Traction Control for Omni-directional Robots

A.W. van Zundert, Student nr.: 0650545

CST 2011.071

CST report

Eindhoven University of Technology
Department of Mechanical Engineering
Control Systems Technology

Eindhoven, September, 2011

Abstract

In this report a traction controller is developed for an omni-directional robot, named Turtle. At first traction and slip are defined and the geometry of the Turtle is discussed. Then the explanation of the relevant software for the motion of the Turtles is spread over three chapters. The motion schemes of the Turtle in Simulink are simplified and discussed. The trajectory planner that creates the driving path for a Turtle is explained in detail and simulations are performed to show what kind of signals the trajectory planner creates and how these signals are alternated as inputs to the motors. After this some research has been done into control systems of other vehicles and some initial ideas are sketched for traction control in the Turtles. Finally the actual traction controller is developed and explained using flowcharts, the implementation (in C-code) of the traction controller and some simulations to give a quick look for how it works. Afterwards a general conclusion is drawn and some future work is suggested.

Contents

List of Figures

Listings

Introduction

A significant aspect in playing soccer is making sure your team captures the ball before the opponents do. This means that the players of your team should be as fast as possible. For this a Turtle is required to reach maximal acceleration values.

As in other vehicles, a major problem in achieving this is slip. Slip can be interpreted as a loss of traction or grip. While slipping a vehicle is not in control of its movement and it can cause a difference between the desired acceleration and the actual acceleration. It should be clear that slip is highly unwanted.

There are moments that a Turtle is pushing against an opponent. In such case the main purpose is to push as hard as a Turtle is capable of. The maximal pushing force can only be achieved when the wheels are almost slipping but not quite yet. Slip makes the pushing force drop so slip should be prevented.

Previously (before September 2011), the Turtles did not reckon with slip at all. Only a difference in position would be taken into account. However, to achieve a maximal acceleration without slip, traction control is desired while or before slipping.

Before a traction controller can be made it should be clear what traction and slip are. This is investigated in Chapter 1 "Traction and slip". Secondly, research has been done into the configuration and kinematics of a Turtle in Chapter 2 "The Turtles". After that the Motion scheme is simplified and examined. This examination is spread over three chapters, Chapter 3 "Motion Scheme", Chapter 4 "Trajectory Planner" and Chapter 5 "Global Signal Analysis". After this is all clear a short search is done to see what other vehicles use for traction control and some initial ideas for traction control systems are sketched (Chapter 6 "Possible control systems"). In Chapter 7 "Traction controller" the final traction controller is developed and explained. Finally a general conclusion is drawn and some future work is suggested.

Chapter 1

Traction and slip

Any movement of a wheel-based vehicle depends on the traction of the wheels. Traction can be defined as the force wheels exert on a riding surface. If this force is not equal to the motor forces, the vehicle will slip. Slip can be defined like in Equation 1.1, with λ the slip. For $\lambda \neq 0$ slip occurs. When the vehicle speed is larger than the wheel speed ($\lambda > 0$), the vehicle is slipping while decelerating. When the vehicle speed is lower than the wheel speed ($\lambda < 0$), the vehicle is slipping while accelerating.

$$\lambda = \frac{vehiclespeed - wheelspeed}{vehiclespeed} \tag{1.1}$$

The maximal traction of a wheel depends on the weight that the vehicle exerts on the wheel (F_{normal}) and on the coefficient of friction μ (see Equation 1.2). The coefficient of friction relates the amount of friction force (traction) between a wheel and the riding surface to the weight resting on the wheel. In Figure 1.1 a free body diagram of one wheel is given.

$$F_{motor} + F_{forward} + F_{friction} = \frac{T}{R_w} - m \cdot a + \mu \cdot m \cdot g = 0 \tag{1.2}$$

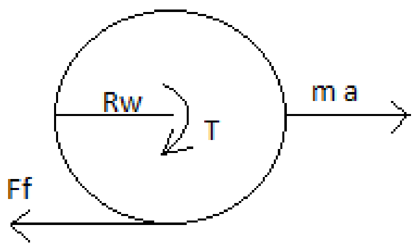


Figure 1.1: Free body diagram of a wheel

Figure 1.2[1] illustrates the relation between wheel slip and the coefficient of friction. It is desired to stay in the stable area of Figure 1.2, so have a slip lower than 20 percent. Material properties, weather circumstances and other external influences are an important factor for the coefficient of friction. In general the coefficient of friction is not constant, so neither for the Turtles.

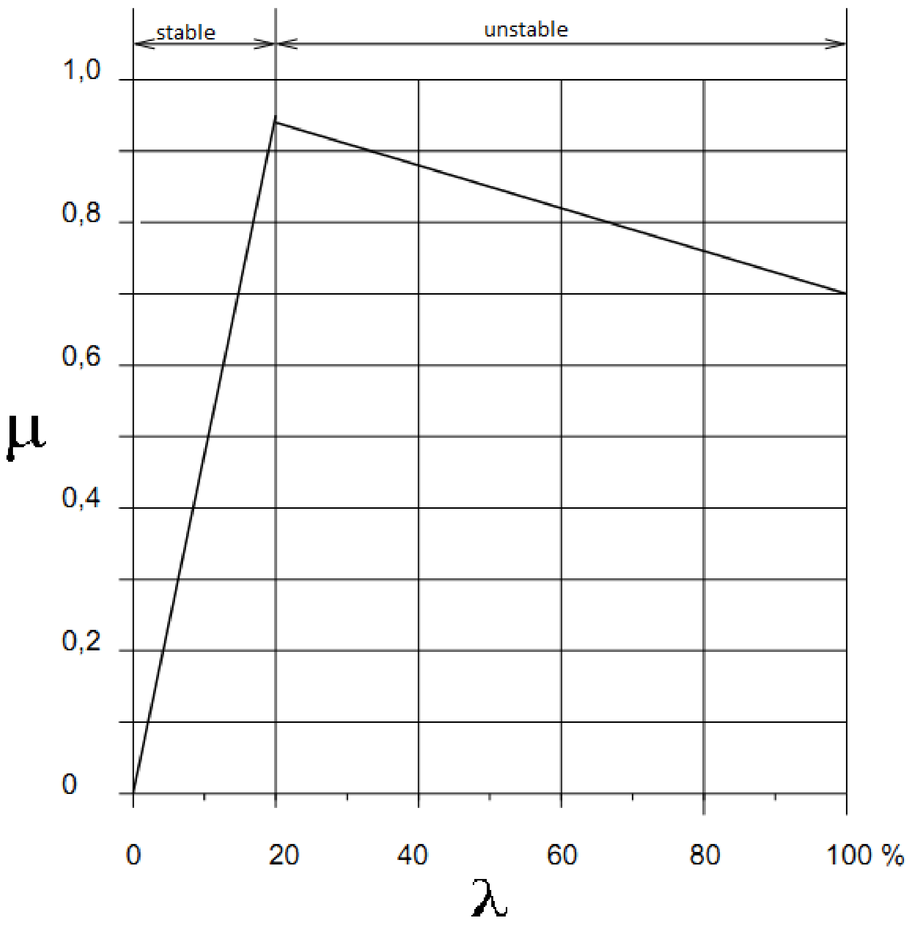


Figure 1.2: Slip (λ) versus coefficient of friction (μ)

Chapter 2

The Turtles

The Turtles are holonomic robots, what means that they can move in every direction (360 degrees). They are capable of this because of their wheelbases. A Turtle consists of three omniwheels, positioned at an angle of 120 degrees (Figure 2.2). Why Turtles have three wheels is explained in section 2.1 "Number of Wheels".

Omniwheels are wheels with other small wheels on the perimeter (Figure 2.1). The small secondary wheels have a degree of freedom perpendicular to the roll direction of the primary wheel. Compared to the primary wheels they have a relatively small coefficient of friction in their free rolling direction.

In section 2.2 "Decoupling Robot- to Wheel Coordinates" a decoupling is given between the local robot coordinate system and the coordinate system of the wheels.

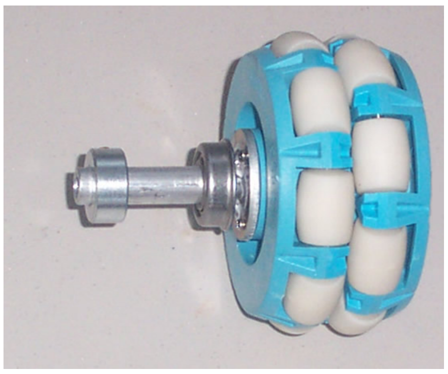


Figure 2.1: Omniwheel; wheel with small wheels on the perimeter

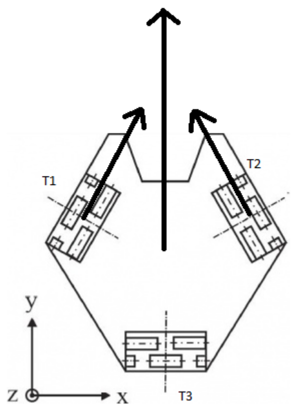


Figure 2.2: Turtles' wheel configuration and movement in y-direction

2.1 Number of Wheels

The advantage of three wheels with respect to four wheels lies in the fact that four wheels make the robot overdetermined. With four wheels always one wheel is from the ground. The mass of the Turtle will not be spread evenly over the three wheels which can cause loss of traction. Prestressing springs are not a solution for overdetermination because this does not bring the center of mass back.

However, an object on four wheels moves more effectively than three wheels in executed configuration (See Figure 2.2). With three wheels in this configuration the direction of movement is never equal to the rotating direction of the wheels. This means that wheels always slip. The wheels have a combined friction coefficient of the primary and secondary wheels dependent on the rolling angle of the wheel.

Any movement of the Turtle is achieved by combining certain motor speeds. These motor speeds depend on the angle of each wheel relative to the movement direction. In Figure 2.2 an example is illustrated for a movement in y-direction. The first wheel has traction force T_1, the second wheel T_2 and the third wheel T_3. If both traction forces, T_1 and T_2, are equal and T_3 is equal to zero the resulting net force is purely in y-direction. Because of the property of omniwheels, wheel 3 has negligible friction. If both traction forces are NOT the same the Turtle will also have a movement in x-direction.

2.2 Decoupling Robot- to Wheel Coordinates

In Figure 2.3 the wheelbases is given once again. However, here the radial distance of each wheel to the center of the Turtle and the positive wheel directions are given. The wheel speeds w_1,w_2 and w_3 are defined in Equation 2.1 and the local Turtle speed V_x, V_y and T_ϕ are derived to Equation 2.2. The intermediate steps of this derivation can be found in Appendix ??.

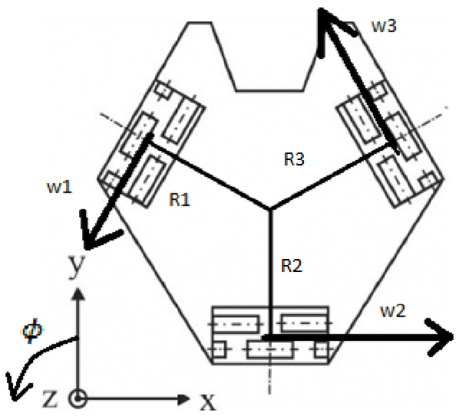


Figure 2.3: Wheelbases with positive speed directions

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -|V_r| \cdot \sin(30+\phi) + R_1 \cdot \dot{\phi} \\ |V_r| \cdot \cos(\phi) + R_2 \cdot \dot{\phi} \\ |V_r| \cdot \sin(-30+\phi) + R_3 \cdot \dot{\phi} \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} V_x \\ V_y \\ \dot{\phi} \end{bmatrix} = \frac{1}{R_1 + R_2 + R_3} \cdot \underline{A} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} \tag{2.2}$$

$$\underline{A} = \begin{bmatrix} -R_2 & R_3 + R_2 & -R_2 \\ -\frac{1}{\sqrt{3}}(2R_3 + R_2) & \frac{1}{\sqrt{3}}(-R_3 + R_1) & \frac{1}{\sqrt{3}}(R_2 + 2R_1) \\ 1 & 1 & 1 \end{bmatrix} \tag{2.3}$$

Chapter 3

Motion Scheme

The motion scheme of the Turtles is very comprehensive and difficult to understand. Because of that a simplified scheme (see Figure 3.1) will be discussed. The global and local coordinate systems are shown in Figure 3.2.

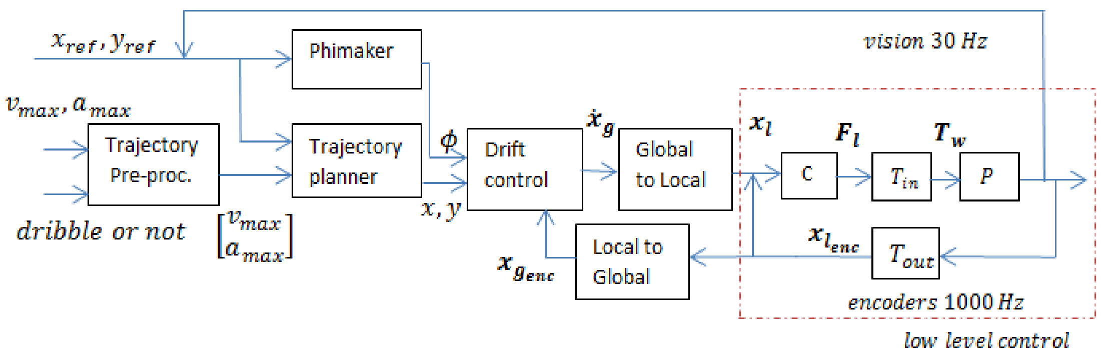


Figure 3.1: Simplified motion scheme of Turtles

- The "Trajectory Preprocessor" determines the values of the maximal acceleration a_{max} or the maximal velocity v_{max}. For dribbling other maximal values are used than for driving.
- The "Trajectory planner" creates a smooth path the Turtle will follow. This will be explained in Chapter 4 "Trajectory Planner". In this block also a parking brake is added. This parking brake takes the overshootflag explained in section 4.2.1 "Overshootflag".
- The "phimaker" creates the path for the ϕ-coordinate and works on the same principle as the trajectory planner.
- The "Drift control" controls the global position of the Turtle on the field using position from encodercounts.
- "C" is the controller that controls the local position of the Turtle using position from the encoders.

- "Global to Local" and "Local to Global" are each others' reciprocal. These blocks contain the decoupling of the coordinate systems shown in Figure 3.2.
- "T_{in}" and "T_{out}" are also each others' reciprocal. These blocks contain the decoupling derived in section 2.2 "Decoupling Robot- to Wheel Coordinates".
- P is the Turtle and has voltages as input since the Turtle have 150 Watt Maxon electric motors. These voltages represent the torque that should be introduced to the wheels. For the outputs, the Turtle has encoders on its wheels that have a sample frequency of 1000 Hertz and can vision be used that has a sample frequency of 30 Hertz.

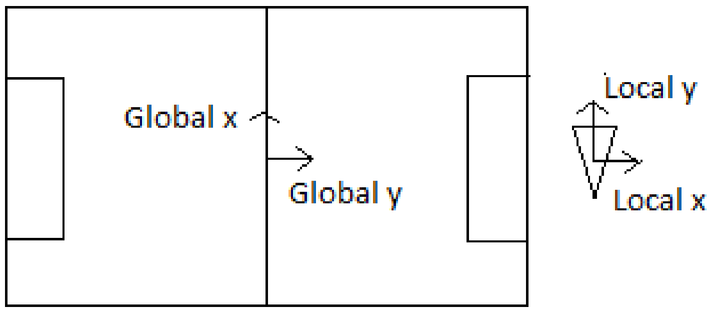


Figure 3.2: Global and local coordinate systems

Because of the operations in "Drift Control" and in the "controllers" the actual input-acceleration is different from the calculated accelerations in the trajectory planner. This is shown in subsection 5 "Global Signal Analysis".

Recently an accelerometer is attached on one Turtle. This accelerometer has another coordinate system than the Turtle does. For example the x axis of the accelerometer is equal to the z axis of the Turtle and the y and z axis of the accelerometer are rotated with respect to the Turtle. Also the accelerometer does not find itself in the middle of the Turtle. For these reasons the values obtained from this accelerometer are nonsense. To correct this error a direction- cosine matrix[2] should be used to correct the rotation of the coordinate system. Because the ϕ of a Turtle is also an important parameter, this should also be obtained from the accelerometer. For this reason the translation of the coordinatesystem of the accelerometer should be corrected too. For example if the Turtle is rotating only in phi direction the accelerometer would constantly detect a movement in x direction. This is not a desired behavior.

Chapter 4

Trajectory Planner

The Turtles receive a certain point on the field where they need to be as fast as possible. The trajectory planner calculates a smooth trajectory by evaluating the acceleration values. The trajectory planner is programmed in C and consists of three relevant subroutines (4.1 "Outputs", 4.3 "Check Case" and 4.2 "Calculate Time") which will be explained in following sections. Usually "Outputs" calls "Calculate Time" and "Check Case" always get called by "Calculate time". "Calculate Time" also uses an overshootflag, this will be explained in subsection 4.2.1. "Overshootflag". Finally in section 4.4 "Simulations trajectory planner" some simulations are performed to give an example of how the trajectory planner works. The complete trajectory planner is added to Appendix C.

4.1 Outputs

The trajectory planner has many inputs:

- Referential global x and y position the Turtle wants to reach.
- Referential x and y velocity the Turtle wants to have at the referential global x- and y coordinates.
- Instantaneous x- and y position of the Turtle.
- Instantaneous x- and y velocity of the Turtle.
- Maximal velocity and maximal acceleration
- RobotID, the trajectory planning of the keeper is different from the trajectory planning of a Turtle on the field.

The outputs of the trajectory planner are:

- Acceleration values in x- and y direction.
- Overshootflag which will be further explained in section 4.2.1 "Overshootflag".

The trajectory planner is an S-function implemented in C. Basically the trajectory planner makes sure the Turtle drives via the shortest path.

At first the trajectory planner checks if the Turtle is converged. This means it checks if the Turtle is arrived at the desired x- or y position and if it has the desired final velocities. In this step there are four possibilities:

1. Both x- and y position respectively velocities are converged. Because a threshold is used, it can be that the desired velocity is not entirely achieved. So the output accelerations are only dependent on the difference between the desired velocity and the instantaneous velocity.

2. Only x position respectively velocity are converged. In this occasion for the acceleration in x direction the same operation will be done as in point one. For the y position respectively velocity other calculations will be done which will be further explained in section 4.2 "Calculate Time".

3. Only y position respectively velocity are converged. In this occasion the same operations are done as in step 2, but then the opposite operation belongs to x and y.

4. The x nor the y position respectively velocity is converged. In this case it is necessary that a straight path will be planned. Otherwise the Turtle would for example first drive in x direction and then in y direction. The ideal angle will be calculated, so that desired x- and y position respectively velocity will be met on the same time.

4.2 Calculate Time

Subroutine Calculate Time calculates the time it will take to reach a converged state. To do that it contains a loop that runs several times, until it knows the estimated time of arrival(ETA) of the Turtle. ETA is calculated by predicting possible future operations and is only an estimation.

Each run subroutine Check Case (See section 4.3) is called. This returns if the estimation is converged or not. If the estimation is converged, Calculate Time can be aborted and returns the estimation (ETA).

The other loops are for possible future operations, so the in the first loop calculated acceleration from Check Case will be returned.

In Calculate Time also the overshootflag is set but this will be further explained in subsection 4.2.1 "Overshootflag".

4.2.1 Overshootflag

In some cases Calculate Time sets an overshootflag. The trajectory planner outputs this overshootflag for x- and for y direction but does not perform further actions with it. From the trajectory planner it goes straight on to a so called parking brake. This overshootflag can have four values:

- 0: default value, nothing happens.

- 1: only in the first loop of Calculate Time this value can be introduced. Also can this value occur only if the calculated acceleration is negative, but the distance the Turtle need to pass is still positive. This can be the case if the Turtle is decelerating to stop in time, but also when the Turtle is driving too fast.

- 2: just like for value 1, this value can only be introduced in the first loop of Calculate Time. This value has almost the same condition as for value 1. The difference is that for this value the robot was driving in negative direction. So this value can only occur if the calculated acceleration is positive, but the distance the Turtle need to pass is still negative.
- 3: this value can only occur in the second loop and if in the first loop the overshootflag is set to 1 or 2. For value 1 it was unknown if the Turtle was decelerating to stop in time or driving too fast. If in this second loop the estimation is still not converged the overshootflag gets set to 3. If the Turtle was decelerating to stop in time the estimations would be converged and the overshootflag stays 1 or 2.

The parking brake takes the overshootflag into account. If the overshootflag has value 3 the parking brake amplifies the acceleration values with 3. If the overshootflag is 1 or 2 the parking brake passes the acceleration values through without changing.

4.3 Check Case

Check Case calculates which acceleration values are desired. Calculate Time also calls Check Case for possible future operations which does not relate to instantaneous action. For that reason in the following list of important cases there will be spoken of incoming values and not of the Turtle's instantaneous behavior.

- Case -1: default value.
- Case 1: driving in wrong direction, so the Turtle should accelerate in the other direction.
- Case 2: wanting to accelerate to a certain velocity. Mostly this will be maximal velocity (called Case 21) but when the distance is not big enough this could be another velocity (called Case 22). In both cases maximal acceleration is desired.
- Case 3: driving at maximal velocity. Because of the threshold it could be that the actual velocity is not equal to the maximal velocity. In this case the acceleration will be evaluated to make sure the maximal velocity will be reached entirely.
- Case 4: needs to decelerate to stop in time. In this case the acceleration value should be such that maximal braking is attained.
- Case 5: driving at a velocity that is higher than the maximal velocity. In this case maximal deceleration is desired.
- Case 6: distance is too small to reach desired final velocity. In this case maximal acceleration or deceleration is wanted, dependent on the desired velocity.

When the final position and velocity are converged the acceleration value will be dependent on the difference between the instantaneous and the desired position and velocity.

4.4 Simulations trajectory planner

The simulations are run in Simulink. This is to show what the trajectory planner does to the position, velocity and acceleration of the Turtle. In these simulations no external effects affect the results.

At first in subsection 4.4.1 a simple movement in purely x direction is simulated. After that a movement in x- and y direction is simulated in subsection 4.4.2. Finally in subsection 4.4.3 it is attempted to simulate a small piece of the real world. After all in subsection 4.4.4 a summary will be given over the three simulations. In appendix ?? the setup for the simulations is given and further explained.

4.4.1 Movement in x direction

In this simulation just one point in x direction is given to the trajectory planner. The final velocities and y-position are set to zero. The case determined in subroutine Check Case is logged, so as the overshootflag.

What should be expected for position, velocity and acceleration is shown in Figure 4.1.

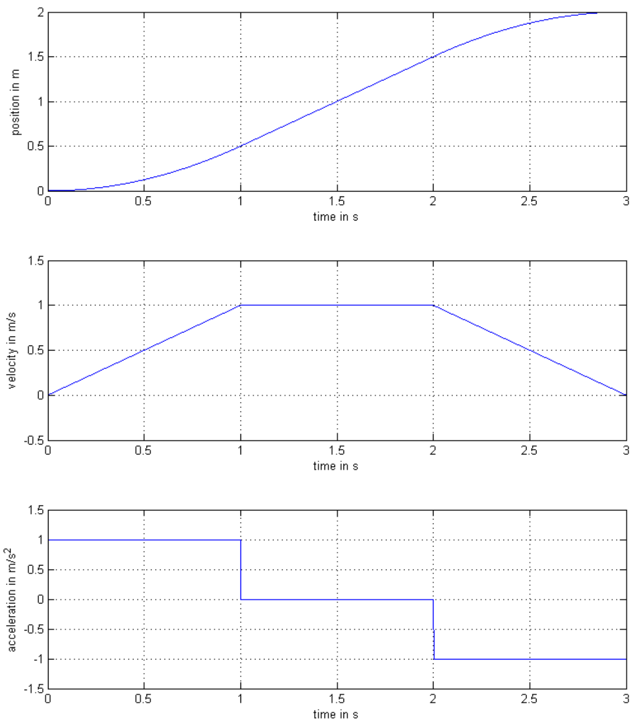


Figure 4.1: Theoretical position, velocity and acceleration

Firstly the Turtle should want to accelerate at maximal acceleration until maximal velocity is achieved (Case 21). After the Turtle drives at maximal velocity for a while (Case 3) the Turtle would see it must decelerate to stop in time (Case 4). When the Turtle is in Case 4 the overshootflag should become 1, because the acceleration is negative and the Turtle should still pass a distance.

Results

The results for position, velocity and the case are as expected, but the results for the acceleration have a small abnormality like shown in Figure 4.2.

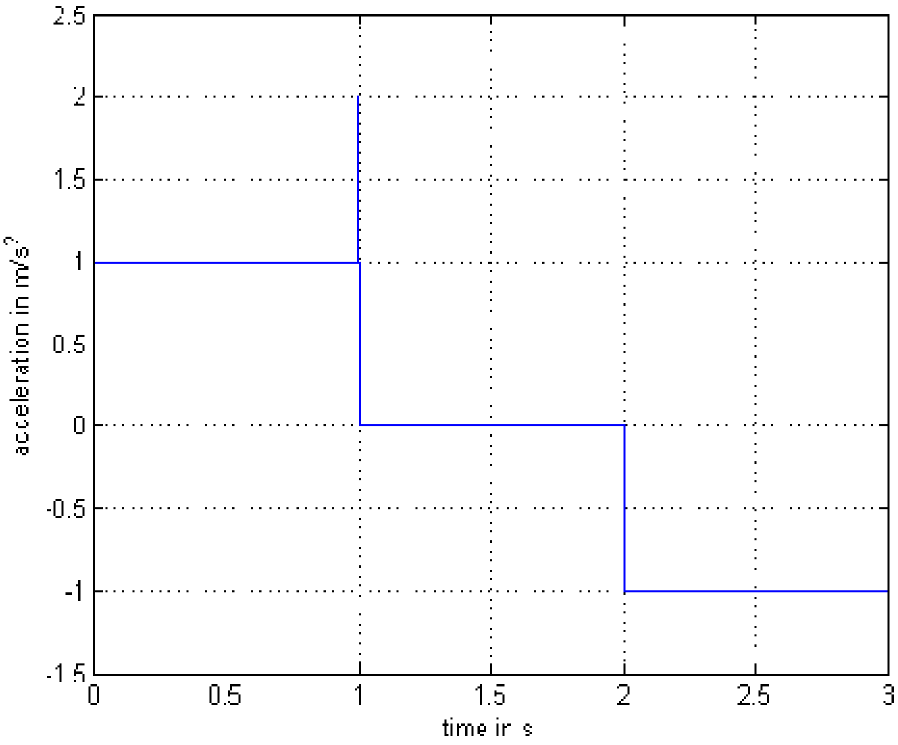


Figure 4.2: Results for acceleration simulation 1

Since the maximal acceleration is set to 1 the value of 2 on 1 second is not allowed. If one looks at the cases in Figure 4.3 this value can be explained.

On 1 second the case becomes three. Equation 4.1 shows the calculation for the acceleration in case 3.

$$a = \frac{v_{max} - v_{instantaneous}}{SAMPLE\_TIME} \tag{4.1}$$

When the numerator in this equation is lower than the denominator this value becomes higher than 1. Because the a_{max} in this case is 1, the calculated acceleration is higher than a_{max}. Since it does not get checked if the calculated acceleration is higher than a_{max} the peek in Figure 4.2 is possible. This is easily solved by adding following two lines.

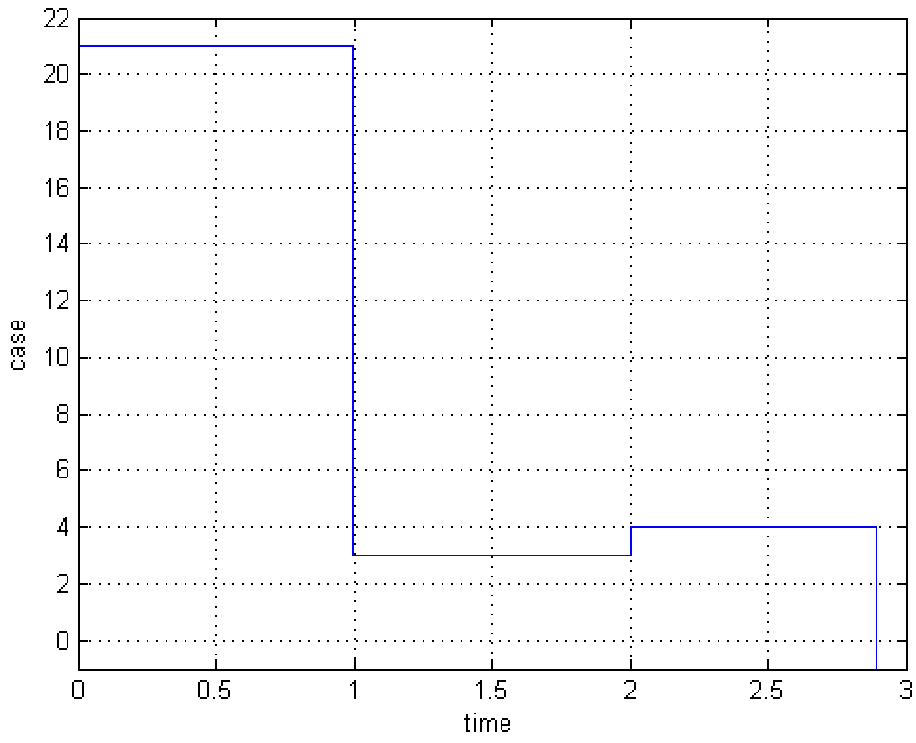


Figure 4.3: Results for cases simulation 1

Listing 4.1: Two lines to limit accelerations

```
y[0]=dmin(dmax(y[0],-a_max),a_max);
y[1]=dmin(dmax(y[1],-a_max),a_max);
```

Y is the output acceleration vector and dmin respectively dmax takes the minimal respectively maximal value. If these two lines are included in the trajectory planner right at the end the acceleration never exceeds the a_{max}.

4.4.2 Movement in x direction and y direction

For this simulation the same set up is used, but the value for the desired y direction is set to 1 meter. Now the distance in x- and y direction is different and both are not converged. The trajectory planner will try to calculate the ideal angle to drive what means that the acceleration in x- and y direction will not be the same.

The Case for the x- and y direction and the overshootflag should be the same since the Estimated Time of Arrival (ETA) is desired to be the same. The case will be a bit different from the previous simulation because different accelerations and velocities will be calculated. At first while accelerating, the case will be 21 and after a while the case will be 3. This might look strange because a_{max} is not reached. In calculating the ideal driving angle the v_{max} in x- and y direction are changed so the subroutine Check Case has different a_{max} and v_{max}. This is because now the Turtle probably wants a lower velocity to make sure x- and y direction will be converged at approximately the same time.

Results

In Figure 4.4 are the accelerations, velocities and movements in x- and y direction illustrated. As can be seen is the error, mentioned in previous subsection, for this simulation not entirely solved by adding the two mentioned lines. The calculated acceleration does not exceed a_{max} but there is still a peek. This is still a problem what should be investigated.

As expected the accelerations and velocities will not reach a_{max} and v_{max}. Also are the desired movements on approximately the same time converged. The overshootflag and cases are just like expected.

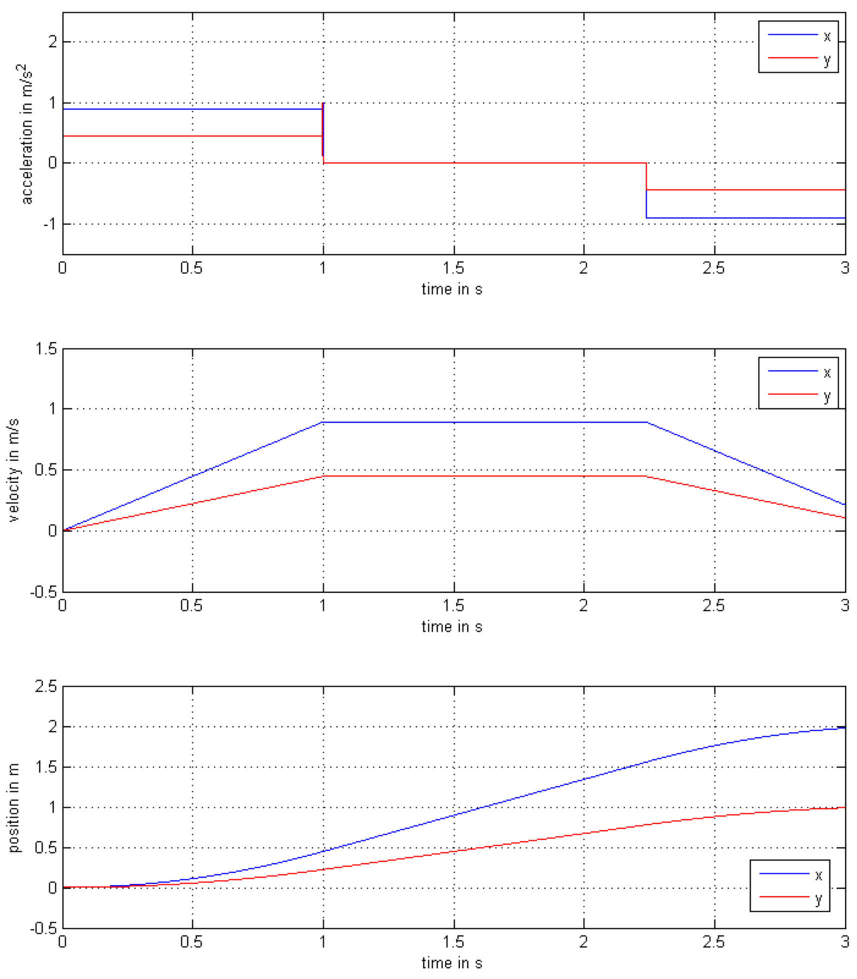


Figure 4.4: Results for accelerations, velocities and movements simulation 2

4.4.3 Attempt to simulate real world

In the real world a Turtle cannot drive in exactly a straight line. There will be deviations where its position differs from any position on the line. When this deviation is too big the Turtle will notice it. However the Turtle does not want to get back on the line, but calculates a new line with a movement in another direction. This influences its x- and y velocity respectively acceleration. This is shown in Figure 4.5. The red line was the first path in purely x direction and the blue line is the new path the Turtle will follow. On the spot of the red cross the Turtle notices that it has a deviation.

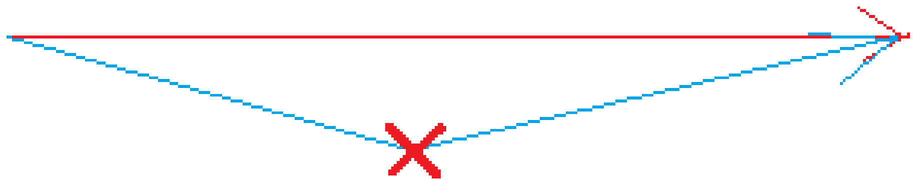


Figure 4.5: A path with a deviation.

In this subsection the just mentioned aspect is tried to simulate. This is realized by having a x movement just like in subsection 4.4.1. Then is with a signal builder shown in appendix ?? in Figure B.2 a certain y movement applied after a certain time.

With knowledge of previous simulations one should expect that when the y movement is introduced the velocity in x direction will decrease. This happens because the driving angle gets calculated so that the Estimated Time of Arrival (ETA) of the x direction is approximately the same as the ETA of the y direction.

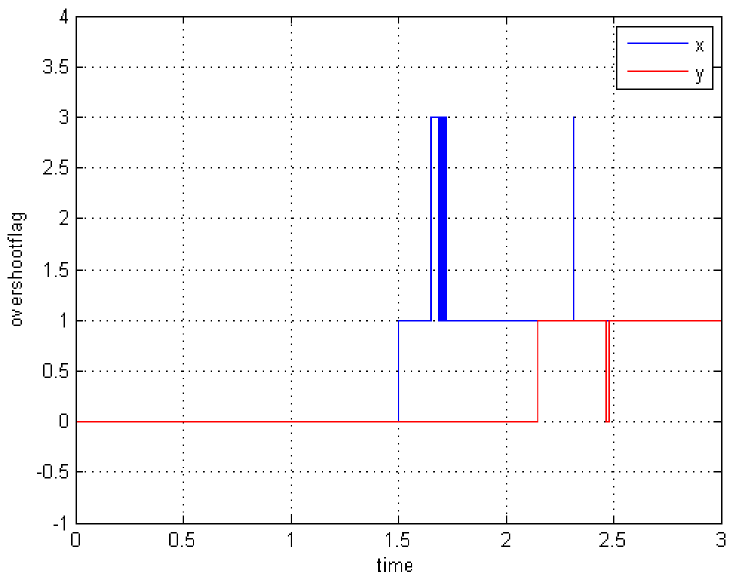


Figure 4.6: The overshootflags of simulation 3

In Figure 4.7 the accelerations, velocities and positions are given. Like expected the velocity has a dip on the moment the y movement is introduced. The accelerations show some strange behavior. The acceleration exceeds the a_{max}. The only situation when this is possible is when the overshootflag is set to 3 and the parking brake amplifies the accelerations with 3. Like shown in Figure 4.6 on the previous page the overshootflag is indeed set to 1 when the case was 4. When the y movement was introduced the overshootflag was set to 3. This makes sure the x- and y positions converge on the same time.

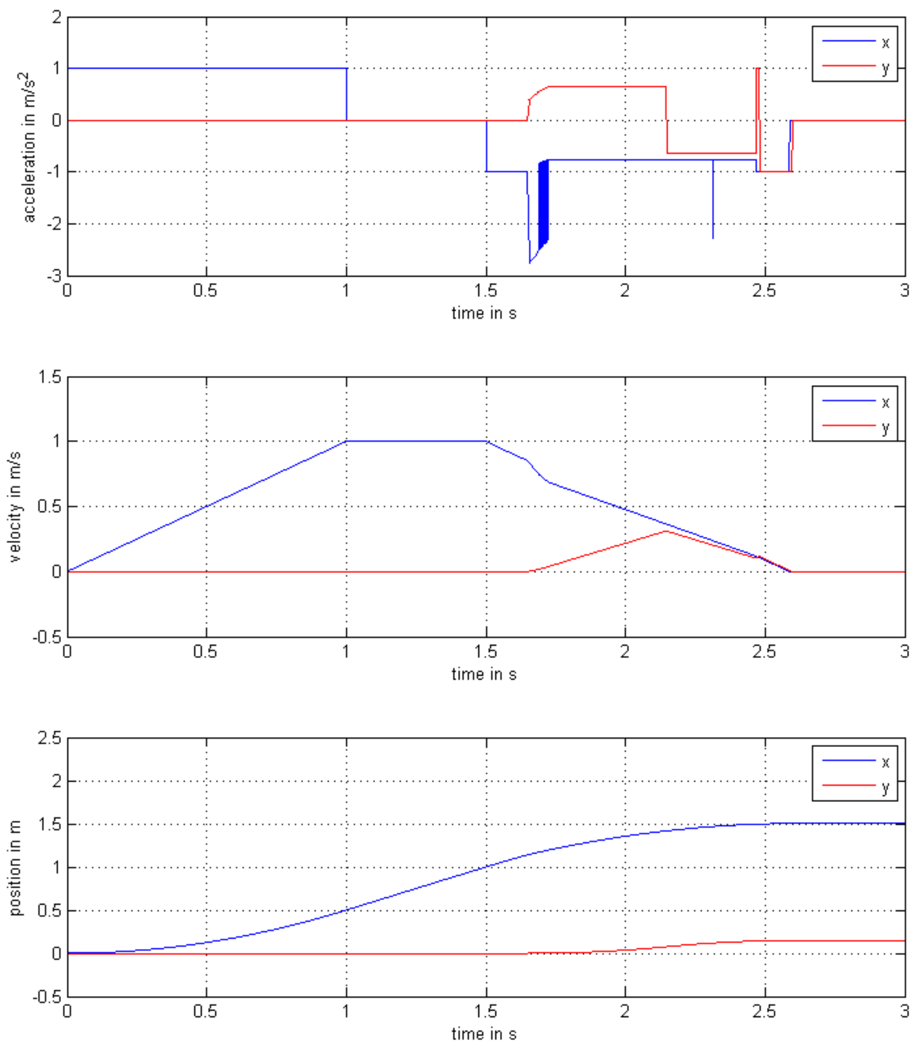


Figure 4.7: The accelerations, velocities and positions of simulation 3

4.4.4 Summary

Since the three simulations do not deal with external influences, these simulations only work to see what the trajectory planner really does. It should be clear that for all three simulations the positions all have the expected behavior. However for the velocities and accelerations there are not always smooth trajectories.

In simulation 1 and 2 there was an unwanted peek in the acceleration. A solution was given to prevent the acceleration to exceed the a_{max}. However no solution is available yet for the peek itself.

Chapter 5

Global Signal Analysis

In this Chapter an experiment will be done with a Turtle. The main purpose is to see how the signals in the motion scheme of a Turtle look like. In section 5.1 "Setup experiment and expectations" the setup of the experiment and the expectations will be explained and the results follow in section 5.2 "Results Turtle- movement".

5.1 Setup experiment and expectations

One should expect that the accelerations that go in the Turtle are different from the accelerations calculated in the trajectory planner. In Chapter 3 "Motion Scheme" it can be seen that the phi is not taken into account in the trajectory planner. The phi (ϕ) has its own trajectory planner, the "phimaker". This ϕ will have an influence on the accelerations.

For this experiment the original motion scheme is uploaded to the Turtle. There are some changes made in the scheme though. The role handler is overwritten with only the referential position, otherwise this is not possible. This block creates normally the referential position, but does this automatically and takes for example dribbling into account. Since for this experiment only driving is desired two constant points are initiated. These two points are in a straight line for the Turtle in y direction. After the Turtle reaches the first point it will wait a moment and then drive further to the second point. Here is a list with signals that are logged:

- v_{max} and a_{max}; To see if this is a constant signal while the Turtle is driving.
- Overshootflag for x- and y direction.
- Cases for x-,y- and ϕ direction.
- $a_{trajectoryplanner}$ for x- and y direction; These are the accelerations that are calculated by the trajectory planner and will not exceed a_{max}.
- $a_{parkingbrake}$ for x- and y direction; These are the accelerations after the parking brake at the "trajectory planner".
- x_{in} for x- and y direction; These are the positions after the "Global to Local"-block in Figure 3.1 in Chapter 3 "Motion Scheme"

- a_{in} for x- and y direction; These are the accelerations after the "Global to Local"- block in Figure 3.1. This is actually a position but because this is a real signal it can be differentiated twice which gives the accelerations.

Firstly the v_{max} and a_{max} should stay constant since the Turtle was not dribbling. Secondly the accelerations calculated by the trajectory planner will be different from the simulations seen in Chapter 4 "Trajectory Planner". The Turtle drives now in the real world so external deviations are taken into account and vision will correct these deviations.

5.2 Results Turtle- movement

Since the path of the Turtle is intended to be in purely y direction the accelerations in x- and ϕ direction are deviations that the robot detects and tries to resolve.

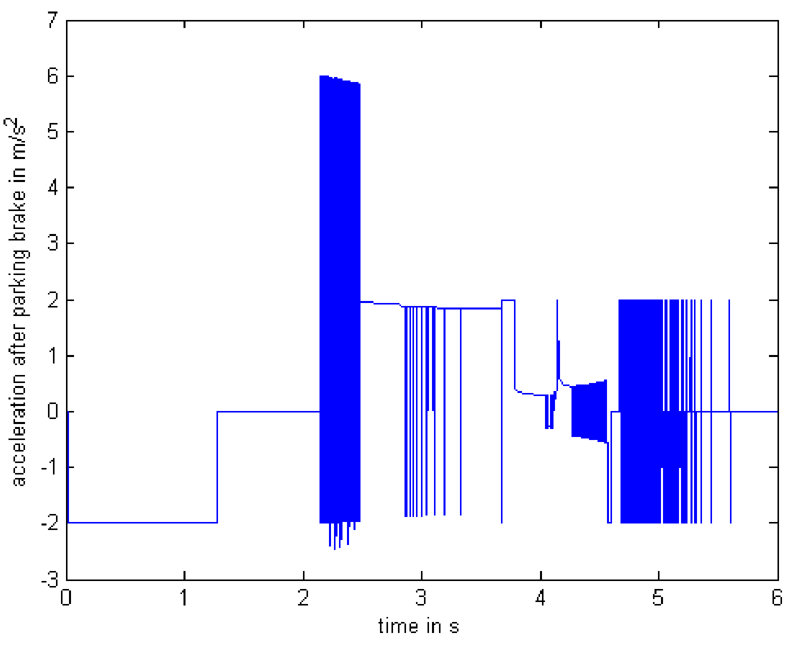


Figure 5.1: The acceleration after the parking brake for 1 movement in y direction

The acceleration after the parking brake in y direction is shown in Figure 5.1. The acceleration after the parking brake in the x direction can be found in Figure D.1 in Appendix D. Every time step will be explained in the following enumeration.

- Time[0 - 2.14]: Normal expected behavior of starting value for acceleration and driving at maximal speed.
- Time[2.14 - 2.5]: Here are some strange peeks in the signal. In Figure 5.2 in the top figure a zoom is given of Figure 5.1 and it shows the peeks more

clear. The lowest figure shows the cases on the time of the peeks. At the cases it can be seen that first the Turtle was in case 3, then went to case 21 for some time steps and then alternately went in case 3 respectively case 5. Effectively this acceleration signal is zero what means that the Turtle drove constantly on maximal velocity. The alternating cases show that the Turtle is constantly trying to drive maximum velocity until he does or drives too fast. This is because the threshold of the velocity is too small. When the threshold of the velocity is somewhat bigger the Turtle's velocity stays in the converged area and stays in case 3. Because the case 5 is visited and the Turtle is not decelerating to stop in time an overshoot flag is set so the acceleration gets amplified with 3.

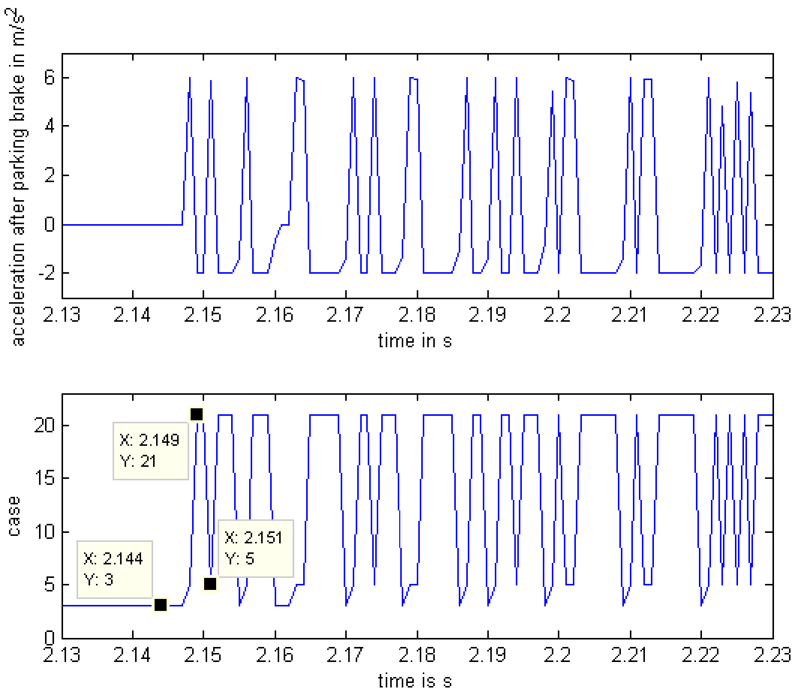


Figure 5.2: A zoom of the strange peeks in Figure 5.1 and the cases on time frame[2.14 - 2.5]

- Time[2.5 - 3.8]: In this time frame of the acceleration after the parking brake there are also some strange spikes but not so frequently as in the previous time frame (see top figure Figure 5.3). These spikes can be declared again by looking at the case (see lower figure in Figure 5.3). There are only two cases, 4 and 22. When one remembers section 4.4.2 "Movement in x direction and y direction", it would be clear that this is because a small deviation in x direction occurs. This can be seen in Appendix D in Figure D.2

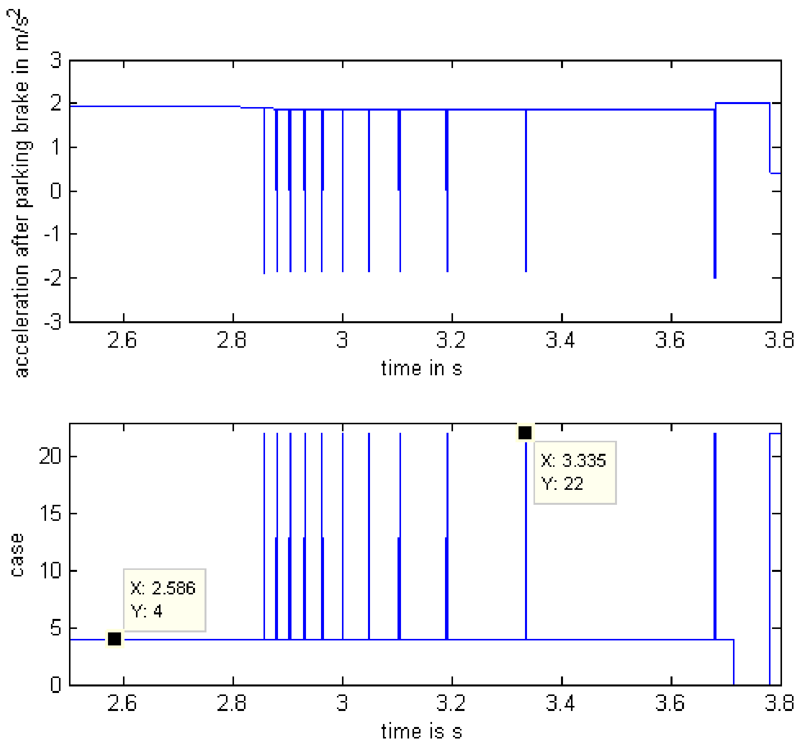
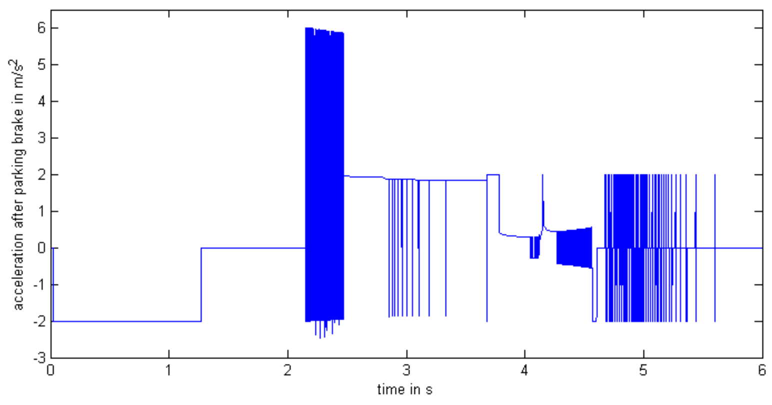


Figure 5.3: A zoom of the strange peeks in Figure 5.1 and the cases on time frame[2.5 - 3.8].

- Time[3.8 - 4.65] This is a very messy signal. In principle this behavior can be declared like the previous time step. The difference now is that the deviation in x direction is much larger and causes more disturbance for the y direction.
- Time[4.7 - 6]: In this time frame it looks like the x direction is converged but the y direction is still in some time steps trying to reach its final position. Effectively speaking this acceleration in y direction is zero.

Before this, nothing has been said about the signal after the block "Global to Local". In Figure 5.4 the acceleration after the parking brake and the acceleration after the block "Global to Local" are drawn. If one compares these two figures with each other it can be seen that the signal after "Global to Local" is wrinkled. This is caused by the the "Drift Control" and the corrections in ϕ direction. This should be no problem by for example comparing this signal with the signal from an accelerometer. The small inaccuracy should be small enough to neglect and some big hills in the signal should be followed by the Turtle. So the accelerometer should also give such acceleration values.



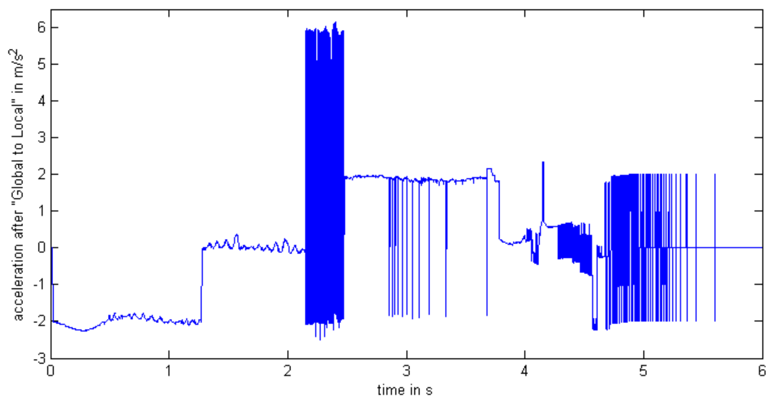


Figure 5.4: Comparison between acceleration after parking brake and acceleration after "Drift Control"

Chapter 6

Possible control systems

Today many slip control systems exist, mostly developed for the automotive industry. The basics of some of these control systems will be explained in section 6.1 "Slip control in other vehicles". Finally in section 6.2 "Initial Ideas" some initial ideas are sketched for a traction controller in the Turtles.

6.1 Slip control in other vehicles

A well-known slip control used in cars is Anti-lock braking system (ABS). This control system only works for decelerating. For accelerating another system can be used like Anti-Slip Regulation (ASR). Here in short the basic principle of ABS and ASR. There is also an Electronic Stability Program (ESP), this contains ABS and ASR, but also controls lateral dynamics.

6.1.1 ABS and ASR

When a car is driving, all its wheels should have the same speed. This speed should be the same as the speed of the vehicle itself. ABS detects that the car is slipping by measuring and comparing the wheel speeds. If the car slips, ABS controls the braking pressure and makes sure that the car decelerates without slipping. ABS makes sure that the slip of a car stays in the stable area, so below 20 percent.

ASR operates on the same principle as ABS, but instead of controlling the braking pressure ASR controls the engine torque.

6.1.2 ESP

ESP is actually a coordinating control system over ABS and ASR. ESP decides whether ABS or ASR should be used. ESP itself detects the loss of steering control. When it does, it controls the wheel speeds individually. For example ESP brakes the outer front wheel to counter oversteer and to counter understeer it brakes the inner rear wheel. Understeer occurs when the maximal traction of front wheels is exceeded. For example when there is understeer in a car it spins out in the turn of the road. Oversteer occurs when the maximal traction of the rear wheels is exceeded. For example this happens when the car spins. To use ESP a car must be equipped with some new sensors,[1] like:

- Steering wheel angle sensor; measures the desired steering behavior applied by the operator
- Yaw rate sensor; measures the rotation speed of the vehicle around its vertical axis.
- Lateral acceleration sensor: This measures the centrifugal forces acting on the vehicle.

6.2 Initial Ideas

In previous section 6.1 "Slip control in other vehicles" anti-slip control systems are illustrated that can be used in for example a car. Cars are non-holonomic vehicles what means that they cannot move in every direction. If a car drives in a straight line without slipping the wheels have at any moment the same velocity as the car. When a car makes a turn it is a bit more complicated, but this can be solved. This is why control systems mentioned in previous Chapter can be used for cars.

In Chapter 2 "The Turtles" is explained that the wheels of a Turtle do not rotate in the direction the Turtle drives. The Turtle has wheel encoders that measures the distance they traveled. A control system like ABS or ASR cannot be used in a Turtle because the wheel speeds can not be compared to the vehicle velocity. The velocity of the vehicle is always different from the angular velocity the wheels have. This is why control systems as ABS or ASR cannot be used for a Turtle. ESP cannot be used for the Turtles either because in the Turtle there is no steering wheel.

Because of this there should be thought of other ways to detect slip. Here are some ideas created during the developing:

- The Turtle could calculate its wheel speeds beforehand. If one looks at the from Equation 1.2 derived Equation 6.2, one would notice that the mass on a robot can be determined and that the R_w and T are known. This means that if F_w would be calculated the acceleration the wheels should have could be achieved.

  $$a_i = \frac{T_(i)}{R_w \cdot m} - \frac{F_w}{m} \tag{6.1}$$

  Now the difference between the with the accelerometer measured actual acceleration and the calculated acceleration can be known. The major disadvantage is that like mentioned in section 1 "Traction and slip" the F_w depends on the coefficient of fraction μ. Since this μ is different for every road(type), every time before a Turtle drives on the field it would need to drive without slipping to do measurements to determine the μ. The calculation of this μ would be very difficult since the omniwheels have for each driving angle a different μ. (See Chapter 2 "The Turtles")

- The Turtles are powered by a DC elektromotor. In Figure 6.1 a simplified scheme is illustrated for an elektromotor. The amplifier is assumed to be ideal, at the motor slip, tolerance , vibrations and inertial load are chosen to be negligible and no noise is assumed at the double differentiation. In this simplified model with known voltage, specifications of the amplifier/motor and with the inertial load of the Turtle it can be calculated what the ratio in Equation 6.2 should be.

$$U/a - ratio = \frac{voltage_{wheel}}{acceleration_{wheel}} \tag{6.2}$$

If the position is measured with encoders the actual ratio can be calculated. When this measured ratio differs too much from the modeled ratio slip is detected.

A major problem in this idea is that the inertial load of the Turtle is not easy to determine. When the robot drives in a certain direction its inertial load will be different for every wheel at any time.

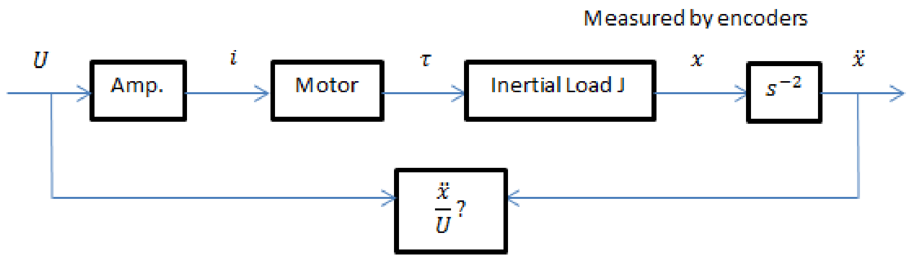


Figure 6.1: Simplified scheme of an electromotor

- As can be seen in Figure 3.1 in Chapter 3 "Motion Scheme" the input for the low level control is the referential position. However like shown in Chapter 5 "Global Signal Analysis" this signal is differentiable. When this referential position is differentiated twice the local acceleration input in the low level control is known. The local acceleration of the Turtle can be measured with an accelerometer. When there is a difference between these accelerations the Turtle can detect slip.

  A problem in this method though is that sometimes the signal has strange behavior like shown in Chapter 4 "Trajectory Planner" and Chapter 5 "Global Signal Analysis. First this looked like the best method to implement a traction controller, but later on better ideas were developed. If one is interested this traction controller is available in Simulink but is not included in this report.

- The big disadvantage in previous point, that the signals sometimes have a strange behavior, can be reduced by controlling in the trajectory planner instead of low level control. In low level control every time step the new calculated path needs to get controlled and the strange behavior in the signals is more.

  In this option the a_{max} is controlled in the trajectory planner. This method will be explained further in the next Chapter 7 "Traction controller".

Chapter 7

Traction controller

In this Chapter a traction controller is developed for preventing slip while accelerating and decelerating. Firstly in section 7.1 "Basic idea" the basic idea of the traction controller will be explained in some more detail. Then in section 7.2 "Flowcharts" two flowcharts are created to show how this traction controller really works. In section 7.3 "Implementation" the implementation in the trajectory planner is shown and in section 7.4 "Simulations traction controller" some simulations are done to show what the traction controller does to the signals in the trajectory planner.

7.1 Basic idea

Like mentioned in previous Chapter is the traction controller developed in this report based on controlling the a_{max}. This control happens after the "trajectory preprocessor" in Figure 3.1 in Chapter 3. This because here a_{max} and v_{max} already get changed for dribbling. The a_{max} and v_{max} get used in the trajectory planner, so the traction controller must have been implemented before or in the trajectory planner. Since the signal has some strange behavior like shown in Chapter 4 "Trajectory Planner" and Chapter 5 "Global Signal Analysis" it would be helpful to use the cases in the traction controller. For that reason the traction controller is implemented ín the trajectory planner.

The a_{max} gets controlled by multiplying it with a so called safety factor. This safety factor is initial equal to 1 and never bigger than 1 or smaller than 0.1. When the Turtle detects slip it changes the safety factor to a value below 1 so the a_{max} is lower. By a feedback loop for the safety factor and the difference in previous safety factor, the ideal safety factor will be evaluated. Since slip behavior is very unstable and it is not desired to drive slowly unnecessary, the safety factor will be increased when slip is not detected anymore.

7.2 Flowcharts

In this section two flowcharts are developed to show how the traction controller works inside the trajectory planner. In the first subsection 7.2.1 "??" the implementation needed for the traction controller is explained. In subsection 7.2.2 "Traction_Controller" the subroutine traction_controller is discussed in detail.

7.2.1 Useful code in trajectory planner

In Figure 7.1 a flowchart is illustrated with the part of the trajectory planner where the traction controller is implemented in. In this flowchart no directions are shown because it holds for both, x- and y direction.

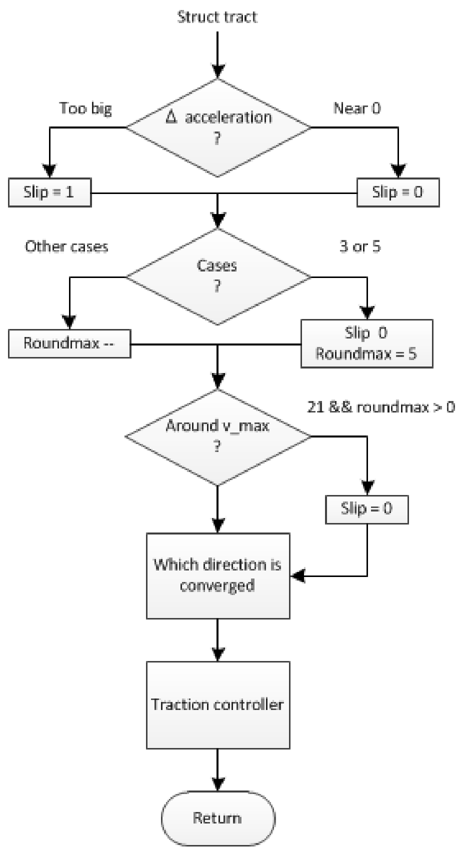


Figure 7.1: Flowchart of a piece of the trajectory planner with the traction controller

At first the Turtle checks if it detects slip. When the difference between the measured acceleration and the input acceleration is too big the variable *slip* is set to 1. The value used as threshold for the difference between the two accelerations should be based on such a diagram as Figure 1.2 in Chapter 1 "Traction and slip". For the accelerations also exists a stable area. This stable are should be found for the slip based on difference in acceleration.

Shown in section 5.2 "Results Turtle- movement" is that while the Turtle is driving around v_{max} there is some strange behavior in the signal input. This only occurs when the cases are 3 or 5. So when the cases of the previous time step (when the Turtle slipped) are 3 or 5, slip is ignored and the corresponding variable is set back to 0. Because sometimes when cases are 3 or 5 alternately it can happen that case 21 occurs. For this reason roundmax is set to 5 when case 3 or 5 is detected. When in a time step the case is not equal to 3 or 5 the roundmax is decreased. Now in short, when case 21 occurs and the Turtle was the previous 5 time steps never in case 3 or 5 the Turtle is not around v_{max} and slip cannot be ignored.

After that in the trajectory planner it is checked if the x- or y direction is converged or not. Depending on this the accelerations are calculated. Always before these get calculated the subroutine traction_controller gets called.

7.2.2 Traction_Controller

In Figure 7.2 a Flowchart is illustrated of the subroutine traction_controller. This is the block in the flowchart of the previous section 7.2.1 "Useful code in trajectory planner".

In the flowchart the safety factor is shorted with Sf and the difference in safety factor of previous time step with Δsf_last.

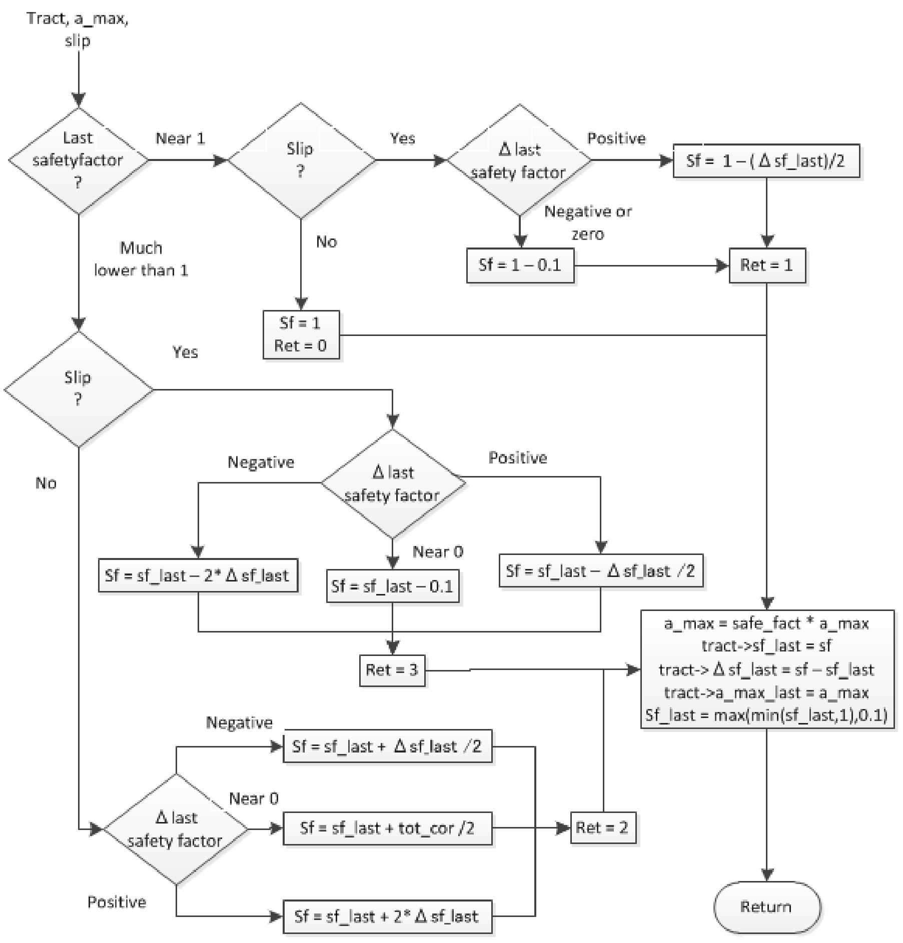


Figure 7.2: Flowchart of the actual traction controller

Before the flowchart will be explained, first a numeration of inputs and variables used in the traction controller:

- a_max_last; stored in a struct called tract. This variable contains the value of the a_{max} from the previous time step. For the traction controller this value will not be used. Because the safety factor is controlled the a_{max} does not have to be controlled.

- sf_last; also stored in tract and contains the safety factor of the previous time step.
- Δsf_last; stored in tract and contains the difference in safety factor in previous time stap.
- cases x respectively y; these contain the cases for respectively x- or y direction visited in subroutine Check Case (section 4.3 "Check Case" in Chapter 4 "Trajectory Planner" in the previous time step.
- overshootflag x respectively y; these contain whether an overshootflag is set for x respectively y or not. This value will also not be used in the traction controller. One could think that this should be considere for the reason that an overshootflag can cause a magnification of the accelerations. Well the decrease of the a_{max} while slipping also is magnified, so indirectly it ís considered.
- roundmax x respectively y; these contain if the Turtle was driving at a velocity near v_{max}.

Firstly the safety factor of the previous time step is evaluated. To keep it clear this is subdivided in two subsections 7.2.2 "Previous safetyfactor much lower than one" (follow arrow down "Much lower than 1" in the flowchart) and 7.2.2 "Previous safetyfactor near one" (follow arrow to the right "Near 1").

Previous safetyfactor much lower than one

If this safety factor is lower that the threshold it means that in the previous time step the a_{max} is controlled. After that the Turtle evaluates its slip variable.

1. If the Turtle slips in this time step the traction controller will decrease the safety factor. When the Turtle slips (follow the right arrow in the flowchart "Yes") the difference in safety factor can be:
    - **positive** (follow arrow down "Positive"). The last time slip was not detected so the safety factor was increased. Because the Turtle is slipping now the positive correction was too high assuming the ideal safety factor is constant. This is corrected by taking the previous time step again, but then multiplied with 0.5 and subtracted from the last safety factor. In this way the safety factor is converging to an ideal value.
    - **negative** (follow left arrow "Negative"). In the previous time step was slip detected and now again. This means that the correction of the previous step was not high enough. The same correction is used as the previous time but then multiplied with 2 and subtracted from previous safety factor. Note that the difference in safety factor is an absolute value.
    - **near 0** (follow right arrow "Near 0"). In the previous time step the difference in safety factor was very very small. In this case when slip is detected nothing will happen so when the difference in last safety factor is really too small a normal correction will be introduced.

2. If the Turtle does not slip the safety factor must be increased. When the Turtle does not slip (follow the arrow down in the flowchart "No") and the difference in safety factor can be:

    - **positive** (follow the middle arrow "Positive"). One time step ago also no slip was detected. This means that the correction back to 1 was not high enough. For this reason the same correction is used as the previous time but then multiplied with 2 and added to the last safety factor.
    - **negative** (follow the upper arrow "Negative"). The correction of the safety factor in the previous time step was negative. This means that the correction was too high. In trying to let the safety factor converge, the difference in safety factor of the previous time is divided by 2 and added to the safety factor of previous time. This in such a way that the safety factor converges to the ideal safety factor.
    - **near 0** (follow the arrow down "Near 0". If the previous difference in safety factor was 0 and no slip is detected the total correction is divided by 2 and added to the previous safety factor to make sure the maximal acceleration does not stay too low.

Previous safetyfactor near one

If the safety factor of the last time step was near 1 it means that there can be two possibilities. The Turtle is now

1. slipping (follow the arrow to the richt "Yes". The difference in the last safety factor can be:

    - **positive** (follow arrow to the right "Positive"). The last time step no slip was detected and the correction of the safety factor was too big. Now to let the safety factor converge the safety factor is now reduced with the difference of the previous safety factor divided by 2.
    - **negative or zero** (follow arrow down "Negative or zero"). In this case the traction controller will change the safety factor with a standard value.

2. not slipping (follow the arrow down "No"). Now there is no slip and previous time step there was no or negligible slip. Set the safety factor to 1.

After all this cases finally the values should be stored and send. In the new a_{max} the safety factor influences its value, the difference in safety factor gets calculated and the safety factor is minimized to 1 or maximized to 0.1 so no strange values will occur for a_{max}.

7.3 Implementation

In this section the actual implementation in the C - file: "trajectory planner" is presented. Sometimes pieces of the original trajectory planner are printed too but in the comments can be seen which pieces that are. Some added pieces especially for the traction controller are not displayed here like the code to implement extra ports for in- and output variables or lines that includes header files. In every case one line is added to define the case, this is also not added. The cases are explained in section 4.3 "Check Case".

This section is divide in several subsections. In the first subsection 7.3.1 "Struct traction" the struct called traction used in the trajectory planner is defined. In the second subsection 7.3.2 "Detecting slip" the code for detecting slip is defined. In subsection 7.3.3 "Converged directions" the code where the traction controller gets called is added and finally the subroutine traction_controller can be found in the fourth and last subsection 7.3.4 "Controlling slip".

7.3.1 Struct traction

Some variables used in the traction controller are stored in a struct. This struct is defined in this header file:

Listing 7.1: Struct traction

```
#ifndef structs_h
#define structs_h

struct traction {

    double a_max_last;  %a_max of previous time step
    double sf_last;     %safety factor of previous time step
    double dif_sf_last; %previous change in safety factor
    int casesx;         %case in x direction
    int casesy;         %case in y direction
    int overshootflagx; %overshootflag in x direction
    int overshootflagy; %overshootflag in y direction
    int roundmaxx;      %counter in x direction
    int roundmaxy;      %counter in y direction

};typedef struct traction tract;
#endif
```

7.3.2 Detecting slip

Here some calculations are done before subroutine traction_controller gets called. Firstly it tries to detect slip and ignores slip when the Turtle has nearly v_{max} as a velocity. Following the implementation for this:

Listing 7.2: Detecting slip

```
if ( dabs(a_measx - a_inx) >= EPSILON_SLIP ){// slip in x
    slip[0] = 1;
}
if ( dabs(a_measy - a_iny) >= EPSILON_SLIP ){// slip in y
    slip[1] = 1;
}
if( tract.casesx == 3 || tract.casesx == 5 ){//case 3 or 5
    slip[0] = 0;
    tract.roundmaxx = 5;
} else{
    tract.roundmaxx--;
}
if( tract.casesy == 3 || tract.casesy == 5 ){
    slip[1] = 0;
    tract.roundmaxy = 5;
} else{
    tract.roundmaxy--;
}
if( tract.casesx == 21 && tract.roundmaxx > 0 ){//v_max?
    slip[0] = 0;
}
if( tract.casesy == 21 && tract.roundmaxy > 0 ){
    slip[1] = 0;
}
```

7.3.3 Converged directions

In section 4.1 "Outputs" there are four possible ways illustrated where the Turtle can be in with. The following order is the same as in that section and the added lines are written in that part of the file.

Outputs trajectory planner

Every part has the same variables as output off course and this looks like:

Listing 7.3: Ouputs trajectory planner

```
y[0] = q_x;                    //from original file
y[1] = q_y;                    //from original file
t[0] = tract.sf_last;
t[1] = tract.dif_sf_last;
t[2] = tract.a_max_last;
c[0] = tract.casesx;
c[1] = tract.casesy;
c[2] = tract.overshootflagx;
c[3] = tract.overshootflagy;
c[4] = tract.roundmaxx;
c[5] = tract.roundmaxy;
```

x- and y direction converged

For the first possibility that both x- and y direction are converged. It is desired that traction control gets applied only once. Otherwise the a_{max} could get corrected twice. If x- and y direction are both converged there is no case or overshootflag determined so these are set to the default value of -1 respectively 0. Finally all values of the struct are outputted to be used as input for the next time step. Here is the implementation:

Listing 7.4: x- and y direction converged

```
controlled = traction_control(&a_max, slip[0],&tract);
q_x = dmin(dmax((xfdot-xodot)/SAMPLE_TIME,-a_max),a_max);//from original file
tract.casesx = -1;
tract.overshootflagx = 0;
if( !controlled ){
    controlled = traction_control(&a_max, slip[1],&tract);
}
q_y = dmin(dmax((yfdot-yodot)/SAMPLE_TIME,-a_max),a_max);//from original file
tract.casesy = -1;
tract.overshootflagy = 0;
```

Only x direction converged

In the below implementation only the x direction is converged. For this the same reasoning holds for the x direction as for the previous implementation. However for the y direction calculate _time gets called. this means that a case and an overshootflag will be defined.

Listing 7.5: Only x direction converged

```
controlled = traction_control(&a_max, slip[0],&tract);
q_x = dmin(dmax((xfdot-xodot)/SAMPLE_TIME,-a_max),a_max);//from original file
tract.casesx = -1;
tract.overshootflagx = -1;
if( !controlled ){
controlled = traction_control(&a_max, slip[1],&tract);
}
ETA_y = calculate_time(yf, yodot, yfdot, v_max, a_max, &q_y);//from original file
povershootflag[1]=overshootflag;//from original file

tract.overshootflagy = overshootflag;
```

Only y direction converged

For the following C- code hold the same reasoning as for previous implementation. However here it is vice versa for x- and y direction.

Listing 7.6: Only y direction converged

```
controlled = traction_control(&a_max, slip[0],&tract);
ETA_x = calculate_time(xf,xodot,xfdot,v_max,a_max,&q_x);\\from original file file
tract.overshootflagy = overshootflag;
if(controlled){
    overshootflag = 0;
}
povershootflag[0]=overshootflag; \\from original file
if ( !controlled ){
    controlled = traction_control(&a_max, slip[1],&tract);
}
q_y = dmin(dmax((yfdot-yodot)/SAMPLE_TIME,-a_max),a_max);
tract.casesy = -1;
tract.overshootflagy = 0;
```

No directions converged

For the fourth case the same idea holds. Following code is implemented before the loop where the ideal driving angle gets calculated (See section 4.1 "Outputs".

Listing 7.7: No direction converged

```
controlled = traction_control(&a_max, slip[0],&tract);
ETA_x = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);//from original file
if(controlled){
    overshootflag = 0;
}
povershootflag[0]=overshootflag;//from original file
tract.overshootflagx = overshootflag;
if ( !controlled ){
        controlled = traction_control(&a_max, slip[1],&tract);
}
ETA_y = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);//from original file
povershootflag[1]=overshootflag;//from original file

tract.overshootflagy = overshootflag;
```

7.3.4 Controlling slip

Finally the C- code of the traction controller traction_controller is given below. At first used variables are declared and the total correction is calculated. Secondly are the steps that are shown in the flowchart in previous section 7.2 "Flowcharts" exactly implemented.

Listing 7.8: Subroutine traction_control

```
int traction_control( double* a_max,int slip ,struct traction *tract){

double EPSILON_SF_LAST    = 0.005;//Value should be tuned
double EPSILON_dSF_LAST   = 0.05; //Value should be tuned
double EPSILON_STAND_VAL  = 0.1;  //Value should be tuned
double safe_fact          = 1;
double ret                = -1;

double sf_last            = tract->sf_last;
double dif_sf_last        = tract->dif_sf_last;
double a_max_last         = tract->a_max_last;

double tot_cor            = 1 - sf_last;

if ( sf_last > 1 - EPSILON_SF_LAST ){
    if ( slip ){                                // Decrease sf
        if ( dif_sf_last >= EPSILON_dSF_LAST ){ //Smaller +corr
            safe_fact = 1 - dabs(dif_sf_last/2);
        } else{                                 // Correction is nearly 0
            safe_fact = 1 - EPSILON_STAND_VAL;
        }
        ret = 1;
    } else{                                             // No correction , no slip
        safe_fact = 1;
        ret            = 0;
    }
} else{                                                 // There is a correction going on
    if ( slip ){                                        // Decrease sf
        if ( dif_sf_last >= EPSILON_dSF_LAST ){         //Smaller +corr
            safe_fact = sf_last - dabs(dif_sf_last/2);
        } else if( dif_sf_last <= -EPSILON_dSF_LAST ){  //Larger -corr
               safe_fact = sf_last - 2*dabs(dif_sf_last);
        } else {// Standard correction
            safe_fact = sf_last - EPSILON_STAND_VAL;
        }
        ret = 3;
    } else{// Increase sf
        if ( dif_sf_last >= EPSILON_dSF_LAST ){         //Larger +corr
            safe_fact = sf_last + 2*dabs(dif_sf_last);
        } else if( dif_sf_last <= -EPSILON_dSF_LAST ){  //Smaller -corr
            safe_fact = sf_last + dabs(dif_sf_last/2);
        } else{// Standard positive correction
            safe_fact = sf_last + dabs(tot_cor/2);
        }
        ret = 2;
    }
}
if ( safe_fact > 1 ){
    safe_fact = 1;
} else if{ safe_fact < 0.1 }
    safe_fact = 0.1;
}
*a_max              = safe_fact * (*a_max);
tract->sf_last           = safe_fact;
tract->dif_sf_last       = safe_fact - sf_last;
tract->a_max_last        = *a_max;

return ret;
}
```

7.4 Simulations traction controller

For the explained traction controller in previous sections the measurements done by an accelerometer are essential. It is not possible to test this traction controller on a Turtle because the accelerometer on the Turtles cannot be used properly. Because of this, only simulations on the computer are run for this traction controller.

In Figure E.1 in Appendix E the simulation setup is illustrated and explained.

With a signal builder an error is added to the simulation. Because the signal builder creates a signal that gets multiplied with the accelerationvalues its default value is 1. This is why in this simulation will be spoken of an error + 1. This error is applied in a referential signal, so whatever the traction controller does, the imposed error stays the same.

In the first two experiments just a movement in the x direction is modeled. In section 7.4.1 "Simulation with no error" a first simulation is performed with no error. In the next section 7.4.2 "Simulation with a small error" a small error is imposed. In section 7.4.3 "Simulation with an error and x- and y direction movement" an y movement is introduced to see what happens if two directions are not converged. Finally in section 7.4.4 "Simulation for slip at maximal velocity it is checked if the traction controller does not do anything while driving at maximal speed.

7.4.1 Simulation with no error

In this simulation no error is introduced to the system. Only one movement in x direction is simulated. This means that the accelerations, velocities and positions should be the same as in simulation 1 of Chapter 4 "Trajectory Planner. The safety factor should be equal to 1 and the variable roundmaxx should only have a value of 5 when the Turtle is in case 3 and drives on maximal speed.

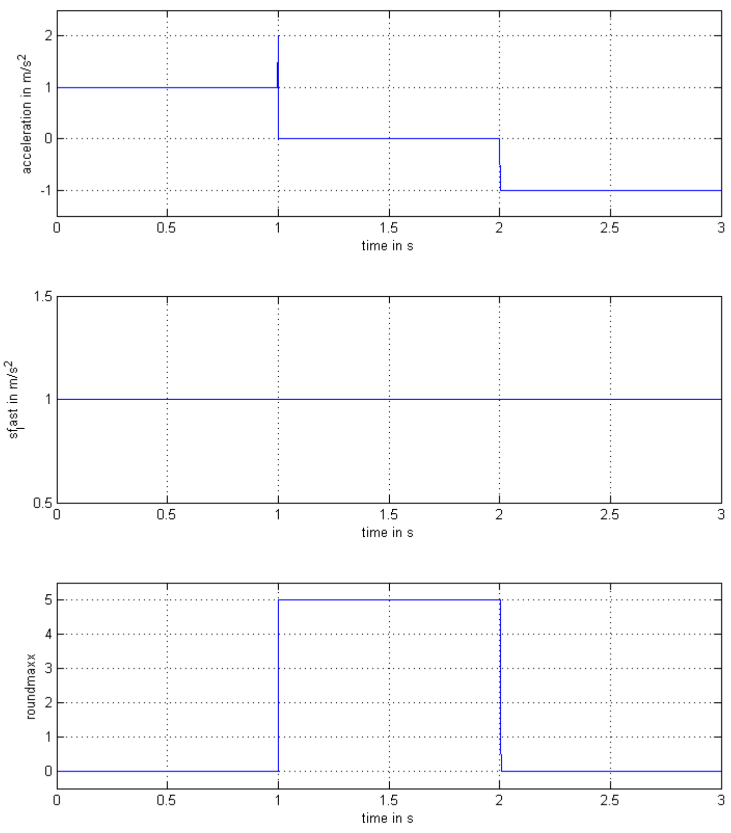


Figure 7.3: Acceleration in x direction, safety factor and roundmax in x direction

Results

The results are exactly the same as the expectations. In Figure 7.3 on the previous page the acceleration in x direction, the safety factor and the roundmax in x direction are shown.

As can be seen the same peek as in section 4.4.1 "Movement in x direction" is visible. Only when the acceleration is zero the roundmaxx is equal to 5. This shows that the traction controller does not do anything if no slip is detected.

7.4.2 Simulation with a small error

In this simulation a certain error is given. In reality this will not occur on this way, because whatever the traction controller does the error will not decrease. This means that the safety factor will get a minimal value defined by the traction controller, 0.1. Nevertheless with this simulation one can see what the traction controller does and how acceleration changes.

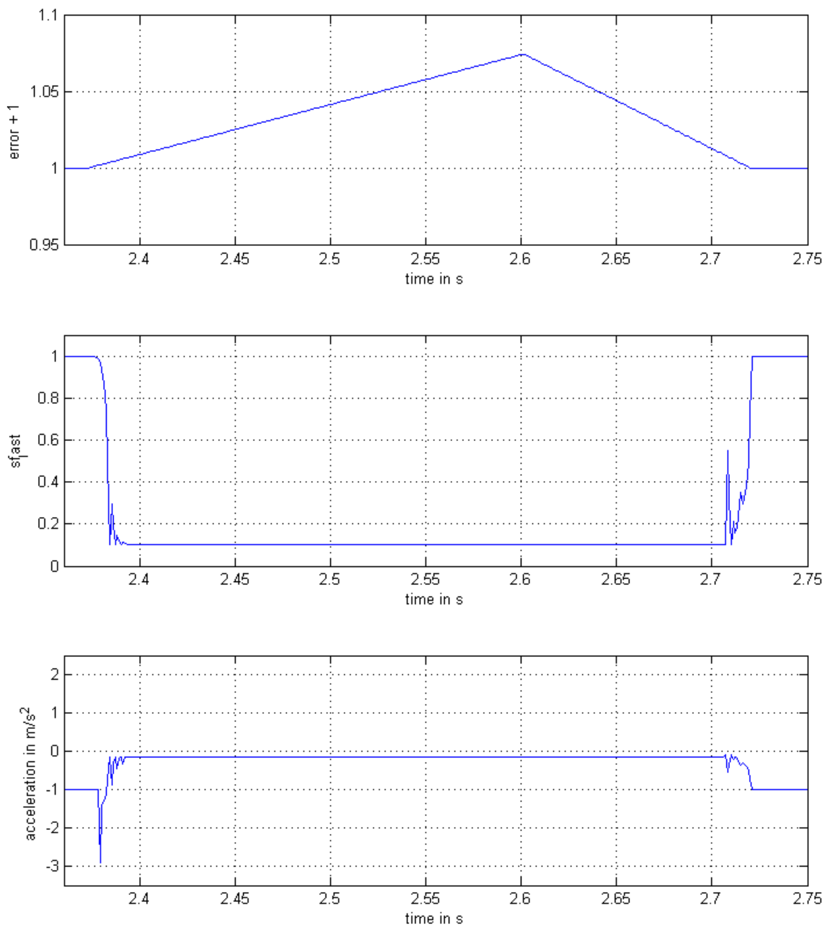


Figure 7.4: The piece where the error+1 is introduced, the safety factor in that same time frame and the acceleration value in x direction

Results

In the top figure of Figure 7.4 one can see what error is applied. These figures are zoomed in on a certain time frame where the error is applied. Outside that time frame the behavior of the signals is obviously exactly the same as in previous simulation.

It can be seen in the middle figure that when the error is bigger than a certain value the safety factor gets decreased. Because the error stays the same the Turtle constantly thinks it is slipping. When the safety factor reaches 0.1 it converges. When then the error becomes small again the safety factor gets increased again and becomes 1.

The maximal acceleration in the lowest figure follows this behavior since those two get multiplied. Right before the acceleration is decreased there is a peek in the signal. This peek is caused because of the overshootflag. Because the Turtle is decreasing its acceleration, but still has to drive some distance, the overshootflag is set. This error is not desired but it does not matter much.

Because the a_{max} is equal to 1, Equation 7.4.2 holds. This can be seen in Figure 7.5. The top figure shows the error+1.

$$a_{max} = safetyfactor \cdot a_{max} = safetyfactor \tag{7.1}$$

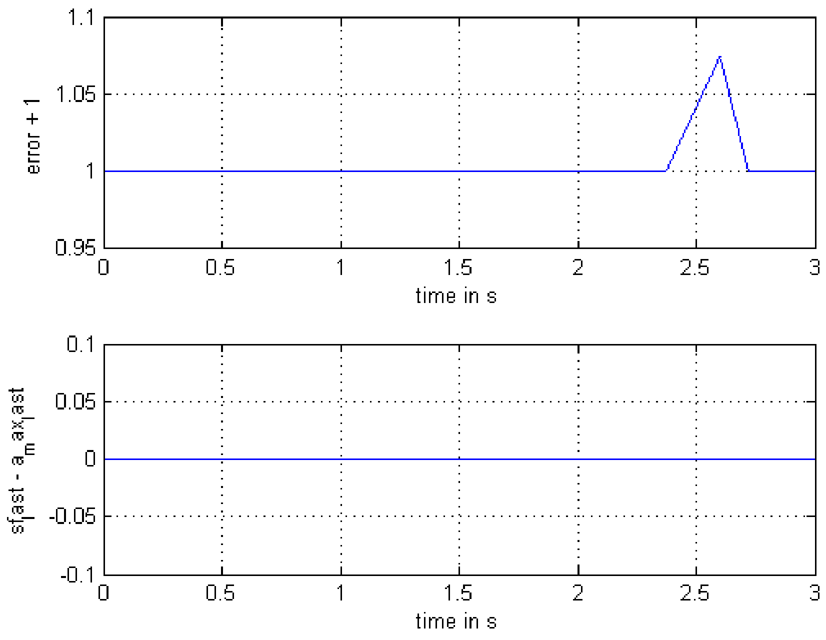


Figure 7.5: The error+1, the acceleration and the x- and y velocity

7.4.3 Simulation with an error and x- and y direction movement

This is a short experiment to show what the traction controller does with the speed of the Turtle. This simulation is somehow similar to that from section 4.4.2 "Movement in x direction and y direction" only here an error is introduced.

Results

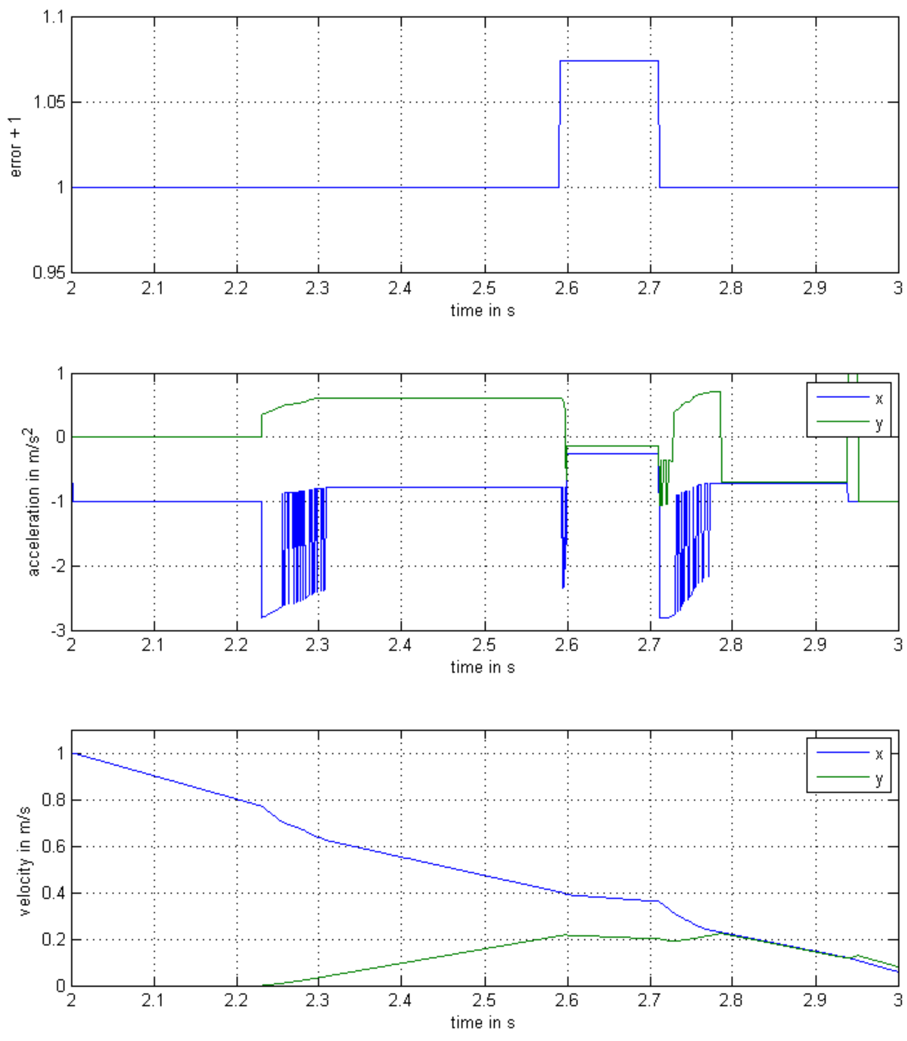


Figure 7.6: The error, acceleration in x- and y direction respectively velocity for movement in x- and y direction

In Figure 7.6 the error+1 imposed on the model is illustrated. In the second figure the acceleration is shown.

It is clear that the traction controller does its work nicely. The acceleration is purely constant around 0.1 and the direction with the overshootflag is 0.3. This is because the safety factor gets limited. It can be seen that from the moment no error is noticed an overshoot gets set. This does not really matter because in reality when the Turtle slips the a_{max} gets controlled. On that moment it does not matter whether an overshootflag is set or not.

On the moment the error + 1 is imposed, the velocity in the third figure has a much lower slope. This is very logic, because the acceleration is limited. The same a_{max} is used for both directions.

7.4.4 Simulation for slip at maximal velocity

In this section a small simulation is performed to show that slip at maximal velocity is ignored. In this case the v_{max} is set to 0.5 so the v_{max} is early reached.

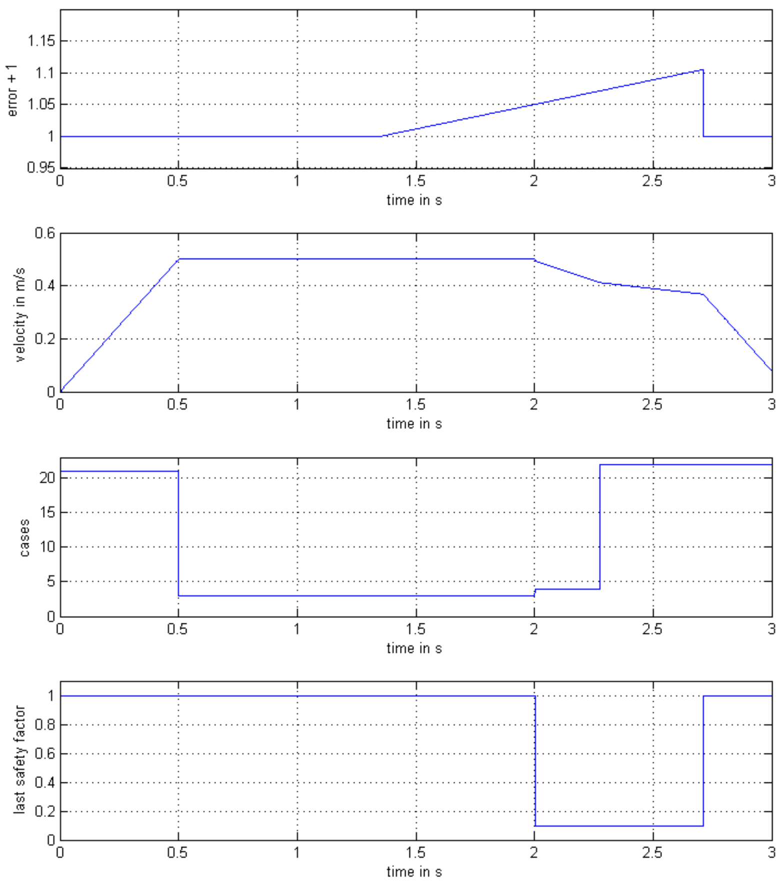


Figure 7.7: The error, velocity in x (driving) direction and the cases in the x direction

Results

In Figure 7.7 one could see that the error + 1 is imposed right before $time = 1.5sec$. In the lowest figure it can be seen that the case is 3 in the time frame [0.5 - 2]. While the error + 1 is imposed the case is 3 and the velocity is 0.5. Right when the Turtle is not in case 3 anymore the the a_{max} is controlled. This can be seen in the safety factor, drawn in the lowest figure.

7.5 Improvements of the traction controller

Because it was not possible to test the traction controller on the Turtles, it was also not possible to tune the parameters and build more features into it. Here some features that are not included in the tractioncontroller explained in this Chapter.

When in the struct tract a counter is included, it would be possible to make sure the safety factor does not variate too much. The idea of this counter is that it waits with the positive corrections and keeps the safety factor for some time steps on a constant value.

The traction controller does not react on an overshootflag that gets set. Even though an overshootflag amplifies the acceleration and could cause slip. It is true that when the safety factor changes a value, and that value gets amplified, the difference also gets amplified. So indirectly the overshootflag is taken into account. However this is probably not reliable. The overshootflag gets set while the traction controller is doing iets work. This should not matter because then the a_{max} is just made 3 times smaller, until the Turtle is not slipping anymore. When one could test the traction controller on the Turtle, one could investigate is this really does not matter.

Concluding Remarks

Slip can be prevented by driving very slowly. For a Turtle this would be a very bad strategy, because it wants to move over the soccer field as quick as possible. For that reason a fast adaptive traction control is desired for the Turtles.

Since the wheels of a Turtle are not in a rectilinear configuration it is not possible to use the same control systems as used in a car. For that reason is in this report chosen for another approach, namely comparing the intended acceleration with the actual acceleration measured by an accelerometer. For this it is important to clearly understand on which principles these intended accelerations are based on. So, this report contains an extensive research into the motion software of the Turtle, mostly in the trajectory planner. During this research some unwanted abnormalities are noticed and summarized in the next Chapter.

Basically the approach used in this report changes the maximal tolerable acceleration. The developed traction controller only works for accelerating and decelerating. A traction controller that also works when a Turtle is pushing against an opponent is shortly explained in the next Chapter, but this are only small basics.

The most trivial part of developing a controller was not possible in this report. It is very unfortnate that the accelerometer in the Turtle does not work properly because now the created traction controller is not validated, tuned and optimized. However in the simulations done on the computer one could see that the traction controller works like explained in the theory.

Future Work

The in this report developed traction controller is not tested yet on the Turtles. For this, one needs the accelerometer, or can use an observer to differentiate the encoders twice. When the traction controller is tested the parameters can be tuned. Parameters like thresholds and standard changes in safety factor.

One undesired property of the developed traction controller that is known for now is that the safety factor never reaches a constant value. This because when there is no slip anymore the safety factor should be 1 again. One could build in a counter that waits with positive corrections. This is not implemented in the in this report developed traction controller because it could not be tested properly.

As important thing in the development to a traction controller the accelerometer must be implemented. The accelerometer does not find itself in the middle of a Turtle, but somewhere on the side in a different coordinate system. This means the values obtained by the accelerometer are nonsense and must be transformed to the coordinate system of the Turtle. The rotating can be done by using a cosine direction matrix.[2] Although position is not important while discussing acceleration also the translation of the coordinate system is important. When the Turtle is only rotating in ϕ direction and the accelerometer is mounted on the side it will constantly detect an acceleration in a certain direction and this is not wanted.

While reading Chapter 4 "Trajectory Planner" one would have noticed some undesirable abnormalities are created by the trajectory planner. Following some examples.

- *In section 4.4.1 "Movement in x direction" a simple simulation is performed with the trajectory planner. The calculated acceleration can exceed the maximal acceleration which should not be possible. In the velocity and position the peek in the acceleration does not matter but nevertheless it is an irregularity in the signal.*
- *The behavior when case 3,5 or 21 (See section 4.3 "Check Case") gets visited, section 5.2 "Results Turtle- movement". Probably this behavior is caused because the threshold for the maximal velocity is too small. In the same section some vague peeks can be seen but then in another time frame, these are caused by a deviation in another direction and a alternate case (21 and 4)*

The traction controller described in Chapter 7 "Traction controller" is based on accelerating and decelerating of a robot on a path. However for the Turtles it is sometimes important to push against another robot. When a Turtle has a maximal acceleration without slip the pushing force against an opponent is maximal. The traction controller in previous section would lower the a_{max} until this is zero. This is not wanted and for that reason another idea should be performed. Following an example of such an idea.

- *Like mentioned in section 6.2 "Initial Ideas" the Turtle cannot compare its wheel speeds. However because in the motion scheme the signal out of the encoders gets decoupled from wheels to local coordinates the displacement of a Turtle is known in x, y and ϕ. With an observer can one trace the accelerations in x, y and ϕ direction the wheels think the Turtle drove. With the accelerometer one would be able to detect slip and correct the desired wheel speeds individually.*

Bibliography

[1] Rem- en slipgedrag.pdf, E. Gernaat (ISBN 987-90-808907-7-0) 02-09-2011

[2] Multibody Dynamics (4J400), N. van de Wouw; Technical University of Eindhoven (Department of Mechanical Engeneering) 2010

Appendix A

Decoupling local- to wheel coordinates

w_i is the wheel speed of the i^{th} wheel and V_r is the velocity vector of the robot. First the wheel speeds are defined like:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -|V_r| \cdot \sin(30+\phi) + R_1 \cdot \dot{\phi} \\ |V_r| \cdot \cos(\phi) + R_2 \cdot \dot{\phi} \\ |V_r| \cdot \sin(-30+\phi) + R_3 \cdot \dot{\phi} \end{bmatrix}$$

Then some conversion equations:

$$-\sin(30+\phi) = -\sin(30)\cdot\cos(\phi) - \cos(30)\sin(\phi) = -\frac{1}{2}\cos(\phi) - \frac{\sqrt{3}}{2}\sin)(\phi)$$

$$\sin(-30+\phi) = -\sin(30)\cdot\cos(\phi) + \cos(30)\sin(\phi) = -\frac{1}{2}\cos(\phi) + \frac{\sqrt{3}}{2}\sin)(\phi)$$

$$V_x = |V_r| \cdot \cos\phi$$

$$V_y = |V_r| \cdot \sin\phi$$

So the wheel speeds can be written as:

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} & R_1 \\ 1 & 0 & R_2 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & R_3 \end{bmatrix} \cdot \begin{bmatrix} V_x \\ V_y \\ \dot{phi} \end{bmatrix}$$

And the speeds in local robot coordinates become:

$$\begin{bmatrix} V_x \\ V_y \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & -\frac{\sqrt{3}}{2} & R_1 \\ 1 & 0 & R_2 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & R_3 \end{bmatrix}^{-1} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$$\begin{bmatrix} V_x \\ V_y \\ \dot{\phi} \end{bmatrix} = \frac{1}{R_1+R_2+R_3} \cdot \begin{bmatrix} -R_2 & R_3+R_2 & -R_2 \\ -\frac{1}{\sqrt{3}}(2R_3+R_2) & \frac{1}{\sqrt{3}}(-R_3+R_1) & \frac{1}{\sqrt{3}}(R_2+2R_1) \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Appendix B

Simulations

In the simulation the Turtle follows the given signal perfectly. The idea is to let the Turtle drive 2 meters purely in x direction with a final velocity of zero. The set up is shown in Figure B.1 and the inputs for the trajectory planner in this simulation are in following order:

- desired movement in x direction
- desired final velocity in x direction
- desired movement in y direction
- desired final velocity in y direction
- instantaneous movement in x direction
- instantaneous movement in y direction
- instantaneous velocity in x direction
- instantaneous velocity in y direction
- maximal velocity
- maximal acceleration
- robotID; 2 for a normal Turtle.

The turn flag is set to zero. Outside this simulation the turn flag is set to one when the Turtle is dribbling. The control enable is set to one to enable the parking brake.

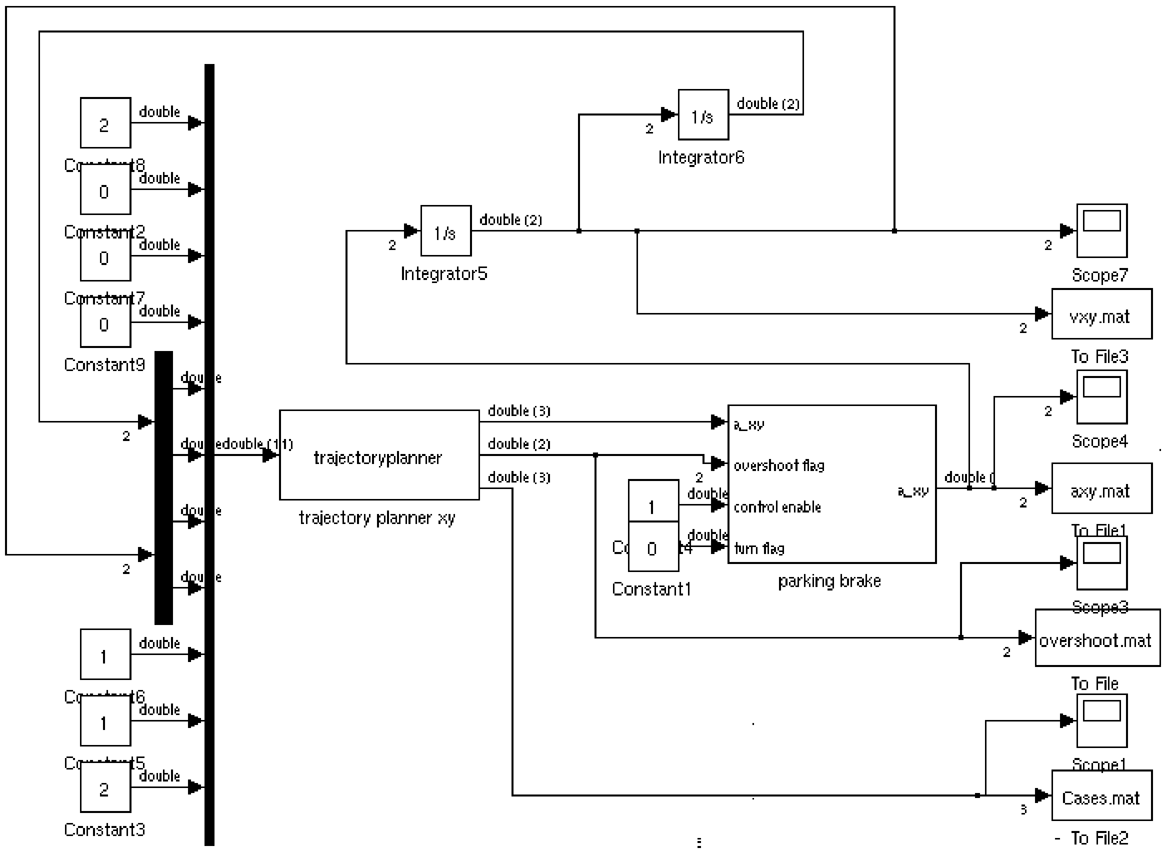


Figure B.1: Set up for simulation 1

For the simulation (simulation 2) where also a movement is introduced in y direction the same set up is used, but with a value for the desired movement in y direction.

For the simulation (simulation 3) where is attempt to simulate the real world a signal builder is used instead of a constant value for the y movement.

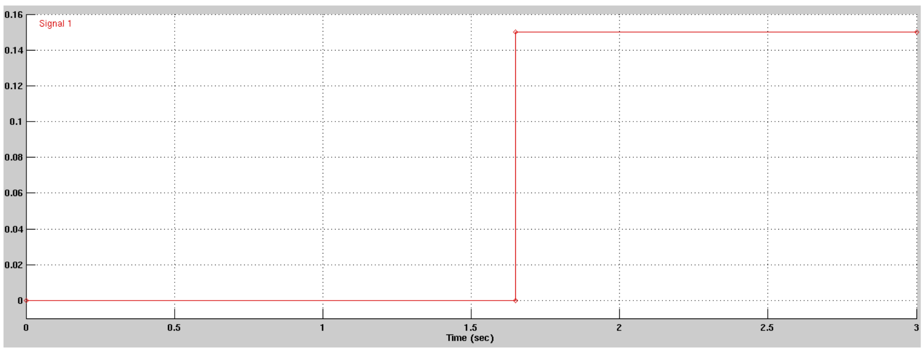


Figure B.2: Signal builder for simulation 3

Appendix C

C file trajectory planner

Listing C.1: C implementation trajectory planner

```
        /*   File      : traj.c
 *   Abstract: Trajectory generator
 *
 */

#define S_FUNCTION_NAME trajectoryplanner
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"
#include <math.h>
#include <stdio.h>
#include "global_par.h"
#include "bus.h"

#define dabs(A)  ((A)>=0?(A):-(A))
#define dmin(A,B)  ((A-B)>=0?(B):(A))
#define dmax(A,B)  ((A-B)>=0?(A):(B))
#define dsgn(A)  ((A)>=0?(1):(-1))

#define EPSILON_POSITION_CONVERGED 0.005 /* sample_time*v_max+0.5*a_max*sample_time*sample_time */
#define EPSILON_VELOCITY_CONVERGED 0.001*2.5 /* sample_time*a_max */
#define EPS 1e-16 /* Machine precision*/

/*====================*
 * S-function methods *
 *====================*/

/* Function: mdlInitializeSizes ===============================================
 * Abstract:
 *    The sizes information is used by Simulink to determine the S-function
 *    block's characteristics (number of inputs, outputs, states, etc.).
 */

static int overshootflag;

static void mdlInitializeSizes(SimStruct *S)
{
    ssSetNumSFcnParams(S, 0);  /* Number of expected parameters */
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch will be reported by Simulink */
    }

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 1)) return;
    ssSetInputPortWidth(S, 0, 11);
    ssSetInputPortDirectFeedThrough(S, 0, 1);
    if (!ssSetNumOutputPorts(S, 2)) return;
    ssSetOutputPortWidth(S, 0, 3);
    ssSetOutputPortWidth(S, 1, 2);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    /* Take care when specifying exception free code - see sfuntmpl_doc.c */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}


/* Function: mdlInitializeSampleTimes =========================================
 * Abstract:
 *    Specifiy that we have a continuous sample time.
 */
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S,0,CONTINUOUS_SAMPLE_TIME);
    ssSetOffsetTime(S,0,0.0);
}

int check_case(double wf, double wodot, double wfdot, double v_max, double a_max, double* q_w,
double* t_accent, double* w_accent, double* wdot_accent){

    double w1dot = 0, t_I, t_II;
    double rev = 1;
    double SAMPLE_TIME = 0.001; /* global variable?! */

    if ( (wf<=0 && wfdot>=0) || (wf<0 && wfdot<0) ){
    /* Case 0: reverse negative cases */
        wf = -wf;
        wodot = -wodot;
        wfdot = -wfdot;
        rev = -rev;
    }

    if( dabs(wf)<=EPSILON_POSITION_CONVERGED && dabs(wfdot-wodot)<EPSILON_VELOCITY_CONVERGED ){
    /* No case: converged! */
        *q_w = dmin(dmax((wfdot-wodot)/SAMPLE_TIME,-a_max),a_max);
        *t_accent = 0;
        *w_accent = 0;
        *wdot_accent = wfdot;
        return 1;
    } else if (wodot>v_max){
    /* Case 5: too large velocity (after target switch, or noise); decelerate */
        *q_w = -a_max;
        *t_accent = (wodot-v_max)/a_max;
        *w_accent = 0.5*(wodot*wodot-v_max*v_max)/a_max;
        *wdot_accent = v_max;
    } else if (dabs(wodot-v_max)<=EPSILON_VELOCITY_CONVERGED &&
    wf>0.5*(wodot*wodot-wfdot*wfdot)/a_max){
    /* Case 3: driving at maximal velocity */
        *q_w = (v_max-wodot)/SAMPLE_TIME;
        *t_accent = wf/v_max+0.5*wfdot*wfdot/v_max/a_max-0.5*v_max/a_max;
        *w_accent = wf+0.5*(wfdot*wfdot-v_max*v_max)/a_max+EPS;
        *wdot_accent = v_max;
    } else if (v_max+EPSILON_VELOCITY_CONVERGED>=wodot && wodot>0 &&
    wf<=0.5*(wodot*wodot-wfdot*wfdot)/a_max){
    /* Case 4: need to decelerate to stop in time */
        *q_w = -a_max;
        *t_accent = (wodot-wfdot)/a_max;
        *w_accent = 0.5*(wodot*wodot-wfdot*wfdot)/a_max;
        *wdot_accent = wfdot;
    } else if (wf>0.5*(wfdot*wfdot-wodot*wodot)/a_max){
    /* Case 2: normal acceleration */
        t_I = (v_max-wodot)/a_max;
        w1dot = sqrt(wf*a_max+0.5*(wodot*wodot+wfdot*wfdot));
        t_II = (w1dot-wodot)/a_max;
        if (t_I<=t_II){
```

```
        /* Case 2.1: accelerate to maximum velocity */
            *q_w = a_max;
            *t_accent = t_I;
            *w_accent = 0.5*(v_max*v_max-wodot*wodot)/a_max;
            *wdot_accent = v_max;
        } else {
        /* Case 2.2: accelerate to w1dot<v_max */
            *q_w = a_max;
            *t_accent = t_II;
            *w_accent = 0.5*wf+0.25*(wfdot*wfdot-wodot*wodot)/a_max+EPS;
            *wdot_accent = w1dot;
        }
    } else if (wf>0.5*(wodot*wodot-wfdot*wfdot)/a_max){
    /* Case aanloopje: distance too small to reach desired final velocity */
        if (wfdot>0){
            w1dot = -1.1*sqrt(-wf*a_max+0.5*(wodot*wodot+wfdot*wfdot));
            *q_w = -a_max;
            *t_accent = (wodot-w1dot)/a_max;
            *w_accent = 0.5*(wodot*wodot-w1dot*w1dot)/a_max;
            *wdot_accent = w1dot;
        } else {
            w1dot = 1.1*sqrt(wf*a_max+0.5*(wodot*wodot+wfdot*wfdot));
            *q_w = a_max;
            *t_accent = (w1dot-wodot)/a_max;
            *w_accent = 0.5*(w1dot*w1dot-wodot*wodot)/a_max;
            *wdot_accent = w1dot;
        }
    } else if(wodot<0){
    /* Case 1: driving in negative (wrong) direction; accelerate */
        *q_w = a_max;
        *t_accent = -wodot/a_max;
        *w_accent = -0.5*wodot*wodot/a_max;
        *wdot_accent = 0;
    } else {
    /* printf("Error\n\n"); */
    }

    /* Compensate for Case 1, reverting */
    *w_accent = rev**w_accent;
    *wdot_accent = rev**wdot_accent;
    *q_w = rev**q_w;

    return 0;
}

double calculate_time(double wf, double wodot, double wfdot, double v_max_w, double a_max_w, double* q_impl){

    double              t = 0;      /* initial time */
    double            q_w = 0;      /* initial position */
    double       t_accent = 0;      /* update time */
    double       w_accent = 0;      /* update position */
    double    wdot_accent = 0;      /* update velocity */
    double            ETA = 0;      /* Estimated Time of Arrival */

    int                 N = 10;
    int                 i = 0;
    int         converged = 0;

    overshootflag = 0;

    for (i=0;i<N;i++){
        converged = check_case(wf,wodot,wfdot,v_max_w,a_max_w,&q_w,&t_accent,&w_accent,&wdot_accent);

        // if i==0 && q_w < 0.0 en je wil naar wf > 0.0 && (if i==1 && !converged)
        if(i==0 && q_w<0.0 && wf > 0.0)
        {
            overshootflag = 1;
        }
        if(i==1 && overshootflag==1 && !converged)
        {
            overshootflag = 3;
        }

        // if i==0 && q_w > 0.0 en je wil naar wf < 0.0 && (if i==1 && !converged)
        if(i==0 && q_w>0.0 && wf < 0.0)
        {
            overshootflag = 2;
        }
        if(i==1 && overshootflag==2 && !converged)
        {
            overshootflag = 3;
        }

        if (i==0){
            *q_impl = q_w;
        }

        /* Update time, position and velocity */
        t = t + t_accent;
        wf = wf - w_accent;
        wodot = wdot_accent;

        if (converged){
            ETA = t;
            break;
        }
    }
    return ETA;
}

/* Function: mdlOutputs =====================================================
 * Abstract:
 *      y = Cx + Du
 */
static void mdlOutputs(SimStruct *S, int_T tid)
{
    double             *y     = ssGetOutputPortRealSignal(S,0);
    real_T             *povershootflag = (real_T *) ssGetOutputPortSignal(S,1);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0); /* x_ref, xdot_ref, y_ref,
    ydot_ref, x, y, xodot, yodot, v_max, a_max, robotID */
    double             xf     = *uPtrs[0]-*uPtrs[4];
    double             xfdot = *uPtrs[1];     /* for the keeper (robot 1), it is assumed that
    xfdot and yfdot are zero! (velocities and accelerations are not adjusted for the keeper
    when end-velocities are considered) */
    double             yf     = *uPtrs[2]-*uPtrs[5];
    double             yfdot = *uPtrs[3];
    double             xodot = *uPtrs[6];
    double             yodot = *uPtrs[7];
    double             v_max = *uPtrs[8];     /* in case of robot 1 (goalkeeper), v_max and
    a_max represent the velocity and acceleration in y-direction */
    double             a_max = *uPtrs[9];
    int               robotID = (int) *uPtrs[10];

    double       SAMPLE_TIME = 0.001;     /* global variable?! */

    double               q_x = 0;
    double             ETA_x = 0;
    double         ETA_x_min = 0;
    double         ETA_x_max = 0;

    double               q_y = 0;
    double             ETA_y = 0;
    double         ETA_y_min = 0;
    double         ETA_y_max = 0;

    static double alpha = 0.25*M_PI;
    double         alpha_min = 0;
    double         alpha_max = 0.5*M_PI;

    double         v_max_x, v_max_y, a_max_x, a_max_y, v_max_d;
    double         SYNC_TIME_PRECISION1 = .001;
    double         SYNC_TIME_PRECISION2 = .001;
    double         EPSILON_VELOCITY_CONVERGED2 = 0.3;

    int                   j = 0;
    int            syncable = 0;

    /* check feasibility of final velocities */
/* this also prevents alpha_min>alpha_max (see below) */
v_max_d = sqrt(xfdot*xfdot+yfdot*yfdot);
if ( v_max_d>0.8*v_max ){
    xfdot = xfdot*(0.8*v_max)/v_max_d;
    yfdot = yfdot*(0.8*v_max)/v_max_d;
}
if (dabs(xfdot)<0.5*EPSILON_VELOCITY_CONVERGED2){
    xfdot = 0;
} else if (xfdot>0){
    xfdot = dmax(xfdot,EPSILON_VELOCITY_CONVERGED2);
} else {
    xfdot = dmin(xfdot,-EPSILON_VELOCITY_CONVERGED2);
}
if (dabs(yfdot)<0.5*EPSILON_VELOCITY_CONVERGED2){
    yfdot = 0;
} else if (yfdot>0){
    yfdot = dmax(yfdot,EPSILON_VELOCITY_CONVERGED2);
} else {
    yfdot = dmin(yfdot,-EPSILON_VELOCITY_CONVERGED2);
}

/* if final velocities are required, alpha_max and/or alpha_min are limited */
if ( dabs(xfdot)>0 ){
    v_max_x = dabs(xfdot);
    alpha_max = acos(v_max_x/v_max);
    v_max_y = sin(alpha_max)*v_max;
    a_max_y = sin(alpha_max)*a_max;
    a_max_x = cos(alpha_max)*a_max;

    ETA_x_max = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);
    povershootflag[0]=overshootflag;
    ETA_y_min = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);
    povershootflag[1]=overshootflag;
}
if ( dabs(yfdot)>0 ){
    v_max_y = dabs(yfdot);
    alpha_min = asin(v_max_y/v_max);
    v_max_x = cos(alpha_min)*v_max;
    a_max_x = cos(alpha_min)*a_max;
    a_max_y = sin(alpha_min)*a_max;

    ETA_x_min = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);
    povershootflag[0]=overshootflag;
    ETA_y_max = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);
    povershootflag[1]=overshootflag;
}
alpha = dmax(alpha,alpha_min);
alpha = dmin(alpha,alpha_max);

/* check whether either one or both directions are converged */
if( (dabs(xf)<=EPSILON_POSITION_CONVERGED && dabs(xfdot-xodot)<EPSILON_VELOCITY_CONVERGED2) &&
(dabs(yf)<EPSILON_POSITION_CONVERGED && dabs(yfdot-yodot)<EPSILON_VELOCITY_CONVERGED2) ){
    q_x = dmin(dmax((xfdot-xodot)/SAMPLE_TIME,-a_max),a_max);
    q_y = dmin(dmax((yfdot-yodot)/SAMPLE_TIME,-a_max),a_max);
    if (dabs(q_x)>0){
        ETA_x = (xfdot-xodot)/q_x;
    } else {
        ETA_x = 0;
    }
    if (dabs(q_y)>0){
        ETA_y = (yfdot-yodot)/q_y;
    } else {
        ETA_y = 0;
    }
    y[0] = q_x;
    y[1] = q_y;
    y[2] = dmax(ETA_x,ETA_y);

} else if ( dabs(xf)<=EPSILON_POSITION_CONVERGED && dabs(xfdot-xodot)<EPSILON_VELOCITY_CONVERGED2 ){
    q_x = dmin(dmax((xfdot-xodot)/SAMPLE_TIME,-a_max),a_max);
    ETA_y = calculate_time(yf,yodot,yfdot,v_max,a_max,&q_y);
    povershootflag[1]=overshootflag;
    y[0] = q_x;
    y[1] = q_y;
```

```
        y[2] = ETA_y;

    } else if ( dabs(yf)<=EPSILON_POSITION_CONVERGED && dabs(yfdot-yodot)<EPSILON_VELOCITY_CONVERGED2 ){
        q_y = dmin(dmax((yfdot-yodot)/SAMPLE_TIME,-a_max),a_max);
        ETA_x = calculate_time(xf,xodot,xfdot,v_max,a_max,&q_x);
        povershootflag[0]=overshootflag;
        y[0] = q_x;
        y[1] = q_y;
        y[2] = ETA_x;

    } else { /* no direction converged yet: synchronize if possible */

        /* check sync'ability of situation */
        if (robotID == 1){
            syncable = 0;           /* do not sync in case of robot 1*/

            ETA_x = calculate_time(xf,xodot,xfdot,v_max,a_max,&q_x);
            povershootflag[0]=overshootflag;
            ETA_y = calculate_time(yf,yodot,yfdot,v_max,a_max,&q_y);
            povershootflag[1]=overshootflag;
            y[0] = q_x;
            y[1] = q_y;
            y[2] = dmax(ETA_x,ETA_y);

        } else if ( dabs(xfdot)>0 && dabs(yfdot)>0 ){
            if ( ETA_y_min<=ETA_x_max && ETA_x_min<=ETA_y_max ){
                syncable = 1;       /* sync'able! */
            } else {
                syncable = 0;       /* non sync'able (yet) */
                if ( ETA_y_min>ETA_x_max ){
                    alpha = alpha_max;
                    v_max_y = sin(alpha)*v_max;
                    a_max_y = sin(alpha)*a_max;
                    ETA_y_min = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);
                    povershootflag[1]=overshootflag;
                    y[0] = 0;
                    y[1] = q_y;
                    y[2] = ETA_y_min;
                } else if ( ETA_x_min>ETA_y_max ){
                    alpha = alpha_min;
                    v_max_x = cos(alpha_min)*v_max;
                    a_max_x = cos(alpha_min)*a_max;
                    ETA_x_min = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);
                    povershootflag[0]=overshootflag;
                    y[0] = q_x;
                    y[1] = 0;
                    y[2] = ETA_x_min;
                } else {
                    /* printf("This should not be possible! ETA_y_min = %f, ETA_x_max = %f,
                     ETA_x_min = %f, ETA_y_max = %f\n",ETA_y_min,ETA_x_max,ETA_x_min,ETA_y_max); */
                }
            }
        } else if ( dabs(xfdot)>0 ){
            if (ETA_y_min<=ETA_x_max){
                syncable = 1;       /* sync'able! */
            } else {
                syncable = 0;       /* non sync'able (yet) */
                alpha = alpha_max;
                v_max_y = sin(alpha)*v_max;
                a_max_y = sin(alpha)*a_max;
                ETA_y_min = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);
                povershootflag[1]=overshootflag;
                y[0] = 0;
                y[1] = q_y;
                y[2] = ETA_y_min;
            }
        } else if ( dabs(yfdot)>0 ){
            if (ETA_x_min<=ETA_y_max){
                syncable = 1;       /* sync'able! */
            } else {
                syncable = 0;       /* non sync'able (yet) */
                alpha = alpha_min;
                v_max_x = cos(alpha_min)*v_max;
                a_max_x = cos(alpha_min)*a_max;
                ETA_x_min = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);
                povershootflag[0]=overshootflag;
                y[0] = q_x;
                y[1] = 0;
                y[2] = ETA_x_min;
            }
        } else {                       /* no final velocities */
            syncable = 1;              /* sync'able! */
        }

        /* if sync'able, check previously derived (static) alpha */
        if ( syncable ){
            v_max_x = cos(alpha)*v_max;
            a_max_x = cos(alpha)*a_max;
            v_max_y = sin(alpha)*v_max;
            a_max_y = sin(alpha)*a_max;

            ETA_x = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);
            povershootflag[0]=overshootflag;
            ETA_y = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);
            povershootflag[1]=overshootflag;

            if ( dabs(ETA_x-ETA_y)>SYNC_TIME_PRECISION2 ){
                /* (renew initial guess alpha omitted / not implemented) */
                while ( dabs(ETA_x-ETA_y)>SYNC_TIME_PRECISION1 ){
                    if (j==0){
                        /* printf("need to iterate! xf=%f, yf=%f, xodot=%f, yodot=%f, alpha_min=%f,
                        alpha_max=%f, alpha=%f, ETA_x=%f, ETA_y=%f\n",xf,yf,xodot,yodot,alpha_min,
                        alpha_max,alpha,ETA_x,ETA_y); */
                    }
                    if ( ETA_x<ETA_y ){
                        alpha_min = alpha;
                        alpha = 0.5*alpha+0.5*alpha_max;
                    } else{
                        alpha_max = alpha;
                        alpha = 0.5*alpha+0.5*alpha_min;
                    }

                    v_max_x = cos(alpha)*v_max;
                    a_max_x = cos(alpha)*a_max;
                    v_max_y = sin(alpha)*v_max;
                    a_max_y = sin(alpha)*a_max;

                    ETA_x = calculate_time(xf,xodot,xfdot,v_max_x,a_max_x,&q_x);
                    povershootflag[0]=overshootflag;
                    ETA_y = calculate_time(yf,yodot,yfdot,v_max_y,a_max_y,&q_y);
                    povershootflag[1]=overshootflag;

                    j++;
                    /* printf("   alpha = %f, ETA_x=%f, ETA_y=%f\n",alpha,ETA_x,ETA_y); */
                    if (j>50){
                        /* printf("   bi-section algorithm not converged!\n"); */
                        break;
                    }
                }
            /* } else { */
            /*     printf("previous alpha was ok\n"); */
            }

            y[0] = q_x;
            y[1] = q_y;
            y[2] = 0.5*(ETA_x+ETA_y);
        }
    }
}


/* Function: mdlTerminate =====================================================
 * Abstract:
 *    No termination needed, but we are required to have this routine.
 */
static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S); /* unused input argument */
```

```
}

#ifdef  MATLAB_MEX_FILE    /* Is this file being compiled as a MEX-file? */
#include "simulink.c"      /* MEX-file interface mechanism */
#else
#include "cg_sfun.h"       /* Code generation registration function */
#endif
```

Appendix D

Acceleration x direction Chapter 5

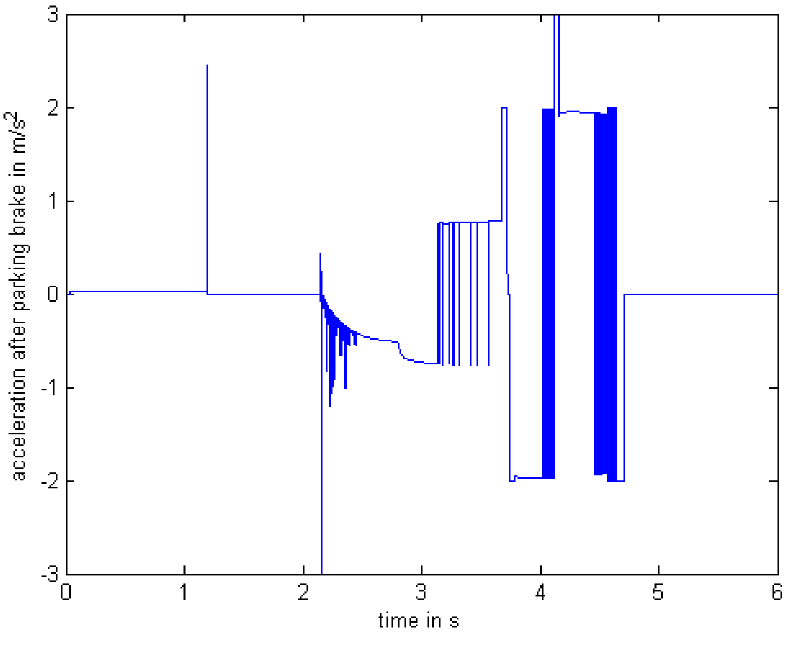


Figure D.1: Acceleration after the parking brake in x direction

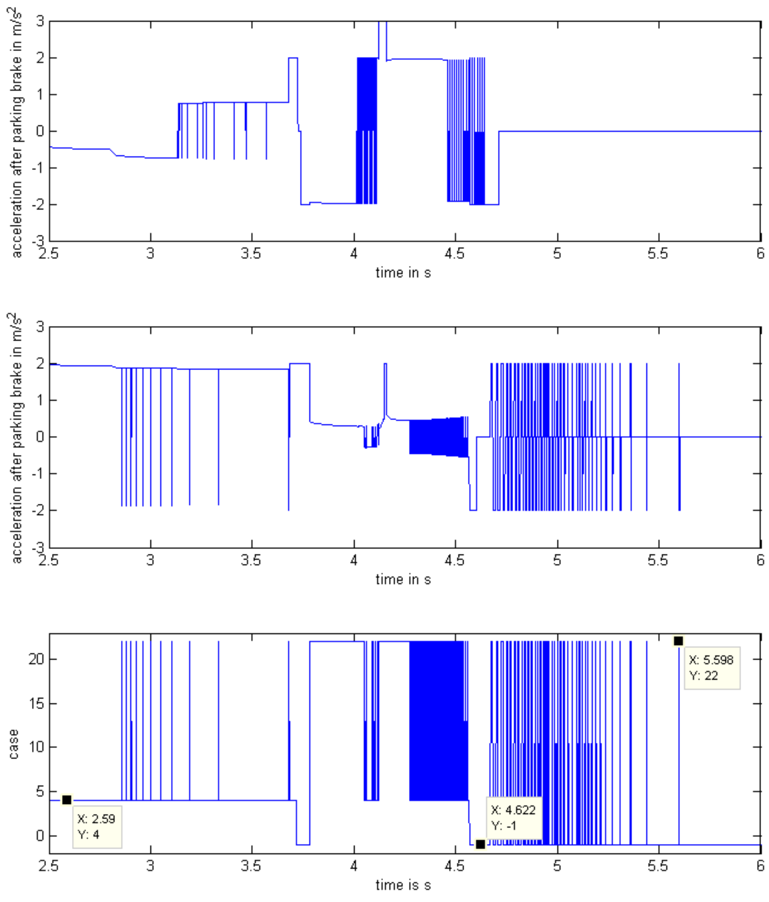


Figure D.2: A zoom of the x- and y acceleration and the case for the y direction on time frame[2.5 - 6].

Appendix E

Set up for simulations traction controller

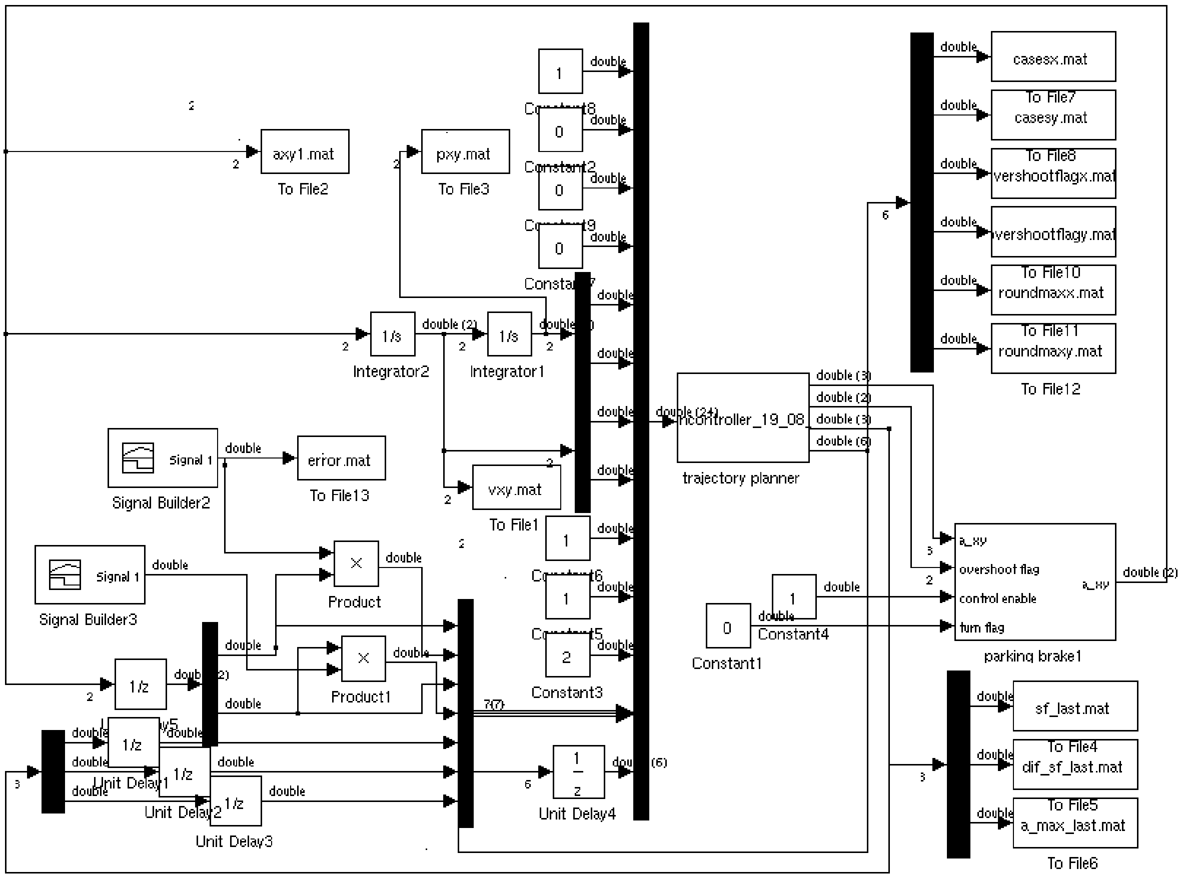


Figure E.1: The set up for the simulation of the traction controller.

Now the inputs are the same as in Appendix A, but there are 13 inputs more:

- The instantaneous input acceleration in x direction.
- The measured acceleration in x position
- The instantaneous input acceleration in y direction.
- The measured acceleration in y position
- The safety factor of previous time step
- The difference in safety factor of previous time step
- The maximal acceleration of previous time step
- the cases in x direction
- the cases in y direction
- the overshootflag in x direction
- the overshootflag in y direction
- the roundmax in x direction
- the roundmax in y direction

With the signal builders there is an error imposed in the signal. The signal builders are initial 1, but for some time frames they have a value above 1.