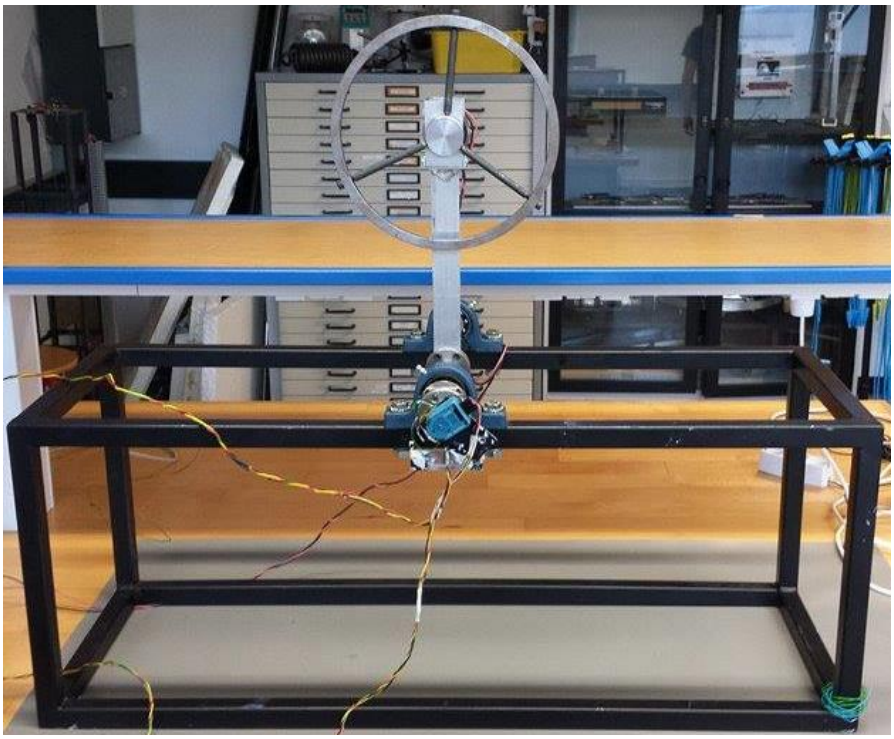
Study of Inverted Pendulum Control using a Reaction Wheel

Basic Control Engineering



Title:

Study of Inverted Pendulum Control using a Reaction Wheel

Theme:

Basic Control Engineering

Project Period:

Spring Semester 2015

Project Group:

ED4-2-F15

Participants:

Aleksandrs Levi

Arturs Gumenuks

Alexandru Popa

Mike Castro Lundin

Supervisors:

Zhenyu Yang

Christian Mai

Pages: 95

Date of Completion:

May 28, 2015

Department of Electronics & Computer Engineering

Niels Bohrs Vej 8

DK-6700 Esbjerg

http://esbjerg.aau.dk

Abstract

This project has the purpose of applying control theory principles, as well as mechanical and electronic modelling practices in order to achieve stability in an unstable system. The system used will be an inverted pendulum, and the stabilization tool is a reaction wheel. The project compares two different modelling results as well as a software-based solution without modelling, and evaluates their viability.

The software solution uses an Arduino microcontroller and a short time loop program, utilizing multithreading techniques. The two modelling solutions are using a transfer function approach, and using a state-space to represent a Single Input Multiple Output (SIMO) system. Both these models utilize a PID controller to implement feedback control.

Additionally, we have created an ADC-DAC communication system to allow additional sensor inputs into our PCI.

Preface

This study project was developed by group ED4-2-F15 as an application of the Fundamental Control Theory and Modelling and Embedded Software Design courses. The project focuses on stabilizing and swinging up an inverted pendulum with a reaction wheel, an application that requires mechanical, electronic and software knowledge in order to understand the system.

The group is aiming to create a functional Simulink model, as well as the working prototype. The Simulink model contains and combines the mechanical and electrical components of the pendulum and the motor in order to reach stability, by analyzing the behavior of the system in each step.

The report is organized in a manner that readers would understand and get the main ideas about the problem analysis, modelling, hardware used, software developed in the project and mechanical part.

The sources used for inherited methods, photos, citations and references were organized according to the Vancouver citation style. All of those sources are mentioned in the section “References” and the software and hardware parts developed and the modelling parts are placed in the “Appendices”.

Aleksandrs Levi

<alevi13@student.aau.dk>

Alexandru Popa

<apopa13@student.aau.dk>

Arturs Gumenuks

<agumen13@student.aau.dk>

Mike Castro Lundin

<mcastr13@student.aau.dk>

Contents

List of Figures

Problem Analysis

Introduction

The task of stabilizing an unstable system by the use of basic control principles is a quite common issue in real life. It is also used in more advanced applications, such as rotation of a spacecraft.

Our system utilizes a rotational wheel to create a stabilizing feedback; however, there are other ways to generate such a force, the most common being moving the pendulum using a cart.

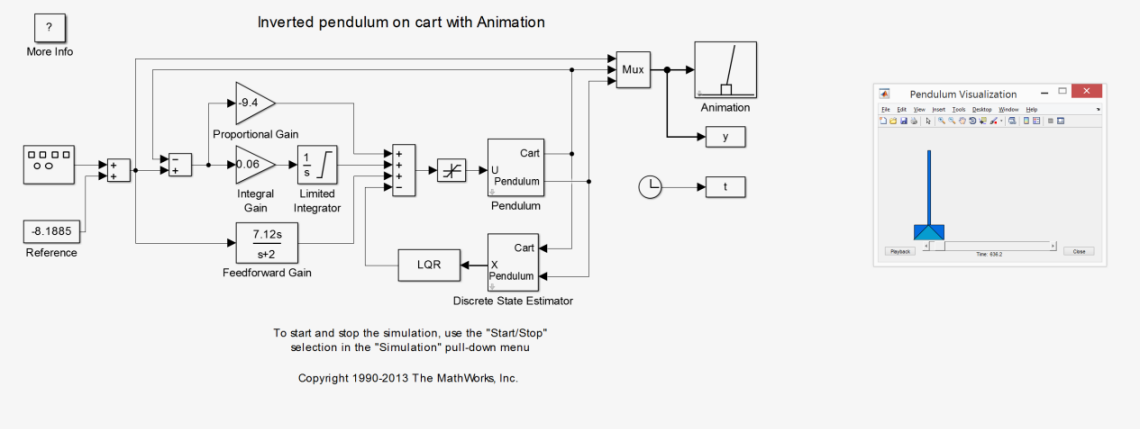


Figure 1. Simulink cart demo.

The inverted pendulum allows the control designer to decide a response time and aggressiveness of the response, as well as the possibility of adding more functions, such as the swing up.

The inverted pendulum is a very unstable system, as can be seen by something as simple as trying to balance a stick on your hand. In the same manner, adding a mass to the end of the pendulum would change the behavior of the system.

The main principle applied to achieve stability is the conservation of angular momentum, which allows us exert a force on the pendulum by motion in the wheel.

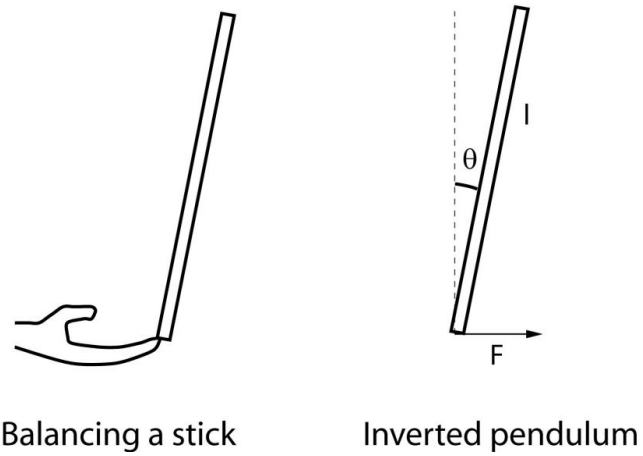


Figure 2. Inverted pendulum analogy [1].

Importance of the Inverted Pendulum [2]

The inverted pendulum is a famous application of control theory engineering and robotics. An application of the pendulum, although it is unusual, is the capacity of human body to maintain the still position. Depending on environments and surfaces, the human body has to face several forces and disturbances to keep in position.

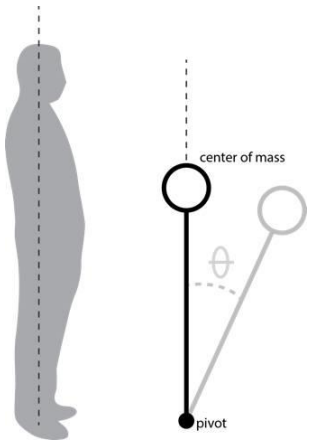


Figure 3. Mechanics of standing [3].

The Segway is another innovative application of the inverted pendulum. The machine is a two-wheeled device that works on a battery and it is controlled by balancing handle bar, which generates velocity to the motors to the way it is inclined. Since its appearance on market the

device proved its expected success among people that want to make small journeys with no effort and consuming small amounts of energy. The top speed that a Segway can reach is 20 km/h.

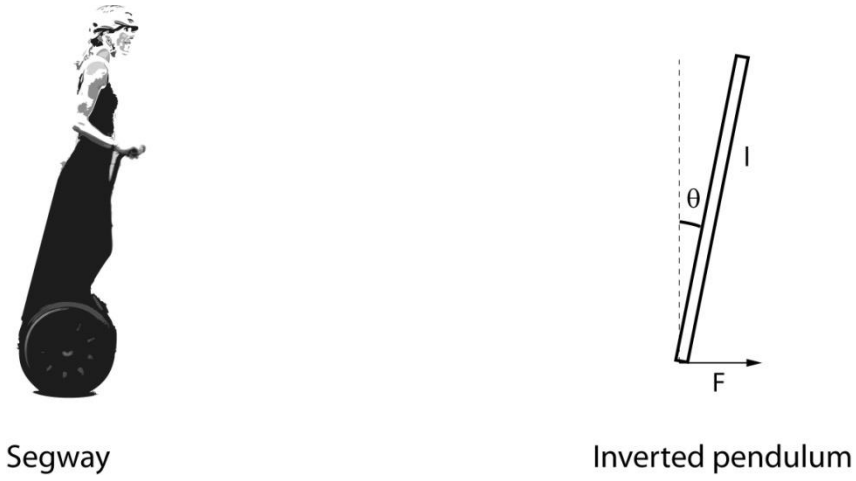


Figure 4. Inverted pendulum application in Segway [1].

Block diagram

This problem requires a control theory solution; the following block diagram shows a general solution to a system that requires feedback to be stabilized.

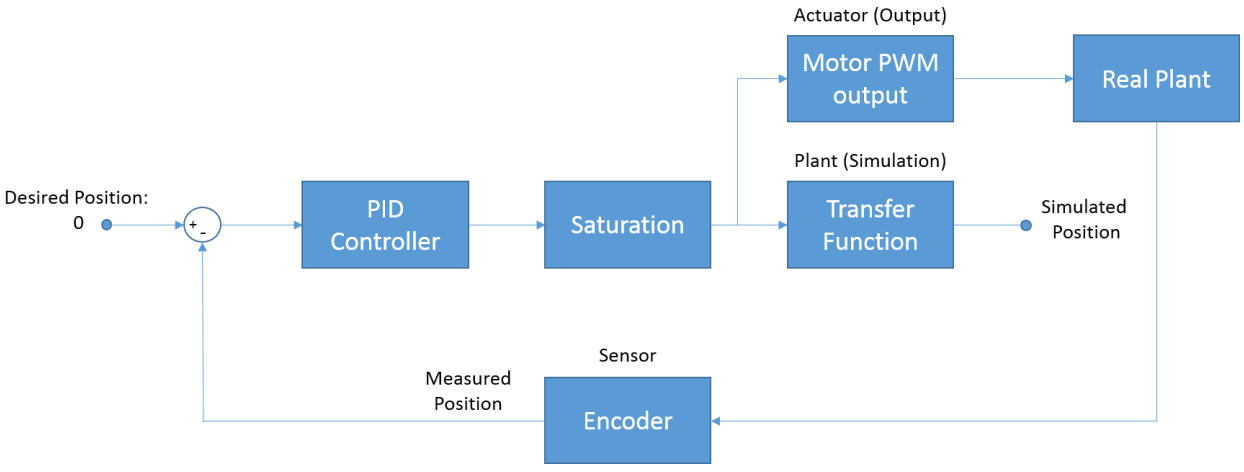


Figure 5. Simulink block diagram.

The specific blocks and parts of this model will be described later on the report.

Prototype

This part of the report shows the starting ideas to solve the problem and an overview on the prototype, including the main parts if it, sensors, block diagrams and briefly the principles used for controlling the device.

Components [4]

By having a general look on the prototype the pendulum is fixed on a rectangular box with no surfaces. One end of the pendulum is welded to a cylindrical crossbar (pendulum axis), which allows it to balance from left to right. On the other end, the rotational wheel was placed, which has a radius of approximately 11.75 centimeters and the inertia of 6.572e-3 kg·m². The rotations of the wheel are controlled by a DC motor (24V of model 440-329), which produces maximum 140 rot/min. The pendulum weights 1.25 kg and is 26.5 cm long.

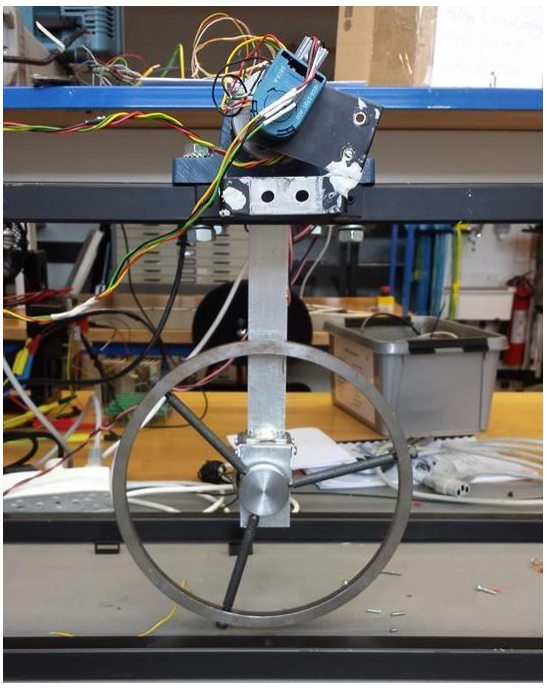


Figure 6. Photo of prototype.

Principles

The physical aspects of the components of the prototype are very important in the calculations for developing the mathematical model. The masses have to be balanced and the materials of which the bars and the pendulum are made of have to be rigid enough so that it will prevent breakings because of balancing and interactions with other forces. For example, the pendulum and its axis are made of aluminum, which has density of 2700kg/m³ and strength between 70 and 700 MPa. [5]

The way the prototype works is by accelerating the wheel and changing the direction of spinning according to the current position. This generates impulses for the pendulum, making it balance. By increasing the acceleration, the pendulum goes higher and higher up until it reaches a full 180° degrees rotation. This is what we refer to as a swing up motion.

One of the main challenges is making the wheel stop and stabilize itself by slow spins. This is a matter of linearizing the mathematical model and using a PID controller to develop a feedback system. Simulink sends the system's outputs in real time so that the model will act accordingly.

Sensors

In order that the program will get the values of the position, angular velocity and acceleration of the pendulum, it is necessary to have sensors attached to the prototype in order to measure these values. Hence, the system can use the values of two encoders and one accelerometer.

In the picture below, one of the encoders is placed on the pendulums axis from where the position of the pendulum is going to be calculated. The encoder's rotor is placed inside of the axis, which moves simultaneously with the pendulum and by outputting the number of cycles the angle can be depicted by the sensor.

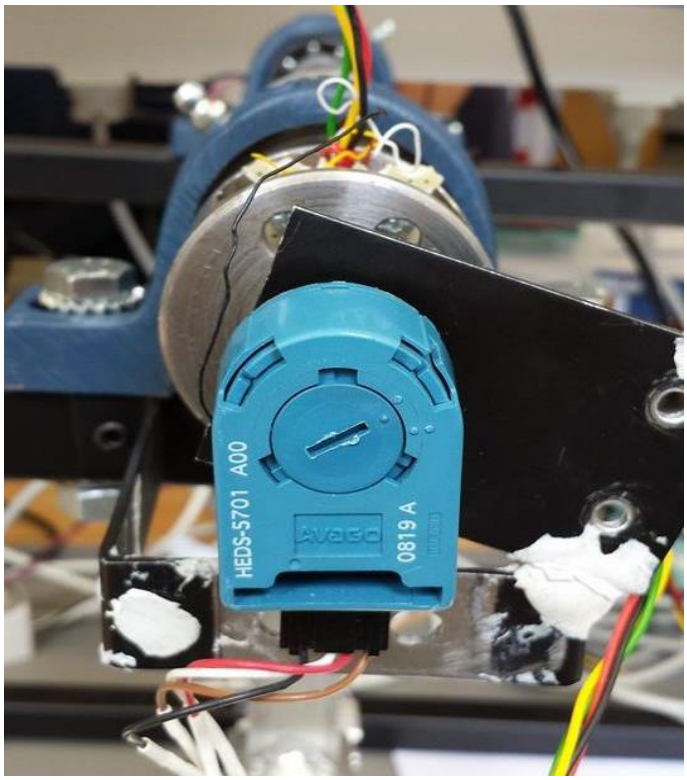


Figure 7. The encoder attached to the prototype.

The accelerometer is placed right behind the first encoder and gives the position of the pendulum. The difference between it and the encoder is that the accelerometer receives pulses as inputs, in order to give an output inside the program. The sensors gives the values of the position of the pendulum in a Cartesian system (X, Y system).

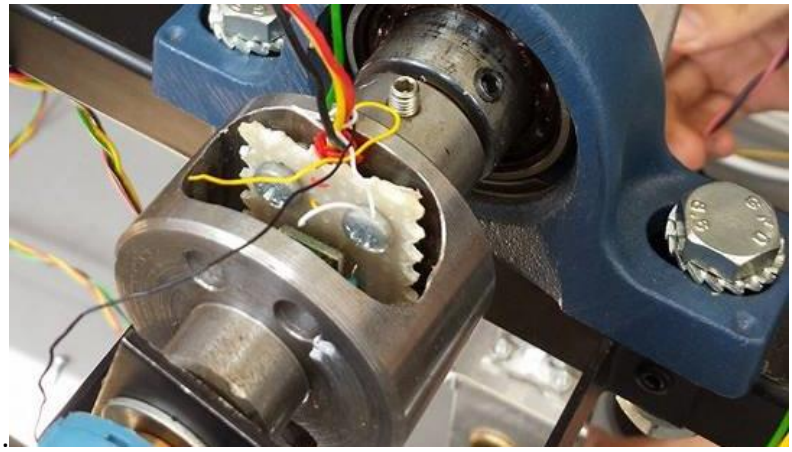


Figure 8. The accelerometer attached to the prototype.

The second encoder is placed on the motors rotor and has the purpose of reading in an accurate way the angular velocity of the motor. As seen in the picture below, the rotor has an extension that would connect it to the wheel and on the disc of the encoder, which acts in a similar principle to a hall sensor.

Block diagram

The hardware setup discussed will be used to develop a Simulink system with the following block diagram:

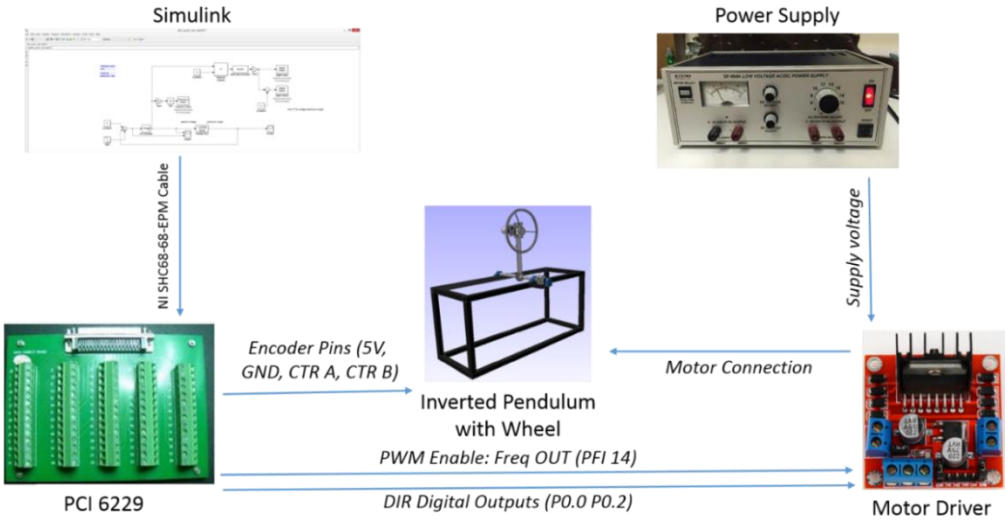


Figure 9. Hardware diagram.

As we can see, Simulink is in charge of reading the sensor values and generating a response according to the model. This is all done via the PCI-6229 board. The output is then further processed into the driver, from two 5 volt digital outputs and a 5 volt PWM signal into the respective voltage into the motor (between ±24 volts) and determining which motor cable will be the V_{CC} and which will be the GND.

Optocoupler [6]

Our initial design motor driver included the use of optocouplers, also called opto-isolators as part of the logical control of the engine, in order to determine the direction the motor would go.

Optocouplers are used when a circuit is split into a low voltage section and a high voltage section dependant on the other section. Since connecting them together could pose risks to sensitive electronic parts, such as LEDs, we utilize a combination of a low voltage LED and a phototransistor to switch on or off depending on the light intensity detected. The general electric circuit they have is the following:

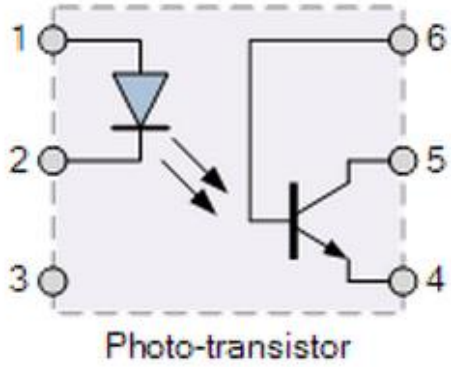


Figure 10. Electric circuit of an optocoupler [7].

There are also more complex optocouplers that support an AC source through pins 1 and 2, by having two diodes in parallel, in opposite directions relative to each other. This ensures that there is always an LED on if there is a current.

In our application we used the Toshiba TLP250 (not a photo-transistor), which uses a gallium-aluminum-arsenide diode in the low voltage side, which we ran from the PCI board, which uses a voltage of 5V. Some important characteristics are that it has a maximum input current of 5 mA in the low voltage side, and an ±1.5 A maximum current in the high voltage side, which we used at 24V. Its voltage supply range is between 10 and 35 volts. The TLP uses two internal transistors to ensure that the output is always either V_{cc} or GND. This means that if one transistor is on, the other will be off. Following is the schematic of the TLP250.

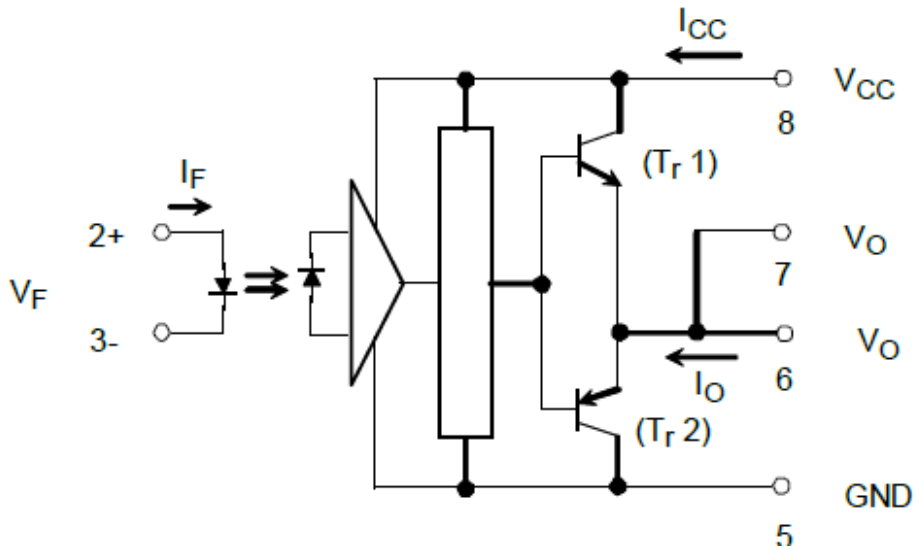


Figure 11. The schematic of the TLP250 [8].

It must also be said that our circuit included a 0.1 µF capacitor between V_{cc} and GND. This capacitor had the function of stabilizing the amplifier part of the circuit. If this capacitor was not provided it could impair the switching of state in the optocoupler.

H-Bridge Theory [9]

For our project, we needed to build a circuit for a DC-motor, used in a reaction wheel. The H-bridge is a typically used electronic circuit that allows voltage to be applied across a load in either direction, allowing a motor to rotate both forwards and backwards.

An H-bridge consists of four switching elements (transistors), with a load in the center, in an H-like configuration (hence the name).

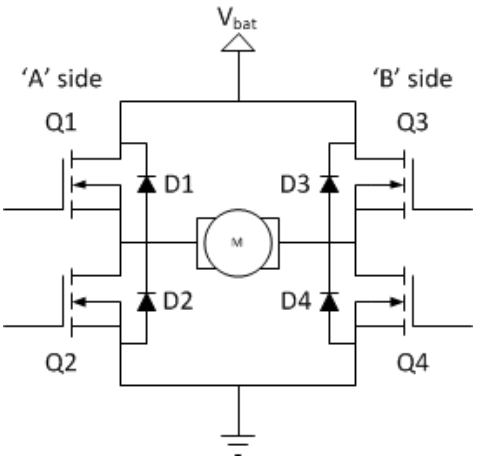


Figure 12. H-bridge configuration [9].

In the Figure 12: Q1 to Q4 are the switching elements (transistors), while D1 to D4 are flyback (catch) diodes, which are used due to their property of flyback (voltage spikes) elimination. The top-end of the bridge is connected to a power supply, while the low-end is connected to the ground.

Normally, all four switches can be turned on/off independently of each other. The load typically is a DC brushed motor or a bipolar stepper motor.

Operation principle

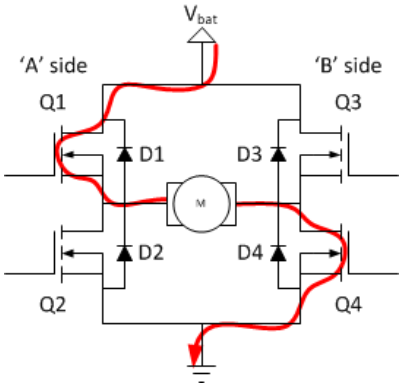


Figure 13. H-bridge operation mode 1 [9].

The basic operating mode of an H-bridge works in the following way: when Q1 and Q4 are on, the left motor lead is connected to the power supply, while the right lead is grounded. Current flows through the motor, causing motor shaft to start spinning in a desired direction (Figure 13).

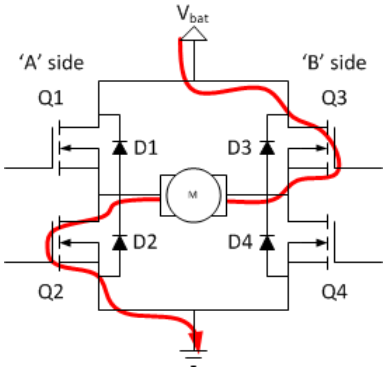


Figure 14. H-bridge operation mode 2 [9].

Vice versa, if Q2 and Q3 are turned on, current flows in another direction, and the shaft starts spinning in the opposite direction (Figure 14).

Turning both Q1 and Q2 (or Q3 and Q4) switches on at the same time should be never done, since it creates a low-resistance path between power and ground, short-circuiting the power supply.

Building a motor driver

As a process of experimenting and learning, the group has started designing a motor board using components available in the laboratory. The board was designed to control the direction, speed and acceleration of the DC motor, controlled digitally by the PCI.

The H-bridge is formed by four IRFZ44V type transistors, which can carry high currents through them. In order to open the gate of the transistors, there are 30 V needed to perform the operation. In order to achieve a defined direction of spinning, we need some digital inputs. In addition, a PWM signal was used to reduce the effective voltage through the motor. In order to separate the 5 volts from the 24 volts to the motor and the 30 volts for the gates of the transistors, 4 optocouplers were used.

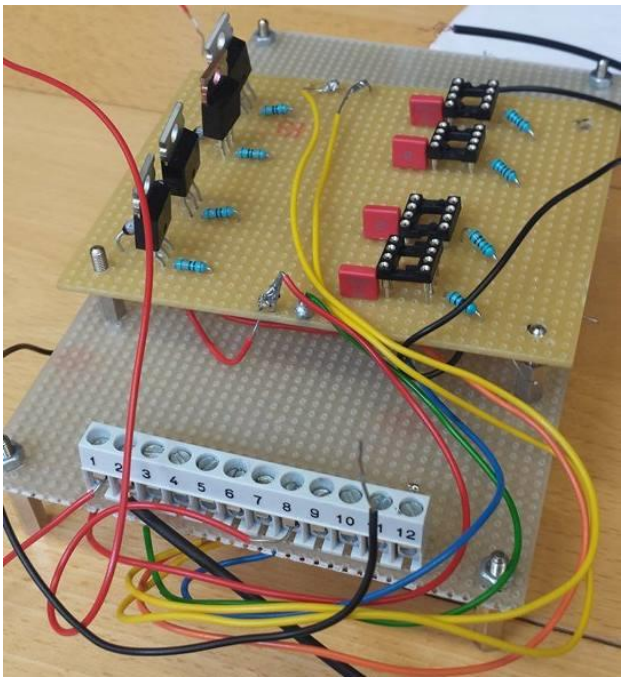


Figure 15. Self-made motor driver.

Finally, this driver was not part of the final prototype, since the testing resulted in a leak of current in a part of the circuit, likely caused by a burnt transistor or a short circuit.

The L298N Motor Board

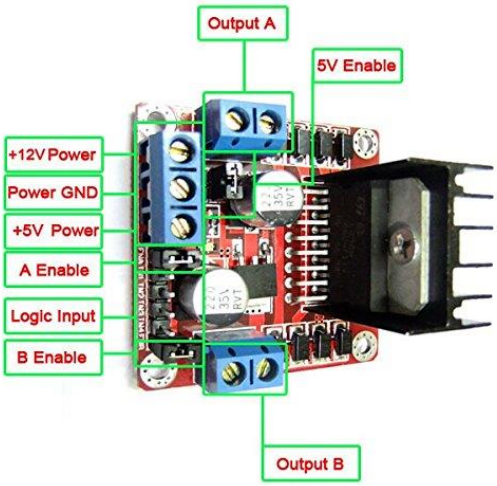


Figure 16. L298N Motor driver [10].

The L298N dual-motor board was the final driver used in developing the prototype. The reason why this board was used is that the SX8847 Motor Driver was burnt after a few usages.

L298 dual H-bridge seemed the most reliable also because of the fact that it can carry electrical charges for two different motors. It is an integrated monolithic circuit which can operate with up to 46V, carrying a maximum 4A current. The Multiwatt 15 and the PowerSO20 are handling the currents and directing them to the motors. They are also taking the digital commands in order to apply the enable a direction and the PWM signals.

By analyzing the block diagram below, it is noticeable that the inputs for the directions of the motors are going through a NOT gate in order to make sure that only one transistor is ON, and the current does not go directly to the ground. In order for the directions to be activated, the Enable pins have to be high, and additionally they can be given a PWM signal so that the voltage is reduced. The directions will be changed by making the pair of Ins accordingly to each motor High and Low (e.g. In1=High and In2=Low - the motor spins right; In1=Low and In2=High - the motor spins left), still the board will offer no response if the two inputs have the same values (High-High and Low-Low).

The power supply is the same for both motors, so they will have the same input voltage, which then may be regulated via the enable pins.

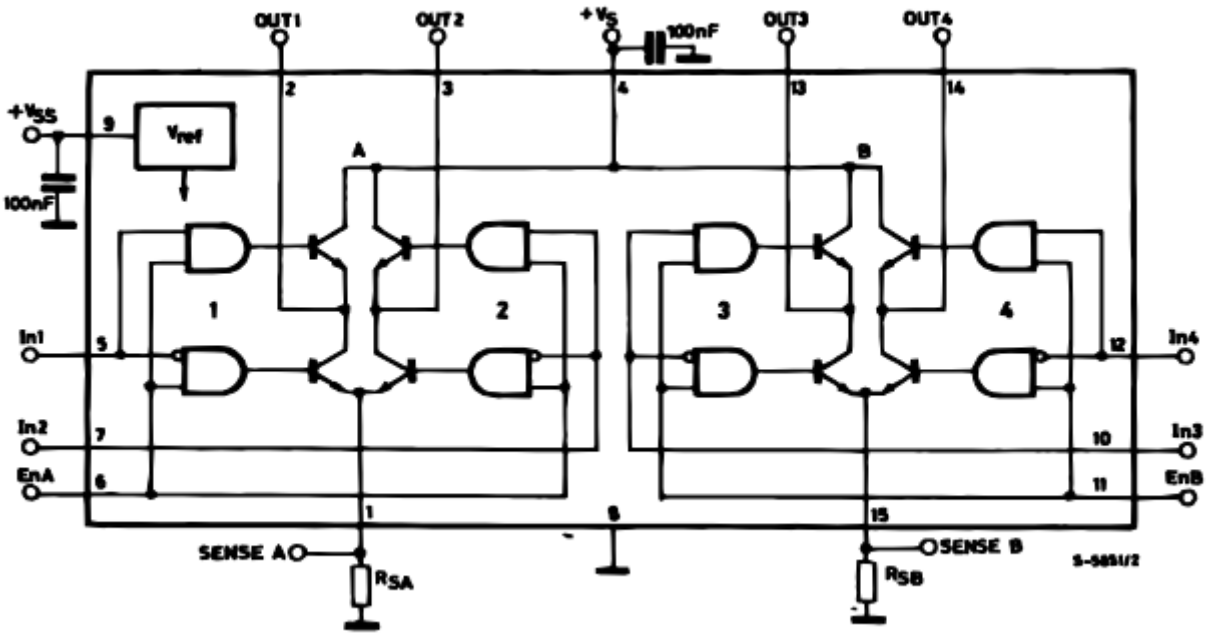


Figure 17. L289N Motor driver logic diagram [11].

The board acts properly without any problems. All the 'safety' measures for the components of the board are inbuilt, such as the capacitor in the power supply and the heat sink placed on the dual H-bridge (which usually warms up when high voltages are supplied).

PCI-6229 [12]

The Conventional PCI (Peripheral Component Interconnect) is a computer bus created by Intel, which is meant to connect hardware devices to the PC.

The PCI-6229 board from National Instruments has a multiple-pin layout, ready to deal with and provide with both analogue and digital signals. Multiple tasks can be performed by using the board, such as constructing integrated circuits, reading sensor values and activating them, and controlling several devices. The pin characteristics and modes are programmable, so that the user is able to select the connections to the circuit or devices via the ports of the board.

Figure 18. NI PCI-6229 board [13].

As shown in the pin layout in the following figure, the board consists of Digital I/O pins (8 pins from 0.0 to 0.7), Analog I/O pins (16 Analog inputs and 2 Analog outputs), 5V power supplies, PFIs (Programmable Function Interface pins), 10 Digital Grounds and Analog grounds (2 for output and 8 for input).

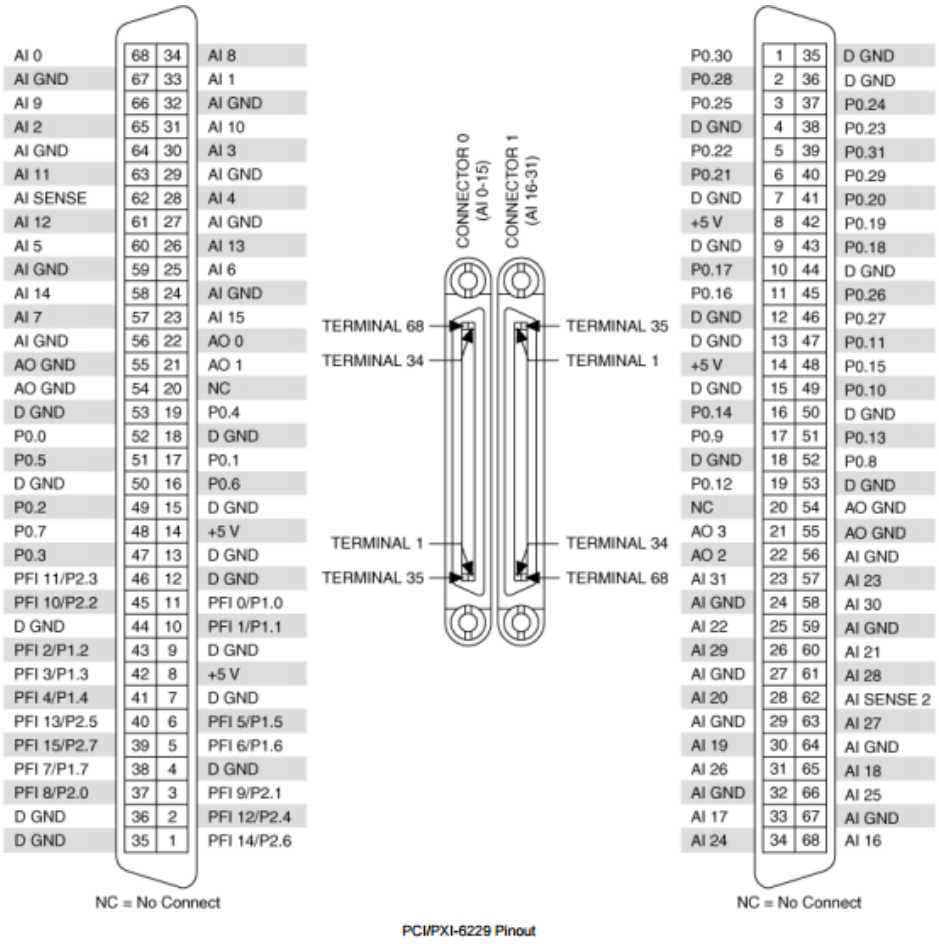


Figure 19. NI PCI-6229 board pinout [14].

Each PFI pin has a specific attribute in order to activate other different functionalities from the board. They are 15 in total and connecting counter signals in the NI-DAQmx (the program that runs the functions of the board after it is installed in the PC). The counters and their corresponding PFI pins are shown in the table below.

Counter/Timer Signal	Default Connector 0 Pin Number (Name)
CTR 0 SRC	37 (PFI 8)
CTR 0 GATE	3 (PFI 9)
CTR 0 AUX	45 (PFI 10)
CTR 0 OUT	2 (PFI 12)
CTR 0 A	37 (PFI 8)
CTR 0 Z	3 (PFI 9)
CTR 0 B	45 (PFI 10)
CTR 1 SRC	42 (PFI 3)
CTR 1 GATE	41 (PFI 4)
CTR 1 AUX	46 (PFI 11)
CTR 1 OUT	40 (PFI 13)
CTR 1 A	42 (PFI 3)
CTR 1 Z	41 (PFI 4)
CTR 1 B	46 (PFI 11)
FREQ OUT	1 (PFI 14)

Figure 20. Counters and their PFIs [15].

Panel Mount Optical Encoders (HEDS5701 series) [15]

The HEDS5701 is an optical encoder designed by HP Company and is used in the project with the purpose of sensing the position of the pendulum.

As a general idea of an encoder, it usually consists of three maximum channels: Channel A ,B and Z; as for encoding types of measurement, there are X1,X2 and X4, which are mostly based on a two channel encoding (Channels A and B).

The X1 type consists of the first channel moving the second one into a quadrature clocked cycle and in this case the clock increases; by switching the channels the clock continuously decreases.

The X2 type is depending on the channel that dictates the encoding (like in X1). Only the output is a double cycle of increments and decrements, which depend on the channel A's increments and decrements.

The X4 encoding type, which is used in the encoding for the type of sensor used in the project, no longer has any channel to lead the changing of the encoding. Each edge of the channels A and B are causing an increment followed by a decrement, and the duty cycle causes four increments and decrements in the Counter Value (in the previous cases only two increments and decrements were caused).

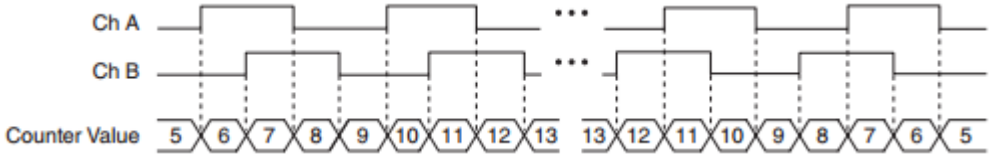


Figure 21. X4 encoding type [15].

The 3rd channel, Z is an alternative one to the A and B. As long as it is high, it can influence the size of the Counter value, while the other two channels are working as in the X4 encoding type.

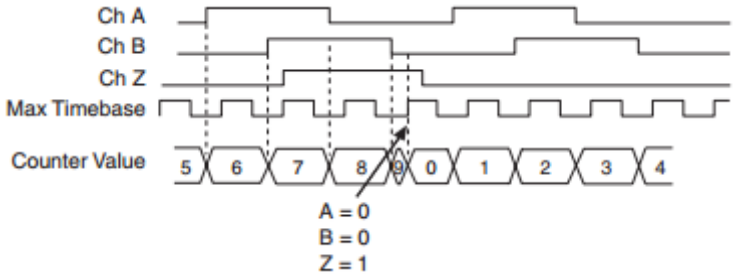


Figure 22. Channel z with X4 encoding type [15].

The sensor is settled into the capsule in the following picture. By taking a look inside of the capsule we can see that the rotor ends up with a disc that spins at the same time with it. The movements of the disk are sensed by the special detectors. The encoder has also a Collimated LED, which works together with the detector circuit. The readings of the sensors are dragged out through the two terminals, which are read by the controller.

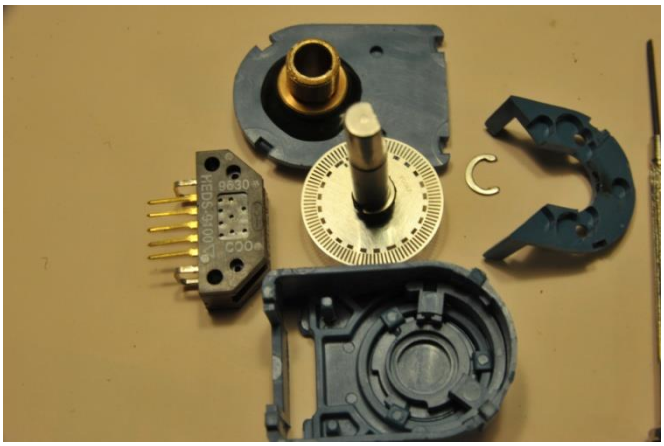


Figure 23. The HEDS5701 series encoder disassembled [16].

The capsule with the LED and detectors ends up with 5 terminals, which consist of Digital Ground (Pin1), the second pin, which has no connection, Channel A(Pin 3), V_{CC} power supply(Pin 4), where 5 V are input to the sensor in order to be activated, and ending up with the B Channel in the last pin.

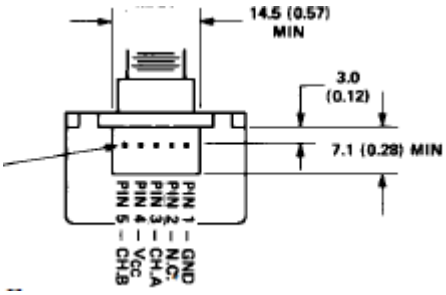


Figure 24. The HEDS5701 series encoder pinout [17].

Arduino Uno [18][19]

Arduino is a single-board microcontroller, regarded as one of the most user-friendly boards on the market nowadays. Arduino can easily collect information through its pins and generate an output on a PC via a communication protocol, or via another pin. Arduino is “an open-source electronics prototyping platform based on flexible, easy-to-use hardware and software”. Having a look at the architecture of the board, there are 14 digital I/O pins, 6 analog ports, and a power connector, where batteries can be connected, so it is not dependent on the USB cable.

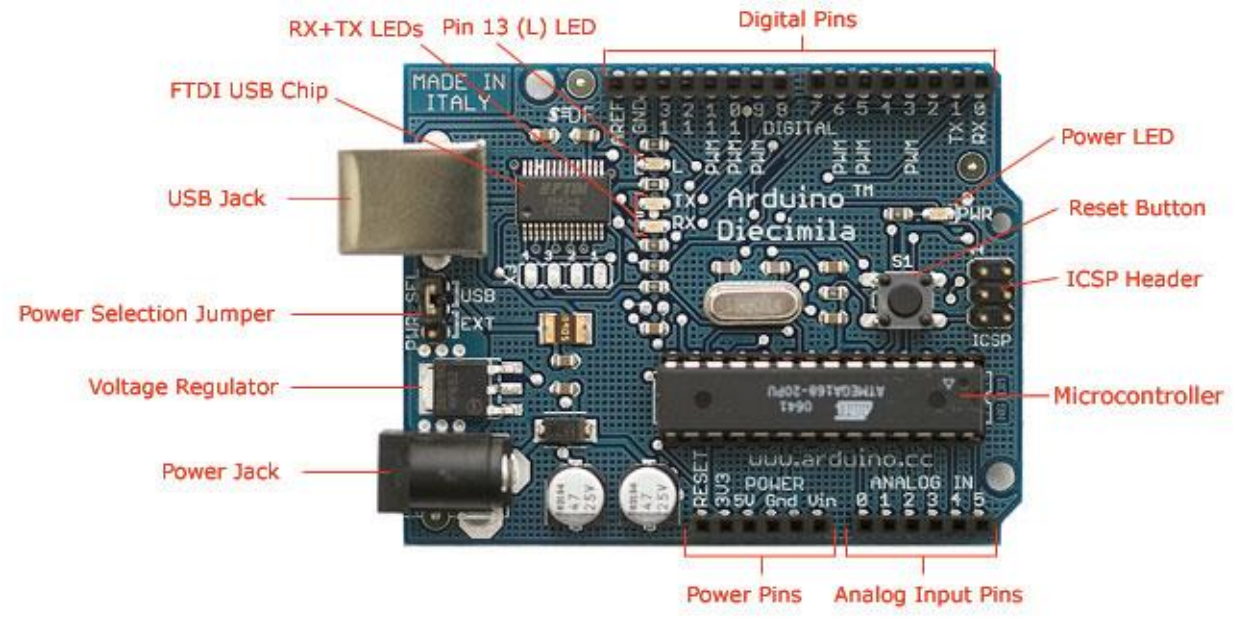


Figure 25. Arduino board elements [20].

The programming language that is used to perform operations on the board is similar to several programming languages in syntax. All combined together resulted in the original programming language working on the Arduino. The languages that influenced it are C and Java. An example for the imprint of the C language would be when importing libraries, called in the same mode:

#include <LiquidCrystal.h> (this is the library used when an LCD is attached to the board)

In addition, the second language that has an influence, Java, can be depicted in most of the actions that are made in the program, but still it is not the same coding. There are only similarities between Java and the Arduino language when we are printing/requesting the output of the program:

System.out.println(); for Java → *Serial.println();* for Arduino.

The software needed for the Arduino can be downloaded from their official website along with other extensions and example programs. There are not high requirements for the installation of the program (around 250 Mb).

Accelerometer [21][22]

The accelerometer is an electromechanical device that is used for sensing acceleration forces ("g-forces", the time rate of velocity changing measured in SI by $\frac{m}{s^2}$), having the capability to measure accelerations either from static (objects pressing constantly on surfaces) or dynamic forces (vibrations or movements of the device).

The accelerometers can be divided in several categories depending on the components and the way of functioning and offering the output. Hence they can differ by the number of axes (single axis, dual axis that could measure accelerations in a plane by X and Y components, and three axes which are used in 3D) and by the sensors they are using (piezoelectric, piezo-resistive, hall effect by hall sensors, magneto resistive).

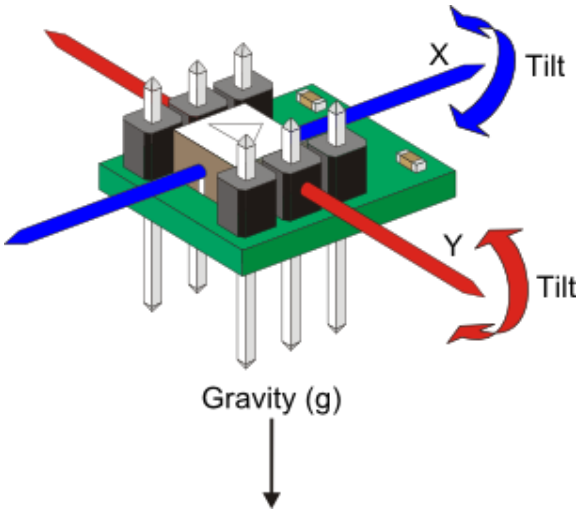


Figure 26. Accelerometer working principle [23].

Dual axis accelerometer

The accelerometer in the project is a Mx2125 type digital piezoelectric on dual axis sensor. The capacity of measured acceleration is up to ±2g. One of the methods for getting output values from the sensor is reading the pulse width modulation duration in microseconds given separately for the X and the Y-axis. This way of processing the data is described more precisely in **Arduino stabilization attempt** part of the report.

Modelling

DC motors

The function of a DC motor is to transform electrical power into mechanical power. The input is voltage (E) and current (I), and the output is torque (T) and speed (ω).

Figure 27. DC Motor block diagram.

DC motor theory [24]

The basic principle of a dc motor is that when a current goes through a magnetic force, it will experience torque, which will cause it to move. Using this principle and a curve magnet design, the armature through which the current is flowing is forced into a circular motion.

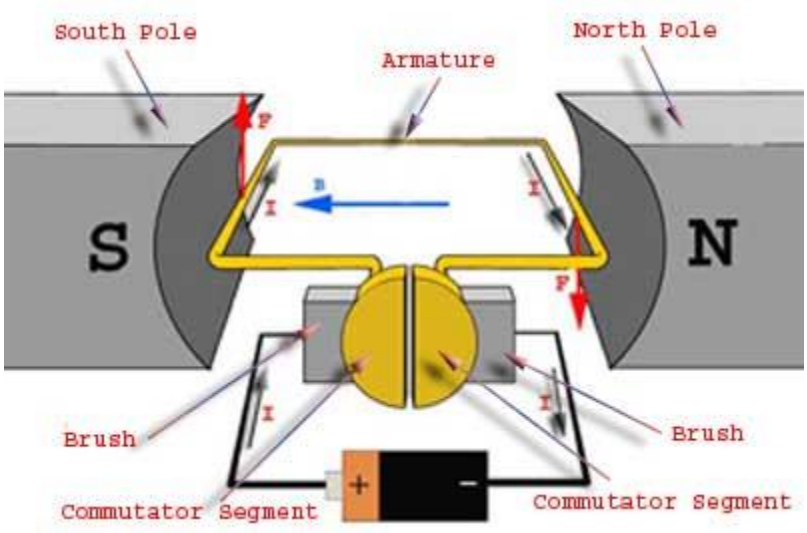


Figure 28. DC Motor parts [24].

In order to calculate the force F, we must use the Lorentz force formula:

$$dF = dq \cdot (E + v - B) \quad (1)$$

Where *F* represents the force, *q* is the charge, *E* is the electric field, *v* is the velocity of the charge and *B* is the magnetic field.

For the operation of a DC motor, it can be considered that *E* = 0. Also, *v* can be replaced by L/t, where L is length of cable and t is time. This results in the following formula:

$$dF = dq \cdot \frac{dL}{dt} \cdot B \quad (2)$$

Now we transform charge over time to current *I*, since they are equivalent. This means that:

$$F = B \cdot I \cdot L \cdot sin\theta \quad (3)$$

Finally, since the force acts perpendicularly to the field, then θ is equal to 90°, and thus its sin is 1, and we arrive at our final formula:

$$F = B \cdot I \cdot L \quad (4)$$

Equally, the force in the other side of the armature will be of the same magnitude, except downwards.

However, it must be said that more important is the torque (τ). The torque will vary, depending on the current position of the armature. The torque is defined by the following equation

$$\tau = \mathrm{F} \cdot \cos\alpha \cdot \mathrm{w} \quad (5)$$

or

$$\tau = \mathrm{B} \cdot \mathrm{I} \cdot \mathrm{L} \cdot \mathrm{w} \cdot \cos\alpha \quad (6)$$

where w is the width of the armature, and α is the angle between the plane of the armature and the reference plane, which is along the magnetic field.

Since α is not constant as θ in the force, this means that the torque will change with time. Following we will analyse the torque for three positions.

The first position is when the planes are equal, and α is 0. This means that the cosine will be equal to 1, and therefore the torque will be

$$\tau = B \cdot I \cdot L \cdot w \tag{7}$$

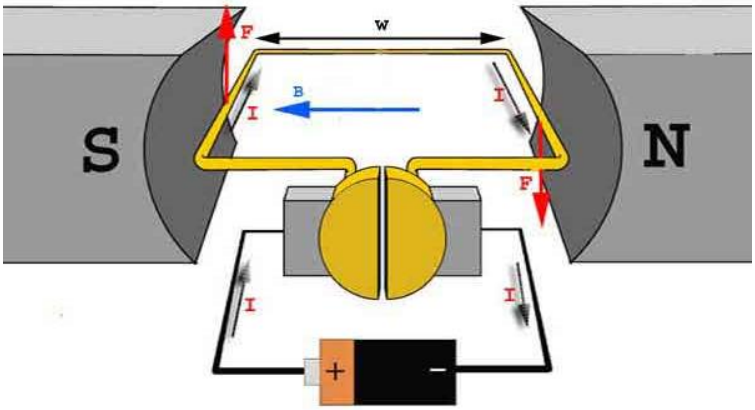


Figure 29. DC Motor shaft initial position [24].

The next position is where 0° < α < 90°. This means that, as α increases, the cosine will decrease, and so will the torque.

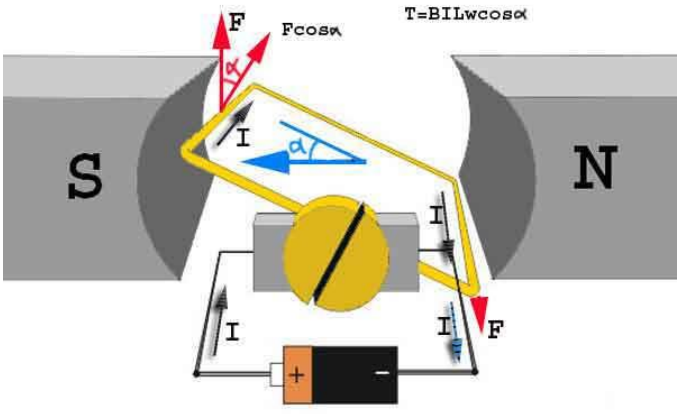


Figure 30. DC Motor shaft in motion [24].

The final step is when the armature is perpendicular to the magnetic field, which will result in cosine 90°, or zero. Therefore, the torque will be zero. This, however, does not mean the armature will stop, since, thanks to inertia, it will rotate further, and the torque will once again increase as α reaches 180°. This cycle will repeat itself every 180°.

Motor model [4][25]

Electrical part

In the circuit below we have the motor 'map' describing the dynamics of the motor spin based of the components inside of it. The current flowing through the engine causes the shaft turn changing the shaft angle. Still the engine is dependent on an interior resistor and inductance for handling the input voltage, which will go through the armature.

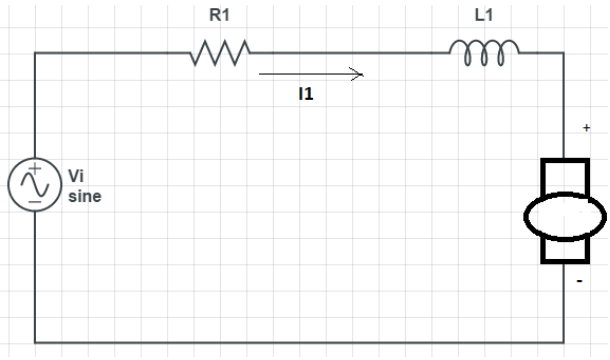


Figure 31. Engine circuit representation.

The supplied input voltage V (24 V in our case) goes through the motor inductance (L_a) and internal resistance(R). The rest of the voltage resulted after going through these is the one going through the coil of the engine generating the rotation, which has the following formula:

$$V_{emf} = Ke \cdot \frac{d\theta m(t)}{dt} = Ke \cdot \dot{\theta} m \qquad (8)$$

Where $\dot{\theta}$ is the shaft's rotational velocity and Ke is the back-electromotive force constant of the motor. By applying Kirchhoff's second law on the circuit, we reach the following relation:

$$V(t) - i_a(t) \cdot R - L_a \cdot \frac{di(t)}{dt} - V_{emf}(t) = 0 \qquad (9)$$

$$V(t) - i_a(t) \cdot R - \mathrm{L}_a \cdot \dot{i_a}(t) - Ke * \dot{\theta}m = 0 \quad (10)$$

Mechanical part

The mechanical part is all about the physical forces generated by the motor and the generated forces as a reaction of these. The electronic part interferes into the mechanical part generating the forces.

Starting with the current that goes through the circuit i_a that generates the torque on the shaft(τ_e), which changes depending on the current, because of the relation of the torque:

$$\tau_e = Kt \cdot i_a(t) \quad (11)$$

where Kt is a torque constant.

The torque that acts opposite to τ_e is called the friction torque which sums three other torques that act oriented to the same way. Those are $\tau_{viscous}$ (dependent on the angular velocity $\dot{\theta}$), τ_{static} (the torque that the motor has to 'defeat' so that it could start spinning) and $\tau_{kinetic}$ (the one needed to keep a constant speed through the spin of the motor). Since the $\tau_{kinetic}$ and τ_{static} are small enough to be considered negligible, they will be considered as 0 in the equation, without impacting the final result.

$$\tau_f = \tau_{viscous} + \tau_{kinetic} + \tau_{static} = \tau_{viscous} = B_m \cdot \frac{d\theta m}{dt} = B_m \cdot \dot{\theta}m \quad (12)$$

B_m is the viscosity constant (depending on materials).

In addition to the friction torque, also the moment of inertia acts as a torque for the motor and has the formula

$$\tau_j = J_m \cdot \frac{d^2}{dt^2} \cdot \theta_m \quad (13)$$

where:

J_m is the moment of inertia for the motor(kg·m² in SI).

Applying the second law of Newton for the torques of the engine, we conclude the following relation:

$$\tau_e - \tau_j - \tau_f = 0 \quad (14)$$

$$\tau_e = \tau_j + \tau_f \quad (15)$$

$$K_e \cdot i_a(t) = J_m \cdot \frac{d^2}{dt^2} \cdot \theta_m + B_m \cdot \dot{\theta} m \quad (16)$$

$$K_e \cdot i_a(t) = J_m \cdot \ddot{\theta}_m + B_m \cdot \dot{\theta} m \quad (17)$$

Motor coefficients:

The coefficients in the following table are the ones found in the previous report, and will be tuned later.

Symbol	Description	Unit	Value (if relevant)
Constant coefficients			
R	Resistance of the motor	Ω	42.8
L_a	Inductance of the motor	H	5.02e-3
K_e	Back-emf constant of the motor	$\frac{V}{rad/s}$	5.51e-2
B_m	Friction constant of the motor	$\frac{N \cdot m}{rad/s}$	**0.190e-2 (untuned)**

K_t	Torque constant of the motor	$\frac{N \cdot m}{V}$	5.51e-2
J_m	Moment of inertia of the motor	$kg \cdot m^2$	**23.8e-6 (untuned)**
Time-dependent coefficients			
V	Voltage applied to the motor	*V*	
i_a	Current of the armature	*A*	
θ	Angle of the shaft of the motor	*Rad*	

Figure 32. Table of values for the motor.

Motor Model Validation

In the *Deriving the transfer functions for the system* section that comes further in the report, we conclude that the motor has a transfer function for Input voltage and output velocity of H(s).

$$\mathrm{H(s)} = \frac{\omega(s)}{V(s)} = \frac{\frac{\mathrm{K}_t}{R}}{J_{mw} \cdot s + \left(B_m + \frac{\mathrm{K}_t \cdot \mathrm{K}_e}{R}\right)} \tag{18}$$

Filling in the variables with their values, we get that this is equivalent to the first order system:

$$\mathrm{H}_{m1}\mathrm{(s)} = \frac{\omega(s)}{V(s)} = \frac{0.6532}{0.0121s+1} \tag{19}$$

In this form, we can see that it has a DC-Gain (k) of 0.6532, which means that the input to output ratio will be 65.32%. The time constant (τ), which reflects the time it will take to reach the maximum output, is 0.0121.

In order to validate our theoretical transfer function, we tested our engine by giving it a step input, from which we got the following graph:

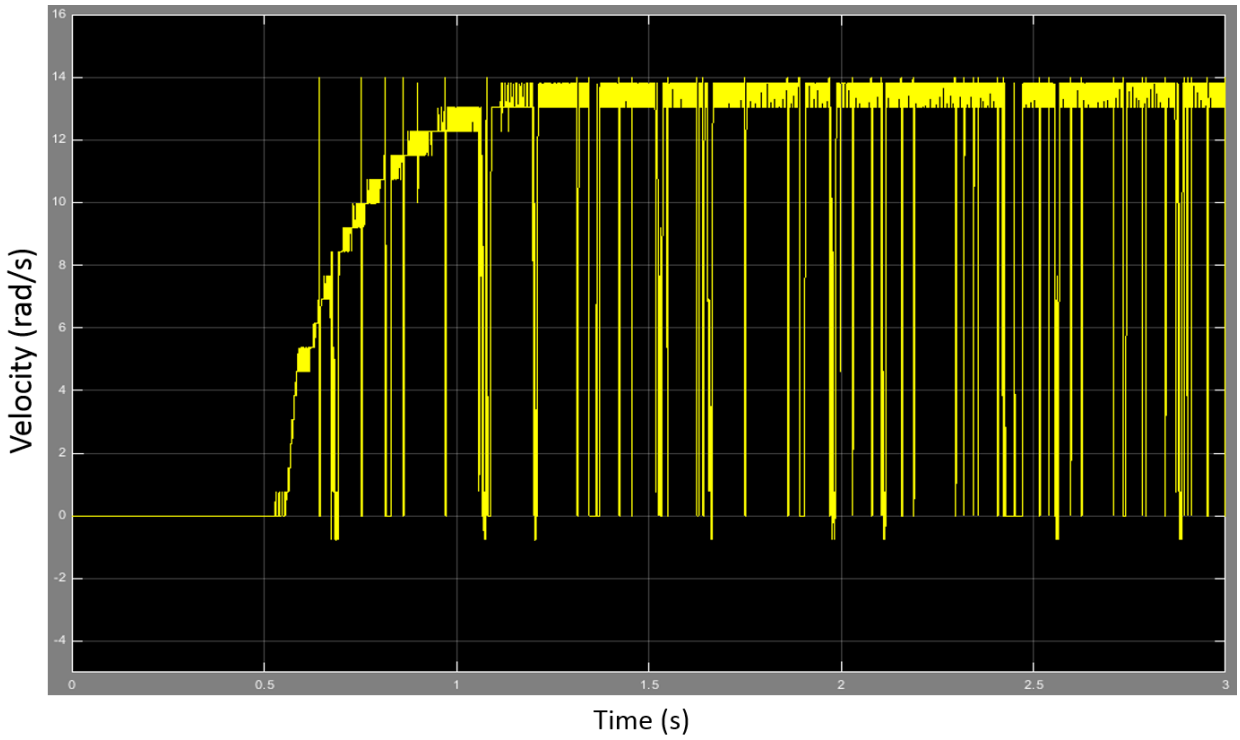


Figure 33. Velocity read by encoder graph.

The used input into the motor was 24 volts with a 100% duty cycle PWM. As seen in the graph, this yielded a maximum output of approximately 13.7. From this, we get a DC-Gain of:

$$K_m = \frac{13.7}{24} = 0.5708\overline{3} \tag{20}$$

We can also get a time constant from this curve. There are several methods to determine the time constant from a graph, but we will record the time it takes the transient to move from the starting value (0) to 63% of the steady state output which in this case is 8.631. Looking at the graph, we determine that it is approximately 0.2.

With these two values, we can determine our practical transfer function:

$$\mathrm{H}_{m2}(\mathrm{s}) = \frac{\omega(s)}{V(s)} = \frac{0.5708\overline{3}}{0.2s+1} \tag{21}$$

Now that we have two transfer functions for the motor, we compare their outputs to a step input, a ramp input and a sinusoid. We use the following Simulink model:

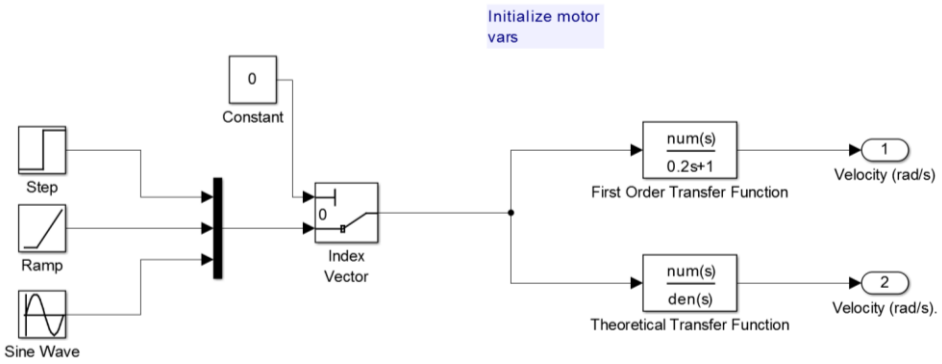


Figure 34. Transfer function comparison.

We select the index in the multiplexed signal to determine which input we are using, and record the results in the scope. Both the step and the sinusoid have an amplitude of 24, since it is our maximum input.

Step Impulse:

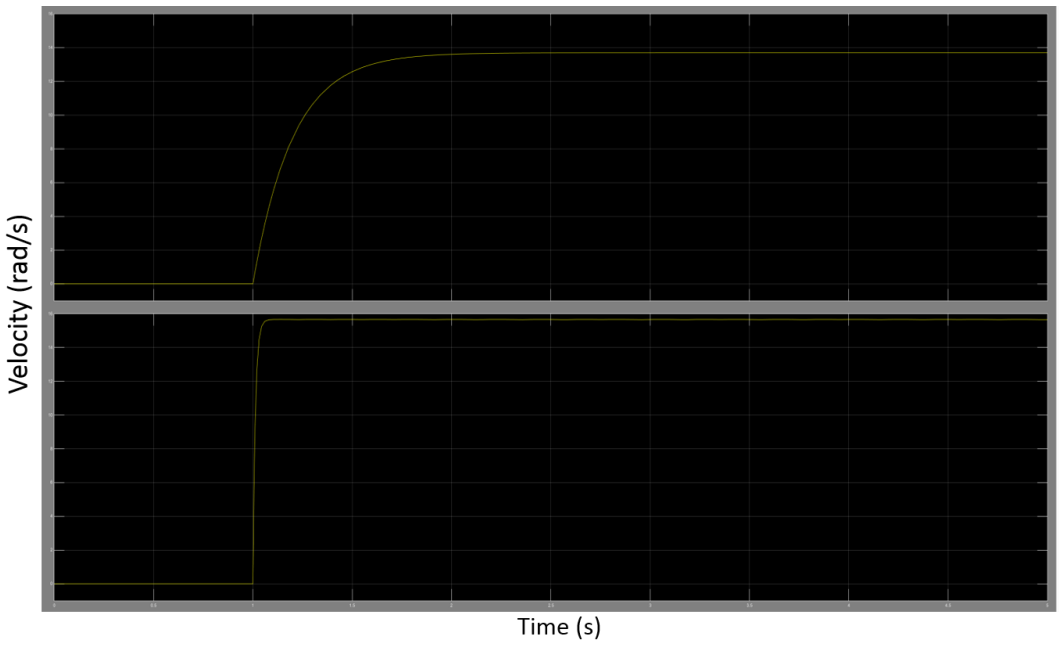


Figure 35. Step input response.

Ramp Input:

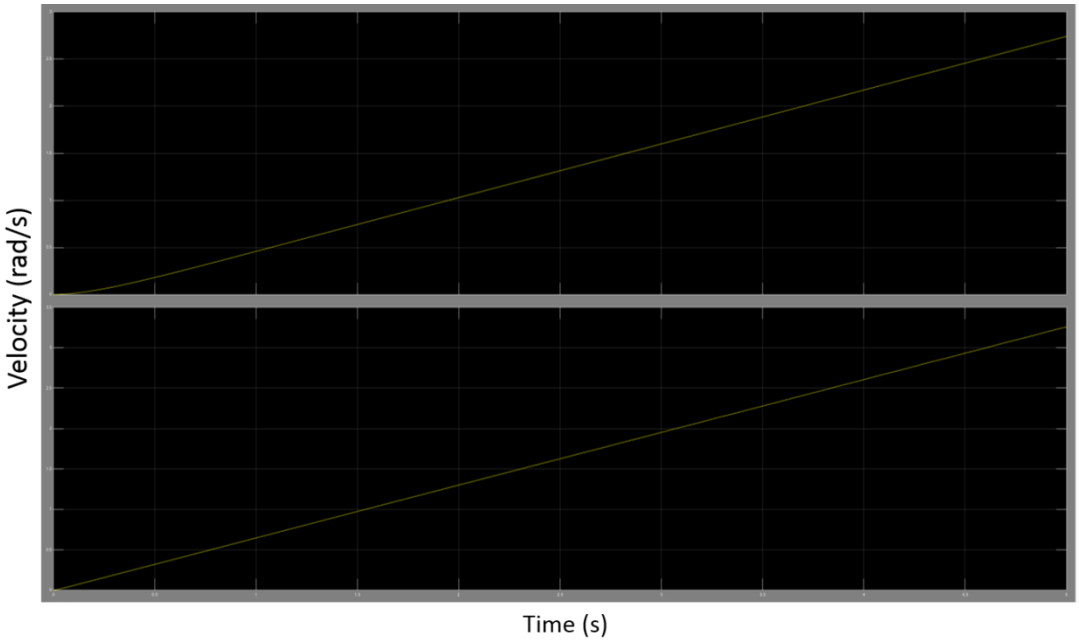


Figure 36. Ramp input response.

Sinusoid Input:

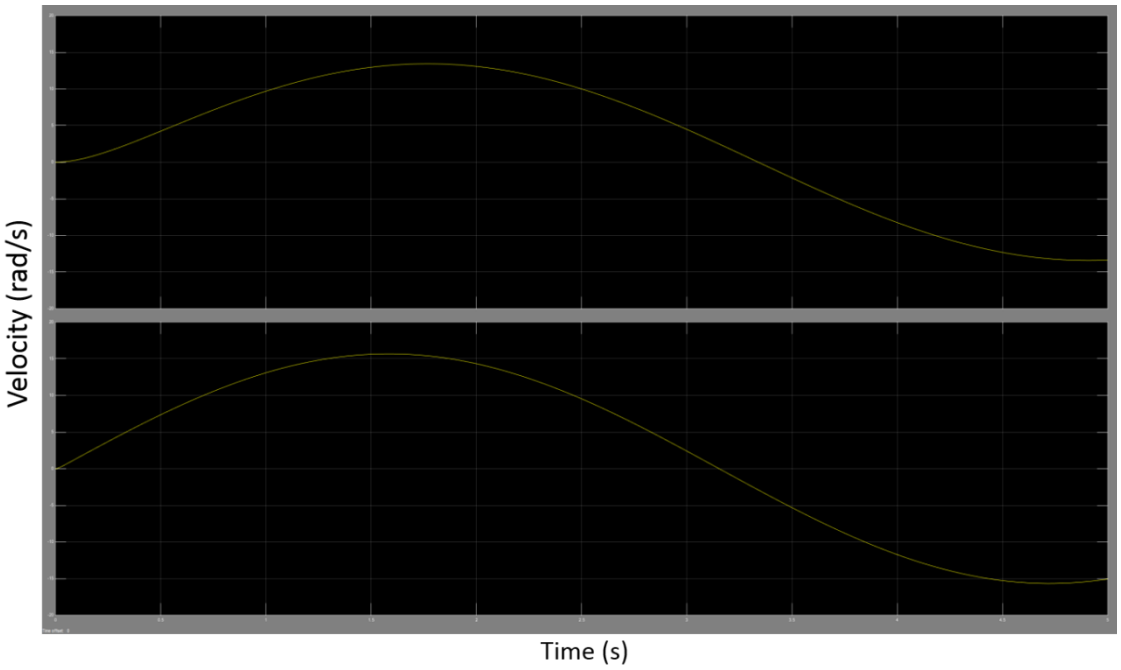


Figure 37. Sinusoid input response.

From the step input, we clearly verify that the real life DC-Gain is lower, since the output is lower. However, more importantly, the transfer function obtained in real life has a much slower reaction than the theoretical transfer function. The same is reflected in the ramp input that curves slightly in the start of the ramp, and ends in a lower output after 5 seconds than the theoretical transfer function. Finally, in the sinusoid, we can see the same pattern, with reduced amplitude, but also it is clearly seen that the output sinusoid is delayed, and thus the two outputs are out of phase of each other by the difference in the time constants.

Motor coefficients tuning

From this we conclude that we have to tune our motor variables, in order to properly represent our system. For this we changed the inertia in order to slow down the speed, and we changed the friction to decrease the final output. The new values found where:

$$J_m = 28.8\, e - 5 \tag{22}$$

$$B = 0.22\, e - 2 \tag{23}$$

With these values, when we compare the two transfer functions, we get very similar graphs, like shown next.

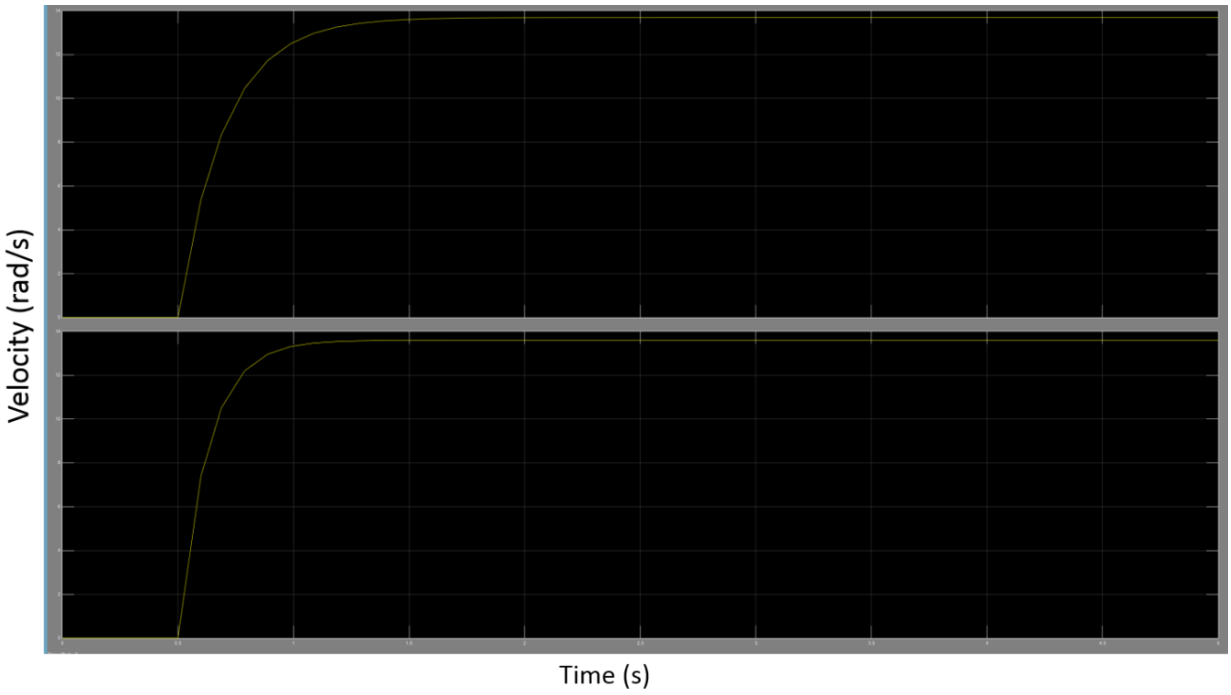


Figure 38. Tuned model values comparison.

Conclusion of motor tuning

The inertia and friction are changed in the model, in order to have a model that represents the real life plant. However, this will only help to a certain extent, since we can see that the friction depends a lot on the encoder plate. Since the plate is not fastened, it means that its position changes and so does the friction therefore. Our work assumes that the variation in friction is small enough so it will not stop the prototype from working.

Additionally, we are not sure if we trust the DC-Gain value read with the sensor, since we noticed that it had some issues reading the position accurately, after several revolutions. Nevertheless, the fact that the final velocity stabilizes in a value, indicates that there is no error being added every x amount of revolutions, which is a good sign.

Pendulum mathematical model [4]

Angular momentum basics [26]

In order to calculate the angular momentum of a pendulum it is important to understand the angular momentum of a particle with respect to its origin.

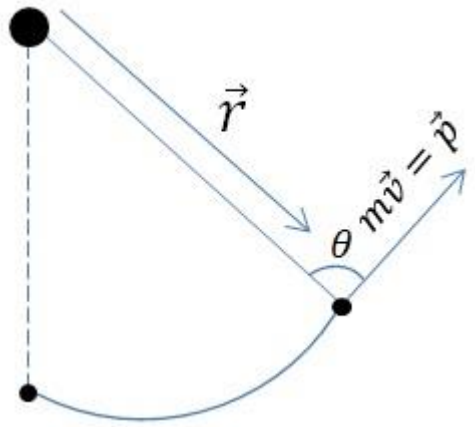


Figure 39. Angular momentum of a particle.

The angular momentum $\vec{\mathrm{L}}$ of a particle with respect to a chosen origin is given by:

$$\vec{\mathrm{L}} = m \cdot \vec{v} \cdot \vec{r} \cdot \sin\theta \tag{24}$$

where:

m is the mass of the particle.

$\vec{v}$ is the velocity.

$\vec{r}$ is the position of the particle from the origin.

θ is the angle between *r* and $m \cdot \vec{v}$.

The equation (24) can be more formally rewritten as:

$$\vec{\mathrm{L}} = \vec{r} \times \vec{p} \tag{25}$$

where $\vec{p} = m \cdot \vec{v}$, which is the linear momentum of the particle.

We can disregard the angular momentum sign for a moment, since the angle θ for the case of a pendulum will always be π/2 or 3π /2 (so that sin(θ)=1 or sin(θ)=-1).

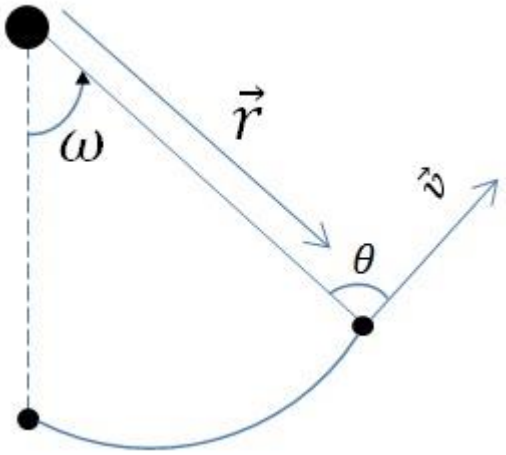


Figure 40. Angular momentum of a particle with velocity.

Figure 40 is similar to the Figure 39, except the velocity and angular velocity of a particle are pictured.

Angular velocity ω can be obtained from the following equation:

$$\omega = \frac{\vec{v}}{\vec{r}} \cdot \sin\theta \tag{26}$$

which allows us to rewrite the equation (24) as:

$$\mathrm{L} = m \cdot \vec{r} \cdot \vec{r} \cdot \ \omega \cdot \sin\theta \tag{27}$$

or, disregarding the sine:

$$\mathrm{L} = m \cdot \vec{r}^{2} \cdot \ \omega \tag{28}$$

Now, we can apply the same principle for the angular momentum of a pendulum, considering the pendulum's center of gravity as the single particle from the previous equations.

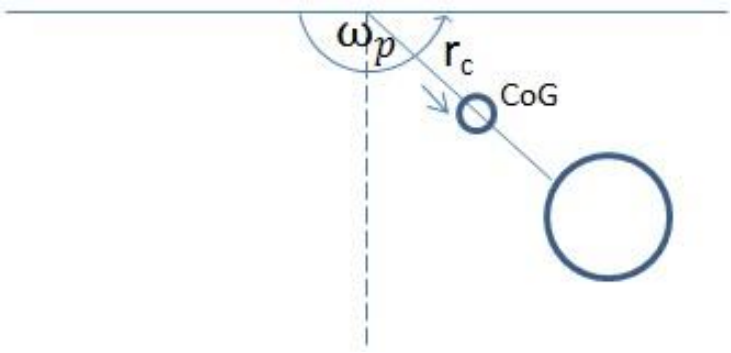


Figure 41. Angular momentum of a pendulum.

Thus, equation (24) for the pendulum looks the following way:

$$\vec{L}_p = m_p \cdot {r_c}^2 \cdot \omega_p \tag{29}$$

where:

m_p is the total mass of the pendulum (constant value).

r_c is the distance from the origin point to the pendulum's CoG (center of gravity) (constant value).

ω_p is the angular velocity of the pendulum.

So, differentiation of (29) with respect to time (t) allows us to get:

$$\dot{\vec{L}_p} = m_p \cdot {r_c}^2 \cdot \dot{\omega}_p \tag{30}$$

Angular momentum of a reaction wheel [27]

The reaction wheel angular momentum is also a crucial part, required in order to compose a complete equation for our system.

The angular momentum expression for the wheel looks the following way:

$$L_{wh} = I_{wh} \cdot \omega_{wh} \tag{31}$$

where:

L_{wh} is the angular momentum of the wheel.

I_{wh} is the moment of inertia of the wheel.

ω_{wh} is the angular velocity of the wheel.

Differentiating (31) with respect to time, we get:

$$\dot{L_{wh}} = I_{wh} \cdot \dot{\omega_{wh}} \tag{32}$$

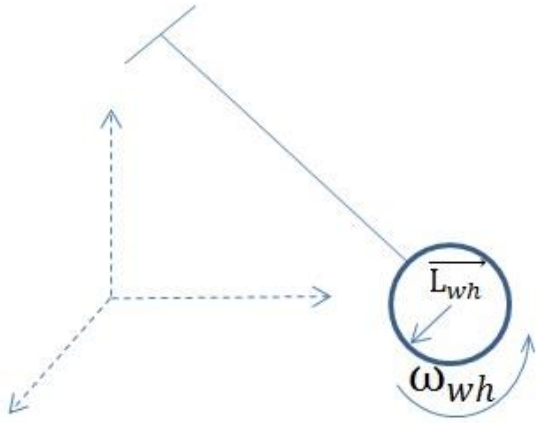


Figure 42. Angular momentum of a reaction wheel.

Torque originating from gravity [28]

The equation of torque gives us:

$$\tau = I \cdot \alpha \tag{33}$$

The equation of torque generated by gravity is:

$$\overrightarrow{\tau_g} = m_p \cdot g \cdot r_c \cdot \sin\theta \tag{34}$$

where:

m_p is the mass of the pendulum.

r_c is the distance from the origin point to the pendulum's CoG (center of gravity).

θ is the angle of the pendulum.

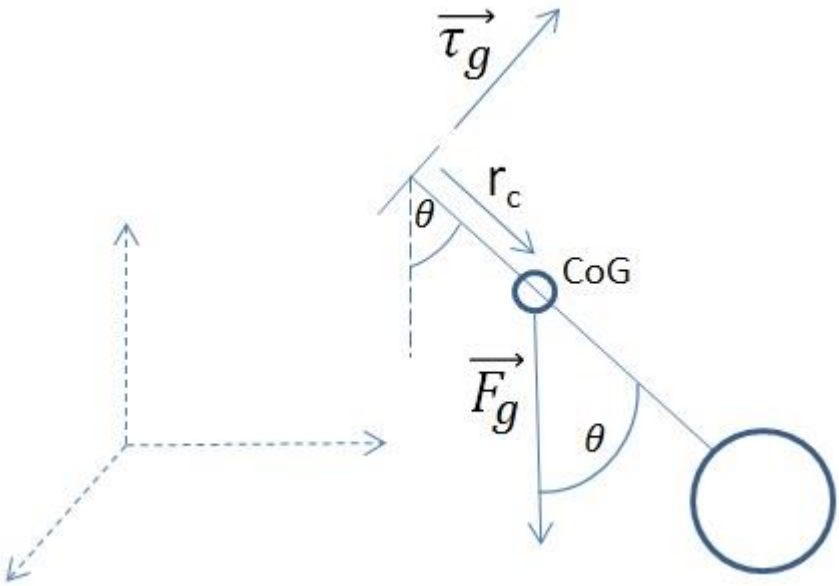


Figure 43. Torque, originating from gravity.

Torque originating from friction [4]

The other part of the torque is the torque originating from friction. The equation for the torque depends on the pendulum angular velocity.

$$\overrightarrow{\tau_f} = B \cdot \omega_p \tag{35}$$

where:

$\overrightarrow{\tau_f}$ is the torque originated from the friction.

B is the pendulum friction constant.

ω_p is the angular velocity of the pendulum.

The friction torque is just a small part of the overall torque; however, it affects directly the stopping of the pendulum swinging. Therefore, it is important to take it into consideration along with the torque originating from gravity.

Both $\overrightarrow{\tau_f}$ and $\overrightarrow{\tau_g}$ will face the opposite direction of the pendulum's angular momentum.

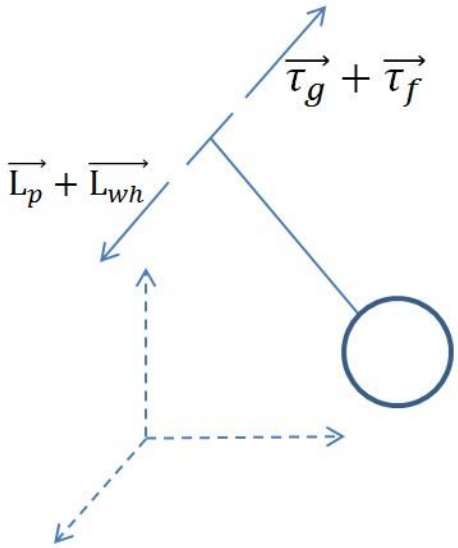


Figure 44. The torques and angular momentums of the system.

The final equation of the pendulum model considering the opposite directions of the angular momentum and the torques looks the following way:

$$\dot{L_p} + \dot{L_{wh}} = -\tau_f - \tau_g \tag{36}$$

Inserting the equations for each element, we get the final equation for the pendulum:

$$m_p \cdot {r_c}^2 \cdot \dot{\omega}_p + I_{wh} \cdot \dot{\omega_{wh}} = -B \cdot \omega_p - m_p \cdot g \cdot r_c \cdot \sin\theta \tag{37}$$

Pendulum coefficients:

Symbol	Description	Unit	Value (if relevant)
Constant coefficients			
m_p	Mass of the pendulum	*Kg*	1.255
r_c	Length from the origin point to the pendulum's center of gravity	m	0.245
I_{wh}	Moment of inertia of the reaction wheel	$kg \cdot m^2$	6.577e-3
B	Pendulum's friction constant	$\frac{N \cdot m}{rad/s}$	**0.14 (untuned)**
g	Gravity of Earth	$\frac{m}{s^2}$	9.82
Time-dependent coefficients			
ω_p	Angular velocity of the pendulum	$\frac{rad}{s}$	
ω_{wh}	Angular velocity of the wheel	$\frac{rad}{s}$	
θ	Angle of the pendulum	*Rad*	

Figure 45. Table of coefficients for the pendulum

Pendulum validation

In this section we will tune the pendulum coefficients. In order to do so we must compare how the transfer function responds to certain conditions compared to our real life prototype. We decided that these conditions would be a pendulum falling from the horizontal axis. In order to cause these conditions in real life we simply manually put it at the desired position. In the case of the model, we input an impulse that will push it up until the desired position, and start measuring it there. The results acquired are shown in the next figures.

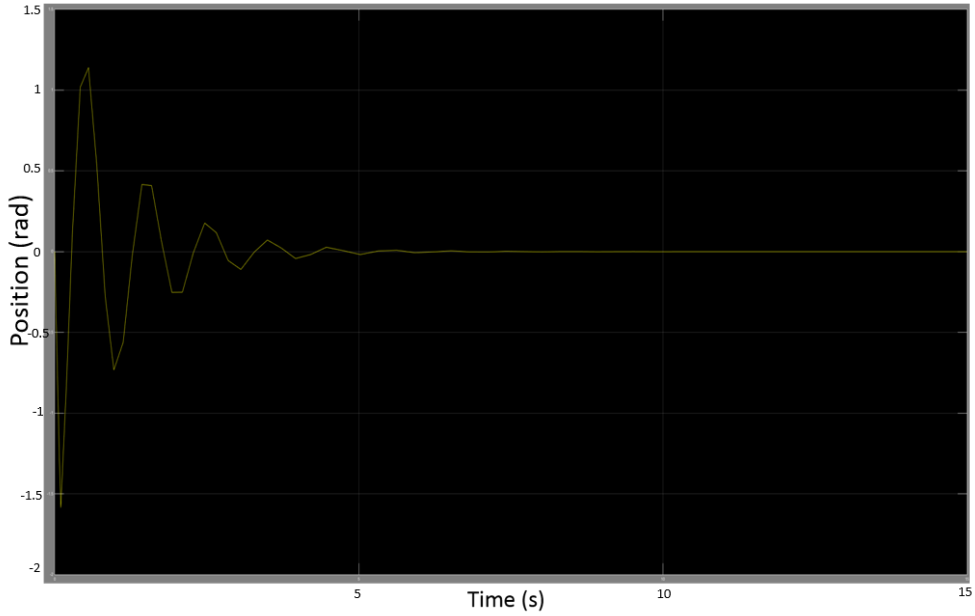


Figure 46. Transfer function untuned values.

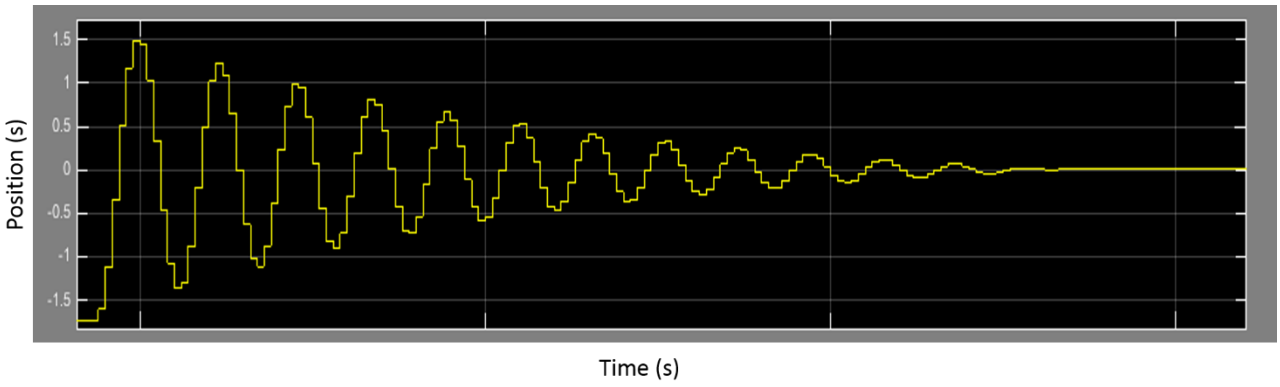


Figure 47. Encoder readings release -π/2 degrees

It is evident that the responses are very different from each other. We can see that the model decays into a static position much faster, which tells us we want to reduce the friction value. With this in mind, we tuned the friction so that the output positions would be similar. The obtained value was

$$B = 0.04 \tag{38}$$

The resulting curve obtained with the new value is shown in the next figure.

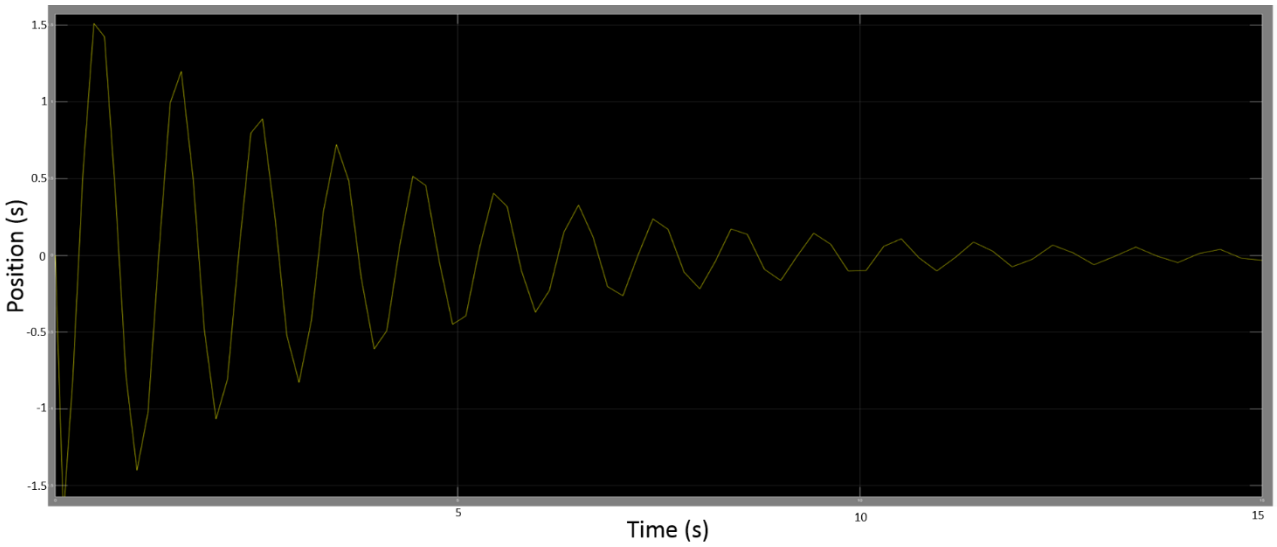


Figure 48. Tuned value output transfer function.

This curve is very similar to the real life plant. In the beginning the results are identical, but deviate with time, especially in the later part, where the real life plant decays into no movement, while the model keeps swinging in a decreasing fashion for longer.

Conclusion of pendulum tuning

In conclusion, the tuning has been done to represent the new friction values with the new parts and the loosened encoder. More detailed tuning will take much more work to make it more accurate, and will bring little benefit, since the error caused by the small oscillations is assumed negligible. Thus, we conclude that the tuning is good enough to support our control technique.

This means that the transfer function, which is obtained in the following section, will have the following coefficients:

$$\mathrm{H(s)} = \frac{\theta(s)}{\omega_{wh}(s)} = \frac{-I_{wh} \cdot s}{m_p \cdot {r_c}^2 \cdot s^2 + B \cdot s + m_p \cdot g \cdot r_c} \tag{39}$$

$$\mathrm{H(s)} = \frac{\theta(s)}{\omega_{wh}(s)} = \frac{-0.00028 \cdot s}{1.255 \cdot 0.245^2 \cdot s^2 + 0.4 \cdot s + 1.255 \cdot 9.82 \cdot 0.245} \tag{40}$$

Deriving the transfer functions for the system

Transfer function for the pendulum

For a hanging pendulum, using the equation (37) from the pendulum mathematical modelling section, we can derive a transfer function for getting a pendulum angle depending on the wheel angular velocity:

$$m_p \cdot {r_c}^2 \cdot \dot{\omega}_p + I_{wh} \cdot \dot{\omega_{wh}} = -B \cdot \omega_p - m_p \cdot g \cdot r_c \cdot \sin\theta \tag{41}$$

The above equation is clearly non-linear, since it has a sine term. Via linearization $\sin\theta$ can be rewritten as simply θ due to the small-angle approximation principle.

The principle works thanks to the Taylor series expansion for the trigonometric functions. For small angles, the trigonometric functions can be approximated by the first term in their corresponding series. The first term of the sine series is equal to the argument of the sine; hence, for small θ, $\sin\theta \approx \theta$. The error becomes more significant after 0.4 radians or 23°, reaching 2.5%, as can be seen in the graph below [29].

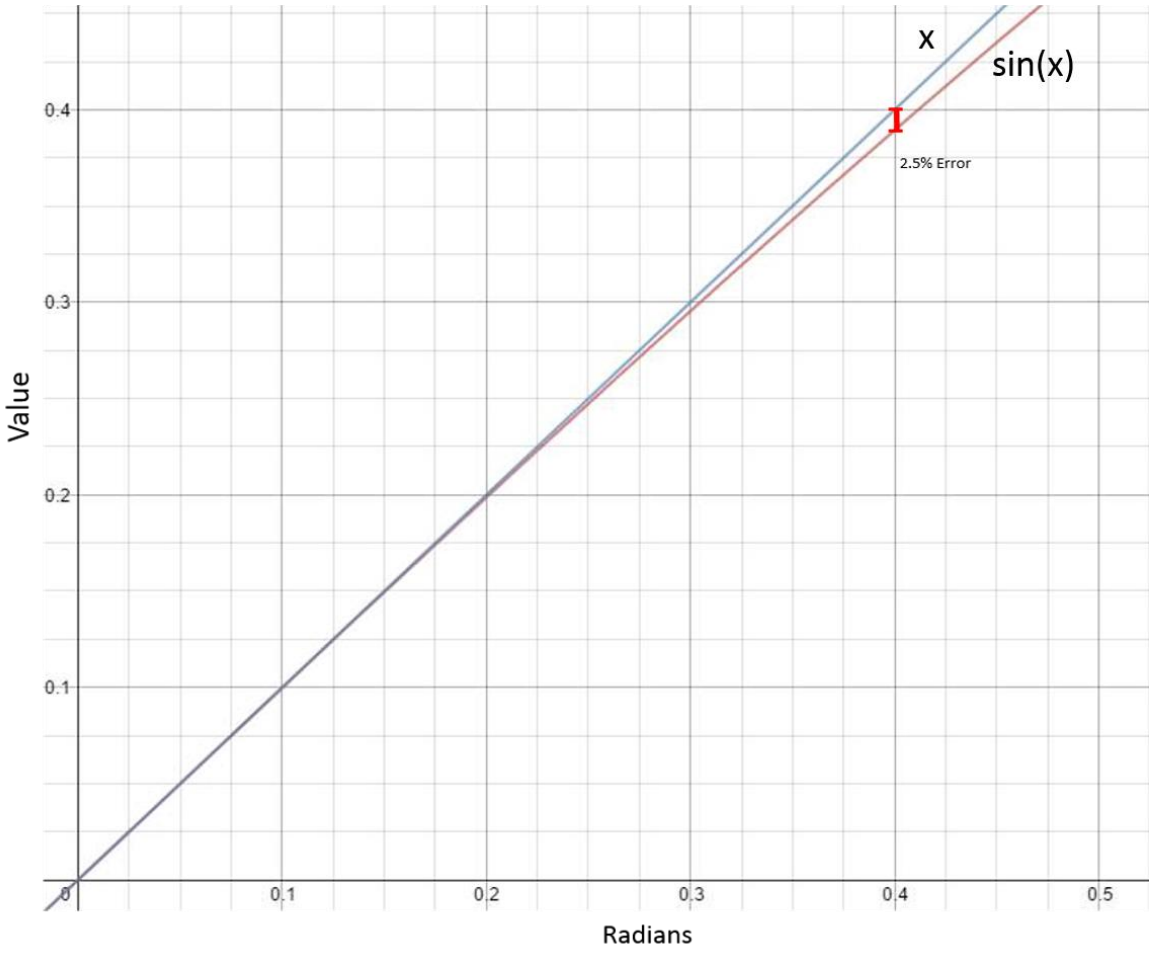


Figure 49. Comparison of graphs of sin(x) and x.

Since our pendulum angle is not supposed to deviate more than 0.4 rad from the stabilization point, the linearization can be used.

Therefore, equation (41) can be rewritten as:

$$m_p \cdot {r_c}^2 \cdot \dot{\omega}_p(t) + I_{wh} \cdot \dot{\omega_{wh}}(t) = -B \cdot \omega_p(t) - m_p \cdot g \cdot r_c \cdot \theta(t) \quad (42)$$

ω_p being a first derivative of θ:

$$m_p \cdot {r_c}^2 \cdot \ddot{\theta}(t) + I_{wh} \cdot \dot{\omega_{wh}}(t) = -B \cdot \dot{\theta}(t) - m_p \cdot g \cdot r_c \cdot \theta(t) \quad (43)$$

Transforming equation (43) into frequency domain, considering the initial conditions to be equal to 0, yields:

$$m_p \cdot {r_c}^2 \cdot s^2 \cdot \theta(s) + B \cdot s \cdot \theta(s) + m_p \cdot g \cdot r_c \cdot \theta(s) = -I_{wh} \cdot \mathrm{s} \cdot \omega_{wh}(s) \quad (44)$$

where:

ω_{wh} is the input (the angular velocity of the wheel, which is depending on the supplied voltage).

θ is the output (the angle of the pendulum).

Rewriting equation (44), we get:

$$\theta(s)\left(m_p \cdot {r_c}^2 \cdot s^2 + B \cdot s + m_p \cdot g \cdot r_c\right) = -I_{wh} \cdot \mathrm{s} \cdot \omega_{wh}(s) \quad (45)$$

Therefore, the transfer function for this relation will be:

$$\mathrm{H(s)} = \frac{\theta(s)}{\omega_{wh}(s)} = \frac{-I_{wh} \cdot s}{m_p \cdot {r_c}^2 \cdot s^2 + B \cdot s + m_p \cdot g \cdot r_c} \quad (46)$$

Transfer function for the motor

In the DC motor mathematical modelling section, we got the following equations, characterizing the mechanical and electrical parts of the motor:

$$\mathrm{K}_t \cdot i_a = J_{mw} \cdot \ddot{\theta_w} + B_m \cdot \dot{\theta_w} \quad (47)$$

$$\mathrm{V} - \mathrm{K}_e \cdot \dot{\theta_w} = i_a(t) \cdot R + L_a \cdot \dot{i_a}(t) \quad (48)$$

In many cases, it is possible to ignore the inductance in the model, since its relative effect is insignificant compared with the mechanical motion. Considering it to be true for our case, we can neglect L_a in equation (48). [25]

Therefore, we can combine equations (47) and (48) to get:

$$\frac{\mathrm{K}_t}{R} \cdot V = J_{mw} \cdot \ddot{\theta_w} + \left(B_m + \frac{\mathrm{K}_t \cdot \mathrm{K}_e}{R}\right) \cdot \dot{\theta_w} \quad (49)$$

Transforming equation (49) into Laplace domain, considering the initial conditions to be equal to 0, yields:

$$\frac{\mathrm{K}_t}{R} \cdot V(s) = J_{mw} \cdot \theta_w(s) \cdot s^2 + \left(B_m + \frac{\mathrm{K}_t \cdot \mathrm{K}_e}{R}\right) \cdot \theta_w(s) \cdot s \quad (50)$$

where:

V is the input (the voltage supplied to the motor).

θ_w is the output (the angular position of the shaft of the motor, which is equal to the angular position of the reaction wheel).

Rewriting equation (50), we get:

$$\frac{K_t}{R} \cdot V(s) = \theta_w(s) \cdot (J_{mw} \cdot s^2 + \left(B_m + \frac{K_t \cdot K_e}{R}\right) \cdot s) \tag{51}$$

Thus, the transfer function will be:

$$H(s) = \frac{\theta_w(s)}{V(s)} = \frac{\frac{K_t}{R}}{J_{mw} \cdot s^2 + \left(B_m + \frac{K_t \cdot K_e}{R}\right) \cdot s} \tag{52}$$

Since the output in a form of an angular velocity is required ($\omega = \dot{\theta_w}$), we modify our transfer function, getting:

$$H(s) = \frac{\omega(s)}{V(s)} = \frac{\frac{K_t}{R}}{J_{mw} \cdot s + \left(B_m + \frac{K_t \cdot K_e}{R}\right)} \tag{53}$$

Combining the transfer functions

In order to get the complete transfer function for the entire system, we combine the transfer functions for the motor and the pendulum represented in the equations (46) and (53).

Since the output of the first transfer function should be the input of the second, we can simply multiply them to get the desired transfer function.

$$H(s) = \frac{-I_{wh} \cdot s}{m_p \cdot r_c^2 \cdot s^2 + B \cdot s + m_p \cdot g \cdot r_c} \cdot \frac{\frac{K_t}{R}}{J_{mw} \cdot s + \left(B_m + \frac{K_t \cdot K_e}{R}\right)} \tag{54}$$

Rewriting it into a standard form:

$$\frac{-I_{wh}\cdot\frac{K_t}{R}\cdot s}{(m_p\cdot r_c{}^2\cdot J_{mw})\cdot s^3+(m_p\cdot r_c{}^2\cdot(B_m+\frac{K_t\cdot K_e}{R})+B\cdot J_{mw})\cdot s^2+(B\cdot(B_m+\frac{K_t\cdot K_e}{R})+m_p\cdot g\cdot r_c\cdot J_{mw})\cdot s+m_p\cdot g\cdot r_c\cdot(B_m+\frac{K_t\cdot K_e}{R})} \quad (55)$$

It is important to note, that the equation (55) represents a complete transfer function for a non-inverted pendulum; in order to make the pendulum inverted, all the g terms must be replaced by $-g$.

State Space [4][30][31]

As when deriving the transfer function we start with:

$$K_t \cdot i_a = J_{mw} \cdot \ddot{\theta_w} + B_m \cdot \dot{\theta_w} \quad (56)$$

$$V - K_e \cdot \dot{\theta_w} = i_a \cdot R + L_a \cdot \dot{i_a} \quad (57)$$

$$m_p \cdot r_c{}^2 \cdot \ddot{\theta} + I_{wh} \cdot \dot{\theta_w} = -B \cdot \dot{\theta} - m_p \cdot g \cdot r_c \cdot \theta \quad (58)$$

From these formulas, we need to find all the states needed. In our case we have θ and $\dot{\theta}$ in the last equation, $\dot{\theta_w}$ in the second and i_a in the first. Now we organize them into a state vector:

$$x \triangleq \begin{bmatrix}\theta & \dot{\theta} & \dot{\theta_w} & i_a\end{bmatrix}^T \quad (59)$$

This is transformed into variables of x.

$$x = \begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix} = \begin{bmatrix}\theta\\\dot{\theta}\\\dot{\theta_w}\\i_a\end{bmatrix} \quad (60)$$

We also know that our input will simply be voltage (V), and our outputs will be x_1, x_2 and x_3, so we can evaluate not only the position, but also the current velocities of both the wheel and the pendulum. Expressing this in vectors, we get the following:

$$u = V \quad (61)$$

$$y = \begin{bmatrix} \theta \\ \dot{\theta} \\ \dot{\theta_w} \end{bmatrix} \tag{62}$$

Now we transform out equations in terms of the x vector. The result is these equations.

$$\mathrm{K}_t \cdot x_4 = J_{mw} \cdot \dot{x_3} + B_m \cdot x_3 \tag{63}$$

$$\mathrm{V} - \mathrm{K}_e \cdot x_3 = x_4 \cdot R + L_a \cdot \dot{x_4} \tag{64}$$

$$m_p \cdot {r_c}^2 \cdot \dot{x_2} + I_{wh} \cdot \dot{x_3} = -B \cdot x_2 - m_p \cdot g \cdot r_c \cdot x_1 \tag{65}$$

Since we can see that x_1 and x_2 are linked, we have our first A-matrix line formula:

$$\dot{x_1} = x_2 \tag{66}$$

Then we can get the other three formulas by rearranging the x formulas.

$$\dot{x_3} = \frac{-B_m}{J_{mw}} \cdot x_3 + \frac{\mathrm{K}_t}{J_{mw}} \cdot x_4 \tag{67}$$

$$\dot{x_4} = x_4 \cdot \frac{R}{L_a} + \frac{\mathrm{K}_e}{L_a} \cdot x_3 - \frac{1}{L_a} \cdot \mathrm{u} \tag{68}$$

$$\dot{x_2} = \frac{-B}{m_p \cdot {r_c}^2} \cdot x_2 - \frac{g}{r_c} \cdot x_1 - \frac{I_{wh}}{m_p \cdot {r_c}^2} \cdot \dot{x_3} \tag{69}$$

Since the last formula has a $\dot{x_3}$, we must insert the formula of it.

$$\dot{x_2} = \frac{-B}{m_p \cdot {r_c}^2} \cdot x_2 - \frac{g}{r_c} \cdot x_1 - \frac{I_{wh}}{m_p \cdot {r_c}^2} \cdot \left(\frac{-B_m}{J_{mw}} \cdot x_3 + \frac{\mathrm{K}_t}{J_{mw}} \cdot x_4\right) \tag{70}$$

Organizing our new formulas, we get our A matrix, by using the respective coefficients.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-g}{r_c} & \frac{-B}{m_p \cdot {r_c}^2} & \frac{B_m \cdot I_{wh}}{m_p \cdot {r_c}^2 \cdot J_{mw}} & \frac{-I_{wh} \cdot \mathrm{K}_t}{m_p \cdot {r_c}^2 \cdot J_{mw}} \\ 0 & 0 & \frac{-B_m}{J_{mw}} & \frac{\mathrm{K}_t}{J_{mw}} \\ 0 & 0 & \frac{\mathrm{K}_e}{L_a} & \frac{R}{L_a} \end{bmatrix} \tag{71}$$

The B matrix is the control matrix, which shows how the input affects the state changes. Our B matrix is:

$$B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{L_A} \end{bmatrix} \tag{72}$$

The output part of the state space contains matrix C, which determines the relation between the states and the output, and matrix D, which represents the feed-forward (in our system 0):

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{73}$$

$$D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{74}$$

Putting all this together gives us the State-Space representation:

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-g}{r_c} & \frac{-B}{m_p \cdot {r_c}^2} & \frac{B_m \cdot I_{wh}}{m_p \cdot {r_c}^2 \cdot J_{mw}} & \frac{-I_{wh} \cdot \mathrm{K}_t}{m_p \cdot {r_c}^2 \cdot J_{mw}} \\ 0 & 0 & \frac{-B_m}{J_{mw}} & \frac{\mathrm{K}_t}{J_{mw}} \\ 0 & 0 & \frac{\mathrm{K}_e}{L_a} & \frac{R}{L_a} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{1}{L_A} \end{bmatrix} - u \tag{75}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \cdot u \tag{76}$$

Transfer function analysis

Now that we have the final combined transfer function, as well as the tuned values, we can finally express the transfer function with number coefficients. The transfer function is the following.

$$H(s) = \frac{-3.605 \cdot 10^{-7} \cdot \mathrm{s}}{2.17 \cdot 10^{-5} \cdot \mathrm{s}^3 + 0.0001826\ \mathrm{s}^2 - 0.0007788\ \mathrm{s} - 0.006857} \tag{77}$$

Now we analyse the step response, and get the following plot.

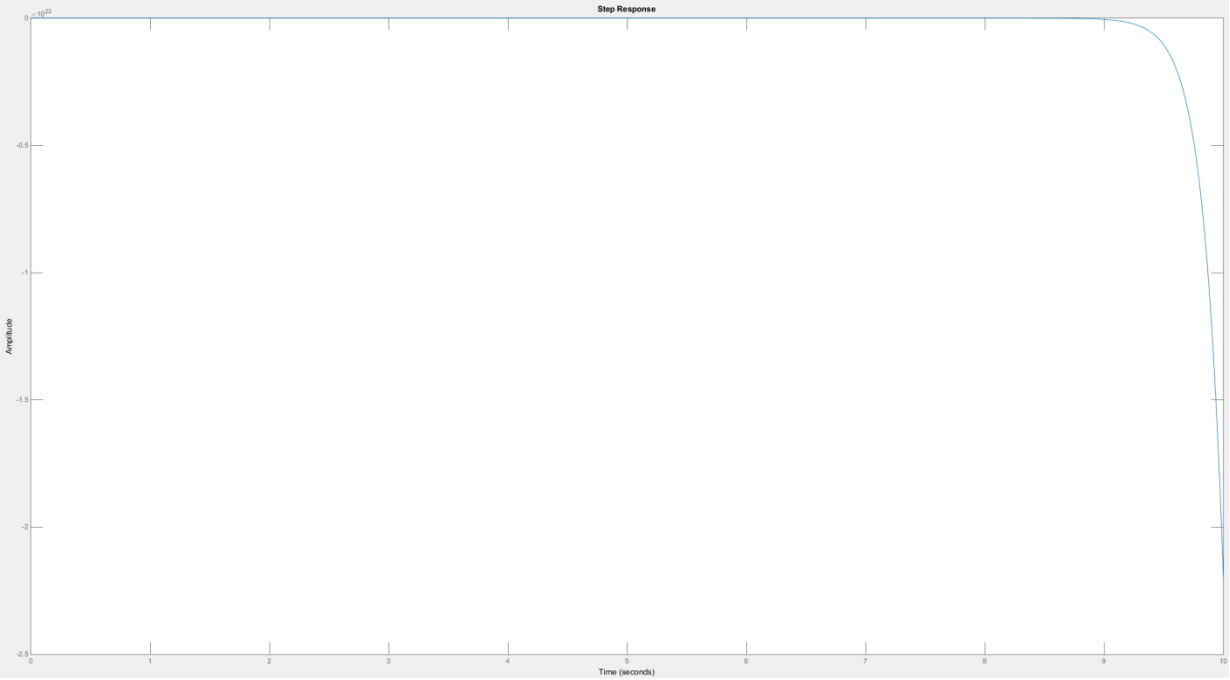


Figure 50. Step response.

From this we can determine that the system is unstable since the output tends to infinity. We can predict that there will be a pole in the positive real region, and, judging by the lack of oscillations, we can predict that it will have no imaginary component. To verify this we find the pole and zero plot, which is the following:

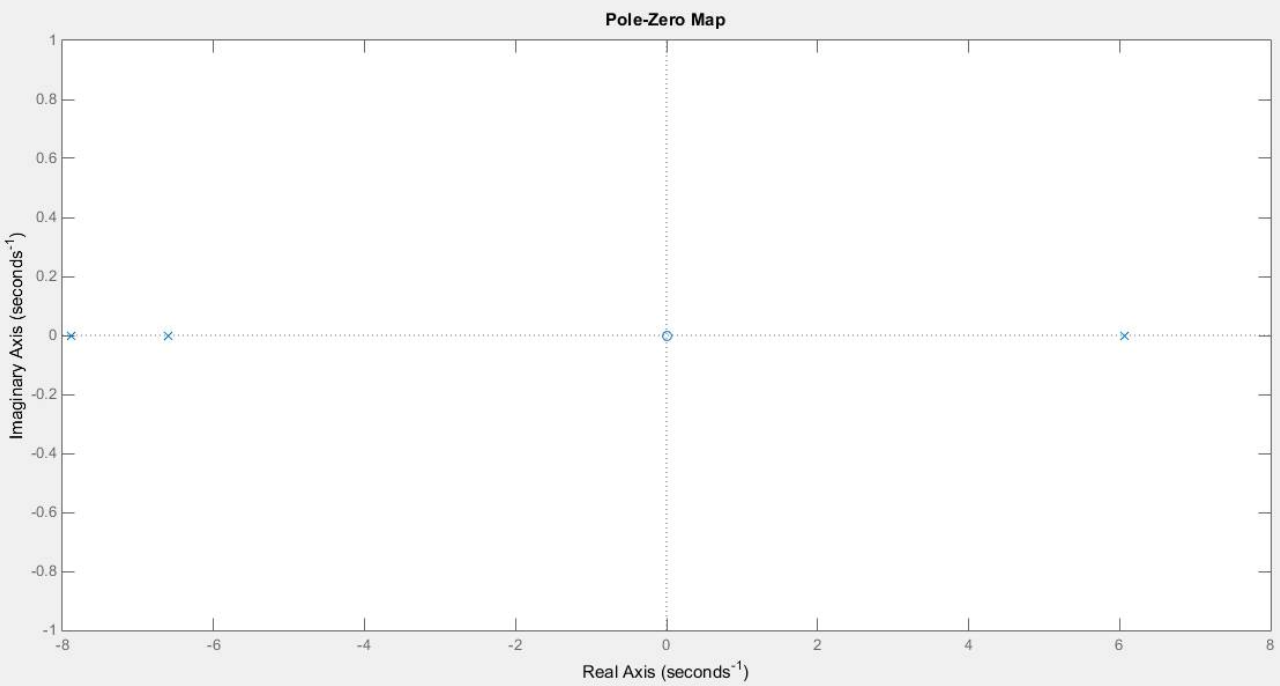


Figure 51. Pole Zero map.

This plot confirms the fact that the system is unstable, as well as the general shape of the step response plot. The zero is at (0,0), since it is a first order nominator with only one term. The unstable pole is (6.07, 0), and the other two poles are located in (-7.89, 0) and (-6.6, 0).

Also, to understand the frequency response, we have the Bode plot of the system, which expresses the change in amplitude and the change in phase of the system for the different frequencies.

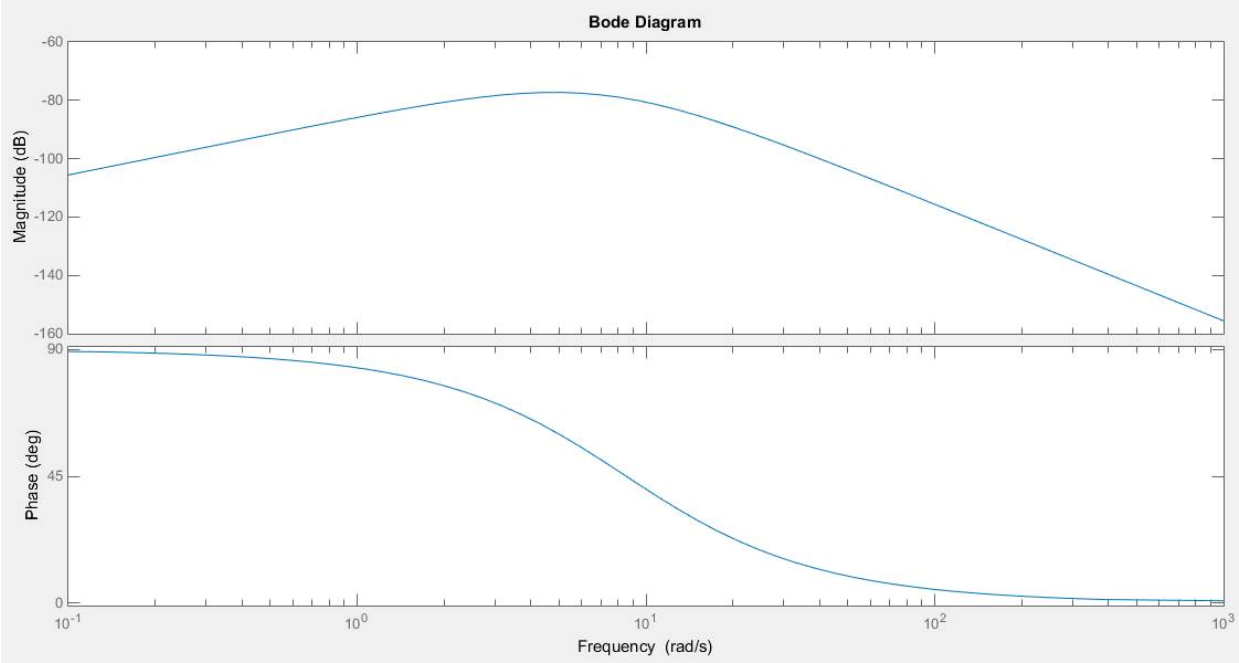


Figure 52. Bode plot for transfer function

Control

Arduino stabilization attempt

For comparison and backup purposes it was decided to make an Arduino program, which could be a simplified alternative to the Simulink stabilization model. For this case, only one sensor was used – accelerometer – to determine the position of the pendulum. The voltage supplying the wheel motor comes through L298N bridge driver, which is connected to Arduino. The expectable algorithm of the program is based on quadrants, obtained from the position. The schematic can be observed in the following picture.

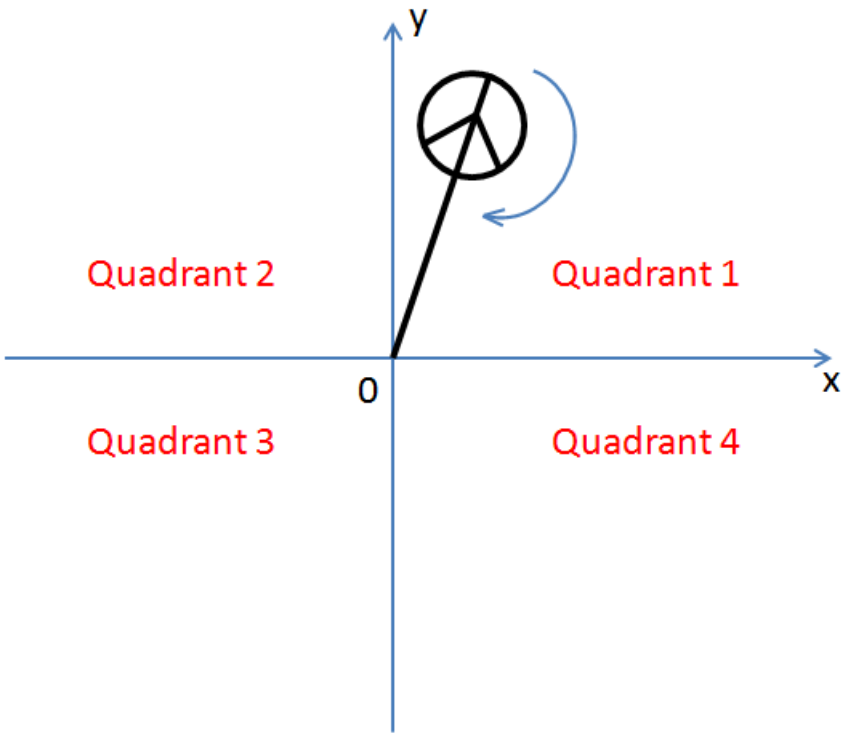


Figure 53. Quadrant placement.

For better comprehension, let us put the conditions for quadrants into the simple table.

Quadrant number	Conditions	
1	x>=0	y>=0
2	x<0	y>=0
3	x<0	y<0
4	x>=0	y<0

Figure 54. Quadrant borders.

It should be taken into consideration, that in order not to be stuck between two quadrants (when, for example, x is exactly 0), one of the quadrants should include this zero – either (x>=0 and x<0) or (x>0 and x<=0). The position of the pendulum - current x and y - is obtained by running these commands in the code:

```
int xPin = 2;
int yPin = 3;
int posX, posY;
pulseX = pulseIn(xPin,HIGH);
pulseY = pulseIn(yPin,HIGH);
posX = ((pulseX / 10) - 500) * 8;
posY = ((pulseY / 10) - 500) * 8;
```

The data acquisition is based on **pulseIn()** function. According to the official Arduino website, **pulseIn(xPin,HIGH)** waits for pin number 2 to go high, then start timing until the pin goes **LOW,** which stops the timing. Eventually, the length of the pulse is returned in microseconds [32]. The duration of pulses corresponds to the acceleration by a particular axis. While **pulseX** variable represents the raw data (large positive number of, for example, 5000), **posX** after conversion (formula provided on Arduino official website) is in milli-g's, can be both negative and positive, so corresponds to the quadrant principle used [33].

Now it is time to discuss the algorithm. The early version of program used to change the spinning direction of the wheel once the pendulum passes the border between Quadrant 1 and Quadrant 2.

Here is the piece of code for spinning the pendulum to the left (when **posX** is greater or equal to zero, so pendulum is in Quadrant 1).

```
int oneen = 10;
int onedir = A5;
int onedil = A4;

void spinL(int str){
  digitalWrite(onedil,HIGH);
  digitalWrite(onedir,LOW);
  analogWrite(oneen,str);
}
```

Variables **onedir** and **onedil** represent pin numbers for directions of spinning. **analogWrite(oneen,str)** function writes a number between 0 and 255 on pin 10 stored in variable **oneen**, which represents the duty cycle of PWM given from the power supply. In the early version of the program, the wheel was rotated with 100% duty cycle.

As it was discovered during the experiments, it is too much, because if we are, for instance, in Quadrant 2 before the very border of Quadrant 1, giving 255 as parameter, then the wheel is revolved so strong that it ends up with moving over 3-4 quadrant border, from which it is not possible to raise the wheel vertically back. This way, it was offered to change the duty cycle dynamically – the variable **str** was set according to the modulus of difference between the current pendulum position and the perfect vertical position. So, the farther the pendulum is from **posX=0**, the greater duty cycle is applied. As well as the previous solution, it made the pendulum stabilized vertically for some 2 seconds.

It goes without saying, that proper stabilization was still required. This was the part, in which the comparison with Simulink model could be done. In proper control theory, it is possible to use PID controller, but if-condition based implementation can be considered as forcing the prototype to stay vertical.

However, the idea was improved moving to rotating the wheel for short period of time not to overthrow it to the area from which the pendulum cannot be stabilized. For higher processing speed, the unnecessary delays were removed from the code. The transmissions between the 2nd and 3rd quadrants, 1st and 4th quadrants and, of course, 3rd and 4th quadrants are not taken into consideration, because in these areas it is too late to try to stabilize the pendulum – there is not enough energy to raise it up. Moreover, there was no need to care about the position of the pendulum on Y axis. Consider the picture below.

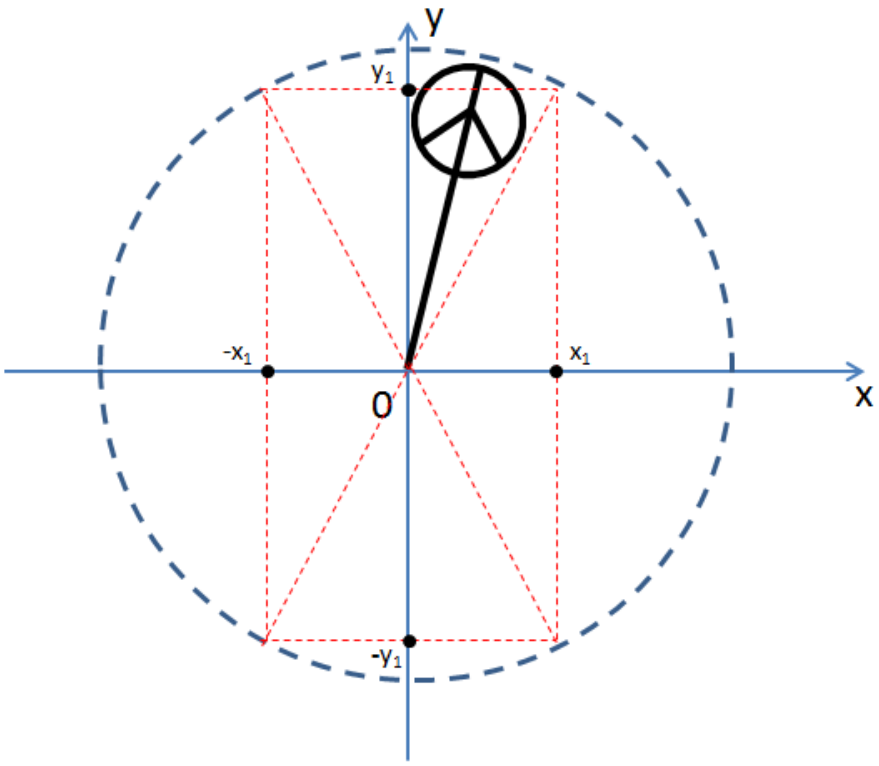


Figure 55. Coordinate accordance.

While current X position of the pendulum is between conditional $-x_1$ and x_1, which form so called range - the angle, which the pendulum is being stabilized within. According to the scheme, it is possible to observe that the pendulum can be at $-x_1$ or x_1 either at y_1 or at $-y_1$. Being at $-y_1$ for the pendulum brings no opportunity to stabilize, since it is in the 3rd quadrant or the 4th quadrant. That is why the Y coordinate reading could be neglected.

Eventually, the algorithm came up with being simple. Here it is.

```
if(posX>=0){
  spinL(255);
  delay(20);
  stopp();
 }
```

Basically, once the border is crossed the wheel is given full power for the time stated in **delay()** and stopped after (by writing digitally **LOW** to both direction pins) not to be overthrown. It should be taken into account, that by that time the prototype was missing the wheel, so testing was postponed, that is why **delay()** parameter value is taken just at random and of course will be maintained and changed. The full code is available in the application section.

Finally, Arduino approach connections are available in the scheme.

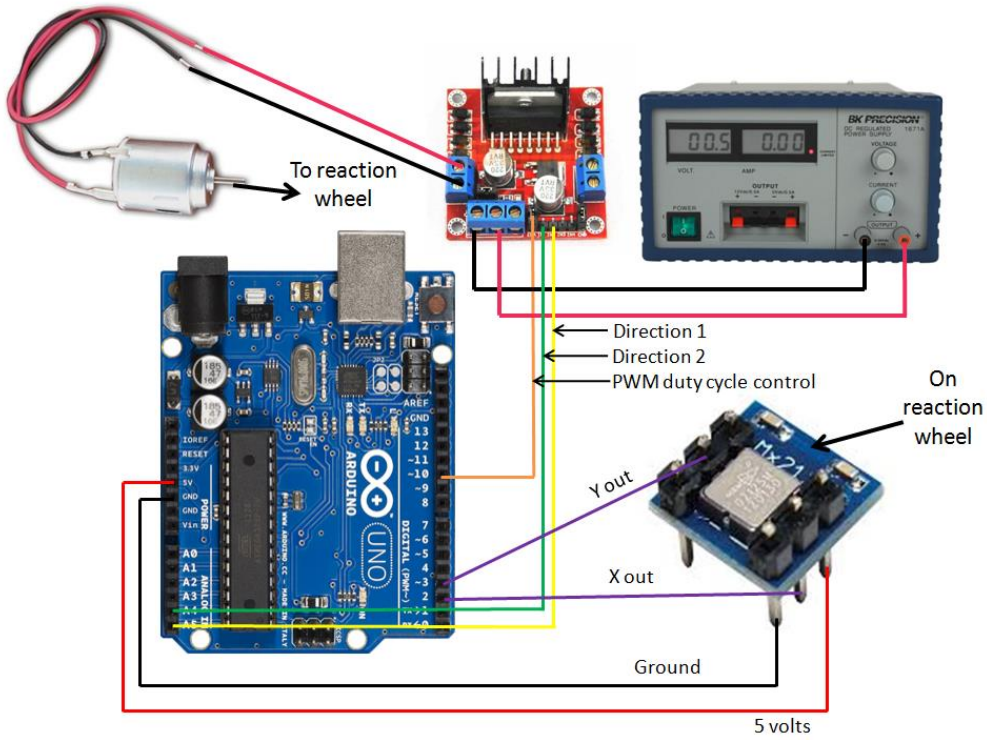


Figure 56. Arduino connections.

Multithreaded version of Arduino program [34]

This aspect of inverted pendulum software development can be considered as a further step. The project work of fourth semester is a deeper insight into control theory, so it could be implemented via mathematical model and corresponding programs only. However, in our group there was given an attempt to engage some computer science concepts. That is why an algorithmic solution has been worked on. As it is possible to understand from the header of the section, a multithreading version of program exists. It should definitely be mentioned, that program itself does not require separation into multiple threads and could be run without it. Multitasking was added mainly for learning purposes – to apply knowledge obtained during Embedded Software Design lectures. Here is the code, which was tested only with the output written to the console, because by the time of writing it the actual prototype wheel was missing, so there was no possibility to test it in practice. The code uses the library <krnl.h> provided by the teacher.

```
#include <krnl.h>
#define STACKSIZE 100
char stack[STACKSIZE], stack1[STACKSIZE]; // , stack2[STACKSIZE];
struct k_t* serialSem;
int oneen = 10;  // enable MOTOR 1
int onedir = A5;
int onedil = A4;
int xPin = 2;     // X output of the accelerometer
int pulseX;
int posX;
void task1(){
 while(1){
  k_wait(serialSem, 0);  //stop kernel forever(0)

          Serial.println("sens");
          pulseX = pulseIn(xPin,HIGH);
          posX = ((pulseX / 10) - 500) * 8;

  k_signal(serialSem);  //re enable it (semaphore)
  k_sleep(1);
 }
}

void task2(){
```

```
 while(1){
  k_wait(serialSem, 0);  //stop kernel forever(0)

          Serial.println("quadrant");

          if(posX>=0){
           spinL(255);
           delay(20);
           stopp();
          }

          else if(posX<0){
           spinR(255);
           delay(20);
           stopp();
          }

  k_signal(serialSem); //re enable it (semaphore)
  k_sleep(1);
 }
}

void spinR(int str){
  digitalWrite(onedir, HIGH);
  digitalWrite(onedil,LOW);
  analogWrite(oneen, str);
}
void spinL(int str){
  digitalWrite(onedil,HIGH);
  digitalWrite(onedir,LOW);
  analogWrite(oneen,str);
}
void stopp(){
  digitalWrite(onedil,LOW);
  digitalWrite(onedir,LOW);
  //analogWrite(oneen,0);
}


void setup() {
  k_init(2, 1, 0);
  k_crt_task(task1, 10, stack, STACKSIZE);
  k_crt_task(task2, 10, stack1, STACKSIZE);
```

```
  pinMode(onedir, OUTPUT);
  pinMode(oneen, OUTPUT);
  pinMode(onedil, OUTPUT);
  pinMode(xPin, INPUT);
  Serial.begin(9600);
  serialSem = k_crt_sem(1,10);
  k_start(1); // switch every millisecond
  Serial.println("bad stuff");
}
void loop() {
}
```

Now the description comes. There are two separate task created with the same high (10) priority – **task1** for sensor measurement and **task2** for controlling the wheel depending on the data received from the accelerometer. Tasks share the same resource – variable **posX** (current position of the pendulum at X axis). **Task1** sets its value and **task2** in its turn spins the wheel in the direction according to that value. The flow of threads is controlled by one semaphore. The command **k_init(2, 1, 0)** tells the kernel that there will be 2 tasks, 1 semaphore and 0 messages. This solution is used to prevent happening of deadlock (when two concurrent processes does not allow each other to use the shared resource) or race conditions (when events are happening in undesired order) – the issues known in concurrent computing.

Nevertheless, in this particular case we would not really meet or be afraid of this thing to happen. Even if in non-multithreaded program accelerometer measurement somehow did not happen before controlling the wheel, it would be moved according to the initial value of the variable and the new iteration of main loop wood run, which would likely not have an error and bring actual values of measurements, so the system would not really be stuck or frozen, because baud rate is 9600 bits per second stated in **Serial.begin(9600).**

The actions of both tasks are performed as critical sections for other tasks not to access the shared resources while current task is being performed. First comes **k_wait(serialSem, 0)**, which, according to the original comment, stops the kernel forever but with a possibility to enable it again. Now we let the critical section to run safely – the content of task aligned in the full code above. Then, as it was supposed, the stuff is enabled by calling **k_signal(serialSem)** method and with **k_sleep(1)** we

inform the program that current task execution is finished and the next task can take the flow. A tiny disadvantage though is certain tame taken to switch between tasks. In the simple diagram below the multithreaded program principle is observable.

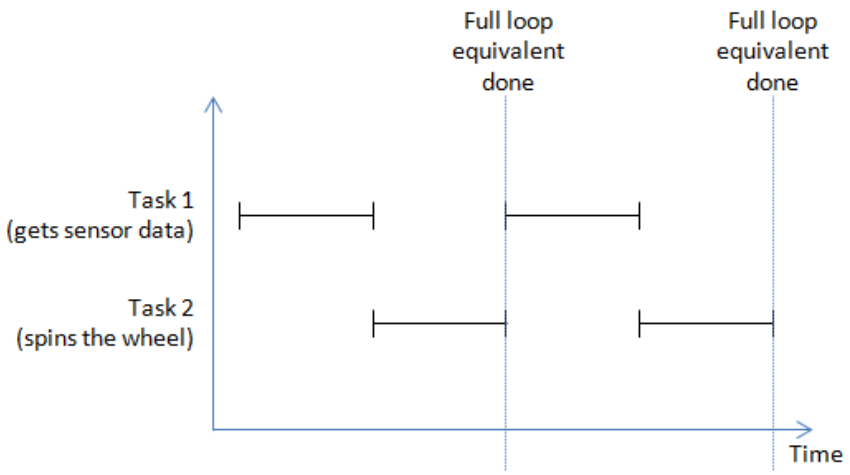


Figure 57. Multitasking diagram.

NI PCI-6229 and Matlab [35]

There are several ways to create communication between actual PCI board and Matlab software. One of these is to develop Simulink model and include in the system particular element as the connection between the digital part of the system and real life. Another way is to use “daq session” in Matlab workspace. The last one was widely used to test PCI pins, Matlab functionality and sensor operability. The following piece of code can be inspected:

```
s = daq.createSession('ni')
addAnalogInputChannel(s,'Dev1','ai9','Voltage')
s.DurationInSeconds = 2.0
s.Rate = 5000
data = startForeground(s);
plot(data)
```

The first line stands for creating an object of a session, which will be used for working with a particular device. In our case it is NI PCI-6229, that is why **'ni'** is given as the parameter. Afterwards analog input channel is added to the session in variable **s**, device ID **'Dev1'**, channel **ai9** (analog input 9, pin marked as 66, according to the datasheet) and **'Voltage'** measurement type. Device ID was obtained by calling **daq.getDevices** method. The next command is **s.DurationInSeconds = 2.0,** which represents the duration of acquisition – for how long the data is being obtained. After this we set the rate, the amount of scans per second. In this way, running the code will give us 5000 scans · 2.0 seconds = 10 000 scans. Matlab provides us with the following information.

```
s =

Data acquisition session using National Instruments hardware:
   Will run for 2 seconds (10000 scans) at 5000 scans/second.
   Number of channels: 1
      index Type Device Channel   MeasurementType        Range       Name
      ----- ---- ------ ------- ------------------- ---------------- ----
      1     ai   Dev1   ai9     Voltage (SingleEnd) -10 to +10 Volts
```

Figure 58. DAQ session info.

data = startForeground(s) instruction runs the operation in session **s** given as a parameter and writes the result of acquisition to the variable **data**. The last command just plots the time-voltage graph of acquired data.

Modelling the system with the means of Simulink

In order to model our system in the Simulink environment, we developed the transfer functions from the equations found in the mathematical modelling section. The total number of 2 transfer functions was developed – one for converting the voltage applied to the motor to the angular velocity of the wheel, and the other for converting the angular velocity of the wheel to the angular position of the pendulum. Later, those two transfer functions were merged into one transfer function for convenience and better tuning.

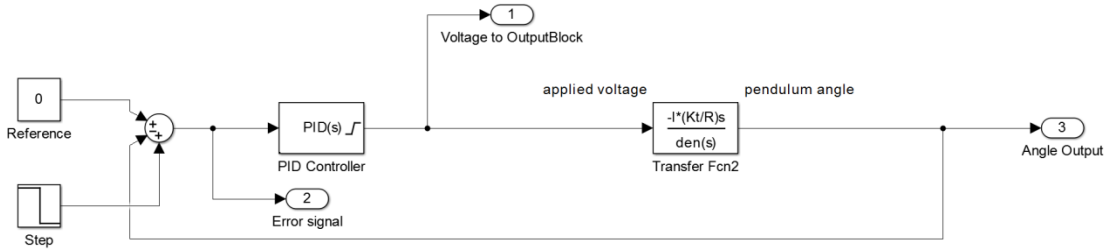


Figure 59. Simulink model of the system.

The reference point for our system is a zero, which is simply a constant, representing the angle, which is the pendulum's stabilization point (when the pendulum is in the upright position).

We also make a step input, in order for the system to never be completely stable – it acts as a disturbance source for the system.

The PID controller is used for controlling the system. We have tuned the PID using the Simulink "Tune" function, to adjust the error in the system to fit the requirements in response time and transient behavior.

The PID block also incorporates a saturation block inside of it, which is used to set the boundaries for the voltage: -24 to 24 volts (since our motor's maximum voltage level is 24).

After the PID transforms the signal, it is input in our plant, which is represented by a transfer function, derived in the transfer function derivation section. The output of the plant is the angle of the pendulum, returned as a negative feedback to the point before the PID. That way, the system is correcting the error in the signal, thereby stabilizing the pendulum angle.

Encoder

The values of the sensor are read using Simulink. The values given by the terminals are used to determine the position of the pendulum (θ). First, they must be converted into radians to be used further by applying a gain of $(2 \cdot \pi/(500 \cdot 4))$. There are 500 cycles per revolution and, since it is a quadrature encoder, the values are divided by 4 and finally multiplied by 2π to get the value in radians. The second gain was placed to display the values in degrees.

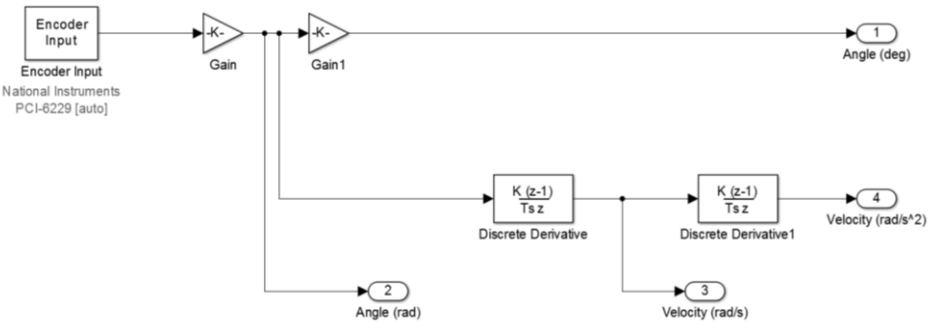


Figure 60. Simulink model for the encoder.

With the position given, the only thing remaining is obtaining the real time values of the angular velocity ($\dot{\theta}$) and acceleration of the pendulum ($\ddot{\theta}$). For these two we apply each time a discrete derivation, to get an accurate derivation without filtering. As shown in the next figure, the signals are not continuous, due to the position only being read with a defined frequency.

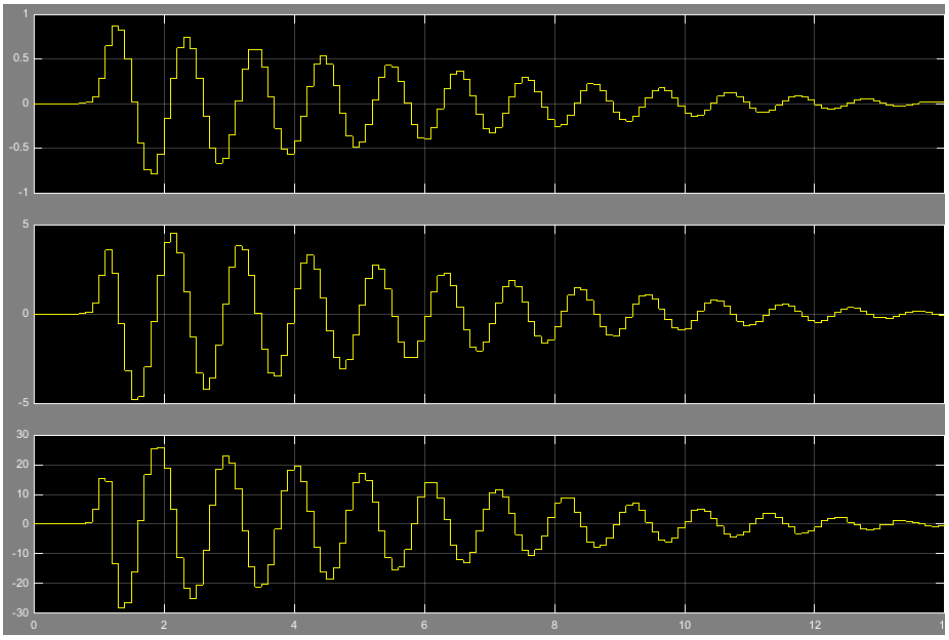


Figure 61. Signal output of θ, $\dot{\theta}$, $\ddot{\theta}$ through time.

The figure above shows the signals printed by Simulink, where the first graph represents the position, the second angular velocity (speed), and the third, respectively, the angular acceleration.

Model Output Block

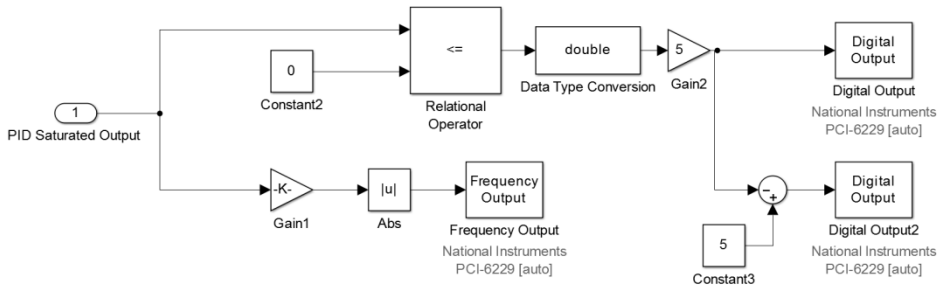


Figure 62. Transformation from ±24 V to PCI output.

After the PID and the saturation, the output must go into both into the real life prototype, as well as the transfer function. This section will discuss how this output is transformed into a PWM signal and a direction combination. We use the Frequency Output block in the Real-Time Windows Target section, in which we get an external duty cycle. This value must be a number between zero and one, representing the percentage duty cycle. In order to transform a ±24 signal into its proportional duty cycle we divide it by 24 by passing through a gain of 1/24, and then we use the Absolute value block, since the duty cycle is always positive.

The direction is determined by comparing the value to zero. Depending if it is negative or positive, it will result in a Boolean, which we convert into a double, which is either one or zero. Then we go through gains and an adder to ensure that if one pin is high, the other one will be low.

Also, for all the blocks we define the final value to be zero, so that the wheel stops when the simulation is over.

Swing up model

In order to put the model to the test, we designed a swing up Simulink model. This is done by getting the velocity reading and sending the signal into the following block:

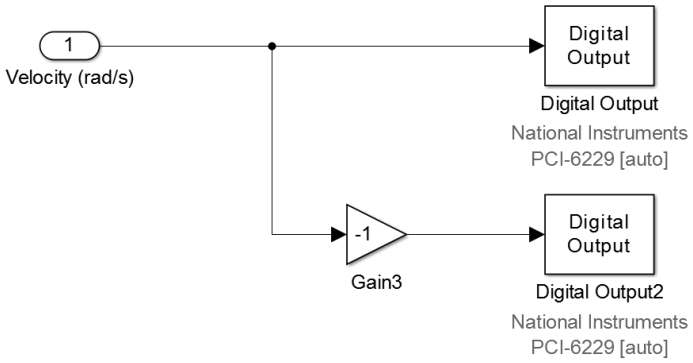


Figure 63. Digital control of output for Swing up.

The negative gain block ensures that the other pin will be the opposite of the first one, to allow the wheel to turn. This model block ensures that the pendulum accelerates from the stable position into swinging 360°. All the model needs to start working is for the velocity to be non-zero, since if it is 0 then both pins will be off, and thus the wheel and the pendulum will stay static.

The resulting curve is displayed in the following figure, for position, velocity and acceleration.

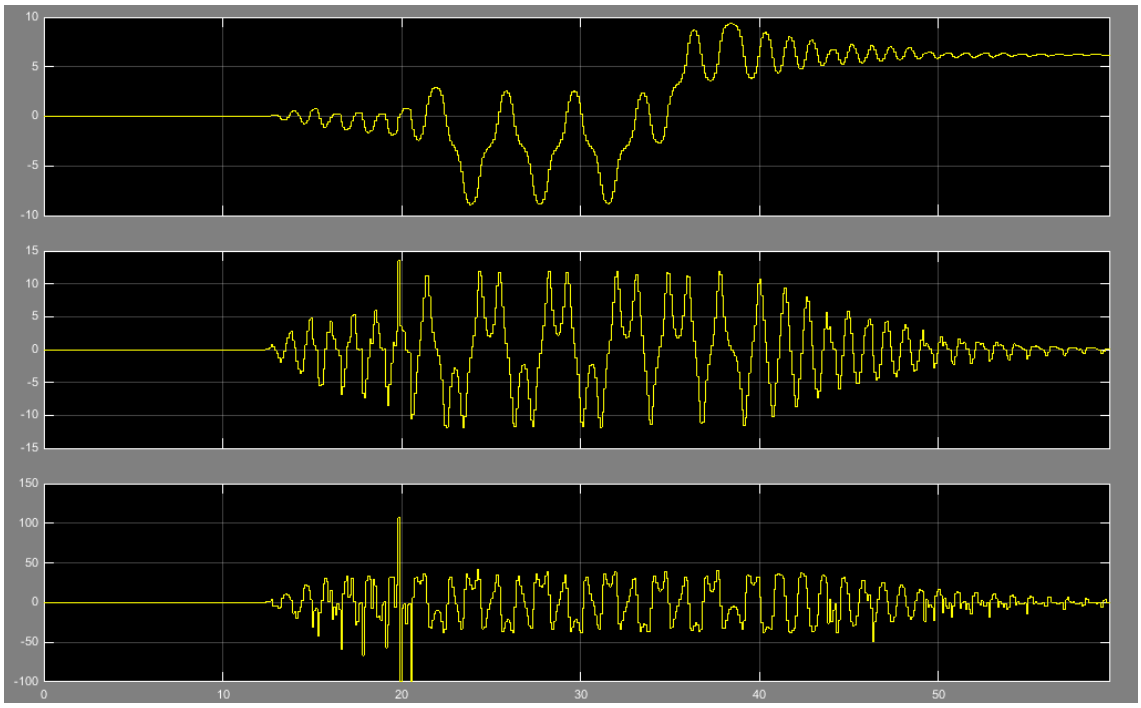


Figure 64. Swing up program results

PID [39]

A PID controller (proportional-integral-derivative controller) aims at calculating and minimizing an error value between a measured value and a set reference point.

The PID controller consists of three constant values: P – proportional, I – integral and D – derivative.

P is a gain, which reduces the rise time. As the gain increases, response time becomes quicker, however, at a cost of larger oscillations. P does not completely eliminate a steady-state error, although it makes it smaller when it is increased. Aside from that, the effect of disturbance remains in the response, causing extra steady-state error.

I is an integral component of the PID. It eliminates the steady-state error, as well as gets rid of the disturbance of the steady-state. However, it is not enough to get a good transient response of the system.

D is a derivative component, which improves the system's transient response. It also increases the damping, which makes the system more stable, and reduces the overshoot. A filter value (*N* in Simulink) is also often used, in order to limit the differentiator value, as disturbances may cause it to be theoretically infinitely high.

Using all three of the components, the PID controller is a good tool for minimizing a system error.

The output of the PID controller can be calculated using the following formula:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\, d\tau + K_d \frac{d}{dt} e(t) \tag{78}$$

where:

K_p, K_i, K_d are respectively the proportional, integral and derivative gains, that can be tuned.

e is error of the system.

t is the present time.

τ is an integration variable, changing value from 0 to the t.

In order to get the PID values for our Simulink model, we used the Simulink "Tune" function. It allowed us to tune the system's response time to be fast enough, at the same time making sure that the motor can follow the model. It also helped us to tune the transient response of the system, which is defined as the response of a system to a change from balance, making sure that the pendulum can be balanced without getting out of stabilization bounds due to being too aggressive or robust.

The values that we have got using the "Tune" function are the following:

$$K_p = -21853.3087451982 \quad (79)$$

$$K_i = -88294.8908046664 \quad (80)$$

$$K_d = -1021.6777828835 \quad (81)$$

$$N = 209.794311620465 \quad (82)$$

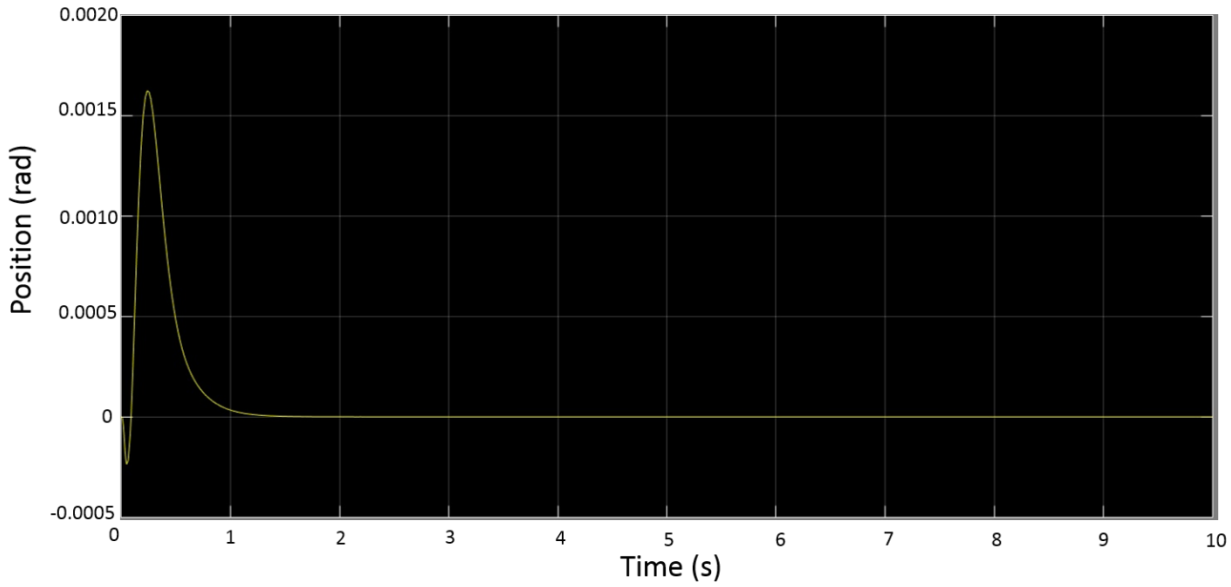


Figure 65. Angular position of the pendulum with respect to time using the designed PID.

As can be seen in the above figure, at first a deviation occurs due to the step input, acting as a disturbance source. Right after that, the PID causes the oscillation, and the output in the form of the angular position is stabilized at 0. It can be understood from the graph, that the transient

response of the system is quite robust, since the stabilization occurs fast and without many oscillations.

Testing

Transfer function Model

The model was tested with the values described in the previous section. We were able to stabilize the pendulum up to 8 seconds; however, it was not always successful in doing so. Additionally even when the control system was correctly initialized and stabilized by itself, it would eventually lose control and fall to the rest position.

In order to achieve these results we used a 100 times faster sample time in the PCI blocks, and an equal amount of tolerance towards missing ticks. This provided what seemed a faster response in the plant.

We also attempted a large amount of other values for the PID, attempting to increase the speed and try many values around the transient response; however, this did not give any better results.

Further work on PID tuning was planned, but was not possible because of a prototype failure caused by a broken screw in the connection between the motor and the wheel, preventing the fast changes in direction that allow the stabilization.

State-Space

As mentioned in the previous testing section, yet another prototype failure prevented testing of the state-space formula. Therefore, instead we did an analysis of the effect the transient response had on the output of the system. We decided to stick to a response time of 0.01, to be similar to the timing in the PCI ports. The first values used where

$$K_p = -16834.0283149906 \quad (83)$$

$$K_i = -27955.0337217976 \quad (84)$$

$$K_d = -491.453259388006 \quad (85)$$

$$N = 22856.5595208321 \quad (86)$$

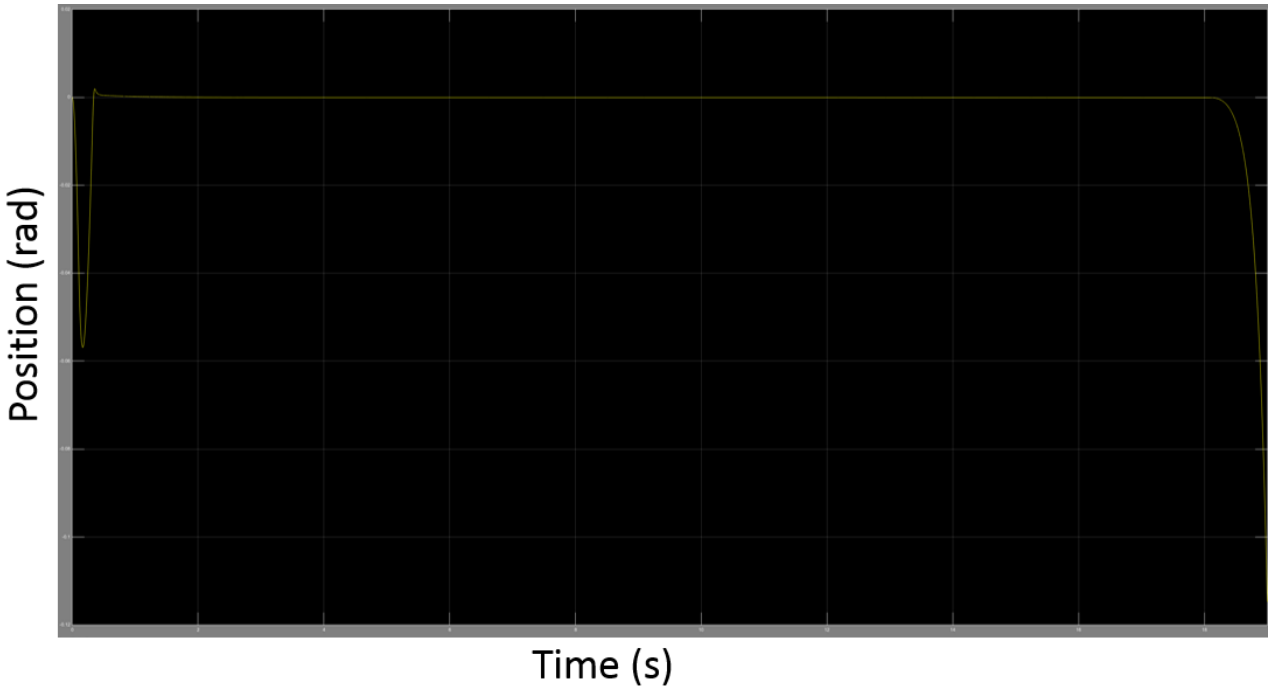


Figure 66. Robust transient PID result.

This kind of PID values give a fast non-oscillating response, and lose stability at around 18 seconds.

Next, we tested the following values.

$$K_p = -87085.7161310396 \quad (87)$$

$$K_i = -634318.9589303 \quad (88)$$

$$K_d = -124.566619549887 \quad (89)$$

$$N = 284.45080404566 \quad (90)$$

This is a much more aggressive transient behavior. The output recorded is shown in the next figure.

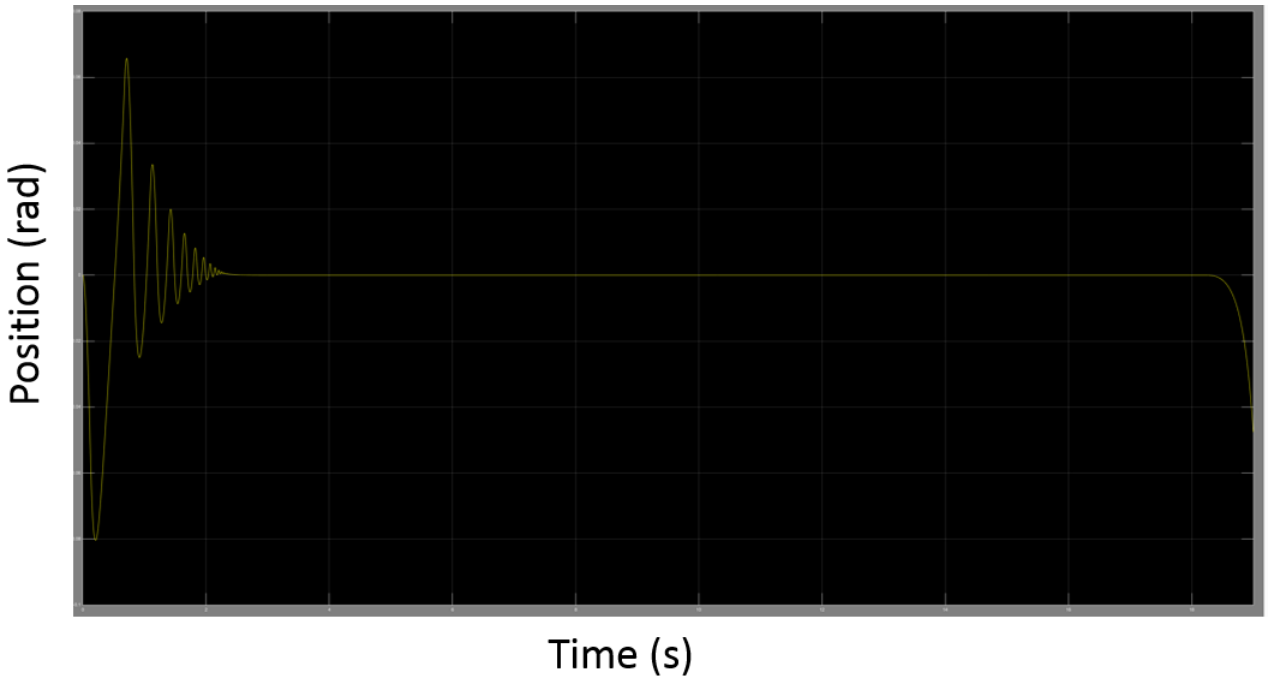


Figure 67. Aggressive transient PID result.

This system response has an oscillating result, which takes longer to stabilize back to zero. In addition, like the previous PID values, it loses stability after 18 seconds.

From a purely theoretical point of view, the robust option is clearly favourable. However, in the plant, we expect that errors in the model values and the accuracy of inputs/outputs will make a completely robust transient response not effective in practice, so we probably must test values between both options to find the ideal solution.

Arduino implementation

Once the new parts arrived, fixing and assembling of the prototype were done. It happened on May 26 and 27 – the two days before the hand-in day. Testing was then possible. Arduino program (the single thread version) used to be tested with different combinations of PWM duty cycle and delay – time, which the impulse (spinning the wheel) was given for. The best (among others) results were obtained with a highest duty cycle (**oneen = 255**) given for a short period of time (around 50

milliseconds). The pendulum got stabilized for a couple of seconds, which could, however, happen thanks to high friction mostly.

Nevertheless, during the process of testing it was discovered that the prototype did not act as supposed. The part connecting the wheel and the engine was broken in the beginning of testing, so the wheel was not connected completely and did not repeat the revolutions of the engine. Because of this problem, we do not really know, what was the reason of Arduino stabilization attempt failure – inadequate software, broken hardware or both. It goes without saying, that multithreading program was not tried out due to the detected issue.

Evaluation of solutions

Matlab solution

Testing was partly successful, but limited due to external factors, such as the prototype breaking apart in most parts, including but not limited to: the wheel shaft, gearbox, encoder, accelerometer, connection between motor and wheel, metal piece fastening wheel.

The transfer function worked as expected with the model, but much like the behavior described in the state-space testing, it reached a point where it was not able to keep the pendulum stable anymore. However, stabilization during the period in which the Simulink model responded in the desired manner was successful.

As for the state-space, the theoretical results seem promising; however, this is not enough to conclude on the accuracy of the model without practical testing and results.

Additionally, if an improved model is desired, we note that the inertia of the pendulum must be tuned. In addition, the total system model should also be validated to real life, to check the accuracy of the final system.

Finally, adding a minimum value for the voltage would also help, as the low voltages do not move the wheel, although they do have a numerical impact on the transfer function.

Arduino solution

The failure brought not only disadvantages. Overthrowing the wheel and aggressive behavior used to happen in any Arduino program version. The algorithm was not based on formulae and precise data, but more on subjective logics. Supplying the algorithm with formulae actually leads to a mathematical model. This occasion let us challenge the mathematical model solution and emphasize the actuality and application of control theory.

Conclusion

In conclusion, the project of stabilizing a pendulum is a problem that is harder to solve than expected. Attempts to stabilize the system without the use of a model resulted in poor results, even when implementing complex algorithms and programs to adjust the output voltage as well as have a fast cycle.

The model was theoretically defined, but had to be precisely tuned to the real life prototype. In addition, we encountered a large variety of solutions to the modelling, and attempted to compare their performance. In addition, we noticed that the addition of a PID block added even more possibility of tuning.

The goal of the project was to stabilize the inverted pendulum, and optionally do a swing up controller. The first objective has been achieved for a duration of 8 seconds. Although this is not as long as the performance shown by the “Swing-up and Stabilization of an Inverted Pendulum using a Reaction Wheel” report, it is a simple proof of concept that very unstable systems can be stabilized, and that there are some issues to be solved in the future. The optional objective of swinging up the pendulum was also achieved, although the combination of the two solutions was not tested.

The prototype appeared to be quite difficult to work on, since multiple parts failed and consecutively had to be replaced during the course of the project. It was a challenging task for us, since the final testing had to be done one day before the hand-in; however, partial success was achieved despite all the trouble.

All in all, we learned a lot during the project, and were able to achieve a partial success in stabilizing the pendulum.

References

1. Segway Challenge. *http://people.kth.se/~crro/segway_challenge/model.html (accessed 11 May 2015).*
2. KHALIL SULTAN. *Inverted Pendulum. : ; no date. http://www.engr.usask.ca/classes/EE/480/Inverted%20Pendulum.pdf (accessed 11 May 2015).*
3. The mechanics of standing balance*. https://www.khanacademy.org/test-prep/mcat/physical-sciences-practice/physical-sciences-practice-tut/e/the-mechanics-of-standing-balance (accessed 11 May 2015).*
4. Frank Jepsen, Anders Roland Pedersen, Anders Søborg. *Swing-up and Stabilization of an Inverted Pendulum using a Reaction Wheel. 2008.*
5. Properties of aluminium. *http://www.aluminiumdesign.net/why-aluminium/properties-of-aluminium/ (accessed 11 May 2015).*
6. Future Electronics. Optocouplers. *https://www.futureelectronics.com/en/optoelectronics/optocouplers.aspx (accessed 17 March 2015).*
7. Wayne Storr. *Optocoupler Tutorial. http://www.electronics-tutorials.ws/blog/optocoupler.html (accessed 15 March 2015).*
8. Toshiba. *TLP250 Datasheet.*
9. H-bridge. *http://www.modularcircuits.com/blog/articles/h-bridge-secrets/h-bridges-the-basics/ (accessed 11 May 2015).*
10. DROK® L298N Motor Drive Controller Board DC Dual H-Bridge. *http://www.amazon.com/DROK-Controller-H-Bridge-Mega2560-Duemilanove/dp/B00CAG6GX2/ref=sr_1_2?ie=UTF8&qid=1432468380&sr=8-2&keywords=L298N (accessed 11 May 2015).*
11. DUAL FULL-BRIDGE DRIVER. *http://www.st.com/web/en/resource/technical/document/datasheet/CD00000240.pdf (accessed 11 May 2015).*
12. PCI. *http://www.computerhope.com/jargon/p/pci.htm (accessed 11 May 2015).*
13. National Instruments. *NI CB-68LP. http://sine.ni.com/nips/cds/view/p/lang/da/nid/1187# (accessed 26 May 2015).*
14. *PCI. : ; no date. http://www.ni.com/pdf/manuals/371290g.pdf (accessed 11 May 2015).*
15. DAQ M Series*. : ; 2008. http://www.ni.com/pdf/manuals/371022k.pdf (accessed 11 May 2015).*
16. Encoder. *http://musicarena.exblog.jp/tags/Mark%20Levinson%20No.38L/ (accessed 11 May 2015).*
17. Hewlet Packard. *HEDS-5700 series Datasheet.*
18. Arduino. *http://www.arduino.cc/ (accessed 11 May 2015).*
19. Tony Olsson. *Arduino Wearables, 1st ed. USA: ; 2012.*

20. What is Arduino?. *http://www.treeelectronics.com/2014/09/arduino.html (accessed 11 May 2015).*
21. A beginner's guide to accelerometers. *http://www.dimensionengineering.com/info/accelerometers (accessed 11 May 2015).*
22. Accelerometers and How they Work How they Work. *: ; 2005. http://www2.usfirst.org/2005comp/Manuals/Acceler1.pdf (accessed 11 May 2015).*
23. Memsic 2125 Dual-axis Accelerometer. *http://learn.parallax.com/KickStart/28017 (accessed 11 May 2015).*
24. electrical4u. *Working or Operating Principle of DC Motor. http://www.electrical4u.com/working-or-operating-principle-of-dc-motor/ (accessed 1 December 2014).*
25. Gene F. Franklin, J. David Powell, Abbas Emami-Naeini. *Feedback Control of Dynamic Systems, 7 ed. Essex, England: Pearson Education Limited; 2015.*
26. Carl R. Nave. *HyperPhysics: Angular momentum of a particle. http://hyperphysics.phy-astr.gsu.edu/hbase/amom.html (accessed 2 May 2015).*
27. MSCI Canada. *What is a Reaction Wheel. http://www.reactionwheel.com/resources/reactionwheel.html (accessed 4 May 2015).*
28. Boundless.com. *The Physical Pendulum. https://www.boundless.com/physics/textbooks/boundless-physics-textbook/waves-and-vibrations-15/periodic-motion-123/the-physical-pendulum-432-1488/ (accessed 6 May 2015).*
29. Carl R. Nave. *Trigonometric Infinite Series. http://hyperphysics.phy-astr.gsu.edu/hbase/trgser.html (accessed 6 May 2015).*
30. Erik Cheever. *Transformation: Transfer Function -> State Space. http://lpsa.swarthmore.edu/Representations/SysRepTransformations/TF2SS.html (accessed 12 April 2015).*
31. Erik Cheever. *State Space Representations of Linear Physical Systems. http://lpsa.swarthmore.edu/Representations/SysRepSS.html (accessed 12 April 2015).*
32. Arduino. *Arduino - PulseIn. http://www.arduino.cc/en/Reference/PulseIn (accessed 19 May 2015).*
33. Arduino. *Arduino - Memsic2125. http://www.arduino.cc/en/Tutorial/Memsic2125?from=Tutorial.AccelerometerMemsic2125 (accessed 19 May 2015).*
34. Jens Frederik Dalsgaard Nielsen, Per Printz Madsen. *Embedded Software Design lecture notes. Aalborg University Esbjerg; Spring 2015.*
35. The MathWorks, Inc. *Create data acquisition session for specific vendor hardware. http://se.mathworks.com/help/daq/ref/daq.createsession.html (accessed 2 May 2015).*
36. Akbar Hussain. *Microprocessors and programming lecture notes. Aalborg University Esbjerg; Autumn 2014.*

37. Maxim Integrated. *Analog, linear and mixed-signal Devices. http://www.maximintegrated.com/en/images/appnotes/1080/1080Fig02.gif (accessed 6 April 2015).*
38. Ibrahim Kamal. *8-bit Digital to Analog converter (DAC). http://www.ikalogic.com/8-bit-digital-to-analog-converter-dac/ (accessed 2 April 2015).*
39. Zhenyu Yang. *Fundamental Control Theory and Modelling lecture notes. Aalborg University Esbjerg; Spring 2015.*

Appendix A

ADC and DAC

During the first stage of the project, we decided to have readings of both encoders as well as a pulse width modulation output. However, this was not possible, since the PCI 6229 has only two Counters available. This is why we decided to handle one of the sensors as a peripheral sensor with its own microcontroller. To communicate between the PCI and the Arduino that controlled the extra sensor, it was necessary to have some sort of communication. We considered three ways to send the resulting number to the PCI.

- Synchronous master slave SPI system
- PWM signal into PCI
- ADC program in peripheral, DAC in Matlab

Evaluating our options, we concluded that option 2 was not possible, since we could not measure the duty cycle without a counter pin. Option one involves establishing a common clock as well as checking protocols and setting up the right blocks, so we decided to use the last option, ADC and DAC.

ADC

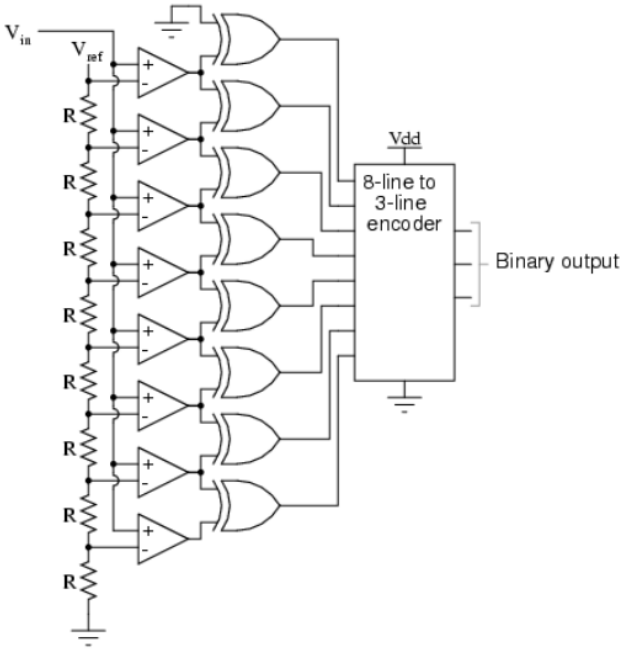


Figure 68. Flash ADC circuit [36].

For the ADC part, we considered building a Flash ADC, however opted against it, since it is highly dependent on the resistors being equal and we had the programming skills to do it using software, via the Arduino program. We were considering a Digital ramp ADC in the program, which is simple, though it has limited time performance because it needs to count from zero every time. This behavior is shown in the following graph.

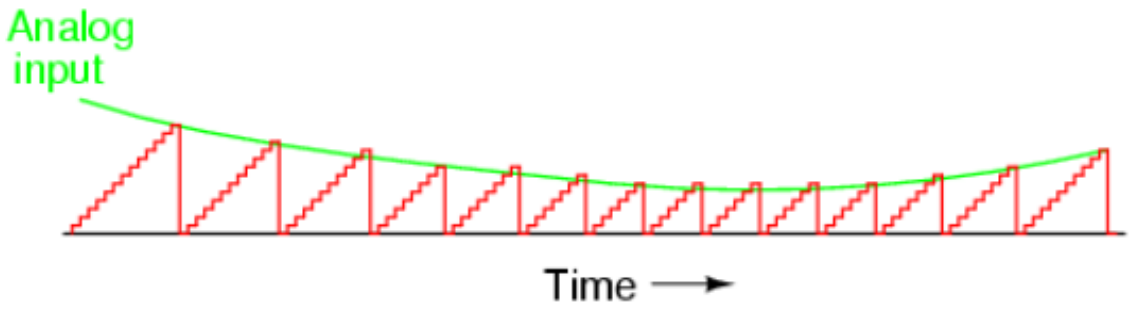


Figure 69. Digital Ramp Method [36].

An improvement would be to implement a tracking ADC by recording the previous value and comparing if it is bigger or smaller. However, this could also add some issues due to bit bobble, which would be a big issue for our system.

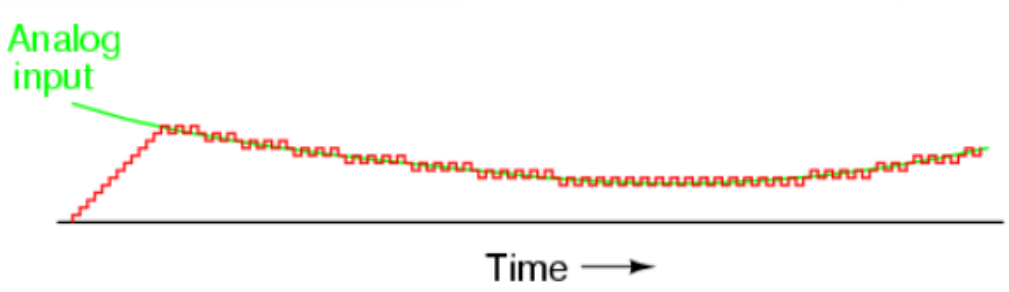


Figure 70. Tracking ADC showing bit bobble [36].

Finally, it is also possible to use an integrator, instead of a ramp; however, this is a more advanced form of ADC, which combines a software block with a hardware block, and its complexity would not bring much benefits to our system.

We, however, decided to use a Successive Approximation Register (SAR) ADC, since it is easily translated into code, and is used for most general-purpose applications. It works by comparing the value to a reference, which decreases or increases by ½ of the reference depending on the comparison result. This is displayed in the following image.

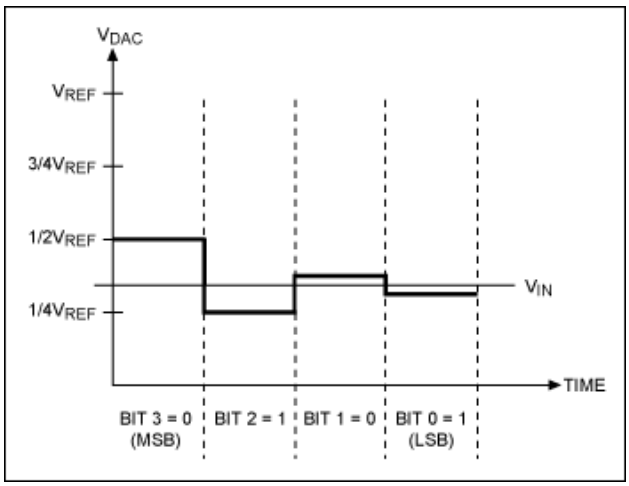


Figure 71. SAR ADC Steps [37].

The program starts by setting up the pin directions by accessing the port D and B registers, and making eight pins outputs.

```
DDRD = B11111100;
DDRB = B00000011;
```

Next in the loop, we read the analogue decimal value and send it through a loop for each bit like the following:

```
if (num>=128){
  PORTB = PORTB|B00000010;
  num = num-128;
}
else{
  PORTB = PORTB&B11111101;
}
```

The first step is to check if the bit should be on. If it should be on then we use the OR (|) operation to make sure that the bit is one. Else, we make it zero, using the AND (&) operation on the register. The same thing is repeated for all multiples of two changing the bit being toggled too.

DAC

For the DAC part, we had two options. The first one was to use a pin for each byte. This would involve a similar principle to the binary weighted resistor DAC, except using gains decreasing from the MSB to the LSB. The Simulink model would look like the following:

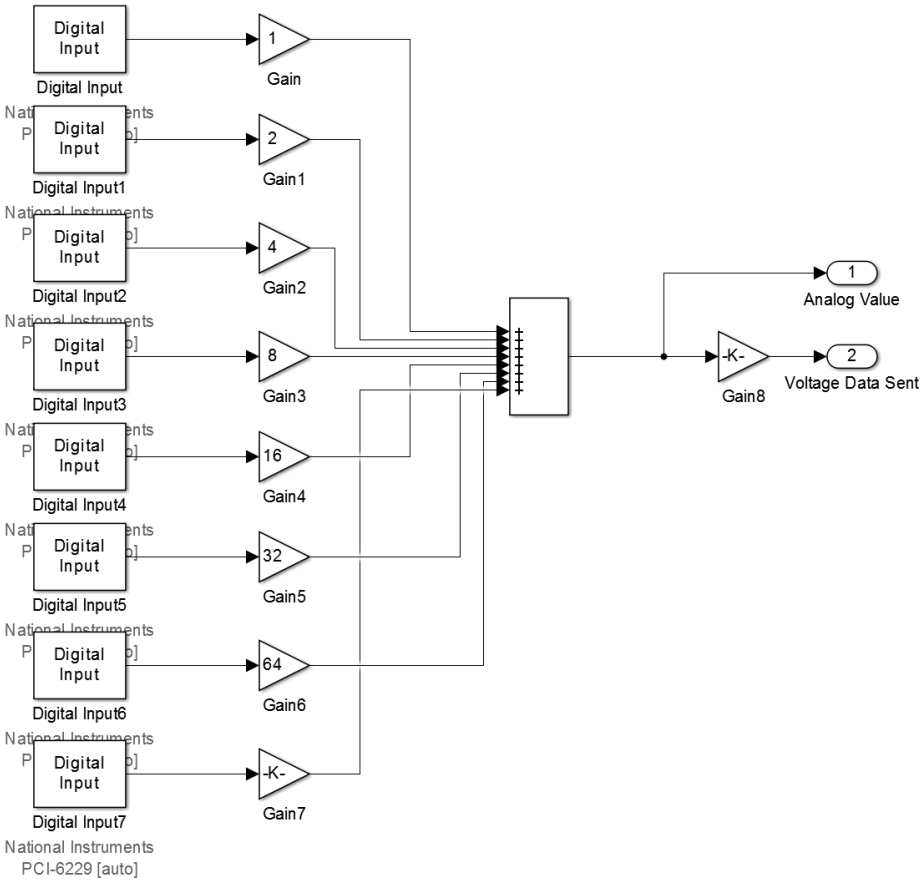


Figure 72. Simulink DAC with gains.

We also developed an electronic circuit to do the same task, in case we did not have enough pins available. We used the R/2R resistor network DAC, like shown in the following diagram.

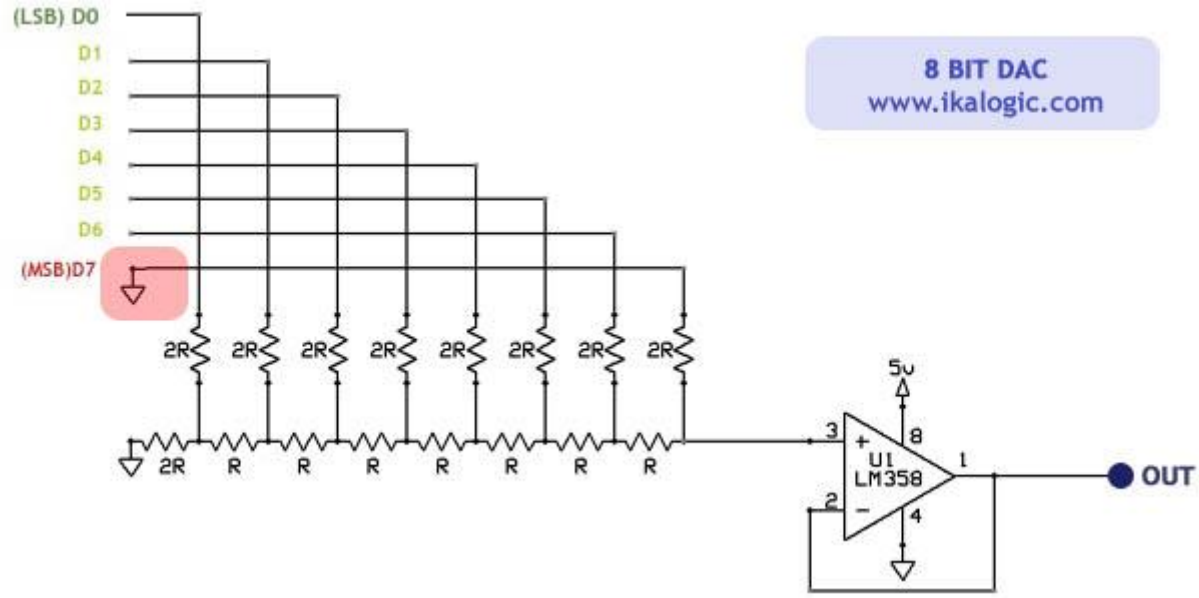


Figure 73. R/2R resistor network DAC [38].

In the end, we had enough digital pins to use the software solution, so we used it, since it does not have the issue of depending on the accuracy of the resistors, and was tested to have an error that was small enough to be irrelevant.

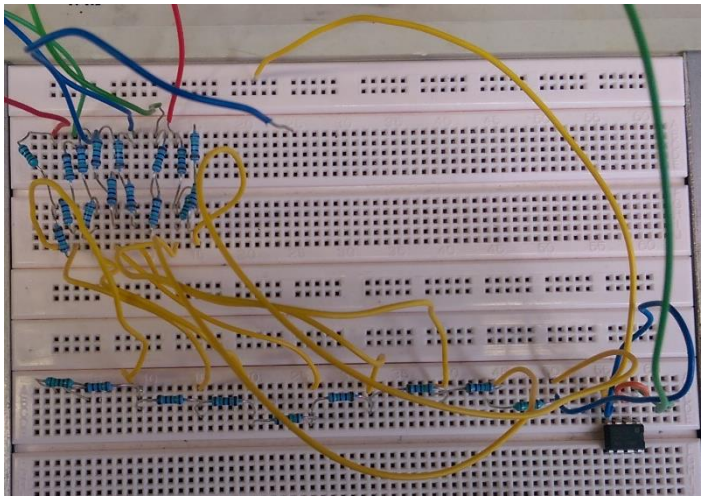


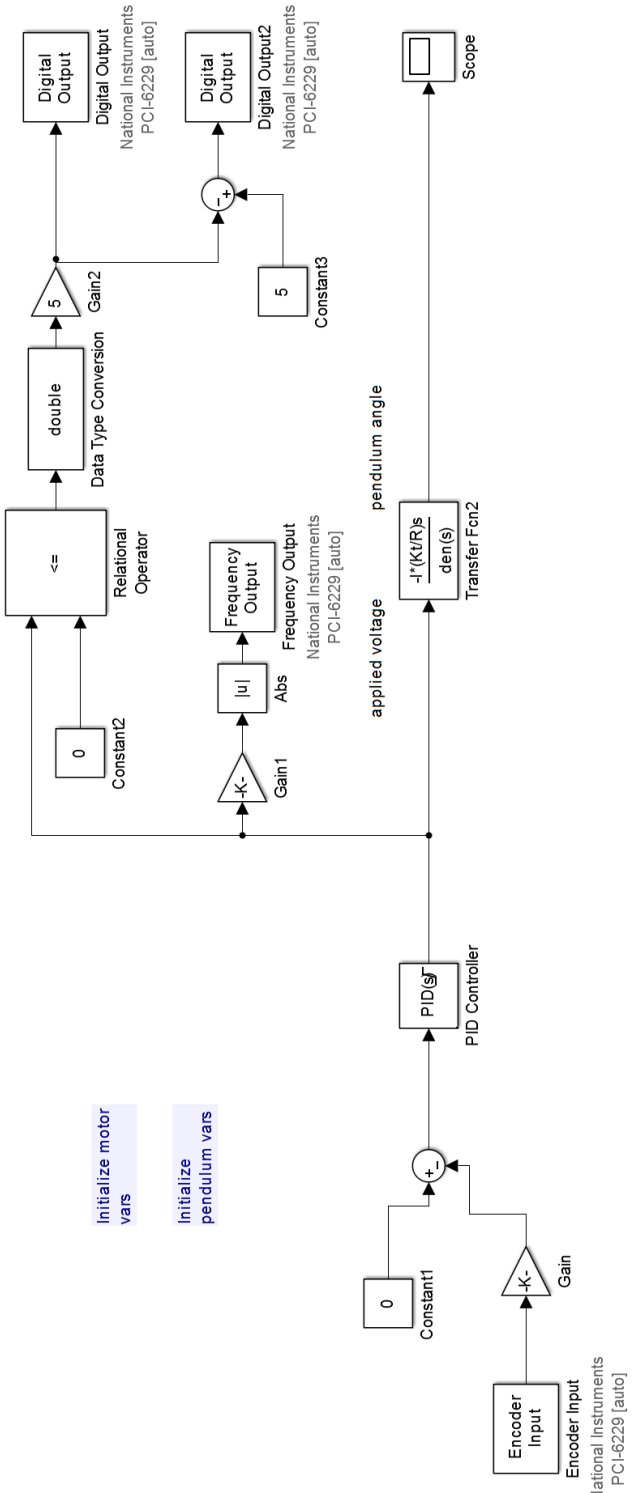
Figure 74. Built R/2R DAC.

Appendix B

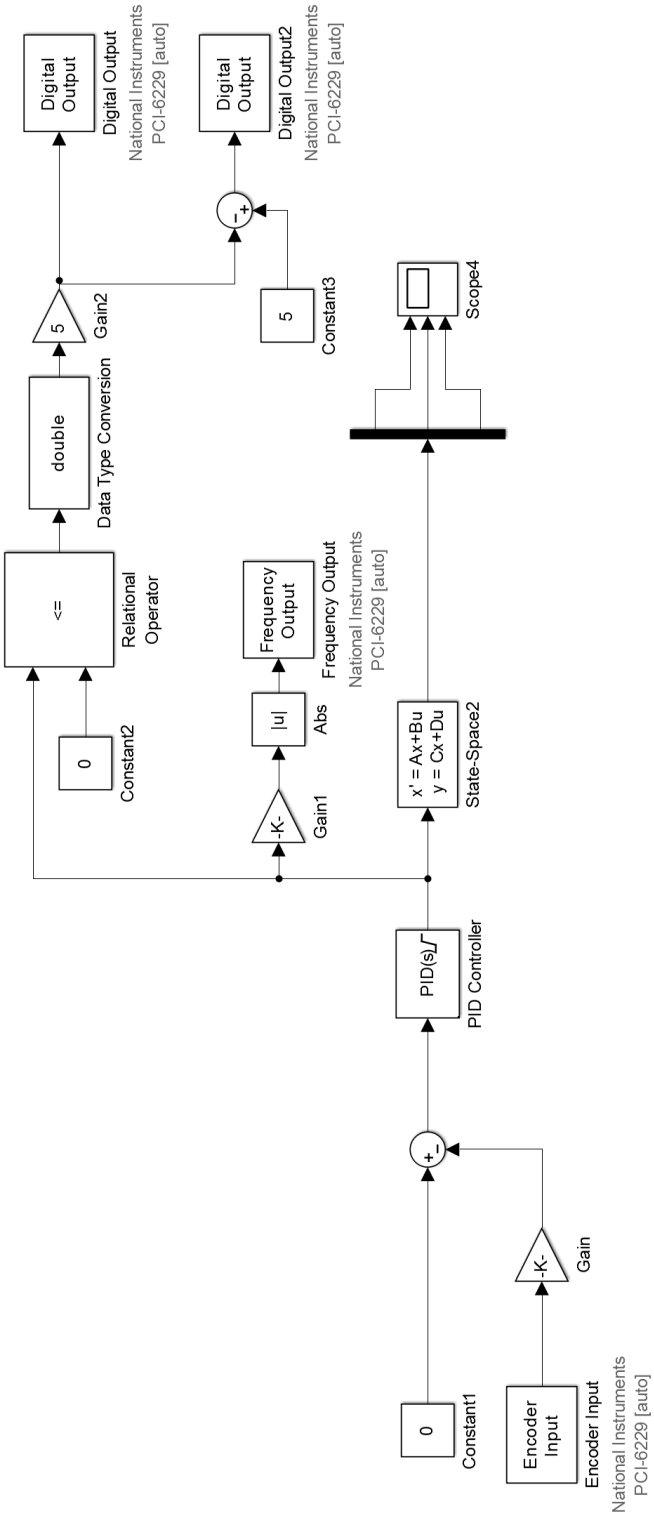
```
int oneen = 10; // enable MOTOR 1
int onedir = A5;
int onedil = A4;
int xPin = 2;    // X output of the accelerometer
int pulseX;
int posX;
void values(){

 pulseX = pulseIn(xPin,HIGH);
 posX = ((pulseX / 10) - 500) * 8;

 if(posX>1000){
  spinL(255);
  delay(50);
  stopp();
 }

 else if(posX<1000){
  spinR(255);
  delay(50);
  stopp();
 }

}


void spinR(int str){
 digitalWrite(onedir, HIGH);
 digitalWrite(onedil,LOW);
 analogWrite(oneen, str);
}
void spinL(int str){
 digitalWrite(onedil,HIGH);
 digitalWrite(onedir,LOW);
 analogWrite(oneen,str);
}
void stopp(){
 digitalWrite(onedil,LOW);
 digitalWrite(onedir,LOW);
 //analogWrite(oneen,0);
}
```

```
void setup() {
  pinMode(onedir, OUTPUT);
  pinMode(oneen, OUTPUT);
  pinMode(onedil, OUTPUT);
  pinMode(xPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  values();
}
```

Appendix C


Digital Output
Digital Output
National Instruments PCI-6229 [auto]
Digital Output
Digital Output2
National Instruments PCI-6229 [auto]
Scope
5
Gain2
5
Constant3
double
Data Type Conversion
<=
Relational Operator
Frequency Output
Frequency Output
National Instruments PCI-6229 [auto]
pendulum angle
-I*(Kt/R)s
den(s)
Transfer Fcn2
0
Constant2
|u|
Abs
-K-
Gain1
applied voltage
PID(s)
PID Controller
Initialize motor vars
Initialize pendulum vars
0
Constant1
-K-
Gain
Encoder Input
Encoder Input
National Instruments PCI-6229 [auto]

Appendix D


Digital Output
Digital Output
National Instruments PCI-6229 [auto]
Digital Output
Digital Output2
National Instruments PCI-6229 [auto]
+
-
5
Gain2
5
Constant3
Scope4
double
Data Type Conversion
<=
Relational Operator
Frequency Output
Frequency Output
National Instruments PCI-6229 [auto]
0
Constant2
|u|
Abs
-K-
Gain1
x' = Ax+Bu
y = Cx+Du
State-Space2
PID(s)
PID Controller
+
-
0
Constant1
Encoder Input
Encoder Input
National Instruments PCI-6229 [auto]
-K-
Gain