Push: Alternative ball handling for the Turtle soccer robots

ing. J.W. Lamers

CST 2012.049

Premaster end project

Coach(es):  ir. R. Hoogendijk
Supervisor:  dr. ir. M.J.G. van de Molengraft

Eindhoven University of Technology
Department of Mechanical Engineering
Control Systems Technology

Eindhoven, August, 2012
5.2 Recommendations ................................................................. 20
  5.2.1 Dynamic movement Turtle .............................................. 20
  5.2.2 Adding “flippers” to the sides ......................................... 20
  5.2.3 Learning algorithms ...................................................... 20
  5.2.4 Redesign housing of the Turtle ........................................ 20

Bibliography .............................................................................. 21

A  Experiment results .............................................................. 22

B  C code .................................................................................. 24
Chapter 1

Introduction

1.1 Background

RoboCup[1] is an international scientific initiative established in 1997 with the goal to advance the state of the art of intelligent robots. The original mission was to field a team of robots capable of winning against the human soccer World Cup champions by 2050. Since 2005, Tech United Eindhoven[2] has been participating in the mid-size league of the RoboCup competition. In this competition robots compete in a completely autonomous soccer match. Tech United has been competing in the top of the RoboCup competition for several years now with as best result winning the 2012 world championship in Mexico after being 4 times world champion runner-up. The robots of Tech United are called Turtles which stands for Tech United RoboCup Team Limited Edition.

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 or shoot the ball to a peer player or preferably in the opponents goal. In certain game situations such as a corner or free kick it is desirable that a robot shoots the ball directly into the ball handling mechanism of a peer player. Two disadvantages of the the ball handling mechanism are:

1. Intercept with ball handling often goes wrong. Takes quite some time to recapture the ball.
2. Intercept, aim and shoot to a certain target takes too much time.

A possibility to avoid shooting directly into the ball handling mechanism would be by bouncing the ball to a desired direction with the housing of the robot. This eliminates the interference of the ball handling, which may result in a faster ball handling.
1.3 Objective

The main objective is to develop a software function that gives the Turtle soccer robots the ability to bounce an incoming ball to an in advance determined target using the housing of the Turtle.

This new software function has to be given a suitable name to distinct between the consisting functions of the Turtle. A available and suitable label/name would be "push function". The current housing design of the Turtle has a triangle like shape where the marked surfaces depicted in Figure 1.1 are suitable for bouncing a ball to a desired target. The software function has to be written in the C programming language and has to be compatible with the current software structure used for the Turtle soccer robots. The Turtle soccer robots are already equipped with additional soft- and hardware functions to determine its position, velocity and angle on the field. The optical sensors on top of the robots give information of the position and velocity of the ball. This data is available and accessible for the C software function.

![Figure 1.1: Suitable sides for bouncing a ball marked in red.](image)

1.4 Outline

Prior to the development of the push function, an experiment is conducted to analyse the behaviour of the ball when it bounces off the housing of the Turtle. The experiment set-up and its results are reviewed in Chapter 2. With the results of the experiment a Matlab simulation with the required functions and mathematical computations to construct a C software function to push a ball with the housing of the Turtle is explained in Chapter 3. The computations developed in the Matlab simulation are used to construct the C software sequences with its additional sub parts such as bail-out conditions and debug information. The layout of the C program structure is explained in Chapter 4. In Chapter 5 conclusions are drawn and recommendations for further research are given.
Chapter 2

Experiment

2.1 Introduction

2.1.1 The Law of Reflection

The general idea behind bouncing a ball using the housing of the Turtle is similar to what you observe during tennis matches. Here, tennis players predict the trajectory of the ball by using the law of reflection. The law of reflection is one of the simplest rules in science. It states that when an object bounces off a flat surface, the angle at which it hits the surface will be equal to the angle at which it bounces away. This behaviour was first discovered through careful observation and measurement by Hero of Alexandria[3]. Both angles are measured from the path of the object to the normal line, which is a line perpendicular to the surface at the point of impact. Under ideal conditions, the law of reflection holds

\[ \theta_r = \theta_i, \]  

(2.1)

where \( \theta_r \) denotes the angle of reflection and \( \theta_i \) denotes the angle of incidence. With 'ideal conditions' the following criteria hold:

- The collision is simple. The ball is not spinning, and the surface it hits is immobile.
- The ball is perfectly round, and its mass is symmetrically distributed.
- No other unbalanced force is applied to the ball to change its direction.
- The surface is truly flat, and if it is compressed by the object hitting it, it compresses and uncompresses uniformly and in the direction of the normal line.
2.1.2 Purpose of the experiment

It is unlikely that the ideal conditions sketched in Section 2.1.1 are present during a RoboCup match. For example, when a Turtle shoots a ball it receives a certain spin and it will rebound at a different angle than the angle of incidence. Besides, the ball and the field conditions have to be optimal. When the shape or the mass distribution of the ball differs from perfectly symmetric, the path of the ball would not be a straight line, and the angles would not be equal. The field surface has to be perfectly flat, or the ball may rebound at an unpredictable angle. Bits of fluff on the field, or bumps, may cause the path of the ball to change unpredictably. Besides the field and ball conditions the surface material of the housing of the Turtle is of great importance. The surface of the current housing of the Turtle is covered with a 1 cm foam layer to protect the referees during a RoboCup match. This layer will properly have a large effect on the bounce behaviour of the ball when it hits the Turtle. To determine how the ball reacts an experiment is conducted to analyse the behaviour of the ball when it bounces of the housing of the Turtle and to verify whether the incoming angle is equal to the outgoing angle as in (2.1).

2.2 Experimental set-up

To determine the behaviour of the ball when it bounces of on the housing of the Turtle an experiment is conducted with two Turtle soccer robots. Turtle A in Figure 2.3 shoots a ball onto the stationary target Turtle B. In this experiment several shots are fired at the stationary target with different angles of incidence and different velocities from a distance of 3 meter. The angle of incidence varied from perpendicular (0°) to an incidence angle of 60° with increments of 15° as depicted in Figure 2.2. At each angle of incidence the ball velocity varied from 15% to 40% of the maximum shot force with increments of 5% between each shot. With a shot force lower then 15% the ball will move too slow while at a force higher then 40% the ball will bounce too much into the air. To avoid measurement errors all shots are repeated five times. To acquire all the measurement data the target Turtle is equipped with additional software to log the following data with a frequency of 1000 Hz: the position and velocity (x, y, dx, dy) of the ball and the location and rotation (x, y, ϕ) of the Turtle. The experiment was conducted on the field in the RoboCup stadium at Eindhoven University of Technology with a standard soccer ball, that has a diameter of 22 cm and a pressure of 0.6 bar.

Figure 2.2: Variation in angle of incidence.

Figure 2.3: Schematic representation experimental set-up.
2.3 Experimental results

2.3.1 Raw data processing

The experiment described in Section 2.2 leads to a large data file containing over 150 ball-Turtle collisions. Each millisecond during the experiment a new row is added to the data file containing the position and velocity of the ball at that moment. A graphical representation of the raw data in the file is depicted in Figure 2.4 where the data is plotted on a time scale. For an improved interpretation of the experiential results the x and y coordinates of the position of the ball are plotted on a x-y coordinate system. As result the raw data transformed to a more readable form depicted in Figure 2.5 in which the shape of the target Turtle (Turtle B in Figure 2.3) and a line perpendicular to the bounce surface has been added. In appendix A all the measurement results are shown.

Figure 2.4: Raw measurement data: shooting at Turtle with a 15 degrees angle of incidence.

Figure 2.5: Enhanced measurement data: shooting at Turtle with a 15 degrees angle of incidence.
2.3.2 Single bounce analysis

Due to the shooting inaccuracy of the Turtle A (Figure 2.3) and the fact that Turtle B was standing still several balls missed the push surface of the target. The measurement data of the missed balls is not taken into account for determining the reaction of the ball on the push surface, only the successful shots will be analysed. By plotting the results of a single successful bounce in Matlab, the incoming trajectory and outgoing trajectory of the ball can be calculated. By detecting the sign of the velocity, Matlab separates the incoming and outgoing measurement data of the ball into two arrays of x and y coordinates. Using Matlab’s polyfit algorithm these data arrays form the linearised incoming and outgoing vectors depicted in Figure 2.6. Besides the incoming and outgoing angle the velocity change caused by the collision of the ball with the push surface of the Turtle can be extracted as well. The ingoing velocity of the ball in the measurement depicted below was 2.02 m/s and the outgoing velocity was 1.33 m/s which is a velocity reduction of 34.2%.

![Figure 2.6: Single bounce at Turtle with a 15 degrees angle of incidence.](image)

2.4 Conclusion

By evaluating all the successful bounces at the push surface of the Turtle the following result is established about the bounce behaviour of the ball. After a collision with the housing of the Turtle the velocity of the ball decreases in the range of 25%~35% of the incoming velocity. The velocity reduction varies a lot and is therefore unpredictable and incalculable. The main reason why the velocity of the ball greatly decreases after a bounce is related to the attached foam layer on the housing of the Turtle as mentioned in Section 2.1.2. However, the outgoing trajectory of the ball after a bounce of the housing of the Turtle is quite predictable, it almost equals the angle of incidence with small variations in the range of -3° ~ 3°. Evaluating all the unsuccessful shots many shots shot with an angle of incidence in the range of 45° and 60° missed or only grazed the Turtle’s push surface. With these angles of incidence the push surface seen from the perspective of the ball is relative small and it is not advisable for larger angles of incidence then 45°.

Concluding, the law of reflection is applicable on a ball that bounces of the housing of the Turtle if the angle of incidence is not to large causing the ball to only graze the push surface instead of resulting in a predictable bounce.
Chapter 3

Matlab simulation

3.1 Introduction

The results of the experiment described in Chapter 2 have shown that the law of reflection (2.1) is applicable to a ball that bounces of the housing of the Turtle. With these results the desired push function described in the objective (Section 1.3) may benefit of the law of reflection equation $\theta_r = \theta_i$. During the development of a software function a simulated environment can be useful to determine whether the applied computations result in the desired result. Because such simulations are useful a simulation has been made for the push function of the Turtle. In this Chapter the mathematical computations used in the simulation will be explained.

3.2 Computations push function

3.2.1 Point of intercept

Before a Turtle is able to push a ball with its housing, it has to position itself in the trajectory of the ball. To predict this ball trajectory the Turtle is equipped with a camera to provide the necessary information such as the velocity and position of the ball and its own position and rotation. When the Turtle is outside the trajectory of the ball it needs to move as fast as possible inside this trajectory to intercept the ball. The fastest path would be the shortest path which is a path perpendicular to the trajectory of the ball as depicted in Figure 3.1 with possible obstacles disregarded.

![Figure 3.1: Perpendicular intercept trajectory Turtle.](image_url)
To calculate the x and y coordinates of the point of intercept (poi) on the field two linear equations with two variables have to be solved. Both the trajectory of the ball and the trajectory of the Turtle satisfy the general linear equation

\[ y = mx + c, \]  

where \( m = \frac{\dot{y}}{\dot{x}} \) = the gradient or slope of the line and \( c = \) the point at which the line crosses the y-axis.

Applying (3.1) on the ball gives

\[ y_{\text{ball}} = m_{\text{ball}}x_{\text{ball}} + c_{\text{ball}}, \]  

with \( m_{\text{ball}} = \frac{\dot{y}_{\text{ball}}}{\dot{x}_{\text{ball}}} \) and \( c_{\text{ball}} = \) the point at which the line crosses the y-axis. The Turtle trajectory satisfies

\[ y_{\text{Turtle}} = m_{\text{Turtle}}x_{\text{Turtle}} + c_{\text{Turtle}}, \]  

where \( m_{\text{Turtle}} = -\frac{1}{m_{\text{ball}}} \) and \( c_{\text{Turtle}} = \) the point at which the line crosses the y-axis. In (3.3) the \( m_{\text{Turtle}} \) is calculated based on the theorem in which states that when a line is perpendicular to another line \(-1 = m_{\text{Turtle}}m_{\text{ball}}\). By substitution and rearranging (3.2) and (3.3) the x-coordinate of the poi can be found with

\[ x_{\text{poi}} = \frac{c_{\text{Turtle}} - c_{\text{ball}}}{m_{\text{ball}} - m_{\text{Turtle}}} \]  

and the y-coordinate with

\[ y_{\text{poi}} = m_{\text{ball}}x_{\text{poi}} + m_{\text{ball}}. \]  

### 3.2.2 Push angle

To push a ball with the housing of the Turtle based on the law of reflection \( \theta_i = \theta_r \) the angle in which the Turtle has to position itself on the soccer field needs to be computed. The angle in which the Turtle has to position itself on the soccer field is called the push angle. To compute the push angle the point of intercept, ball trajectory and desired target location need to be known. Prior of the computation of the push angle, the angle of the normal line on the housing of the Turtle is determined.

![Figure 3.2: Schematic representation push angle.](image-url)
To calculate the normal angle $\theta_{normal}$ the normalized vectors poi-ball and poi-target are added. The x-coordinate of the combined vectors is computed with

$$x_{normal} = \cos(\theta_{ball}) + \cos(\theta_{target})$$

and the y-coordinate with

$$y_{normal} = \sin(\theta_{ball}) + \sin(\theta_{target}).$$

$\theta_{normal}$ is then determined by

$$\theta_{normal} = \arctan\left(\frac{y_{normal}}{x_{normal}}\right).$$

The coordinate system depicted in Figure 3.3 represents the field where the upward pointing black arrow is directed to the opponents goal and represents $0^\circ$. When $\theta_{normal}$ is computed the push angle at which the housing of the Turtle has to align with is obtained by a perpendicular line to the normal line indicated with the green line in Figure 3.2.

\[\text{Figure 3.3: Determination of the normal angle between the ball and target.}\]
3.2.3 Determine optimal push side

Due to the triangular shape of the housing of the Turtle there are two available sides to push a ball as depicted as the red and blue Turtles in Figure 3.4. Seen from the heading of the ball handling they are called left and right side as in Figure 3.5. Prior for moving to the point of intercept, the Turtle has to make a decision between either the left or right side. This will minimize the required time for the Turtle to prepare itself for a push action because the Turtle is then able to rotate during the distance it needs to travel.

Figure 3.4: Two options sides to push.  

After the normal line is determined (Section 3.2.2) the two possible push options \( \theta_{\text{left}} \) and \( \theta_{\text{right}} \) in which the front of the Turtle has to point to preform a push action are known due to the shape of the housing of the Turtle. These two options are

\[
\theta_{\text{left}} = \theta_{\text{normal}} + 120^\circ \quad (3.9)
\]

or

\[
\theta_{\text{right}} = \theta_{\text{normal}} - 120^\circ. \quad (3.10)
\]

By comparing the start angle \( \theta_{\text{start}} \) with the results of (3.9) or (3.10) the smallest value of the comparisons defines the optimal push side. For further computations the optimal push angle (\( \theta_{\text{left}} \) or \( \theta_{\text{right}} \)) is called \( \alpha \).

\( \theta_{\text{start}} \) is the angle in which the front of the Turtle points when a push action is initiated.
3.2.4 Final target position and rotation

The Turtle soccer robot is able to position itself on the field based on the parameters x-coordinate, y-coordinate and an angle $\varphi$. These $x$ and $y$ coordinates are based on the center of the Turtle and the angle $\varphi$ indicates the angle in which the front of the Turtle points where $0^\circ$ is towards the opponents goal. For a successful push action the angle in which the front of the Turtle needs to point has to be equal to $\alpha$ (determined in Section 3.2.2) and the center of the push surface on the housing of the Turtle needs to be positioned at the point of intercept coordinates (determined in Section 3.2.1). The final position and rotation of the Turtle to preform a successful push action are called $\text{target}_{x,y,\varphi}$. Because the center of the push surface on the housing of the Turtle differs of the center coordinates of the Turtle a correction to the final coordinates is needed.

To compute the $x,y$ correction ($\text{correction}_{x,y}$) the center coordinates of the Turtle is placed on the origin of a virtual local coordinate system as depicted in Figure 3.6. The center point of the push surface of the housing of the Turtle seen from the origin point is $-8.25$ cm in the y-direction and either $\pm 17.75$ cm in the x direction depending on the determined push side. Using the theorem of Pythagoras the deviation of the center point of the push surface to the origin point is $\beta = 0.1957 = \sqrt{(-8.25)^2 + (\pm 17.75)^2}$. The origin of the virtual local coordinate system is placed at the point of intercept with the front of the Turtle pointed in the $\theta_{\text{normal}}$ direction depicted as the dashed Turtle in Figure 3.7. Seen from the local coordinate system the Turtle has to move $\beta$ in the y-direction as depicted 3.7. To compute the correction coordinates the local coordinates are multiplied with a coordinate-transform matrix resulting in

$$
\begin{bmatrix}
\text{correction}_x \\
\text{correction}_y
\end{bmatrix} = 
\begin{bmatrix}
0 \\
\beta
\end{bmatrix} 
\begin{bmatrix}
\cos(\theta_{\text{normal}}) & -\sin(\theta_{\text{normal}}) \\
\sin(\theta_{\text{normal}}) & \cos(\theta_{\text{normal}})
\end{bmatrix}
\begin{bmatrix}
\text{poi}_x \\
\text{poi}_y
\end{bmatrix}.
\tag{3.11}
$$

The final target position and rotation on the field is computed with

$$
\begin{bmatrix}
\text{target}_x \\
\text{target}_y \\
\text{target}_\varphi
\end{bmatrix} = 
\begin{bmatrix}
\text{poi}_x \\
\text{poi}_y \\
\theta_{\text{normal}}
\end{bmatrix} + 
\begin{bmatrix}
\text{correction}_x \\
\text{correction}_y
\end{bmatrix} + 
\begin{bmatrix}
\alpha
\end{bmatrix},
\tag{3.12}
$$

where $\alpha$ is determined in Section 3.2.3 and $\text{poi}_x, \text{poi}_y, \theta_{\text{normal}}, \text{correction}_{x,y}$ are computed with respectively (3.6), (3.7), (3.8), (3.11).

Figure 3.6: Virtual coordinate system with the origin at center of the Turtle.

Figure 3.7: Aligning the push surface of the Turtle with the poi.
Chapter 4

C program

4.1 Introduction

The software used in Turtle soccer robots is written in the programming language C. As mentioned in the objective (Section 1.3) the push function is an extension on the available action functions of which the strategy algorithm of Turtle soccer robot can choose during a RoboCup match. Therefore, the push function is added in the `action_function.c` file which is located in the software directory `action_handler` of the Turtle. In this chapter the layout of push the functions is explained and the final C code is included in appendix B.

4.2 Program structure

After the strategy algorithm suggested a push action, the program sequence depicted on the next pages starts see Figure 4.1. The push function consists out of the computations developed in the Matlab simulation described in Section 3.2. To increase the programs modularity these computations are split into a main function and several sub functions. Besides the main function and several sub functions there is additional code for bail-out situations and debug/test information to make the program more robust and readable.

4.2.1 Main function: Push

The main function `push` is responsible for positioning the Turtle at the correct place and angle on the field to push the ball to a desired location. The main function starts after it is triggered by the strategy algorithm. Immediately, after the start of the main push functions, all variables such as the location and angle of the Turtle and velocity of the ball are initialized. The function consists of computations to predict the trajectory of the ball, point of intercept and final position and rotation. The final coordinates $target_{x,y,\phi}$ are then used by the `Goto_target_AH` command to move the Turtle to its final destination. Besides the initialization of the variables, the computations to determine the final position are executed every millisecond. The square shape symbols in Figure 4.1 represent a computation with the input data on the left side and the result on the right side.

4.2.2 Sub function: Determine left or right

The sub function `determine_left_or_right` is responsible for calculating the optimal push side using the computations described in Section 3.2.3. The main function executes this function every millisecond
to determine the angle correction for the final position.

Figure 4.1: Program sequences.
4.2.3 Sub function: Check if possible

The sub function \textit{Check\_if\_possible} determines if the Turtle is able to perform a successful push action. Different from the main function, this sub function is only executed once 500 ms after the start of the push function. During those 500 ms the average x, y-direction velocity of the ball is stored. This data is used for other computations in this sub function. The average value is taken to prevent possible computation with accidentally incorrect optical sensor values. The average is determined using a moving average algorithm

\begin{equation}
\dot{x}_{avg} = 0.9\dot{x}_{avg} + 0.1\dot{x}_{current} \tag{4.1}
\end{equation}

for the x-velocity and for the y-velocity

\begin{equation}
\dot{y}_{avg} = 0.9\dot{y}_{avg} + 0.1\dot{y}_{current}. \tag{4.2}
\end{equation}

To perform a successful push action the Turtle has to arrive at the point of intercept prior the ball arrives. Therefore, the travel time of the ball

\[ time_{ball} = \frac{dest_{ball}}{ball_{avgvelocity}} \tag{4.3} \]

and the travel time of the Turtle

\[ time_{Turtle} = \frac{dest_{Turtle}}{Turtle_{vmax}} \tag{4.4} \]

are predicted, where \( dest_{ball} = \sqrt{(poi_x - ball_x)^2 + (poi_y - ball_y)^2} \), \( dest_{Turtle} = \sqrt{(poi_x - Turtle_x)^2 + (poi_y - Turtle_y)^2} \), parameter \( Turtle_{vmax} \) which is the maximal Turtle velocity and the \( ball_{avgvelocity} \) which is computed using the stored average ball velocity (4.1) and (4.2). To assess whether the Turtle arrives at the point of intercept prior the ball arrives (4.3) and (4.4) are compared.

\[ 0,5 \quad 1,0 \quad 1,5 \]
\[ 0,5 \quad 1,0 \quad 1,5 \]
\[ 0,0 \quad x \]
\[ y \]
\[ dest_{ball} \]
\[ dest_{Turtle} \]
\[ ball_{x,y} \]
\[ poi \]
\[ Turtle_{x,y} \]
\[ \theta_{push} \]

Figure 4.2: Check if possible.
One of the conclusions stated in the experiment (see Section 2.4) advises not to bounce a ball with a too large angle of incidence therefore the \textit{maxpushangle} parameter is introduced. To determine if $\theta_{\text{push}}$ in Figure 4.2 is less than the parameter \textit{maxpushangle} a comparison of both values is executed where $\theta_{\text{push}}$ is computed by

$$\theta_{\text{push}} = \arccos(\text{normvec}_{\text{ball}} \cdot \text{normvec}_{\text{Turtle}}),$$

where $\text{normvec}_* = \frac{\* - \text{poi}_{\text{dest}}}{d_{\text{dest}}}$.

### 4.2.4 Bailout conditions

To prevent an infinite push function loop, six bail-out conditions listed in Table 4.1 are added to the C code to abort the function. These bailout conditions are continually monitored during a program cycle and while the program is running in debug mode the associated debug message is displayed on the screen.

<table>
<thead>
<tr>
<th>Bailout code</th>
<th>Debug message</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ball velocity to low</td>
<td>Triggered in the initialization phase of the main push function. (Section 4.2.1)</td>
</tr>
<tr>
<td>2</td>
<td>Turtle out of range</td>
<td>Triggered in the check if possible sub function. (Section 4.2.3)</td>
</tr>
<tr>
<td>3</td>
<td>Push angle unacceptable</td>
<td>Triggered in the check if possible sub function. (Section 4.2.3)</td>
</tr>
<tr>
<td>4</td>
<td>Ball behind robot</td>
<td>Triggered when the ball has a negative x-coordinate seen in the local virtual coordinate system based on the computations in Section 3.2.4.</td>
</tr>
<tr>
<td>5</td>
<td>Time out</td>
<td>Triggered when none of the other bailout conditions are triggered before the time out timer with parameter \textit{push_timeout} runs out.</td>
</tr>
<tr>
<td>6</td>
<td>Ball moved away</td>
<td>Triggered when a push action has occurred. Based on a change in the direction of the ball detected in the local virtual coordinate system described in Section 3.2.4.</td>
</tr>
</tbody>
</table>

Table 4.1: Bailout definitions.
Chapter 5

Conclusions and Recommendations

5.1 Conclusions

5.1.1 Increased passing speed
The developed push function gives the Tech United Turtle soccer robots an alternative ability for passing a ball to an desired location besides using the ball handling mechanism. Using the ball handling, the Turtle always has to face the incoming ball with its ball handling mechanism. After an incoming ball is grabbed the Turtle has to rotate to the direction of the desired target prior to shooting the ball. By bouncing an incoming ball with the housing of the Turtle the time to handle the ball practically decreases to zero.

5.1.2 Application scope push function
The increased passing speed is useful during diverse RoboCup soccer match situations such as corners and free kicks. Using push function the Turtles have the possibility to score directly out a corner ball. Besides refbox situations the push function can be used for a quick pass to an other Turtle or a pass to an empty space on the field. A ball passed to an empty space on the field enlarges the space between all the robots on the field what results in an increase of the strategic opportunities. For an impression of the push function applications, a corner demonstration has been made which can be reviewed on YouTube[5].

5.1.3 Limited control on the velocity of the ball
The experiments conducted before the development of the push function (see chapter 2) have shown that the velocity of the ball diversified and greatly decreases after it made contact with the housing of the Turtle. Due this velocity decrease it is uncertain if the ball reaches its desired target location. The limited control of the decrease in velocity is the main disadvantage of the developed push function. For further development of the push function, there are two recommendations suggested in Sections 5.2.1 and 5.2.2 to compensate this disadvantage.
5.2 Recommendations

5.2.1 Dynamic movement Turtle

The developed push function is based on ball-Turtle collision with a stationary Turtle standing in the trajectory of the ball. As mentioned in Conclusion 5.1.3 the velocity of the ball is uncontrollable during such collisions. To have more influence on the velocity of the ball, the Turtle should have a certain velocity while getting in contact with the ball. The velocity of the Turtle then affects the velocity of the ball. Depending on the movement directions of the Turtle this leads to an increase or decrease of the velocity of the ball. However, a dynamic Turtle pushing an incoming ball affects the “ideal conditions” described in the law of reflection Section 2.1.1. These affects result in a difference between the angle of incidence and the angle of reflection. Therefore new computations to calculate these angles need to be derived.

5.2.2 Adding “flippers” to the sides

The limited control on the velocity of the ball described in Conclusion 5.1.3 can be compensated by adding flippers to the pushing sides of the Turtle. By ejecting these flippers during the contact moment of the ball with the housing of the Turtle, it gives the ball additional velocity similar as seen in pinball machines. The behaviour of the ball on these accelerations are unknown and need to be examined before combining the flippers with the push function.

5.2.3 Learning algorithms

The push function is available for all the Tech United Turtle soccer robots except the goalkeeper. Each Turtle is equipped with its own unique housing, which may lead to a variance in the behaviour of the ball when pushed with different housings. To compensate the individual variance of each Turtle an additional learning algorithm can be applied on the push function. These learning algorithms evaluate each ball-Turtle collision and apply corrections on the final position and rotation of the individual Turtle, which enhance the precision of the push action.

5.2.4 Redesign housing of the Turtle

To optimize the result of the push function, the push surface on the Turtle may benefit of a redesign. As mentioned under the “ideal conditions” stated in the law of reflection (Section 2.1.1), the surface hit by the ball has to be truly flat and if it is compressible the should be uniformly in the direction of the normal line. Due to the foam layer attached on the housing of the Turtle these conditions are not applicable. In a redesign the push surfaces on the housing of the Turtle can be constructed with a material that is compatible with these “ideal conditions”.
Bibliography

[1] RoboCup website
   http://www.robocup.org

   http://www.techunited.nl

[3] Sir Thomas Little Heath
   A history of Greek mathematics. Volume II: From Aristarchus to Diophantus 1981

   Clegg, Dai; Barker, Richard 2004

[5] YouTube link demonstration movie push function
   http://www.youtube.com/watch?v=0IFgoTJkXhI
Appendix A

Experiment results

Figure A.1: Shots fired with a 0 degree angle of incidence at the push surface of the Turtle.

Figure A.2: Shots fired with a 15 degree angle of incidence at the push surface of the Turtle.
Figure A.3: Shots fired with a 30 degree angle of incidence at the push surface of the Turtle.

Figure A.4: Shots fired with a 45 degree angle of incidence at the push surface of the Turtle.

Figure A.5: Shots fired with a 60 degree angle of incidence at the push surface of the Turtle.
Appendix B

C code

```c
static int Push_AH(double* Push_Target, InputStruct* pS_in, OutputStruct* pS_out, SimStruct* S)
{
    //push function

    /*debugging*/
    int printit=1;
    /*variables*/

    enum{eInit=0,
        ePush,
        eBailout,
        eFinalize};

double pi=M_PI;
double vturtlemax=0.6*pS_in->va_max[0]; //approx. max speed turtle
double pos_correction[2]={0.0, -0.1957}; //correction push side from turtle center
double turtle[3]={pS_in->cur_xyo[0], pS_in->cur_xyo[1], pS_in->cur_xyo[2]};
double ball[4]={pS_in->ball_xyz_xyzdot[0], pS_in->ball_xyz_xyzdot[1],
pS_in->ball_xyz_xyzdot[2], pS_in->ball_xyz_xyzdot[3]}; // x y vx vy

double Gdxdy[2],Lball[4],phipoint[2],Turtle_Target[3],LPOI[3];
double totarget[2],toball[2],aimpoint[2],sidetarget[3],TT[2],Ltarget[2];
double Ldxdy[2]={-0.1957,0.0};
double zeros[3]={0.0,0.0,0.0};
double turtle01_phi[3]={turtle[0],turtle[1],0.0};
double zeros_phi[3]={0.0,0.0,0.0};
double vball,push_angle,Gangle_ball,Gangle_target,aimx,aimy,aimangle;
double alpha,Bdx_average,Bdy_average;
int    bailoutcode,check,l_or_R;

    /*DEMO*/
    DEM0wait= 5.0;

    /*global data*/
    psfun_global_data psfgd;
    get_pointers_to_global_data(&psfgd, S);
```

24
/ball velocity

/* push parameters*/
double maxpushangle = 90; // adjust maximum angle between ball and target [degree]
double minBallspeed = 1.0; // adjust minimum ball speed to push

/* bail out parameters*/
double push_timeout = 5.0; // bail out when push takes longer than this time
double moveawaydist = 1.0; // bail out when ball moves away from turtle
double towardsballvel = 2.0; // new phi when ball is going towards turtle

/* if the function was not finished properly previous time it was called, re_initialize */
if(psfgd->Previous_Action!=Action_Push){psfgd->pass_stage = eInit;}

/* default: stay where you are */
Turtle_Target[0] = pS_in->cur_xyo[0];
Turtle_Target[1] = pS_in->cur_xyo[1];
Clip_to_fieldborder(Turtle_Target);
Goto_target_AH(Turtle_Target, pS_in, pS_out);

/* global variables*/
L_or_R = psfgd->L_or_R;
alpha = psfgd->alpha;
bailoutcode = psfgd->bailoutcode;
Bdx_average = psfgd->average_ball_dx;
Bdy_average = psfgd->average_ball_dy;

/* cases*/
switch (psfgd->pass_stage){

/* case init*/
case eInit:
    if(printit){printf("init push\n");}
    psfgd->pass_phi = getangle(ball,turtle);
    psfgd->pass_start_dist = getdistance(ball,turtle);
    psfgd->pass_starttime = ssGetT(S);
    psfgd->pass_min_dist_to_ball = 100.0;
    psfgd->pass_stage = ePush;if(printit){printf("goto push action...\n");}
    psfgd->bailoutcode = 0;
    psfgd->average_ball_dx = ball[2];
    psfgd->average_ball_dy = ball[3];
    psfgd->check=1;
    psfgd->L_or_R=1;
    psfgd->alpha=0.0;
    psfgd->DEMOwaittime=5.0;
    return 0;
    break;

-2-
/*case push*/
case ePush:

    /* after 500ms check if there can be a succesfull push action using average
    speed during those 500ms to get an accurate result average ball velocity
    is calculated with (9 times old value + 1 time new value) devided by 10 */

    psfgd->average_ball_dx=(9*Bdx_average+1*ball[2])/10;
    psfgd->average_ball_dy=(9*Bdy_average+1*ball[3])/10;

    if (ssGetT(S)-psfgd->pass_starttime > 0.500 && psfgd->check){
        bailoutcode=Checkifpossible(ball[0], ball[1], Bdx_average, Bdy_average,
                                   minBallspeed, 
                                   maxpushangle, turtle, vturtlemax, Push_Target);
        psfgd->check=0;
        if(printit){printf("Check if possible\n");}
    }

    if (bailoutcode > 0){psfgd->bailoutcode=bailoutcode;psfgd->pass_stage=eBailout;}

    /*compute local ball*/
turtle01_phi[2]=psfgd->pass_phi;
zeros_phi[2]=psfgd->pass_phi;
global2local(ball, 1, turtle01_phi, Lball);//ball
global2local(ball+2, 1, zeros_phi, Lball+2);//vball

    /*update ball approach angle if vball towards turtle*/
if(Lball[3]<-towardsballvel) { //vel lower limit
    phipoint[0]=ball[0]+ball[2];
    psfgd->pass_phi=getangle(ball,phipoint);}

    /*target location to local*/
global2local(Push_Target, 1, turtle01_phi, Ltarget);

    /*intercept point*/
    /*x,y*/
    LPOI[0]=Lball[0];
    LPOI[1]=0.0;
    /*intercept angle*/
    //vector to target
totarget[0]=(Push_Target[0]-turtle[0])/getdistance(Push_Target,turtle);
totarget[1]=(Push_Target[1]-turtle[1])/getdistance(Push_Target,turtle);
    //vector to ball
toball[0]=(ball[0]-turtle[0])/getdistance(ball,turtle);
toball[1]=(ball[1]-turtle[1])/getdistance(ball,turtle);
    aimpoint[0]=totarget[0]+toball[0];
    LPOI[2]=getangle(aimpoint,zeros);
// target for the side of the turtle
sidetarget[2]=LPOI[2];
local2global(LPOI,1,turtle01_phi,sidetarget);

/* now compute turtle target, which has offset in angle and position */
if(LTarget[0]>0){Ldxdy[0]=-0.1957;}
else{Ldxdy[0]=0.1957;}
local2global(Ldxdy,1,zeros_phi,Gdxdy);

/* determine optimal push side */
Gangle_ball=getangle(ball,zeros);
Gangle_target=getangle(Push_Target,zeros);
aimx=cos(Gangle_ball)+cos(Gangle_target);
aimy=sin(Gangle_ball)+sin(Gangle_target);
aimangle=atan2(aimy,aimx);
if (psfgd->L_or_R){psfgd->alpha=determine_left_or_right(turtle,aimangle);
psfgd->L_or_R=0;};

/* set actual target */
Turtle_Target[0]=sidetarget[0]+Gdxdy[0];
Clip_to_fieldborder(Turtle_Target);
Goto_target_AH(Turtle_Target, pS_in, pS_out);

/* bailout conditions stop intercept if one of these is true */
/* ball is behind robot in local coords */
if(Lball[1]<0){
    psfgd->pass_stage=eBailout;
    psfgd->bailoutcode=4;}

/* timeout */
if((ssGetT(S) - psfgd->pass_starttime)>push_timeout){
    psfgd->pass_stage=eBailout;
    psfgd->bailoutcode=5;}

/* ball has moved away from me more than moveawaydist meters */
if(getdistance(ball,turtle)>psfgd->pass_min_dist_to_ball+moveawaydist _
    && psfgd->pass_min_dist_to_ball<0.5){
    psfgd->pass_stage=eBailout;
    psfgd->bailoutcode=6;}
else{
    if(getdistance(ball,turtle)<psfgd->pass_min_dist_to_ball){
        psfgd->pass_min_dist_to_ball=getdistance(ball,turtle);
    }
}
return 0;
break;
/*case bail out*/

    case eBailout:
        printf("Bail out: ");

            if(printit==1 && psfgd->bailoutcode==1){printf("ball velocity to low.\n");}
            if(printit==1 && psfgd->bailoutcode==2){printf("turtle out of range.\n");}
            if(printit==1 && psfgd->bailoutcode==3){printf("push angle unacceptable.\n");}
            if(printit==1 && psfgd->bailoutcode==4){printf("ball behind robot.\n");}
            if(printit==1 && psfgd->bailoutcode==5){printf("timeout.\n");}
            if(printit==1 && psfgd->bailoutcode==6){printf("ball moved away.\n");}

            psfgd->DEMOwaittime=ssGetT(S);
            psfgd->pass_stage=eFinalize;
            //if(printit){printf("finalize push\n");}
            if(printit){printf("finalizing push please wait 5 sec\n");} //DEMO only
            return 0;
            break;
}

    //Case finalize
    case eFinalize:/*finalize and reinit*/

        /* default: stay where you are */
        Turtle_Target[0] = pS_in->cur_xyo[0];
        Turtle_Target[1] = pS_in->cur_xyo[1];
        Clip_to_fieldborder(Turtle_Target);
        Goto_target_AH(Turtle_Target, pS_in, pS_out);
        if ((ssGetT(S) - psfgd->DEMOwaittime)>DEM0wait)
        {return 1;printf("ready for new push\n"); psfgd->pass_stage=eInit;}
        return 0;
        break;

        default:
        psfgd->pass_stage=eInit;
        return 1;
        break;

    }//switch
    if(printit){printf("\n");}
}//function
/*OTHER FUNCTIONS Push_AH*/
SUB FUNCTION
/*************************** CHECK IF POSSIBLE ***************************/
int Checkifpossible(double Bx, double By, double Bdx_average, double Bdy_average, _
    double minBallspeed, double maxpushangle, double* turtle, _
    double vturtlemax, double* Push_Target){
    //variables
    double pi=M_PI;
    int result_Checkifpossible=0;
    double a1,b1,a2,b2,poi[2],vectarget[2],vecball[2],dist_ball,dist_turtle;
    double ball_time,turtle_time,vball,push_angle,dotproduct;
    double ball[4]={Bx, By, Bdx_average, Bdy_average};
    vball=sqrt(Bdx_average*Bdx_average+Bdy_average*Bdy_average);
    /* if ball is to slow bail out*/
    if (vball<minBallspeed){result_Checkifpossible=1;}
    // ball trajectory y=x*a1+b1
    a1=ball[3]/ball[2];
    b1=ball[1]-a1*ball[0];
    // turtle trajectory y=x*a2+b2, perpendicular slope =-1
    a2=-1/a1;
    b2=turtle[1]-a2*turtle[0];
    //for determing the poi we assume constant ball velocity and constant turtle velocity v
    //point of intercept (poi)
    poi[0]=(b2-b1)/(a1-a2);
    poi[1]=a1*poi[0]+b1;
    //distances to poi
    dist_ball=getdistance(poi,ball);
    dist_turtle=getdistance(poi,turtle);
    //time to get to poi
    ball_time=dist_ball/vball;
    turtle_time=dist_turtle/vturtlemax;
    if (turtle_time > ball_time){result_Checkifpossible=2;}
    /* Determin angle between ball and target */
    //normalized vector to ball
    vecball[0]=(Bx.poi[0])/getdistance(ball,poi);
    vecball[1]=(By.poi[1])/getdistance(ball,poi);
    //normalized vector to target
    vectarget[0]=(Push_Target[0]-poi[0])/getdistance(Push_Target,poi);
    vectarget[1]=(Push_Target[1]-poi[1])/getdistance(Push_Target,poi);
    //angle between vecball and vectarget
    dotproduct=vecball[0]*vectarget[0]+vecball[1]*vectarget[1];
    push_angle=acos(dotproduct);
    if (push_angle > maxpushangle*(2.0*pi)/360.0){result_Checkifpossible=3;}
    return result_Checkifpossible;
}
SUB FUNCTION
/******************** DETERMINE LEFT OR RIGHT ********************/
double determine_left_or_right(double* turtle, double aimangle){
    //variables
    double result;
    double pi=M_PI;
    //double corr_angle=1.0/6.0*pi;
    double corr_left=2.0/3.0*pi; //correction angle to target to push with the left side
    double corr_right=-2.0/3.0*pi; //correction angle to target to push with the right side
    double A,B,C,absB,absC;
    if(turtle[2]<0){
        A=wrappen(aimangle+pi);
        B=A-wrappen(turtle[2]+corr_right);
        C=A-(turtle[2]+corr_left);
        absB=sqrt(B*B);
        absC=sqrt(C*C);
        if (absB<absC){result=corr_right;}
        else {result=corr_left;}
    }
    else{
        A=wrappen(aimangle+pi);
        B=A-(turtle[2]+corr_right);
        C=A-wrappen(turtle[2]+corr_left);
        absB=sqrt(B*B);
        absC=sqrt(C*C);
        if (absB<absC){result=corr_right;}
        else {result=corr_left;}
    } 
    return result;   
}//best_push_side function