DYNAMIC PUSH

Alternative kicking by intercepting and pushing the ball with an omni-directional soccer robot.

Eindhoven University of Technology / CST 2014.093
Melvin de Wildt, 21/10/2014

BACHELOR REPORT
Eindhoven University of Technology
Department of Mechanical Engineering
Control Systems Technology

AUTHOR
M.S. de Wildt
0776941
m.s.d.wildt@student.tue.nl

TUTOR
C. Lopez
c.lopez@tue.nl

SUPERVISOR
M.J.G. van de Molengraft
m.j.g.v.d.molengraft@tue.nl

ABSTRACT

A Push action is a move where a robot bounces a ball with his side instead of controlling it with his ball-handling mechanism. This project is about developing and implementing an algorithm to integrate the Push action into a dynamic situation. Experiments on the field show that a Push success rate of 90% is achieved in a dynamic situation with obstacles.

CONTENTS

1 Introduction 6
1.1 RoboCup 6
1.2 Motivation 6
1.3 Goals 7
1.3.1 Main goal 7
1.3.2 Context 7
2 Soccer robot general 8
2.1 Hardware 8
2.1.1 Housing 8
2.1.2 Software processing 8
2.1.3 Omni-wheels 8
2.1.4 Ball Handling 9
2.1.5 Sensors 9
2.1.6 Actuators 9
2.1.7 Communications 9
2.2 Software 10
2.2.1 Software architecture 10
2.2.2 Motion Software 11
2.2.3 Trajectory Planner 13
2.2.4 Mu-Fields 13
3 Dynamic Push 15
3.1 State machine 15
Go to a start position 15
Communicate 15
Drive to a position 16
Wait until a pass can be given 16
Push pre orient 16
Pass 16
Push the ball to the desired target 16
3.2 Push physics 17

3.2.1 Static Push 17
3.2.2 Dynamic 18
3.3 Velocity of the robot 21
3.4 Communication 23
3.5 Dynamic Mu field 24
3.5.1 Check after Mu field 25
3.6 Passing velocity 27
3.7 Implementation of Turtle target 29
4 Results 30
4.1 Simulation 30
4.2 Experiments 33
5 Conclusion 34
5.1 Future work 34
5.2 Recommendations 34
6 Appendix 35
7 Bibliography 36

1 INTRODUCTION

1.1 RoboCup

RoboCup was founded in 1997 with the main goal of “developing by 2050 a Robot Soccer team capable of winning against the human team champion of the FIFA World Cup”. In the past years, RoboCup proposed several soccer platforms that have been established as standard platforms for robotics research. This domain demonstrated the capability of capturing key aspects of complex real world problems and stimulating the development of a wide range of technologies. This includes the design of integrated techniques for autonomous robots. After more than 15 years of RoboCup, nowadays robot soccer represents only a part of the available platforms. RoboCup encompasses other leagues that, in addition to Soccer, cover Rescue (Robots and Simulation), @Home (assistive robots in home environments), Sponsored and @Work (Industrial environments), as well as RoboCup Junior leagues for young students. These domains offer a wide range of platforms for researchers with the potential to speed up the developments in the robotics field.

RoboCup has already grown into a project that gets worldwide attention. Every year, multiple tournaments are organized in different countries, where teams from all over the world participate in various disciplines. [1]

1.2 Motivation

The soccer robots of the Eindhoven University of Technology have a ball handling mechanism, consisting of two small wheels attached to levers to catch and control the ball. After a ball is grabbed by this ball handling mechanism the robot is able to dribble across the field. If a robot has the ball controlled in its ball handling mechanism it can shoot the ball with its kicking mechanism. In some cases it might not be desirable to grab the ball before shooting it in another direction, mainly because the next two reasons:

1. Intercepting the ball sometimes fails, since the ball handling system is quite small, the robot has to position itself accurately.
2. Controlling the ball, turning and shooting is time consuming. An opponent gets the chance to anticipate on the robot with the ball.

A way of avoiding the use of the ball handling mechanism is to push the ball with the housing of the robot. This is referred to as a Push. It is faster to push the ball than to control it with the ball handling mechanism and then shoot.

1.3 Goals

1.3.1 Main goal

Develop and implement an algorithm that allows to Push the ball to a certain target in a dynamic situation. This is done with an omni-directional soccer robot. Pushing means bouncing the ball with the side of a moving robot, see Figure 1.1. The target could be anywhere, but in this case the focus lies on scoring, meaning that the target is the goal of the opponent. A dynamic situation means that the two robots, in this case the attacker main and the attacker assist (which are the shooter and the pusher respectively) can stand anywhere on the pitch and are still able to execute a Push. Obstacles are taken into account in the algorithm.

1.3.2 Context

At the start of the project there was a dynamic Push function created by Okke Hendriks [2]. That function was based on the static Push function created by J.W. Lamers [3]. This dynamic function needed improvement since it wasn't accurate enough. It worked correctly for a relatively small range of ball velocities. When the ball was moving too slow it touched the wrong surface of the Turtle, when the ball was moving too fast the Turtle missed it completely. The Push skill existed of one action, that is one robot pre orienting and starting the Push action when it saw a moving ball.

The Push skill will be expanded so it becomes an integrated routine that can be used in play. The attacker main will give a depth pass to the attacker assist so it can Push the ball towards the goal. The accuracy and the success rate of the Push should increase. The robot giving the depth pass has to determine when, where and how hard to give the depth pass.

Figure 1.1 / Suitable sides for bouncing the ball marked in red.

2 SOCCER ROBOT GENERAL

In this Chapter the current state of the Turtles is described [2]. The hardware of the Turtles that is relevant for a Push is discussed in Section 2.1. The software modules that are relevant for a Push are discussed in Section 2.2. The mu-fields, a method to make fuzzy decisions, are discussed in more detail in Section 2.2.4.

2.1 Hardware

The Turtles are equipped with sensing, actuating, processing and communication hardware on board. Each Turtle, except the keeper, has exactly the same hardware and software.

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 [3] the foam absorbs a lot of the energy of a ball hitting the housing in the form of elastic deformation. The energy dissipation lies between 25 and 35 percent. Besides the velocity change the impact of the ball may also cause the push surface to temporarily deform. Due to the impact the Push surface can differ slightly.

By using a dynamic push some kinetic energy is added to the ball because the Turtle is moving when it hits the ball.

2.1.2 Software processing

The software processing at each of the Turtles is performed by a Beckhoff industrial computer, the C9900-C543 with an Intel® Core i7 2710QE, 2.1 GHz processor.

2.1.3 Omni-wheels

Robots have three omni-wheels. An omni-wheel is lined with smaller wheels such that it can move freely in direction perpendicular to the conventional rotation direction. The platform becomes fully holonomic, meaning that it can immediately accelerate in any direction.

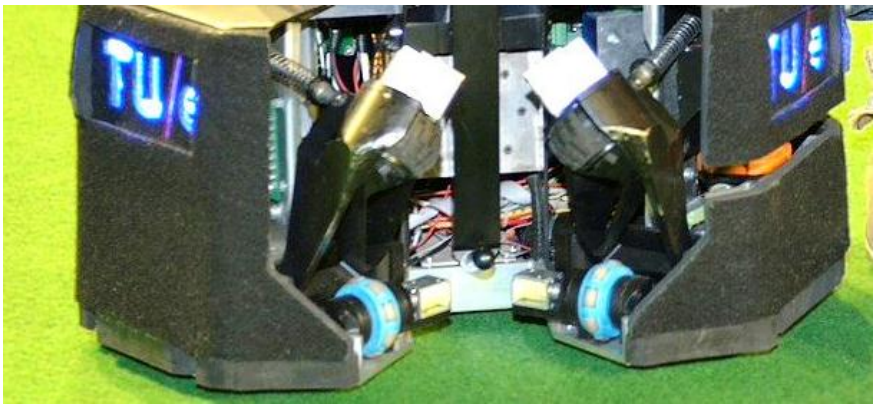

Figure 2.1 / Turtle ball handling mechanism

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.

2.1.5 Sensors

The Turtles have sensors to detect their position in the field, the ball position, obstacles and to control the ball. 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 driving motors which control the velocity of the motors.

2.1.6 Actuators

The actuators are used by the Turtles to move around and to control and shoot the ball.

Main driving motors

Each Turtle has three 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.

Shooting lever

The shooting lever consist out of an piston inside an electromagnet. This electromagnet is connected to a high voltage capacitor by an electronic switch.

2.1.7 Communications

Each Turtle is connected using a Wi-Fi network, the robots communicate with the referees computer and with their peer players. The inter-Turtle communication, is used to share information about the environment and agree on which actions to perform. The UDP protocol is used with a custom software interface which tries to keep the overhead as low as possible in order to minimize packet loss.

2.2 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, which consists of c-code together with an interface such that the s-function can be viewed and connected together inside Simulink. The usage of these blocks gives a clear overview of the code structure.

2.2.1 Software architecture

Globally the software is divided into three main parts: Motion, Vision and the Worldmodel. An example of a top level scheme can be found in Figure 2.2. In all schemes there are a number of blocks connected by lines. The blocks can either be a 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.

Vision and the worldmodel are briefly introduced. The Push is implemented in the Strategy and Control sub-scheme of Motion and is therefore most relevant in this project.

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 World-model. The Worldmodel receives communicated information from all the 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.

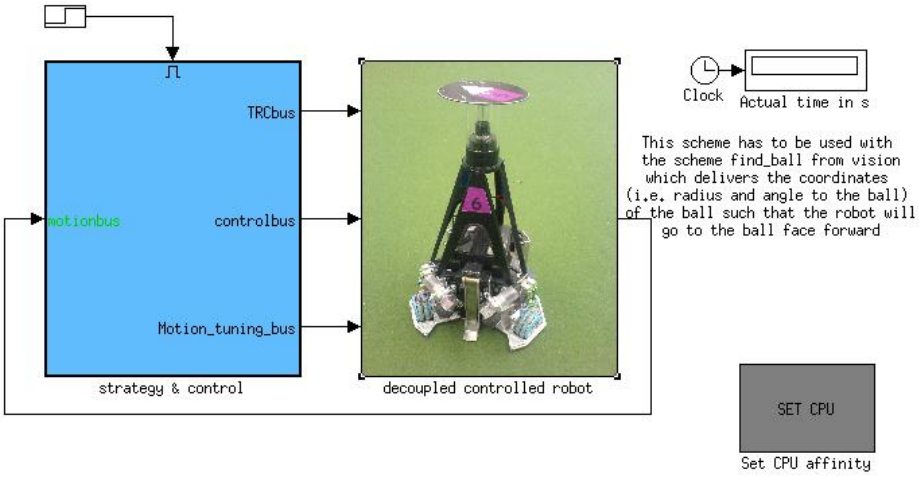

Figure 2.2 / Motion top leven Simulink scheme

2.2.2 Motion Software

All code needed to execute the push action is located in the Strategy and Control sub-scheme 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 2.3, is to analyze the game situation and ensure that the Turtle acts accordingly. During play a Turtle determines which actions to take.

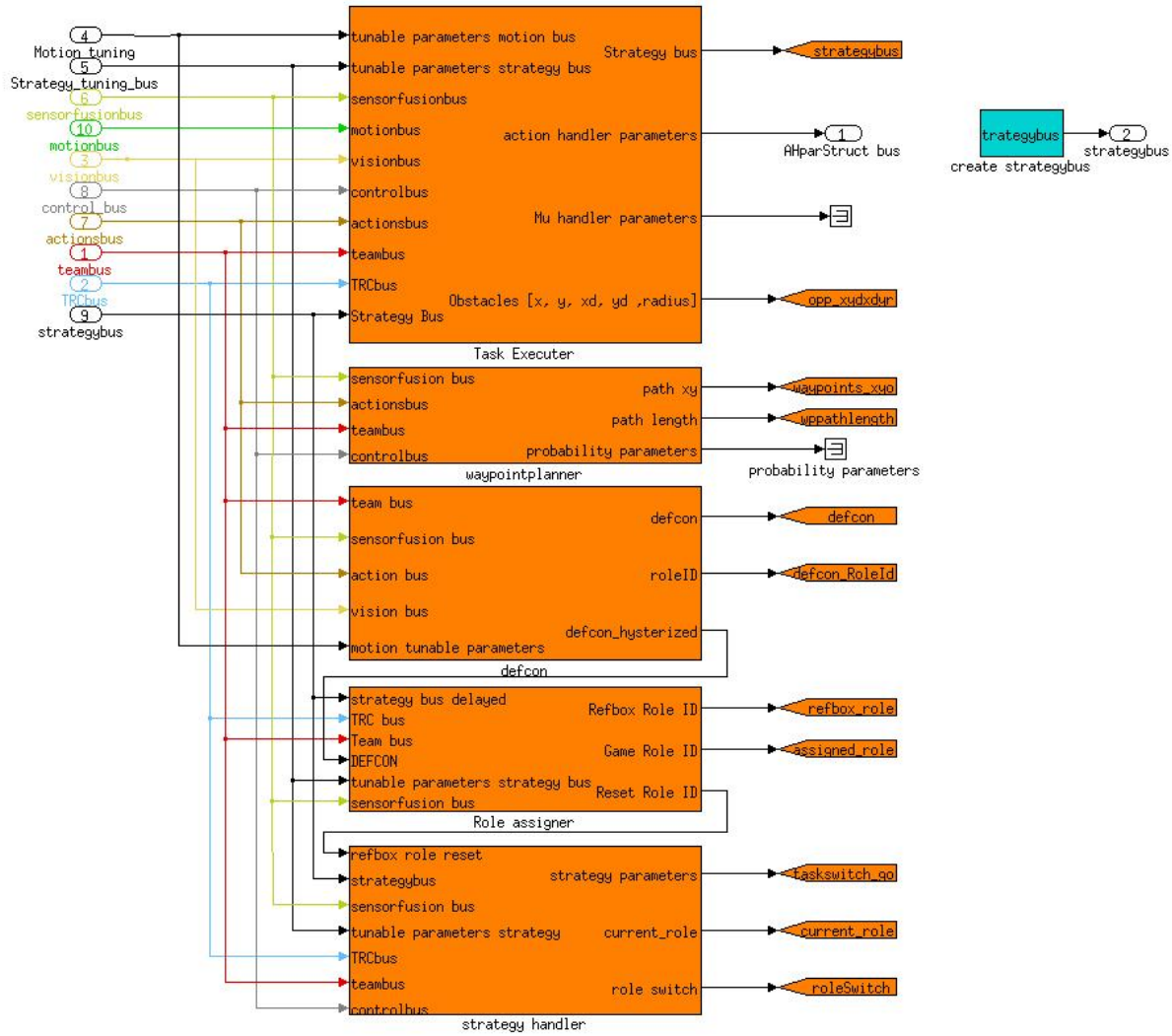

Figure 2.3 / Strategy Scheme

Actions are implemented in functions that reside in the action handler. These functions configure all the different actions that can be performed, e.g. `go to a specific position on the field' or `shoot at the goal'. The Push is implemented as one of these actions. A flow chart of how the actions are executed is given in Figure 2.4.

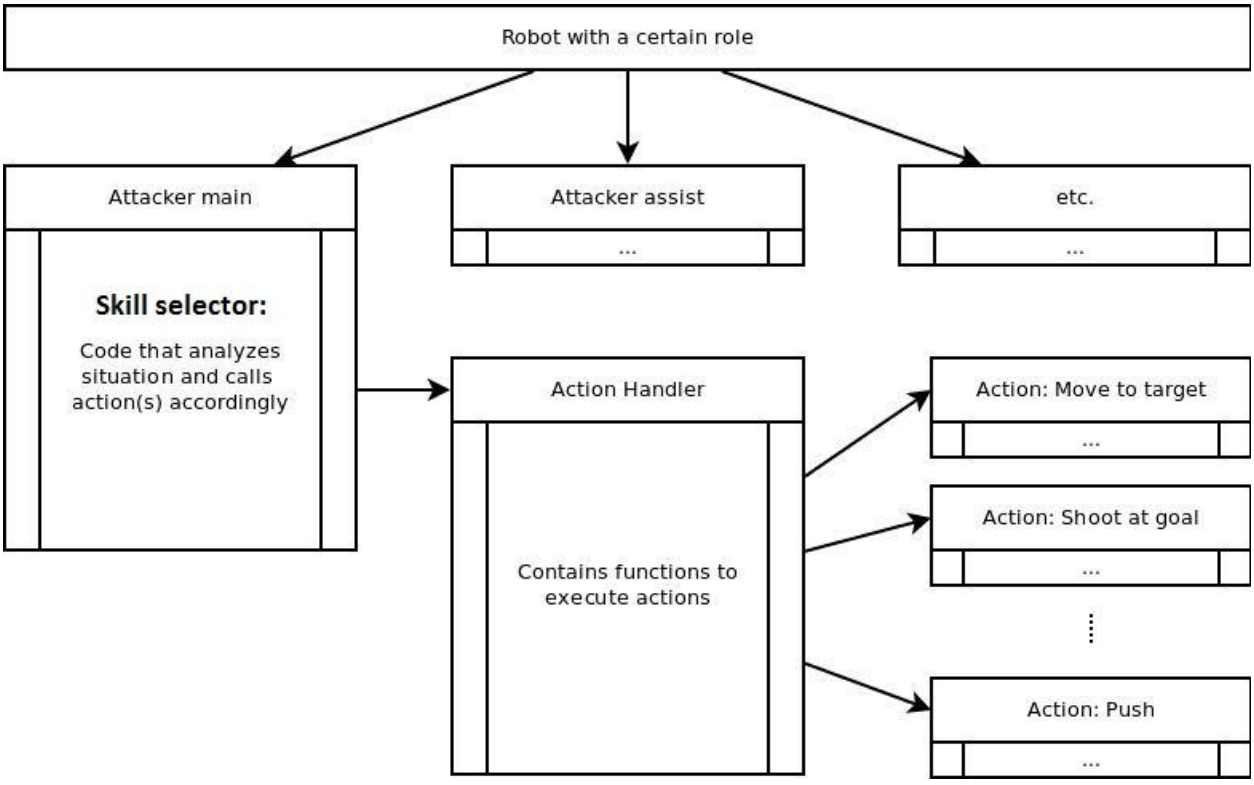

Figure 2.4 / Control flow of the Turtles

At the attacker main the behavioral algorithms are implemented in the form of the so-called skill selector. A skill is a group of actions. The skill selector takes all possible skills and calculates the success chance of each skill. This function returns a success chance from 0 to 1. The skills are manually ranked by giving each skill a tunable weight factor. This outcome of the success chance is multiplied with the tunable weight factor and compared by the skill selector. The skill selector chooses the skill that has the highest weighted success chance and executes the corresponding set of actions. The skill selector uses hysteresis to prevent constant switching between two skills.

2.2.2.2 Actions

The Action scheme contains the action handler that ensures that the functions of the actions that belong to a certain skill are called. Every action function 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 out of a number of waypoints. The sub-target planner ensures that the Turtles take the fastest path while avoiding obstacles between its current location and a target location.

2.2.3 Trajectory Planner

The Turtles have three DOF, the local x-axis, the local y-axis and the rotation. The trajectory planner uses the method described in Trajectory generation for four wheeled omnidirectional vehicles [4]. 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. Therefore the acceleration and deceleration will always be a_{max} or minus a_{max} respectively. The maximal velocity is always set to v_{max}.

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, such that a smooth transition from 1 to 0 is achieved.

For the position of an opponent on the field, a Gaussian bell function, see Figure 2.5, is 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.

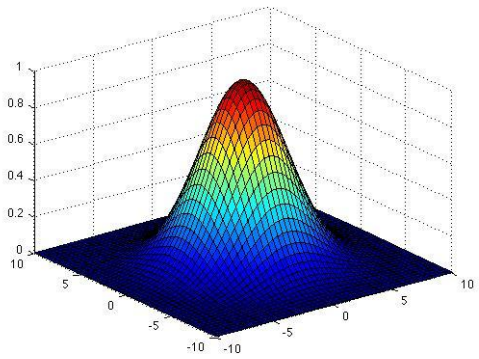

Figure 2.5 / Gaussian Bell function

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 2.6. Different actions require different mu-fields.

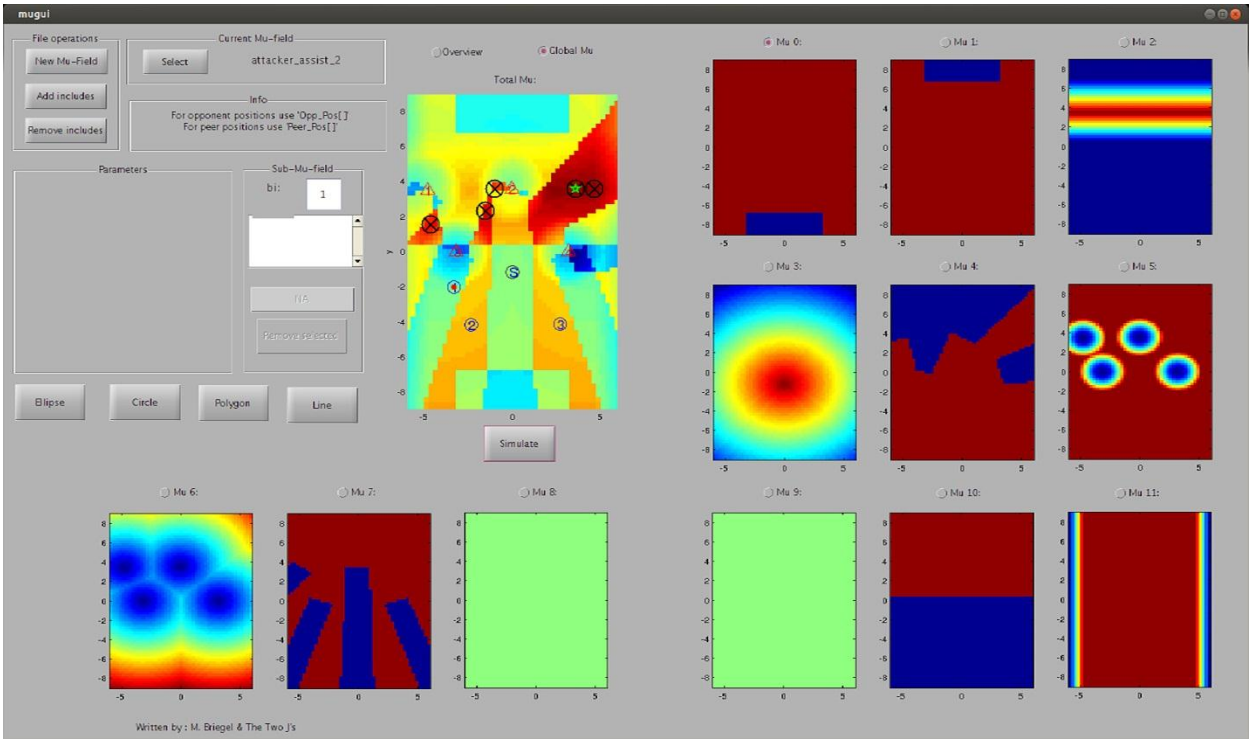

Figure 2.6 / GUI of the Mu field of the attacker assist role

The above figure shows the attacker assist mu-field in the Tech United mu-field GUI. 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 by a black circle with a cross in it, the one that has the highest value is indicated using a green star. This position of this highest value is returned by the mu-field function. The location of the top 5 best positions are saved, however the height of these top 5 points is not saved to increase computation speed and because they are scarcely used. The best position in the above image 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.

Furthermore, each sub mu field has its own weight factor. A sub mu field with a higher weight factor has more effect in the calculation of the best position.

3 DYNAMIC PUSH

Now that the important parts of the soccer robots are introduced, the Dynamic Push will be described in this chapter. The state machine to test the software, the theory and implementation of the Dynamic Push and the improvements made on the Push will be discussed.

3.1 State machine

For the testing environment of the Push, a state machine is introduced. A specific part of the state machine would actually be used in a match situation. The match part starts at the green block containing 'Match position'. Grey blocks indicate a state of a robot, see Figure 3.1.

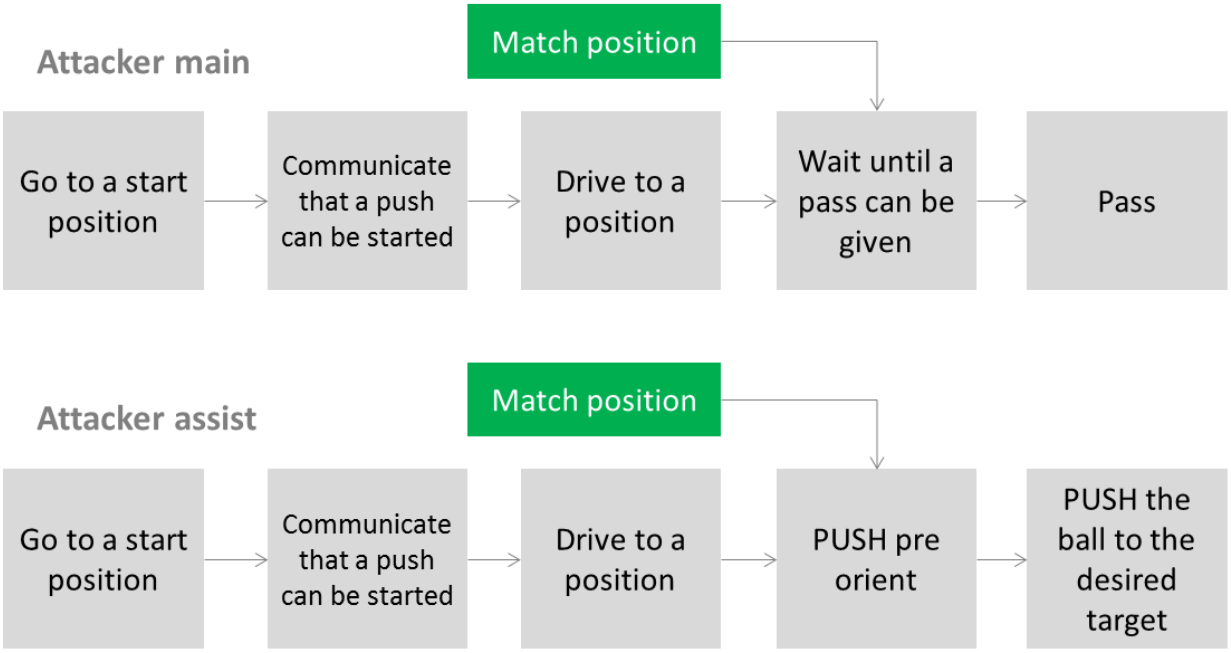

Figure 3.1 / State machine of the Push

All actions in the states will be briefly discussed in the following sections. They will be further discussed later on.

Go to a start position

In this block the Turtles simply move to a start position. This position relative to the size of the field.

Communicate

When the robots are at within a certain margin of the desired position they start sending messages to the team bus. This means they are ready to start a Push. This will be further discussed in Section 3.4.

Drive to a position

When the robots are both ready, they start the Push action. They both drive to a position on the field from where they are going to execute the actual Push.

Wait until a pass can be given

The attacker main calculates a mu field to give a depth pass on the Pushing robot. This will be discussed in Section 3.5.

Push pre orient

The robot that is going to Push waits for the ball to move. While waiting it is already rotated. This angle is calculated using the static Push angle as will be discussed in Section 3.2.1.

Pass

When the attacker main determines an opportunity to pass the ball, it will give the depth pass. This is further discussed in Section 3.6.

Push the ball to the desired target

The robot starts driving to the point of interception with an angle and a velocity. Eventually it will Push the ball and then the ball will bounce to the desired target. This will be discussed in Section 3.2.2.

3.2 Push physics

3.2.1 Static Push

A static Push is when the ball bounces on a surface which has velocity zero. The following assumptions are made.

- The ball is not rolling or spinning. This simplifies the equations.
- The surface at which the ball is bouncing has infinite stiffness and is completely static
- The ball is perfectly round and had an equal mass distribution
- The surface is perfectly flat

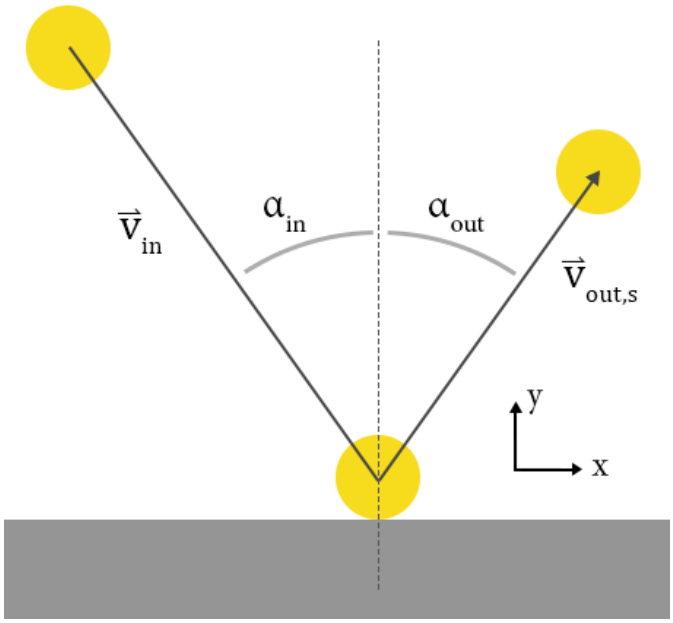

Figure 3.2 / Static Push, grey represents the housing of a Turtle

The $\vec{v}_{in}$ is the ingoing ball vector, $\vec{v}_{out,s}$ is the outgoing static ball vector, α_{in} is the angle of the incoming ball with respect to the y-axis, or the normal of the surface, α_{out} is the angle of the outgoing ball.

At the point of impact the ball dissipates some energy. Experiments have been performed to determine the 'Coefficient of Restitution'. According to O. Hendriks [2] this COR has a value of 0.71. Together with the figure above these simple equations follow.

$$\alpha_{in} = \alpha_{out} = \alpha \tag{3.1}$$

$$\vec{v}_{out,s} = COR * \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} * \vec{v}_{in} = COR * \begin{bmatrix} |v_{in}| * \sin(\alpha) \\ |v_{in}| * \cos(\alpha) \end{bmatrix} \tag{3.2}$$

The static Push calculation is used in the pre-orientation state of the dynamic Push. When the position of the ball, the point of interception (or an estimation of this) and the target are known, the static Push angle can be calculated. The Turtle has to position itself in such a way that the ball will bounce as shown in Figure 3.2.

The calculated angle and pre-orientation position are an input argument of function 'Goto_target_AH', the Turtle will drive to the position with the calculated angle. The calculation of the desired angle is done by the function 'getStaticPushAngle'. The point of interception, ball position and the target are input arguments of this function.

3.2.2 Dynamic

A dynamic Push is a static push plus an extra velocity vector. In the following figure the velocity of the ball is zero. The Turtle bounces the ball with an angle ϑ. This extra velocity vector can be calculated using Equation (3.3). As can be seen in Figure 3.3, ϑ is the angle between the normal of the pushing surface and the Turtle's velocity. The ball will bounce perpendicular to the Push surface.

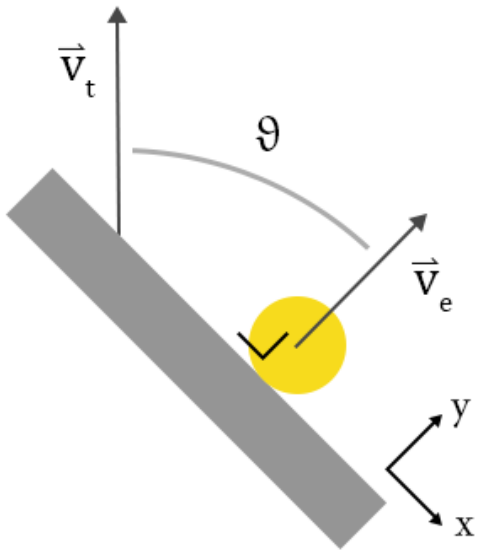

Figure 3.3 / Calculation of extra velocity vector

The Turtle angle is ϑ, the Turtle velocity is $\vec{v}_t$ and the extra velocity added to the ball is $\vec{v}_e$.

$$\vec{v}_e = \mathrm{COR} * \begin{bmatrix} 0 \\ \cos(\vartheta) * |\vec{v}_t| \end{bmatrix} \tag{3.3}$$

Now the static Push can be combined with the extra velocity vector from (3.3). This results in two vectors which can be superposed. This is shown in Figure 3.4 on the next page.

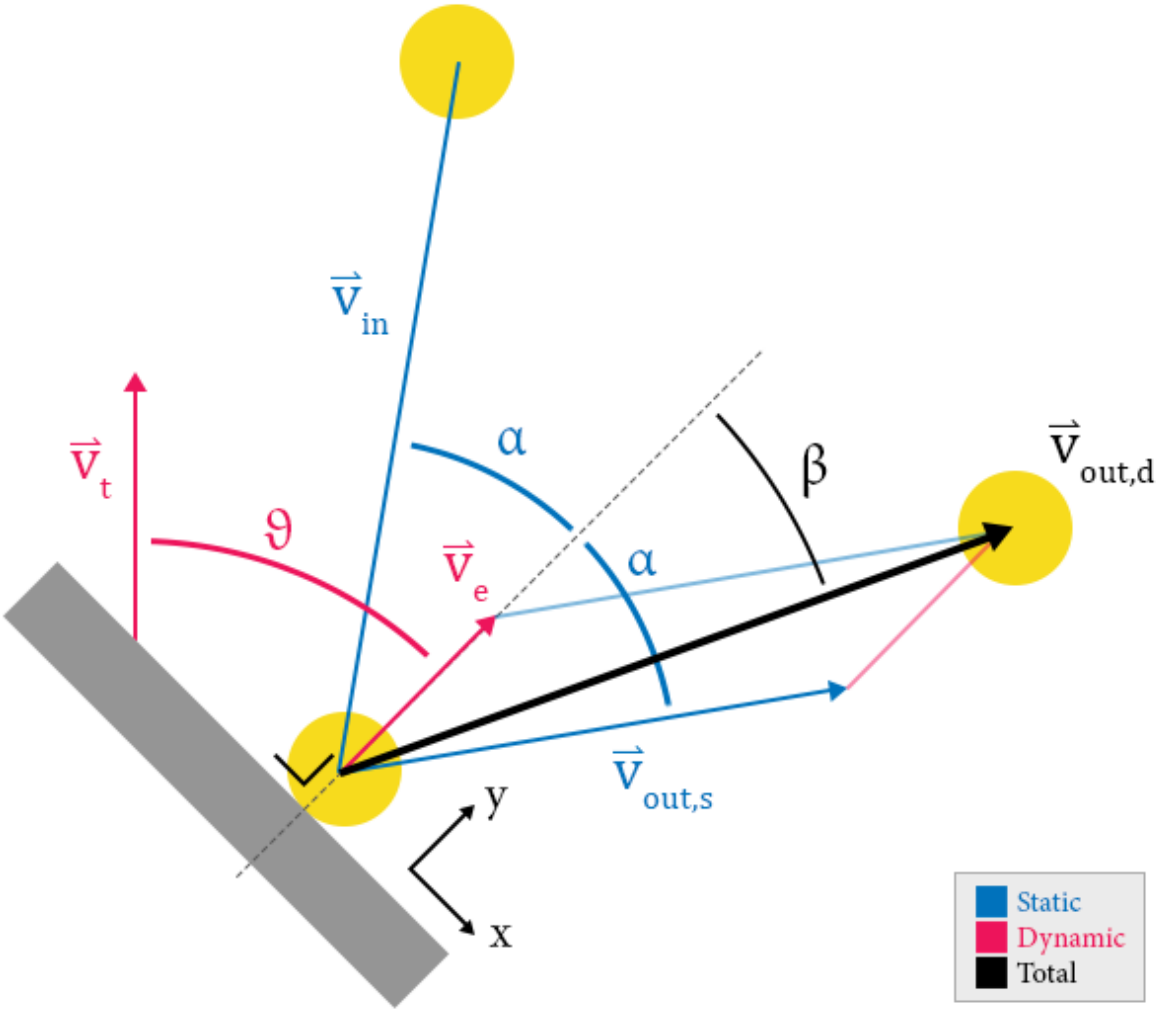

Figure 3.4 / Dynamic Push total

The $\vec{v}_{in}$ is the ingoing velocity vector of the ball, α is the angle of the ball with respect to the y-axis (the normal of the Push surface) and $\vec{v}_{out,s}$ is the static outgoing velocity vector of the ball. The $\vec{v}_t$ is the velocity vector of the Turtle and ϑ is the angle of the Turtle velocity vector with respect to the y-axis. The $\vec{v}_e$ is the extra velocity vector of the ball caused by the impact of the Turtle. The $\vec{v}_{out,d}$ is the outgoing dynamic velocity vector of the ball and β is the angle of the outgoing velocity vector with respect to the y-axis.

Superposition results in the following equation.

$$\vec{v}_{out,d} = \begin{bmatrix} \vec{v}_{out,d}\ dx \\ \vec{v}_{out,d}\ dy \end{bmatrix} = \vec{v}_e + \vec{v}_{out,s} = COR * \begin{bmatrix} |\vec{v}_{in}| * \sin(\alpha) \\ |\vec{v}_{in}| * \cos(\alpha) + |\vec{v}_t| * \cos(\vartheta) \end{bmatrix} \quad (3.4)$$

The outgoing angle can be determined using the x and y components of the final velocity vector.

$$\beta = \operatorname{atan}\left(\frac{\vec{v}_{out,d}\ dy}{\vec{v}_{out,d}\ dx}\right) = \operatorname{atan}\left(\frac{|\vec{v}_{in}| * \cos(\alpha) + |\vec{v}_t| * \cos(\vartheta)}{|\vec{v}_{in}| * \sin(\alpha)}\right) \tag{3.5}$$

The velocity of the Turtle $\vec{v}_t$ and the angle of the push surface ϑ can be set. This is done in the Push algorithm. With $\vec{v}_t$, ϑ and $\vec{v}_{in}$ known, the outgoing velocity and direction of the ball can be calculated. The calculation of $\vec{v}_t$ is discussed in Section 3.3. The velocity of the Turtle is set by sending it in the action bus. When $\vec{v}_t$ is set, the desired angle can be calculated. This is done in the function 'getDynamicPushAngleBF'. An iteration over a number of possible surface angles is executed in this function. Each iteration the outgoing ball vector is calculated for that surface angle. The surface angle with the closest outcome is chosen as desired surface angle. The calculated angle is then used as input argument for the function 'Goto_target_AH'.

The other input argument for the function 'Goto_target_AH' is the desired Turtle target position on the field. The trajectory planner computes a trajectory with a final velocity of zero to the desired position. To avoid the part of the trajectory where the Turtle slows down and thus only use the constant velocity part of the trajectory, the Turtle target is moved forward over the desired intercept point. The vector from the Turtle to the point of intercept is extended by setting the length of the vector. This length is equal to the magnitude of the calculated velocity of the Turtle, because it is assumed that the magnitude of the velocity is significantly larger than the distance of the trajectory. This velocity will be discussed in Section 3.3.

3.3 Velocity of the robot

At the start of the project the Push function delivered non consistent results in the simulation software. When the ball was rolling too slow the ball touched the wrong side of the Turtle and when the ball was rolling very fast the Turtle was unable to reach the point of intercept in time. Especially with a slow ball it is easier for the Turtle to perform a good Push because it has more time to reach the point of intercept. However, a slow ball resulted in a wrong Push.

The old situation is indicated with dashed lines. The dashed ball in Figure 3.5 indicated the ball at the moment of impact in the old implementation. The ball is hit by the edge of the Turtle and not with the Push surface. This results in an undesired ball velocity.

The calculation of the Turtle velocity was not implemented correctly. The velocity was calculated with respect to the center of the robot, while the Push action is executed with the side of the robot.

The velocity calculation in the old implementation was done as following. The distance from the Turtle to the intercept point is measured. This was the distance from the center of the Turtle to the intercept point. However, the ball is Pushed with the side surface of the Turtle. The actual distance that the Turtle has to travel is indicated with the solid arrows in Figure 3.5.

To predict when a ball arrives at a point, the velocity is measured and the deceleration is assumed to be constant. The prediction is calculated using (3.6).

$$t_b = -\frac{|\vec{v}_{in}| - \sqrt{|\vec{v}_{in}|^2 - 2 * d_b * s_n}}{d_b} \tag{3.6}$$

In the above equation t_b represents the estimated time the ball will arrive at the intercept point, $\vec{v}_{in}$ the ball velocity vector, s_b the distance from the ball to the intercept point and d_b the deceleration of the ball, which is equal to -0.089.

The Turtle needs to reach the point of intercept at the same time as the ball. The velocity of the Turtle is calculated using (3.7).

$$|\vec{v}_t| = \frac{s_n}{t_b} \tag{3.7}$$

At the start of the trajectory the robot will not have the desired velocity, in order to reach the desired velocity the Turtle will need to accelerate or decelerate. To quickly reach the desired velocity, the acceleration or deceleration is set to the maximum that the Turtle is capable of. Hence, the velocity will quickly converge to desired velocity.

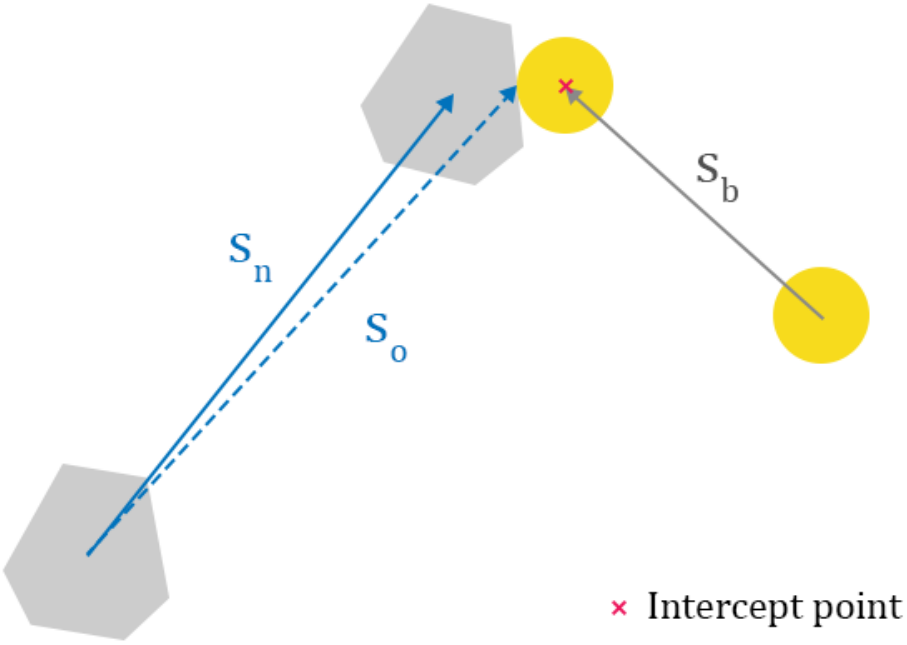

Figure 3.5 / Calculation of the Turtle velocity

s_n is the distance that is now used to calculate the velocity of the Turtle, s_o is the distance that was used to calculate the velocity of the Turtle and s_b is the distance from the ball to the intercept point.

The correction for the diameter of the ball was already implemented in the algorithm. This is done by taking the ball radius, rotating the vector such that it is perpendicular to the Push surface and then subtracting this vector from the target of the Turtle. This target is used for the distance calculation as discussed in this section.

The calculated velocity is sent to the action bus, then the Turtle drive with the calculated velocity.

The success rate is largely dependent on the ball velocity. When the ball velocity is very slow the difference becomes larger. This experiment is done by setting the ball velocity in the simulator and without any objects. This returns the same graphs as the other experiments, only the ball is passed from a different point. The old implementation tends to fail with low ball velocities. The new implementation performs better. The success rate at slow ball velocities went up from 20% to 100%. Ten repetitions were performed in both cases.

The new implementation improved the Push results. In the simulator, with obstacles, the success rate went up from 60% to 90%. Twenty repetitions were performed in both cases. The results are discussed in Chapter 4. To perform the Push experimentally, the passing velocity needs to be adjusted by the attacker main. This will be discussed in Section 3.6.

3.4 Communication

When one robot is ready to start the Push action it starts sending messages into the team bus. This is a communication channel to share information between the robots of the team. Both robots send these messages, however this information is 'local', meaning every robot knows only his own status. Therefore the information is processed in the multicast communication of the team status. In here all local databases of the Turtles are combined.

The software checks if both robots are in the same state. Then the software will overwrite the local Push state variable of both Turtles such that they can proceed. When both robots echo that they are in state 1 (communicate) at the same time, then the local variable at both Turtles will be set to 2 (give depth pass and push). The next action can be started when this local variable is equal to 2.

This can for example be used in a match to determine if a Push action should continue or if the Push action should abort. A possible reason for aborting could be when a Turtle is not able to reach the point of intercept in time because there are too many obstacles in its path. The current Push is designed so the Pushing Turtle is likely to reach the point of intercept in time, assuming there are few objects in his trajectory. This will be further discussed in Section 3.6.

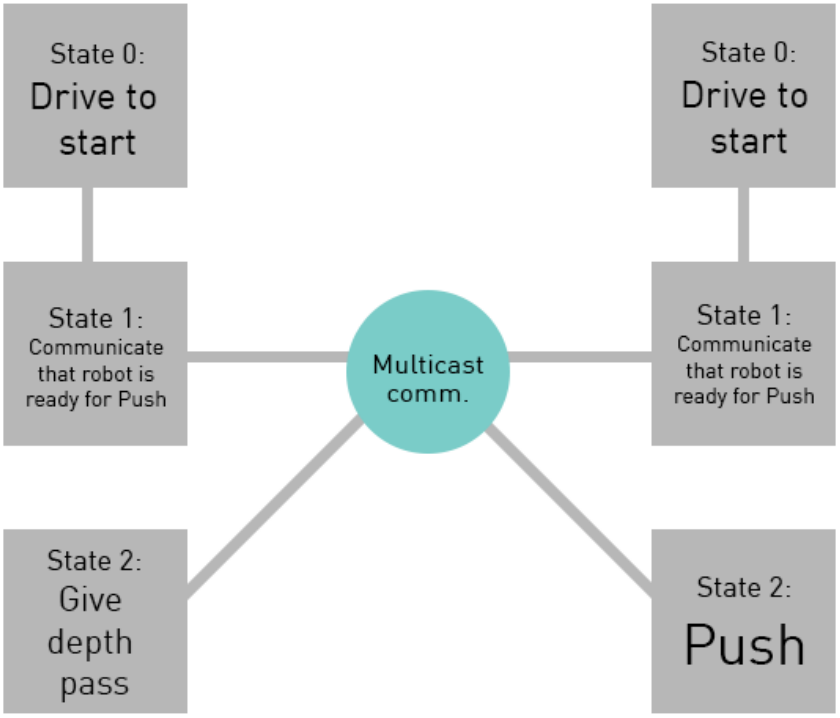

Figure 3.6 / Flowchart of communication

3.5 Dynamic Mu field

In order to determine the optimal location to give a depth pass, a Mu field is created. Mu fields are explained in Section 2.2.4. An overview of the Mu field is given in Figure 3.7.

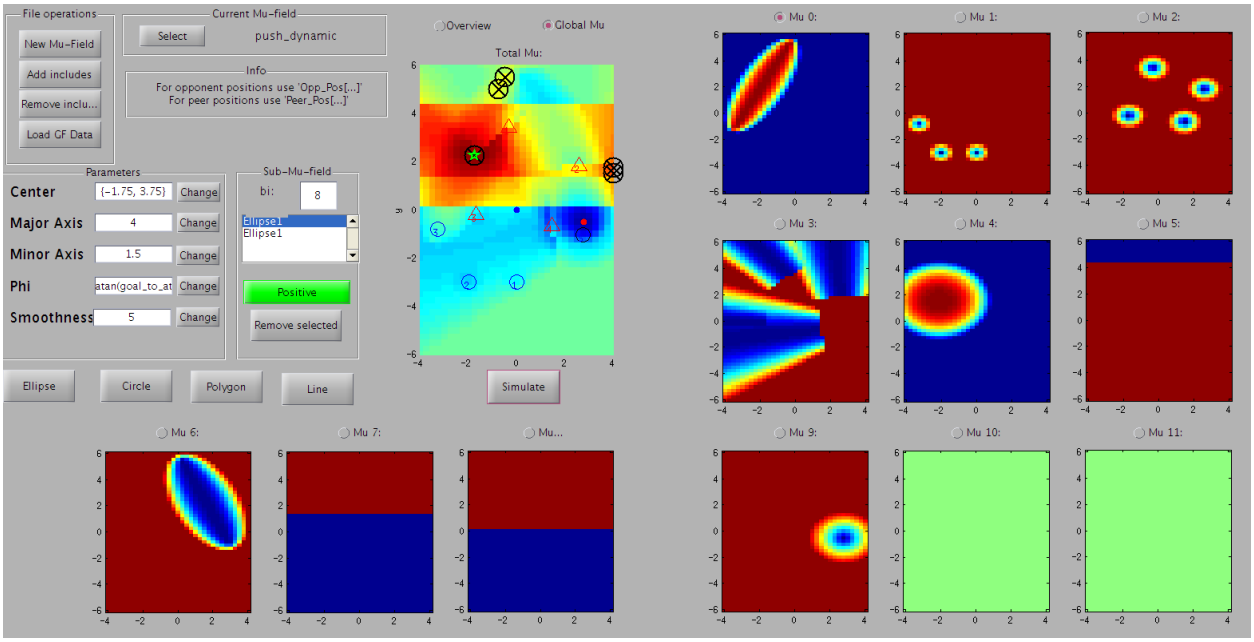

Figure 3.7 / Dynamic Push Mu field

All sub Mu fields have a number, starting from 0 at the top left, ending at 11 at the bottom right. Each Mu field will be discussed briefly.

0. The first Mu field always lies between the Pushing robot, in this case the attacker assist, and the target, in this case the goal. By placing a long ellipse there are lots of opportunities to shoot the ball. It is preferable to shoot between the robot and the target because it is fast. Other options might nog even be possible due to the time constraints. An action needs to be executed as quick as possible. This Mu field has the largest weight factor, because the ball should always be passed somewhere in this ellipse.
1. A small negative field is placed around the peer players, since it is not desirable to pass a ball at a robot.
2. Same as field 1, but now for opponent players.
3. This Mu field places a negative field behind all the opponents. The reference point is the ball.
4. This Mu field has the same center as the ellipse of field 0. It has a small weight factor. This field adds a little blur around the ellipse.
5. Field 5 prevents that the depth pass is given too close to the goal.
6. Field 6 ensures that the push angle is large enough. It does this by placing a wide, negative ellipse between the ball and the push target, the goal.

7. Field 7 ensures that the depth pass is actually a depth pass by making everything behind the middle line and 1 meter in front of the middle line negative.
8. Field 8 ensures that the depth pass is not behind the robot that is going to Push.
9. And field 9 adds a minimum distance for the Push, because it is not desirable to give a depth pass which is actually very close to the Turtle who just shot the ball.

All these Mu fields add up to the Total Mu field. The best position is computed for the total Mu field and indicated with a green star.

3.5.1 Check after Mu field

The possibly "best position" computed from the total Mu field is stored, but not the value of this solution. It is possible that this computed "best position" is not a good solution. It is possible that a Turtle is covered by opponents in such a way that no depth pass can be given. An example of such a situation is given in Figure 3.8. In this situation it is not desirable for the Turtle to give the depth pass. This still results in a position in the ellipse because it has the highest Mu value.

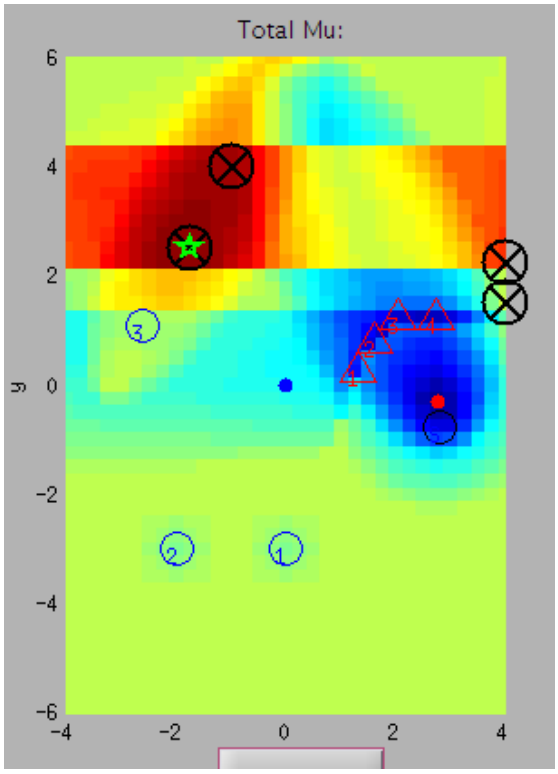

Figure 3.8 / No chance to give a depth pass

After the calculation of the Mu field, the Turtle has to determine if the depth pass can be given. The Mu fields computes a solution which is always between the Pushing robot and the target where the ball is aimed at. A check after the Mu field is done to determine if there are objects in the trajectory where the attacker main will shoot, see Figure 3.9. If there are no obstacles it passes immediately, if there are obstacles in the rectangle the robot recalculates the mu field to see if there is another option. In practice

it might be desirable to look for other skills such as passing the ball to a free player. This decision can be made after the best interception point is calculated by the mu field. In the current implementation, it cannot be determined if there is enough space for a depth pass without knowing the best interception point first.

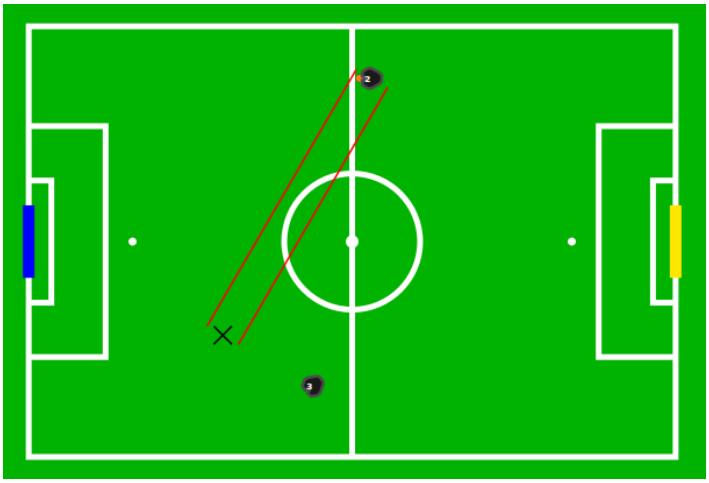

Figure 3.9 / Check after Mu field

3.6 Passing velocity

Now that the point where the attacker main will shoot is determined, it needs to determine how fast it can shoot. In most cases it is desirable to shoot as fast as possible. However, increasing the ball speed might lead to unstable Pushes when objects are involved. A compromise between accuracy of the Push and velocity of the ball has to be made.

The distance from the shooting Turtle to the desired interception point and the distance from the Pushing robot to that point are taken into account in the calculation of the parameter D. This parameter is referred to as the 'Kick effort'. The kick effort is proportional to the passing velocity:

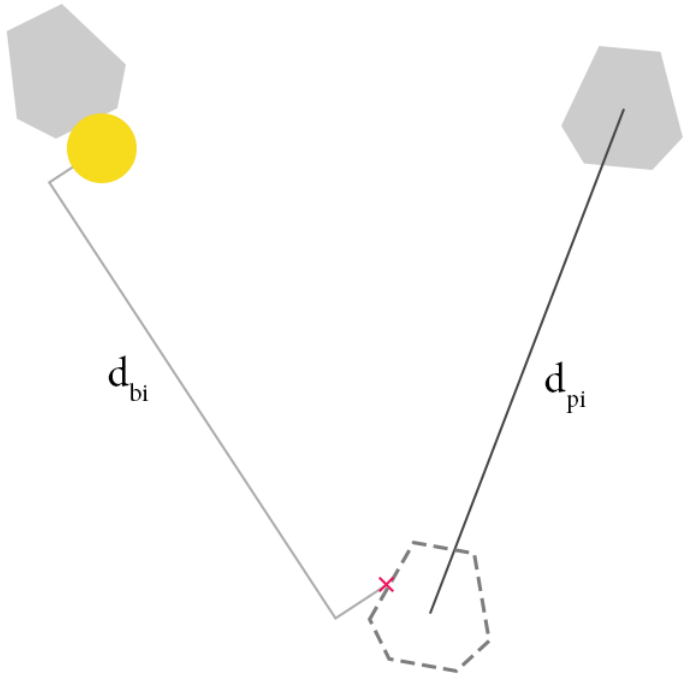

Figure 3.10 / Passing velocity parameters

$$D = A + B * d_{bi} - C * |\vec{v}_t| + D * (E - d_{pi}) \quad (3.8)$$

The d_{bi} is the distance from the ball to the intercept point, d_{pi} is the distance from the Pusher to the intercept point and $\vec{v}_t$ is the Turtle velocity vector.

D is the variable used in the algorithm where the Turtle decides how hard to shoot. A, B, C, D and E are tunable parameters. A is a Basis value for dynamic push, B is a Rate value for dynamic push, C is the Robot velocity compensation rate. D is the rate value for distance compensation and E is the offset. The velocity, distance and Pushing distance are measured.

The first term, A, is a base kick effort. The second term involves distance, the further the passing target, the faster the passing velocity will be. The third term is the compensation for the Turtle's own velocity. When it drives towards the target it will pass slower, when it drives away from the target it will pass faster. The last term is essential for a successful depth pass.

The Pushing robot needs to reach the intercept point in time, therefore the passing velocity is adapted on the Push distance. If the desired interception point lies very close to the Turtle (under 0.7 meters), the robot will pass faster, if the robot if very far from the desired interception point the passing robot will give a slower depth pass. By adapting the velocity to the Pushing robot, it will be able to reach the desired interception point fast enough to Push the ball.

3.7 Implementation of Turtle target

When the depth pass is given, the Turtle will move in the direction of the Push target, by doing so it will cross the path of the ball. While driving to the intercept point, the robot calculates the dynamic Push angle. When rotating around its own axis, the Push surface is rotated around the center of the Turtle. The robot re-calculates the Turtle target and the velocity every time, it takes ball radius and baseplate radius into account as discussed in Section 3.3. This quickly converges to a constant Turtle velocity.

As discussed in Section 3.2.2, the Turtle target is moved behind the intercept point to prevent that the Turtle reaches a velocity of zero at the intercept point. The calculated offset is applied to the Turtle target that is placed behind the intercept point. This results in an incorrect offset at the intercept point. This is shown in Figure 3.11. The grey shape indicates the position of the Turtle at the moment of impact in the current implementation.

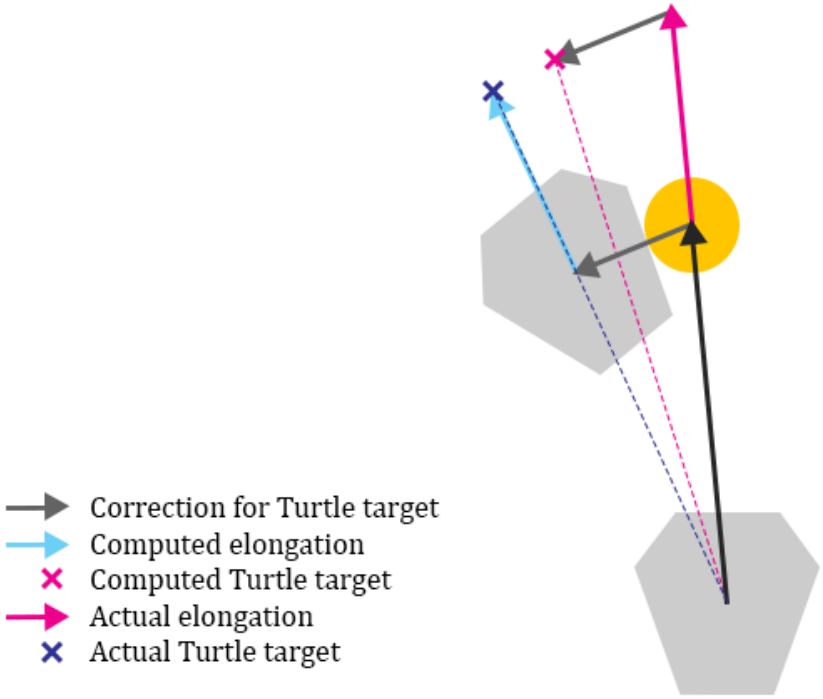

Figure 3.11 / Position offset calculation

The magnitude of the Turtle velocity is calculated correctly, however the direction of the Turtle velocity is slightly off. This results in a Push that has a constant offset of around 5 degrees. When the ball is coming from the right, the ball offset is to the left and vice-versa.

Implementing the offset at the right point should result in an even more accurate Push. Unfortunately this is not implemented due to time constraints.

4 RESULTS

4.1 Simulation

The following results are gathered with the current implementation of the Push. Figure 4.1 shows an example of a Push in simulation software. The lines are the trajectories of the ball and the Turtle. The crosses represent the center positions at the moment of impact. The Push target lies at (0,-9).

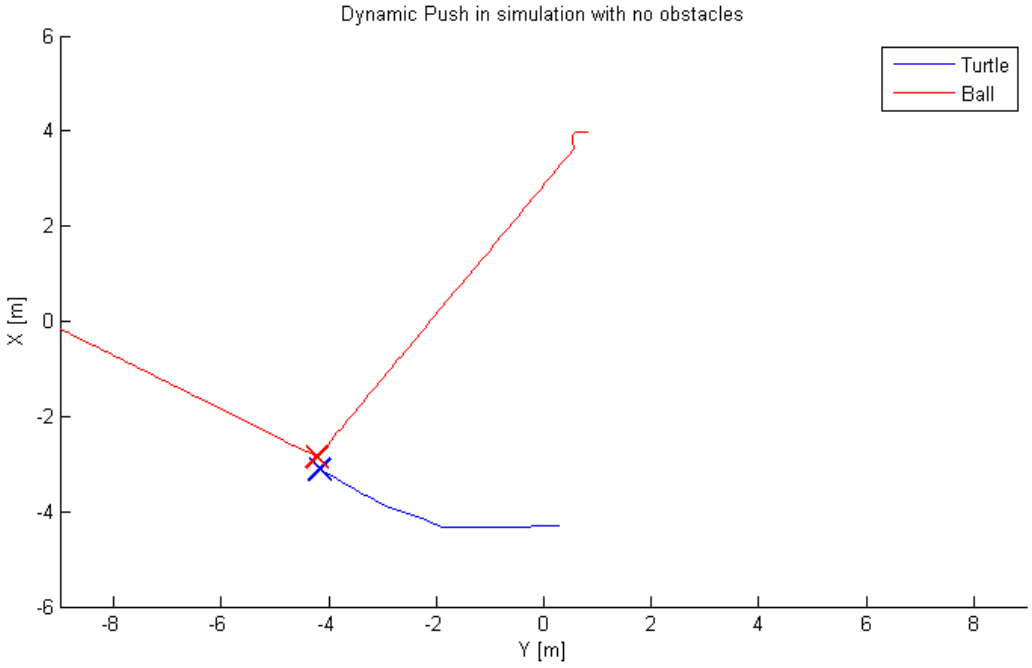

Figure 4.1 / Dynamic Push in simulation with no obstacles

The above simulation is performed 20 times. The Push success rate was 100%.

In this simulation, two obstacles are involved. The obstacles are moved to a different position every simulation.

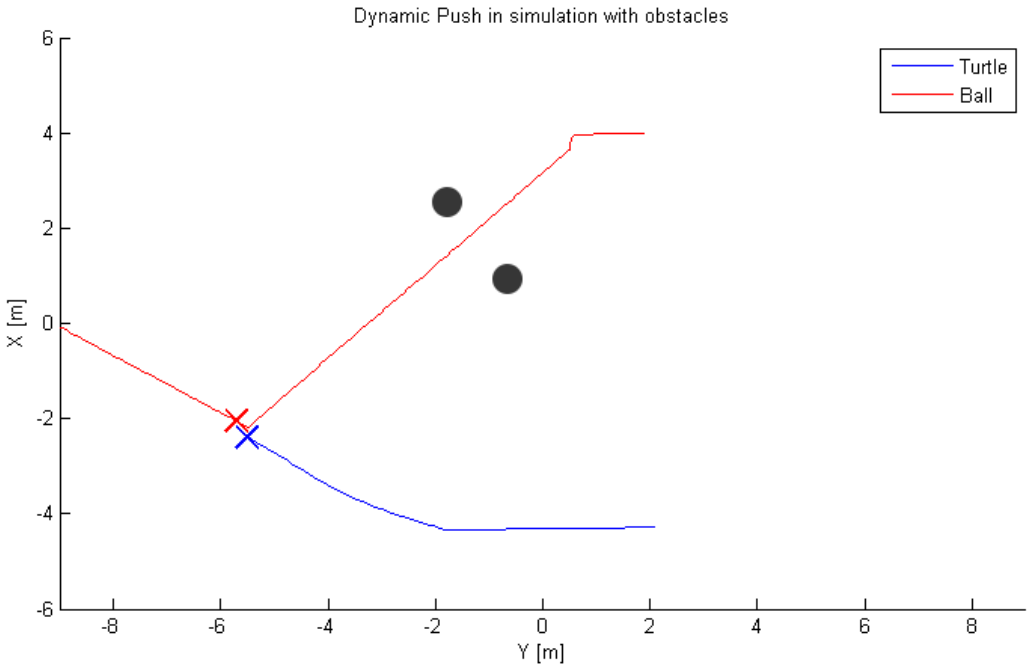

Figure 4.2 / Dynamic Push in simulation with obstacles

This simulation is performed 20 times. In the case above, one obstacle was located above the red line, the other one below the red line. The Push was successful 18 times. A success rate of 90% was achieved. The 2 times it went wrong was due to inaccurate passing. In those two simulations the Push target lied close to the goal, combined with a passing offset of 5 degrees this resulted in a ball that was aimed directly at the goal. Since the Turtles will not drive too close to the goal, these balls were not reached by the robots. All balls that the robot could reach were Pushed correctly in the goal.

This simulation the obstacles are placed in the trajectory of the attacker assist, the Pusher.

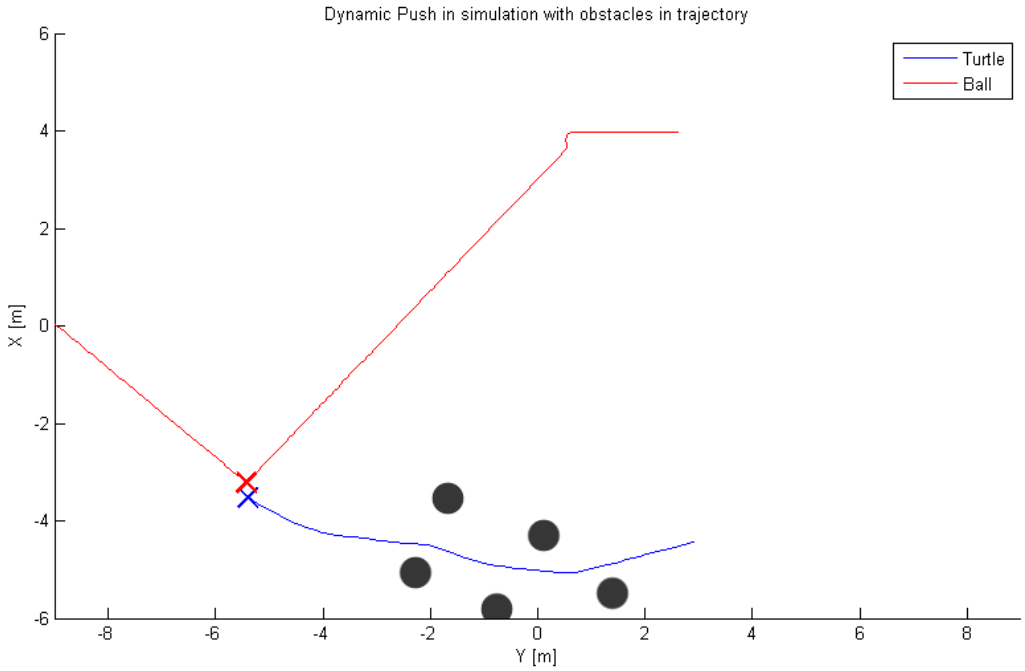

Figure 4.3 / Dynamic Push with obstacles in the trajectory of the Pusher

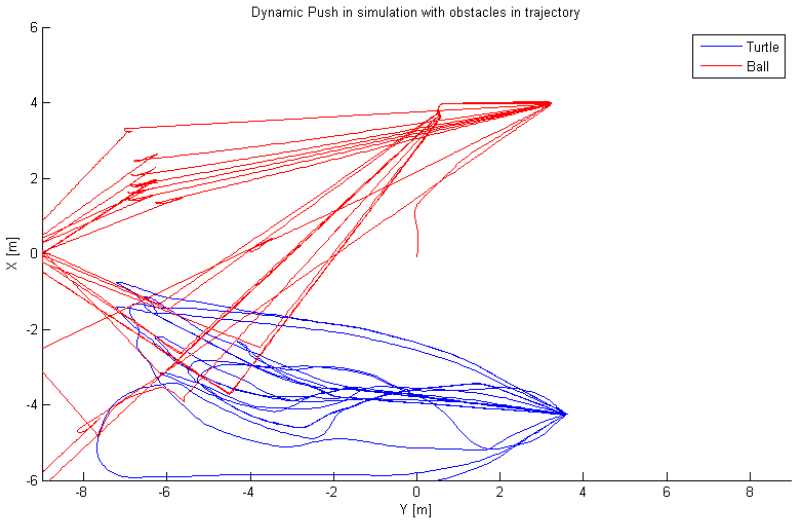

Figure 4.4 / Multiple dynamic Pushes with different obstacles in the trajectory of the Pusher

This simulation was performed 30 times. A success rate of 80% was achieved. The failures were caused by too many obstacles on the field. This resulted in a long Turtle trajectory which caused the Turtle to arrive when the ball was already past the intercept point.

4.2 Experiments

Now that the simulations performed well, experiments will be performed on the field. There was one obstacle involved which was placed somewhere between the shooter and the path of the Pusher.

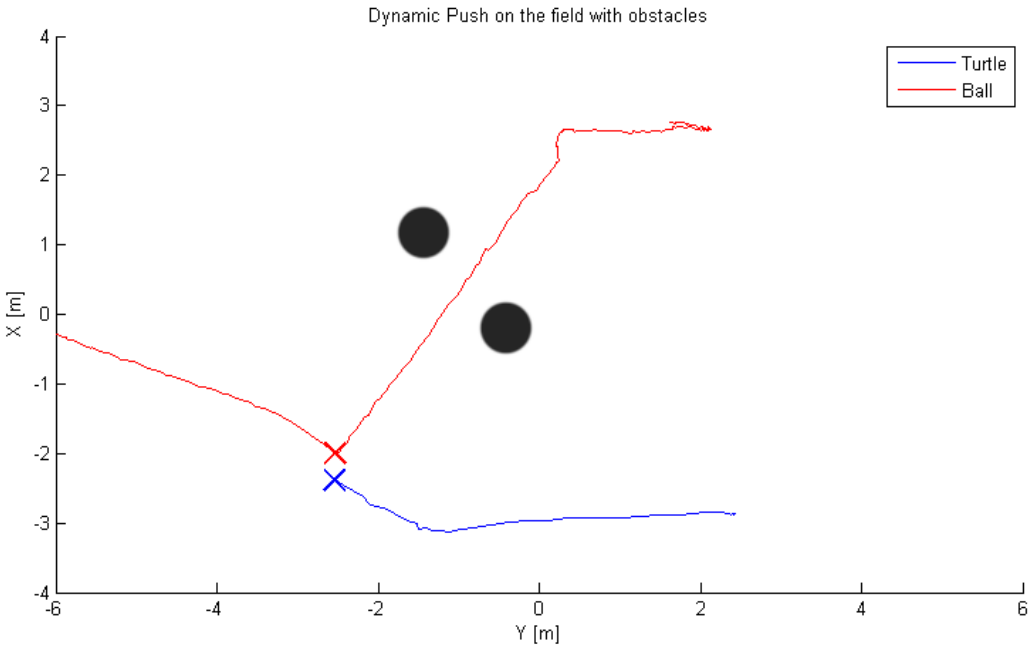

Figure 4.5 / Dynamic Push on the field with obstacles

Without tuning or modifying the algorithm and its parameters, experiments on the field were performed. There was one obstacle involved that was moved around the pitch. The Push was executed 10 times, whereof 9 successful. The 9 successful Pushes resulted in a ball in the center of the goal. One time the ball hit the left post of the goal. The velocity of the Turtle was low this time, this resulted in a ball that was spinning relatively more than usual. That caused the ball to hit the post of the goal.

5 CONCLUSION

The dynamic Push is more robust by solving an implementation mistake of the previous function. It works 90% of the time on the field in a dynamic situation involving different positions and obstacles. These results suggest that a small error in the angle of the Turtle has a relatively large effect on the outgoing ball. Therefore it is important to calculate the angle of the Turtle precisely. A possible best position for a depth pass is calculated with a Mu field. This "best" solution needs to be checked so that it is free of obstacles before the depth pass is given. The passing velocity of the depth pass is adapted according to the situation. To increase Push robustness, the parameters used to calculate the kick effort can be lowered, resulting in a lower ball velocity. The goal of the project *"Develop and implement an algorithm that allows to Push the ball to a certain target in a dynamic situation."* is achieved in the simulations and experiments. The accuracy of the Push might be improved by implementing the Turtle target, as discussed in section 3.7, correctly. According to the experiments, 90% of the cases the ball was intercepted and pushed towards the goal.

5.1 Future work

Implementing the offset correctly as discussed in Section 3.7.

5.2 Recommendations

The following recommendations will improve the Push and the soccer robots in general.

1. A strip of stiff material on the push surface so that less energy is dissipated when Pushing a ball. This material has to withstand all the collisions a robot makes with other robots.
2. Add flippers to the push surfaces of the Turtles to give extra velocity to the ball. This also adds a large number of other possibilities, such as quick passing without having to rotate a full 180 degrees. A normal soccer player can move his foot in many directions to shoot the ball, why would we limit the robot to have only one possible shooting direction?
3. Passing more accurate at a desired point. Some passes are inaccurate up to 5 degrees. When it is desired to pass through two opponents who are close to each other, this inaccuracy could cause the ball to collide with an opponent robot.

6 APPENDIX

Code documentation will be provided along the report on an USB stick with a readme file which explains how to implement the software.

Most important files:

Push action	trunk/src/Turtle2/Motion/src/action_handler/action_functions.c @ line 3380
Mu field	trunk/src/Turtle2/Motion/src/mu_positioning_lib/mu_field_push_dynamic.c
Call of mu field	trunk/src/Turtle2/Motion/src/role_handler/role_attacker_main.c
Call of Push	trunk/src/Turtle2/Motion/src/role_handler/role_attacker_assist.c

7 BIBLIOGRAPHY

[1] About RoboCup - www.robocup2014.org/?page_id=238
[2] M.J.G. van de Molengraft, A.A. Basten, O. Hendriks and R. Hoogendijk. Pushing Further. Master's thesis, Eindhoven University of Technology, August 2013. - www.techunited.nl/wiki/index.php?title=Pushing_Further
[3] M.J.G. van de Molengraft, J.W. Lamers, and R. Hoogendijk. Push: Alternative ball handling for the Turtle soccer robots. Master's thesis, Eindhoven University of Technology, August 2012. - www.techunited.nl/wiki/index.php?title=Push:_Alternative_ball_handling_for_the_Turtle_soccer_robots
[4] 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.