different kinds of robots. The Turtles (Tech United RoboCup Team Limited Edition) are the teams soccer robots, AMIGO the care robot and TUlip the humanoid robot. A team of about 80 enthusiastic members is working hard on these robots, largely in their spare time. The team participates in the RoboCup Challenge to compare the level of the Tech United robots to other robots from all over the world and to stimulate cooperation and the sharing of knowledge.

The Tech United Turtles are omni-wheeled robots, which are 80 cm high, weigh 35 kg, reach a top speed of 5 m/s and can shoot a ball with 10 m/s. The robots are required to have all sensors and processing on-board. The real-time software of the robots is written in C-functions that are placed in Matlab-Simulink blocks to create a modular software design. The industrial computers on the robots run Ubuntu with a real-time kernel to execute the software generated from the Simulink schemes.

1.2 Motivation

During tournaments Tech United noticed that being quick, especially while handling the ball, is essential to win a RoboCup soccer match. If a robot takes too much time receiving, aiming and shooting the ball, the opponent has time to react. The Turtles currently have a ball handling mechanism, consisting of two actuated wheels attached to levers used to intercept and control the ball, see Chapter 2. There are two main disadvantages of this system,

- Intercepting, aiming and shooting towards a certain target takes too long.
- The Turtle velocity has to be low in the direction perpendicular to the movement of the ball.

The second disadvantage has become a larger problem in recent years because the defensive positions of the opponents have been getting better. They are defending in between the ball and a receiving Turtle. An indirect pass is a solution to this defensive behavior, but the Turtles have to slow down to catch the ball. A solution has to be found which does not require the Turtle to slow down.

1.3 Challenges

The push action will give the Turtles the ability to hit the ball using its housing, see Figures 1.1 and 1.2. The angle between the incoming- and outgoing trajectory (θ_{in}, θ_{out}), as well as the velocity of the ball, will have to be controlled. They have to be controlled such that a target, which is being determined on the fly, can be hit. The Turtle can influence the outgoing ball trajectory by changing and controlling its rotation and velocity.

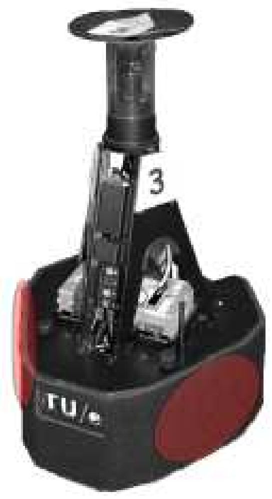


Figure 1.1: Surfaces of the Turtle used for pushing are indicated in red [8].

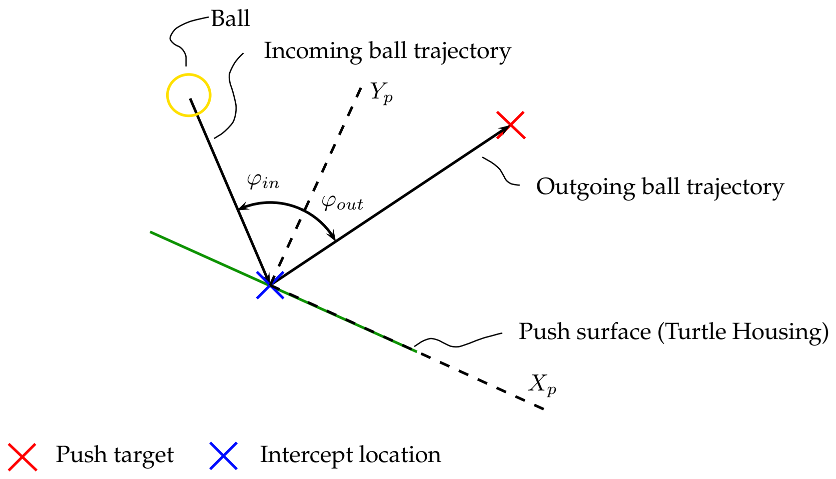


Figure 1.2: Schematic representation of a push action. With φ_{in} the angle between the push coordinate y-axis and the incoming ball trajectory and φ_{out} the angle between the same y-axis and the outgoing ball trajectory.

A previous implementation of the push action has been created by J.W. Lamers [8]. This implementation can only push from a position determined beforehand and controls only the outgoing direction of the ball, and therefore gives no control over the velocity of the ball. Furthermore, the very simple model used to predict how the ball bounces of the Turtle proved to be inaccurate. The function was not integrated in the strategy part of the software and therefore it could not be used during a real match.

The main objective of this work is to

design and implement an improved and reliable version of the push action. Pushing from a fixed position as well as while the robot is moving must be implemented. The accuracy and integration into the strategy must simultaneously be improved.

The analysis of the previous implementation and the specifications of the team lead to a number of requirements and challenges.

Requirements and sub-challenges The following three requirements apply to the main objective, for each requirement its sub-challenges have been indicated.

1. Control the direction and velocity of a pushed ball

   - Determine the physics involved while pushing a ball. Develop a model that can be used to predict the movement of a pushed ball and calculate the required position, rotation and velocity of a Turtle for a given incoming ball and push target.
   - Static and dynamic pushing actions should be implemented. Static only controls the direction of the ball and assumes that the Turtle has no velocity at the moment of impact. The dynamic push controls the direction and velocity of a pushed ball.
   - Adapt the current, or design and implement a new, trajectory generator that can generate a trajectory for a Turtle to follow such that it has a given motion state (MS) at a given time. The current implementation of the trajectory planner is not able to create a trajectory with a certain end velocity and cannot control the time of arrival.

2. Integrate the push action into the current strategy and increase the usability of the push action.
    - Ensure that external influences: opponents, obstacles, etc. are taken into account.
    - The software should decide when and how to use the push action. Therefore, a method should be developed in order to predict the successfulness of a hypothetical push action.
3. The developed software should be robust for sensor and control imperfections. Therefore, the push action should be tested thoroughly, both in simulations and on the Tech United soccer field.
    - Determine performance of the push function in terms of its accuracy.
    - Determine performance of the implemented push function with respect to normal ball handling.
    - Update the existing Tech United simulator with the push physics, based on the model found from requirement 1.

1.4 Outline

The current state of the hard- and software of the Turtles is described in Chapter 2.

The details of the push action are described in Chapter 3, which is subdivided into a number of sections. First, the goals and types of push are presented in Sections 3.1 and 3.2. Secondly, the push software is described by following the software flow of the shooting (Section 3.3.1) and pushing (Section 3.3.2) Turtles. This includes determining the point of intercept, predicting the motion of the ball, calculating the angle and velocity of the Turtles using an impact model and determining the probability of success for a hypothetical push action.

Subsequently, the pushing Turtle will need to generate a trajectory to intercept the ball, this is done using a trajectory planner described in Chapter 4.

To conclude this project, an experiment testing the accuracy of the implemented push action has been performed and is documented in Chapter 5.

A reflection on the requirements is preformed and future work is stated in Chapter 6.

Chapter 2

State of the Art

In this Chapter the current state of the Turtles is described. The hardware including the housing, ball handling, omni-wheels, sensors and actuators of the Turtles are discussed in Section 2.1. The different software modules are discussed in Section 2.2, the trajectory controller and mu-fields, a method to make fuzzy decisions, are discussed in more detail in Sections 2.2.3 and 2.2.4 respectively.

2.1 Turtle Hardware

The Turtles each have their own sensing, actuating, processing and communication hardware on board.

2.1.1 Housing

The housing of the current Turtles are covered in a 0.5cm layer of foam to protect the robots and referees during a match. As mentioned by J.W. Lamers in [?, page 7] the foam absorbs a lot of the energy of a ball hitting the housing in the form of elastic deformation. This also causes the push surface to temporarily deform a little. When this deformation is not incorporated in the physics model a deviation between the calculated and achieved outgoing ball trajectory will occur. A new housing has been developed in parallel during this project which incorporates a new type of foam which will deform more evenly on impact. Also, the new housing that has been developed is more stiff. This has as a result that less energy is dissipated as elastic deformation and that the model matches the reality more closely.

The absorption of energy during impact is the main reason to introduce dynamic pushing. Dynamic pushing, where the velocity of the Turtle is non-zero at the moment of impact, can be used to ensure that the ball velocity after a push is sufficiently high.

2.1.2 Processing

The processing at each of the Turtles consists out of a Beckhoff industrial computer, the C9900-C543 with an Intel® Core™ i7 2710QE, 2.1 GHz processor. On this PC a real-time kernel runs the compiled Matlab-Simulink models. The kernel is a patched version of the Ubuntu Kernel, see [14]. Currently the most processing power goes into the vision part of the software. Furthermore, very little is documented about the Turtles software performance.

2.1.3 Omni-wheels

Three omni-wheels, together with the driving motors, form the propulsion system of the Turtle. An omni-wheel is a wheel that has an extra degree of freedom compared to a normal wheel. The wheel is lined with smaller wheels such that it can move freely in the direction perpendicular to the conventional rotation direction. This creates a big advantage because the platform becomes fully holonomic, meaning that it can immediately accelerate in any direction.

2.1.4 Ball Handling

The ball handling consists of two movable arms on the front of the Turtle. On the ends of these arms two powered wheels are attached, see Figure 2.1. These wheels are controlled such that the ball will be kept close to the Turtle when it is maneuvering over the field. The ball, by regulation, cannot be enclosed by more than one third and must always move/rotate in the natural direction.

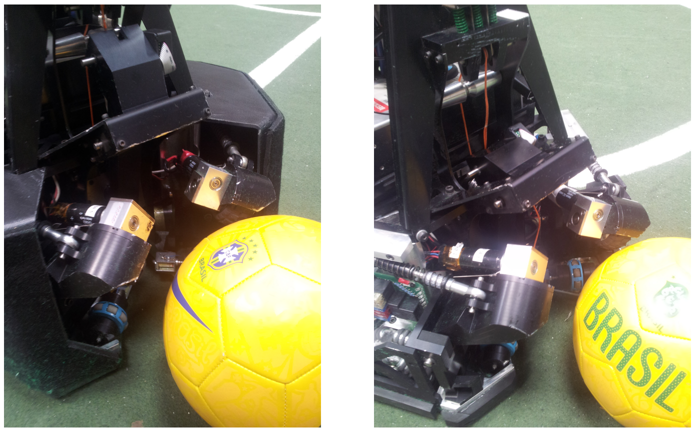


Figure 2.1: Turtle ball handling mechanism.

2.1.5 Sensors

The Turtles have sensors to detect their position in the field, detect the ball and obstacles and give feedback about the ball handling mechanism. The main sensor for the perception of the environment is the vision camera.

Vision The vision system is the combination of a camera on the top of each Turtle and a parabolic mirror on top of this camera. By looking into this mirror the Turtle has a field of view of 360 degrees. The image that is retrieved from the camera is analyzed. By analyzing position and direction of the field lines the Turtle can determine its own position in the field. The same method is used to detect the position of the ball and the opponents.

Encoders The Turtles have encoders on their main driving motors. These encoders on the main driving wheels are used in the inner feedback loop, which controls the velocity of the motors. The encoders are also used to determine the displacement of the robot in case the vision system is unable to find a fix on the field lines and thus its position. This can sometimes happen because too much of the field of view is blocked by obstacles.

Tachometers The ball handling has tachometers attached to the motors that drive the ball handling wheels. These tachometers are used in the feedback loop to control the rotation speed of the ball handling wheels.

Ball handling position On each arm of the ball handling a potentiometer measures the angle of the arms. This signal is also used in the feedback loop that controls the ball handling mechanism.

Laser Range Finder Only the keeper, which is the only Turtle with a different design, has a Laser Range Finder (LRF). The LRF is used only when the goalkeeper is positioned in the goal area. It detects the goal behind itself to improve localization.

2.1.6 Actuators

To move around, and to control and shoot the ball, the Turtles have a number of actuators.

Main driving motors Each Turtle has three (except for the keeper which has four) motors. Each motor is connected to one of the omni-wheels.

Ball handling motors There are two ball handling motors, one on each ball handling arm of every Turtle. They are used to drive two wheels with LEGO tires that grip and move the ball when the Turtles are moving around.

Shooting lever The shooting lever consist of a piston inside an electromagnet. This electromagnet is connected to a capacitor by an electronic switch. Whenever the ball needs to be shot, the electronic switch is opened and closed using a PWM signal. This forces the piston forward with a certain force and shoots the ball in the direction at which the Turtle is aimed. The height of the lever can be adjusted such that a flat shot, lob shot or anything in between can be performed.

2.1.7 Communications

Each Turtle is connected using a Wi-Fi network. The robots communicate with the referee's computer and with their peer players. Communication with the referee's computer is used to receive start, stop, free kick, penalty, etc. commands from the referee. The inter-Turtle communication, is used to share information about the environment and agree on which actions to take. The UDP protocol is used with a custom software interface which tries to keep the overhead as low as possible in order to decrease packet loss as much as possible.

2.2 Turtle Software

The software is a combination of code generated by Matlab-Simulink and plain C-code. In Simulink a number of blocks are created which define the architecture of the software. Each of these blocks is a so-called S-Function; This is normal C-code but with such an interface that it can be viewed and connected together inside Simulink. The usage of these blocks gives a clear overview of the code structure. This structure, as well as more detailed explanations of the independent blocks, are elaborated upon in the following Sections 2.2.1 and 2.2.2.

2.2.1 Software architecture

Globally the software is divided into three main parts: Motion, Vision and the Worldmodel. The top level of these schemes can be found in Figures 2.2, 2.3 and 2.4. In all of these schemes there are a number of blocks connected by lines. The blocks can either be an S-Function sub-scheme or a standard function block of Simulink. The lines connecting all the blocks carry the information. A line can either be a single variable or a bus carrying a number of variables.

Only a short description is given of the vision and the worldmodel scheme. The Strategy and Control sub-scheme of Motion is of most interest for this thesis because all necessary adaptations to enable push have been made in this part of the code. Therefore they are explained in more detail in Section 2.2.2. The decouple robot scheme performs the low-level control and forms the interface with the Turtle hardware.

Vision receives the information from the camera and processes it. Next, it will calculate the position of the Turtle, ball and obstacles. This information will be communicated to the Worldmodel. The Worldmodel does not only receive information about the environment from the Vision scheme running on that specific Turtle, it also receives communicated information from the other Turtles. The task of the worldmodel is to merge all information into a consistent representation of the environment. This will then be used in the motion scheme to decide on and perform the actions of the Turtle. Vision runs at approximately 68Hz, the Worldmodel at 10Hz and Motion at 1000Hz.

2.2.2 Motion software modules

All changes needed to implement the push action have been performed in the *Strategy and Control* sub-scheme, see Figure 2.5, of the *Motion scheme*. Specifically in the *Strategy*, *Actions* and *Control* sub-schemes which are explained in more detail at the following sections.

2.2.2.1 Strategy

The task of the *Strategy* scheme, see Figure 2.6, is to analyze the situation and make sure the Turtle acts accordingly. During play a Turtle makes decisions on what actions to take.

The Tech United software is in the mids of a transition from role oriented to task oriented acting of the Turtles. At the time of writing, role oriented acting is still used, role oriented acting works as follows. A number of roles are defined that are assigned to the different Turtles in the field. Based on a number of rules the roles can switch between the Turtles. The Turtle that has possession of the ball will for example always be *attacker main*. There are six different roles: *attacker main, attacker assist, defender main, defender assist, defender assist 2 and goalkeeper*. Each of these roles depict which behavioral algorithm is running.

These algorithms basically tests certain parameters, as in: ‘does our team have the ball?’ or ‘is another player in a position to receive a pass?’, and takes actions according to the results of these tests. The actions are implemented as function calls to functions that reside in the action handler. These functions make up all the different actions that can be performed, e.g. ‘go to a specific position on the field’ or ‘shoot at the goal’. Push is one of these actions. The actions, on their turn, contain the code to control the Turtle such that a specific action is actually executed. Figure 2.7 shows the control flow in the form of a schematic diagram.

At the *attacker main* the behavioral algorithms are implemented in the form of the so-called skill selector. The skill selector takes all possible actions and compares the probability for success of all of them. The probability of success of each action is given by probability functions which are designed specifically for these actions. The probability functions return a probability from 0.0 to 1.0. This is subsequently multiplied with a tunable weight factor and compared by the skill selector. The skill selector chooses the action that has the highest weighted probability and executes this action. The skill selector uses hysteresis to prevent constant switching between two actions.

These probabilities are more a sort of ranking, as opposed to probabilities in the mathematical sense, because a ’probability‘ of one will not guarantee 100% success or vice versa. Therefore, the probability functions for the push will be called ranking functions.

2.2.2.2 Actions

The Action scheme, see Figure 2.8, contains the action handler that makes sure the functions of the actions that belong to a certain role and situation are called. Every action function basically calculates a target motion state for the Turtle which is required to perform a certain action. There are two other sub-schemes, the *waypoint handler* and the *sub-target planner*. The *waypoint handler* is used in case of a special action that follows a trajectory which consists of a number of waypoints. The *sub-target planner* makes sure the Turtles take the fastest path in the case there are obstacles between its current location and a target location.

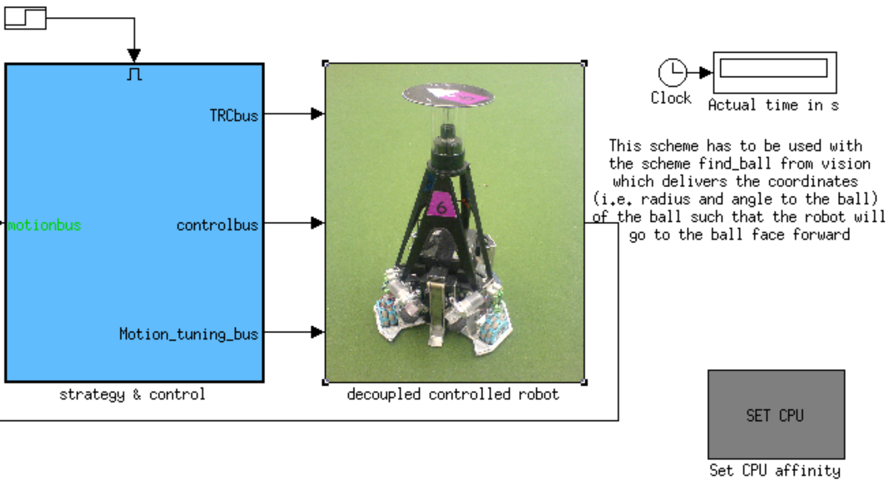

TRCbus
motionbus
controlbus
Motion_tuning_bus
strategy & control
decoupled controlled robot
Clock
Actual time in s
This scheme has to be used with the scheme find_ball from vision which delivers the coordinates (i.e. radius and angle to the ball) of the ball such that the robot will go to the ball face forward
SET CPU
Set CPU affinity

Figure 2.2: Motion top level Simulink scheme.

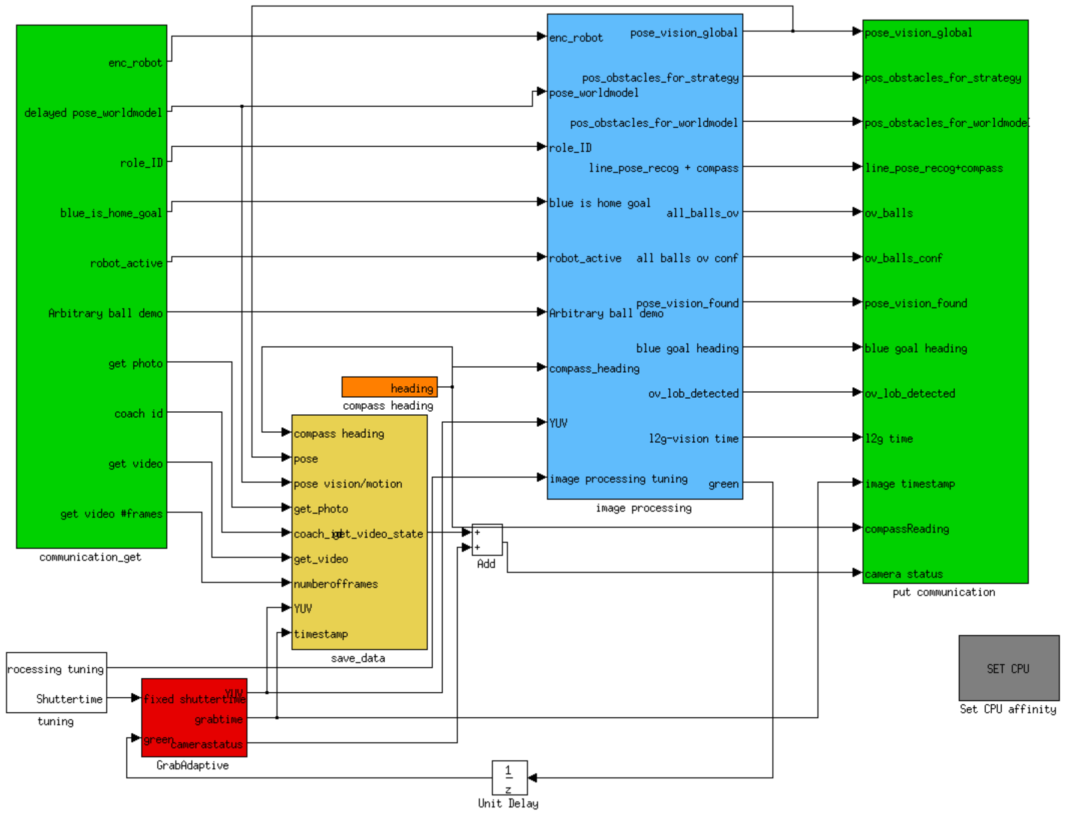


Figure 2.3: Vision top level Simulink scheme.

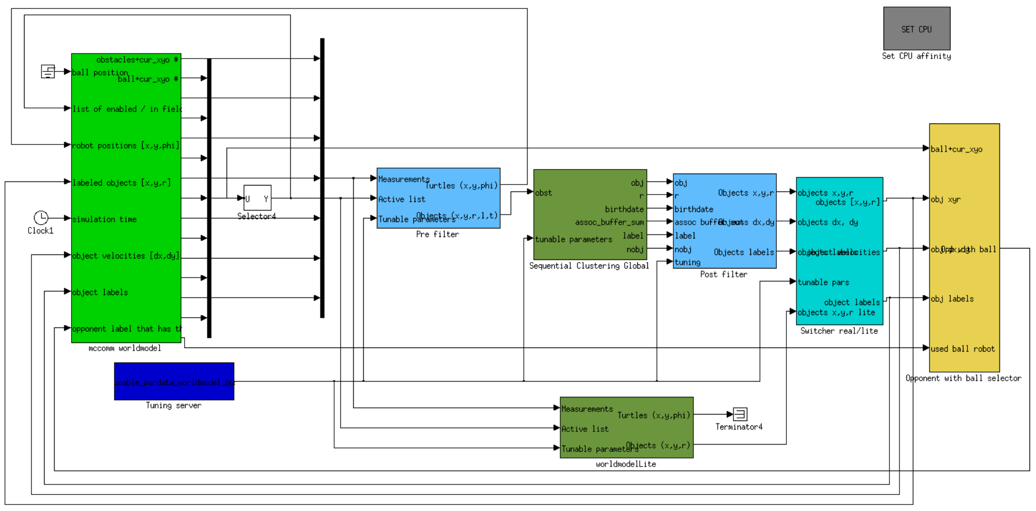


Figure 2.4: Worldmodel top level Simulink scheme.

2.2.2.3 Control

The Control scheme, see Figure 2.9 on page 21, forms the last layer before the low-level control. It is divided into a number of sub-schemes for the different parts of the Turtle. The manual control sub-scheme is used when the Turtles are controlled using the joystick or keyboard. The emergency handler shuts down the motor amplifiers in case the emergency button has been pressed. The trajectory controller takes the (sub-)targets from the Actions scheme and creates the control signals which are used as inputs for the low-level controller. The kick control, controls the shooting lever, the ball handling control the ball handling mechanism and the led control the LEDs at the bottom edge of the Turtle.

The challenge when intercepting a moving ball is to move a Turtle to a certain location, with a certain orientation and velocity at a certain time. Therefore, the trajectory planner is of interest in the light of this thesis. To understand the motion control of the turtles the analysis by A.W. van Zundert in his report 'Traction control for Omni-directional robots' [15], has been studied. In this report a simplified diagram is created from the different Matlab-Simulink schemes and c-code, which is shown in Figure 2.10.

2.2.3 Trajectory Planner

The Turtles have three DOF, the local x-axis, the local y-axis and the rotation. The current trajectory planner uses the method described in [10]. This method only describes the xy-DOF, the rotational DOF is calculated separately and added to the solution of the xy-DOF. The final velocity is always chosen to be zero in order to prevent discontinuities in the solution when getting close to the final motion state. In practice the final motion state will hardly ever be reached because the target location is altered continuously.

The optimal control problem is to minimize the complete trajectory time t_f. Therefore the acceleration and deceleration will always be a_{max} or $-a_{max}$ respectively. The maximal velocity is always set to v_{max}. These values can be determined analytically, see [10], but in practice the values are tuned manually on every field such that the Turtles can just follow the trajectory. In order to make the problem more tractable, each DOF is handled independently at first. In the end, both DOFs are synchronized using a bi-section algorithm.

The problem is broken down into a number of distinct cases, in which either the maximal acceleration or deceleration is applied or the Turtle is cruising with v_{max}. These cases are checked until the final motion state is reached with a velocity of zero. The cases are:

1. The initial velocity is negative, the vehicle has to accelerate with maximal control effort.
2. The vehicle is cruising at maximum velocity until it has to decelerate.
3. The vehicle has to decelerate until it reaches zero final velocity.
4. The vehicle moves faster than its maximum velocity. This can happen due to noise or a change in the wanted maximal velocity. The vehicle has to decelerate until it reaches its maximum velocity.

2.2.4 Mu-Fields

A mu-field is the name of a technique employed by Tech United to build a gradient field which can, for example, be used to determine an attractive location on the playing field depending on the state of the surrounding world.

The name comes from the field of fuzzy logic, where μ-functions are used to translate linguistic definitions to fuzzy sets. These fuzzy sets can then be used in the code to asses certain situations. For example, the phrase ‘Close to a Turtle’ is usually defined as a function which returns 1 when the distance to a turtle is below a certain threshold and 0 otherwise. Using fuzzy logic a function is defined that changes this transition, and introduces a weight factor β, such that a smooth transition from $1 \cdot \beta$ to 0 is achieved.

For the position of an opponent on the field, a Gaussian bell function, see Figure 2.11, is usually used. The highest value of the function being placed at the center of an opponent which then represents a non favorable location. More functions are evaluated and added together to give certain areas of the playing field a lower or higher value such that the lowest (or highest, depending on the implementation details) point on the field is the most favorable position.

A GUI is available to design these mu-fields. An example of a mu-field used to determine good pass opportunities is shown in Figure 2.12. Different actions require different mu-fields. For the push action mu-fields could be used to identify the best targets and possibly also to identify the best intercept position on the trajectory of a ball.

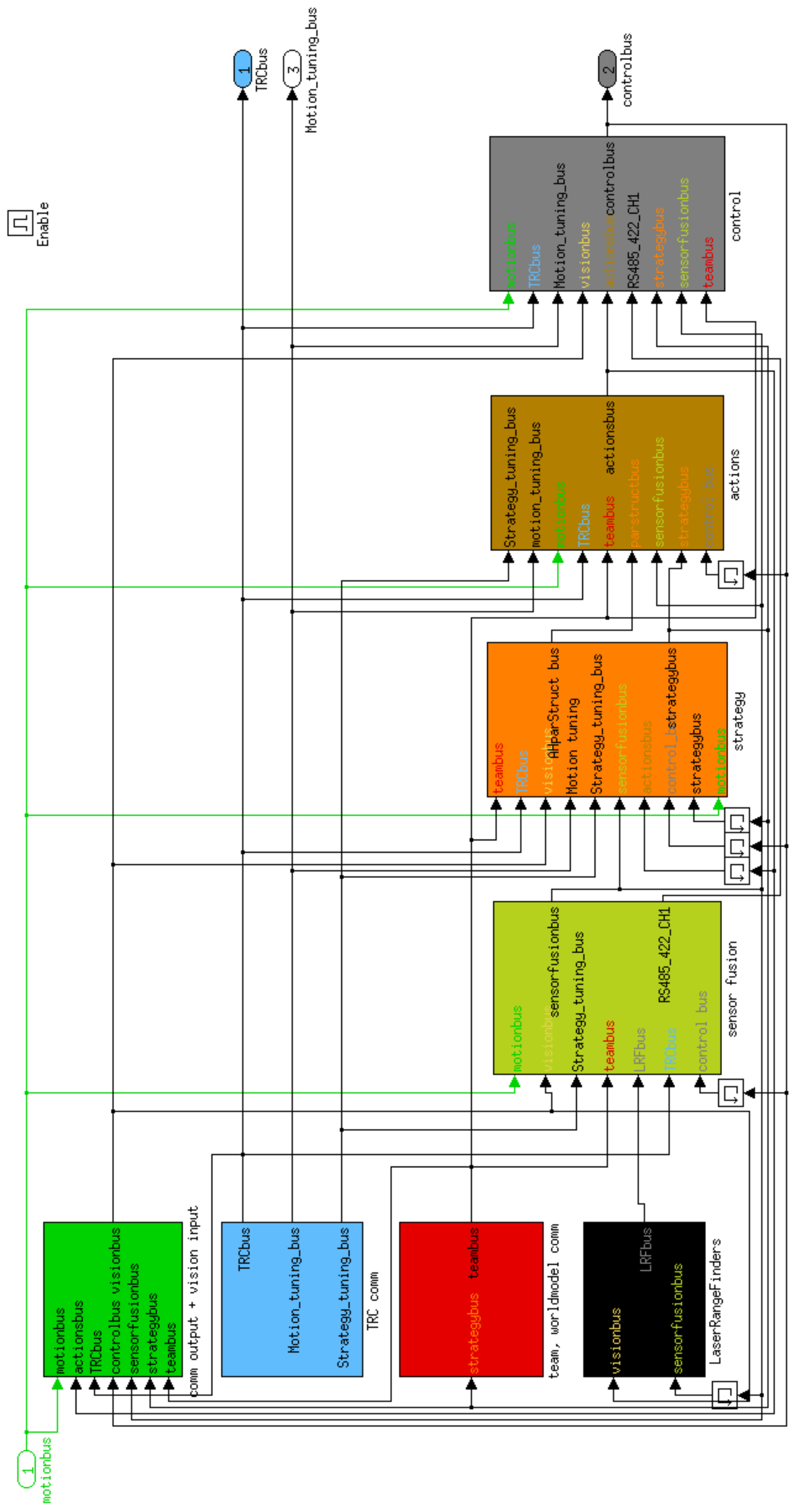


Figure 2.5: Strategy and Control scheme.

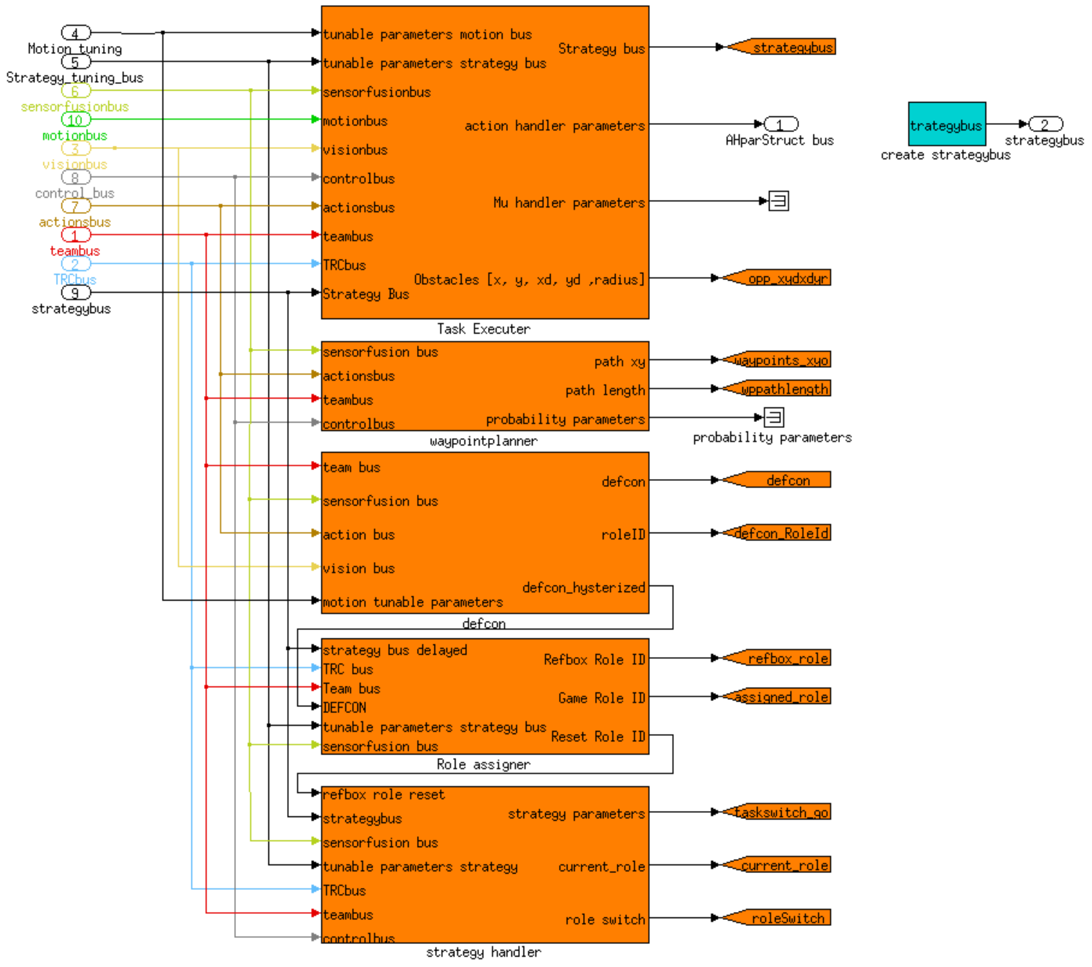


Figure 2.6: Strategy scheme.

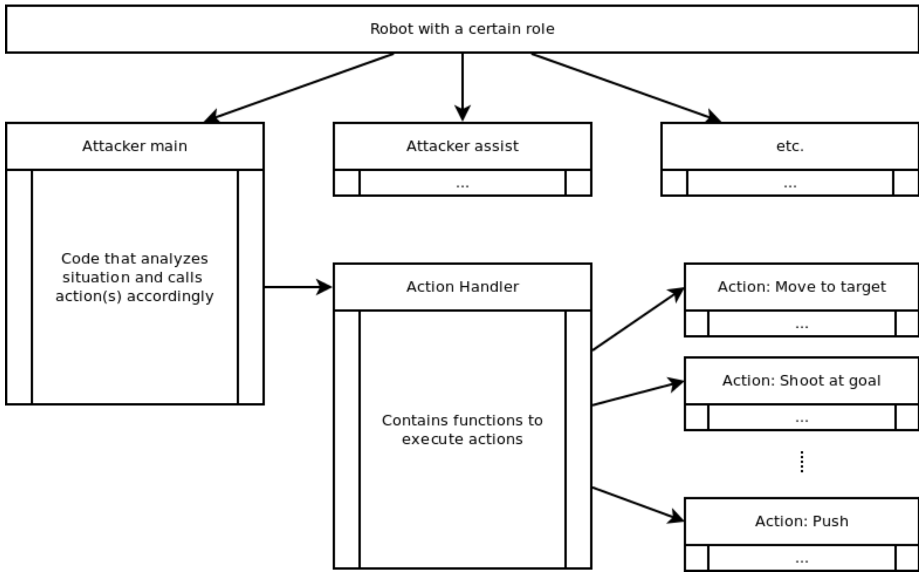

Robot with a certain role
Attacker main
Attacker assist
...
etc.
...
Code that analyzes situation and calls action(s) accordingly
Action Handler
Contains functions to execute actions
Action: Move to target
...
Action: Shoot at goal
...
Action: Push
...

Figure 2.7: Current control flow of the Turtles.

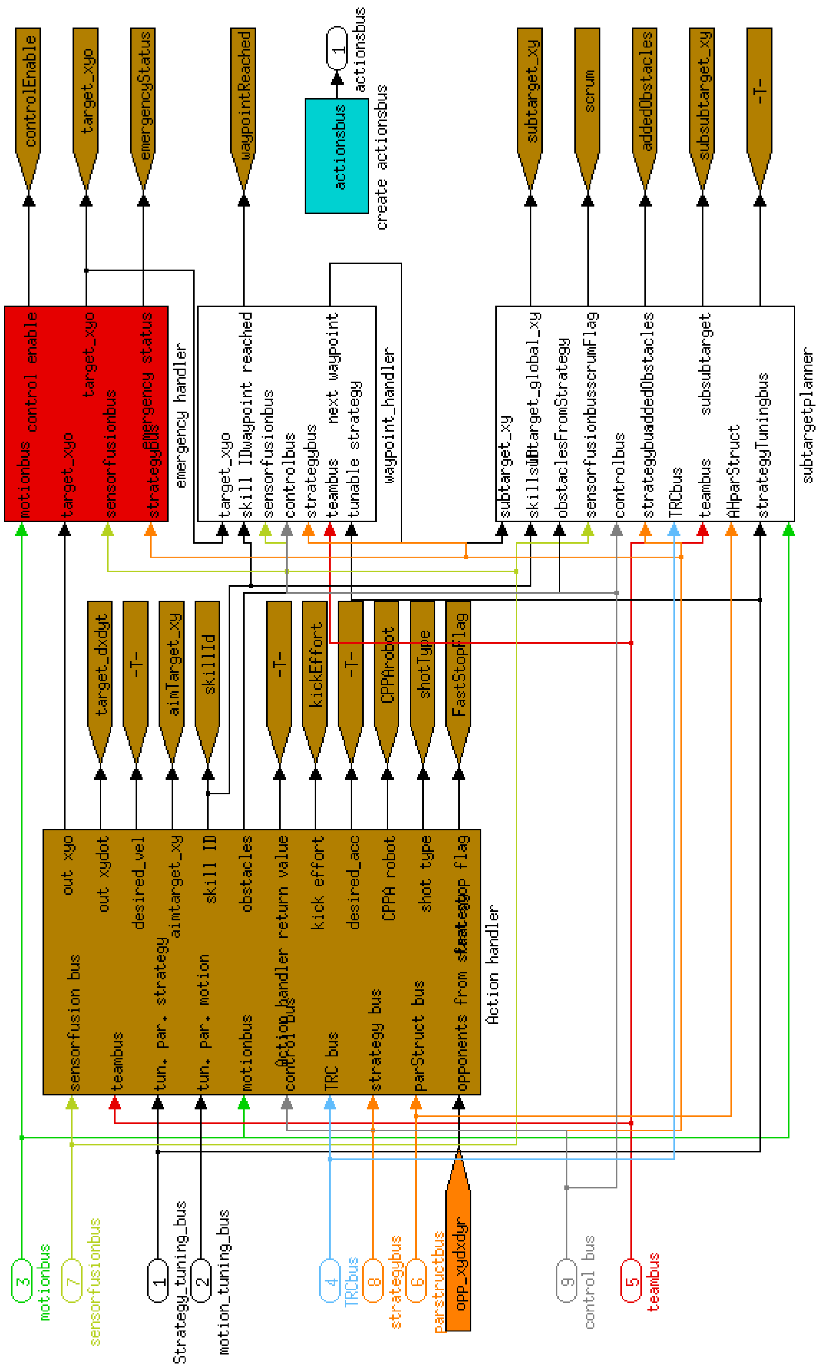


Figure 2.8: Actions scheme.

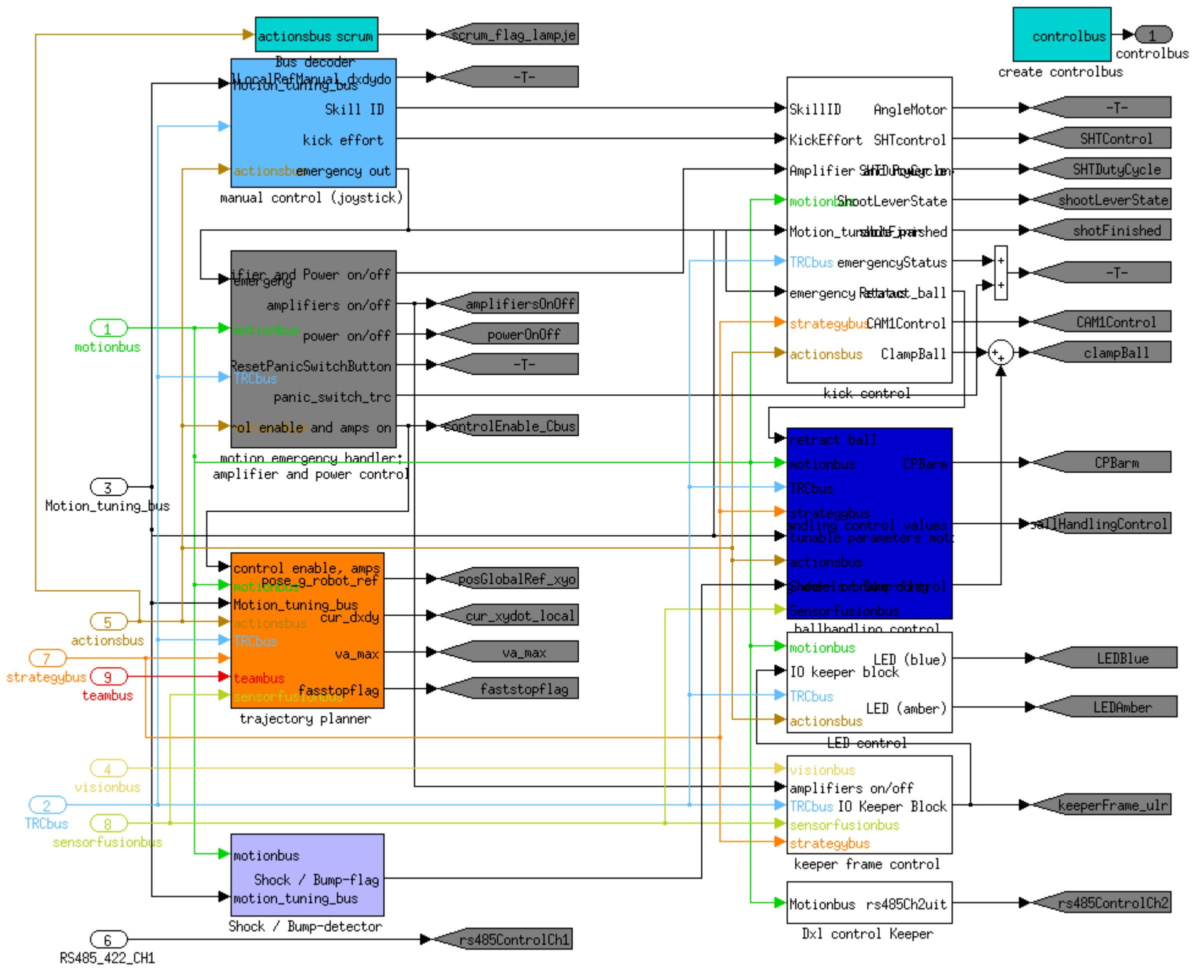


Figure 2.9: Control scheme.

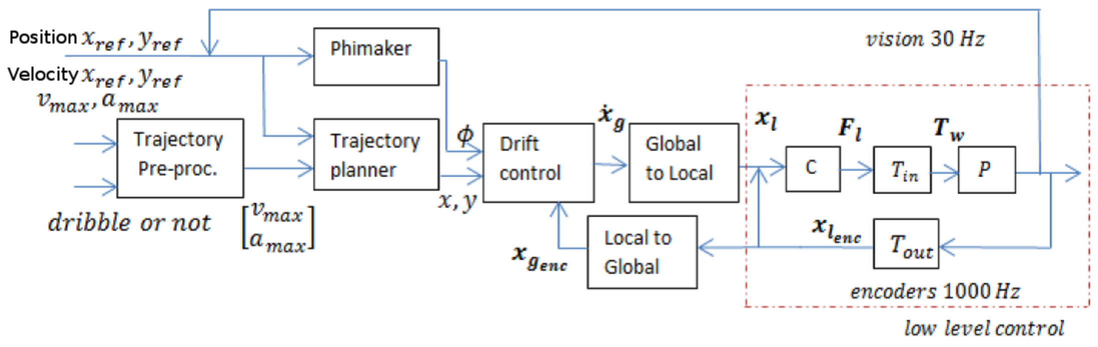


Figure 2.10: Simplified motion scheme as used in the Turtles. [15]

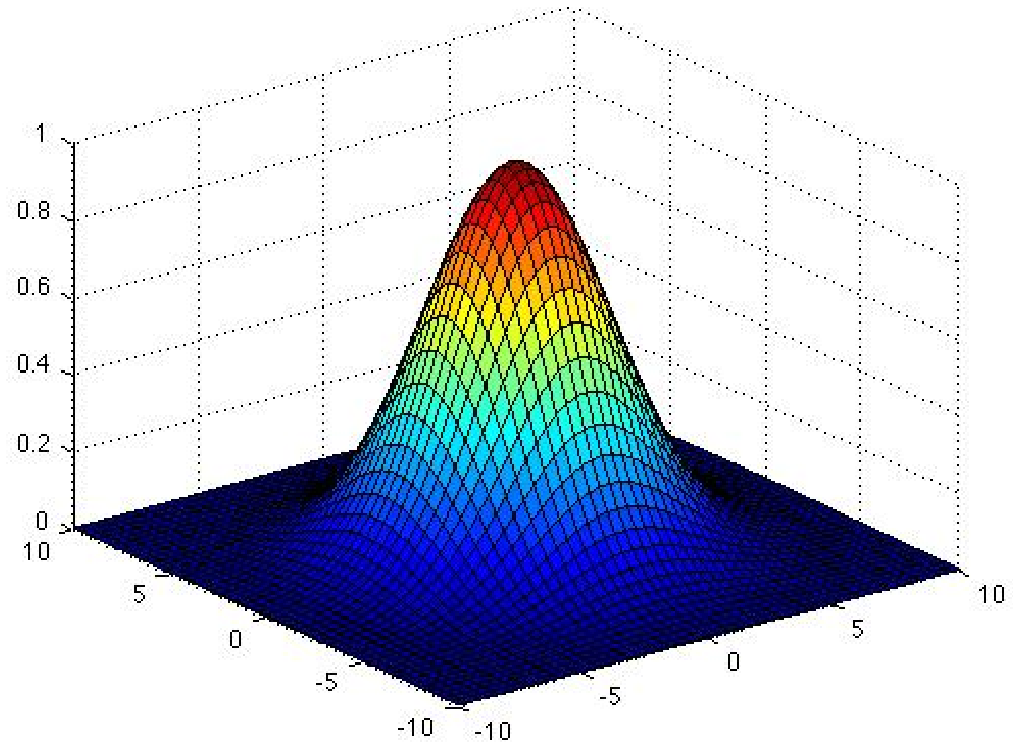


Figure 2.11: Example of a Gaussian bell function.

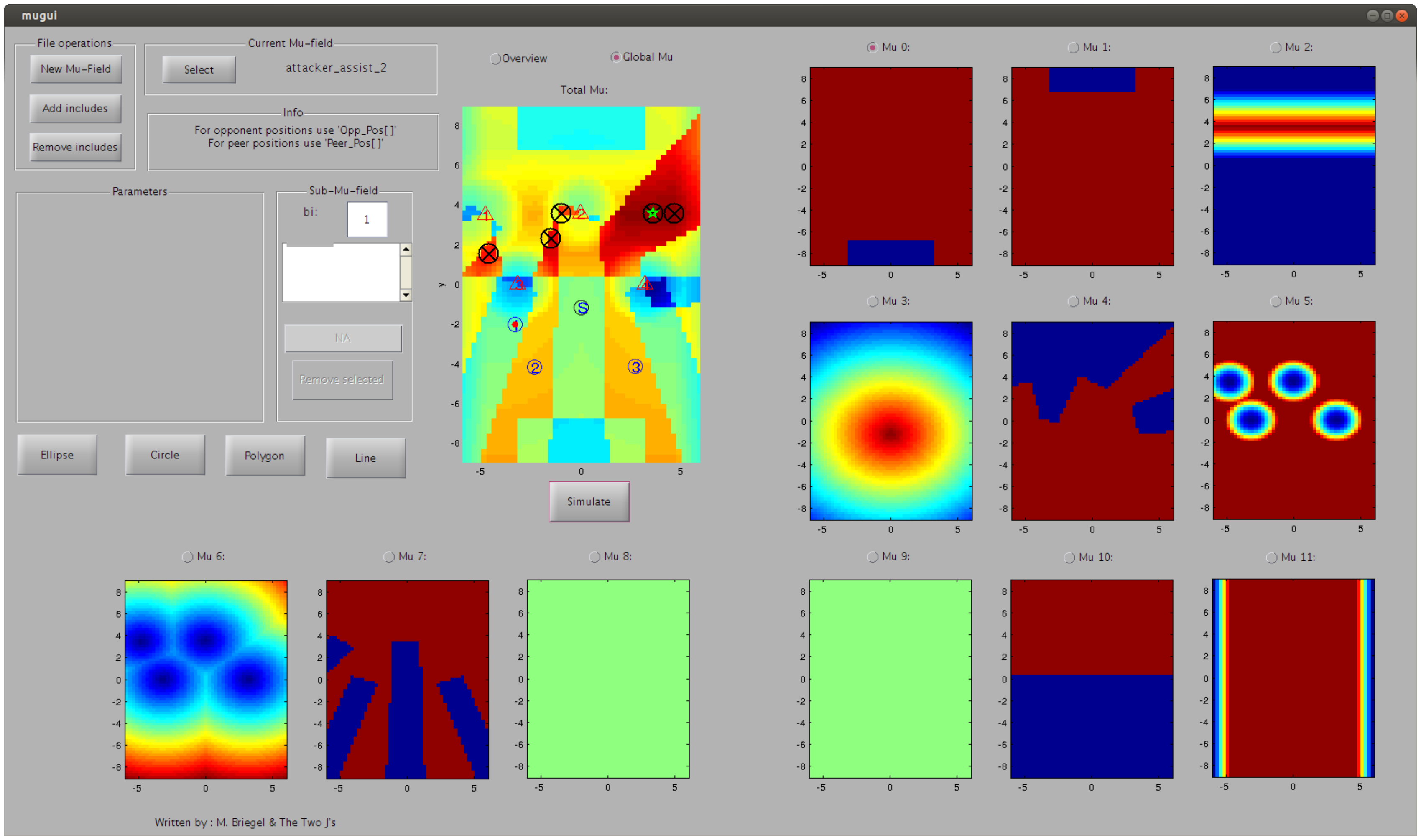


Figure 2.12: This figure shows the Attacker assist mu-field in the Tech United mu-field editor. The window ‘Total mu’ displays the result of adding all sub mu-fields 0-11 together. The blue circles, in the ’Total Mu‘-window, indicate the Turtles of the friendly team. The S indicates ‘self’, the Turtle that ran this mu-field generation. The red triangles indicate the robots of the opposing team and the red dot at Turtle 1 indicates the location of the ball. This field tries to find an attractive position for the Turtle indicated by S, which has the role ‘attacker assist’. The top five positions are indicated using the ⊗ symbol, the one that has the highest value is indicated using a star. This position is defined by the different sub mu-functions (mu-0 till mu-11) as a position close to the opponents goal having a free line of sight to the ball, and the opponents goal.

Chapter 3

Push

The push action is the controlled intercepting and pushing of a moving ball using the housing of a Tech United Eindhoven Turtle. This chapter is divided into a number of sections that explain for what goal and how push is implemented. First, in Section 3.1, the exact goals of the push action are described. Next, in Section 3.2, two types of push, static- and dynamic, are distinguished and the concept of pre-orienting is introduced. Finally, the software flow of the push action is described in Section 3.3.

3.1 Goals

The major goal of the push action is to increase the speed of play and thereby create more possibilities to score or pass. There are many situations from which the push action can be initiated. These situations are distinguished into two main groups, those during an active refbox task and those during normal play. Whenever there is a goal kick, free kick, throw in, penalty or corner given to the friendly team it is called an active refbox task. If the opposing team has been given one, it is a passive refbox task. During normal play there are two forms of the push action that can be performed, planned and opportunistic. A push action is called a planned push whenever a peer has the ball and will shoot it to a pushing Turtle. Opportunistic pushing is the pushing of a ball that is rolling free on the field or has been shot by the opponent, this is not implemented in the current version. For the RoboCup 2013 world championship the push has been implemented at the active corner situation, which is a form of planned push. The goal of the implementation is to score using the push action at an active corner refbox task, see Figure 3.1.

3.2 Types

Two types of push are distinguished:

1. Static push
   The turtle has an intercept velocity $\approx$ zero.
2. Dynamic push
   The turtle has an intercept velocity $>$ zero

Static vs. dynamic push The static push has as an advantage that the Turtle does not need a specific velocity at the moment of intercepting the ball. This is easier in terms of the control part of the problem. The static push has as a disadvantage that the velocity of the outgoing ball will always be lower than that of the incoming ball due to the loss of kinetic energy during the collision. Also, the Turtle has to come to a full stop at the intercept point, which takes time. Static

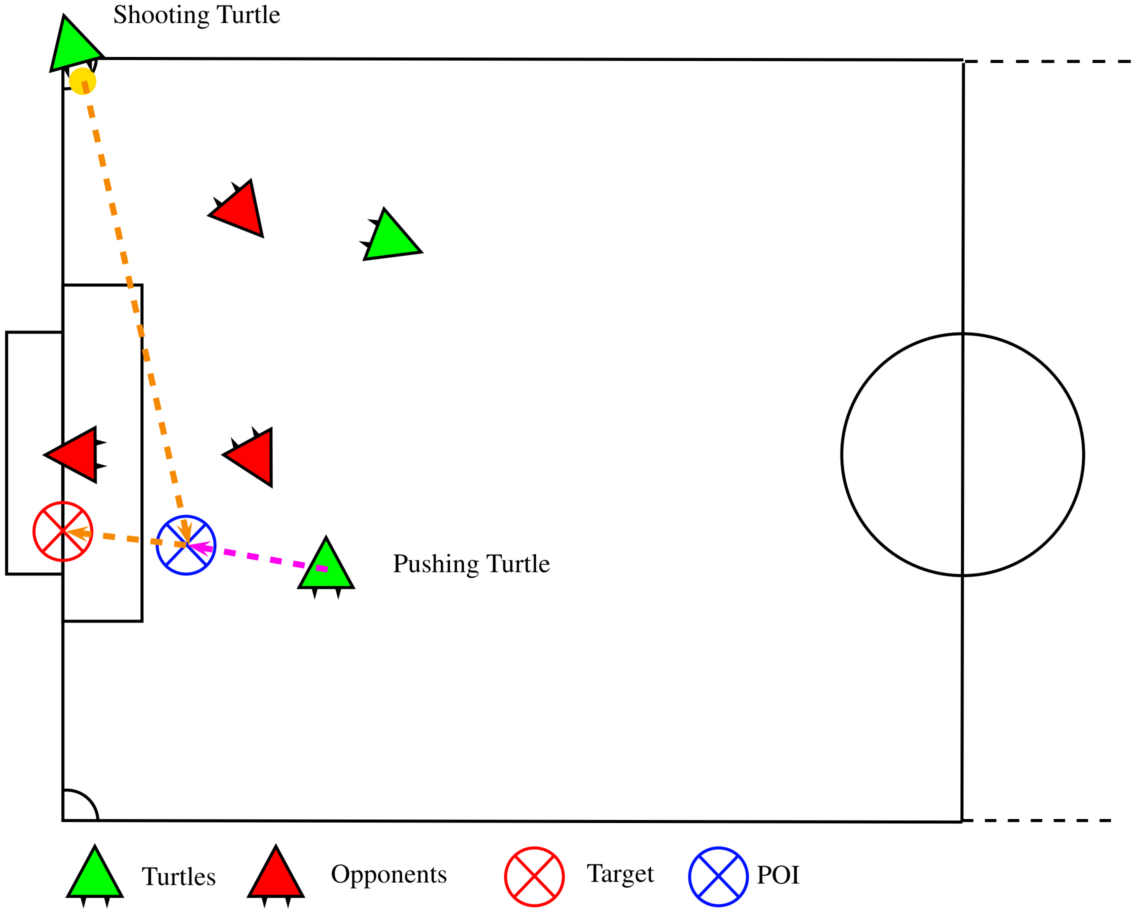


Figure 3.1: Situation sketch of the push action during the corner active refbox task. The Turtle that will give the indirect pass is called the shooting Turtle, the Turtle that performs the actual pushing is called the pushing Turtle. The two triangles at the robots indicate the location of the ball handling mechanisms.

push will, in the current implementation, only be used during pre-orientating. As a solution to the disadvantages of static push the dynamic push is introduced. During a dynamic push action the Turtle has an absolute velocity greater than zero at the moment of impact. This creates the ability to control, or at least increase, the outgoing ball velocity and removes the necessity to come to a full stop before intercepting.

Pre-orientating In situations where the ball has not yet been shot, the pushing Turtle has time to prepare, it will pre-orientate itself. While pre-orientating a Turtle will orientate itself, before the ball has been shot, such that its orientation is very close to what is required during the push. This is determined by calculating the static push orientation as if the ball is shot directly at the Turtle. A certain pre-orientation position can be determined and the Turtle will pre-orientate at this location. This makes it possible to prepare the pushing Turtle at a certain location for a planned push action.

3.3 Push software flow

The software flow is made out of two main parts, the control of the shooting Turtle (Section 3.3.1) and the control of the pushing Turtle (Section 3.3.2). If and when a push action is executed depends on the strategy of the Turtles, see Section 2.2.2. When push is being executed the push function is called at every iteration of the strategy software, which is running at 1000 Hz.

3.3.1 Shooting turtle

The shooting Turtle has to give a indirect pass that the pushing Turtle can intercept. The shooting Turtle first determines a point of intercept (POI) that can be shot at without hitting any obstacles and that can be reached by the pushing Turtle, see Figure 3.1. When a possible POI is found the so-called hard-conditions, which determine if a push is possible, are checked. After these checks are passed successfully, the shooting Turtle shoots the ball at the POI and is finished with the push action. The software that implements this is schematically represented in Figure 3.2. The separate sections of the software are explained in the following sub-sections.

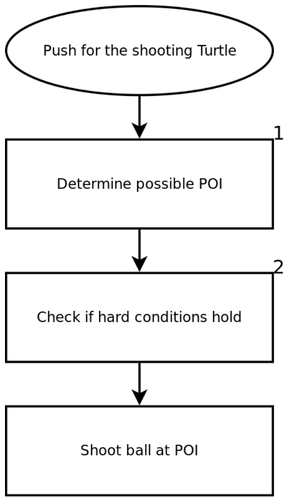


Figure 3.2: Schematic representation of the push functions for the shooting Turtle. The numbered actions in this diagram are explained in more detail in the following sections: determine possible POI (1) in Section 3.3.1.1 and checking the hard conditions (2) in Section 3.3.1.2.

3.3.1.1 Determining point of intercept

When determining a point of intercept it is assumed that the target is given by the strategy. When the intention is to score a goal, the target is set at an empty part of the goal and when passing it is set at a receiving Turtle. To simplify the problem the POI will be placed at the line from the pushing Turtle towards the target. Obviously this line should be free of obstacles. The Tech United software has a number of functions that can test for the presence of obstacles in a trapezium from a certain location towards another location. A trapezium is used because the outgoing angle of the pushed ball will always differ from the predicted angle because the accuracy is not 100%. The exact width, and rate of expansion, of this trapezium is a function of the accuracy.

At his point there are still a number of potential points of intercept at which the shooting Turtle can shoot the ball, which are visualized as blue dots in Figure 3.3. The POI that is ultimately selected depends on the ranking functions as described in Section 3.3.1.2. Due to the high number of potential points of intercept, the evaluation of the ranking at each point is performed using a push mu-field.

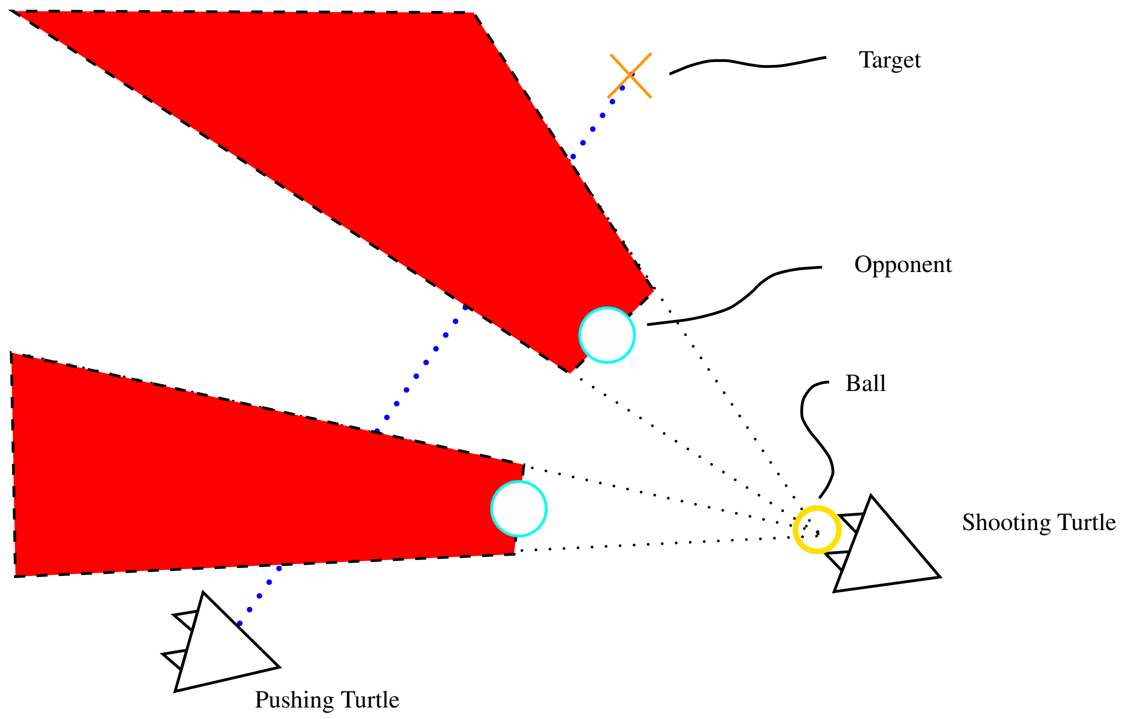


Figure 3.3: Schematic representation of the potential points of intercept at which the shooting Turtle can shoot. The blue dots, located on the line from the pushing turtle to the target, represent the potential points of intercept, the red areas are the areas that are blocked by opponents.

Push Mu-field To decide between the different potential POI, a special type of mu-field is developed, a mu-field on a line instead of the complete field. In Section 2.2.4 the previous implementation of the mu-fields is explained. It was already possible to calculate a mu-field over the complete field but this is too computationally intensive to be performed on-line. To be performed on-line a mu-field has to be calculated in less than the time required for one iteration of the motion executable, which runs at $1000Hz$, minus the time for other calculations that are performed. To be able to use the mu-fields they are handled in a separate thread which evaluates a complete mu-field in about one second. The mu-field library gives access to a number of useful functions which can be used to evaluate a situation on the playing field. This functionality is reused.

The mu-field library has been extended by adding the ability to calculate a mu-field not only on the complete field but by passing it a list of points describing a line or square of any size. If the number of points is kept low enough the calculations can be performed on-line.

This is not as generic as calculating a mu-field over any arbitrary shape but this limitation has been included because, for arbitrary shapes, the determination of the local maxima of the mu-field is too complex for the time available to implement it. Also there is currently no demand for mu-fields on an arbitrary shape.

In conjunction with the extra functionality the average computation time has been decrease by a factor of approximately 0.75. This speed improvement has been reached by removing a number of redundant memory copy operations and moving some calculations outside certain for loops.

The ranking for push is calculated on the line from the pushing Turtle towards the target location. The resolution of the points on this line can be adapted depending on the available computation time. After the complete line has been evaluated the largest local maximum is taken as the POI. This POI will be used as a target for the shooting Turtle.

3.3.1.2 Push ranking

During play a Turtle has to make decisions on what actions to execute. In this section the ranking function of the push action is described. There are a number of hard conditions, which result in a ranking of zero. If all hard conditions are met, the ranking subsequently depends on a number of partial ranking functions. These functions are partial because they are added together to form the complete push ranking.

Hard conditions The hard conditions are used to excluded situations in which it is obvious that push should not be executed.

- There has to be a free trajectory from the ball to the POI and from the pushing Turtle to the target.
  In Section 3.3.1.1 the POI is chosen such that these conditions hold. As time progresses however the situation on the field changes and it is possible that an opponent blocks the path of the ball. The option that an opponent is only crossing the path of the ball could be considered but this is currently not implemented. Therefore the push action will be aborted if there are any obstacles blocking the path of the ball at any time during the push.
- There has to be a free path from the pushing Turtle to the POI.
  If the path from the pushing Turtle to the POI is blocked push can not be performed. As with the previous hard condition, the POI is chosen such that this condition holds but the situation can change during execution of the action.
- The POI should be in the opponent half .
  It is not allowed to score from your own half by regulation.
- The pushing Turtle must be in a trapezium before the goal
  This condition prevents a push from the corner regions from where it is impossible or implausible to push the ball into the goal. The exact dimensions of this trapezium are tunable.
- The distance between the shooting Turtle and the pushing Turtle has to be above a certain threshold, but only before the shooting Turtle has shot the ball.
  If the shooting Turtle and the pushing Turtle are too close together the pushing Turtle will not have enough time to react.
- The absolute velocity of the pushing Turtle must be below a certain threshold, but only before the shooting Turtle has shot the ball.
  This condition is added because a straight intercept trajectory, see Section 4.4, will be used. When the turtle velocity is too large, the POI will shift too much during the push action. However, as soon as the ball is shot by the shooting Turtle the pushing Turtle will need to start moving and this condition should be ignored.

Partial ranking functions The partial ranking functions used for push are of the form of a linear equation, which is easily tunable

$$y(x) = \min\left(b,\, \max\left(0,\, -\frac{b}{a}\cdot x + b\right)\right), \tag{3.1}$$

in which y is the resulting partial ranking, x represents the measured variable (for example the distance), a equals the value of x at which the partial ranking y must be zero, b equals the maximal ranking that the measured variable can contribute to the complete push ranking. The minimal and maximal functions guarantee that the values will always be in the range [0, b]. For readability we introduce the variables λ for y, var_m for x, λ_{max} for b and var_{λ^0} for a yielding

$$\lambda_{mv}\left(var_m\right) = \min\left(\lambda_{max,\, mv},\, \max\left(0,\, -\frac{\lambda_{max,\, mv}}{mv_{\lambda^0}}\cdot var_m + \lambda_{max,\, mv}\right)\right), \tag{3.2}$$

in which mv indicates the different measured variables.

The sum off all maximal partial ranking functions

$$0.0 \geq \sum_{mv=1}^{\#mv} \lambda_{mv} \leq 1.0, \tag{3.3}$$

must be in the range [0, 1], this must be ensured by the programmer. For push the following variables: distance, push angle and required turtle orientation are taken into account.

Distance toward target Because the accuracy of a push action will never be 100%, the distance between the POI and the target determines, for a part, the ranking of a successful push. By substituting

$$\begin{aligned} var_m &= dist \\ \lambda_{mv} &= \lambda_{dist} \\ \lambda_{max,\,mv} &= \lambda_{max,\,dist} \\ mv_{\lambda^0} &= dist_{\lambda^0}, \end{aligned}$$

in (3.2) the partial distance ranking is calculated. In the resulting equation, *dist* equals the distance from the POI to the target.

Push angle The push angle, the shortest absolute angle between the incoming and outgoing trajectory of the ball see Figure 1.2, is another indicator for the success for the push action. If the push angle equals zero the ball will be pushed back to where it came from. This is the easiest situation because the used impact model, see Section3.3.2.3, agrees best with this situation. The timing of the Turtle being at the POI is less critical, the ball will still bounce of in the opposite direction.

The larger the push angle the larger the discrepancy between the impact model and the real situation and also the timing of the intercept will be more and more critical. By substituting

$$\begin{aligned} var_m &= |\varphi_{push}| \\ \lambda_{mv} &= \lambda_{\varphi_{push}} \\ \lambda_{max,\,mv} &= \lambda_{max,\,\varphi_{push}} \\ mv_{\lambda^0} &= \varphi_{push\,\lambda^0}, \end{aligned}$$

in (3.2) the push angle ranking is calculated. In the resulting equation φ_{push} equals the push angle between the incoming and outgoing ball trajectories.

Required pushing Turtle rotation The amount the pushing Turtle has to rotate in order to get one of its push surface at the correction orientation is yet another indicator for the successfulness of a push at that moment. For this function we take the absolute difference between the current Turtle orientation and the required orientation as the measured variable. When this difference is zero the Turtle already has the correct orientation and thus the ranking will be higher. By substituting

$$\begin{aligned} var_m &= |\varphi_{orient}| \\ \lambda_{mv} &= \lambda_{\varphi_{orient}} \\ \lambda_{max,\,mv} &= \lambda_{max,\,\varphi_{orient}} \\ mv_{\lambda^0} &= \varphi_{orient\,\lambda^0}, \end{aligned}$$

in (3.2) the required Turtle orientation ranking is calculated. In the resulting equation φ_{orient} equals the difference between the Turtle its current orientation and the required orientation.

3.3.2 Pushing Turtle

The pushing Turtle has to intercept and push the ball which the shooting Turtle has shot. The software that implements this is schematically represented in Figure 3.4. The separate sections of the software are explained in the following sub-sections.

3.3.2.1 Finished condition

The Turtle needs to detect if a push action has finished or if the action should be aborted. An action is finished if the ball has been intercepted and pushed. In the Tech United software an action normally indicates its state by returning a zero when the action is still in progress and a one as soon as the action is finished or aborted. There are four situations that can occur in which the push action is finished or should be aborted:

1. If the ball is intercepted, by the opponent, on the path from the shooting Turtle to the pushing Turtle, the action should be aborted.
   A flag in the Tech United software will indicate if the ball has been intercepted by the opponent.

2. If the ball is not intercepted by the pushing Turtle but the ball has moved past the POI, the action should be aborted.
   After the shooting Turtle has shot the ball, the POI will be updated continuously. It will be located at the interception point of the line from the pushing Turtle to the target and the trajectory of the ball. If the ball has moved past the POI there will be no interception point. This will be detected and the push action will be aborted.

3. If one of the hard conditions, as mentioned in Section 3.3.1.2, does not hold any longer, the action should be aborted.
   These hard conditions should always hold during the entire push, except for the last two that only need to hold before the shooting Turtle has shot the ball.

4. If the ball is successfully intercepted by the pushing Turtle, the action is finished.
   This check is implemented by measuring the distance between the Turtle and the ball. If this distance has been below a certain threshold and it increases again to above this threshold it is assumed that the ball has been intercepted and the push action is finished. The threshold is the radius of the circumference of the Turtle plus the diameter of the ball.

3.3.2.2 Predicting ball movement

An equation to predict the motion state of the ball is required. As our experience tells us, a ball will slow down when rolling along a surface. The ball will slow down due to friction with the surface and deformation of the surface and ball. As described in [7], there are two phases in the movement of the ball. In the first phase the ball is slipping and rolling while in the second phase there is only rolling. Air resistance is ignored due to its negligible influence.

In the first phase, the ball will lose some of its velocity due to the transfer of energy from forward momentum into angular momentum and due to deformation losses. In the second phase, the angular momentum is such that there is no slip between the ball and the surface. Therefore, only deformation losses will occur. Deformation losses will occur due to deformation of both the ball and the surface.

In [7] it is concluded that both the frictional component, due to slip, and the deformation component can be described using a constant factor. Ideally, both of these components are taken into account when predicting the ball location as a function of time. There is no way to measure

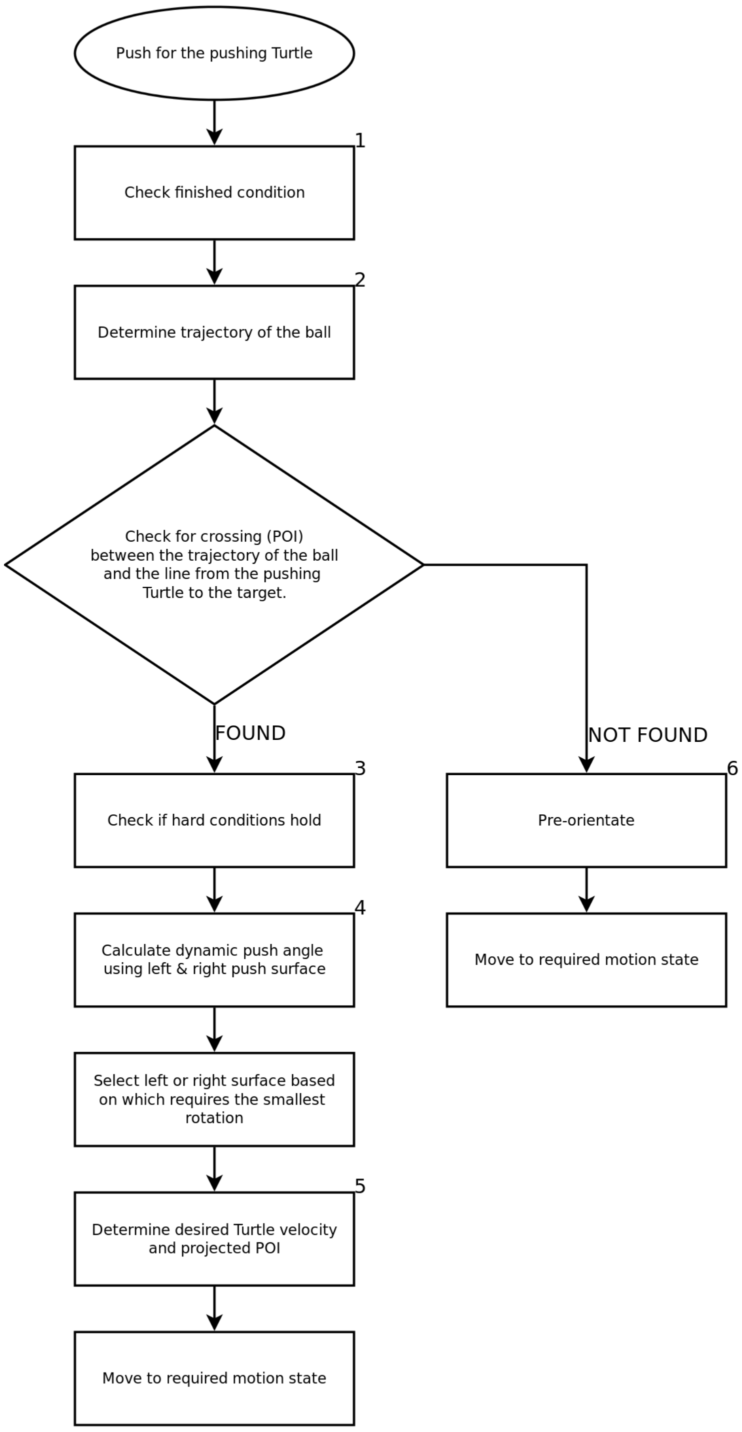

Push for the pushing Turtle
1
Check finished condition
2
Determine trajectory of the ball
Check for crossing (POI) between the trajectory of the ball and the line from the pushing Turtle to the target.
FOUND
NOT FOUND
3
Check if hard conditions hold
6
Pre-orientate
4
Calculate dynamic push angle using left & right push surface
Move to required motion state
Select left or right surface based on which requires the smallest rotation
5
Determine desired Turtle velocity and projected POI
Move to required motion state

Figure 3.4: Schematic representation of the push functions for the pushing Turtle. The numbered actions in this diagram are explained in more detail in the following sections: The finished condition (1) is explained in Section 3.3.2.1, the ball prediction (2) in Section 3.3.2.2, checking the hard conditions (3) is equal to that at the shooting Turtle, already explained in Section 3.3.1.2. Calculating the dynamic push angle (4) is described in Section 3.3.2.3, determining the desired Turtle velocity (5) is described in a separate chapter, Chapter 4, due to its complexity. Pre-orientating (6) has already been described in Section 3.2.

the rotation of the ball and thus no way to check in which phase a ball is in. Therefore, an equation will be used that is based on an average ball deceleration factor (*BDF*).

The value of this deceleration factor will between different playing fields, the exact amount of variance is not known. The deceleration factor will be extracted manually from measurements made by the Turtles. If the value varies too much between different fields, they have to be calibrated for every field manually or an algorithm should be developed which determines this value automatically, for example during a warm-up round at the beginning of a game.

It has been considered to take into account the spin of the ball, in any other direction than the rolling direction, because it could potentially alter the way the ball bounces of the Turtle. Because the ball will have no or little spin other than in its rolling direction and it is currently not possible to detect the spin using the Turtles it will not be considered in the model. It could however be of interest to apply spin to a pushed ball. In [16] a model is developed for two billiard balls colliding and experiments are performed in order to compare the model with the reality. In this model the assumption is made that the balls have a coefficient of restitution of one. Therefore, the model is not directly applicable to the ball-Turtle collisions. They do however include the friction between the ball and the surface which influences the spin.

When given a ball with a certain motion state MS_b its position and velocity can be predicted given a certain Δt and (*BDF*) as follows. The ball either comes to a stop or has some non-zero final velocity after Δt. The time t_{stop} needed to come to a full stop

$$t_{stop} = -(|v_b(0)| / BDF), \tag{3.4}$$

in which the minus sign counters the negative sign of the *BDF* and $|v_b(0)|$ is the current absolute velocity. If $\Delta t \geq t_{stop}$ the ball comes to a stop before Δt. In this case t_{stop} needs to be substituted for Δt in (3.5) and (3.6).

The distance that a ball will travel along its trajectory

$$dist(\Delta t) = |v_b(0)| \cdot \Delta t + 0.5 \cdot BDF \cdot (\Delta t)^2, \tag{3.5}$$

can be used to determine the position of the ball after Δt by first creating a vector of *dist* length and rotating this vector such that it is parallel with the trajectory of the ball, and adding this rotated vector to the current location. The resulting ball velocity

$$|v_b(\Delta t)| = |v_b(0)| + BDF \cdot \Delta t, \tag{3.6}$$

is translated into its components at Δt in the same manner.

Of these functions a generic ball prediction library has been created for easy use in the Tech United software. A number of other functions have also been included, in Listing 3.1 an overview of these functions can be found.

```
// These functions do not include obstacles
// Predict ball motion state (position and velocity) delta t in the future
ball_t      predictBallMotion(ball_p ball, double dt);

// Calculate the time a ball needs, to travel a certain distance
// dist_reached equals the distance actually traveled (this can be used
// in the case the ball comes to a standstill before traveling dist
double      calcBallTime(ball_p ball, double dist, double* dist_reached);

// Returns a line over the trajectory of the ball of 2x fieldlength
line_t      getBallTrajectory(ball_t ball);

// Ball bounce physics
vector_t    ballBouncePhysics(vector_p V1, vector_p Vr, double surfacePhi);
```

Listing 3.1: Ball Prediction Library

The last function, *ballBouncePhysics(...),* contains the equations deducted in Section 3.3.2.3 which calculates the outgoing ball trajectory given an incoming velocity vector, Turtle velocity vector and the rotation of the push surface. The functionality of the rest of the functions is explained in their accompanying comments.

3.3.2.3 Push physics

In order be able to intercept and push a moving ball a prediction of its position and velocity is required. The physics involved and an equation for predicting the ball motion state can be found in Section 3.3.2.2.

An impact model is needed in order to predict and control the result of a push action. The previous implementation of the push action, by J.W. Lamers [8], used the reflection law as a model. The reflection law states that the angle $\varphi_{in} = \varphi_{out}$, this does not take into account the loss of energy during a collision or the velocity of the pushing Turtle. This results in incorrect predictions of the trajectory of the ball after a push action. Therefore an impact model that predicts the behavior of the ball more accurately is described. Next, a reversed impact model is deducted. This reversed model calculates the required Turtle velocity vector and orientation at the moment of impact given an incoming and outgoing ball velocity vector. The reverse model returns a solution, but not necessarily the easiest in terms of the difficulty for a Turtle to reach the computed motion state.

To make use of this reverse model the ability to control the Turtle such that it has the required motion state, including its velocity, is needed. The cubic polynomial trajectory planner described in Section 4.3 is able to do this. However, the cubic polynomial trajectory planner proofed problematic to implement in practice. Therefore, an alternative method has been used which is described in Section 4.4. This method requires a function that calculates the required Turtle orientation given an incoming and outgoing velocity vector of the ball as well as the velocity vector of the Turtle. This function proved very hard to find analytically, therefore an algorithm has been designed that solves this problem in a brute force manner.

Ball-Turtle collision physics In [5] a literature survey on the subject of impact dynamics has been performed. According to this survey, the main interest using impact theory is usually the relationship between the velocities (before and after impact) and/or the impact force with its resulting material deformation. The velocities before and after impact can be found using a ‘classical mechanics’ model, which uses fundamental laws to predict the velocities after impact. Here, the assumption is made that the colliding objects are perfectly rigid, consequently the impact duration will be zero. The kinetic energy loss during an impact is modeled by means of the normal coefficient of restitution (COR) e_n, first introduced by Newton and used to solve many practical problems, examples of which can be found in [2]. The use of simple laws makes the model relatively easy to implement, however the COR will usually have to be determined experimentally.

If the assumption that the colliding objects are perfectly rigid results in a discrepancy between the model and reality which is too large, or when the impact forces have to be calculated, the simplified COR model cannot be used. During real collisions the impact time will always be strictly greater than zero, which led to the development of continuous-time dynamic models. Although these models provide a more complete analysis, they are much more complex and it is difficult to get correct values for all the involved parameters. Therefore, the authors of [5] suggests that when a system has to be modeled, while the main interest is the relation between the velocities before and after impact, the COR model is usually sufficient. Because the goal of the model during a push action is to determine the velocity of the ball after impact, with respect to the velocity of the ball before impact, the COR model will be used.

Coefficient of restitution model In [5] the COR model is described in more detail. Besides the normal COR, e_n, the notion of the tangential COR, e_t, is introduced. The tangential coefficient is used to relate tangential velocities before and after impact which results in a more complete model. Applying the model to the situation of a ball impacting with the push surface of a Turtle, we can define the initial velocity and final velocity of the ball and Turtle as being the velocities just before

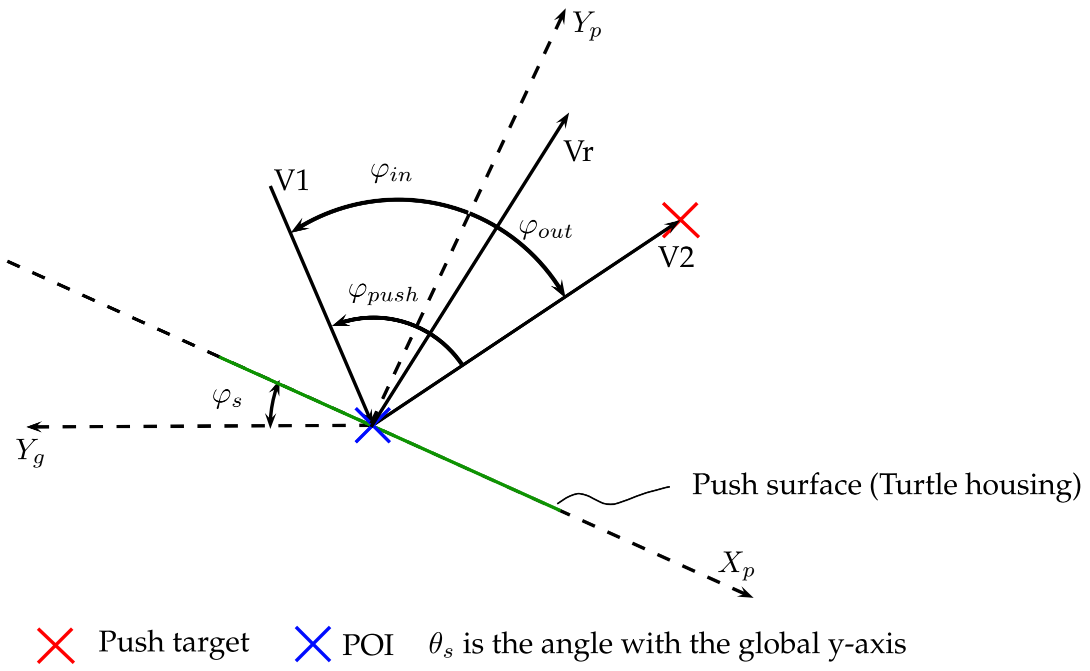


Figure 3.5: Push physics, $V1$ incoming ball velocity, $V2$ outgoing ball velocity, Vr Turtle velocity. All at the time of impact. The push coordinate system is positioned such that the origin lies at the impact point, and its x-axis is aligned with the push surface.

and just after the push. The assumption is made that the Turtle has a sufficient high weight, such that the kinetic energy transferred to the Turtle is negligible. This implies that the velocity of the Turtle before and after impact will be identical. The model states that these velocities are related by the COR e_n and e_t. The accuracy of the model largely depends on the accuracy of e_n and e_t which have to be found experimentally. In [3] methods are presented to measure the COR.

According to [5] both coefficients are not constant with respect to the impact velocity, however with relatively low impact velocities the hypothesis is that this effect is relatively small such that it can be neglected. To make sure this is indeed the case, an experiment, in which multiple measurements with varying impact velocities, while keeping other variables constant, has been performed. The results of this experiment can been found in Section 3.3.2.3.

Using the COR model an equation can be derived to calculate the outgoing ball velocity $V2$ as a function of the incoming ball velocity $V1$ and Turtle velocity Vr at the moment of impact, see Figure 3.5. Because the normal and tangential coefficient of restitution are defined with respect to the push surface, the push coordinate system is used in these calculations. Before using (3.13) and (3.14), the global velocity vectors and coordinates have to be translated into the push coordinate system using a rotation and translation. The relations between the push and global velocity vectors are as follows,

$$\begin{aligned}
V1_{xp} &= \cos(\varphi_s) \cdot V1_{xg} - \sin(\varphi_s) \cdot V1_{yg} && (3.7)\\
V1_{yp} &= \cos(\varphi_s) \cdot V1_{yg} + \sin(\varphi_s) \cdot V1_{xg} && (3.8)\\
V2_{xp} &= \cos(\varphi_s) \cdot V2_{xg} - \sin(\varphi_s) \cdot V2_{yg} && (3.9)\\
V2_{yp} &= \cos(\varphi_s) \cdot V2_{yg} + \sin(\varphi_s) \cdot V2_{xg} && (3.10)\\
Vr_{xp} &= \cos(\varphi_s) \cdot Vr_{xg} - \sin(\varphi_s) \cdot Vr_{yg} && (3.11)\\
Vr_{yp} &= \cos(\varphi_s) \cdot Vr_{yg} + \sin(\varphi_s) \cdot Vr_{xg} && (3.12)
\end{aligned}$$

The COR model consists of two main equations, one used to calculate the resulting x-component and one for the resulting y-component of $V2$. The x-component

$$V2_{xp} = e_t \cdot (V1_{xp}), \tag{3.13}$$

in which the x subscript indicates the velocity over the x-axis, and p indicates the use of the push coordinate system. Notice that the x-velocity of the Turtle does not affect the resulting x-velocity of the ball. The used model does not incorporate the friction between the ball and the robot, therefore the x-velocity of the Turtle is assumed to have no influence on the final speed of the ball. In contrast, the resulting y-velocity is indeed influenced by the velocity of the Turtle. The resulting y-velocity

$$V2_{yp} = e_n \cdot (Vr_{yp} - V1_{yp}), \tag{3.14}$$

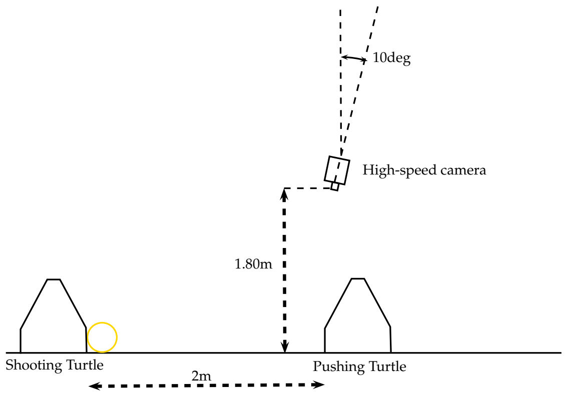
in which the y subscript indicates the velocity over the y-axis. The sign of the ball its y-velocity is inverted because its momentum will be reversed after impacting with the Turtle.

Looking at these equations it seems that there is no possibility to control $V2_{xp}$, but this is not the case. To be able to control $V2_{xp}$ the angle φ_{in} has can be altered, this can only be achieved when looking at the global coordinate system by altering the rotation of the push surface φ_s. This effectively changes the incoming angle φ_{in} and thereby $V1_{xp}$ and $V2_{xp}$ in the local coordinate system. This implies that the control of the velocity parallel to the local x-axis is decoupled from the control on the local y-axis. However, when looking at the global coordinate system a change in the global x-velocity will alter the x- and y-velocity in the push coordinate system.

Determining coefficients To determine the coefficients of restitution an experiment has been conducted. In this experiment the ball is shot at one of the push surfaces of a Turtle with varying speeds, while the velocity before and after the impact are measured. Only the normal coefficient of restitution has been determined in this manner, the tangential coefficient has been determined by tuning during testing until the ball hits its intended target.

Experiment setup Two Turtles are used during the experiment, one shooting Turtle and one pushing Turtle. The Turtles are positioned such that the shooting Turtle shoots at the pushing Turtle at a perpendicular angle to its push surface. The distance between the two Turtles has been fixed at 2 meters. A ruler with $5cm$ increments is positioned on the ground, also on a perpendicular angle of the push surface. A high-speed camera, filming at 300 frames per second, has been placed directly above the pushing Turtle at a height of $1.80m$. The camera is looking down and forward with an angle of 10 degrees from the vertical. Figure 3.6 shows the schematic representation and the camera view of the experimental setup.

Data Analysis The experiment generates a lot of data to analyze due to the high frame rate of the high-speed camera. Therefore, the data analyzing process has been automated using a Matlab script. This script first asks the user to mark the ruler bars on the screen. The measurement bars are entered from top to bottom setting the first ruler as distance 0. Because the distance between the bars is known, a function translating screen coordinates to a real world relative distance is obtained by fitting a quadratic polynomial through the measurement bars, see Figure 3.7. The script analyses the generated movies frame by frame, it determines the screen position of the ball in every frame by using the standard Matlab function 'imfindcircles'. The center point of the returned circle is used as the screen position of the ball which is transformed into real world positions using the fitted function. See Figure 3.8 for a visualization of the measurement bars and the ball finding algorithm.



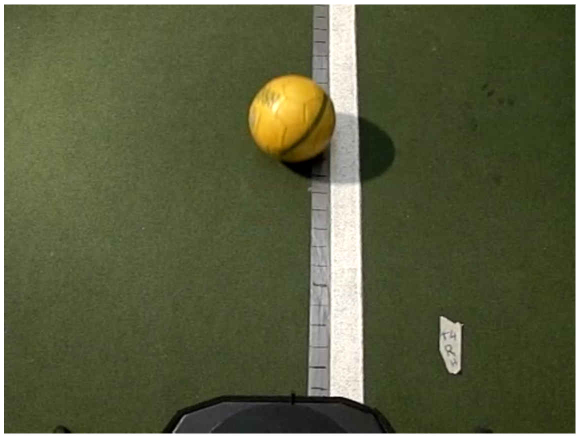


Figure 3.6: Left: Schematic representation of the COR experiment, right: Image from the point of view of the high-speed camera.

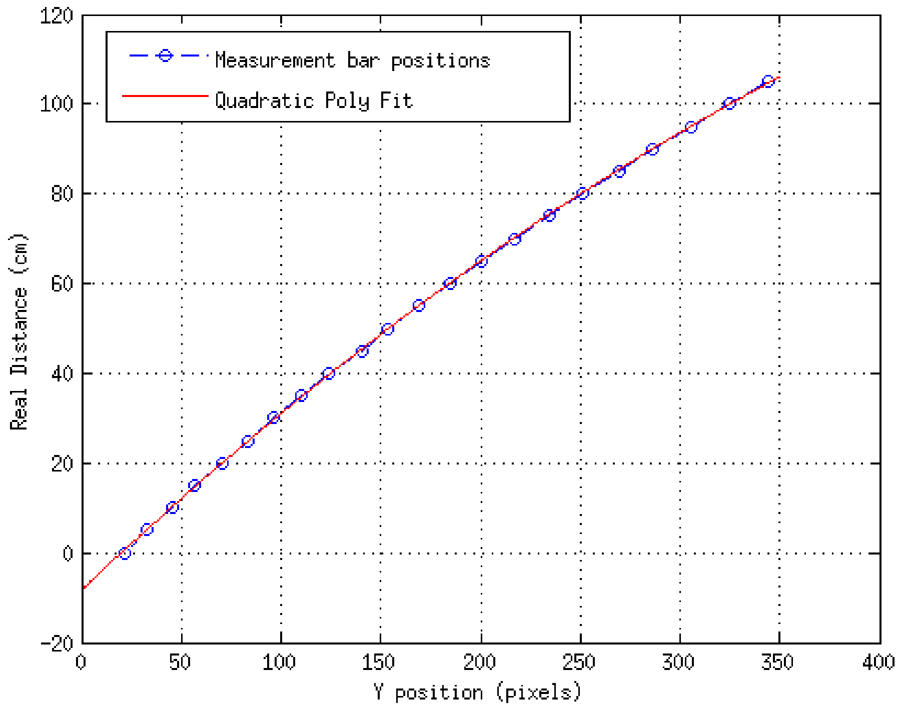


Figure 3.7: Polynomial fit through the positions of the measurements bars relating the screen y pixel coordinate to the real distance.

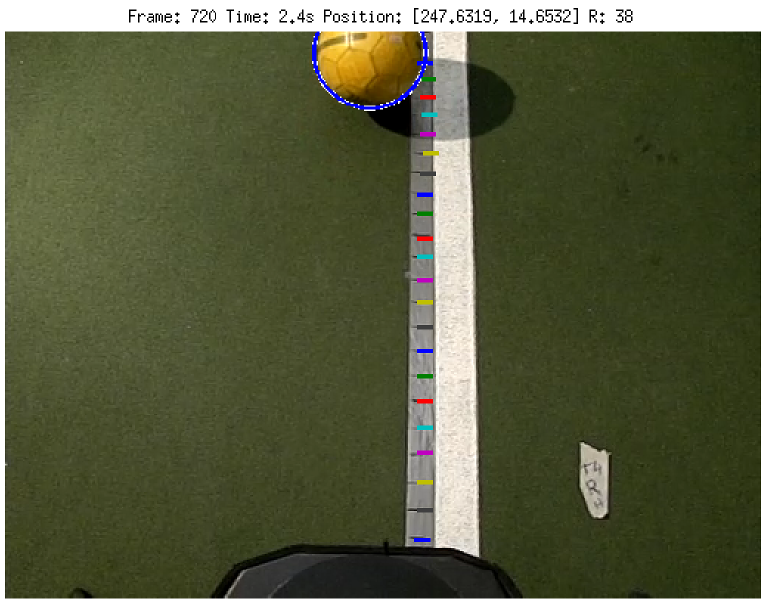


Figure 3.8: Virtual measurement bars overlain on top of the real measurement bars. The top most bar is taken as the origin. The real measurement bars are spaced at a regular distance of $5cm$. The circle around the ball is the visualization of the ball finding algorithm.

The difference of the ball position between two frames is

$$\Delta Q_i = Q_{ball,\,i} - Q_{ball,\,i-1}, \tag{3.15}$$

with $Q_{ball,\,i}$ the relative position of the ball at frame i. The time between every frame is known, $1/300$ seconds. Therefore the velocity of the ball can be calculated by dividing the difference in position between two frames by the time between two frames

$$v_{ball,\,i} = \frac{\Delta Q_i}{1/300}, \tag{3.16}$$

with $V_{ball,\,i}$ the velocity of the ball at frame i.

Noise is present at the position signal of the ball due to inaccuracy of the circle finding algorithm. Therefore the distance is filtered using a moving average filter (MAF) with a window size of 20 samples after which the the velocity, that is calculated using the filtered distance signal, is also filtered by a MAF of size 20. resulting in the plot shown in Figure 3.9. The ball is not moving perfectly flat over the floor but bounces a little. This bouncing, together with the algorithm inaccuracy, causes the velocity variations which are seen.

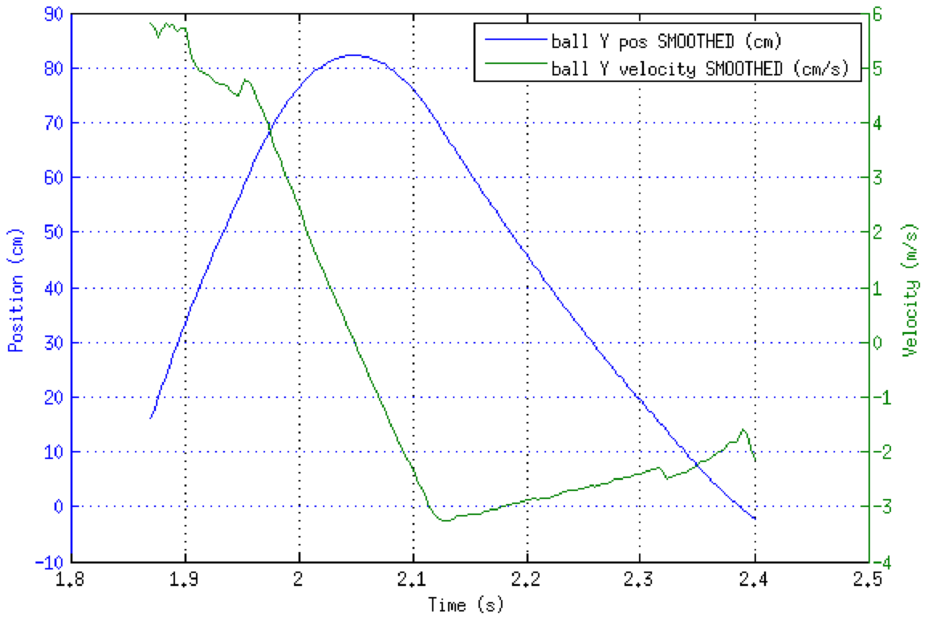


Figure 3.9: Position and velocity plot of the ball. The positive part of the velocity is before and the negative part after impacting the Turtle.

The surface and the ball deform during impact, therefore the impact duration is non-zero. This effect, together with the smoothing effect of the moving average filter, causes the transition of the velocity before and after the impact. In order to determine the normal coefficient of restitution the velocities of the ball, just before and just after impact, are compared. These moments are determined manually as the moments just before and after the steep descent in velocity.

Results The experiment shows a fairly constant normal COR with varying velocity, see Figure 3.10. The normal COR of all measurements is averaged ($COR_n = 0.71$) and used in the COR model.

Reversed coefficient of restitution model Using the COR model we can calculate $V2$ as a function of $V1$ and the motion state of a Turtle. The main interest when pushing and when the velocity of the outgoing ball has to be controlled is the exact opposite, the ability to calculate the required motion state of the Turtle as a function of $V1$ and $V2$.

$V1$ represents the velocity of the ball before the push and is thus known. $V2$ can be derived as soon as an intercept location P_i, a point on the trajectory of the incoming ball, and a target motion state are known. The exact mathematics differ depending on whether static or dynamic push is used. With static push only the target location is of importance. A vector is created from the intercept location towards the target location

$$V2_{static} = \mathbf{p_t} - \mathbf{p_i}, \tag{3.17}$$

in which $\mathbf{p}_t$ is the position of the target and $\mathbf{p}_i$ is the position of the intercept location. The magnitude of this vector is ignored because only the direction is of importance. With dynamic push an absolute velocity is given, the vector is scaled to have the correct magnitude

$$V2_{dynamic} = (\mathbf{p_t} - \mathbf{p_i}) \cdot \frac{v_t}{\|(\mathbf{p_t} - \mathbf{p_i})\|}, \tag{3.18}$$

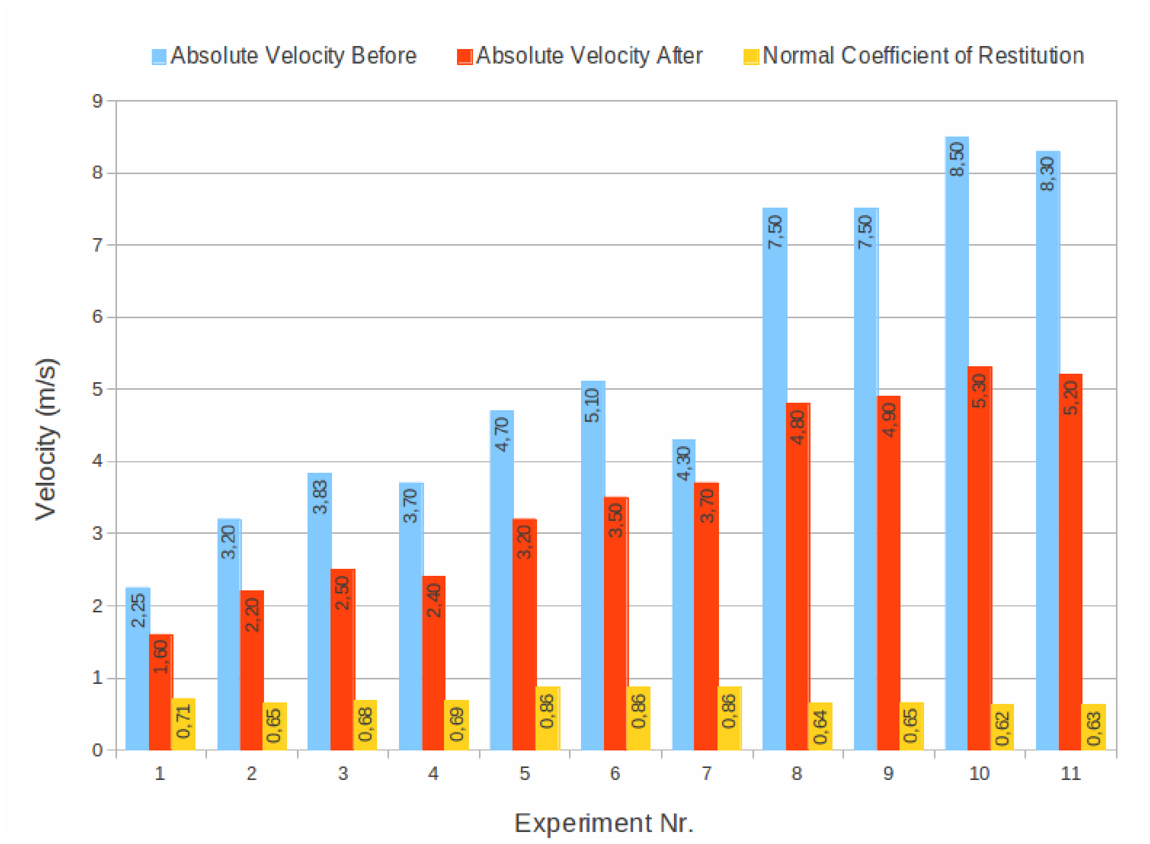


Figure 3.10: Measurement results of the normal Coefficient of Restitution, the velocity before the impact is shown in blue, the velocity after impact in red and the normal COR in yellow. The average normal COR is 0.71.

in which v_t is the absolute velocity value at the target. Now that $V2$ is known, the required motion state of the Turtle can be calculated. For the static push only the rotation of the Turtle is of importance, for dynamic push the rotation and the velocity vector of the Turtle have to be calculated. In [?] a solution has been found for the static push, but this solution does not apply here due to the introduction of the coefficients of restitution. Due to the addition of these coefficients, the rule that $\varphi_{in} = \varphi_{out}$ does not apply.

Static push To calculate the required orientation for static push an analytical solution has not been found. Instead a lookup table (LUT) is build which can be found to find the required Turtle orientation.

The LUT is build by applying the COR model to a number of vectors with varying φ_{in}. A loop will apply all rotations in the range $[0, \pi]$ with a certain step size. The step size determines the accuracy of the LUT. For every φ_{in} the COR model will return a corresponding φ_{out} the shortest difference between these two angles φ_{push} will be stored in the LUT.

When a static push action is executed the incoming and outgoing trajectories of the ball are known, therefore φ_{push} is known. Using the LUT the corresponding φ_{in} can be found. The orientation of the Turtle will be set such that the incoming ball trajectory has the same angle φ_{in} as was found using the LUT.

Dynamic push With dynamic push a degree of freedom is gained due to the fact that the velocity of the ball can be influenced using the velocity of the Turtle. As explained in section

3.3.2.3 the x-velocity in the push coordinate system will not influence the velocity of the ball, but when looking at the global coordinate system we can influence this push x-velocity by altering the rotation of the Turtle. The required rotation is calculated substituting (3.7) and (3.9) in (3.13)

$$\cos(\varphi_s) \cdot V2_{xg} - \sin(\varphi_s) \cdot V2_{yg} = e_t \cdot (\cos(\varphi_s) \cdot V1_{xg} - \sin(\varphi_s) \cdot V1_{yg}) \tag{3.19}$$

This equation can be solved for φ_s

$$\varphi_s(V1_{xg}, V1_{yg}, V2_{xg}, V2_{yg}, e_t) = \tag{3.20}$$

$$\begin{cases} 0 \\ if \quad (V2_{xg} - e_t \cdot V1_{xg}) = 0 \\ -2 \cdot \tan^{-1}\left(\frac{V2_{yg} + \left(e_t^2 \cdot V1_{xg}^2 + e_t^2 \cdot V1_{yg}^2 - 2 \cdot e_t \cdot V1_{xg} \cdot V2_{xg} - 2 \cdot e_t \cdot V1_{yg} \cdot V2_{yg} + V1_{xg}^2 + V2_{yg}^2\right)^{1/2} - e_t \cdot V1_{yg}}{V2_{xg} - e_t \cdot V1_{xg}}\right) \\ if \quad (V2_{xg} - e_t \cdot V1_{xg}) \neq 0 \end{cases},$$

in which a case distinction is made in order to prevent a division by zero if no rotation is needed.

With this equation the angle of the push surface φ_s, which will result in the correct x-velocity of the ball, is calculated. φ_s will be used in the following equations to calculate the velocity vector of the Turtle. Substituting (3.10), (3.8) and (3.12) in (3.10)

$$\cos(\varphi_s) \cdot V2_{yg} + \sin(\varphi_s) \cdot V2_{xg} = \tag{3.21}$$

$$e_n \cdot [(\cos(\varphi_s) \cdot Vr_{yg} + \sin(\varphi_s) \cdot Vr_{xg}) - (\cos(\varphi_s) \cdot V1_{yg} + \sin(\varphi_s) \cdot V1_{xg})],$$

results in an equation with two unknowns Vr_{xg} and Vr_{yg}. To solve for Vr_{xg} and Vr_{yg} another equation is required. This equation is obtained by setting the push x-velocity of the Turtle to zero

$$Vr_{xl} = 0, \tag{3.22}$$

and substituting (3.11)

$$\cos(\varphi_s) \cdot Vr_{xg} - \sin(\varphi_s) \cdot Vr_{yg} = 0 \tag{3.23}$$

Combining (3.21) and (3.23) results in two equations with two unknowns. This can be solved for Vr_{xg} and Vr_{yg}

$$Vr_{gx}(\varphi_s, V1_{xg}, V1_{yg}, V2_{xg}, V2_{yg}, e_n) = \tag{3.24}$$

$$\frac{1}{e_n} \cdot \left[\frac{(V2_{yg} \cdot sin(2\varphi_s))}{2} + V2_{xg} \cdot sin(\varphi_s)^2\right] + \left[\frac{(V1_{yg} \cdot sin(2\varphi_s))}{2} + V1_{xg} \cdot sin(\varphi_s)^2\right],$$

and

$$Vr_{gy}(\varphi_s, V1_{xg}, V1_{yg}, V2_{xg}, V2_{yg}, e_n) = \tag{3.25}$$

$$\frac{1}{e_n} \cdot [cos(\varphi_s) \cdot (V2_{yg} \cdot cos(\varphi_s) + V2_{xg} \cdot sin(\varphi_s))] + cos(\varphi_s) \cdot [V1_{yg} \cdot cos(\varphi_s) + V1_{xg} \cdot sin(\varphi_s)]$$

Thus for dynamic push, (3.20) can be used to calculate the rotation of the push surface φ_s. This φ_s is then entered into (3.21) and (3.23) to calculate the corresponding velocity vector for the Turtle. In the plots in Figure 3.11, a couple of examples are shown. In these examples the variables E_t, E_n, $V1$, $V2$ are entered in the above equations and the motion state of the Turtle is calculated. This is subsequently fed into the COR model and the resulting orientation of the push surface and the resulting ball velocity are shown. Note that the resulting ball velocity vector is plotted on top of the $V2$ vector, which can therefore not be seen if the results are correct.

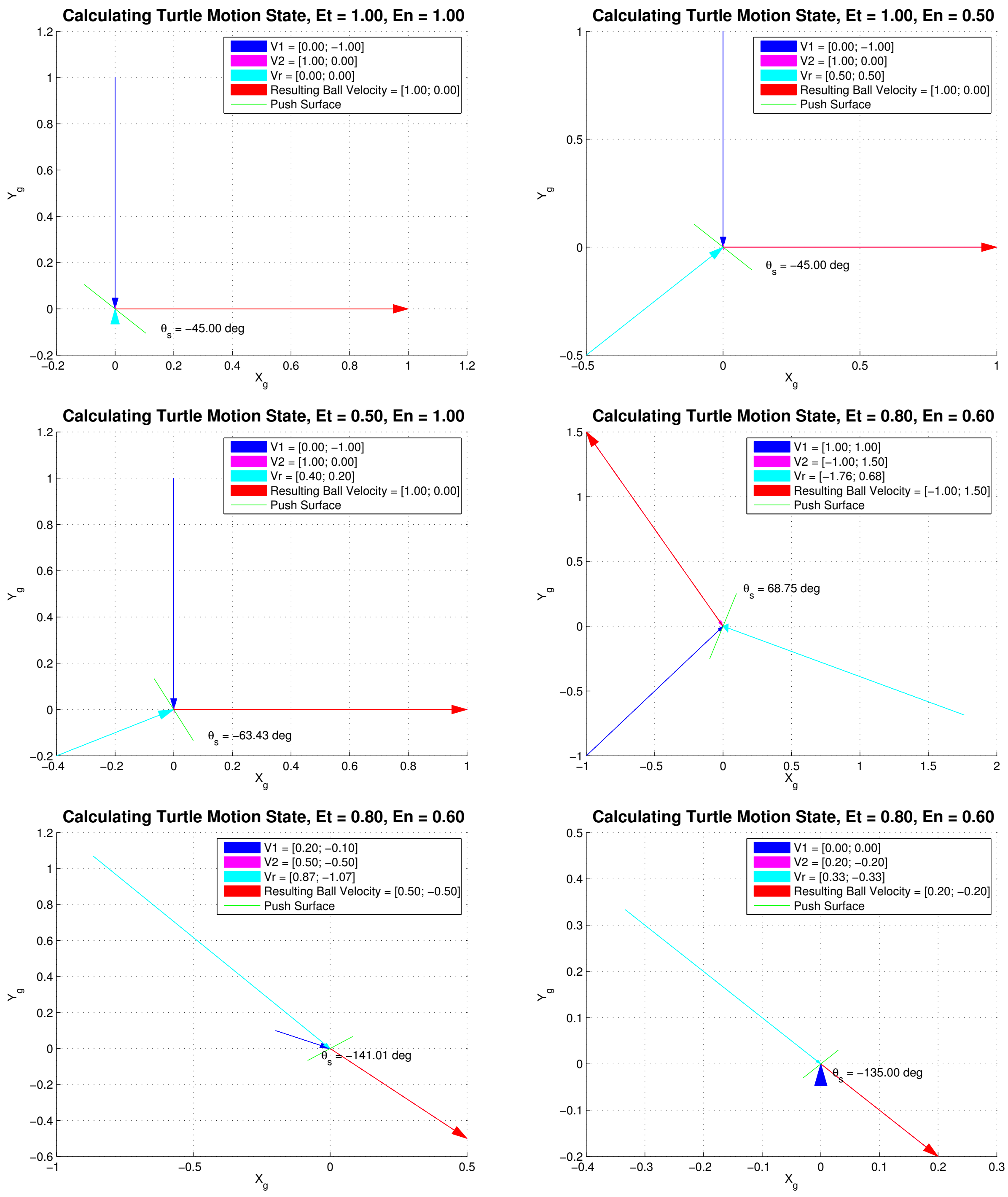


Figure 3.11: Examples of Turtle motion states required to obtain a given outgoing ball trajectory using push.

Brute force algorithm To be able to find the angle of the push surface φ_s while $V1$ and $V2$ as well as Vr are given, a brute force algorithm has been implemented. The algorithm tries to minimize the difference between the angular deviation between the outgoing ball vector $V2$ and the required vector. The velocity of the ball is thus not controlled by this method. A brute force algorithm has been implemented because no analytical method has been found.

The brute force algorithm, see Algorithm 3.1, loops through all push surface angles in the range $[0, \pi]$, using a certain step size, and returning the push surface angle at which the difference between the required and predicted outgoing ball vector is minimal.

Algorithm 3.1 Brute Force Dynamic Push Angle

```
V1 = incoming ball velocity vector
V2 = required outgoing ball velocity vector
Vr = Turtle impact velocity
smallestDiff = inf;
smallestAngle = 0.0;
for(angle = 0 until Pi step angleSTEP)
        V2temp = CORmodel(V1, V2, Vr, angle)
        angleDiff = getAngleDifference(V2, V2temp)
        if(angleDiff < smallestDiff)
                smallestAngle = angle
next
return smallestAngle
```

The step size of φ_s can be decreased or increased to achieve a higher or lower accuracy respectively. The number of loops which are executed depends on the step size, therefore a higher accuracy results in a long execution times and vice versa. To measure the influence of a varying step size, an experiment has been performed. In this experiment the algorithm is executed 10000 times using a varying incoming ball trajectory, the Turtle velocity and required outgoing ball trajectory have been kept constant. An initial incoming ball velocity is chosen to be the vector $\mathbf{V1} = \begin{bmatrix} 1.0 \\ 0.0 \end{bmatrix}$. The ball trajectory is then rotated by taking the initial $V1$ and rotating it by 0.01 times the current iteration number in radians.

The experiments are executed on an Asus N56V with Intel Core i7-3610 2.3 GHz processor and 6 GB DDR3 SODIMM running Ubuntu 12.04. In Figure 3.12 the mean, minimal and maximal execution times are shown. As expected the execution times are smaller with larger step sizes. The resulting discrepancy between the required and calculated surface angle can be found in Figure 3.13. As a reference Figure 3.14 shows the offset resulting from a certain difference in push angle.

Recommended resolution The recommended resolution is chosen such that at a distance of $3m$, a distance around which most push actions will be performed, the maximal offset is a factor 5 smaller than the average accuracy measured with the step size at 0.1. The average offset, see Section 5.1, is $0,14m$ thus the step size has to be chosen such that the angular offset

$$offset_{\varphi_s} < \tan^{-1}\left(\frac{offset_{dist}}{dist}\right), \tag{3.26}$$

in which $offset_{dist}$ is the maximal offset at distance $dist$, is smaller than the offset at $\frac{0.14}{5} = 0.028m$. Substituting $offset_{dist}$ and $dist$ in (3.26) yields

$$\begin{aligned} offset_{\varphi_s} &< \tan^{-1}\left(\frac{0.028}{3}\right) \\ &< 0.0093, \end{aligned} \tag{3.27}$$

the value of the maximal angular offset. From Figure (3.13) we can see that in order to achieve this value a step size of approximately 0.01 is required.

Surface offset and ball diameter correction The rotation and positioning of the push surfaces, with respect to the center point of the Turtle, is ignored up to this point. A simple translation and rotation are used to correct for this, see Figure 3.15.

The assumption has been made that the ball is a point mass, in reality the ball has a certain diameter. The Turtle has to move a distance equal to the radius of the ball into the direction of the normal of the push surface, to make sure that the ball will hit the Turtle at the moment the center of the ball is at the POI, see Figure 3.16.

Whenever a POI is determined, the motion state, to which the Turtle has to move, is equal to this POI but translated by

$$\overline{\mathbf{t_t}} = \overline{\mathbf{t_f}} + \overline{\mathbf{t_b}}, \tag{3.28}$$

in which $\overline{t_t}$ is the total translation, $\overline{t_f}$ the fixed translation and $\overline{t_b}$ the ball translation, and rotated by φ_f depending on the push surface and the diameter of the ball.

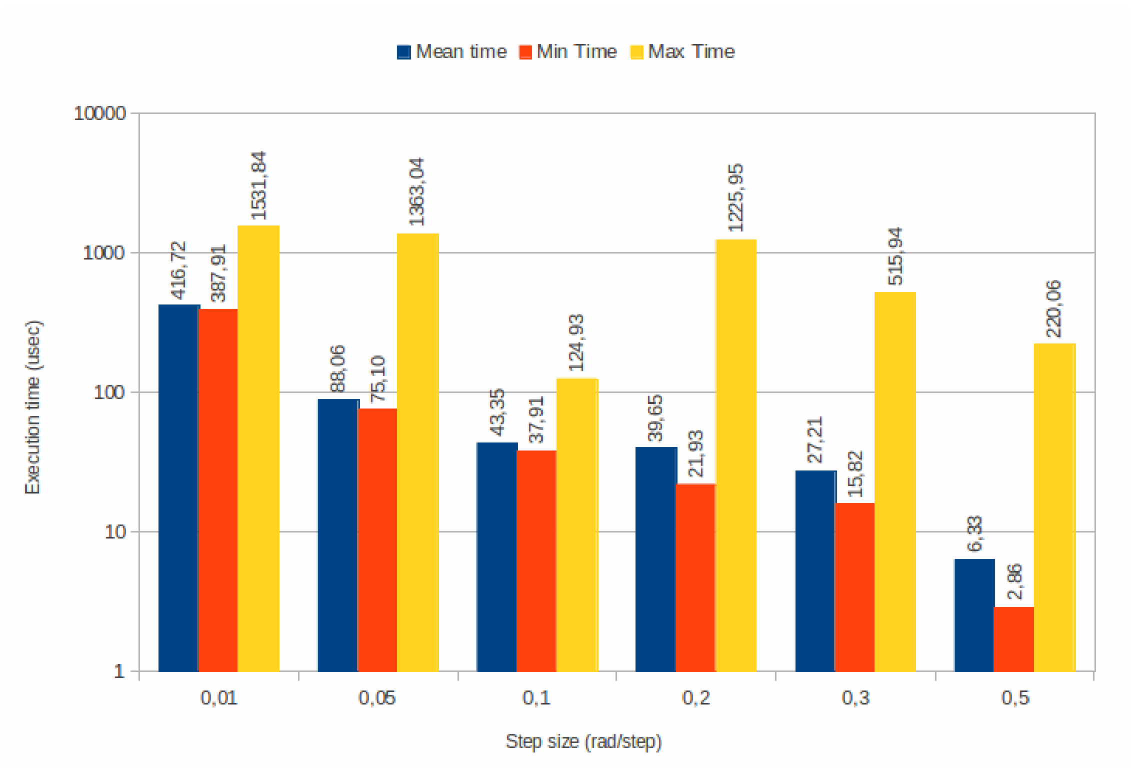


Figure 3.12: Execution times as a result of different φ_s step sizes.

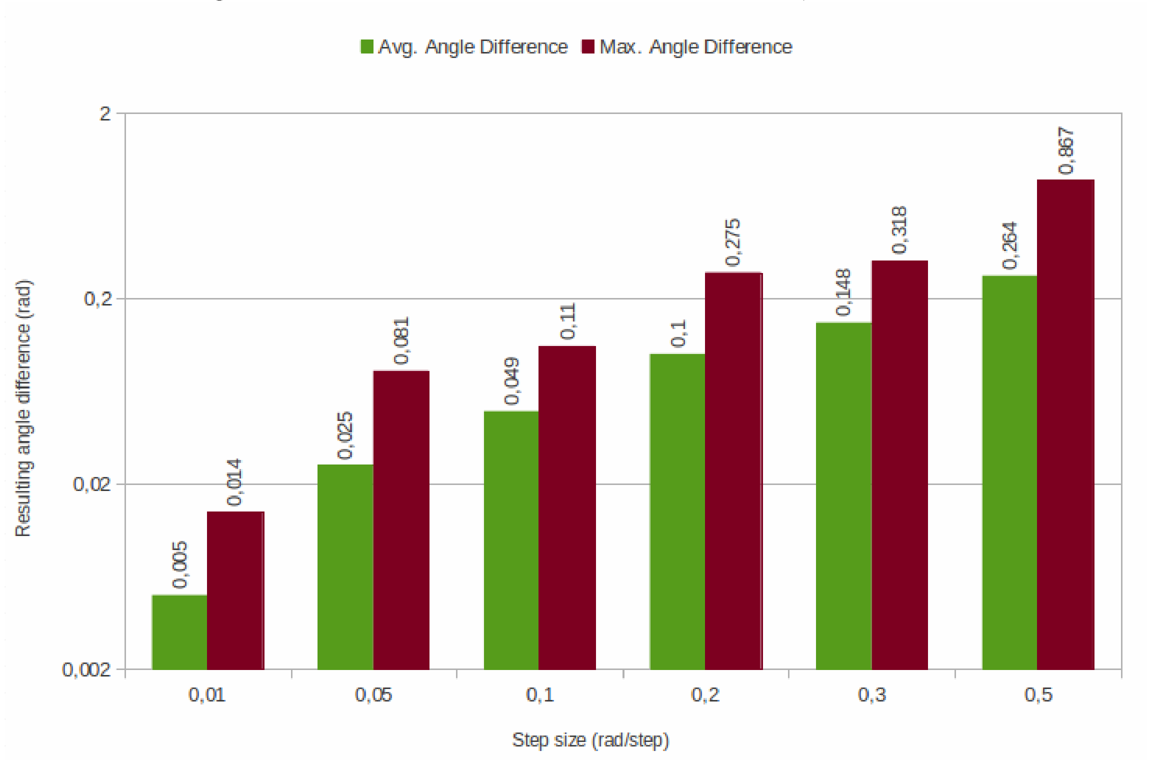


Figure 3.13: Average and maximal radial difference between required and calculated push angle at different φ_s step sizes. The resulting offset at the target location, due to these differences, can be found in Figure 3.14.

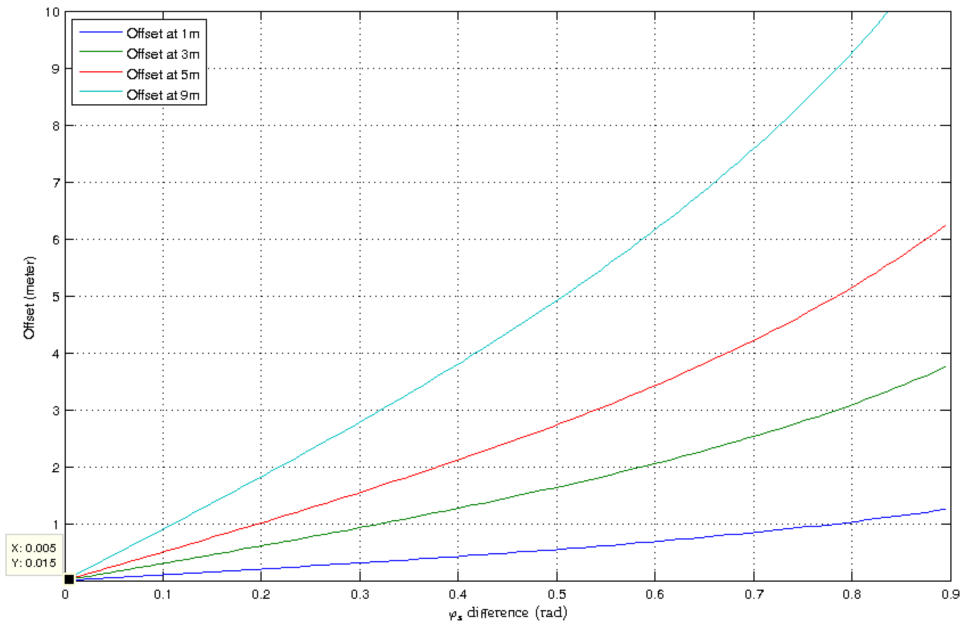


Figure 3.14: Offset at several target distances given a certain push angle difference. The indicator is positioned on the offset at $3m$ line.

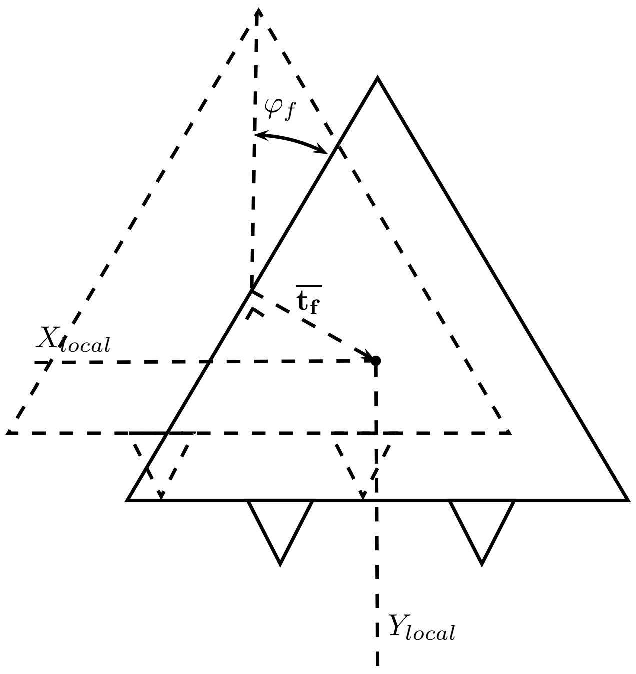


Figure 3.15: The fixed translation $\overline{\mathbf{t_f}}$ and rotation $\varphi_\mathbf{f}$ to correct for the offset of the push surface.

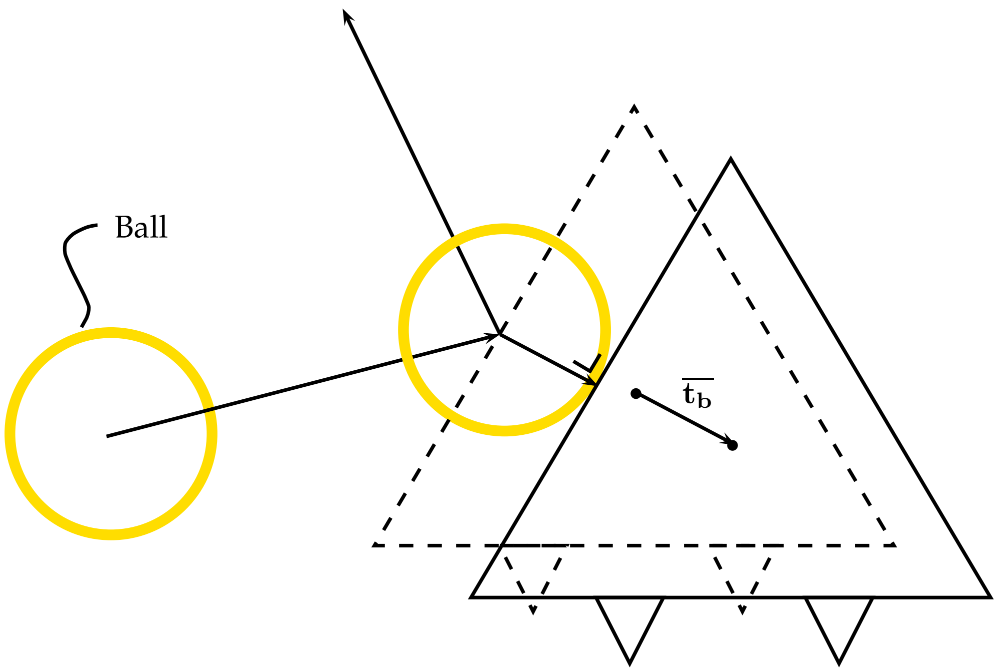


Figure 3.16: The translation of the Turtle to correct for the diameter of the ball $\overline{\mathbf{t_b}}$.

Chapter 4

Turtle movement

The movement of the Turtles is separated into two stages. The trajectory planner which generates a trajectory, and the low-level motion control that follows it. We will only discuss the trajectory planner and keep the controller as it is. First, the current implementation of the trajectory planner is discussed. Secondly, the capabilities which are required of a trajectory planner used for push are stated and are compared with the capabilities of the current controller. Next, a comparison between different methods for trajectory generation is made and the cubic polynomial method is explained in more detail. Finally, a simple but elegant solution for the trajectory generation is shown that uses the currently implemented trajectory planner.

4.1 Trajectory planner requirements

Looking at the push action there are a number of requirements that can be formulated for the trajectory planner. These requirements follow from the fact that the Turtles need to have a certain motion state at the time of intercepting a ball. The requirements that push asks from the trajectory planner are:

1. Controlling OR predicting the time of arrival of the Turtle.
   There is only one moment in time that a moving ball is at a certain location. Therefore being able to control, or at least predict, the time of arrival is of uttermost importance.

2. Having a non-zero velocity at the final motion state.
   In order to have control over the outgoing ball velocity, control over the velocity of the Turtle at the moment of impact is needed.

3. Real-time performance.
   Because all calculations have to be performed during the robot soccer game in a real-time environment the required calculation time has to be sufficiently low.

The current trajectory planner is able to predict the arrival time of the Turtle at a final motion state, it cannot control the arrival time. The final motion state cannot have a non-zero final velocity and is therefore not usable for the push action. The on-line performance requirement is obviously met by the current controller because it is being used successfully at the moment.

4.2 Trajectory generation

There are many different methods to generate trajectories [1, 4, 6, 9]: linear, parabolic, trigonometric, polynomial, trapezoidal, combinations of those and many more. The current planner, see Section 2.2.3, is a trapezoidal trajectory planner.

Usually a trajectory is defined by specifying initial, intermediate and final conditions on: time, position, velocity and acceleration. At the initial, intermediate and final points some or all of these conditions are given. A relatively easy method to generate these trajectories is the polynomial method. A polynomial function going through these points can easily be formulated using the equations explained in Section 4.3. This method proved to be successful at generating trajectories but some problems arose during practical tests, see Section 4.3.3. This was the reason to develop another very simple but effective method that works by only calculating the desired Turtle velocity at every iteration. This velocity limiting method, which uses the already available trajectory generator, is described in Section 4.4.

4.3 Cubic polynomials

The order of the polynomial depends on the number of points and the number of conditions at each point. For n points and m_i conditions at point i, a$(\sum_{i=1}^{n} m_i) - 1$ order polynomial is required to describe the desired path. For every point at the very least both the $time(n)$ and one other property have to be specified. In practice it is possible to generate a trajectory through as many points as needed with constraints on the time, position, velocity, acceleration, jerk, etc. but in practice these high order polynomials will lead to large oscillations in between the defined points. Therefore, using $n-1$ polynomials with a degree $(p < n-1)$ is preferred over a polynomial of degree $n-1$. The $n-1$ polynomials will then each represent a section of the complete trajectory. For the push action a movement from an initial point to the POI is required in a specified time frame. At both points the position and the velocity are specified.

4.3.1 1D polynomial trajectories

To be able to specify the time, position and velocity at an initial and final point a cubic polynomial is required. The trajectory between two points, $Q(t_0)$ and $Q(t_f)$, is specified using this third order polynomial

$$q(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3, \tag{4.1}$$

in which a_0, a_1, a_2 and a_3 are the unknowns that have to be solved. By differentiating (4.1), the velocity and acceleration can be calculated

$$\begin{aligned} \dot{q}(t) &= a_1 + 2a_2 t + 3a_3 t^2 && (4.2)\\ \ddot{q}(t) &= 2a_2 + 6a_3 t, && (4.3) \end{aligned}$$

$2n$ constraints are needed to solve these equations. These are obtained by specifying the position and velocity at the start time t_0 and the end time t_f

$$\begin{aligned} q(t_0) &= a_0 + a_1 t_0 + a_2 t_0^2 + a_3 t_0^3 && (4.4)\\ \dot{q}(t_0) &= a1 + 2a_2 t_0 + 3a_3 t_0^2 && (4.5)\\ q(t_f) &= a0 + a_1 t_f + a_2 t_f^2 + a_3 t_f^3 && (4.6)\\ \dot{q}(t_f) &= a_1 + 2a_2 t_f + 3a_3 t_f^2 && (4.7) \end{aligned}$$

Equations(4.4) till (4.7) are combined into the following matrix equation

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 0 & 1 & 2t_0 & 3t_0^2 \\ 1 & t_f & t_f^2 & t_f^3 \\ 0 & 1 & 2t_f & 3t_f^2 \end{bmatrix}^{-1} \begin{bmatrix} q(t_0) \\ \dot{q}(t_0) \\ q(t_f) \\ \dot{q}(t_f) \end{bmatrix}, \tag{4.8}$$

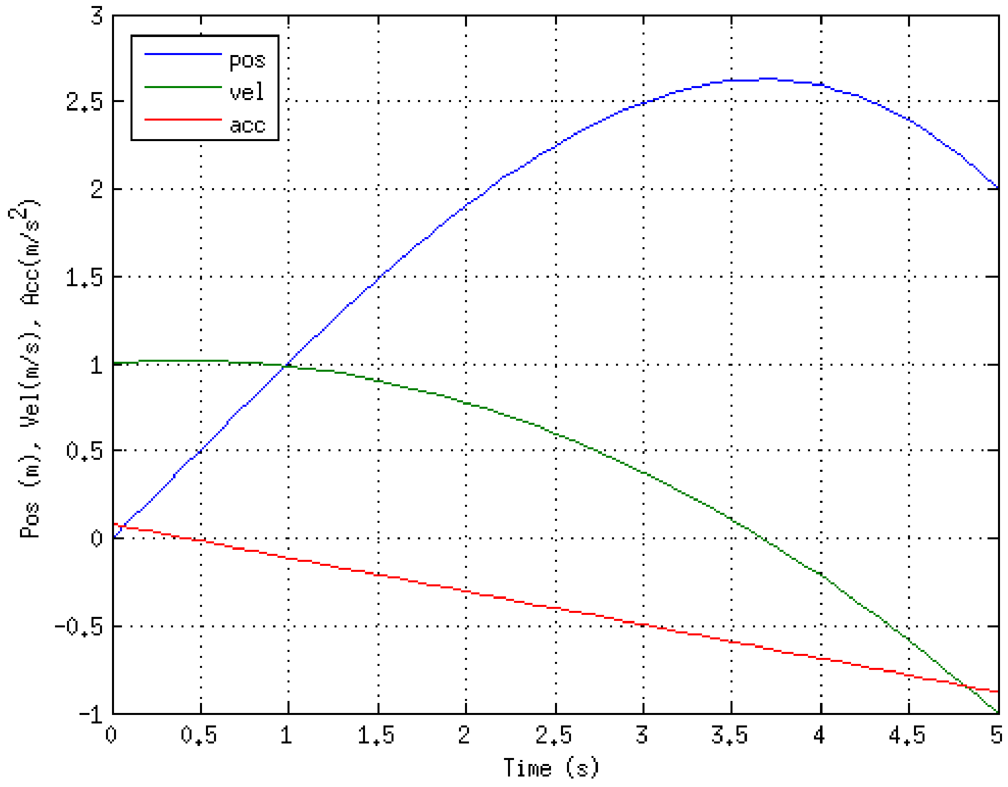


Figure 4.1: Example trajectory through start point $q(0) = 0$, $\dot{q}(0) = 1$ and end point $q(5) = 2$, $\dot{q}(5) = -1$.

which can be solved for the unknowns a_{0-3}.

For every time t the associated position, velocity and acceleration can be calculated using (4.1) and (4.3). An example is shown in Figure 4.1.

The main drawback of this method is that the generated trajectories are suboptimal, because the acceleration is a linear equation as opposed to phases with maximal acceleration or deceleration as in the current implementation of the trajectory planner.

This method generates a trajectory for one DOF. It is therefore executed once for both the x- and y-DOF, after which they are combined into a single 2D trajectory for the Turtle. Combining both trajectories is done using the low-level controller, this controller expects a separate x- and y-acceleration and follows this signal.

4.3.2 Trajectory feasibility check

Using this polynomial method it is possible to generate a trajectory between any two motion states, not regarding the duration of the trajectory. The physical constraints of the trajectory are not guaranteed in any way. For example, in the case a very small duration is specified a solution will be found, but the maximal accelerations and velocities will exceed the physical constraints of the Turtles. Checking for the physical constraints on the separated trajectories for one degree of freedom is not enough. To guarantee the feasibility, of the complete 2D trajectory, a combined check has to be performed.

Physical constraint checking First the 1D trajectories are checked separately, because if the constraints will not hold in the separated trajectories they will not hold in the 2D trajectory either. The 1D velocity is given by

$$\dot{q}(t)_{1D} = a_1 + 2a_2 t + 3a_3 t^2, \tag{4.9}$$

with a_1, a_2 and a_3 the polynomial factors and where a_1 represents the initial velocity at $t = 0$. In the case $a_3 = 0$ there will be a constant acceleration or deceleration depending on the sign of a_2. In the case $a_3 \neq 0$ there is a extrema at

$$t_{extrema} = \frac{-a_2}{3 \cdot a_3}, \tag{4.10}$$

which can be positioned outside the interval $[t_0,\, t_f]$, therefore a check if $t_{extrema}$ is within this interval has to be performed. The maximal 1D velocity

$$\dot{q}(t)_{1D\,MAX} = \begin{cases} \dot{q}(t_0)_{1D} & a_3 = 0 \; and \; sign(a_2) = -1 \\ \dot{q}(t_f)_{1D} & a_3 = 0 \; and \; sign(a_2) = 1 \\ \dot{q}(t_{extrema})_{1D} & a_3 \neq 0 \; and \; if \; t_0 < t_{extrema} < t_f \\ \max\left(\left|\dot{q}(t_0)_{1D}\right|, \left|\dot{q}(t_f)_{1D}\right|\right) & a_3 \neq 0 \; and \; (if \; t_{extrema} < t_0 \; or \; t_{extrema} > t_f)\,, \end{cases} \tag{4.11}$$

can then be computed. The maximal 1D acceleration is found by taking the acceleration of a cubic polynomial

$$\ddot{q}(t)_{1D} = 2a_2 + 6a_3 \cdot t, \tag{4.12}$$

with a_2 and a_3 the polynomial factors and t the time, and solving this equation for $t = t_0$ and $t = t_f$. The maximal acceleration or deceleration is always found at these points because the acceleration is a linear equation. The largest acceleration or deceleration

$$\ddot{q}(t)_{1D\,MAX} = \max\left(\left|\ddot{q}(t_0)_{1D}\right|, \left|\ddot{q}(t_f)_{1D}\right|\right), \tag{4.13}$$

is then found by taking the maximal value of the absolute values at $t = t_0$ and $t = t_f$.

Both $\dot{q}(t)_{1D\,MAX}$ and $\ddot{q}(t)_{1D\,MAX}$ have to be smaller or equal to the maximal velocity and acceleration of the Turtle respectively. Even if the separate 1D trajectories agree with the physical constraints of the Turtle, the combined 2D trajectory can still violate the constraints. An analytical solution for the combined trajectory is given by the differentiation of the 2D velocity

$$\dot{q}(t)_{2D} = \sqrt{\dot{q}(t)_x^2 + \dot{q}(t)_y^2}, \tag{4.14}$$

with $\dot{q}(t)_{2D}$ the combined velocity, and for the 2D acceleration

$$\ddot{q}(t)_{2D} = \sqrt{\ddot{q}(t)_x^2 + \ddot{q}(t)_y^2}, \tag{4.15}$$

with $\ddot{q}(t)_{2D}$ the combined acceleration, such that the peaks in the acceleration and velocity can be found. This solution is very involved and in order to finish the project in time, for the RoboCup World Championship 2013, a brute force algorithm has been used instead.

The 2D trajectory constraint checking algorithm, see Algorithm 4.1, evaluates (4.14) and (4.15) in a loop over the interval $[t_0,\, t_f]$ while checking them against the Turtle constraints. A one is returned if the constraints are met and a zero otherwise.

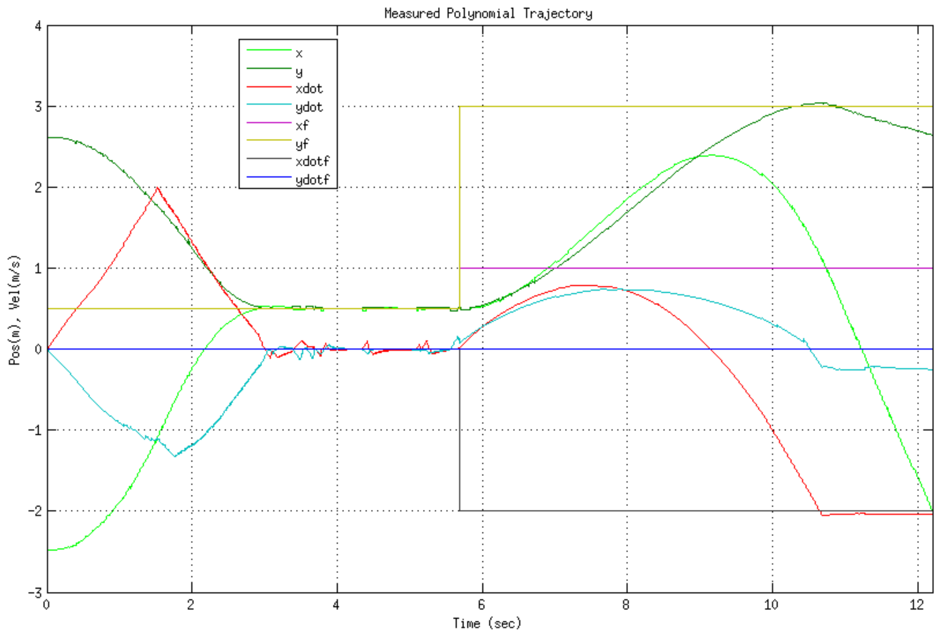


Figure 4.2: Up to $t = 5.9$ the robot has positioned itself on the location (0.0, 0.5) and is waiting with zero velocity. At $t = 5.9$ a movement is initiated, the Turtle must be at (1, 3) with velocity vector $(-2, 0)$ in five seconds. As can be seen this motion state is indeed reached, approximately at $t = 10.9$.

Algorithm 4.1 2D Trajectory Constraint Check

```
for($t =t_0$ until $t_f$ step timeSTEP)
    if($\dot{q}(t)_{2D} >$ max Turtle velocity)
        return 0
    if($\ddot{q}(t)_{2D} >$ max Turtle acceleration)
        return 0
next
return 1
```

4.3.3 Implementation problems

The polynomial trajectory generation has successfully been implemented in the Turtles, see Figure 4.2. However, tests with the push action have shown that the resulting behavior is instable due to noise in the tracking of the trajectory. The measured position and velocity of the Turtle are subject to noise as well as the control signal for, and the behavior of, the motors. These two factors are the cause of the Turtle not following the generated trajectory exactly. The effect of the Turtle being unable to follow the trajectory is larger at greater accelerations. During the push action there is only a small time window to react which causes the accelerations to be relatively large.

The trajectory is generated once at t_0 and then followed until the desired final motion state is reached at t_f, if the noise or inability to follow the generated trajectory causes the Turtle to deviate from this trajectory by a certain amount, a response is required:

1. Recalculate the polynomial from that moment.

2. Stop following the trajectory, abort the action that required the movement.

Recalculating the polynomial, from the moment in time at which the deviation was getting too large, is preferred. During soccer, a best effort try is usually better than no action at all. However, recalculating is only useful if the initial polynomial does not contain accelerations and/or velocities equal to the physical limits. If it did, a recalculation will always result in an infeasible polynomial.

Due to these problems and the limited time available to implement the push another method is used, see the following section.

4.4 Velocity limiting

The method that has been implemented during the RoboCup world championship 2013 is presented here. This method uses the original trajectory planner that was already available. The distance between the Turtle and the POI $dist_{tp}$ is divided by the time the ball needs to arrive at the POI t_{bp}. The desired velocity of the Turtle

$$v_{desired} = \frac{dist_{tp}}{t_{bp}}, \tag{4.16}$$

will be set. Therefore, the Turtle will arrive at the POI at the same time as the ball. This calculation is performed every iteration, and therefore a feedback loop is created that will result in a best effort movement to be at the POI at the correct time. The original trajectory planner will normally decelerate before it arrives at the POI because it does not incorporate a non-zero final velocity. To prevent this behavior, the target of the Turtle is projected at a certain distance behind the POI from the point of view of the Turtle.

This method does not take the initial velocity, and the acceleration and velocity limits into account. The current controller always accelerates using the physical acceleration limit of the Turtles and therefore tries to assume $v_{desired}$ as quickly as possible. If the set $v_{desired}$ is not reached in the next sample it will be automatically corrected through the feedback behavior. A check on the feasibility of the velocity and acceleration has not been implemented because it is assumed that the push action will only be called when the POI is reachable. Not checking for the velocity and acceleration constraints will result in a best-effort push action. This is beneficial in the case of incorrect ball measurements, which occur due to the noisy measurements by the vision system.

Chapter 5

Experiments

An experiment has been conducted to test the accuracy of the push action during an active corner, see Section 5.1. A comparison with the current ball handling has not been made due to the limited time available and has to be performed during future work.

5.1 Push accuracy

The accuracy of the Turtle, during an active corner, is the precision at which it is able to push the ball towards a certain target at the goal line. The offset from the target to the point where the ball crosses the goal line is used as the measure of accuracy.

Experiment setup To test the accuracy of the push action during an active corner, a setup with two Turtles has been used at two situations, see Figures 5.1 and 5.2, one shooting Turtle and one pushing Turtle. The shooting Turtle will shoot from the corner region, and the pushing Turtle will pre-orientate itself at a fixed position of the field. The shooter will then shoot to a point between the pusher and the target, which is located at the center of the goal line. The point that is shot at, is fixed at a shooting distance of $D_s = 1,2\,m$ on the line from the pushing Turtle towards the target. The offset from the target is measured by placing a measuring tape along the goal line and determining, by eye, what the deviation from the target is.

The shooting Turtle is inaccurate when shooting the ball, which is something that has to be improved during the coming year. Therefore, it will sometimes be harder or impossible to correctly intercept the ball. The push action is designed to be robust towards these errors but limits have to be defined. If the point at which the shooting Turtle actually shoots is too far off its intended target D_s, the result will be discarded. The result will be discarded if the ball crosses the line, from the pushing Turtle to the target, at a distance larger than $0.5\,m$ from its intended target distance D_s.

Results The results of the experiment can be found in Figures 5.3 and 5.4. The average absolute offset at the first situation is found to be $19\,cm$; at the second situation the average absolute offset is $10\,cm$. The combined average absolute offset is $14\,cm$.

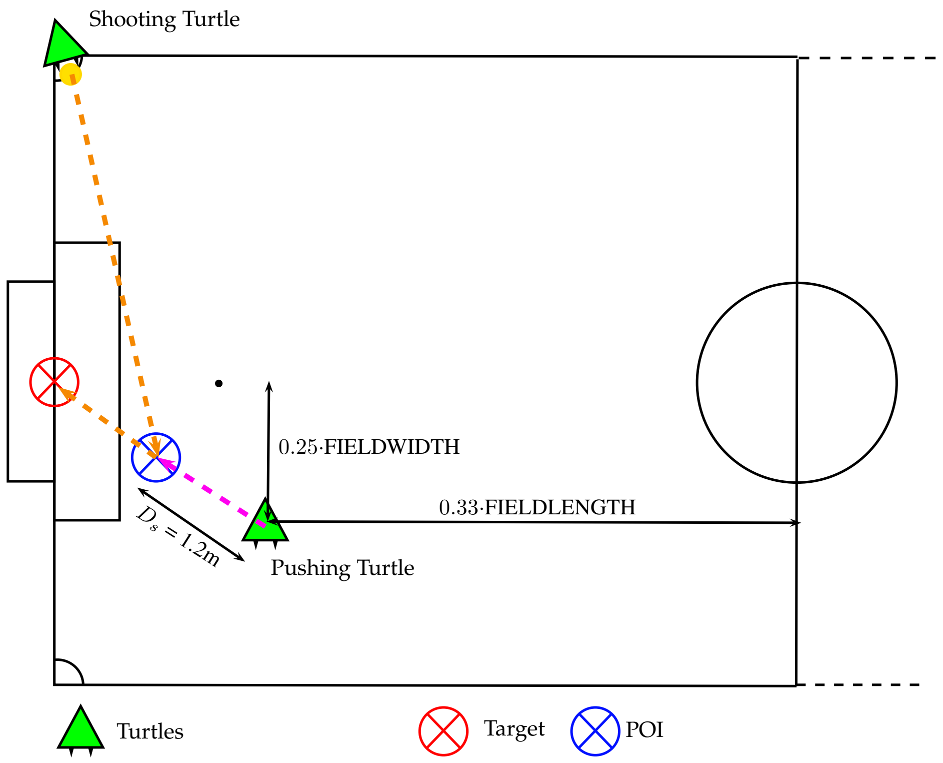


Figure 5.1: Setup of the push accuracy experiment, situation 1 (not to scale).

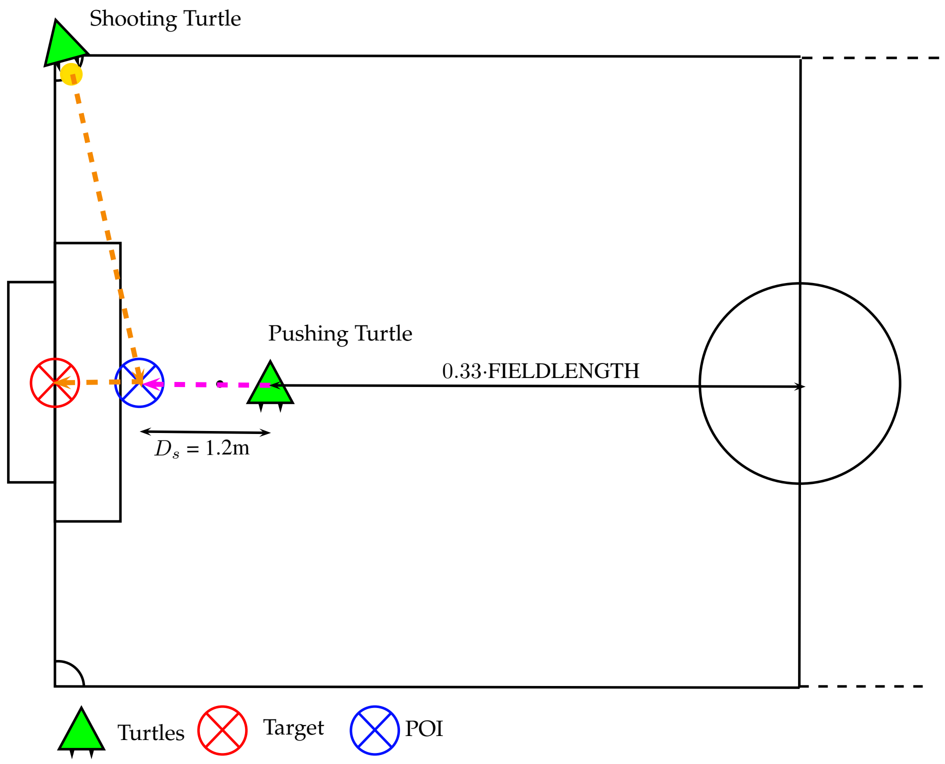


Figure 5.2: Setup of the push accuracy experiment, situation 2 (not to scale).

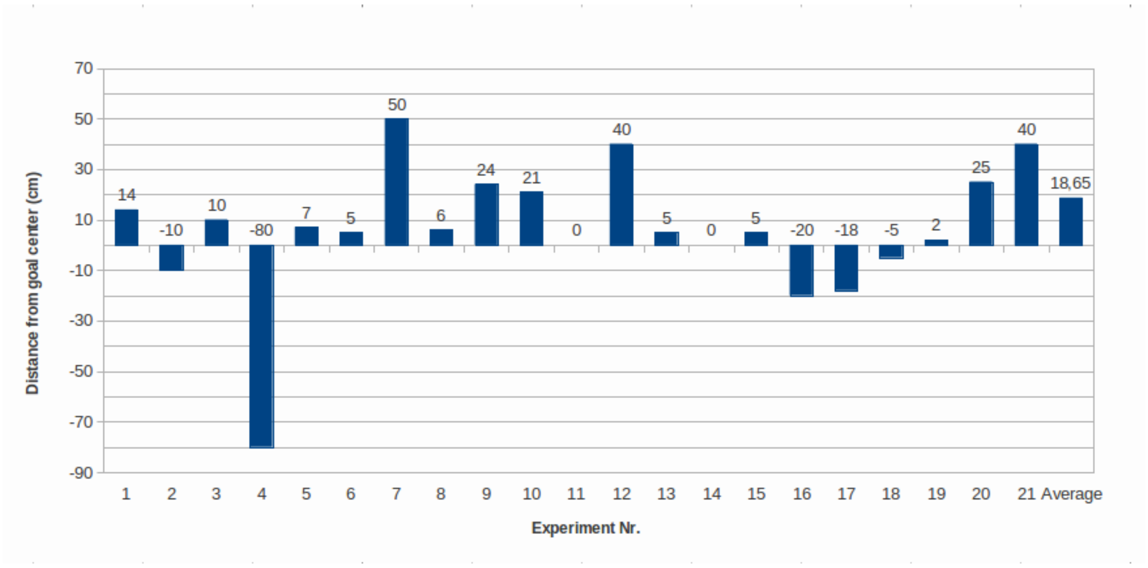


Figure 5.3: Result of push accuracy experiment, situation 1.

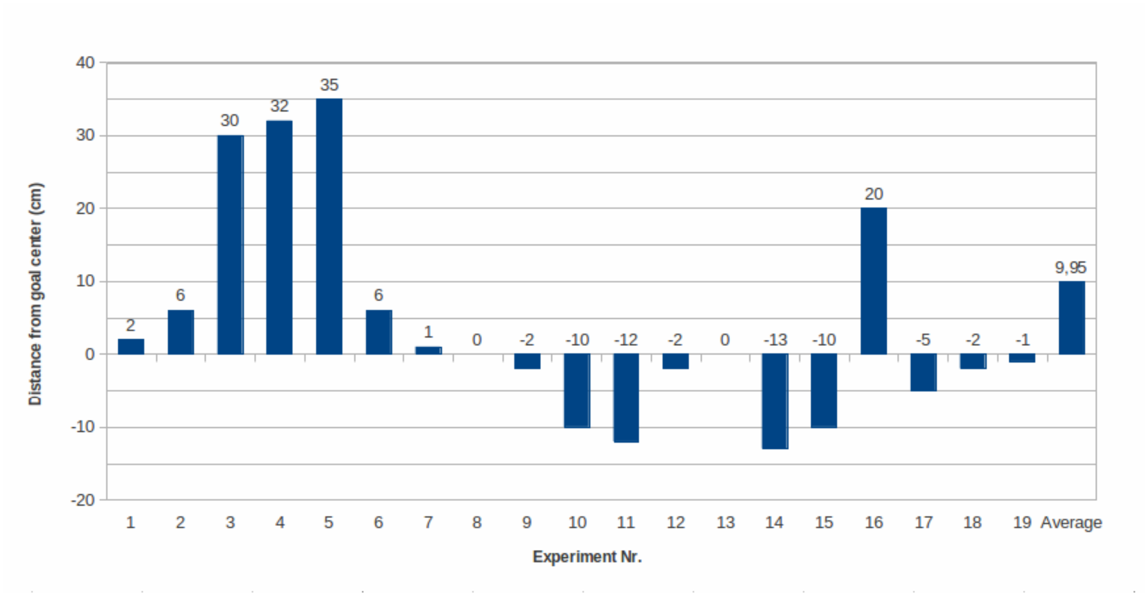


Figure 5.4: Result of push accuracy experiment, situation 2.

Chapter 6

Conclusion

This chapter summarizes the main results of this master thesis. A reflection on the the requirements, which have been stated in Section 1.3, is made in Section 6.1. In Section 6.2 the future work is presented.

6.1 Reflection on requirements

The requirements and their accompanying challenges, see Section 1.3, are mostly fulfilled. Some requirements still need work to fine tune the implementations.

The first requirement, *'Control the direction and velocity of a pushed ball'* has been partly fulfilled. Using the current implementation it is possible to control the outgoing trajectory of the ball but it is not possible to also control the velocity. The accuracy has been tested during an experiment, see Chapter 5. At a corner situation, the push action is able to achieve an average absolute offset from the target of $14\,cm$. The velocity cannot be controlled due to problems with the trajectory planner. In order to be able to control the outgoing trajectory an impact model and a trajectory planner have been implemented. The trajectory planner, which has been implemented in the final version, is not the cubic-polynomial trajectory planner as intended at the beginning of the project. The cubic-polynomial planner has been replaced by a the relative simple 'velocity limiting' method due to problems at the following of the generated trajectories in combination with the deadline of the project. It is recommended that a trajectory planner that is able to realize non-zero final velocities is implemented in future work.

The second requirement *'Integrate the push action into the current strategy and increase the usability of the push action'* has been achieved. The push has been implemented at the active corner task. Functions have been written, in the form of hard conditions and partial ranking functions, which analyze the situation and predict the successfulness of a hypothetical push action. Based on this prediction a decision is taken whether or not to execute the push action.

The third requirement *'The developed software should be robust for sensor and control imperfections. Therefore, the push action should be tested thoroughly, both in simulations and on the Tech United soccer field.'* has also been fulfilled, except for an experiment comparing the current ball handling with the push action. The performance in terms of accuracy has been determined by performing an experiment and the COR impact model has been implemented in the Tech United simulator.

6.2 Future work

There are a number of items that can be recommended as future work.

- Trajectory planner
  During this project one of the main goals was to implement a cubic-polynomial trajectory planner. The following problems arose during the development of the trajectory planner.

  - Whenever noise causes the Turtle to deviate from the planned trajectory and the final motion state becomes unreachable it has to stop following the trajectory. This behavior is unwanted because in most situations a best effort movement is desirable.
  - The trajectories generated are sub-optimal because the acceleration is a linear function of time.
  - Two separate sub-planners are used in the current implementation, one for the position and one for the rotation. The control signals of these two sub-planners are superimposed. This yields sub-optimal or even impossible trajectories.

  It is recommended that work is continued on developing a trajectory planner. The following points have to be taken into account.

  - The programmer should be able to specify a maximal deviation at every intermediate motion state. This maximal deviation will indicate how closely a motion state has to be achieved.
  - A planner that can generate trajectories that have discontinuities in the acceleration profile is preferred. This way the trajectories will not be limited by the trajectory planner but by the maximal physical acceleration.
  - The developed planner should be a 6D state space and 3D control space trajectory planner with the state space
    $$x,\, y,\, \varphi,\, \dot{x},\, \dot{y},\, \dot{\varphi},$$
    and the control space
    $$\ddot{x},\, \ddot{y},\, \ddot{\varphi}.$$
    In [12] a method is presented that can generate these trajectories by a search based approach; using a cost function trajectories with different properties can be achieved.

- Push mu-field
  The mu-field on a line has been developed but is at the moment not yet used in the strategy. To further increase the usability of the push action, this should be implemented.

- More situations
  The push action is currently implemented in the active corner refbox task. To increase the usability of the push it has to be used in more situations. For example, during other active refbox tasks and normal play. During normal play the push can not only be used for scoring but also for passing, intercepting and to take the ball from an opponent.

- Accuracy experiments
  An experiment that compares the accuracy and speed of the normal ball handling versus the push action, in the corner situation, has to be performed. Experiments must also be performed whenever the push action is implemented in other situations. This can then be used to tune the partial ranking functions such that the best decision on whether to push or to use the normal ball handling can be made.

- Search instead of brute force algorithm
  The algorithm to find the orientation of the push surface, see Section 3.3, is currently implemented in a brute force manner. With additional effort this can be done using a search algorithm because the function relating incoming ball velocity, outgoing ball velocity, turtle velocity and the orientation seems to be continuous. This will reduce the computation time and enable a higher accuracy.

- Adding 'Flippers'
  As previously stated by J.W. Lamers in [8], the addition of some kind of actuators on the side of the Turtles enables a push action that can counteract any energy loss of, or add kinetic energy to, the ball using these actuators. The advantage is that this energy does not have to come from the motion state of the Turtle. This simplifies the Turtle control problem. Because the addition of actuators is a very large adaptation, it is better to first focus on fully implementing the push as it is at this moment.

Bibliography

[1] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer Berlin Heidelberg, 2008.

[2] R.M. Brach. *Mechanical Impact Dynamics: Rigid Body Collisions*. Wiley-Interscience, 2007.

[3] R. Cross. Measurements of the horizontal coefficient of restitution for a superball and a tennis ball. *American Journal of Physics*, 70(1):482, 2002.

[4] T. A. de Araujo Costa and M. Suell Dutra. Parametric trajectory generation for mobile robots. In *ABCM Symposium Series in Mechatronics*, volume 3, pages 300–307, 2008.

[5] S. Faik and H. Witteman. Modeling of impact dynamics: A literature survey. Proceedings International ADAMS User Conference, 2000.

[6] Y. Guan, K. Yokoi, O. Stasse, and A. Kheddar. On robotic trajectory planning using polynomial interpolations. In *Robotics and Biomimetics (ROBIO). 2005 IEEE International Conference*, pages 111–116, 2005.

[7] J. Hierrezuelo and C. Carnero. Sliding and rolling: the physics of a rolling ball. *Physics Education*, 30(3):177, 1995.

[8] J.W. Lamers. Push: Alternative ball handling for the turtle soccer robots. Master's thesis, Eindhoven University of Technology, August 2012.

[9] Claudio Melchiorri. Trajectory planning for robot manipulators. Presentation PDF, www-lar.deis.unibo.it/people/cmelchiorri/Files_Robotica/FIR_07_Traj_1.pdf.

[10] O. Purwin and R. D'Andrea. Trajectory generation for four wheeled omnidirectional vehicles. In *Proceedings of the 2005 American Control Conference*, volume 7, pages 4979–4984, 2005.

[11] RoboCup Federation Website. www.robocup.org.

[12] C. Sprunk, B. Lau, P. Pfaffz, and W. Burgard. Online generation of kinodynamic trajectories for non-circular omnidirectional robots. In *Robotics and Automation (ICRA), 2011 IEEE International Conference*, pages 72–77, 2011.

[13] Tech United Eindhoven Website. www.techunited.nl.

[14] Ubuntu Real-Time Kernel. www.kernel.org/pub/linux/kernel/projects/rt/3.2/, rt.wiki.kernel.org/index.php/config_preempt_rt_patch.

[15] A.W. van Zundert. Traction control for omni-directional robots. Master's thesis, Eindhoven University of Technology, Mechanical Engineering and Control Systems Technology, September 2011.

[16] R. E. Wallace and M. C. Schroeder. Analysis of billiard ball collisions in two dimensions. *Am. J. Phys*, 56(9):815–819, 1988.