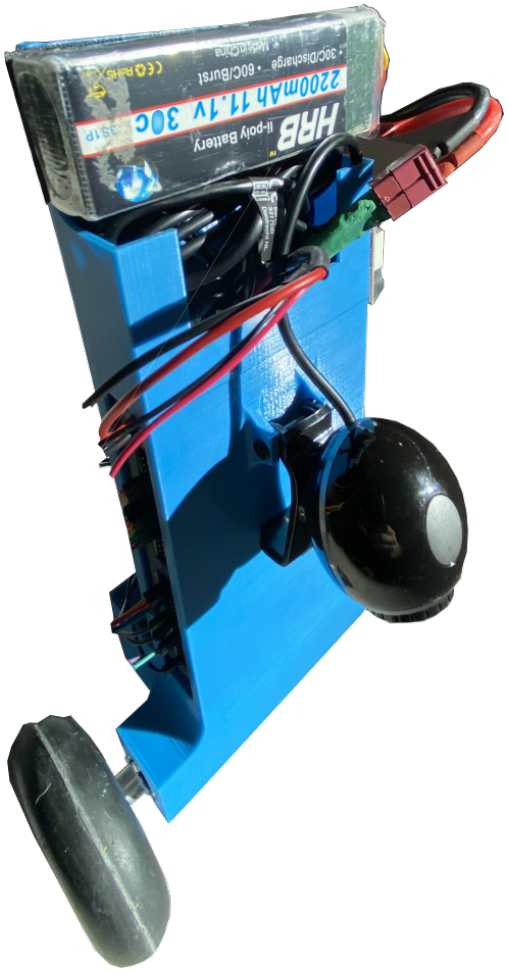
University of Agder
Norway

MAS506 - Self balancing robot



Group 4
Sindre Bokneberg,
Even Falkenberg Langås,
Jørgen Fone Pedersen

Supervisor
Morten Ottestad

University of Agder, 2020
Faculty of Engineering and Science
Department of Engineering Sciences

Abstract

In this project, a balancing robot was assembled and programmed. The system ran on a MyRIO embedded device, which was programmed in LabVIEW. Several controllers were designed and tuned to make the robot balance, to compensate for wheel friction, and to control the position and heading of the robot. An inertial measurement unit (IMU) with an accelerometer and gyro was used to measure the pitch angle through a complementary filter. Finally, the robot was programmed to follow a black line by the use of computer vision.

Contents

List of Figures

List of Tables

1 Introduction

Many mobile robots today have the advantages of being easy to control and maneuver. However, many also suffer from being unpractical, lumpy, and in the need of much floor space to maneuver. With a two-wheeled balancing robot, the advantages are the disadvantages of ordinary designs, it can rotate on the spot, it requires less space in the environment, and in this project, it offers much insight into control theory. Due to its unstable nature as an inverted pendulum, the balancing act requires sensor data, filtering, actuation, and locomotion to be able to stand straight.

The result will be a balancing robot following a flat path on the floor using camera vision with MyRIO as the embedded device and LabVIEW as the software development tool. In this project, the implementation of these variables with control and odometry of a propulsion system, sensor fusion from accelerometer and gyro including filtering, control system of balancing act and path following from camera vision are to be discovered. This report is combined from four previous exercises, each focusing on the parts mentioned.

At first, the preparation for a highly accurate control system, with an emphasis on velocity control and accuracy of the motors, including the positioning of the robot using odometry was acquired. With this, compensation for friction in the drive system and dead reckoning was utilized.

Secondly, the focus was on implementing the IMU for sensing the pitch and yaw angles of the robot. These angles are crucial information for enabling the self-balancing act of the robot. The IMU on the robot consists of the built-in accelerometer on the MyRIO and a Digilent PmodGYRO breakout board.

The third and most comprehensive exercise focused on enabling the robot to balance autonomously and the control of the robot's linear velocity and heading. Either a cascade controller, state feedback, or a simple PID controller may be used to get the robot balancing. The implementation of the controller scheme will be done using LabVIEW to program the MyRIO. The robot's linear velocity and heading shall be able to be controlled using the LabVIEW VI.

Lastly, the implementation of the camera vision system to control the robots heading based on the tracking of black lines taped to the floor was done. The lines on the floor are reference feedback to be fed into the controller with an eye-in-hand configuration by observing the relative position of the nearby line.

The four exercises are here combined by first summarizing the equipment and connections used, followed by a theory part focusing on signals, control, and camera vision. The method chapter goes through all aspects and solutions of the robot and its control.

2 Equipment and Connections

The robot was assembled with the given equipment. Each motor was connected with 4 machine screws to the chassis and wired to a common PCB. The PCB and the motor controller breakout board were then connected, then to the MyRIO and the battery.

The given equipment are listed below.

- 1x MyRIO
- 2x Electric motors with gearbox and encoder
- 1x Motor controller
- 1x Gyro
- 1x 3s 11.1V 2200mAh LiPo battery
- 1x LiPo voltage meter and alarm
- 1x LiPo Charger
- 1x Trust exis Webcam
- 2x Wheels with shaft connection
- Various cables and connectors

The connections between the motors, motor PCB, motor controller, and MyRIO are illustrated in Figure 2.1. The DC power source is the battery connected to both the motor controller and MyRIO with a self-made wire harness. The webcam was connected to the USB-input on MyRIO.

As opposed to the exercise description, the connections between the gyro and MyRIO were moved to the B-side. As the motor connections were using pins 11 and 13 on the A-side, the equal B-side pins were used for the gyro. The connections can be seen in Table 2.1.

MyRIO B-pins	**Pin description**	**Gyro pins**
33	(+3.3V)	Voltage supply (+3.3V)
30	GND	Ground
34	I2C.SDA	Serial data (SDA)
32	I2C.SCL	Serial data (SCL)
11	DIO0	Interrupt 2
13	DIO1	Interrupt 1

Table 2.1: *MyRIO and gyro connections*

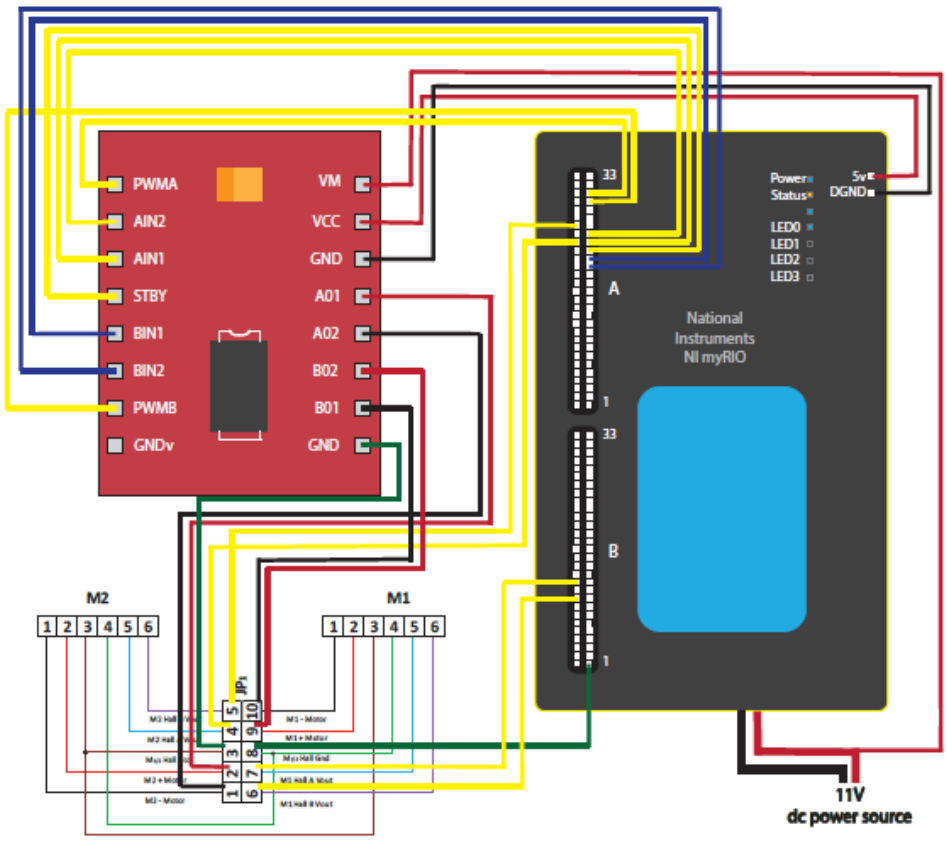


Figure 2.1: Connections for the robot

3 Theory

3.1 Pulse Width Modulation

Pulse width modulation (PWM) is a digital approach to reduce the average power of an electrical signal. It uses rectangular-shaped pulses produced with a given frequency. The length of the pulse divided by the time cycle is called a duty cycle, see Figure 3.1. This decides the average power produced. PWM is used to control the input to the motors.

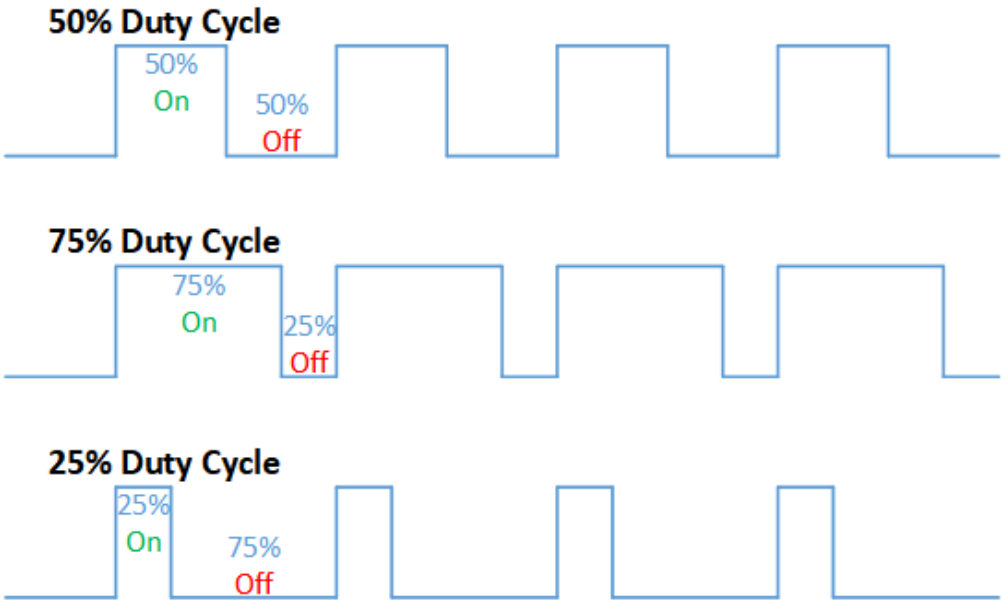


Figure 3.1: *PWM example with different duty cycles.[1]*

3.2 Low pass filter

A low pass filter is a filter that only lets the low-frequency signals pass and thereby filtering out higher frequency disturbance and noise in a signal. The advantage is a more readable and clean signal making the controller more stable. On the other hand, it causes a lag in the signal resulting in a less responsive system. It is therefore important to find a reasonable trade-off between stability and delay.

3.3 Odometry

To decide the position of a two-wheeled robot, odometry can be used. Odometry uses the data from motion sensors to estimate the change in position over time by knowing the rotation and the radius of the wheel. Figure 3.2 shows some parameters describing the change in the position of a robot.

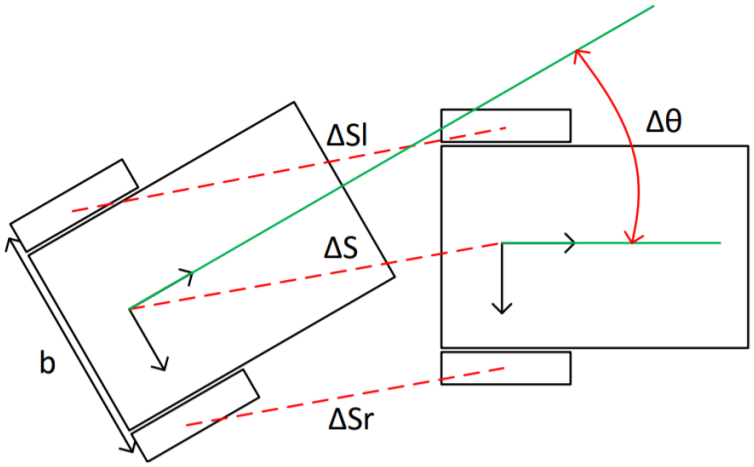


Figure 3.2: *Illustration of the robot changing position*

The equations used to derive the new position of the robot is given in equations (3.1-3.3).

$$\Delta s = \frac{\Delta s_r + \Delta s_l}{2} \tag{3.1}$$

$$\Delta\theta = \frac{\Delta s_r - \Delta s_l}{b} \tag{3.2}$$

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \Delta s \cdot cos\left(\theta_{t-1} + \frac{\Delta\theta}{2}\right) \\ \Delta s \cdot sin\left(\theta_{t-1} + \frac{\Delta\theta}{2}\right) \\ \Delta\theta \end{bmatrix} \tag{3.3}$$

Where:

Parameter	Description	Value	Unit
Δs	Change in position of the center point of the robot	-	m
Δs_r	Change in position of the right wheel	-	m
Δs_l	Change in position of the left wheel	-	m
$\Delta\theta$	Change in direction of the robot	-	rad
b	Width between the wheels	0.188	m
x_t	x-position of the robot at time t	-	m
y_t	y-position of the robot at time t	-	m
θ_t	Heading of the robot at time t	-	rad

3.4 Euler angles, roll and pitch

Euler angles describe the orientation of an object with the help of three angles, roll, pitch, and yaw. With the help of an accelerometer, roll and pitch can be calculated from trigonometric relationships between the acceleration vectors. This relationship relies on acceleration due to gravity which will always give a constant acceleration in one direction. This is also why yaw angle cannot be computed by this relationship as the rotation will always be around the gravitational axis. The

angles are calculated with the accelerometer values as shown in equations (3.4-3.5). For the self-balancing robot, the pitch angle is how much the robot leans forward or backward with respect to the wheelbase, the yaw angle is the horizontal direction the robot is pointing. The roll is not of importance for this robot as the surface it will be driving on will always be flat.

$$\phi = tan^{-1}\left(\frac{a_y}{a_z}\right) \tag{3.4}$$

$$\theta = -tan^{-1}\left(\frac{a_x}{\sqrt{a_y^2 + a_z^2}}\right) \tag{3.5}$$

Where:

Parameter	Description	Value	Unit
ϕ	Calculated roll	-	rad
θ	Calculated pitch	-	rad
a_y	Acceleration in y-direction	-	g
a_z	Acceleration in z-direction	-	g
a_x	Acceleration in x-direction	-	g

3.5 Sensor fusion

To find the pitch and roll of an object, an inertial measurement unit (IMU) is often used. An IMU often consists of an accelerometer and a gyroscope. The accelerometer and the gyroscope have some flaws if they are used separately. An accelerometer is very sensitive to disturbances, thus the accelerometer data is only reliable in the long term. The gyroscope has a tendency to drift over time because it only senses changes and has no fixed reference frame, this makes it only reliable in the short term. Figure 3.3 shows the behavior of the measurements from the accelerometer and the gyroscope. The green line in Figure 3.3 is an estimation of the true angle and is based on the fact that the sensor was in an idle state at approximately 0°.

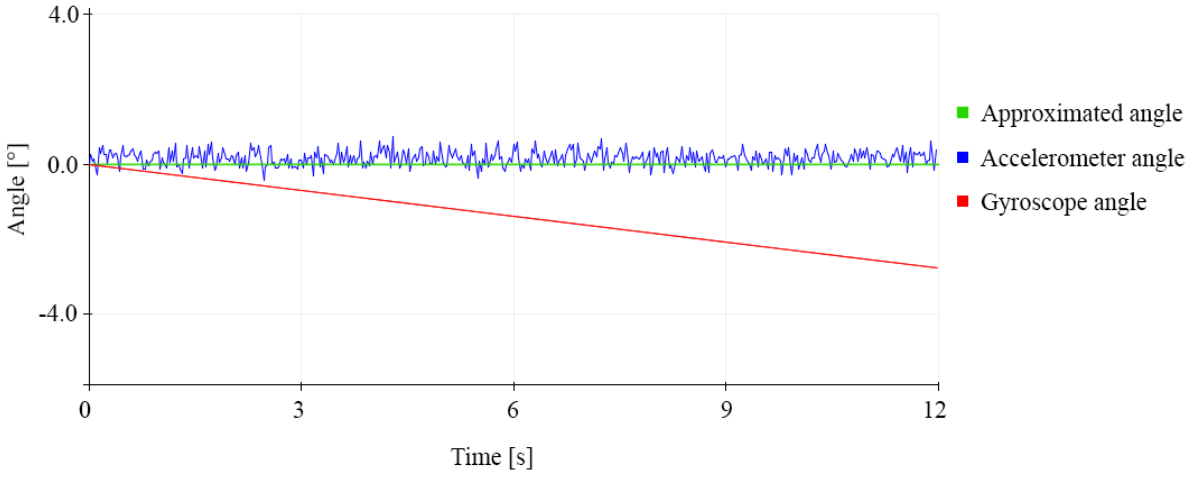


Figure 3.3: *Angle measurements from the gyro and accelerometer.*

There is a way to use the IMU while eliminating most of the issues explained above. One such method to combine the two sensors in a complimentary filter.

3.5.1 Complementary filter

A complimentary filter is a way to fuse two sensors by appreciating the advantages of each sensor and depreciating their disadvantages. For an accelerometer and gyroscope fusion, it will be based on a high pass gyroscope filter and a low pass accelerometer filter and then combine the signals to get the final filtered signal.

The greatest advantage of the complementary filter is that it is computationally simple. However, it tends to lag behind compared to some more complex filters, such as the Kalman filter. The filter is expressed in equation (3.6), the balance parameter α controls the balance between calculated and measured angle.

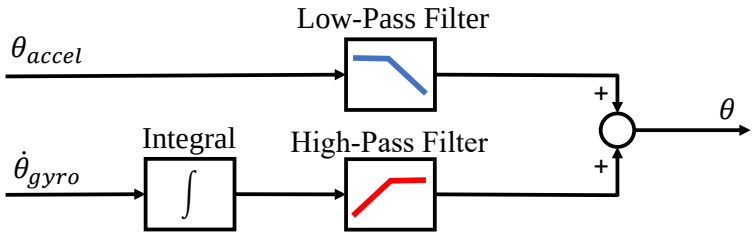


Figure 3.4: *Block diagram of a sensor fusion system*

$$\theta_n = \alpha \cdot (\theta_{n-1} + \omega \cdot dt) + (1 - \alpha) \cdot \theta_{nf} \tag{3.6}$$

Where:

Parameter	Description	Value	Unit
θ	Calculated angle	-	rad
α	Balance parameter	-	$-$
ω	Measured angular velocity	-	rad/s
θ_{nf}	Measured non-filtered angle	-	rad

3.6 Camera Vision

A digital camera creates a 2D image of the environment by converting the light which hits the camera lens to an electric charge within each pixel cell. The charge is accumulated and converted to a voltage by an amplifier and then run through an analog to digital converter which converts it to a digital number. This number is then the digital representation of the image and can be stored in a digital memory block. This image can then be manipulated by changing the pixel values or to extract various data from each pixel to identify objects, edges, brightness, or colors in the picture.

3.6.1 Grayscale image

Grayscale images in contrast to color images only contain different shades of gray, from white as the weakest to black as the strongest. It is possible to convert an image to grayscale by extracting the luminosity plane from the image. Luminosity is the intensity of light emitted from any color in the picture, this plane corresponds to the grayscale image. For an 8 bit image, each pixel is assigned a value between 0 and 255 that represents the light intensity of that pixel, resulting in 256 shades of gray where 0 is fully white and 255 is fully black.

3.6.2 Regions of interest

Regions of interest (ROI) is any region in the picture which is of particular interest for detection. This can help narrow the search and speed up computational time by not having to compute the detection algorithm over the full range of pixels.

3.6.3 Binary image

A binary image is an image that contains pixels of only two colors, usually black and white. This means that objects in the image can be differentiated from the background by comparing the luminosity of the pixels against a set threshold value. To find the optimal threshold value, for a black and white binary image, the graylevel histogram is used to calculate the optimal threshold value. This can be done using a variety of different algorithms for continuously finding the best solution in each frame or set a manual static threshold.

3.6.4 Particle Analysis

The binary image detects and differentiates background from objects in the image, but to gain useful feedback data from this image it is necessary to do some computations on the pixel locations that make up the object. To find the center of the object in either direction, the leftmost and the rightmost pixel location must be found and then divided by 2 for finding the center of the object. Many other such parameters can thus be found by doing some simple mathematical operations on the pixel locations of the object. This data can then be used for example to track objects.

4 Method

4.1 Motor velocity and direction control

The motors velocity would be controlled by a PWM signal making it possible to steplessly control the velocity of the motors. The motors would need to be programmed to be able to spin in both directions making forward and reverse motion of the robot possible. Figure 4.2 shows the LabVIEW code that was used.

For each motor, two pins on the motor controller PCB control the direction of the motor, also a common standby signal is used to turn the motors on or off. To control the direction of the motors, the input connections on the motor controller, together with the STBY connection, were connected to the MyRIO. In LabVIEW, using the MyRIO digital pin blocks the pins on the MyRIO were assigned to boolean values using a simple switch in the LabVIEW front panel. The motion of the motors was given according to the controller scheme given in Table 4.1. The switches in the LabVIEW front panel could thus be used to control the on/off power and direction of the motors.

For the PWM signal to control the velocity of the motors, the PWMA and PWMB connections on the motor controller were connected to the MyRIO. In LabVIEW, the MyRIO PWM block set to a frequency of 10 kHz was used to assign the pins, a slider from the LabVIEW front panel was used as input to the duty cycle. The slider could then be used to control the PWM signal and thus controlling the velocity of the motors.

STBY - pin 19	**AIN2 - pin 15/ BIN2 - pin 11**	**AIN1 - pin 17/ BIN1 - pin 13**	**Motor**
Low	Low	Low	Off
High	High	High	Short brake
High	High	Low	CCW
High	Low	High	CW
High	Low	Low	Stop

Table 4.1: Motor motion controller scheme

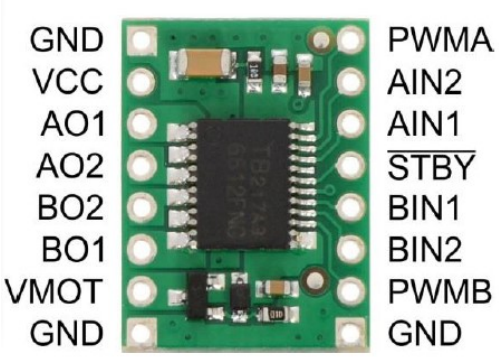


Figure 4.1: *Motor controller layout*

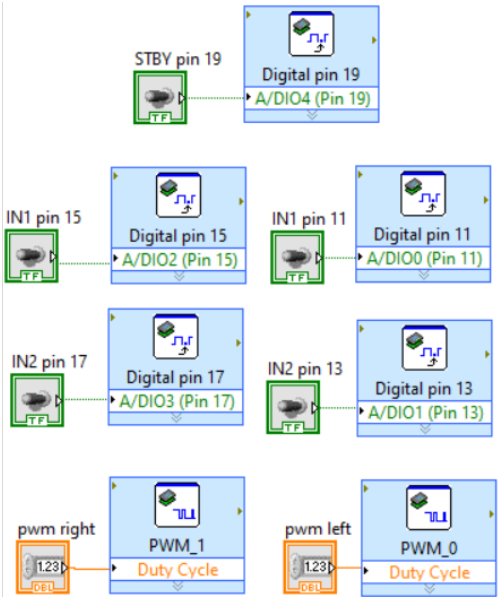


Figure 4.2: *Digital pin and PWM control in LabVIEW*

4.2 Measurement of motor angular position and velocity

Each motor was equipped with an optical encoder for measuring the motor position and from the position measurement, the velocity of the motors could be derived.

To be able to read the motor encoders the output connections of the motor controller PCB were connected to the MyRIO. The encoder block in LabVIEW was then used to be able to read the values from the encoders. The optical encoders were read at a timestep of 10 ms as stated in the exercise documentation.

The digital signal read directly from the encoders are raw data relating to how much the wheel has been turned. The encoder had 57 periods per revolution of the wheel, but because the type of encoder used was a quadrature encoder this value would be multiplied by 4 making it 228 counts per revolution. These raw data then had to be converted to a more usable notation like radians. To do this equation 4.1 was used.

$$\text{Motor position [radians]} = \frac{\text{Counter value} \cdot 2\pi}{228} \tag{4.1}$$

The motor position could then be read directly in radians in the LabVIEW front panel using a numeric indicator block.

From the motor position signal, the motor velocity could be derived using the backward difference rule. Using a sample time of 0,01s the velocity signal was derived and displayed in a numeric indicator block. The velocity signal suffered from noise and needed to be filtered to get a more stable reading. Thus a low-pass filter was used to filter the signal. The filter coefficients were then manually tuned to get a reasonable trade-off between stability and delay.

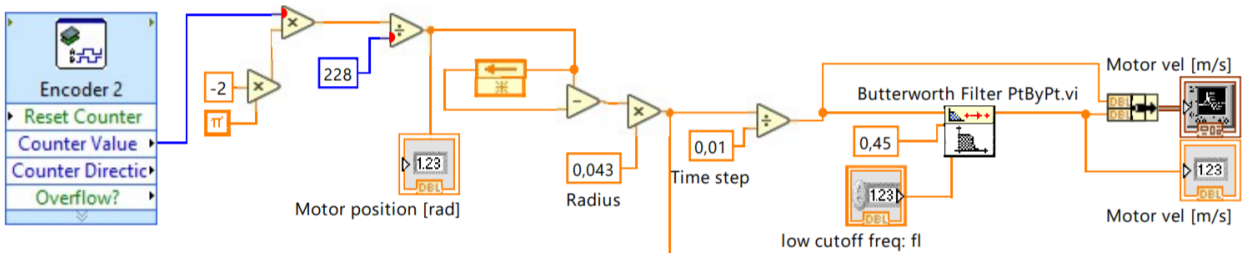


Figure 4.3: *Velocity calculations in LabVIEW*

4.3 Friction identification and compensation

The two motors used to drive the robot had some different dynamics because of the internal friction. A friction test where performed to compensate for this. The test was performed by running the motors at different PWM gains while having the wheels connected to the motors without any external contact between the wheels and the surroundings. Figure 4.4 shows the LabVIEW blocks used to implement the regression functions to the signal controlling the PWM.

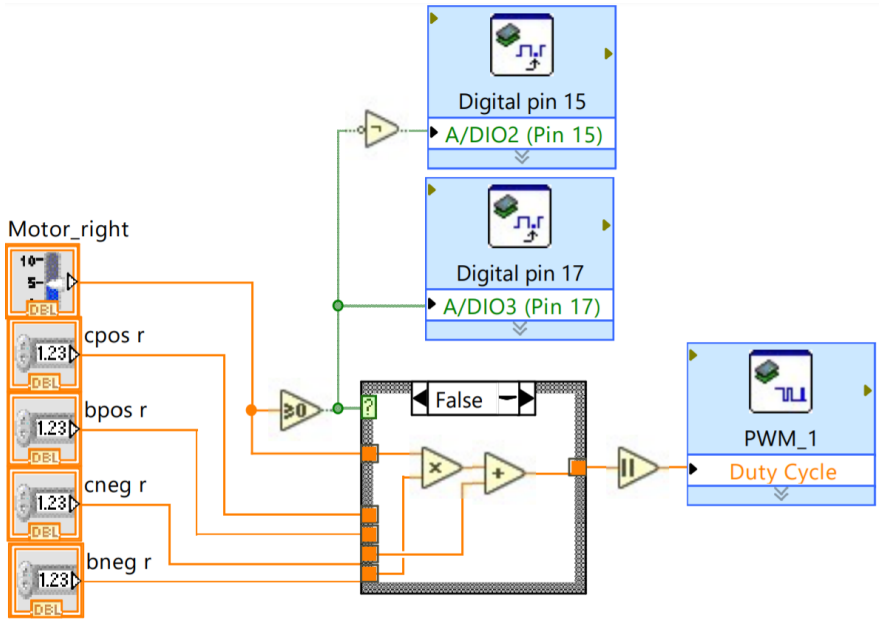


Figure 4.4: *Friction compensation by linear regression in LabVIEW*

4.4 Robot localization by odometry and dead reckoning

Odometry was used to calculate the position, heading (yaw angle) and yaw rate of the robot. The equations from chapter 3.3 was implemented in LabVIEW with a math script as seen in Figure 4.5. A first order Butterworth lowpass filter is added to the yaw rate calculations to filter out high frequency disturbances in the signal.

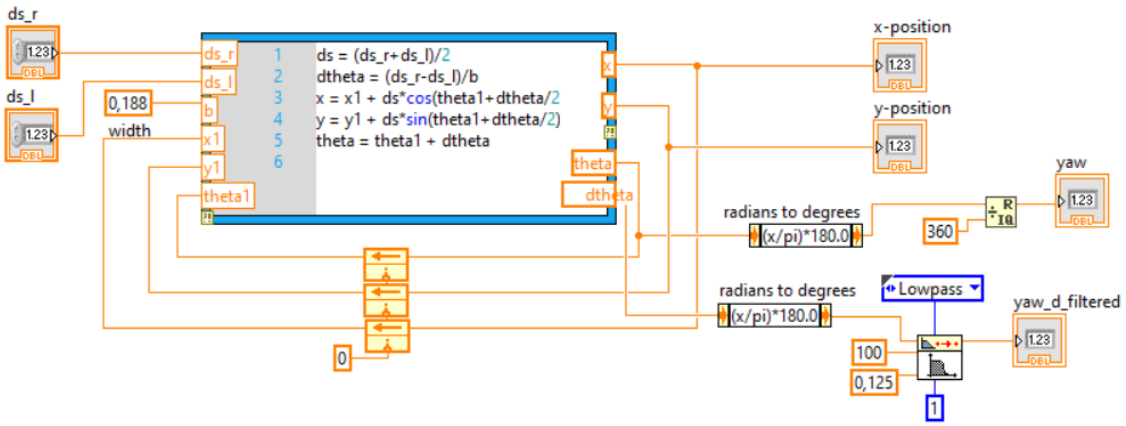


Figure 4.5: *Odometry in LabVIEW*

4.5 Accelerometer calculations

From the accelerometer data the pitch angle was calculated using the equations from chapter 3.4. The LabVIEW implementation is shown in Figure 4.6.

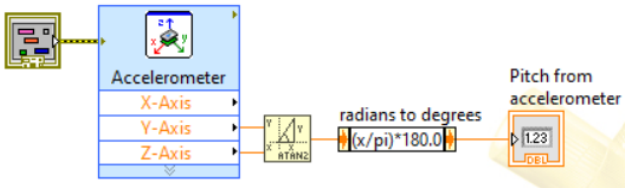


Figure 4.6: *Accelerometer calculations in LabVIEW*

4.6 Gyro installation

Before the gyro were fastened to the chassis with double-sided tape the coordinate system of the gyro had to be determined. By plotting the output of the gyro while rotating it by each axis, its coordinate system were found. The coordinate system of the gyro can be seen in Figure 4.7.

When fastening the gyro to the chassis the coordinate system of the gyro and chassis were lined up making the axes parallel. Figure 4.7 represents the two coordinate systems aligned. With the aligned coordinates, possible inverted axes was fixed by multiplying the corresponding accelerometer or gyro values with -1.

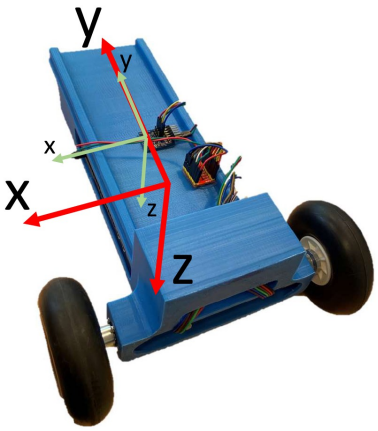


Figure 4.7: *Robot and gyro coordinate systems*

4.7 Gyroscope calculations

LabVIEW was used to get sensor data from the gyroscope. A premade LabVIEW program provided by National Instruments was used to obtain the sensor data from both the gyroscope and the accelerometer. The program had to be modified to include zero-rate level compensation, unit conversion, and sensor fusion to be used with together with the main program.

The zero rate level is the nonzero average output of the gyro when the gyro is completely still. To compensate for this, the average rate was collected over 1000 samples. This value was then added from the subsequent measurements, such that the gyro was calibrated to start non-biased. The average nonzero output used for compensation for each axis is displayed in table 4.2.

Axis	**Average nonzero output [LSB]**
x	-142
y	46
z	36

Table 4.2: Zero rate level compensation parameters

The gyro data was then converted to degrees/sec to get an intuitive representation of the data. Initially, the gyro was set to measure a maximum of 250 degrees per second (dps). This was outputted through 16 bits of information with 1 bit representing the sign. The maximum digital number the gyro could output was $2^{15} = 32768$. Therefore, the least significant bit (LSB) of the output corresponds to $250/32768 = 0.007629$ dps. The conversion done in LabVIEW is shown in Figure 4.8.

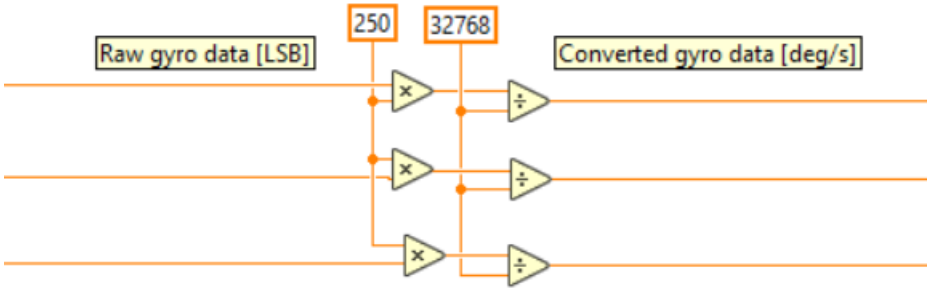


Figure 4.8: *Unit conversion in LabVIEW*

After the compensation above, the sensor data still had a bit of an offset from true angles. When changing the angle of the gyro 90°, the measured angle from the gyro showed approximately 79°. A gain of $90/79 = 1.14$ was therefore used to compensate for this offset. The properties of a gyro in a sensor fusion system are to get the low-frequency behavior of the system while the accelerometer provides the more accurate value of the angle. This means that an accurate angle measurement is not crucial from the gyro.

4.8 Centre of gravity

The target of the balancing act of an inverted pendulum is to keep the center of gravity(CoG) above the rotational point. To compensate for the error in angle offset θ, which can be seen in Figure 4.10, the rotational point is driven underneath the CoG.

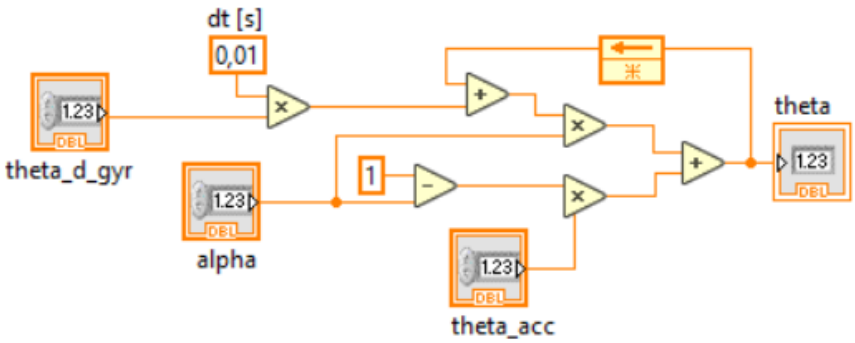


Figure 4.9: *Complementary filter in LabVIEW*

Any magnitude of θ other than the angle that correlates to the direct upright position would accelerate the angle due to the gravitational force. This is why the inverted pendulum is only a locally stable system, with its only stable position in the direct upright position. For the inverted pendulum to keep this position a control system is needed to counteract any disturbances. It is desired to keep the acceleration of the pendulum, $\ddot{\theta}$ at a minimum rate. This makes the system easier to control, requiring less input on the motors at the point of rotation.

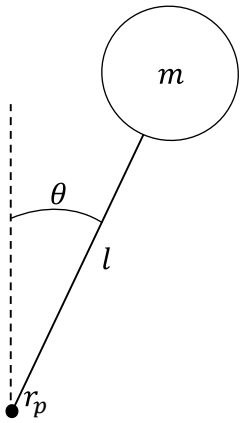


Figure 4.10: *Centre of gravity offset of rotational point*

The input on the motors is acting as a counter-balance on the system. The output on the motors is torque moving the rotational point and acting against the torque induced by gravity. The torque at the rotational point is equal to the acceleration of θ multiplied with the moment of inertia of the system, illustrated as a point mass. The correlation between the acceleration $\ddot{\theta}$ and the length l are shown in equations (4.2-4.4)[2].

$$T_{r_p} = I \cdot \ddot{\theta} \tag{4.2}$$

$$l \cdot m \cdot g \cdot sin\theta = m \cdot l^2 \cdot \ddot{\theta} \tag{4.3}$$

$$\ddot{\theta} = \frac{g}{l} \cdot sin\theta \tag{4.4}$$

where,

	Description	**Unit**
T_{r_p}	Torque at rotational point	Nm
I	Moment of inertia	kgm^2
$\ddot{\theta}$	Angular acceleration	rad/s^2
m	Point mass	kg
g	Gravitational acceleration	m/s^2
θ	Offset angle	rad

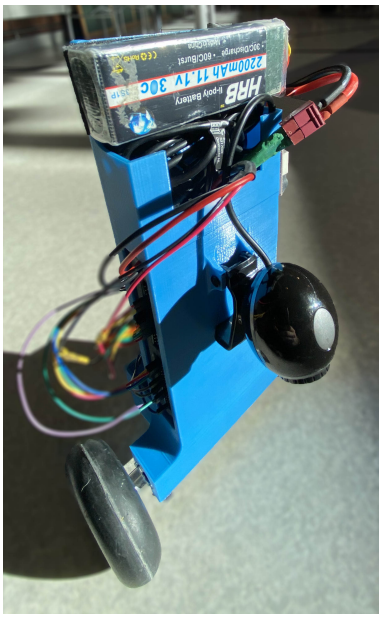


Figure 4.11: *Balancing robot with top-mounted battery*

As can be seen in equation (4.4) an increased length l from the rotational point to the point mass would decrease the angular acceleration making the system easier to control. Just as a broom it is easier to balance it with the mass at the top, i.e the CoG is higher.

For the balancing robot, the design was fixed making alternatives changing the CoG limited. The heaviest component on the robot is the battery, this is luckily also a component that can be positioned all around the robot's chassis. By placing the battery at the top of the chassis the stability increased drastically making it even more controllable. The placement of the battery on the balancing robot can be seen in Figure 4.11.

4.9 Control system

When the pitch measurements and the friction compensation was done properly, a control system could be implemented. The final controller was set up as shown in Figure 4.12. This section will explain the balance controller, the velocity controller, and the heading controller in detail.

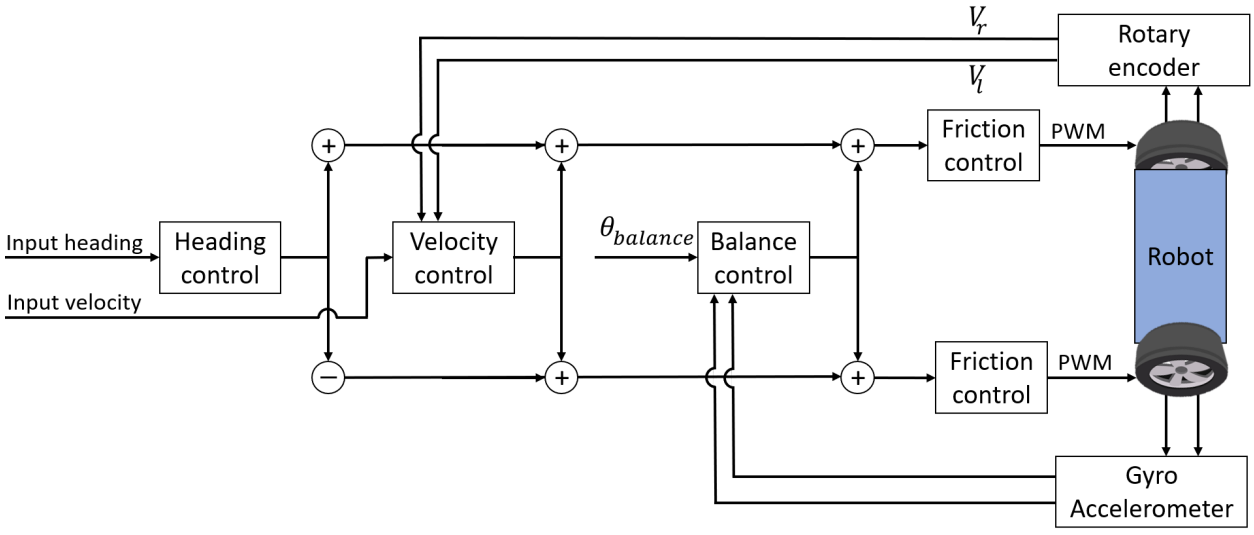


Figure 4.12: *Overview of all controllers*

4.9.1 Balance controller

To balance the robot, a cascade controller was used. The cascade controller uses the robot's pitch angle and angular velocity as inputs and gets the desired wheel velocity as output. The outer loop of the balancing controller is a full PID controller that uses feedback from the pitch angle from the complementary filter. The output of the outer controller is then used as the setpoint for the inner controller, which is a simple P controller that uses the angular pitch velocity directly from the gyroscope as feedback.

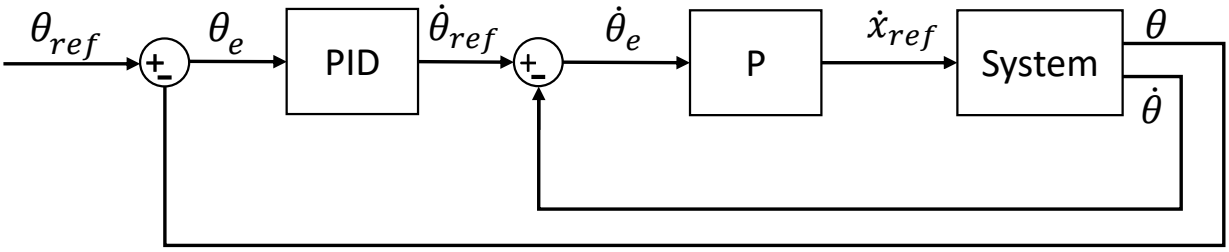


Figure 4.13: *Balancing controller*

4.9.2 Velocity controller

To enable the robot to drive forward or backward while balancing, a second controller was added. This controller determines the velocity bias in either direction independent of the balancing controller. A PI controller was used for this purpose, by adding an integral term to the velocity controller the robot would drift less. The setpoint of the controller is set in the main VI of the LabVIEW program, where positive values give forwards motion and negative gives the reverse motion. The feedback of the controller is from the velocity calculation based on the wheel encoders as seen in chapter 4.1. Figure 4.12 shows how the velocity controller interacts with the overall controller scheme.

4.9.3 Turn rate controller

A turn rate controller was implemented to enable the robot to turn while moving and balancing. This controller determines the turning bias either to the left or right independent of the balancing controller. The turn rate controller is set up with a PD controller. The input for the controller was initially a controller in the main VI used to turn the robot manually. After the camera was added the controller gained its input setpoint from the camera detection. The feedback to the controller is the turn rate that was calculated in the odometry calculations in chapter 3.3. The output of the controller was added to one wheel's velocity setpoint and subtracted from the other's as seen in Figure 4.12. This way a standstill turn should happen around the robot's axis without any significant change in x- and y-position.

4.10 LabVIEW program

The LabVIEW was built up as visualized in Figure 4.14. The initialization and gyro code was mostly from a premade example code, and therefore it will not be explained in detail. This section will cover the control system block inside the loop.

SubVIs was made to simplify the program. To be able to understand the program, these are explained in Table 4.3 and they can be seen in detail in Appendix A.

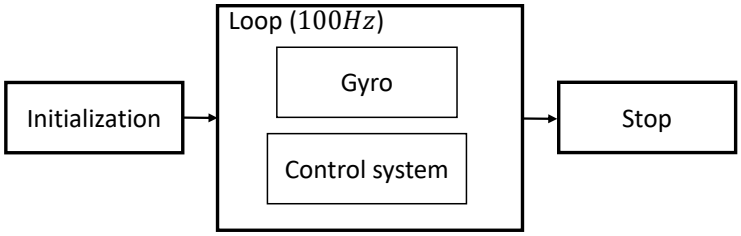


Figure 4.14: *Simplified representation of the LabVIEW control program*

Icon	Description
	Pitch calculation from the accelerometer
	The balance controller explained in chapter 4.9.1
	Complementary filter
	A fail safe to stop the motors if the angle is more than 30 °from the balance point
	Friction compensation from exercise 1
	Odometry calculations to find yaw, x- and y-position
	Calculations to find the speed from an encoder
	Turn rate controller explained in chapter 4.9.3

Table 4.3: *Custom made subVIs*

4.11 Camera detection

The camera is a Trust Exis color camera with a resolution a maximum of 640x480p. It has a wide field of view making it possible to see much of the path in front of the robot. The camera is connected to the MyRIO via USB and is sufficient to use as a vision sensor. The built-in IMAQ library in Labview is used to get the image output from the camera and for further analysis.

4.12 Image analysis in LabVIEW

The raw image provided by the camera is a color image, this needs to be converted to grayscale before the conversion to binary can take place. By using the IMAQ ExtractSingleColorPlane VI from the IMAQ library it is possible to extract the luminance plane which corresponds to the grayscale image. The IMAQ AutoBThreshold 2 VI can then be used to convert the image to a binary image for detection. Various detection algorithms can be used for detection in this block, or a manual threshold can be set. This subVI also takes input from an ROI block that creates an ROI rectangle that is modifiable from the main VI. The auto threshold block can either be set to detect dark objects or bright objects. By setting it to detect dark objects, the black line should be identified as the object and the floor as the background.

Image operations is a very processor-intensive task as a large number of computations need to be performed. The number of computations is proportional to the number of pixels in the image, which is decided by the resolution. The camera that is used for the project has several different resolutions that can be assigned through the video mode attribute from a property node in Labview. To find

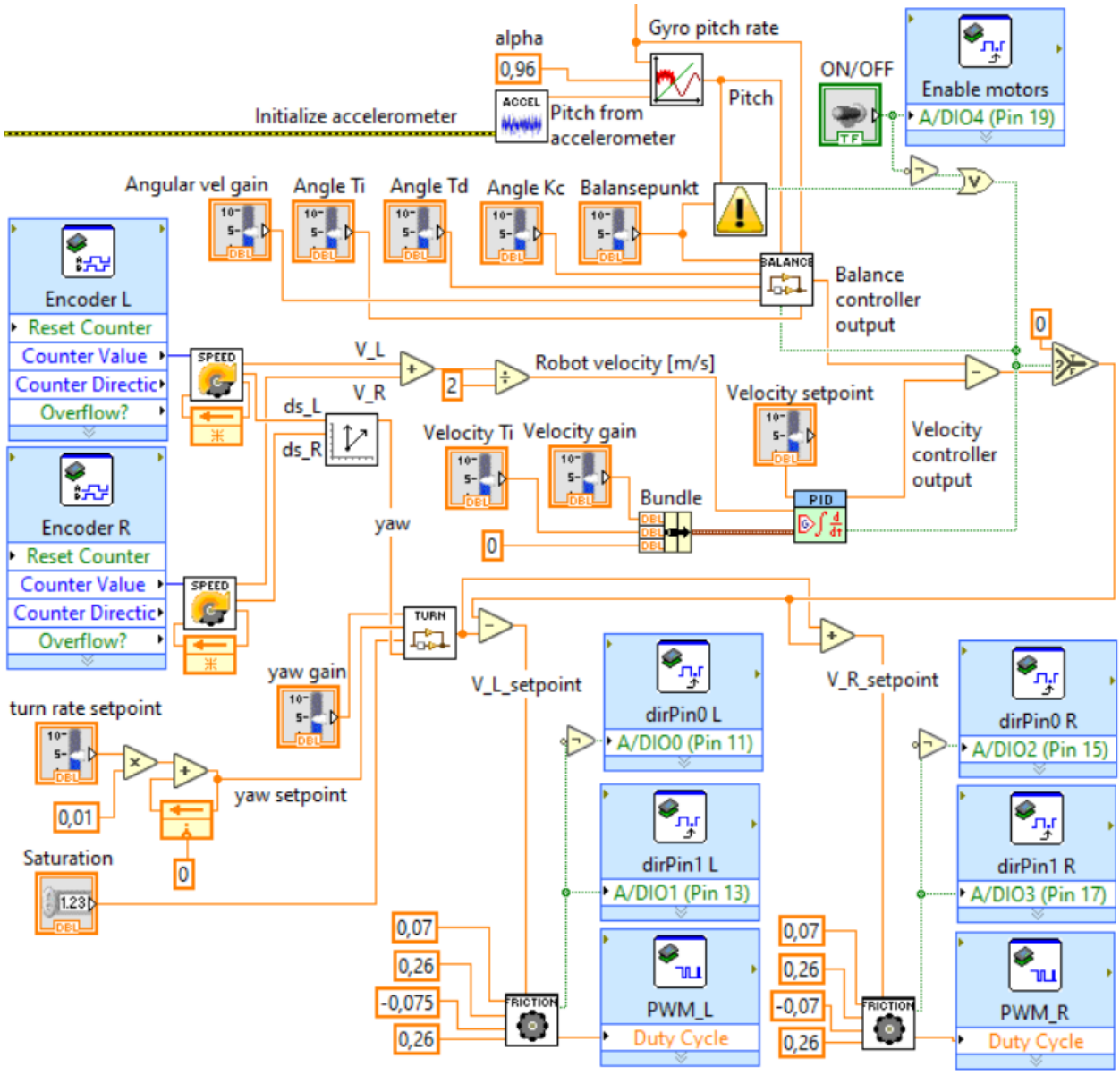


Figure 4.15: *LabVIEW Program*

the various video modes, a separate LabVIEW code was run that outputs a list of the various video modes available on the chosen camera. The various video modes of the Trust camera can be seen in Figure 4.16. To aid in computation time the resolution was set to the lowest setting of 160x120 pixels. The framerate output of the camera can also be set in the video mode property node, although this only corresponds to the camera framerate, not the framerate of the timed loop that performs the computations. These framerates should match however to optimize the efficiency of computation.

To compute the center of the detected line the IMAQ Particle Analysis VI was used and set to detect the center of mass in the horizontal direction. As the image's resolution is set to 160x120p the center of object detection needed to be subtracted a value of 80 for the feedback to the heading controller to have the value of zero in the middle of the image. This should give the heading controller a positive error when it turns to the left and a negative error when it turns to the right, thus correcting its course keeping the robot centered on the black line.

If the robot should lose the line detection during operation, a feedback loop was implemented that feeds back the previous value from the x position dataset to the turn rate controller. This gives the turn rate controller the setpoint from the last iteration before the line was lost, giving the controller an error that should correct the heading back towards the line.

The final LabVIEW program can be seen in Figure 4.17.

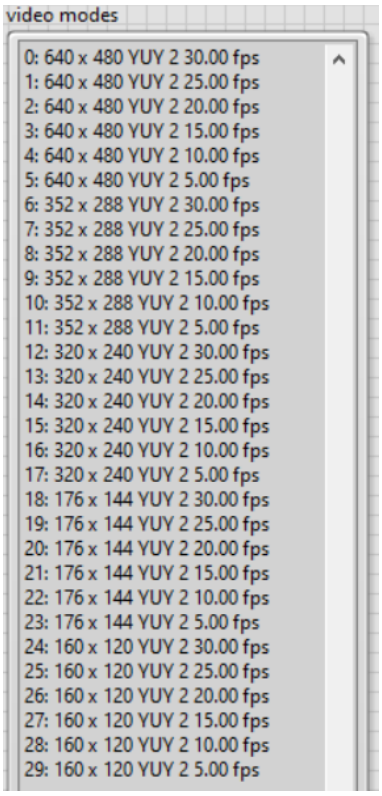


Figure 4.16: *Available video modes on the Trust camera*

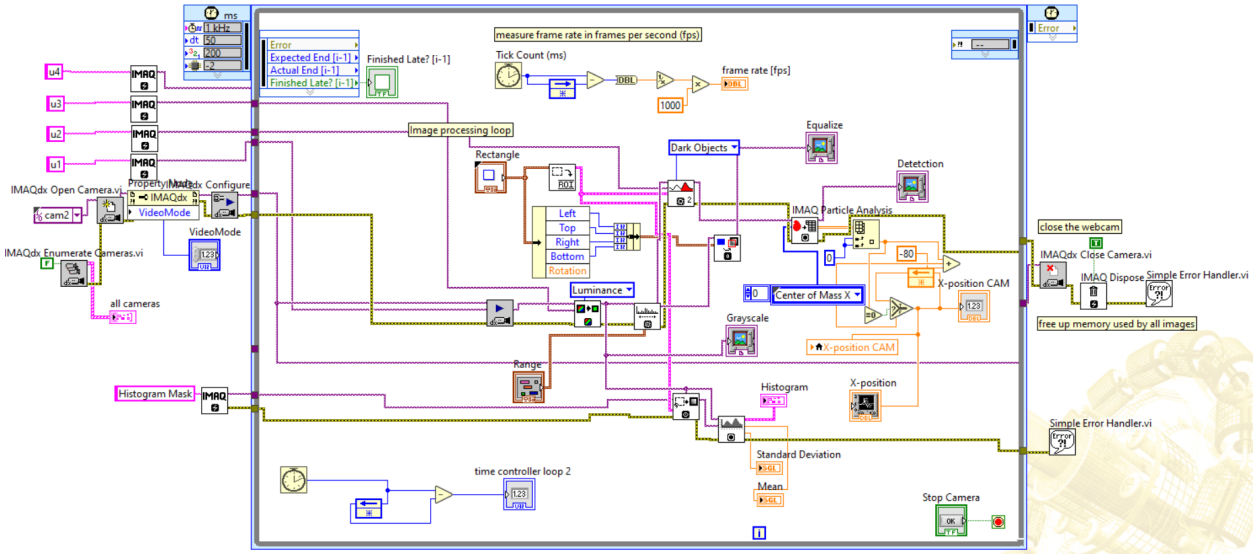


Figure 4.17: *LabVIEW camera program*

5 Results

5.1 Friction compensation

The velocity measurements from the friction identification test is shown in table 5.1. The table is plotted in Figure 5.1 where the points were plotted together with a linear regression line for both the left and right motor. Figure 5.1 shows that the right motor had more friction than the left motor. That points out the importance of tuning the two motors individually with respect to friction.

PWM input	**Left wheel velocity [m/s]**	**Right wheel velocity [m/s]**
0,9	2,95	3,05
0,7	2,18	2,29
0,5	1,34	1,56
0,3	0,6	0,79
0,2	0,23	0,4
-0,2	-0,21	-0,42
-0,3	-0,6	-0,8
-0,5	-1,38	-1,58
-0,7	-2,15	-2,3
-0,9	-2,84	-2,99

Table 5.1: *Results from the friction identification*

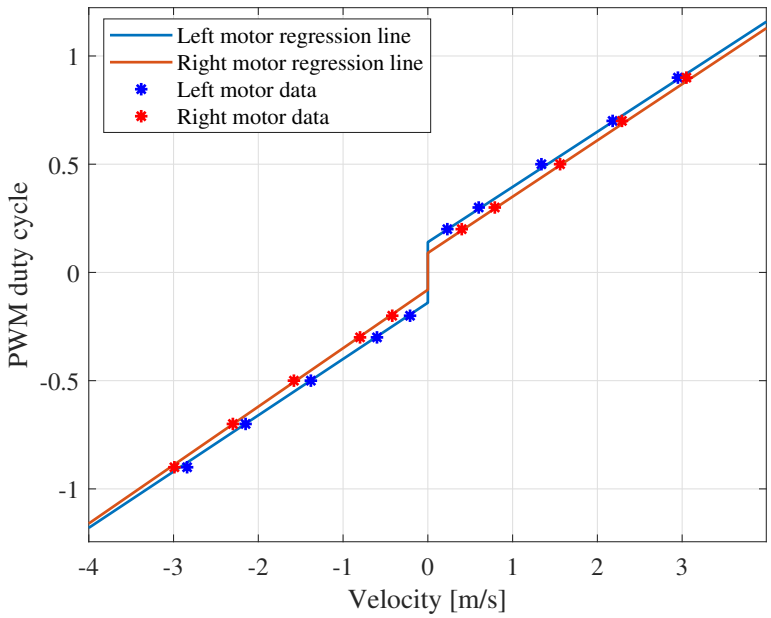


Figure 5.1: *Linear regression*

5.2 Robot localization by odometry and dead reckoning

The result was verified by measuring a length of 4 meters on the floor and driving the robot in a straight line for 4 meters. The x-value showed approximately 4 meters, and the heading was close to zero which is as expected. The heading calculations were verified by running one motor faster than the other until the robot reached a heading of 90 degrees. Results showed approximately 90 degrees as expected.

5.3 Data fusion

The results of the complementary filter output are shown in Figure 5.2 and Figure 5.3. The results are shown for a balance parameter alpha of 0.98.

Figure 5.2 shows the output when the robot was held at a constant level surface resulting in a 0-degree pitch. From this graph, one can see the stability of the filtered output.

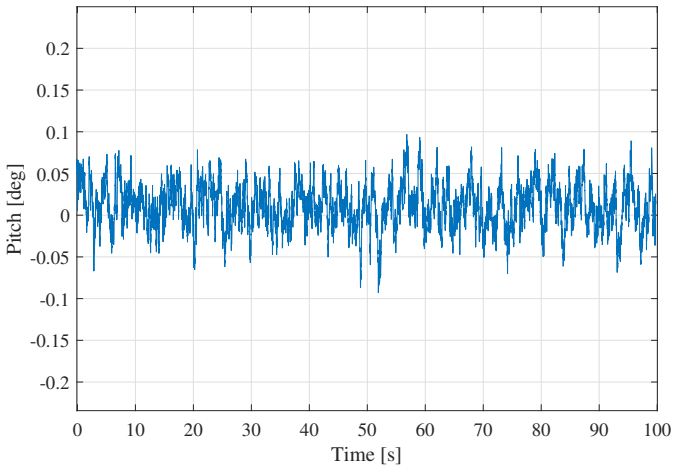


Figure 5.2: *Results*

Figure 5.3 was produced by taking the robot from 0 degrees pitch and then lifting it to stand in a vertical position. From this graph, one can see that the filtered output follows the actual pitch angle of the robot with little delay and noise.

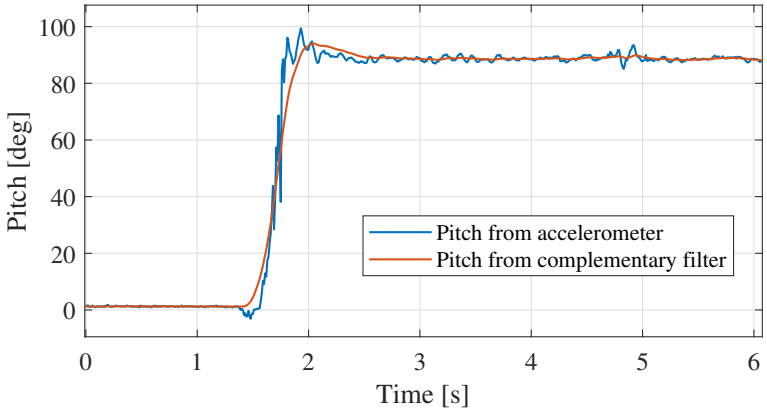


Figure 5.3: *Results from the accelerometer and complementary filter*

5.4 Two-wheeled balancing

After tuning the controllers, the robot could balance. The robot was able to keep a given velocity in both forwards and backwards direction and then it could stop by setting the velocity to 0. The same applies for turning. The maximum turning rate was tested for 180°turns which is the most that the robot will have to perform. For a 180°turn, the robot could turn at a maximum of approximately 490 [°/s] without falling over, this corresponds to ±0.8 [m/s] for each wheel, and a saturation was set to 0.8 on the output of the turn rate controller.

The pitch angle when balancing is shown in Figure 5.4.

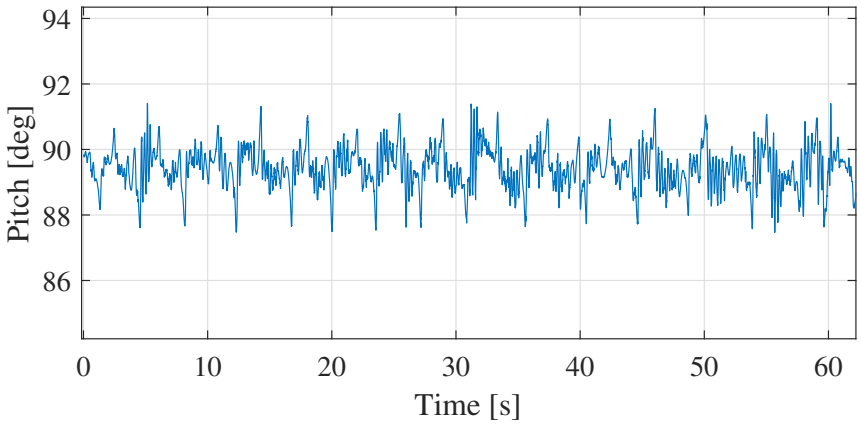


Figure 5.4: *Results of pitch while balancing*

The yaw and position results from balancing the robot without a yaw-controller are shown in Figure 5.5 and 5.6. Figure 5.5 shows that without a yaw controller, the position drifted quite a lot in both x- and y-position. The large drift in y-position is caused by the change in yaw angle which is also drifting with a value of 225 degrees after 1 minute (see Figure 5.6).

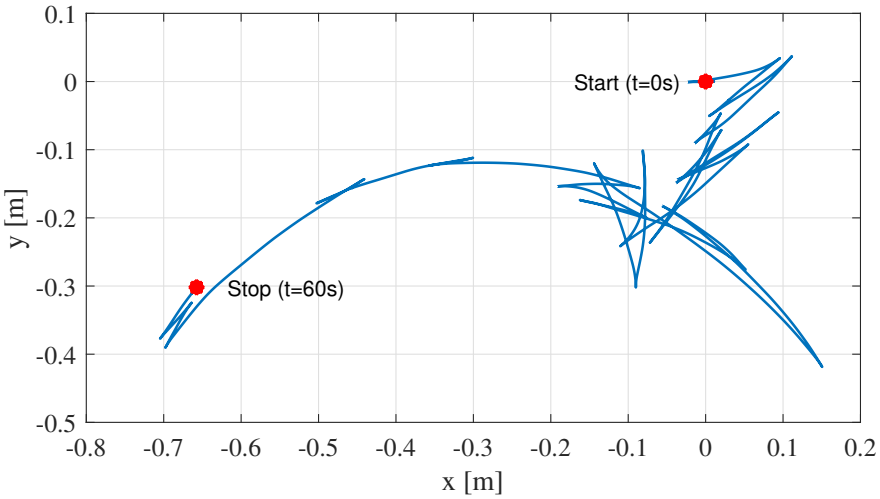


Figure 5.5: *Results of x- and y-position while balancing*

When adding the yaw controller, the robot was still moving quite a bit in the x-direction. However, the y-position remained within ±5 cm which is very satisfying. This is shown in Figure 5.9. The yaw angle was correcting the error and the large drift from before was reduced significantly. However, the controller had to be tuned additionally to make the robot follow a line.

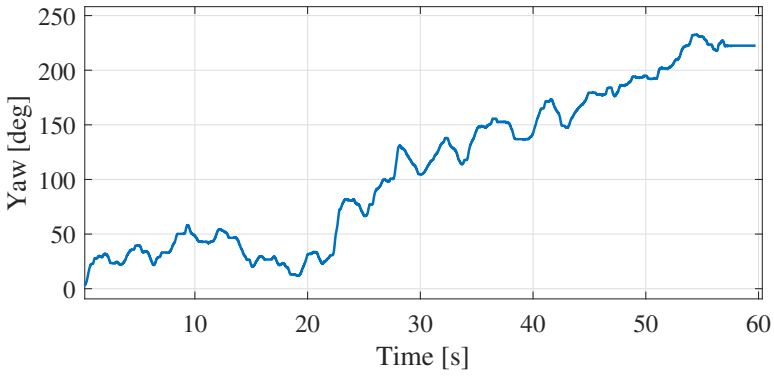


Figure 5.6: *Results of yaw while balancing*

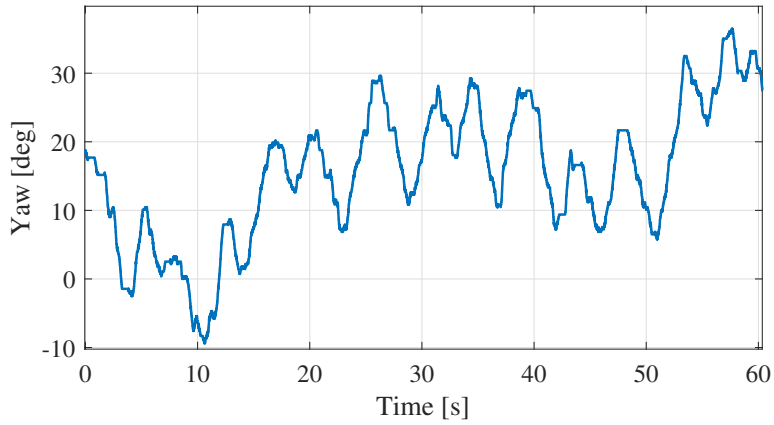


Figure 5.7: *Results of yaw while balancing with yaw controller*

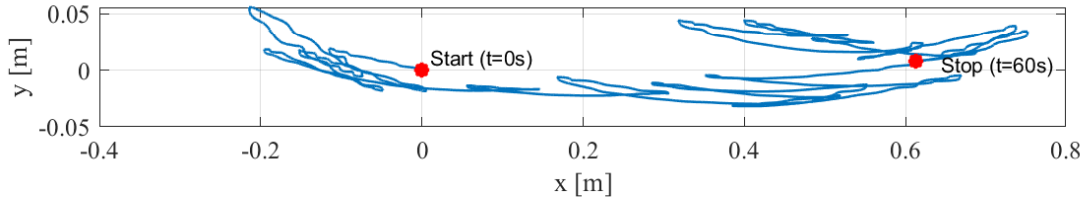


Figure 5.8: *Results of x-, and y-position while balancing with yaw controller*

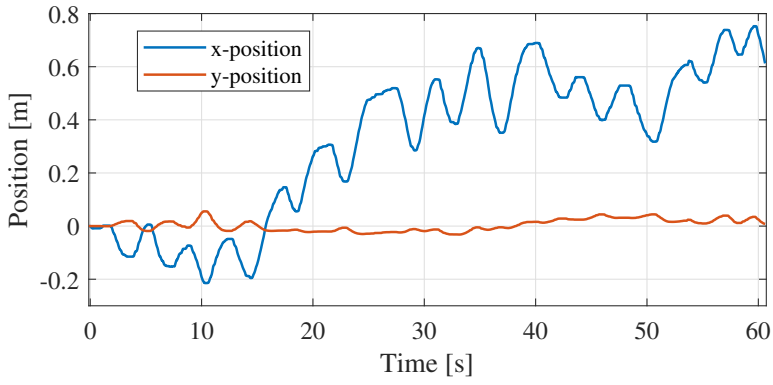


Figure 5.9: *Results of x-, and y-position while balancing with yaw controller*

5.4.1 Tuning

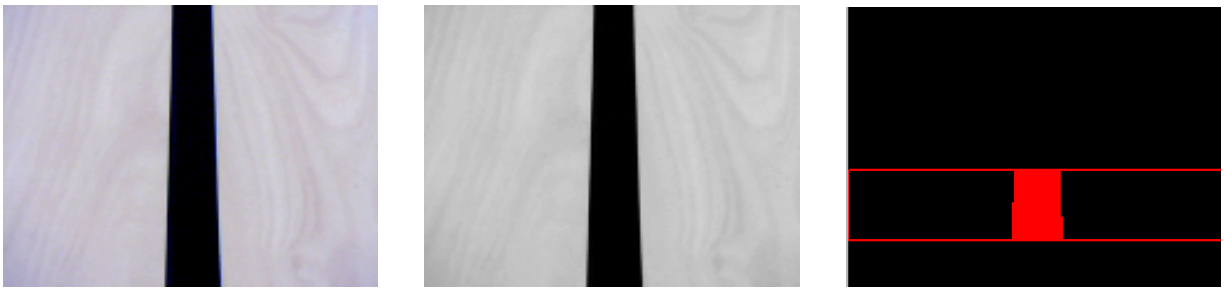
The final values of each controller is shown in Table 5.2

Controller	**Type**	$\mathbf{K_C}$	$\mathbf{T_I}$	$\mathbf{T_D}$
Balance outer controller	PID	400	0.005	0.00025
Balance inner controller	P	0.0015	-	-
Velocity Controller	PI	1	0.005	-
Turn rate controller	PD	0.002	-	0,05

Table 5.2: *Controller parameters*

5.5 Camera detection and line following robot

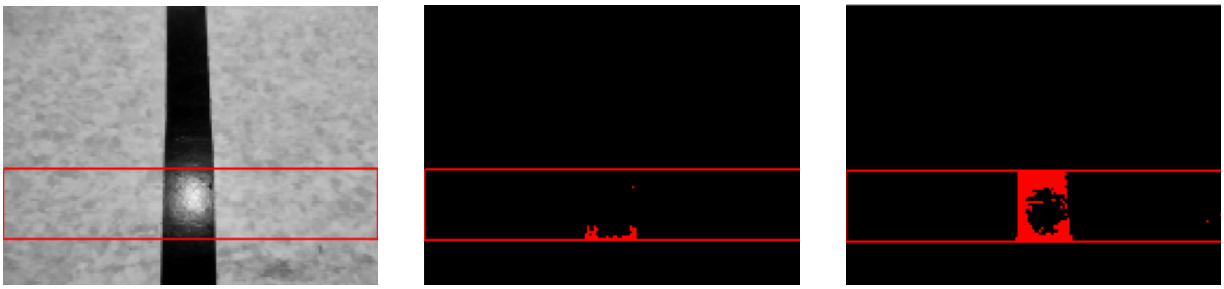
The various images used by the detection algorithm can be seen in Figure 5.10. Figure 5.10a shows the raw image coming directly from the camera on the robot. Figure 5.10b shows the image after the luminosity plane has been extracted which corresponds to the grayscale image. Finally, image 5.10c shows how the binary image detects the black line within the ROI. It can be seen that the detection algorithm works very well with a still image.



(a) *Raw image from camera* (b) *Image converted to grayscale* (c) *Binary image, detection within ROI*

Figure 5.10: *Images calculated in the detection loop*

The impact on detection by light reflections on the floor can be seen in Figure 5.12. Two different detection methods were tested, a manual threshold limit, and the clustering detection algorithm. During light reflections, it is clear that the clustering algorithm is the best choice. However, the manual threshold is still able to detect the line, as long as there is some detection in the image, the center of the object can be calculated.



(a) *Example of light reflection in raw image* (b) *Manual detection threshold* (c) *Clustering detection alghoritm*

Figure 5.11: *Impact on line detection by light reflections*

During some lighting conditions, shadows become a problem with detection. Especially if there is a bright light source behind the robot, this will cause a false detection by the shadow. In Figure 5.12a the impact of shadows can be seen from both the manual threshold and the clustering detection algorithm. Here it is the manual threshold that does the best job at detecting the line, the clustering algorithm is not able to separate the shadows from the line.

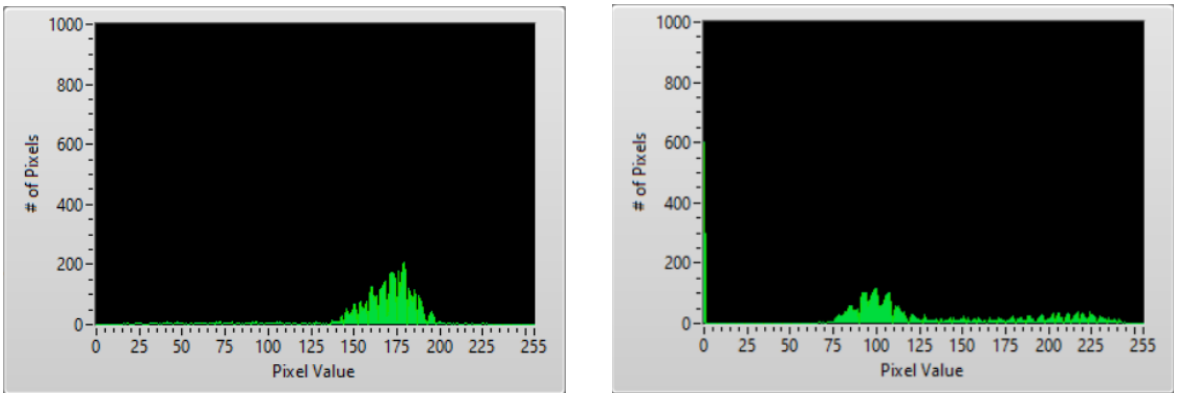
(a) *Example of shadows in raw image*

(b) *Manual detection threshold*

(c) *Clustering detection alghoritm*

Figure 5.12: *Impact on line detection by shadow*

Figure 5.13 shows the histograms of the graylevel output during both reflections and shadows in the image. From these histograms the manual threshold was set, the best performance on both problem cases was seen with a manual threshold from 0 to 35. This causes every pixel with a luminosity lower than 35, within the ROI, to be detected as an object, and everything else will be detected as background. The manual threshold was chosen as the best option overall and was thus used in the final source code.



(a) *Grayscale histogram during reflections*

(b) *Grayscale histogram during shadows*

Figure 5.13: *Grayscale histograms*

6 Discussion

One time-consuming issue that was experienced was a hardware issue with the motor controller that proved to be faulty. The controller had to be replaced 2 times as the first replacement controller also proved to be faulty. Also when trying to connect the gyroscope, and determine its coordinate system, the gyroscope did not work properly either. The gyroscope was thus also replaced.

The identification and compensation of friction was the task that would require the most time to be performed. After the first identification of friction, it was discovered that one of the wheels was about 1 cm closer to the base than the other. This was then corrected and the friction identification had to be performed anew due to the new wheel mounting settings. During testing on the bench, these parameters made the recorded velocity of the wheels coincide fairly good, however during testing on the ground the robot would diverge from a straight line slightly to the left side during forwarding motion. The friction compensation parameters were thus manually tweaked slightly until the forward motion had little to no divergence from the straight line. The testing was only performed at a distance of around 4 meters, for longer stretches the steady-state error would only increase making the deviation from the straight line even larger. This is however the nature of the linear regression compensation that has been implemented.

To tune the controllers, a thoughtful variation of the controller parameters was performed until a satisfactory result was achieved. Other methods for tuning the controllers could have been used, but they all require a complete mathematical model. For the general dynamics of the robot, the inverted pendulum dynamics could have been used for a mathematical model which could then have been tuned. However, there is some doubt for whether these parameters would provide a sufficiently good response on the real robot as the robot's wheels have proven to have very varying friction dynamics. To model these friction dynamics accurately would be a very time-consuming job and would greatly complicate the model. The cascade controller was therefore chosen when it proved to be sufficient for this project.

During testing of the camera detection, it was seen that the detection algorithm is very dependent on the lighting conditions. An option would have been that the robot had a light source to illuminate its path making detection easier under varying light conditions.

The detection loop in the main program needed to be run at a much lower frequency than the control loop as the detection algorithm is very computationally expensive. Initially, a maximum frequency of 5 Hz was the maximum the MyRIO could handle before being overloaded. At 5Hz the line following act was affected by overshooting the line when turning due to the position not updating frequently enough. Otherwise, when the loop was not able to finish its computation in time, the control loop would not receive the X-position in time, and thus the entire line following act would be delayed and the same problem occurred. After reducing the camera resolution the frequency could be increased to 10 Hz comfortably. At this frequency, the performance of the line following was noticeably improved.

The WiFi used by the wireless hotspot used to connect LabVIEW to the MyRIO was noticed to have a very large impact on the performance of the connection. At Fjære Skole the WiFi connection

is very poor which caused the LabVIEW VI to lag by many seconds. Also, the uploading of new source code would take a very long time. A solution was found in connecting the LabVIEW PC to a mobile data hotspot through a cell phone. Then sharing this WiFi to the MyRIO, this improved the transfer rate between the PC and MyRIO greatly and the video feed could almost be viewed in real-time.

7 Conclusion

All tasks of 4 of the exercises have been performed with good results without any major problems. The LabVIEW code works efficiently on the MyRIO and the LabVIEW front panel gives all requested outputs with little noise and delay. The friction compensation is not optimal, however as friction function is highly nonlinear and non-consistent.

The implementation of the gyro and accelerometer in a complementary filter has proved to give good results for detecting the pitch and yaw angles of the robot. The balance parameter alpha in the complementary filter was set to 0,98 which showed the best results for the accelerometer's low term accuracy and the gyroscope's long term accuracy.

The robot has been made to satisfactorily balance autonomously. The robot has very little drift while only set to balance. The robot also can bring itself back to balance from small manual disturbances. The linear velocity can be controlled directly from the Labview VI. The turn rate controller makes sure that the heading is straight despite the wheels having very varying friction dynamics, even after a good friction compensation.

When using the line following function of the robot it performs well even at tight turns in the path. A manual detection threshold has been set that provides the best results in varying light conditions. Using a larger velocity bias gives a smoother performance at straight sections in the path at a trade-off for some overshoot from the path during tight corners.

A video has been made of the final robot which can be viewed on YouTube.

Link: https://youtu.be/AZ8HnyVm6Bw

Bibliography

[1] Pulse width modulation (pwm). [Online]. Available: https://en.wikipedia.org/wiki/Pulse-width_modulation

[2] Inverted pendulum. [Online]. Available: https://en.wikipedia.org/wiki/Inverted_pendulum

A Labview program

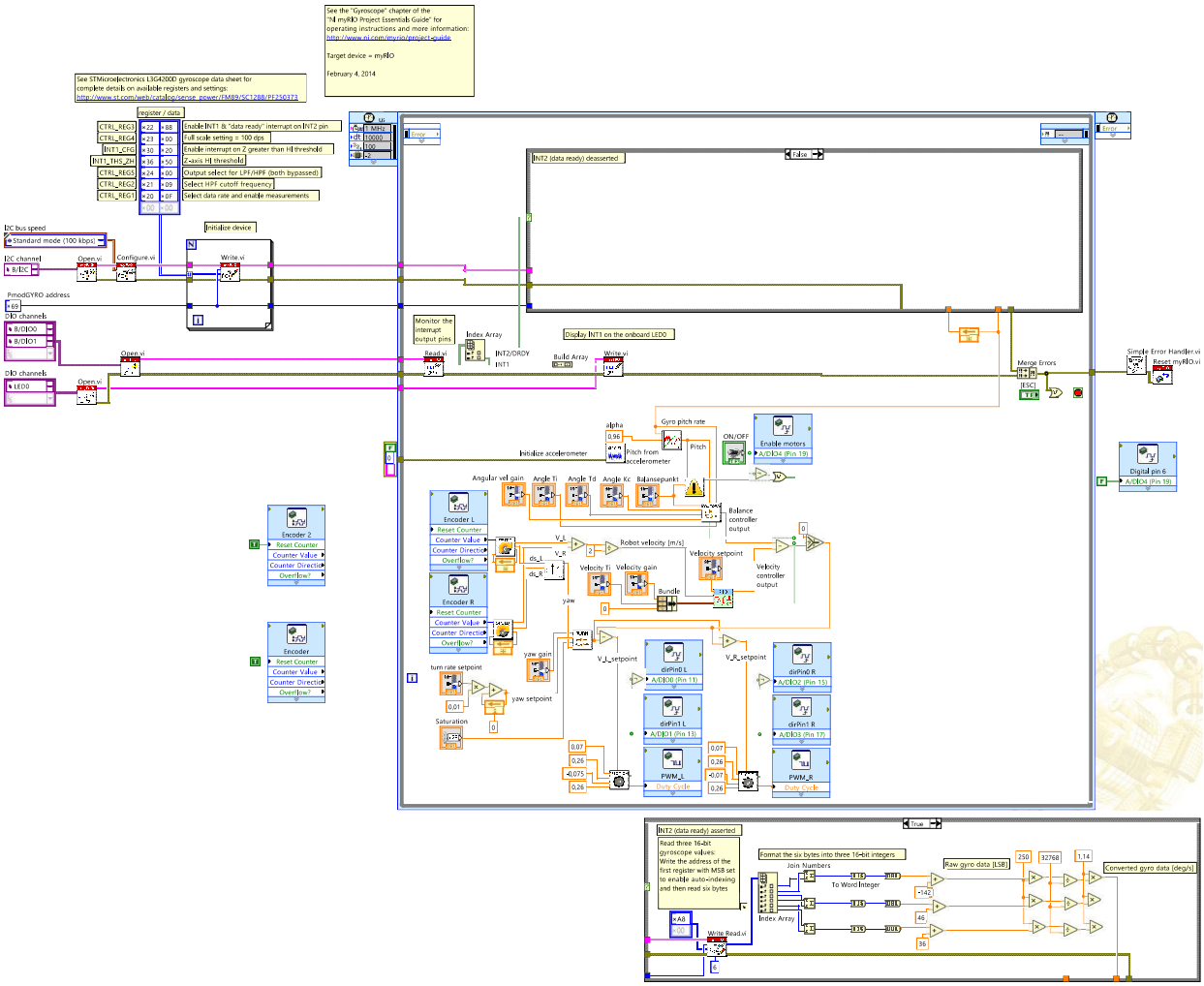


Figure A.1: *Labview main file*

A.1 SubVIs

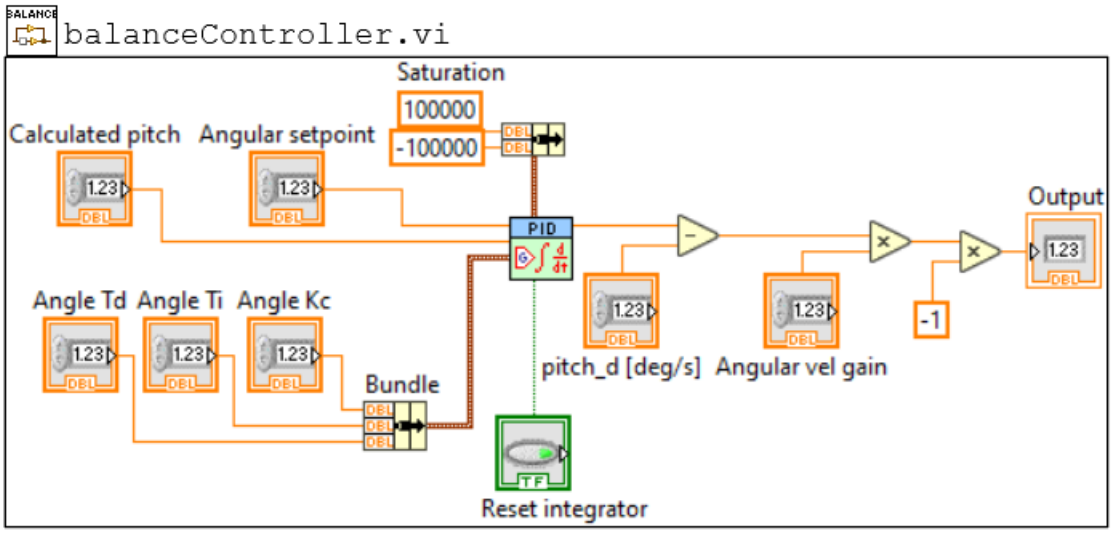


Figure A.2: *Balance controller*

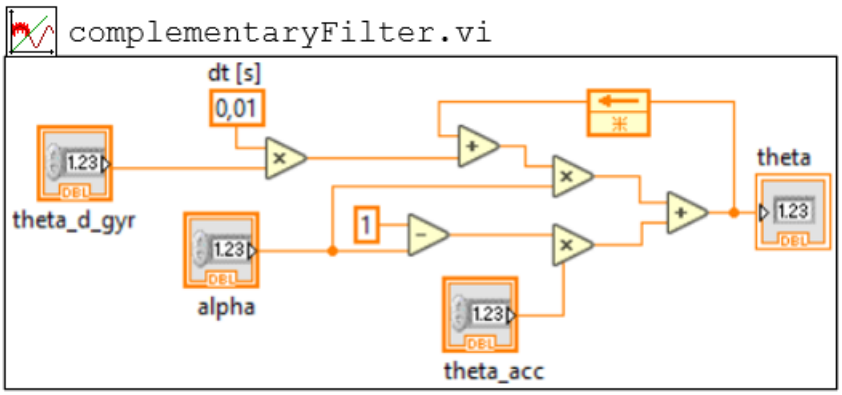


Figure A.3: *Complementary filter*

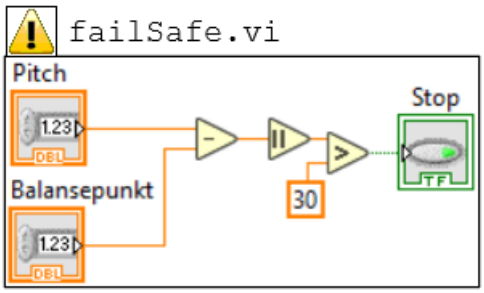


Figure A.4: *Fail safe*

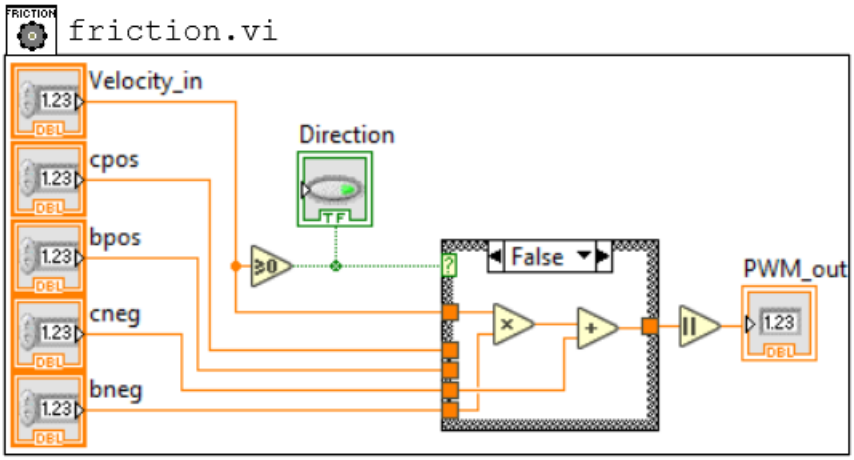


Figure A.5: *Friction compensation*

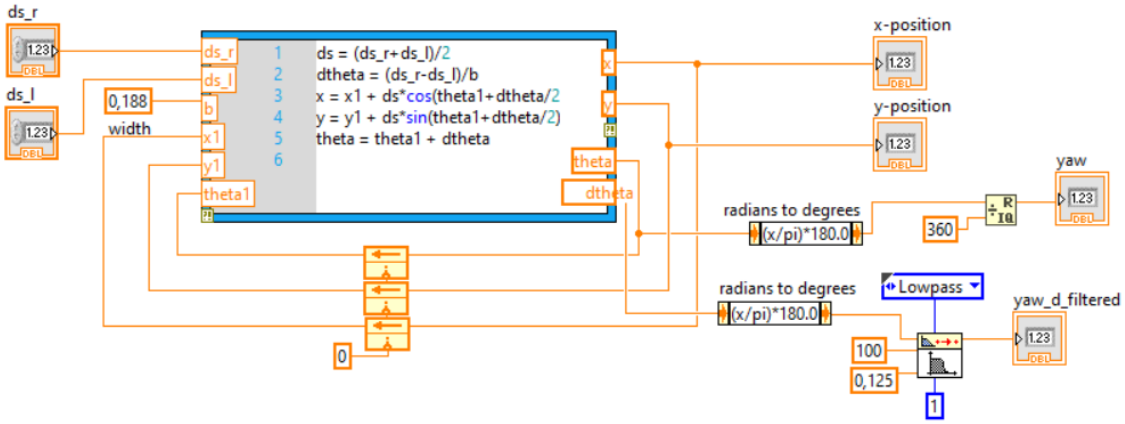


Figure A.6: *Odometry*

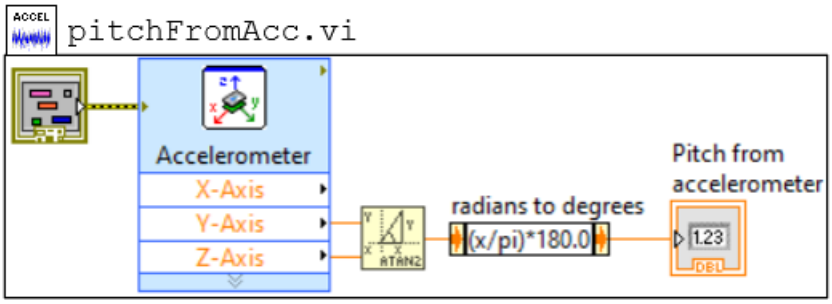


Figure A.7: *Pitch calculations from the accelerometer*

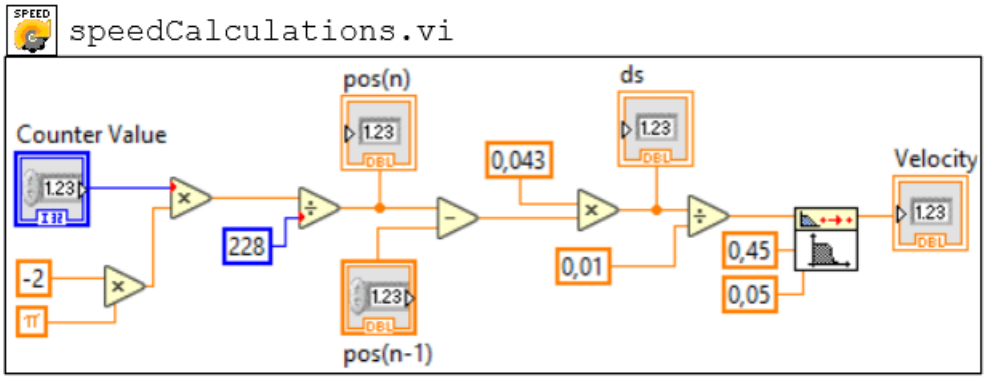


Figure A.8: *Speed calculations from encoder*

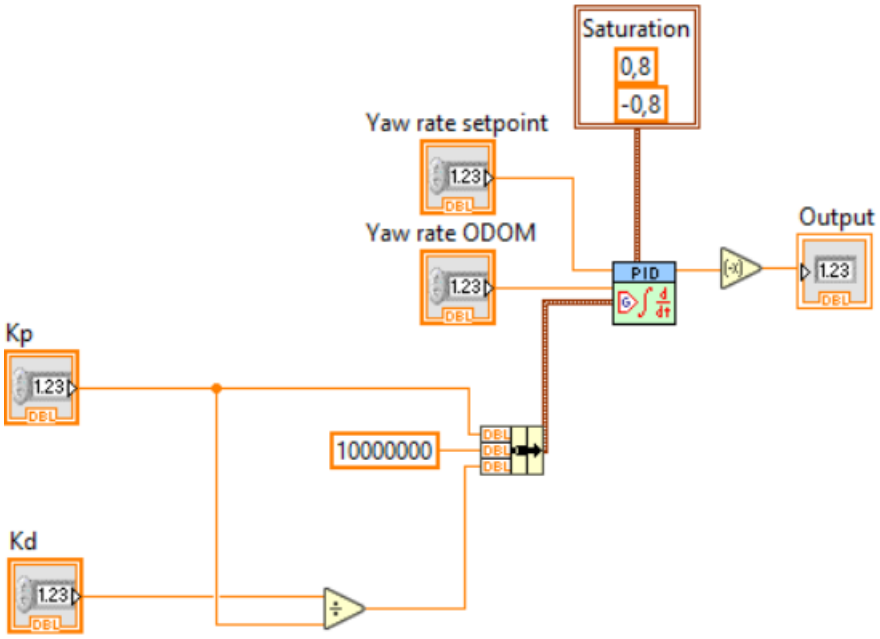


Figure A.9: *Turn rate controller*