FH | JOANNEUM
Elektronik und
Computer Engineering

BalanBot

Model-Based-Design

Knapp Christopher

07.03.2020

Table of Content

1 Task Description

The end goal of this assignment was to program a BalanBot so that it could balance itself and counteract small forces applied to it. To better understand the System, a model of the robot had to be created and tested. The findings of this exercise should help to configure and control the BalanBot properly.

2 Model Development

2.1 Deriving the System Equations

In the first step, the four base equations have been derived from the system.

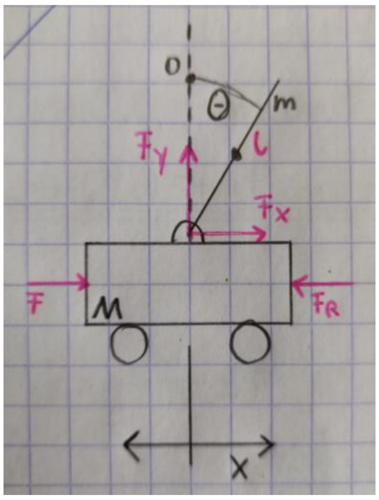
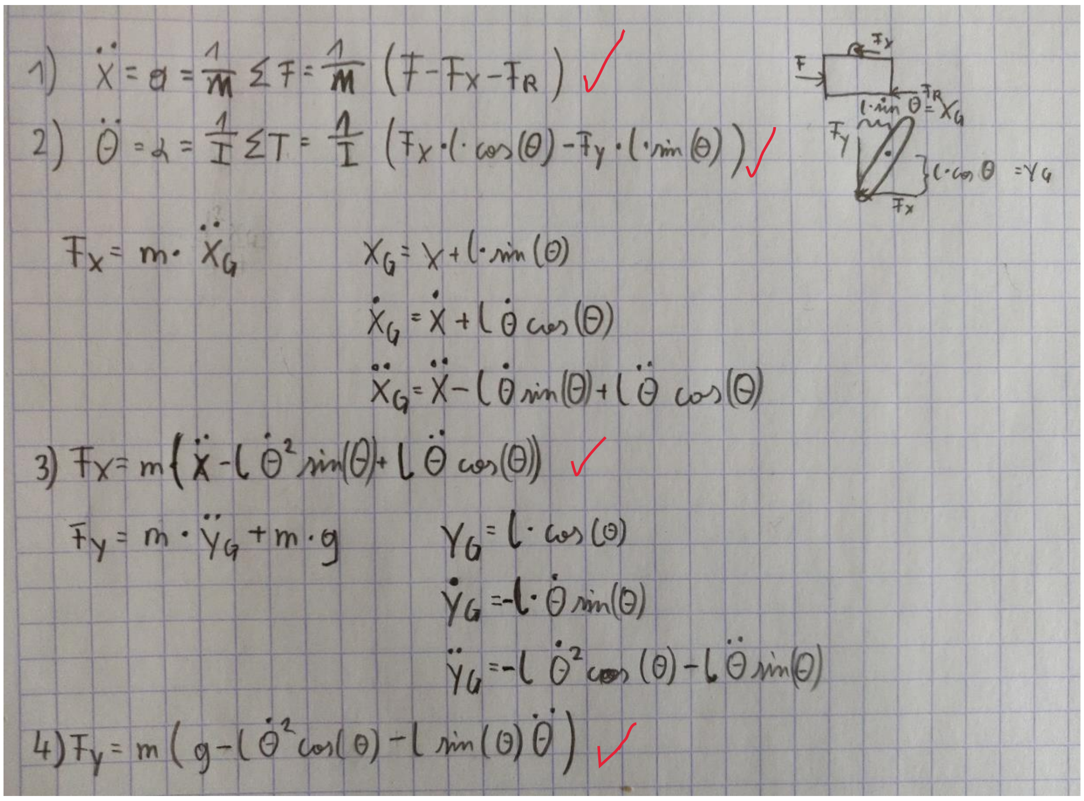


Figure 1: sketch of the forces in the system



1) $\ddot{x} = a = \frac{1}{M}\sum F = \frac{1}{M}(F - F_x - F_R)$ ✓

2) $\ddot{\theta} = \alpha = \frac{1}{I}\sum T = \frac{1}{I}(F_x \cdot l \cdot \cos(\theta) - F_y \cdot l \cdot \sin(\theta))$ ✓

$F_x = m \cdot \ddot{x}_G$ $\quad x_G = x + l \cdot \sin(\theta)$

$\dot{x}_G = \dot{x} + l\dot{\theta}\cos(\theta)$

$\ddot{x}_G = \ddot{x} - l\dot{\theta}\sin(\theta) + l\ddot{\theta}\cos(\theta)$

3) $F_x = m(\ddot{x} - l\dot{\theta}^2\sin(\theta) + l\ddot{\theta}\cos(\theta))$ ✓

$F_y = m \cdot \ddot{y}_G + m \cdot g$ $\quad y_G = l \cdot \cos(\theta)$

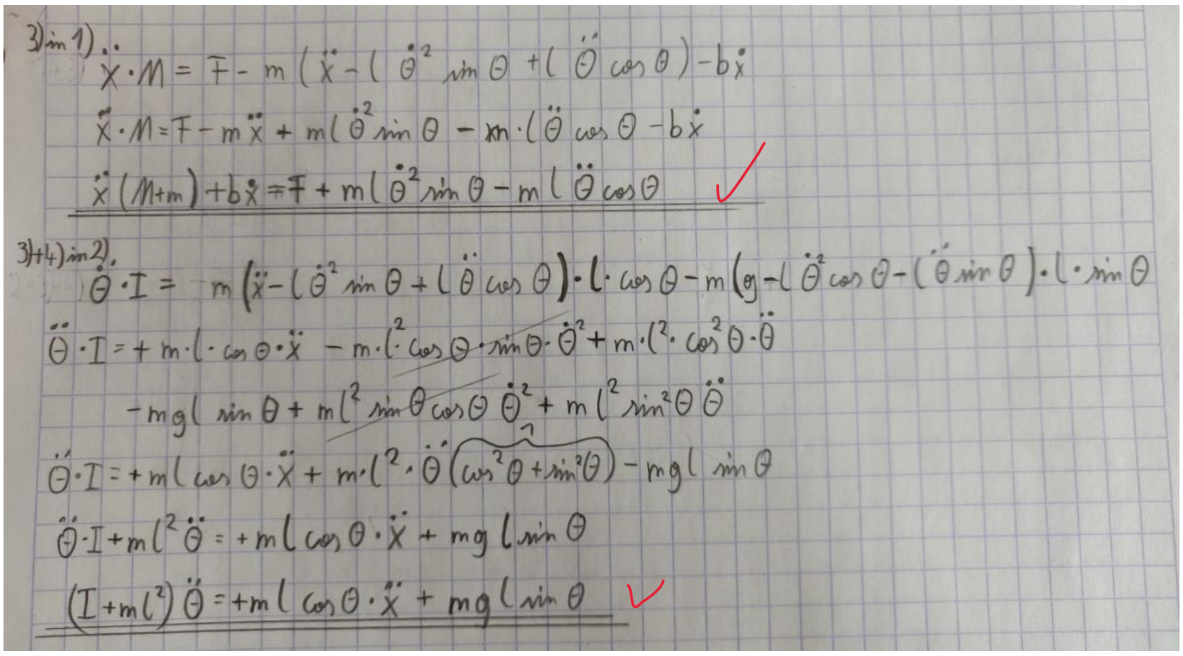
$\dot{y}_G = -l \cdot \dot{\theta}\sin(\theta)$

$\ddot{y}_G = -l\dot{\theta}^2\cos(\theta) - l\ddot{\theta}\sin(\theta)$

4) $F_y = m(g - l\dot{\theta}^2\cos(\theta) - l\sin(\theta)\ddot{\theta})$ ✓

Figure 2: base equations of the system

After that, the four base equations have been merged into two. One to describe the behavior of the angle, and one to describe the behavior of the position. ✓



3) in 1)

$$\ddot{x}\cdot M = F - m(\ddot{x} - l\,\dot{\theta}^2 \sin\theta + l\,\ddot{\theta}\cos\theta) - b\dot{x}$$

$$\ddot{x}\cdot M = F - m\ddot{x} + m\,l\,\dot{\theta}^2\sin\theta - m\cdot l\,\ddot{\theta}\cos\theta - b\dot{x}$$

$$\underline{\underline{\ddot{x}(M+m) + b\dot{x} = F + m\,l\,\dot{\theta}^2\sin\theta - m\,l\,\ddot{\theta}\cos\theta}}$$ ✓

3)+4) in 2)

$$\ddot{\theta}\cdot I = m(\ddot{x} - l\,\dot{\theta}^2\sin\theta + l\,\ddot{\theta}\cos\theta)\cdot l\cdot\cos\theta - m(g - l\,\ddot{\theta}^2\cos\theta - l\,\dot{\theta}\sin\theta)\cdot l\cdot\sin\theta$$

$$\ddot{\theta}\cdot I = +m\cdot l\cdot\cos\theta\cdot\ddot{x} - \cancel{m\cdot l^2\cos\theta\cdot\sin\theta\cdot\dot{\theta}^2} + m\cdot l^2\cdot\cos^2\theta\cdot\ddot{\theta}$$

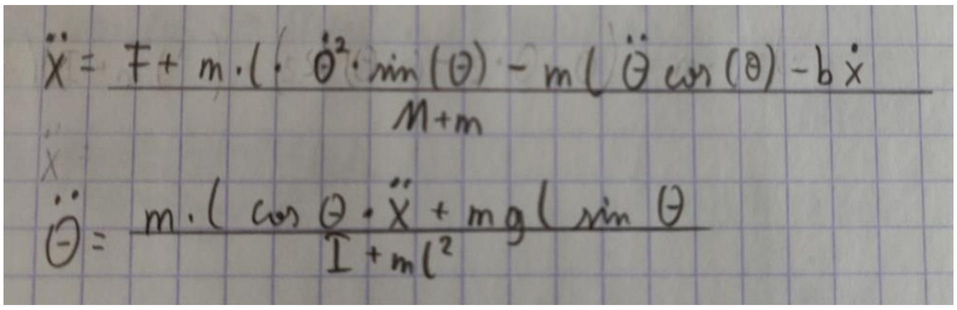
$$-mgl\sin\theta + \cancel{m\,l^2\sin\theta\cos\theta\,\dot{\theta}^2} + m\,l^2\sin^2\theta\,\ddot{\theta}$$

$$\ddot{\theta}\cdot I = +m\,l\cos\theta\cdot\ddot{x} + m\cdot l^2\cdot\ddot{\theta}\overbrace{(\cos^2\theta + \sin^2\theta)}^{1} - mgl\sin\theta$$

$$\ddot{\theta}\cdot I + m\,l^2\ddot{\theta} = +m\,l\cos\theta\cdot\ddot{x} + mg\,l\sin\theta$$

$$\underline{\underline{(I + m\,l^2)\ddot{\theta} = +m\,l\cos\theta\cdot\ddot{x} + mg\,l\sin\theta}}$$ ✓

Figure 3: merge of the base equations



$$\ddot{x} = \frac{F + m\cdot l\cdot\dot{\theta}^2\cdot\sin(\theta) - m\,l\,\ddot{\theta}\cos(\theta) - b\dot{x}}{M+m}$$

$$\ddot{\theta} = \frac{m\cdot l\cos\theta\cdot\ddot{x} + mg\,l\sin\theta}{I + m\,l^2}$$

✓

Figure 4: final equations describing the system

2.2 Implementing the Differential Equation in MATLAB/Simulink

For a better overview, the two differential equations have been implemented separately into two subsystems. The two subsystems have been placed in another one to simplify the usage of the system as a whole.

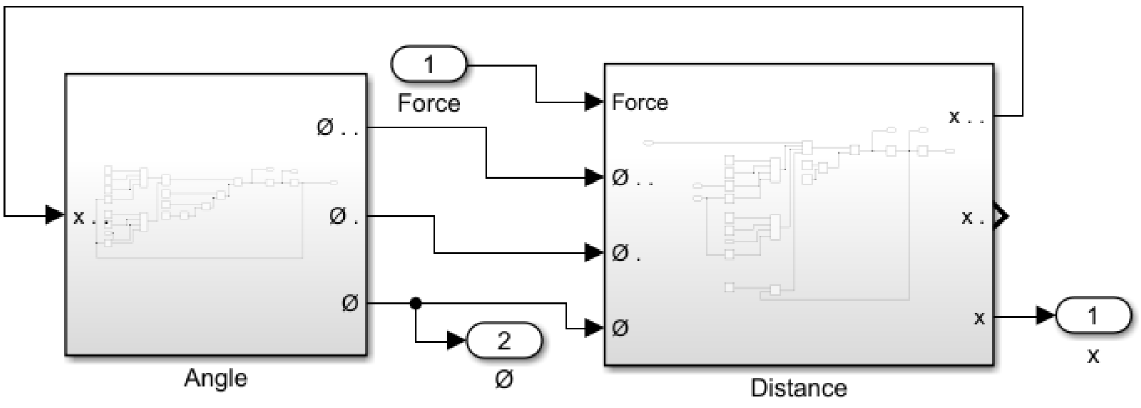


Figure 5: the system as a whole

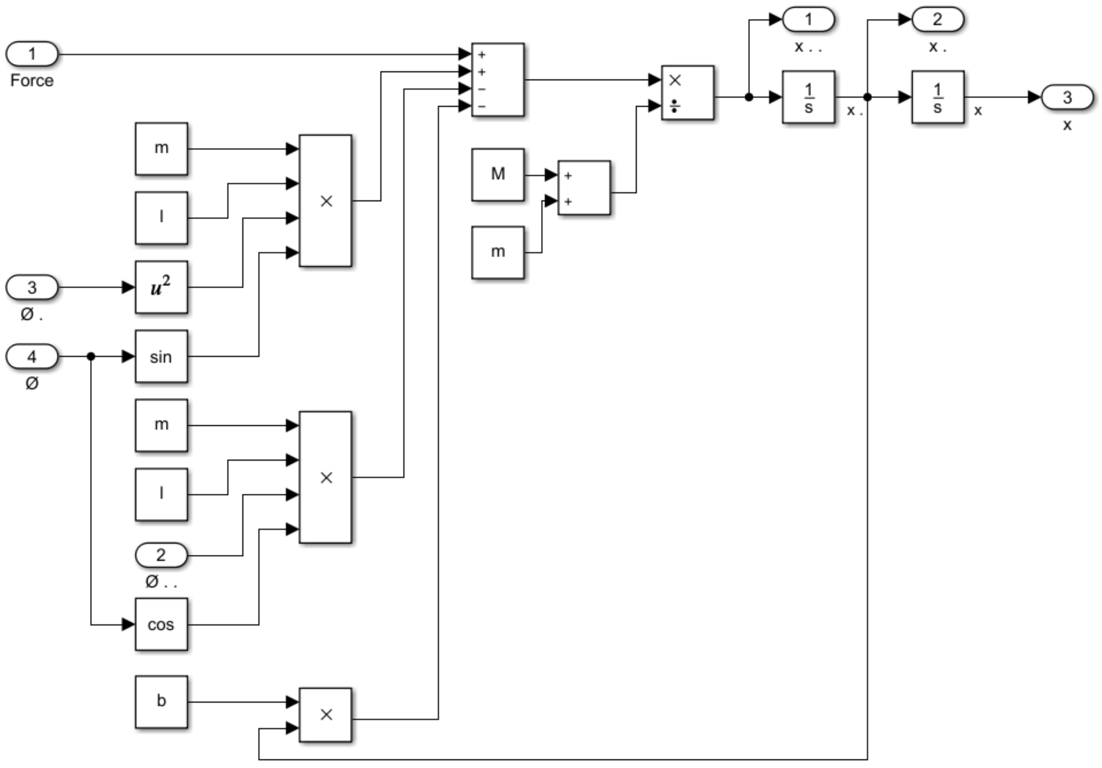


Figure 6: implementation of the Subsystem for the position equation

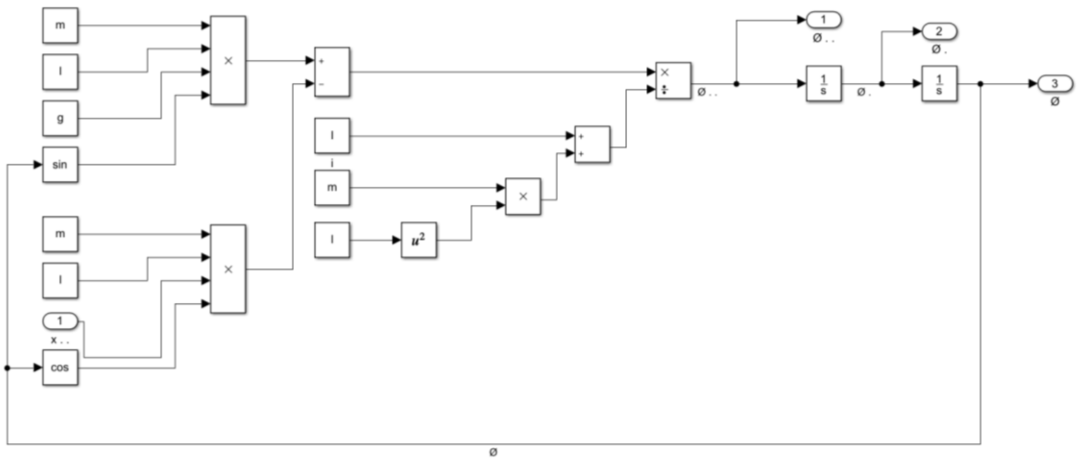


Figure 7: implementation of the subsystem for the angle equation

The values for the constants have been saved as m-file and have to be loaded into the workspace beforehand. The file is called constatnts.m and the values are as follows:

M = 0.716	mass of the cart in kg
m = 0.1756	mass of the pendulum in kg
b = 0.1	coefficient of friction for the cart in $\frac{N}{\frac{m}{s}}$
I = 0.001	mass moment of inertia of the pendulum in $kg\,m^2$
g = 9.81	gravitational acceleration in $\frac{m}{s^2}$
l = 0.11	length of the pendulum in meters

2.3 Analysis of the System

2.3.1 Open Loop Response

The first open loop response was measured with a starting angle of the pendulum of 0 degrees, a starting position of 0 meters and no force applied. In this situation, the gravitational force is perfectly centered and is stable because there are no outside factors that can tip the pendulum to either side.✓

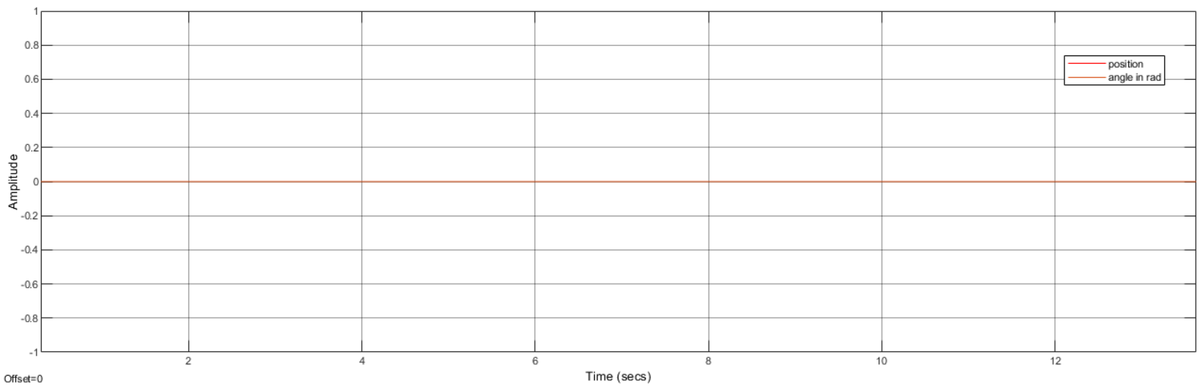


Figure 8: Open Loop resonse with 0 degrees starting angle

For the second open loop response, the starting angel had been changed to 5 degrees (0,0872 radians). This value had to be set as the starting value for the integrator that outputs the angle in the angle subsystem.

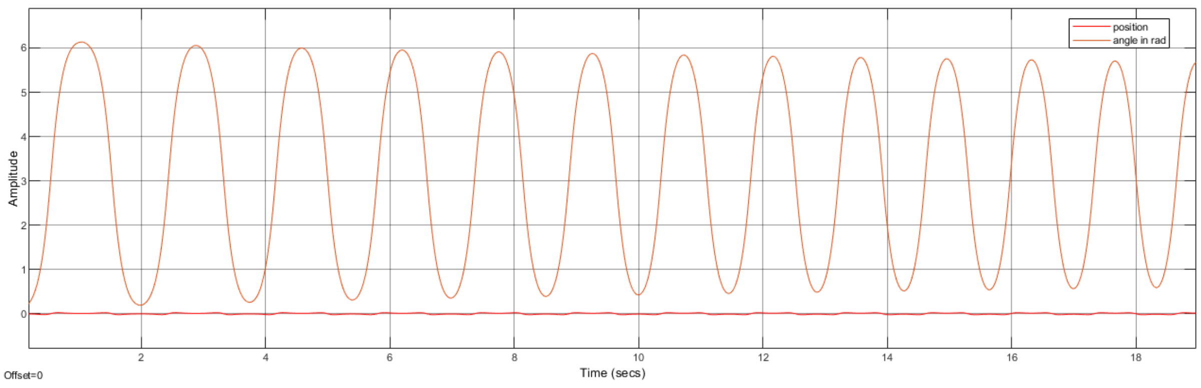


Figure 9: Open Loop response with 5 degrees starting angle visibilty of the position

Because the pendulum is out of center, it starts to accelerate in the direction of the displacement. The pendulum starts to oscillate between 0 and 360 degrees. The only thing that takes away energy from the rotation of the pendulum, is the movement in x and therefore the amplitude of the oscillation only declines very slowly. Eventually, the oscillation will settle at pi (180 degrees), which would be the position where the pendulum faces straight down.

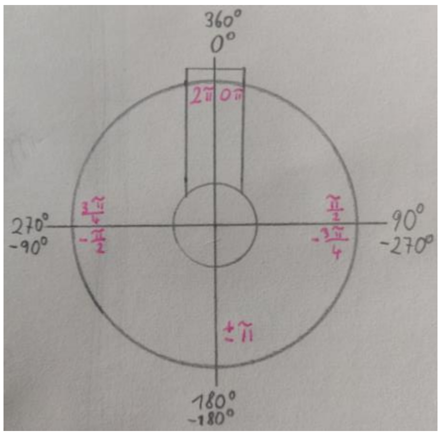


Figure 10: pendulum positions along the rotation

2.3.2 P-Controller

For my situation, I had to take negative P-Gain factors for the systems to behave as expected. I believe that I might have a sign error somewhere in my equations that I could not locate. The results of the different Kp values are as follows:

Kp	Oscillation Period (sec)	Stable?	Stable Time (sec)
-1	1,634	No	0
-2	1,462	No	0
-5	1,118	No	0
-10	1,055	No	3
-11	1,203	No	8
-12	1,143	No	15
-15	1,039	Yes	41
-17	0,985	Yes	65

As long as the system is oscillating strongly in the beginning, the P-Controller can counteract this oscillation and reduce it, but because of a steady state error, the angle does not go to zero. This causes the angle to drift away eventually. This behavior gets delayed further, the higher the gain factor is.

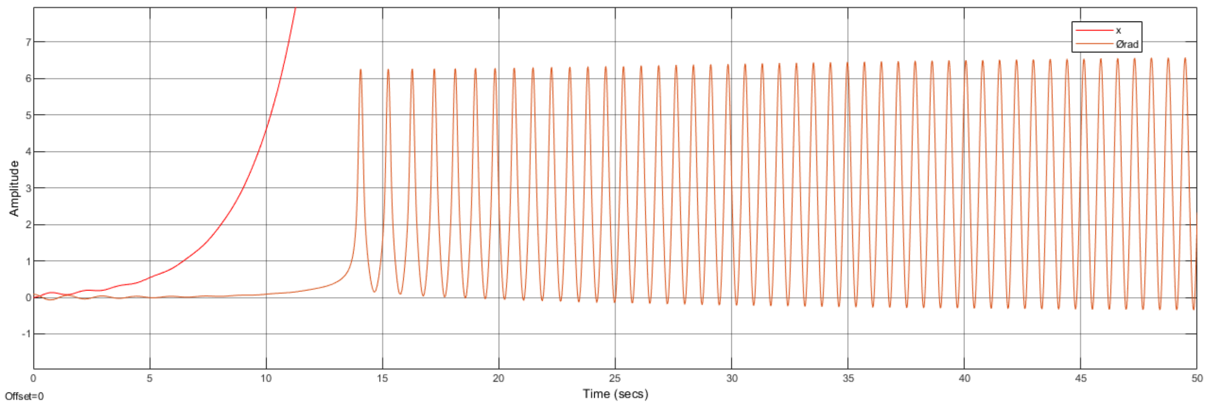


Figure 11: P-Controller with gain -11

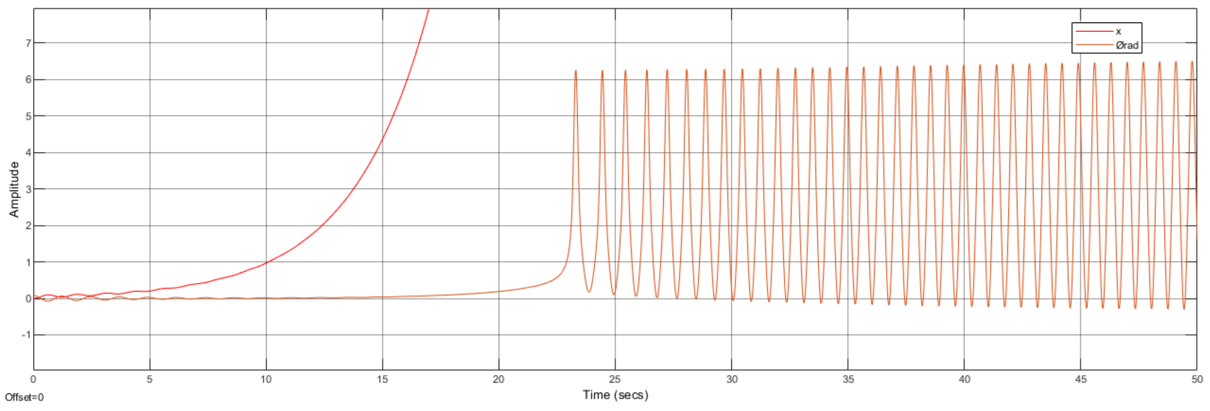


Figure 12: P-Controller with gain -12

2.4 Linearization

In the next step, the system had to be linearized to be applicable to a time invariant system. That means, that the formula is not allowed to have components such as sine or cosine operators. This is possible, because we only need the system to operate in a small range of its capabilities. In this case, the controller does not have to be able to operate on 360 degrees of the pendulum. Therefore we can linearize the system for a few degrees left and right of the center. ✓

2.4.1 Linearizing the Equation

For the linearization in the upright position, we have to make three assumptions: one for the sine, one for the cosine and one for the angle squared.

The cosine for a range of ±15 degrees (0,261799 radians) stays close to 0, so we can replace it with a 0. ✓

The sine for a range of ±15 degrees (0,261799 radians) stays very close to its value in radians, so we can use the angle in radians directly. ✓

Furthermore we can replace the squared derivative of a small angle with 0, because the value will be very small. ✓

If we now insert these three changes in the formulas in Figure 4, we get the linear system equations.

$$\cos\theta \approx 1 \qquad \dot{\theta}^2 \approx 0$$

$$\sin\theta \approx \theta$$

$$1)\ (M+m)\ddot{x} + b\cdot\dot{x} = F - ml\dot{\theta}$$ ✓

$$2)\ (I+ml^2)\cdot\ddot{\theta} = ml\ddot{x} + mgl\theta$$ ✓

Figure 13: linearization of the system equations

Transfer functions missing

2.4.2 Comparison

For this comparison, both systems, the linearized and the non-linearized one, had been injected with an offset of around 5 degrees during the first 0.1 seconds to analyze their behavior. The systems show a very similar behavior. The oscillation frequency and settling times are very similar, but the non-linear model has a slightly larger amplitude. The difference in the direction the cart is moving is most likely caused by the sign error mentioned in 2.3.2.

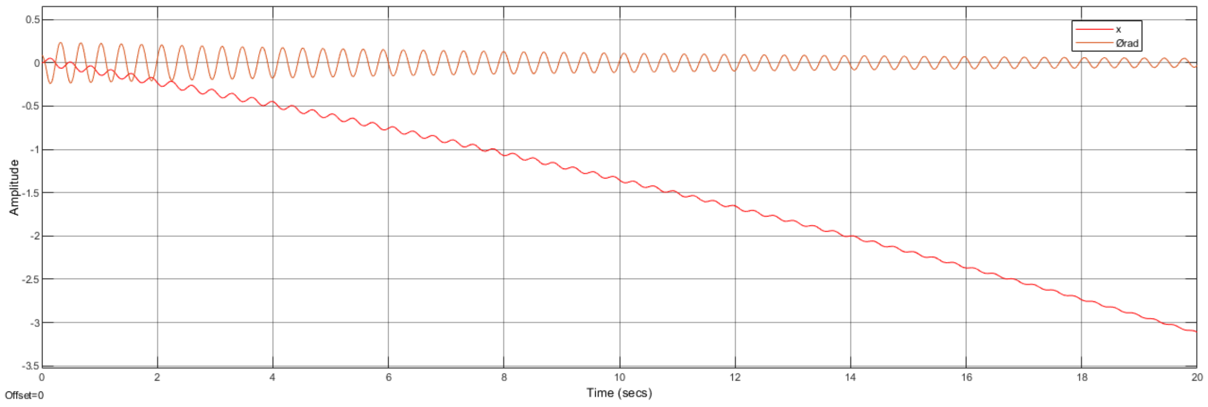


Figure 14: response of the non-linear model to a starting pulse of 5 degrees

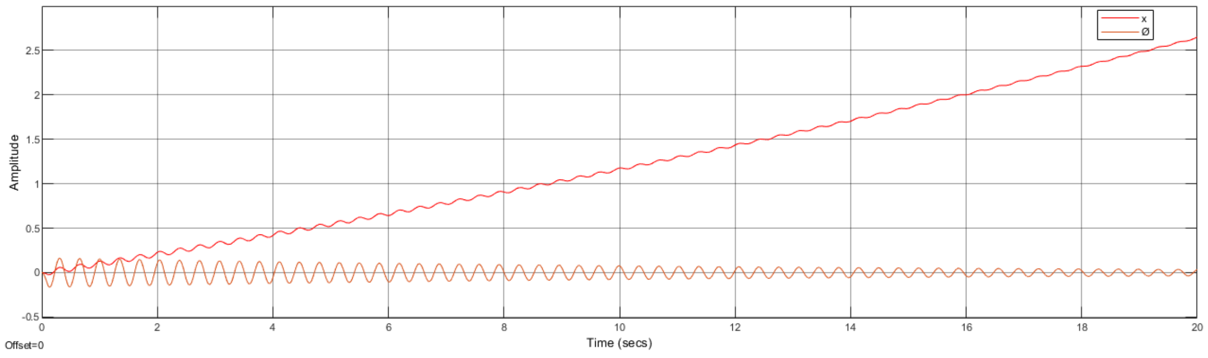


Figure 15: response of the linear model to a starting pulse of 5 degrees

To produce the pulse, a discrete impulse with an amplitude of 1 has been divided by 11.5 to get approximately a radian value of 5 degrees. This value was then added to the feedback of both systems.

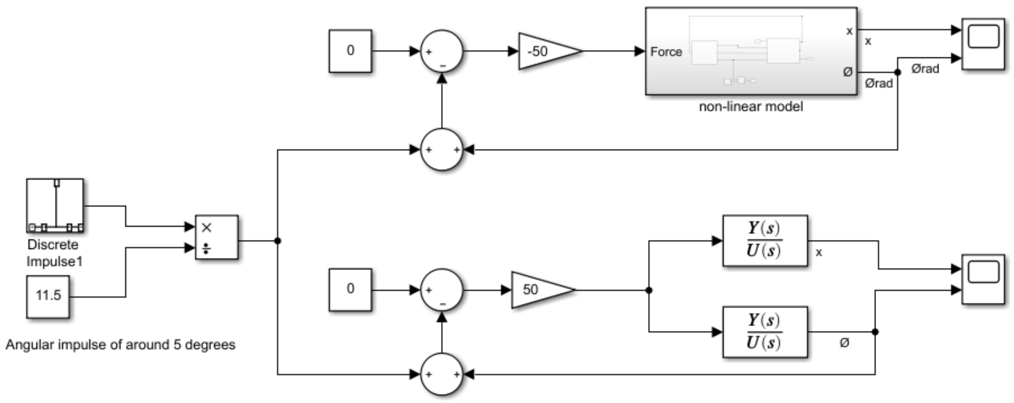
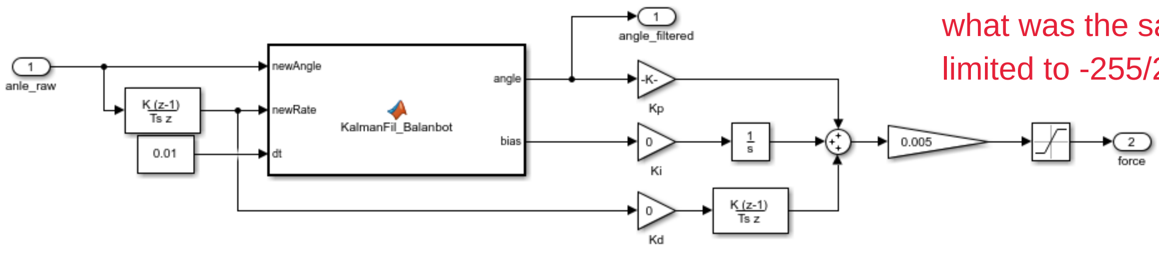


Figure 16: testing structure for a pulse in the beginning

2.5 PID-Controller

At first, the PID-Controller had been implemented manually without the usage of pre-made blocks. Every part of the PID-Controller takes a different input as described in the task assignment. This did not work for either the non-linear system or the linear one in my situation. The controller would not work if the input of the p-gain was the output of the Kalman filter. When I switched the input to the raw angle, the controller worked fine. I was not able to find out why that happened, because even if the output and the input of the Kalman filter were identical, it still would not work. Because of this, I used the raw angle as input for the p-gain for both systems.



what was the saturation limited to -255/255?

Figure 17: controller with kalman filter

not required

Kp shall be multiplied by angle (output of Kalman filter)
c. Ki shall be multiplied by the integral of angle (output of Kalman filter)
d. Kd shall be multiplied directly by newRate (input of the Kalman filter)
Note: In the hardware implementation this is a reliable signal coming from the gyroscope

2.5.1 Non-Linear Model

The proportional gain factor for a stable system after 20 seconds was around -5000 and its oscillation time about 356,375 ms. Therefore, the PID values for the Ziegler-Nichols method are as follows:

Kp = -3000

Ti = 178,187

Td = 44,547

This did not work at all for me because the angle drifts away immediately. I checked my system equations and the implementation multiple times but I was not able to get a better result.

The only working controllers I was able to achieve, were the premade blocks with autotuning and without the Kalman filter. There, the output gets stable after around 0,75 s with a constant drift of the position.

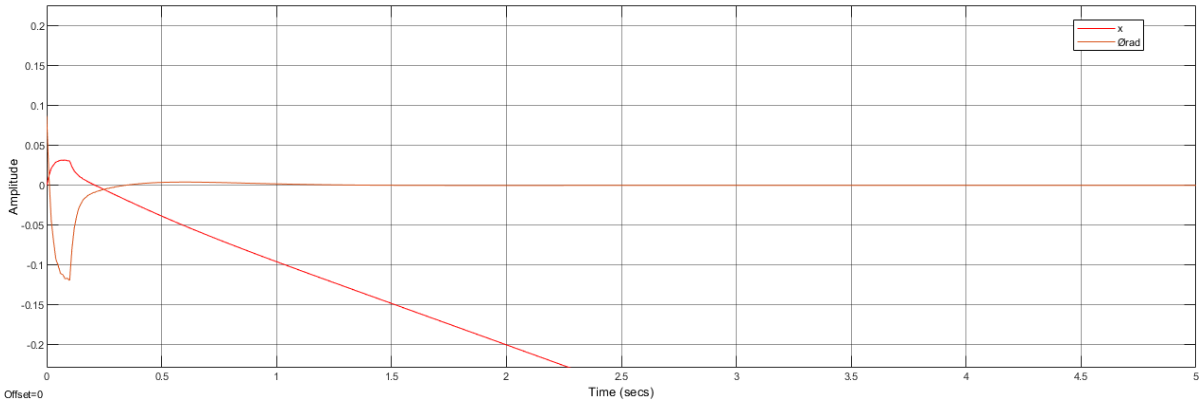


Figure 18: response of the auto tuned controller for the non-linear system

Here the angle overcorrects a bit into the negative (around 10 degrees) but recorrects itself and gets stable after about half a second. The parameters for this controller were:

Kp = -8555,1

Ki = -16107,3

Kd = -1112,0

2.5.2 Linear System

The same experience holds true for the linear system. The controller counteracts and stabilizes at around 0.4 seconds.

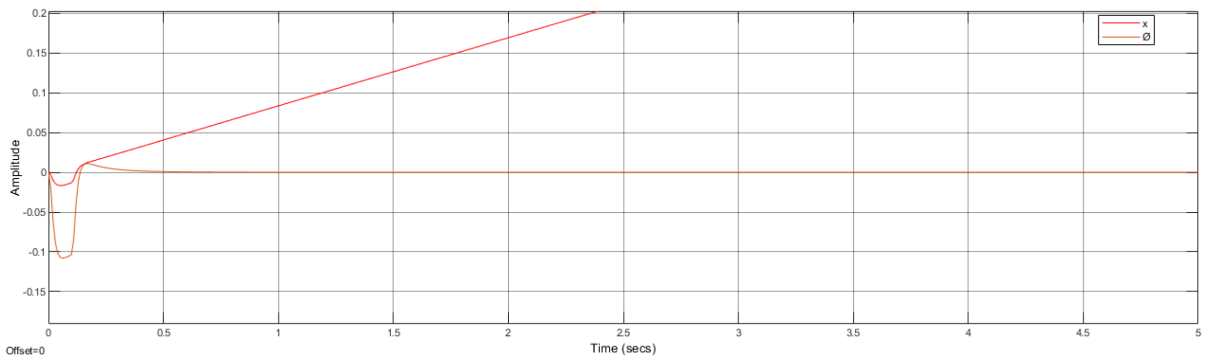


Figure 19: response of the auto tuned controller for the linear system

The parameter for this controller are as follows:

Kp = 21311,7

Ki = 66398,1

Kd = 1365,2

2.5.3 PID-Summary

I was not able to find out why my systems would not work at all with the Kalman filter. I tried different solver settings and constants but I could not achieve even close to stable result.

Hard to say, would require a analysis

Are you working with radiants or degrees?

3 BalanBot

3.1 Overview

The program for the BalanBot consists of five different Subsystems. That are triggered at different times. The triggering is handled with unit delays. This helps to achieve the staged execution needed for the system to initialize. In the first iteration, only the path after the constant is 1. This means, that no subsystems are triggered, except the initialization, because it has an inverter in front. In the second iteration, the part after the first unit delay gets high. This switches off the Initialization and enables the read MPU. In the third iteration, the last subsystem, the controller, gets enabled.

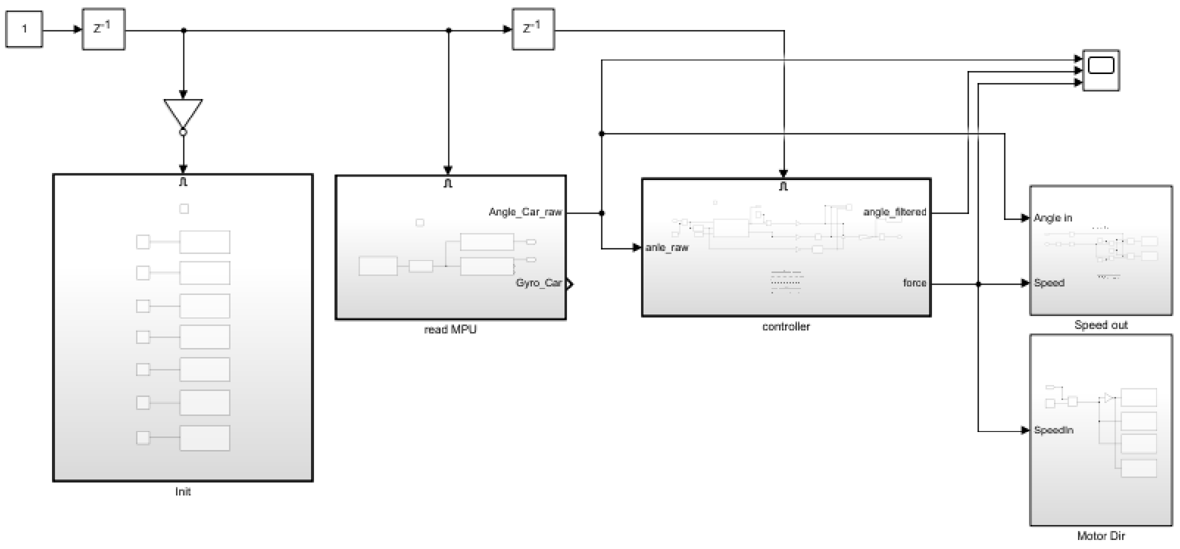


Figure 20: whole program of the balanbot

3.2 Initialization

In the initialization, all the registers of the MPU are set, so that the program can read it later. The other two blocks configure these output pins as PWM.

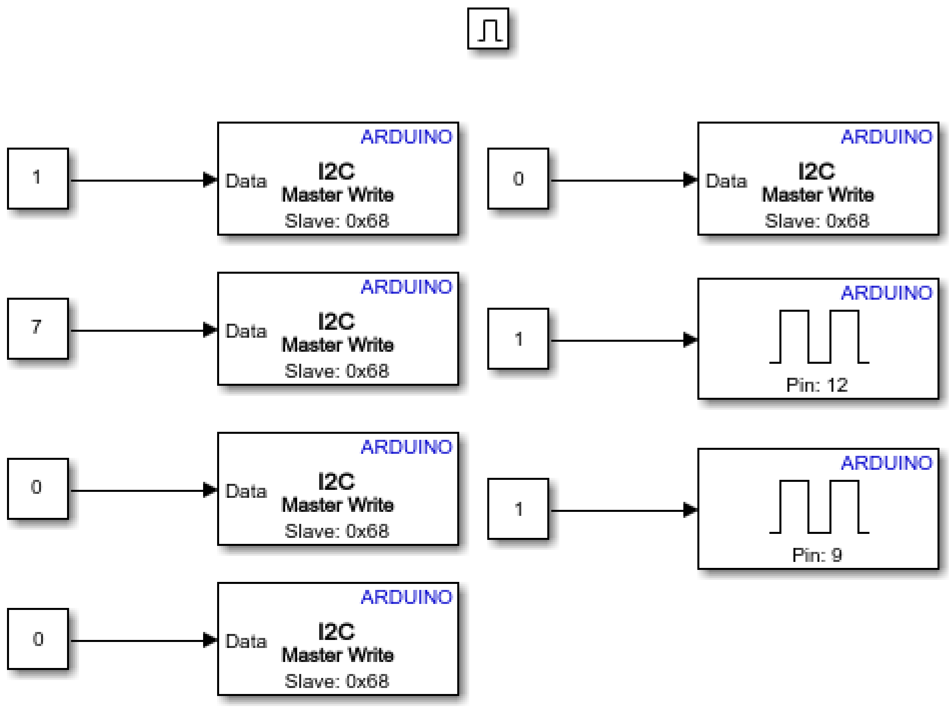


Figure 21: initialization subsystem

3.3 Read MPU

This subsystem handles the raw input from the MPU and converts it into usable values. The desirable value in this case is the roll angle.

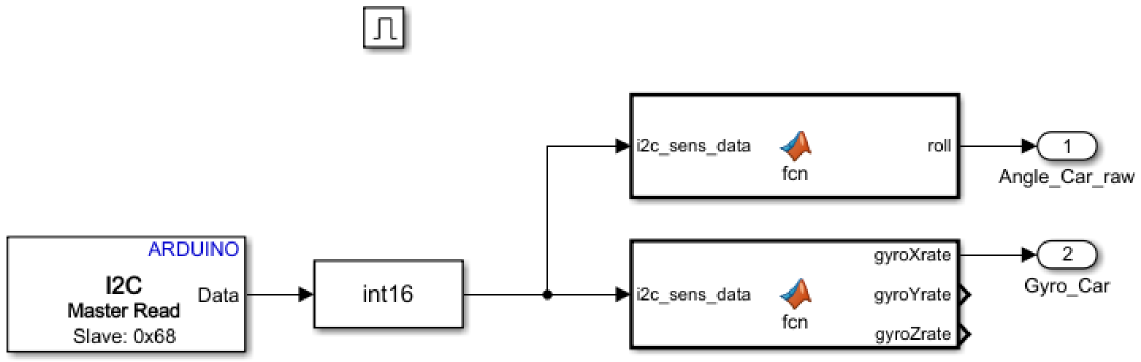


Figure 22: read MPU subsystem

3.4 Controller

The controller has two main purpose.

The first one is to filter the angle coming from the MPU. Therefore, the constant dt has to be set correctly. If the value is too high, the output will not be filtered. If the value is too low, the output will be much smoother, but it will also add a substantial delay so that the controller gets changes too slowly to react correctly. I have played around with it and found the value of 0.004 to be suitable.

The second one is to react to changes in the angle with a PID controller. The values for the gains were determined through testing. After the controller there is also a limiter which limits the output to a range of ±255, which corresponds to the maximum values the motor can take.

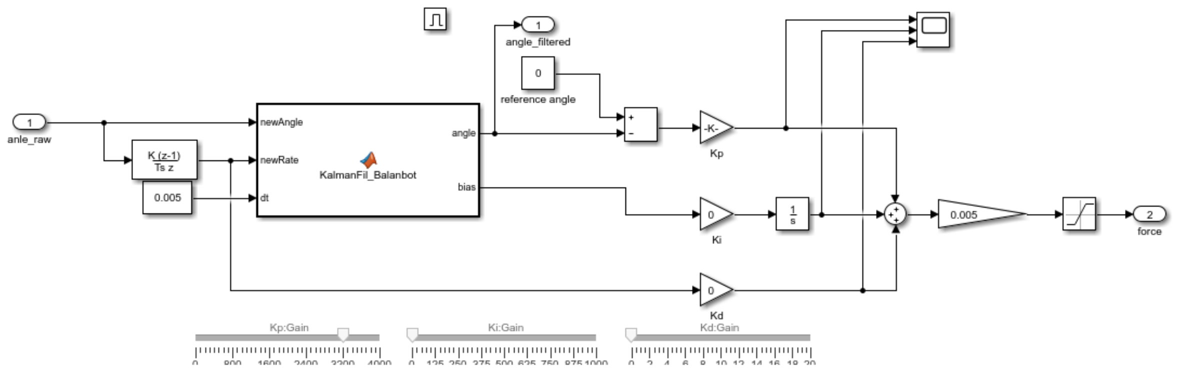


Figure 23: subsystem of the controller

3.5 Speed Output

This subsystem takes the force output of the controller and converts it to a value that can be sent to the motors. It also includes the safety mechanism that shuts off the motors if the bot tilts further than 40 degrees in either direction. I also added a feature to counteract the fact, that the motors do not start spinning at the exact same values. ✓

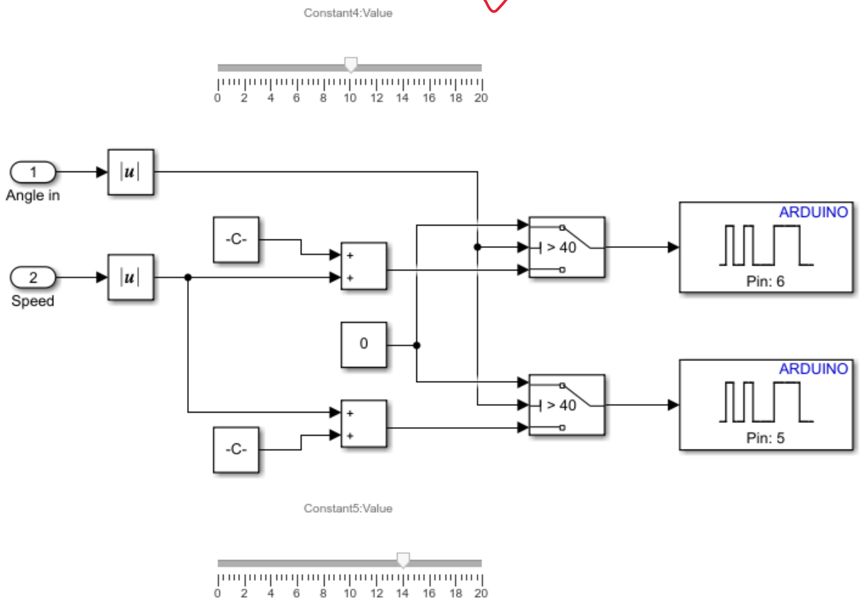


Figure 24: subsystem speed output

3.6 Direction Handler

This subsystem looks at the speed output of the controller and sets the direction of the motor accordingly.

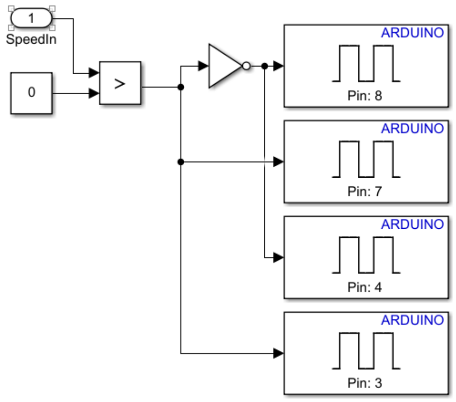


Figure 25: direction handler subsystem

3.7 Finding PID Parameters

To evaluate suitable PID parameters, I planned to work in three stages.

In the first stage, I would tune the P-Gain. This would have to be a value that has to hold the bot upright for an extended amount of time but with oscillations.

In the second stage, I would tune the D-Gain to reduce the oscillation and stabilize the bot. I also might have to up the P-Gain because of the dampening of the D-Gain.

The third stage would be to tune the I-Gain. This would be needed if the controller would not regulate to 0 degrees but to a slight offset.

In reality, it was looking very different. The first step was achievable but after a few oscillations the bot would fall. I also noticed, that the bot reacted differently on either direction and so the bot always fell to the same side, regardless on where it fell the first time. I therefore tried to correct the angle by a degree but that did not change the situation. In summary, I was not able to achieve a stable balanbot. My best attempt with only the P-Gain enabled is attached in the zip of the project.

The sequence should always be first the P then the I and then D part!

There must be something wrong in your setup

5 File Index

This index lists all the files of the zip and describes what they are for.

5.1 Matlab Files

- Calc.m

  Contains the Calculation of the PID parameters for the linearized system.

- Constatnts.m

  Contains the constants needed for the model of the no-linear system.

5.2 Simulink Files

- Pulse_comparison.slx

  Contains the comparison of the linear and non-linear systems with an applied angle of five degrees during the first 0.1 seconds.

- Pid_comparison.slx

  Contains the linear and non-linear system, both with auto-tuned pid controllers.

- BalanBot_new.slx

  Contains the program which I tried to run on the hardware.

6 Table of Figures