

FH JOANNEUM
GRAZ

Model Based Design

Balanbot

Training Unit 05

Authors

David B. Heer

Jakob Soukup

Graz, February 3, 2019

Lecturer

Alfred Steinhuber

Contents

I	Laboratory Session 06	2
1	Description of the Model	3
1.1	Implementation in simulink	4
2	Discretization from non-linear model	5
2.1	Applying zero force to the system	6
3	Linearization	7
4	Discretization linear model	10
4.1	Forward Euler	10
4.2	Backward Euler	10
4.3	Trapezoidal or Tustin	10
4.4	Discretizing using Matlab	10
5	System analysis	11
6	Control function	15
II	Laboratory Session 07	19
7	Introduction	19
8	System Initialization	19
9	Read Sensor Datas	20
9.1	Angle	20
9.2	Gyro	20
10	Controller	21
11	Actuators	21
12	Whole Structure	22
13	Conclusion	22
A	Appendix	24

Contents	2
----------	---

List of Figures	43
-----------------	----

Part I

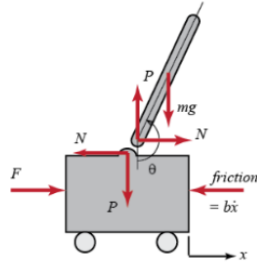
Laboratory Session 06

Introduction

In this laboratory unit the model of an inverse pendulum on a moving cart will be implemented and simulated in simulink. In a first step the non linear model will be implemented and then discretized. After that the non linear model shall be linearized and discretized again. The differences between the two models are to be investigated. The two models shall be controlled with a PID controller. If the simulation works the model shall be deployed onto an actual moving robot to see if it holds up in real life.

1 Description of the Model

The model consists of a moving part with a hinged pendulum atop. The goal for the controller is to accelerate the cart in the right direction depending on the angle of the pendulum in order to keep it upright at all times.



Where:

x : cart's position	b : coefficient of friction for cart
\dot{x} : cart's velocity	l : length to pendulum center of mass
\ddot{x} : cart's acceleration	J : moment of inertia of the pendulum
θ : pendulum's position (angle)	F : external force applied (by motors)
$\dot{\theta}$: angular velocity	N : interaction force between cart and pendulum in x direction
$\ddot{\theta}$: angular acceleration	P : interaction force between cart and pendulum in y direction
m : mass of pendulum	
M : mass of cart	
g : gravitational constant	

Figure 1: graphical description of the model

The equations of the model are given by:

$$\ddot{x} = \frac{1}{M} \sum_{cart} F_x = \frac{1}{M} (F - N - b\dot{x}) \quad (1)$$

$$\ddot{\Theta} = \frac{1}{I} \sum_{pend} \tau = \frac{1}{I} (-Nl\cos\Theta - Pl\sin\Theta) \quad (2)$$

$$N = m(\ddot{x} - l\dot{\Theta}^2\sin\Theta + l\ddot{\Theta}\cos\Theta) \quad (3)$$

$$P = m(l\ddot{\Theta}\cos\Theta + l\dot{\Theta}^2\sin\Theta) \quad (4)$$

1.1 Implementation in simulink

The non linear model can be implemented using the equations above, this was already done in a previous lecture in the third semester. The resulting model can be seen in Figure 5.

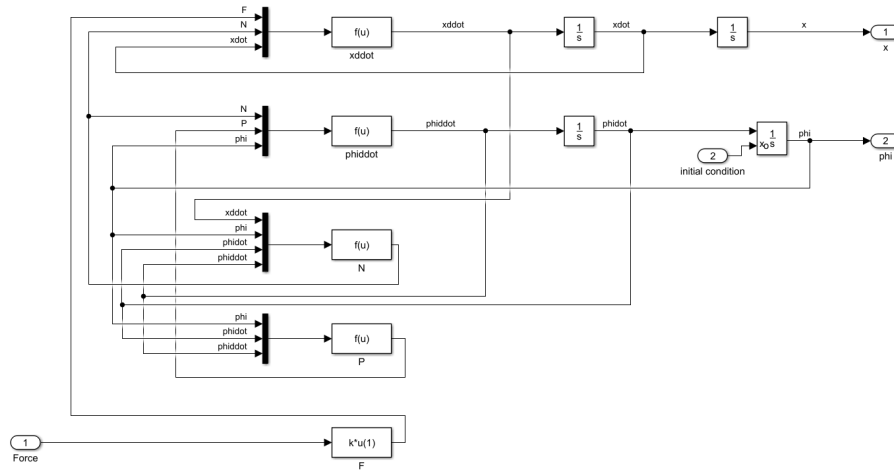


Figure 2: Non linear continuous model in simulink

2 Discretization from non-linear model

Since the model will later be used on an actual hardware, it is important to discretize the system. This is done by simply replacing the continuous time integrators with discrete time integrators. The settings of the integrators are shown in Figure 3.

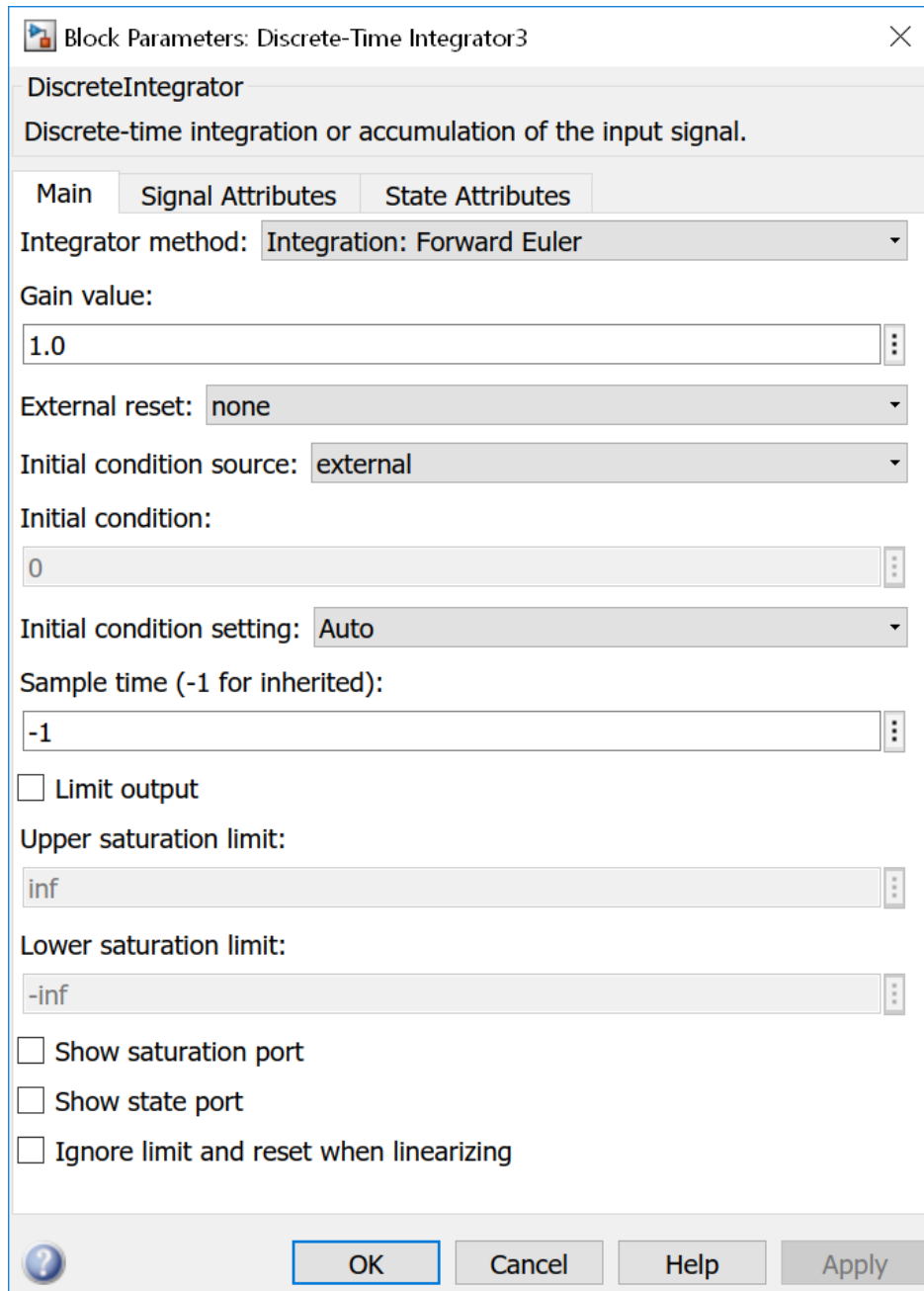


Figure 3: discrete time integrator settings

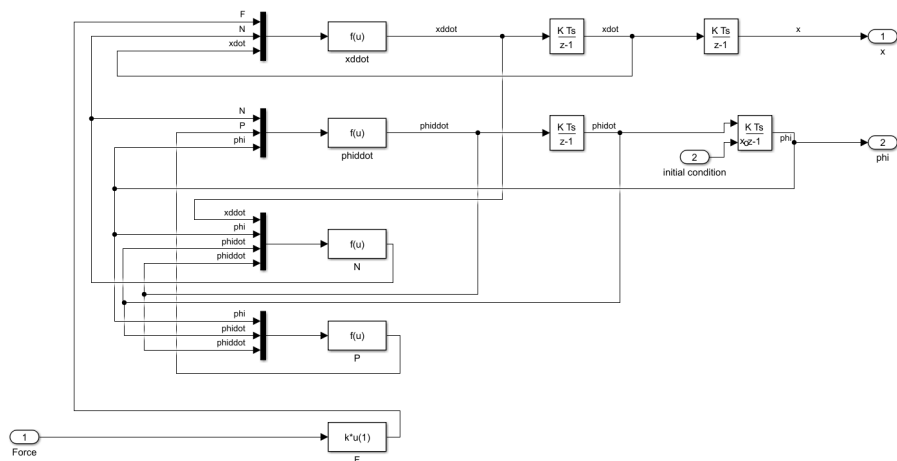


Figure 4: Non linear discrete model in simulink

2.1 Applying zero force to the system

As a first test the model was tested with a constant of zero at its input. It would be expected to do nothing but stay upright since there are no external forces applied to the pendulum in the horizontal axis.

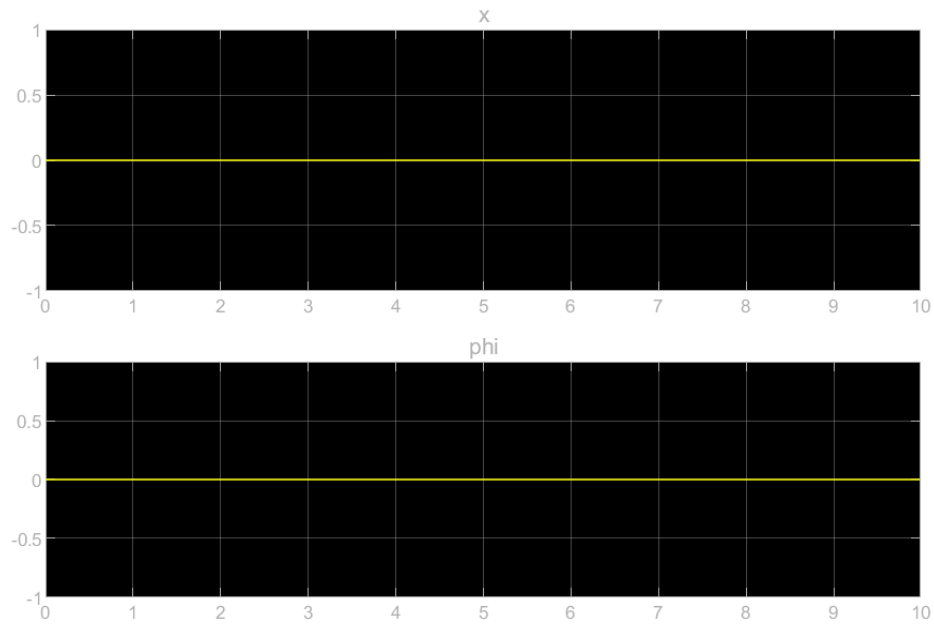


Figure 5: Non linear discrete model simulation with zero force applied

As a small test an initial step was applied to the system to see if it behaves correctly. The disturbance of the angle was accomplished by using the step function of simulink with an initial

value of $10 \cdot \frac{\pi}{180}$, which is 10° in radians. As shown in Figure 6 the pendulum swings left and right and slowly loses height, so the model seems to be behaving correctly.

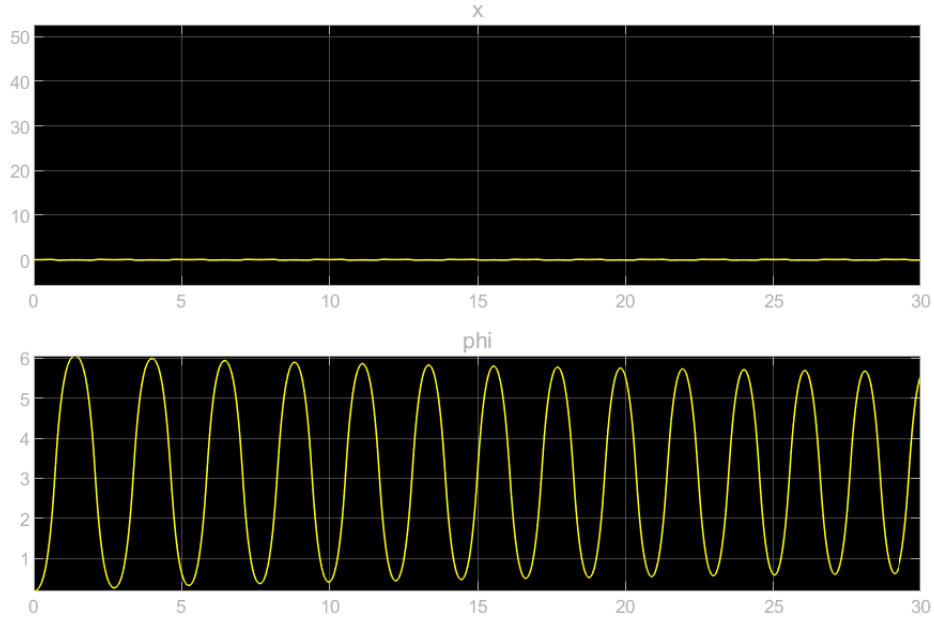


Figure 6: Non linear discrete model simulation with an offset step in the angle

3 Linearization

In order to further investigate the system for stability to make tuning the controller easier, it is mandatory to linearize the model. This is done by assuming that the slope of a sine wave is linear which of course is not the case but it is a valid approximation. The linearized equations are given by

$$X(s) s^2 = \frac{1}{M} (F(s) - mX(s) s^2 + ml\Phi(s) s^2 - bX(s) s) \quad (5)$$

$$\Phi(s) s^2 = \frac{1}{I} (mlX(s) s^2 + mlg\Phi(s) - ml^2\Phi(s) s^2) \quad (6)$$

From these equations the two transfer functions $G_1(s) = \frac{\Phi(s)}{F(s)}$ and $G_2(s) = \frac{X(s)}{F(s)}$ are to be found. This is simply a fact of rearranging the equations. $G_1(s)$ Solving equation 5 for $X(s)$:

$$X(s) s^2 M = F(s) - mX(s) s^2 + ml\Phi(s) s^2 - bX(s)s \quad (7)$$

$$X(s) s^2 M = F(s) + ml\Phi(s) s^2 + X(s) [-ms^2 - bs] \quad (8)$$

$$X(s) s^2 M - X(s) [-ms^2 - bs] = F(s) + ml\Phi(s) s^2 \quad (9)$$

$$X(s) [s^2 M + ms^2 + bs] = F(s) + ml\Phi(s) s^2 \quad (10)$$

$$X(s) = \frac{F(s) + ml\Phi(s) s^2}{s^2 M + ms^2 + bs} \quad (11)$$

Inserting into equation 6 we get

$$\Phi(s) s^2 = \frac{1}{I} \left[mls^2 \cdot \frac{F(s) + ml\Phi(s) s^2}{s^2 M + ms^2 + bs} + mlg\Phi(s) - ml^2\Phi(s) s^2 \right] \quad (12)$$

$$\Phi(s) Is^2 = mls^2 \cdot \frac{F(s) + ml\Phi(s) s^2}{s^2 M + ms^2 + bs} + mlg\Phi(s) - ml^2\Phi(s) s^2 \quad (13)$$

$$Is^2 = mls^2 \cdot \frac{\frac{F(s)}{\Phi(s)} + mls^2}{Ms^2 + ms^2 + bs} + mlg - ml^2 s^2 \quad (14)$$

$$Is^2 = mls^2 \cdot \frac{\frac{F(s)}{\Phi(s)} + mls^2}{Ms^2 + ms^2 + bs} + mlg - ml^2 s^2 \quad (15)$$

$$\frac{Is^2}{ml} = s^2 \cdot \frac{\frac{F(s)}{\Phi(s)} + mls^2}{Ms^2 + ms^2 + bs} + g - ls^2 \quad (16)$$

$$\frac{Is^2}{ml} = \frac{\frac{F(s)}{\Phi(s)} + mls^2}{M + m + \frac{b}{s}} + g - ls^2 \quad (17)$$

$$\frac{F(s)}{\Phi(s)} = \left[\frac{I_s^2}{ml} - g + ls^2 \right] \left[M + m + \frac{b}{s} \right] - mls^2 \quad (18)$$

$$\frac{F(s)}{\Phi(s)} = \frac{MI}{ml} s^2 + \frac{I}{l} s^2 + \frac{Ib}{ml} s - gM - gm - \frac{gb}{s} + Mls^2 + mls^2 - mls^2 \quad (19)$$

$$\frac{F(s)}{\Phi(s)} = s^2 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s \left(\frac{Ib}{ml} + lb \right) - gM - gm - \frac{gb}{s} \quad (20)$$

$$\frac{\Phi(s)}{F(s)} = \frac{1}{s^2 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s \left(\frac{Ib}{ml} + lb \right) - gM - gm - \frac{gb}{s}} \quad (21)$$

$$\frac{\Phi(s)}{F(s)} = \frac{s}{s^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s^2 \left(\frac{Ib}{ml} + lb \right) + s(-gM - gm) - gb} \quad (22)$$

$$\frac{\Phi(s)}{F(s)} = \frac{s}{s^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s^2 \left(\frac{Ib}{ml} + lb \right) + s(-gM - gm) - gb} \quad (23)$$

$G_2(s)$ Solving equation 6 for $\Phi(s)$:

$$\Phi(s) s^2 = \frac{1}{I} (mlX(s) s^2 + mlg\Phi(s) - ml^2\Phi(s) s^2) \quad (24)$$

$$I\Phi(s) s^2 = mlX(s) s^2 + mlg\Phi(s) - ml^2\Phi(s) s^2 \quad (25)$$

$$\Phi(s) = \frac{mlX(s) s^2}{Is^2 + ml^2s^2 - mlg} \quad (26)$$

Inserting into the previously calculated transfer function:

$$\frac{\frac{mlX(s)s^2}{Is^2+ml^2s^2-mlg}}{F(s)} = \frac{s}{s^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s^2 \left(\frac{Ib}{ml} + lb \right) + s(-gM - gm) - gb} \quad (27)$$

$$\frac{X(s)}{F(s)} = \frac{Is^2 + ml^2s^2 - mlg}{mls^2} \cdot \frac{s}{s^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s^2 \left(\frac{Ib}{ml} + lb \right) + s(-gM - gm) - gb} \quad (28)$$

$$\frac{X(s)}{F(s)} = \frac{Is^2 + ml^2s^2 - mlg}{mls \cdot \left[s^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s^2 \left(\frac{Ib}{ml} + lb \right) + s(-gM - gm) - gb \right]} \quad (29)$$

$$\frac{X(s)}{F(s)} = \frac{s^2(I + ml^2) - mlg}{s^4(MI + Im + Mml^2) + s^3(Ib + mbl^2) + s^2(-gml[M + m]) - s(gbml)} \quad (30)$$

$$(31)$$

Our two transfer functions are therefore

$$G_1(s) = \frac{\Phi(s)}{F(s)} = \frac{s}{s^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml \right) + s^2 \left(\frac{Ib}{ml} + lb \right) + s(-gM - gm) - gb} \quad (32)$$

$$G_2(s) = \frac{X(s)}{F(s)} = \frac{s^2(I + ml^2) - mlg}{s^4(MI + Im + Mml^2) + s^3(Ib + mbl^2) + s^2(-gml[M + m]) - s(gbml)} \quad (33)$$

To validate the calculations, the results were compared to the ones yielded in the online documentation¹. The poles and zeros matched and therefore it can be assumed that the calculations are correct.

¹<http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum§ion=SystemModeling>

4 Discretization linear model

4.1 Forward Euler

$$z = e^{sT} \approx 1 + sT \rightarrow s \approx \frac{z-1}{T} \quad (34)$$

$$G_1(z) \approx \frac{\frac{z-1}{T}}{\left(\frac{z-1}{T}\right)^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml\right) + \left(\frac{z-1}{T}\right)^2 \left(\frac{Ib}{ml} + lb\right) + \frac{z-1}{T}(-gM - gm) - gb} \quad (35)$$

$$G_2(z) \approx \frac{\left(\frac{z-1}{T}\right)^2 (I + ml^2) - mlg}{\left(\frac{z-1}{T}\right)^4 (MI + Im + Mml^2) + \left(\frac{z-1}{T}\right)^3 (Ib + mbl^2) + \left(\frac{z-1}{T}\right)^2 (-gml[M + m]) - \frac{z-1}{T} (gbml)} \quad (36)$$

4.2 Backward Euler

$$z = e^{sT} \approx \frac{1}{1 + sT} \rightarrow s \approx \frac{z-1}{Tz} \quad (37)$$

$$G_1(z) \approx \frac{\frac{z-1}{Tz}}{\left(\frac{z-1}{Tz}\right)^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml\right) + \left(\frac{z-1}{Tz}\right)^2 \left(\frac{Ib}{ml} + lb\right) + \frac{z-1}{Tz}(-gM - gm) - gb} \quad (38)$$

$$G_2(z) \approx \frac{\left(\frac{z-1}{Tz}\right)^2 (I + ml^2) - mlg}{\left(\frac{z-1}{Tz}\right)^4 (MI + Im + Mml^2) + s^3 (Ib + mbl^2) + \left(\frac{z-1}{Tz}\right)^2 (-gml[M + m]) - \frac{z-1}{Tz} (gbml)} \quad (39)$$

4.3 Trapezoidal or Tustin

$$z = e^{sT} \approx \frac{1 + sT/2}{1 - sT/2} \rightarrow s \approx \frac{2(z-1)}{T(z+1)} \quad (40)$$

$$G_1(z) \approx \frac{\frac{2(z-1)}{T(z+1)}}{\left(\frac{2(z-1)}{T(z+1)}\right)^3 \left(\frac{MI}{ml} + \frac{I}{l} + Ml\right) + \left(\frac{2(z-1)}{T(z+1)}\right)^2 \left(\frac{Ib}{ml} + lb\right) + \frac{2(z-1)}{T(z+1)}(-gM - gm) - gb} \quad (41)$$

$$G_2(z) \approx \frac{\left(\frac{2(z-1)}{T(z+1)}\right)^2 (I + ml^2) - mlg}{\left(\frac{2(z-1)}{T(z+1)}\right)^4 (MI + Im + Mml^2) + \left(\frac{2(z-1)}{T(z+1)}\right)^3 (Ib + mbl^2) + \left(\frac{2(z-1)}{T(z+1)}\right)^2 (-gml[M + m]) - \frac{2(z-1)}{T(z+1)} (gbml)} \quad (42)$$

4.4 Discretizing using Matlab

Matlab has a built in function `c2d()` that can discretize a continuous time transfer function. It only requires the transfer function and the sample time as an input. We using 0.0001 seconds as the sampling time. The Matlab code is shown below:

```

1 cart_n2 = (I+m*l^2)/q;
2 cart_n1 = 0;
3 cart_n0 = -g*m*l/q;
4 cart_d4 = 1;
5 cart_d3 = b*(I+m*l^2)/q;
6 cart_d2 = ( (M + m)*m*g*l)/q;
7 cart_d1 = - b*m*g*l/q;
8 cart_d0 = 0;
9
10 pend_n1 = m*l/q;
11 pend_n0 = 0;
12 pend_d3 = 1;
13 pend_d2 = (b*(I + m*l^2))/q;
14 pend_d1 = -( (M + m)*m*g*l)/q;
15 pend_d0 = -b*m*g*l/q;
16
17 P_cart = (cart_n2*s^2 + cart_n1*s + cart_n0)/(cart_d4*s^4 + cart_d3*s^3 + cart_d2*s
    ^2 + cart_d1*s + cart_d0)
18 P_pend = (pend_n1*s + pend_n0)/(pend_d3*s^3 + pend_d2*s^2 + pend_d1*s + pend_d0)
19
20
21 %% discretizing the transfer functions
22 d_P_cart = c2d(P_cart, 0.0001)
23 d_P_pend = c2d(P_pend, 0.0001)

```

This script puts out the discrete transfer function

$$\frac{2.273 \cdot 10^{-8} z^2 - 1.377 \cdot 10^{-13} z - 2.273 \cdot 10^{-8}}{z^3 - 3z^2 + 3z - 1} \quad (43)$$

[TODO - put in both equations]

5 System analysis

As a next step the transfer function of the systems shall be analysed using Matlab. This can be done using the command `pzmap()`.

```

1 %Plotting poles and zeros
2 figure
3 pzmap(P_cart, P_pend)
4 legend('cart', 'pendulum');
5
6 figure
7 pzmap(d_P_cart, d_P_pend)
8 legend('cart', 'pendulum');

```

This results in the following two figures.

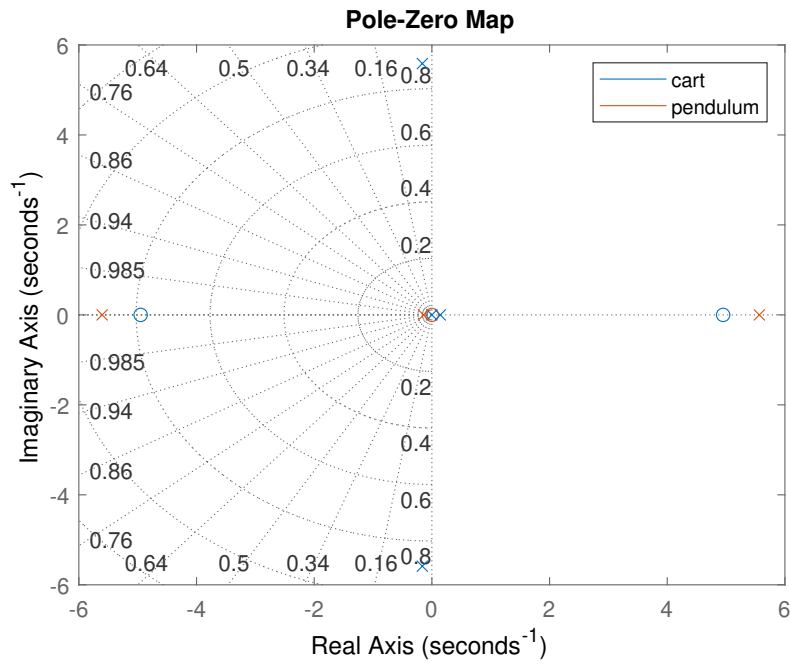


Figure 7: pole and zero map of the continuous systems

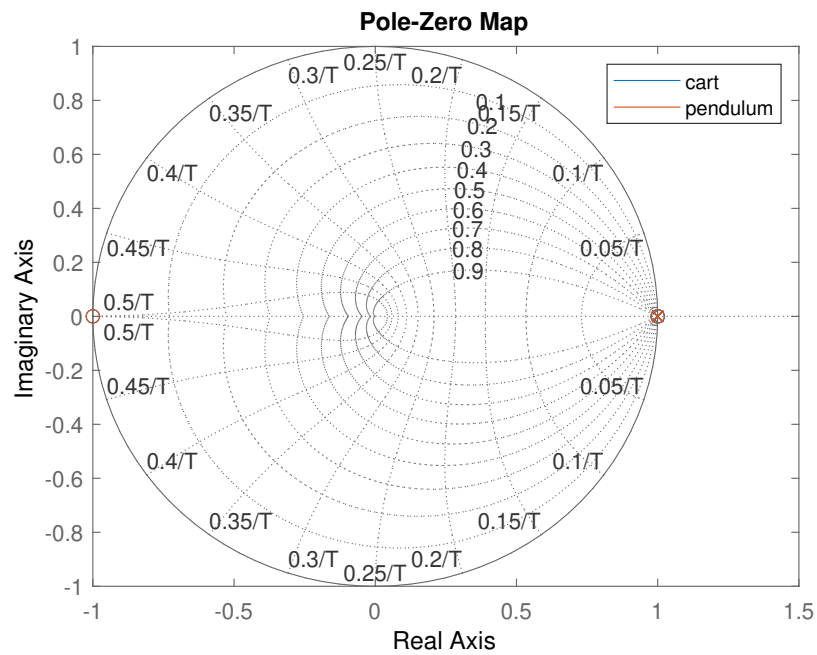


Figure 8: pole and zero map of the discrete systems

As figure 7 and 8 show both the cart and the pendulum are unstable no matter if discretized or not. For the continuous system this can be detected because the poles do not all have a negative imaginary part. Looking at the discrete system at first glance it looks like all poles are within or at last at the unit circle. However after zooming in (see Figure 9) it appears that a pole is outside of the unit circle which results in an unstable system.

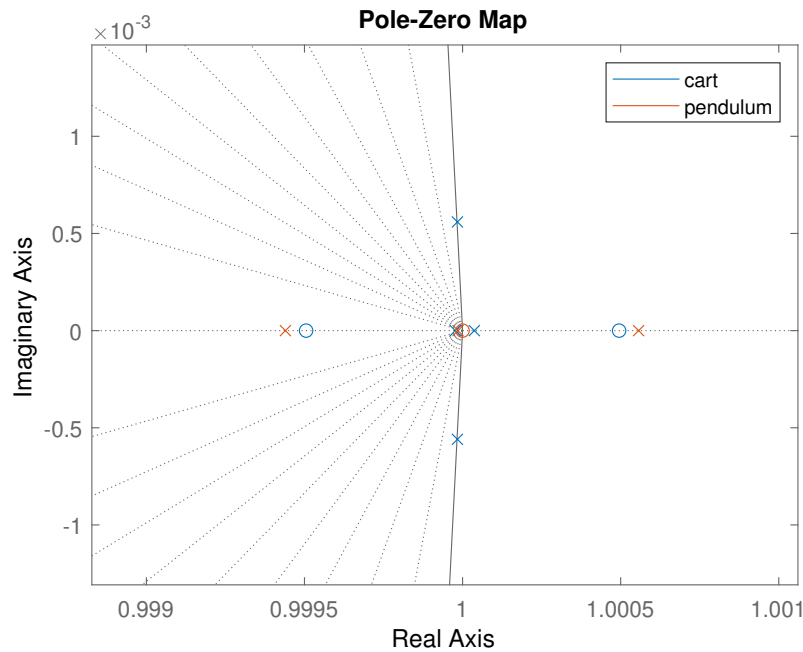


Figure 9: zoomed in pole and zero map of the discrete systems

Next the both the discrete and the continuous model of the linearized model were implemented in simulink and tested next to each other. In order to get the denominator and the numerator of the transfer function for simulink, Matlabs *tfdata* function was used.

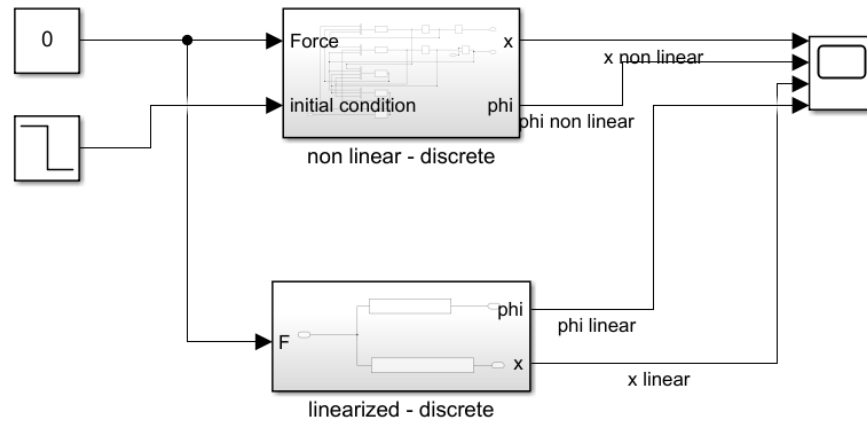


Figure 10: linear and non linear version of the discrete system

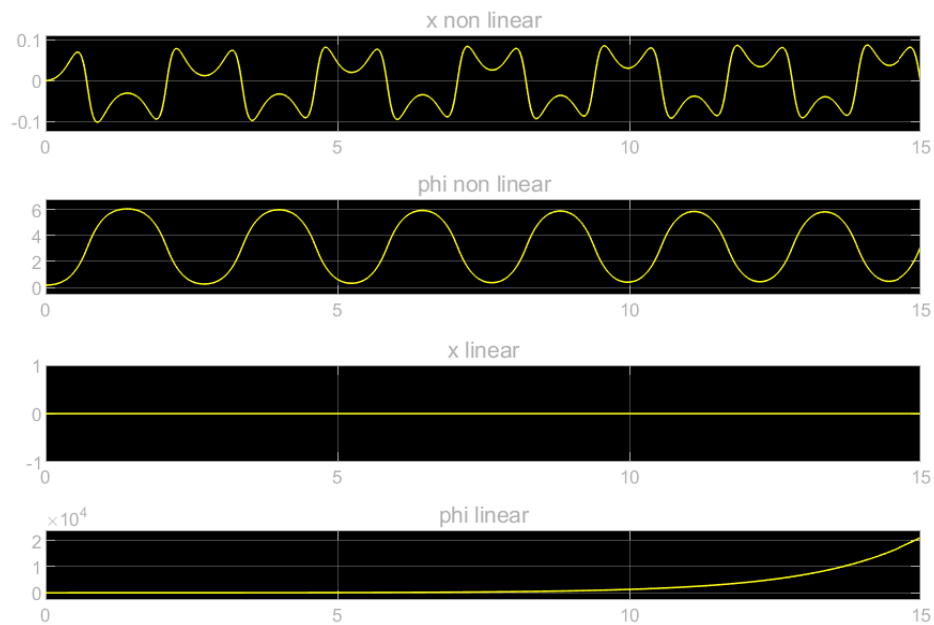


Figure 11: simulation result of linear and non linear version of the discrete system

As figure 11 shows, the non linear system behaves the way it should. The pendulum starts swinging and slowly decreases in height. Due to the inertia of the system the cart moves a bit. At first sight the linearized model looks to be wrong. However, this is not the case as it shows an exponential function which is the solution for the differential equation we're trying to solve. the linearized model only works with small deviations and small time slots, so in order for it to behave correctly we need to implement a controller that gets executed regularly.

6 Control function

Now that we have a working model of the pendulum we are going to have to control the cart in such a way, that it always keeps the pendulum upwards. For this purpose two models are going to be developed: one using the continuous plant and the other using the discrete one. The controller we'll be using is a simply PID controller that simulink offers. In addition a Kalman filter will be implemented in order to provide a more plausible vertical position coming from the plant. Firstly, the provided Kalman filter was implemented using the Matlab function block and to test it the following model was built and simulated:

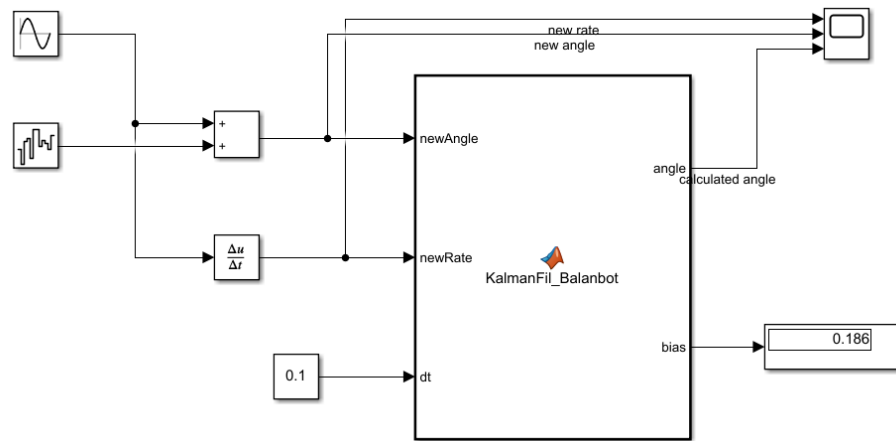


Figure 12: Kalman filter test model

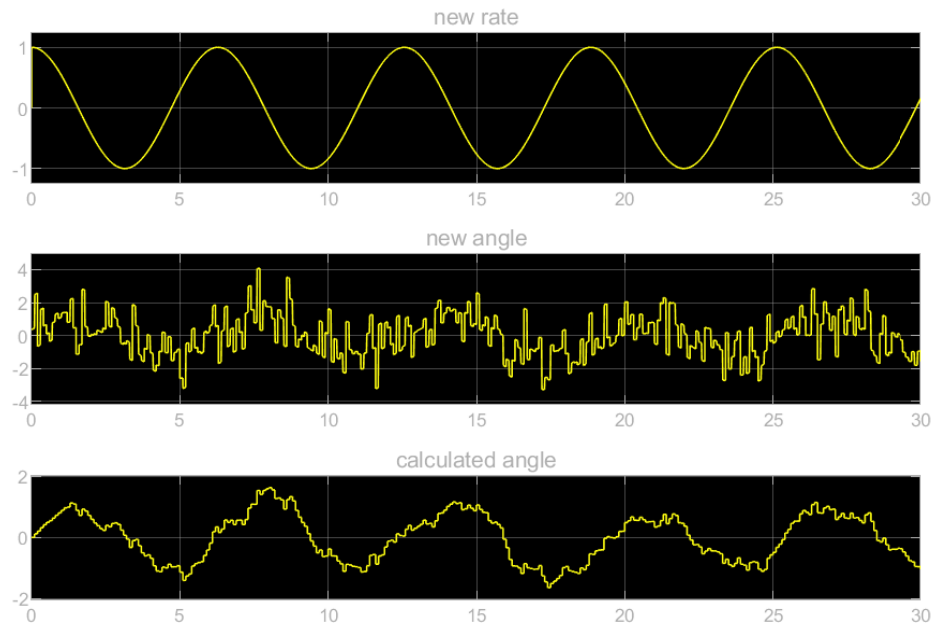


Figure 13: Kalman filter test results

As Figure 12 shows, the filter works quite well, so we can move onto implementing the PID controllers. Now two simulink models were developed. One with the discrete plant and one with the continuous one. The controller was in both cases discrete. For simulation purposes the Kalman filter was implemented using the deviation of the angle in order to gain the angular velocity.

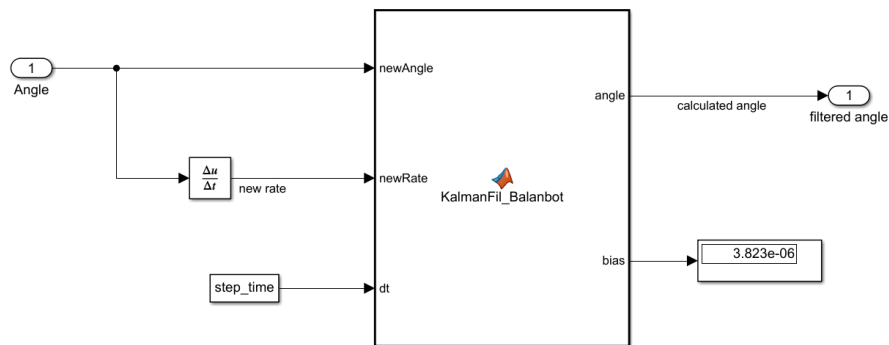


Figure 14: Kalman filter for simulation

Now the continuous system was built and simulated using a variable step solver.

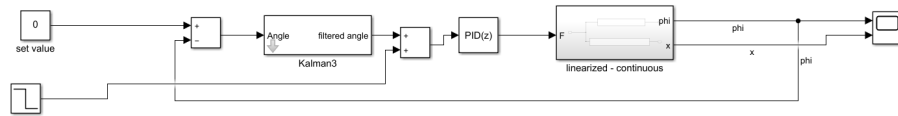


Figure 15: linearized continuous system simulation setup

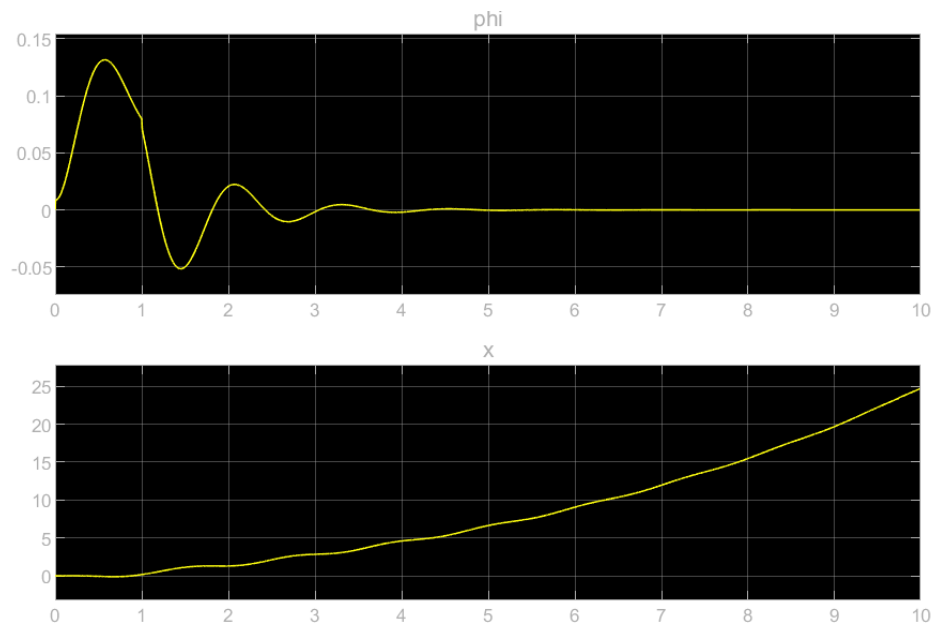


Figure 16: linearized continuous system simulation results

After that the same was done for the discrete system. This time a fixed step discrete solver was used.

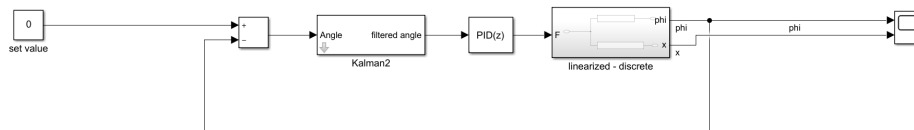


Figure 17: linearized discrete system simulation setup

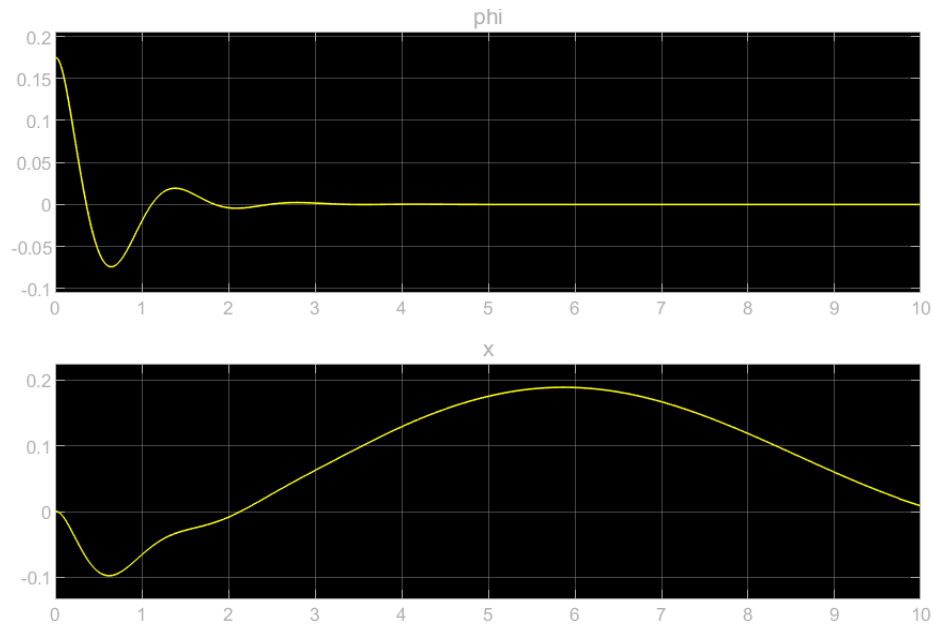


Figure 18: *linearized discrete system simulation results*

In both cases the models seem to behave the way they should, the controllers also seem to be working well so we can move on to the next step which is deploying the whole thing onto the actual hardware.

Part II

Laboratory Session 07

7 Introduction

In this session everything was prepared to put the BalanBot into operation. For this purpose, a block was created for initialization. Another block reads the sensors. These evaluated sensor data are filtered and converted into wheel rotation with a PID controller. Unfortunately all BalanBots were already borrowed when everything was ready on friday the first of february.

8 System Initialization

First the MPU6050 has to be initialized. This was done using I^2C communication. A value was assigned to the individual registers at the slave address 0x68. For example, register 0x19 was assigned the value 7. So the sample rate was set with the formula:

$$1000Hz - \frac{8000Hz}{n - 1} \quad (44)$$

In this formula n was replaced by seven.

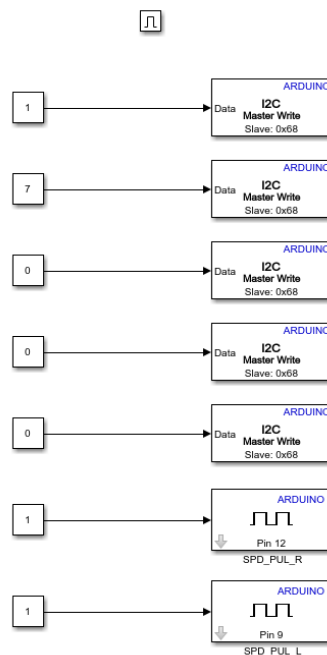


Figure 19: The initialization
Source: Own presentation

9 Read Sensor Datas

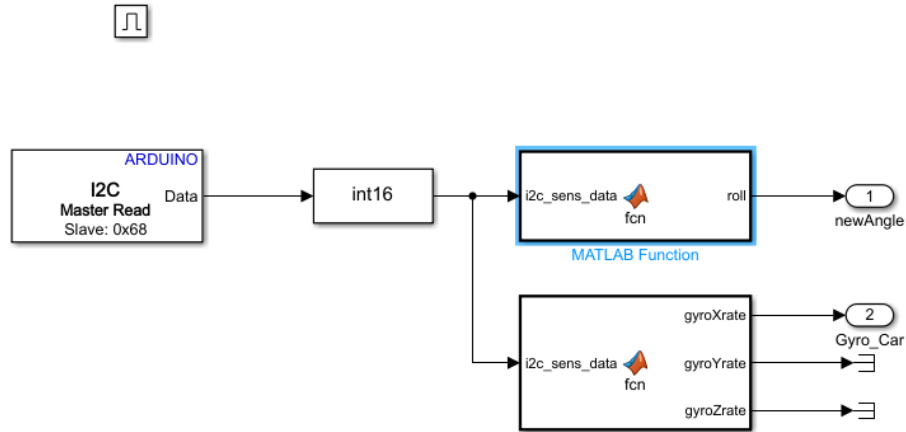


Figure 20: Processing of the read datas

Source: Own presentation

The sensor data is processed in this block. First they are converted into the data type int16. Then the bus signal is split into angle and gyro.

9.1 Angle

The Pythagorean theorem is used to calculate the absolute value of accX and accZ. With this value and accY the arctan finally calculates the angle in radians. At the end the angle is converted into degrees. Sown in Listing 1.

Listing 1: Conversion from the acceleration to the angle

```

1      function roll = fcn(i2c_sens_data)
2      accX = double(i2c_sens_data(1));
3      accY = double(i2c_sens_data(2));
4      accZ = double(i2c_sens_data(3));
5      roll = atan(accY / sqrt(accX * accX + accZ * accZ)) *
          180/pi;
  
```

9.2 Gyro

The data from the gyro sensor is converted to double format and then divided by 131. Shown in Listing 2.

Listing 2: Conversion to the gyros

```

1 function [gyroXrate,gyroYrate,gyroZrate] = fcn(
    i2c_sens_data)
2 gyroX = double(i2c_sens_data(5));
3 gyroY = double(i2c_sens_data(6));
4 gyroZ = double(i2c_sens_data(7));
5 gyroXrate = gyroX/131;
6 gyroYrate = gyroY/131;
7 gyroZrate = gyroZ/131;

```

10 Controller

The controller now uses the sensor data prepared in Section 9. First, measurement errors are reduced with the kalman filter. Then the filtered angle is converted into a PWM signal by a PID controller. For the factors of the PID controller the values recommended in the script were taken first. In external mode the PID parameters would have been finally adjusted. Due to the lack of hardware this could not be done.

[TODO - Simulationsparameter erklären und beschreiben wieso si nicht stimmen können]

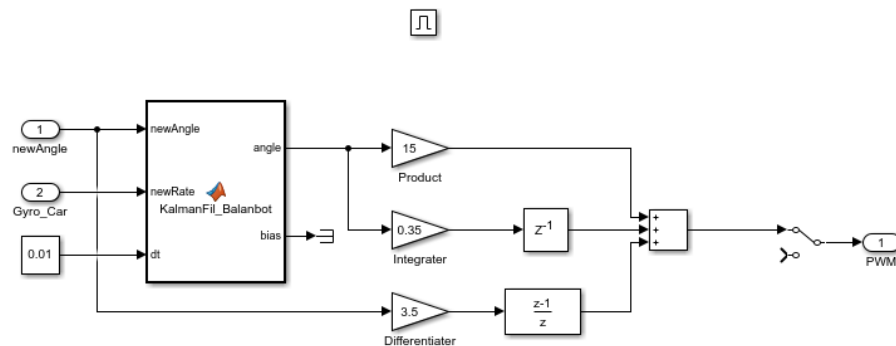


Figure 21: Kalman Filter and PID Controller

Source: Own presentation

11 Actuators

The direction of rotation of the motors is done with an H-bridge of the digital outputs at pins 3, 4, 7 and 8. If the PWM input is greater than zero, the motors rotate in one direction. If it is smaller, they rotate in the other direction. This PWM input value is finally converted into a PWM signal. If the input is zero, the signal is constant zero. The larger this input is, the wider the PWM signal

becomes. If this value is greater than or equal to 255, the signal is constant 1. This conversion is done with the blocks assigned to ports 5 and 6.

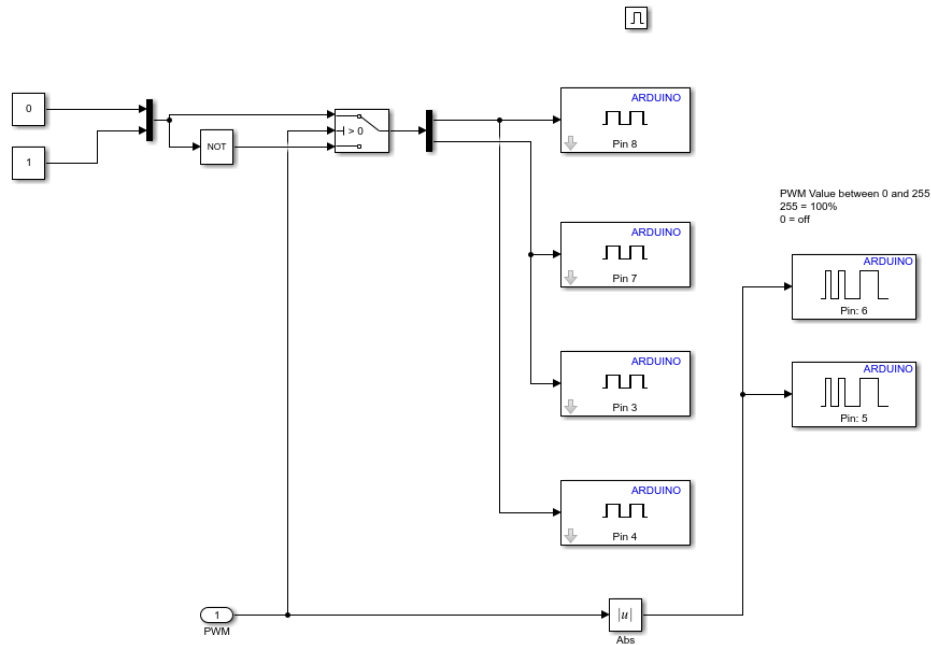


Figure 22: Converts the PWM input in direction of rotation and speed
Source: Own presentation

12 Whole Structure

Finally, all the blocks from the previous sections were assembled to form a single structure shown in Figure 23.

13 Conclusion

Everything was prepared to load the automatically generated code onto the BalanBot and to adjust the PID parameters and to perform further tests and analyses. Unfortunately all BalanBots were borrowed three days before the delivery.

A part of the automatically generated C code can be found in the appendix.

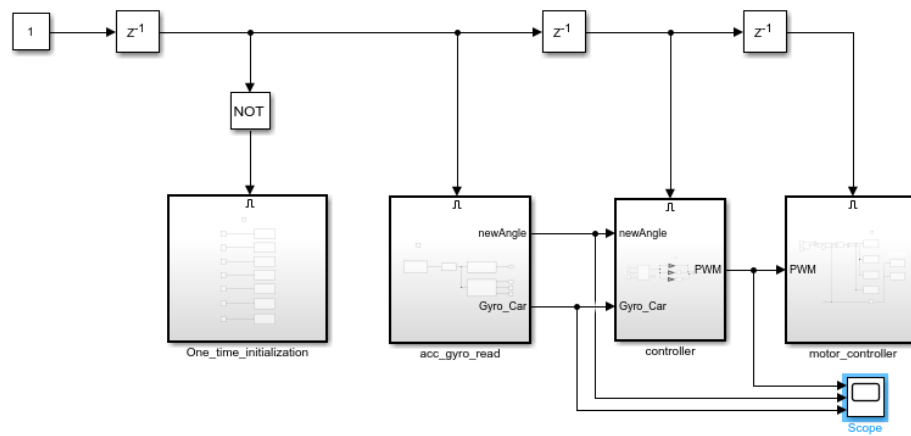


Figure 23: The whole structure consist of blocks from the previous sections
Source: Own presentation

A Appendix

Listing 3: Automatically generated C code

```

1  /*
2  * framework.c
3  *
4  * Academic License - for use in teaching, academic research, and meeting
5  * course requirements at degree granting institutions only. Not for
6  * government, commercial, or other organizational use.
7  *
8  * Code generation for model "framework".
9  *
10 * Model version          : 1.7
11 * Simulink Coder version : 9.0 (R2018b) 24-May-2018
12 * C source code generated on : Fri Feb 1 19:59:15 2019
13 *
14 * Target selection: grt.tlc
15 * Note: GRT includes extra infrastructure and instrumentation for prototyping
16 * Embedded hardware selection: Intel->x86-64 (Windows64)
17 * Code generation objectives: Unspecified
18 * Validation result: Not run
19 */
20
21 #include "framework.h"
22 #include "framework_private.h"
23
24 /* Block signals (default storage) */
25 B_framework_T framework_B;
26
27 /* Block states (default storage) */
28 DW_framework_T framework_DW;
29
30 /* Real-time model */
31 RT_MODEL_framework_T framework_M_;
32 RT_MODEL_framework_T *const framework_M = &framework_M_;
33
34 /* Forward declaration for local functions */
35 static codertarget_arduino_base_int_e_T *arduinoI2CWrite_arduinoI2CWrite
36 (codertarget_arduino_base_int_e_T *obj);
37 static void framework_SystemCore_release(const codertarget_arduino_base_int_e_T
38 *obj);
39 static void framework_SystemCore_delete(const codertarget_arduino_base_int_e_T
40 *obj);
41 static void matlabCodegenHandle_matlabCodegen(codertarget_arduino_base_int_e_T *obj);
42
43 /* Forward declaration for local functions */
44 static codertarget_arduino_base_int_f_T *f_arduinoI2CRead_arduinoI2CRead
45 (codertarget_arduino_base_int_f_T *obj);
46 static codertarget_arduino_base_in_fe_T *arduino_PWMOutput_arduino_PWMOu
47 (codertarget_arduino_base_in_fe_T *obj);

```

```

48 static void matlabCodegenHandle_matlabCod_f(codertarget_arduino_base_block_T *obj);
49 static void framework_SystemCore_release_f(const codertarget_arduino_base_int_f_T
50 *obj);
51 static void framework_SystemCore_delete_fex(const
52 codertarget_arduino_base_int_f_T *obj);
53 static void matlabCodegenHandle_matlabC_fex(codertarget_arduino_base_int_f_T *obj);
54 static void matlabCodegenHandle_matlab_fex2(codertarget_arduino_base_int_fe_T *obj);
55 static codertarget_arduino_base_int_e_T *arduinoI2CWrite_arduinoI2CWrite
56 (codertarget_arduino_base_int_e_T *obj)
57 {
58     codertarget_arduino_base_int_e_T *b_obj;
59     obj->isInitialized = 0;
60     b_obj = obj;
61     obj->Hw.AvailablePwmPinNames.f1 = '2';
62     obj->Hw.AvailablePwmPinNames.f2 = '3';
63     obj->Hw.AvailablePwmPinNames.f3 = '4';
64     obj->Hw.AvailablePwmPinNames.f4 = '5';
65     obj->Hw.AvailablePwmPinNames.f5 = '6';
66     obj->Hw.AvailablePwmPinNames.f6 = '7';
67     obj->Hw.AvailablePwmPinNames.f7 = '8';
68     obj->Hw.AvailablePwmPinNames.f8 = '9';
69     obj->Hw.AvailablePwmPinNames.f9[0] = '1';
70     obj->Hw.AvailablePwmPinNames.f9[1] = '0';
71     obj->Hw.AvailablePwmPinNames.f10[0] = '1';
72     obj->Hw.AvailablePwmPinNames.f10[1] = '1';
73     obj->Hw.AvailablePwmPinNames.f11[0] = '1';
74     obj->Hw.AvailablePwmPinNames.f11[1] = '2';
75     obj->Hw.AvailablePwmPinNames.f12[0] = '1';
76     obj->Hw.AvailablePwmPinNames.f12[1] = '3';
77     obj->matlabCodegenIsDeleted = false;
78     return b_obj;
79 }
80
81 static void framework_SystemCore_release(const codertarget_arduino_base_int_e_T
82 *obj)
83 {
84     if ((obj->isInitialized == 1) && obj->isSetupComplete) {
85         MW_I2C_Close(obj->MW_I2C_HANDLE);
86     }
87 }
88
89 static void framework_SystemCore_delete(const codertarget_arduino_base_int_e_T
90 *obj)
91 {
92     framework_SystemCore_release(obj);
93 }
94
95 static void matlabCodegenHandle_matlabCodeg(codertarget_arduino_base_int_e_T *obj)
96 {
97     if (!obj->matlabCodegenIsDeleted) {
98         obj->matlabCodegenIsDeleted = true;

```

```

99         framework_SystemCore_delete(obj);
100     }
101 }
102
103 /*
104 * Start for atomic system:
105 *     synthesized block
106 *     synthesized block
107 *     synthesized block
108 *     synthesized block
109 *     synthesized block
110 */
111 void framework_I2CWrite4_Start(DW_I2CWrite4_framework_T *localDW)
112 {
113     codertarget_arduino_base_int_e_T *obj;
114     uint32_T i2cname;
115
116     /* Start for MATLABSystem: '<S1>/I2C Write4' */
117     localDW->obj.matlabCodegenIsDeleted = true;
118     arduinoI2CWrite_arduinoI2CWrite(&localDW->obj);
119     localDW->obj.isempty = true;
120     obj = &localDW->obj;
121     localDW->obj.isSetupComplete = false;
122     localDW->obj.isInitialized = 1;
123     i2cname = 0;
124     obj->MW_I2C_HANDLE = MW_I2C_Open(i2cname, 0);
125     localDW->obj.BusSpeed = 100000U;
126     MW_I2C_SetBusSpeed(localDW->obj.MW_I2C_HANDLE, localDW->obj.BusSpeed);
127     localDW->obj.isSetupComplete = true;
128 }
129
130 /*
131 * Output and update for atomic system:
132 *     synthesized block
133 *     synthesized block
134 *     synthesized block
135 *     synthesized block
136 *     synthesized block
137 */
138 void framework_I2CWrite4(real_T rtu_0, DW_I2CWrite4_framework_T *localDW)
139 {
140     uint8_T SwappedDataBytes[8];
141     uint8_T b_SwappedDataBytes[9];
142     real_T b_x;
143     uint8_T xtmp;
144     int32_T i;
145
146     /* MATLABSystem: '<S1>/I2C Write4' */
147     memcpy((void *)&SwappedDataBytes[0], (void *)&rtu_0, (uint32_T)((size_t)8 *
148 sizeof(uint8_T)));
149     xtmp = SwappedDataBytes[0];

```

```

150     SwappedDataBytes[0] = SwappedDataBytes[7];
151     SwappedDataBytes[7] = xtmp;
152     xtmp = SwappedDataBytes[1];
153     SwappedDataBytes[1] = SwappedDataBytes[6];
154     SwappedDataBytes[6] = xtmp;
155     xtmp = SwappedDataBytes[2];
156     SwappedDataBytes[2] = SwappedDataBytes[5];
157     SwappedDataBytes[5] = xtmp;
158     xtmp = SwappedDataBytes[3];
159     SwappedDataBytes[3] = SwappedDataBytes[4];
160     SwappedDataBytes[4] = xtmp;
161     memcpy((void *)&b_x, (void *)&SwappedDataBytes[0], (uint32_T)((size_t)1 *
162     sizeof(real_T)));
163     memcpy((void *)&SwappedDataBytes[0], (void *)&b_x, (uint32_T)((size_t)8 *
164     sizeof(uint8_T)));
165     b_SwappedDataBytes[0] = 107U;
166     for (i = 0; i < 8; i++) {
167         b_SwappedDataBytes[i + 1] = SwappedDataBytes[i];
168     }
169
170     MW_I2C_MasterWrite(localDW->obj.MW_I2C_HANDLE, 104U, b_SwappedDataBytes, 9U
171     ,
172     false, false);
173
174     /* End of MATLABSystem: '<S1>/I2C Write4' */
175 }
176
177 /*
178  * Termination for atomic system:
179  *     synthesized block
180  *     synthesized block
181  *     synthesized block
182  *     synthesized block
183  *     synthesized block
184  */
185 void framework_I2CWrite4_Term(DW_I2CWrite4_framework_T *localDW)
186 {
187     /* Terminate for MATLABSystem: '<S1>/I2C Write4' */
188     matlabCodegenHandle_matlabCodeg(&localDW->obj);
189 }
190
191 static codertarget_arduino_base_int_f_T *f_arduinoI2CRead_arduinoI2CRead
192 (codertarget_arduino_base_int_f_T *obj)
193 {
194     codertarget_arduino_base_int_f_T *b_obj;
195     obj->isInitialized = 0;
196     b_obj = obj;
197     obj->Hw.AvailablePwmPinNames.f1 = '2';
198     obj->Hw.AvailablePwmPinNames.f2 = '3';
199     obj->Hw.AvailablePwmPinNames.f3 = '4';
200     obj->Hw.AvailablePwmPinNames.f4 = '5';

```

```

200     obj->Hw.AvailablePwmPinNames.f5 = '6';
201     obj->Hw.AvailablePwmPinNames.f6 = '7';
202     obj->Hw.AvailablePwmPinNames.f7 = '8';
203     obj->Hw.AvailablePwmPinNames.f8 = '9';
204     obj->Hw.AvailablePwmPinNames.f9[0] = '1';
205     obj->Hw.AvailablePwmPinNames.f9[1] = '0';
206     obj->Hw.AvailablePwmPinNames.f10[0] = '1';
207     obj->Hw.AvailablePwmPinNames.f10[1] = '1';
208     obj->Hw.AvailablePwmPinNames.f11[0] = '1';
209     obj->Hw.AvailablePwmPinNames.f11[1] = '2';
210     obj->Hw.AvailablePwmPinNames.f12[0] = '1';
211     obj->Hw.AvailablePwmPinNames.f12[1] = '3';
212     obj->matlabCodegenIsDeleted = false;
213     return b_obj;
214 }
215
216 static codertarget_arduino_base_in_fe_T *arduino_PWMOutput_arduino_PWMOut
217 (codertarget_arduino_base_in_fe_T *obj)
218 {
219     codertarget_arduino_base_in_fe_T *b_obj;
220     obj->isInitialized = 0;
221     b_obj = obj;
222     obj->Hw.AvailablePwmPinNames.f1 = '2';
223     obj->Hw.AvailablePwmPinNames.f2 = '3';
224     obj->Hw.AvailablePwmPinNames.f3 = '4';
225     obj->Hw.AvailablePwmPinNames.f4 = '5';
226     obj->Hw.AvailablePwmPinNames.f5 = '6';
227     obj->Hw.AvailablePwmPinNames.f6 = '7';
228     obj->Hw.AvailablePwmPinNames.f7 = '8';
229     obj->Hw.AvailablePwmPinNames.f8 = '9';
230     obj->Hw.AvailablePwmPinNames.f9[0] = '1';
231     obj->Hw.AvailablePwmPinNames.f9[1] = '0';
232     obj->Hw.AvailablePwmPinNames.f10[0] = '1';
233     obj->Hw.AvailablePwmPinNames.f10[1] = '1';
234     obj->Hw.AvailablePwmPinNames.f11[0] = '1';
235     obj->Hw.AvailablePwmPinNames.f11[1] = '2';
236     obj->Hw.AvailablePwmPinNames.f12[0] = '1';
237     obj->Hw.AvailablePwmPinNames.f12[1] = '3';
238     obj->matlabCodegenIsDeleted = false;
239     return b_obj;
240 }
241
242 static void matlabCodegenHandle_matlabCod_f(codertarget_arduino_block_T *obj)
243 {
244     if (!obj->matlabCodegenIsDeleted) {
245         obj->matlabCodegenIsDeleted = true;
246     }
247 }
248
249 static void framework_SystemCore_release_f(const codertarget_arduino_base_int_f_T
250 *obj)

```

```

251 {
252     if ((obj->isInitialized == 1) && obj->isSetupComplete) {
253         MW_I2C_Close(obj->MW_I2C_HANDLE);
254     }
255 }
256
257 static void framework_SystemCore_delete_fex(const
258 codertarget_arduino_base_int_f_T *obj)
259 {
260     framework_SystemCore_release_f(obj);
261 }
262
263 static void matlabCodegenHandle_matlabC_fex(codertarget_arduino_base_int_f_T *obj)
264 {
265     if (!obj->matlabCodegenIsDeleted) {
266         obj->matlabCodegenIsDeleted = true;
267         framework_SystemCore_delete_fex(obj);
268     }
269 }
270
271 static void matlabCodegenHandle_matlab_fex2(codertarget_arduino_base_in_fe_T *obj)
272 {
273     if (!obj->matlabCodegenIsDeleted) {
274         obj->matlabCodegenIsDeleted = true;
275     }
276 }
277
278 /* Model step function */
279 void framework_step(void)
280 {
281     real_T F[4];
282     real_T y;
283     real_T K[2];
284     static const real_T a[4] = { 0.001, 0.0, 0.0, 0.003 };
285
286     int16_T b_output[7];
287     uint8_T status;
288     uint8_T output_raw[14];
289     uint8_T y_0[2];
290     int16_T x;
291     uint8_T b_x[2];
292     real_T rtb_Delay1;
293     real_T rtb_Delay;
294     int32_T i;
295     real_T F_0[2];
296     real_T tmp[2];
297     real_T F_1[4];
298     real_T F_2[4];
299     real_T K_0;
300     real_T P_tmp;
301

```

```

302     /* Delay: '<Root>/Delay' */
303     rtb_Delay = framework_DW.Delay_DSTATE;
304
305     /* Outputs for Enabled SubSystem: '<Root>/One_time_initialization'
        incorporates:
306     *   EnablePort: '<S1>/Enable'
307     */
308     /* Logic: '<Root>/Logical Operator' incorporates:
309     *   Constant: '<S1>/Constant1'
310     *   Constant: '<S1>/Constant2'
311     *   Constant: '<S1>/Constant3'
312     *   Constant: '<S1>/Constant4'
313     *   Delay: '<Root>/Delay'
314     */
315     if (!(framework_DW.Delay_DSTATE != 0.0)) {
316         framework_I2CWrite4(framework_P.Constant1_Value, &framework_DW.
            I2CWrite);
317         framework_I2CWrite4(framework_P.Constant2_Value, &framework_DW.
            I2CWrite1);
318         framework_I2CWrite4(framework_P.Constant3_Value, &framework_DW.
            I2CWrite2);
319         framework_I2CWrite4(framework_P.Constant4_Value, &framework_DW.
            I2CWrite3);
320
321         /* MATLABSystem: '<S1>/I2C Write4' incorporates:
322         *   Constant: '<S1>/Constant1'
323         *   Constant: '<S1>/Constant2'
324         *   Constant: '<S1>/Constant3'
325         *   Constant: '<S1>/Constant4'
326         *   Constant: '<S1>/Constant5'
327         */
328         framework_I2CWrite4(framework_P.Constant5_Value, &framework_DW.
            I2CWrite4);
329
330         /* DataTypeConversion: '<S6>/Data Type Conversion' incorporates:
331         *   Constant: '<S1>/Constant6'
332         */
333         if (framework_P.Constant6_Value < 256.0) {
334             if (framework_P.Constant6_Value >= 0.0) {
335                 status = (uint8_T)framework_P.Constant6_Value;
336             } else {
337                 status = 0U;
338             }
339         } else {
340             status = MAX_uint8_T;
341         }
342
343         /* End of DataTypeConversion: '<S6>/Data Type Conversion' */
344
345         /* MATLABSystem: '<S6>/Digital Output' */
346         writeDigitalPin(12, status);

```

```

347
348     /* DataTypeConversion: '<S5>/Data Type Conversion' incorporates:
349     * Constant: '<S1>/Constant7'
350     */
351     if (framework_P.Constant7_Value < 256.0) {
352         if (framework_P.Constant7_Value >= 0.0) {
353             status = (uint8_T)framework_P.Constant7_Value;
354         } else {
355             status = 0U;
356         }
357     } else {
358         status = MAX_uint8_T;
359     }
360
361     /* End of DataTypeConversion: '<S5>/Data Type Conversion' */
362
363     /* MATLABSystem: '<S5>/Digital Output' */
364     writeDigitalPin(9, status);
365 }
366
367 /* End of Logic: '<Root>/Logical Operator' */
368 /* End of Outputs for SubSystem: '<Root>/One_time_initialization' */
369
370 /* Outputs for Enabled SubSystem: '<Root>/acc_gyro_read' incorporates:
371 * EnablePort: '<S2>/Enable'
372 */
373 /* Delay: '<Root>/Delay' */
374 if (framework_DW.Delay_DSTATE > 0.0) {
375     /* MATLABSystem: '<S2>/I2C Read' */
376     if (framework_DW.obj.SampleTime != framework_P.I2CRead_SampleTime)
377     {
378         framework_DW.obj.SampleTime = framework_P.
379             I2CRead_SampleTime;
380     }
381
382     status = 59U;
383     status = MW_I2C_MasterWrite(framework_DW.obj.MW_I2C_HANDLE, 104U, &
384         status,
385         1U, true, false);
386     if (0 == status) {
387         MW_I2C_MasterRead(framework_DW.obj.MW_I2C_HANDLE, 104U,
388             output_raw, 14U,
389             false, true);
390         memcpy((void *)&b_output[0], (void *)&output_raw[0], (
391             uint32_T)((size_t)7 *
392             sizeof(int16_T)));
393         for (i = 0; i < 7; i++) {
394             x = b_output[i];
395             memcpy((void *)&y_0[0], (void *)&x, (uint32_T)((
396                 size_t)2 * sizeof
397                 (uint8_T)));

```

```

392         b_x[0] = y_0[1];
393         b_x[1] = y_0[0];
394         memcpy((void *)&b_output[i], (void *)&b_x[0], (
            uint32_T)((size_t)1 *
395             sizeof(int16_T)));
396     }
397 } else {
398     for (i = 0; i < 7; i++) {
399         b_output[i] = 0;
400     }
401 }
402
403 /* MATLAB Function: '<S2>/MATLAB Function' incorporates:
404 *   MATLABSystem: '<S2>/I2C Read'
405 */
406 framework_B.roll = atan((real_T)b_output[1] / sqrt((real_T)(
    b_output[0] *
407     b_output[0]) + (real_T)(b_output[2] * b_output[2]))) * 180.0 /
408     3.1415926535897931;

```

Listing 4: Automatically generated C code

```

407 /* MATLAB Function: '<S2>/MATLAB Function1' incorporates:
408 *   MATLABSystem: '<S2>/I2C Read'
409 */
410 framework_B.gyroXrate = (real_T)b_output[4] / 131.0;
411 }
412
413 /* End of Outputs for SubSystem: '<Root>/acc_gyro_read' */
414
415 /* Delay: '<Root>/Delay1' incorporates:
416 *   Constant: '<S3>/Constant'
417 *   MATLAB Function: '<S3>/MATLAB Function'
418 */
419 rtb_Delay1 = framework_DW.Delay1_DSTATE;
420
421 /* Outputs for Enabled SubSystem: '<Root>/controller' incorporates:
422 *   EnablePort: '<S3>/Enable'
423 */
424 if (framework_DW.Delay1_DSTATE > 0.0) {
425     /* MATLAB Function: '<S3>/MATLAB Function' incorporates:
426     *   Constant: '<S3>/Constant'
427     */
428     F[0] = 1.0;
429     F[2] = -framework_P.Constant_Value;
430     F[1] = 0.0;
431     F[3] = 1.0;
432     F_0[0] = -framework_P.Constant_Value * framework_DW.x[1] +
        framework_DW.x[0];
433     F_0[1] = 0.0 * framework_DW.x[0];
434     F_0[1] += framework_DW.x[1];
435

```

```

436      /* MATLAB Function: '<S3>/MATLAB Function' incorporates:
437      *   Constant: '<S3>/Constant'
438      */
439      tmp[0] = framework_P.Constant_Value * framework_B.gyroXrate;
440      tmp[1] = 0.0 * framework_B.gyroXrate;
441      for (i = 0; i < 2; i++) {
442          framework_DW.x[i] = F_0[i] + tmp[i];
443          F_1[i] = 0.0;
444          F_1[i] += F[i] * framework_DW.P[0];
445          y = F[i + 2];
446          F_1[i] += y * framework_DW.P[1];
447          F_1[i + 2] = 0.0;
448          F_1[i + 2] += F[i] * framework_DW.P[2];
449          F_1[i + 2] += y * framework_DW.P[3];
450      }
451
452      y = 0.0;
453      for (i = 0; i < 2; i++) {
454          P_tmp = F_1[i + 2];
455          framework_DW.P[i] = (P_tmp * -framework_P.Constant_Value +
456              F_1[i]) + a[i] *
457              framework_P.Constant_Value;
458          framework_DW.P[i + 2] = (F_1[i] * 0.0 + P_tmp) + a[i + 2] *
459              framework_P.Constant_Value;
460          y += (1.0 - (real_T)i) * framework_DW.x[i];
461      }
462
463      y = framework_B.roll - y;
464      P_tmp = (0.0 * framework_DW.P[3] + framework_DW.P[2]) * 0.0 + (0.0
465          *
466          framework_DW.P[1] + framework_DW.P[0]);
467      K_0 = (framework_DW.P[2] * 0.0 + framework_DW.P[0]) / (P_tmp +
468          0.03);
469      framework_DW.x[0] += K_0 * y;
470      K[0] = K_0;
471
472      /* MATLAB Function: '<S3>/MATLAB Function' */
473      K_0 = (framework_DW.P[3] * 0.0 + framework_DW.P[1]) / (P_tmp +
474          0.03);
475      framework_DW.x[1] += K_0 * y;
476      K[1] = K_0;
477
478      /* MATLAB Function: '<S3>/MATLAB Function' */
479      F[1] = 0.0;
480      F[2] = 0.0;
481      F[0] = 1.0;
482      F[3] = 1.0;
483      for (i = 0; i < 2; i++) {
484          F_1[i] = F[i] - K[i];
485          F_1[i + 2] = F[i + 2] - K[i] * 0.0;
486          F_2[i] = 0.0;

```

```

483         F_2[i] += F_1[i] * framework_DW.P[0];
484         y = F_1[i + 2];
485         F_2[i] += y * framework_DW.P[1];
486         F_2[i + 2] = 0.0;
487         F_2[i + 2] += F_1[i] * framework_DW.P[2];
488         F_2[i + 2] += y * framework_DW.P[3];
489     }
490
491     framework_DW.P[0] = F_2[0];
492     framework_DW.P[1] = F_2[1];
493     framework_DW.P[2] = F_2[2];
494     framework_DW.P[3] = F_2[3];
495
496     /* Gain: '<S3>/Differentiator' incorporates:
497     *   MATLAB Function: '<S3>/MATLAB Function'
498     */
499     y = framework_P.Differentiator_Gain * framework_DW.x[0];
500
501     /* Sum: '<S3>/Add' incorporates:
502     *   Delay: '<S3>/Delay3'
503     *   Gain: '<S3>/Product'
504     *   MATLAB Function: '<S3>/MATLAB Function'
505     *   Sum: '<S9>/Diff'
506     *   UnitDelay: '<S9>/UD'
507     */
508     framework_B.Add = (framework_P.Product_Gain * framework_DW.x[0] +
509     framework_DW.Delay3_DSTATE) + (y - framework_DW.UD_DSTATE);
510
511     /* Update for Delay: '<S3>/Delay3' incorporates:
512     *   Gain: '<S3>/Integrator'
513     *   MATLAB Function: '<S3>/MATLAB Function'
514     */
515     framework_DW.Delay3_DSTATE = framework_P.Integrator_Gain *
        framework_DW.x[0];
516
517     /* Update for UnitDelay: '<S9>/UD' */
518     framework_DW.UD_DSTATE = y;
519 }
520
521 /* End of Delay: '<Root>/Delay1' */
522 /* End of Outputs for SubSystem: '<Root>/controller' */
523 /* Outputs for Enabled SubSystem: '<Root>/motor_controller' incorporates:
524 *   EnablePort: '<S4>/Enable'
525 */
526 /* Delay: '<Root>/Delay2' */
527 if (framework_DW.Delay2_DSTATE > 0.0) {
528     /* Switch: '<S4>/Switch' incorporates:
529     *   Constant: '<S4>/Constant3'
530     *   Constant: '<S4>/Constant4'
531     *   Logic: '<S4>/Logical Operator1'
532     */

```

```

533     if (framework_B.Add > framework_P.Switch_Threshold) {
534         K[0] = framework_P.Constant3_Value_b;
535         K[1] = framework_P.Constant4_Value_n;
536     } else {
537         K[0] = !(framework_P.Constant3_Value_b != 0.0);
538         K[1] = !(framework_P.Constant4_Value_n != 0.0);
539     }
540
541     /* End of Switch: '<S4>/Switch' */
542
543     /* DataTypeConversion: '<S11>/Data Type Conversion' */
544     if (K[0] < 256.0) {
545         if (K[0] >= 0.0) {
546             status = (uint8_T)K[0];
547         } else {
548             status = 0U;
549         }
550     } else {
551         status = MAX_uint8_T;
552     }
553
554     /* End of DataTypeConversion: '<S11>/Data Type Conversion' */
555
556     /* MATLABSystem: '<S11>/Digital Output' */
557     writeDigitalPin(8, status);
558
559     /* DataTypeConversion: '<S12>/Data Type Conversion' */
560     if (K[1] < 256.0) {
561         if (K[1] >= 0.0) {
562             status = (uint8_T)K[1];
563         } else {
564             status = 0U;
565         }
566     } else {
567         status = MAX_uint8_T;
568     }
569
570     /* End of DataTypeConversion: '<S12>/Data Type Conversion' */
571
572     /* MATLABSystem: '<S12>/Digital Output' */
573     writeDigitalPin(7, status);
574
575     /* DataTypeConversion: '<S13>/Data Type Conversion' */
576     if (K[1] < 256.0) {
577         if (K[1] >= 0.0) {
578             status = (uint8_T)K[1];
579         } else {
580             status = 0U;
581         }
582     } else {
583         status = MAX_uint8_T;

```

```

584     }
585
586     /* End of DataTypeConversion: '<S13>/Data Type Conversion' */
587
588     /* MATLABSystem: '<S13>/Digital Output' */
589     writeDigitalPin(3, status);
590
591     /* DataTypeConversion: '<S14>/Data Type Conversion' */
592     if (K[0] < 256.0) {
593         if (K[0] >= 0.0) {
594             status = (uint8_T)K[0];
595         } else {
596             status = 0U;
597         }
598     } else {
599         status = MAX_uint8_T;
600     }
601
602     /* End of DataTypeConversion: '<S14>/Data Type Conversion' */
603
604     /* MATLABSystem: '<S14>/Digital Output' */
605     writeDigitalPin(4, status);
606
607     /* Abs: '<S4>/Abs' */
608     y = fabs(framework_B.Add);
609
610     /* MATLABSystem: '<S4>/PWM' */
611     MW_PWM_SetDutyCycle(framework_DW.obj_h.MW_PWM_HANDLE, y);
612
613     /* MATLABSystem: '<S4>/PWM1' */
614     MW_PWM_SetDutyCycle(framework_DW.obj_n.MW_PWM_HANDLE, y);
615 }
616
617 /* End of Delay: '<Root>/Delay2' */
618 /* End of Outputs for SubSystem: '<Root>/motor_controller' */
619
620 /* Update for Delay: '<Root>/Delay' incorporates:
621 *   Constant: '<Root>/Constant'
622 */
623 framework_DW.Delay_DSTATE = framework_P.Constant_Value_c;
624
625 /* Update for Delay: '<Root>/Delay1' */
626 framework_DW.Delay1_DSTATE = rtb_Delay;
627
628 /* Update for Delay: '<Root>/Delay2' */
629 framework_DW.Delay2_DSTATE = rtb_Delay1;
630
631 /* Matfile logging */
632 rt_UpdateTXYLogVars(framework_M->rtwLogInfo, (&framework_M->Timing.
633     taskTime0));

```

```

634     /* signal main to stop simulation */
635     {
636         /* Sample time: [0.01s, 0.0s] */
637         if ((rtmGetTFinal(framework_M)!=-1) &&
638             !((rtmGetTFinal(framework_M)-framework_M->Timing.taskTime0) >
639              framework_M->Timing.taskTime0 * (DBL_EPSILON))) {
640             rtmSetErrorStatus(framework_M, "Simulation_finished");
641         }
642     }
643
644     /* Update absolute time for base rate */
645     /* The "clockTick0" counts the number of times the code of this task has
646     * been executed. The absolute time is the multiplication of "clockTick0"
647     * and "Timing.stepSize0". Size of "clockTick0" ensures timer will not
648     * overflow during the application lifespan selected.
649     * Timer of this task consists of two 32 bit unsigned integers.
650     * The two integers represent the low bits Timing.clockTick0 and the high
651     * bits
652     * Timing.clockTickH0. When the low bit overflows to 0, the high bits
653     * increment.
654     */
655     if (!(++framework_M->Timing.clockTick0)) {
656         ++framework_M->Timing.clockTickH0;
657     }
658
659     framework_M->Timing.taskTime0 = framework_M->Timing.clockTick0 *
660     framework_M->Timing.stepSize0 + framework_M->Timing.clockTickH0 *
661     framework_M->Timing.stepSize0 * 4294967296.0;
662 }
663
664 /* Model initialize function */
665 void framework_initialize(void)
666 {
667     /* Registration code */
668
669     /* initialize non-finites */
670     rt_InitInfAndNaN(sizeof(real_T));
671
672     /* initialize real-time model */
673     (void) memset((void *)framework_M, 0,
674                 sizeof(RT_MODEL_framework_T));
675     rtmSetTFinal(framework_M, 10.0);
676     framework_M->Timing.stepSize0 = 0.01;
677
678     /* Setup for data logging */
679     {
680         static RTWLogInfo rt_DataLoggingInfo;
681         rt_DataLoggingInfo.loggingInterval = NULL;
682         framework_M->rtwLogInfo = &rt_DataLoggingInfo;
683     }
684
685     /* Setup for data logging */

```

```

683     {
684         rtliSetLogXSignalInfo (framework_M->rtwLogInfo, (NULL));
685         rtliSetLogXSignalPtrs (framework_M->rtwLogInfo, (NULL));
686         rtliSetLogT (framework_M->rtwLogInfo, "tout");
687         rtliSetLogX (framework_M->rtwLogInfo, "");
688         rtliSetLogXFinal (framework_M->rtwLogInfo, "");
689         rtliSetLogVarNameModifier (framework_M->rtwLogInfo, "rt_");
690         rtliSetLogFormat (framework_M->rtwLogInfo, 4);
691         rtliSetLogMaxRows (framework_M->rtwLogInfo, 0);
692         rtliSetLogDecimation (framework_M->rtwLogInfo, 1);
693         rtliSetLogY (framework_M->rtwLogInfo, "");
694         rtliSetLogYSignalInfo (framework_M->rtwLogInfo, (NULL));
695         rtliSetLogYSignalPtrs (framework_M->rtwLogInfo, (NULL));
696     }

```

Listing 5: Automatically generated C code

```

698
699     /* block I/O */
700     (void) memset ((void *) &framework_B, 0,
701         sizeof (B_framework_T));
702
703     /* states (dwork) */
704     (void) memset ((void *) &framework_DW, 0,
705         sizeof (DW_framework_T));
706
707     /* Matfile logging */
708     rt_StartDataLoggingWithStartTime (framework_M->rtwLogInfo, 0.0, rtmGetTFinal
709         (framework_M), framework_M->Timing.stepSize0, (&rtmGetErrorStatus
710         (framework_M)));
711
712     {
713         codertarget_arduino_base_int_f_T *obj;
714         uint32_T i2cname;
715         codertarget_arduino_base_in_fe_T *obj_0;
716
717         /* Start for Enabled SubSystem: '<Root>/One_time_initialization' */
718         /* Constant: '<S1>/Constant1' */
719         framework_I2CWrite4_Start (&framework_DW.I2CWrite);
720
721         /* Constant: '<S1>/Constant2' */
722         framework_I2CWrite4_Start (&framework_DW.I2CWrite1);
723
724         /* Constant: '<S1>/Constant3' */
725         framework_I2CWrite4_Start (&framework_DW.I2CWrite2);
726
727         /* Constant: '<S1>/Constant4' */
728         framework_I2CWrite4_Start (&framework_DW.I2CWrite3);
729
730         /* Start for MATLABSystem: '<S1>/I2C Write4' incorporates:
731         * Constant: '<S1>/Constant5'
732         */

```



```

733     framework_I2CWrite4_Start(&framework_DW.I2CWrite4);
734
735     /* Start for MATLABSystem: '<S6>/Digital Output' */
736     framework_DW.obj_g.matlabCodegenIsDeleted = true;
737     framework_DW.obj_g.isInitialized = 0;
738     framework_DW.obj_g.matlabCodegenIsDeleted = false;
739     framework_DW.obj_g.isempty_f = true;
740     framework_DW.obj_g.isSetupComplete = false;
741     framework_DW.obj_g.isInitialized = 1;
742     digitalIOSetup(12, true);
743     framework_DW.obj_g.isSetupComplete = true;
744
745     /* Start for MATLABSystem: '<S5>/Digital Output' */
746     framework_DW.obj_lq.matlabCodegenIsDeleted = true;
747     framework_DW.obj_lq.isInitialized = 0;
748     framework_DW.obj_lq.matlabCodegenIsDeleted = false;
749     framework_DW.obj_lq.isempty_e1 = true;
750     framework_DW.obj_lq.isSetupComplete = false;
751     framework_DW.obj_lq.isInitialized = 1;
752     digitalIOSetup(9, true);
753     framework_DW.obj_lq.isSetupComplete = true;
754
755     /* End of Start for SubSystem: '<Root>/One_time_initialization' */
756
757     /* Start for Enabled SubSystem: '<Root>/acc_gyro_read' */
758     /* Start for MATLABSystem: '<S2>/I2C Read' */
759     framework_DW.obj.matlabCodegenIsDeleted = true;
760     f_arduinoI2CRead_arduinoI2CRead(&framework_DW.obj);
761     framework_DW.obj.isempty_dv = true;
762     framework_DW.obj.SampleTime = framework_P.I2CRead_SampleTime;
763     obj = &framework_DW.obj;
764     framework_DW.obj.isSetupComplete = false;
765     framework_DW.obj.isInitialized = 1;
766     i2cname = 0;
767     obj->MW_I2C_HANDLE = MW_I2C_Open(i2cname, 0);
768     framework_DW.obj.BusSpeed = 100000U;
769     MW_I2C_SetBusSpeed(framework_DW.obj.MW_I2C_HANDLE, framework_DW.obj
        .BusSpeed);
770     framework_DW.obj.isSetupComplete = true;
771
772     /* End of Start for SubSystem: '<Root>/acc_gyro_read' */
773     /* Start for Enabled SubSystem: '<Root>/motor_controller' */
774     /* Start for MATLABSystem: '<S11>/Digital Output' */
775     framework_DW.obj_l.matlabCodegenIsDeleted = true;
776     framework_DW.obj_l.isInitialized = 0;
777     framework_DW.obj_l.matlabCodegenIsDeleted = false;
778     framework_DW.obj_l.isempty_d = true;
779     framework_DW.obj_l.isSetupComplete = false;
780     framework_DW.obj_l.isInitialized = 1;
781     digitalIOSetup(8, true);
782     framework_DW.obj_l.isSetupComplete = true;

```

```

783
784      /* Start for MATLABSystem: '<S12>/Digital Output' */
785      framework_DW.obj_na.matlabCodegenIsDeleted = true;
786      framework_DW.obj_na.isInitialized = 0;
787      framework_DW.obj_na.matlabCodegenIsDeleted = false;
788      framework_DW.obj_na.isempty_o = true;
789      framework_DW.obj_na.isSetupComplete = false;
790      framework_DW.obj_na.isInitialized = 1;
791      digitalIOSetup(7, true);
792      framework_DW.obj_na.isSetupComplete = true;
793
794      /* Start for MATLABSystem: '<S13>/Digital Output' */
795      framework_DW.obj_d.matlabCodegenIsDeleted = true;
796      framework_DW.obj_d.isInitialized = 0;
797      framework_DW.obj_d.matlabCodegenIsDeleted = false;
798      framework_DW.obj_d.isempty_i = true;
799      framework_DW.obj_d.isSetupComplete = false;
800      framework_DW.obj_d.isInitialized = 1;
801      digitalIOSetup(3, true);
802      framework_DW.obj_d.isSetupComplete = true;
803
804      /* Start for MATLABSystem: '<S14>/Digital Output' */
805      framework_DW.obj_j.matlabCodegenIsDeleted = true;
806      framework_DW.obj_j.isInitialized = 0;
807      framework_DW.obj_j.matlabCodegenIsDeleted = false;
808      framework_DW.obj_j.isempty = true;
809      framework_DW.obj_j.isSetupComplete = false;
810      framework_DW.obj_j.isInitialized = 1;
811      digitalIOSetup(4, true);
812      framework_DW.obj_j.isSetupComplete = true;
813
814      /* Start for MATLABSystem: '<S4>/PWM' */
815      framework_DW.obj_h.matlabCodegenIsDeleted = true;
816      arduino_PWMOutput_arduino_PWMOu(&framework_DW.obj_h);
817      framework_DW.obj_h.isempty_m = true;
818      obj_0 = &framework_DW.obj_h;
819      framework_DW.obj_h.isSetupComplete = false;
820      framework_DW.obj_h.isInitialized = 1;
821      obj_0->MW_PWM_HANDLE = MW_PWM_Open(6U, 0.0, 0.0);
822      MW_PWM_Start(framework_DW.obj_h.MW_PWM_HANDLE);
823      framework_DW.obj_h.isSetupComplete = true;
824
825      /* Start for MATLABSystem: '<S4>/PWM1' */
826      framework_DW.obj_n.matlabCodegenIsDeleted = true;
827      arduino_PWMOutput_arduino_PWMOu(&framework_DW.obj_n);
828      framework_DW.obj_n.isempty_e = true;
829      obj_0 = &framework_DW.obj_n;
830      framework_DW.obj_n.isSetupComplete = false;
831      framework_DW.obj_n.isInitialized = 1;
832      obj_0->MW_PWM_HANDLE = MW_PWM_Open(5U, 0.0, 0.0);
833      MW_PWM_Start(framework_DW.obj_n.MW_PWM_HANDLE);

```

```

834         framework_DW.obj_n.isSetupComplete = true;
835
836         /* End of Start for SubSystem: '<Root>/motor_controller' */
837     }
838
839     /* InitializeConditions for Delay: '<Root>/Delay' */
840     framework_DW.Delay_DSTATE = framework_P.Delay_InitialCondition;
841
842     /* InitializeConditions for Delay: '<Root>/Delay1' */
843     framework_DW.Delay1_DSTATE = framework_P.Delay1_InitialCondition;
844
845     /* InitializeConditions for Delay: '<Root>/Delay2' */
846     framework_DW.Delay2_DSTATE = framework_P.Delay2_InitialCondition;
847
848     /* SystemInitialize for Enabled SubSystem: '<Root>/acc_gyro_read' */
849     /* SystemInitialize for Outport: '<S2>/newAngle' */
850     framework_B.roll = framework_P.newAngle_Y0;
851
852     /* SystemInitialize for Outport: '<S2>/Gyro_Car' */
853     framework_B.gyroXrate = framework_P.Gyro_Car_Y0;
854
855     /* End of SystemInitialize for SubSystem: '<Root>/acc_gyro_read' */
856
857     /* SystemInitialize for Enabled SubSystem: '<Root>/controller' */
858     /* InitializeConditions for Delay: '<S3>/Delay3' */
859     framework_DW.Delay3_DSTATE = framework_P.Delay3_InitialCondition;
860
861     /* InitializeConditions for UnitDelay: '<S9>/UD' */
862     framework_DW.UD_DSTATE = framework_P.Difference_ICPrevInput;
863
864     /* SystemInitialize for MATLAB Function: '<S3>/MATLAB Function' */
865     framework_DW.P[0] = 0.0;
866     framework_DW.P[1] = 0.0;
867     framework_DW.P[2] = 0.0;
868     framework_DW.P[3] = 0.0;
869     framework_DW.x[0] = 0.0;
870     framework_DW.x[1] = 0.0;
871
872     /* SystemInitialize for Outport: '<S3>/PWM' */
873     framework_B.Add = framework_P.PWM_Y0;
874
875     /* End of SystemInitialize for SubSystem: '<Root>/controller' */
876 }
877
878 /* Model terminate function */
879 void framework_terminate(void)
880 {
881     /* Terminate for Enabled SubSystem: '<Root>/One_time_initialization' */
882     framework_I2CWrite4_Term(&framework_DW.I2CWrite);
883     framework_I2CWrite4_Term(&framework_DW.I2CWrite1);
884     framework_I2CWrite4_Term(&framework_DW.I2CWrite2);

```

```

885     framework_I2CWrite4_Term(&framework_DW.I2CWrite3);
886
887     /* Terminate for MATLABSystem: '<S1>/I2C Write4' */
888     framework_I2CWrite4_Term(&framework_DW.I2CWrite4);
889
890     /* Terminate for MATLABSystem: '<S6>/Digital Output' */
891     matlabCodegenHandle_matlabCod_f(&framework_DW.obj_g);
892
893     /* Terminate for MATLABSystem: '<S5>/Digital Output' */
894     matlabCodegenHandle_matlabCod_f(&framework_DW.obj_lq);
895
896     /* End of Terminate for SubSystem: '<Root>/One_time_initialization' */
897
898     /* Terminate for Enabled SubSystem: '<Root>/acc_gyro_read' */
899     /* Terminate for MATLABSystem: '<S2>/I2C Read' */
900     matlabCodegenHandle_matlabC_fex(&framework_DW.obj);
901
902     /* End of Terminate for SubSystem: '<Root>/acc_gyro_read' */
903
904     /* Terminate for Enabled SubSystem: '<Root>/motor_controller' */
905     /* Terminate for MATLABSystem: '<S11>/Digital Output' */
906     matlabCodegenHandle_matlabCod_f(&framework_DW.obj_l);
907
908     /* Terminate for MATLABSystem: '<S12>/Digital Output' */
909     matlabCodegenHandle_matlabCod_f(&framework_DW.obj_na);
910
911     /* Terminate for MATLABSystem: '<S13>/Digital Output' */
912     matlabCodegenHandle_matlabCod_f(&framework_DW.obj_d);
913
914     /* Terminate for MATLABSystem: '<S14>/Digital Output' */
915     matlabCodegenHandle_matlabCod_f(&framework_DW.obj_j);
916
917     /* Terminate for MATLABSystem: '<S4>/PWM' */
918     matlabCodegenHandle_matlab_fex2(&framework_DW.obj_h);
919
920     /* Terminate for MATLABSystem: '<S4>/PWM1' */
921     matlabCodegenHandle_matlab_fex2(&framework_DW.obj_n);
922
923     /* End of Terminate for SubSystem: '<Root>/motor_controller' */
924 }

```

List of Figures

1	graphical description of the model	3
2	Non linear continuous model in simulink	4
3	discrete time integrator settings	5
4	Non linear discrete model in simulink	6
5	Non linear discrete model simulation with zero force applied	6
6	Non linear discrete model simulation with an offset step in the angle	7
7	pole and zero map of the continuous systems	12
8	pole and zero map of the discrete systems	12
9	zoomed in pole and zero map of the discrete systems	13
10	linear and non linear version of the discrete system	14
11	simulation result of linear and non linear version of the discrete system	14
12	Kalman filter test model	15
13	Kalman filter test results	16
14	Kalman filter for simulation	16
15	linearized continuous system simulation setup	17
16	linearized continuous system simulation results	17
17	linearized discrete system simulation setup	17
18	linearized discrete system simulation results	18
19	The initialization	19
20	Processing of the read datas	20
21	Kalman Filter and PID Controller	21
22	Converts the PWM input in direction of rotation and speedo	22
23	The whole structure consist of blocks from the previous sections	23