



**COMILLAS**

UNIVERSIDAD PONTIFICIA

ICAI

# GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

## INVERTED PENDULUM TEST RIG FOR CLOSED LOOP CONTROL DEMONSTRATIONS

Autor: Carmen Mora-Figueroa Cruz-Guzmán

Director: Enrique S. Gutiérrez-Wing

Madrid

Junio de 2019



**AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS**

**1. Declaration of authorship and accreditation thereof.**

The author Mr. /Ms. CARMEN MORA-FIGUEROA CRUZ-GUZEMÁN

**HEREBY DECLARES** that he/she owns the intellectual property rights regarding the piece of work: INVERTED PENDULUM TEST RIG FOR CLOSED-LOOP CONTROL DEMONSTRATIONS that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

**2. Subject matter and purpose of this assignment.**

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

**3. Transfer and access terms**

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

- a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.
- b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.
- c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.
- d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.
- e) Assign these pieces of work a Creative Commons license by default.
- f) Assign these pieces of work a **HANDLE** (*persistent URL*). by default.

**4. Copyright.**

The author, as the owner of a piece of work, has the right to:

- a) Have his/her name clearly identified by the University as the author
- b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.
- c) Request that the work be withdrawn from the repository for just cause.
- d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

**5. Duties of the author.**

The author agrees to:

- a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.

- b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
- c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
- d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.

**6. Institutional Repository purposes and functioning.**

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

- a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
- b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
- c) The University shall adopt the necessary measures to safeguard the work in the future.
- d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on ...15... of ...JUNE....., 2019

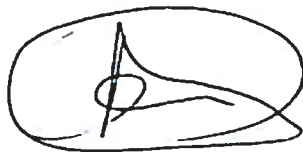
**HEREBY ACCEPTS**

Signed..........

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

I, hereby, declare that I am the only author of the project report with title:

Inverted Pendulum Test Rig for Closed Loop Control Demonstrations  
which has been submitted to ICAI School of Engineering of Comillas Pontifical  
University in the academic year 2018/19. This project is original, has not been  
submitted before for any other purpose and has not been copied from any other  
source either fully or partially. All information sources used have been rightly  
acknowledged.



Fdo.: Carmen Mora-Figueroa Cruz-Guzmán

Date: 23/07/2019

I authorize the submission of this project  
PROJECT COORDINATOR (ELECTRONICS)



Fdo.: Aurelio García Cerrada

Date: 23/07/19





**COMILLAS**

UNIVERSIDAD PONTIFICIA

ICAI

# GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

## INVERTED PENDULUM TEST RIG FOR CLOSED LOOP CONTROL DEMONSTRATIONS

Autor: Carmen Mora-Figueroa Cruz-Guzmán

Director: Enrique S. Gutiérrez-Wing

Madrid

Junio de 2019





# **Inverted pendulum test rig for closed-loop control demonstrations**

**Autora: Mora-Figueroa Cruz-Guzmán, Carmen**

Director: Gutiérrez-Wing, Enrique S.

## **RESUMEN DEL PROYECTO**

Conseguir el equilibrio de un péndulo invertido es un problema de control completo y ejemplificador, que es muy útil a la hora de explicar conceptos como la controlabilidad y observabilidad de un sistema o la imposición de polos. El sistema físico que vamos a estudiar en este proyecto, de forma simplificada, está formado por una base y un péndulo que, a través de un control aplicado al sistema, trata de conseguir que el péndulo se mantenga en equilibrio siguiendo el eje vertical y (con el centro de masa del péndulo por encima del centro de gravedad del sistema). Además, con el control se pretenderá que el péndulo vuelva siempre a su coordenada original, ya que el espacio que tiene la base para recorrer es limitado. El problema se puede reducir a un plano, moviéndose la base a lo largo del eje  $x$  y el péndulo siguiendo un movimiento circular de radio la longitud del péndulo  $L$ .

En este proyecto se va a estudiar la aplicación de un sistema de control para conseguir el equilibrio de un péndulo invertido. Este control se aplicará a un sistema mecánico real, formado en principio por una base, un motor, un péndulo y una placa de control Arduino. Este proyecto servirá como fundamento para la asignatura Senior Design I, impartida en la universidad Boston University (Boston, EEUU). Se analizarán diversos métodos de control con sus respectivos beneficios y limitaciones, incluyendo un control P (Proporcional), un control PD (Proporcional Derivativo), un control PI (Proporcional Integral), un control PID (Proporcional, Integral y Derivativo), un control por imposición de polos y un control LQR (Linear Quadratic Regulator). El sistema físico será tenido en cuenta en este proyecto, pero el montaje será llevado a cabo como parte del trabajo de fin de grado del compañero de escuela Miguel Rodríguez.

Con este trabajo de fin de grado se busca conseguir un proyecto simple, ejemplificador y replicable. Se quiere conseguir un prototipo que cubra diversos conceptos teóricos (en particular el diseño de un control PID, un método de ajuste del control y los parámetros para el diseño de este), que ejemplifique distintos métodos de control y su eficacia, y que ayude al alumno a lidiar con los problemas que surjan al tratar con un sistema real imperfecto que siga un modelo teórico.

Con el proyecto se pretenden conseguir por tanto los siguientes objetivos:

- Aplicar una forma de control a un sistema inestable.
- Facilitar la comparación del uso de diferentes estrategias de control en un sistema mecánico.
- Comprobar la fiabilidad de un modelo teórico en comparación con el sistema físico que representa.
- Evaluar resultados de forma metódica y científica, y redactar un informe oficial que sea legible, riguroso y que aporte información nueva de interés.
- Conseguir un proyecto que pueda ser replicable en una asignatura oficial de la universidad Boston University por otros alumnos (se quiere reducir el tiempo que se requiere para diseñar y construir este sistema).

Con el fin de conseguir los objetivos expuestos anteriormente, se ha seguido un exhaustivo modelo de trabajo:

En primer lugar, se ha hecho un estudio teórico a fondo del sistema físico: se ha definido el modelo de estado del sistema evaluando las ecuaciones en forma diferencial y después se han linealizado en torno a un punto de equilibrio para poder aplicar los distintos controles.

En segundo lugar, se han estudiado los distintos controles que podrían ayudarnos a conseguir nuestro objetivo de estabilizar el péndulo: control P, control PD, control PI, control PID, posicionamiento de polos y regulador LQR. Tras el estudio nos hemos decantado por el más adecuado siguiendo los criterios de eficacia y mejor relación potencia necesaria para su correcto funcionamiento frente a amortiguamiento, y también teniendo en cuenta cuál sería el más ejemplificador para el estudiante. Después de haber elegido el control más adecuado, se ha implementado el controlador en Arduino UNO. Relativo al Arduino se han tratado los temas de los *interrupts* para el correcto traspaso de instrucciones del programa al sistema físico, el cálculo del ángulo que forma el péndulo con la vertical a través de las medidas disponibles y, por último, la programación e implementación del control.

En tercer lugar, se ha analizado el funcionamiento del sistema físico. Se han elegido los elementos adecuados para conseguir un sistema físico sencillo pero funcional y al que se le pueda aplicar el control elegido anteriormente. Se ha construido el sistema de

cero, eligiendo los distintos componentes que lo formen, como por ejemplo el tipo de motor (motor DC o motor paso a paso) o el tipo de transistores (BJT o MOSFETs), y diseñando y montando el circuito que lo controla. Este diseño ha tenido en cuenta el dimensionamiento del sistema teniendo en cuenta temas físicos y relacionados con la programación del control, incluyendo el torque del motor, la velocidad de respuesta del sistema, la inercia de la base y el péndulo y la fricción existente por el contacto entre las bandas y el motor. Todos estos elementos se han tenido en cuenta de forma empírica. Este paso ha incluido pruebas preliminares para comprobar la eficacia del control, aplicándolo a un motor DC, con el que buscamos conseguir una determinada velocidad constante en el mínimo tiempo posible teniendo en cuenta detalles como el sobrepaso y el amortiguamiento de la respuesta del sistema.

La base teórica del proyecto se ha basado en información encontrada en búsquedas por internet y en la biblioteca, física y virtual, de la universidad de Boston University, además de apuntes correspondientes a diversas asignaturas cursadas a lo largo de la carrera. Se pueden encontrar las obras consultadas en el apartado de *Referencias*. Para desarrollar el modelo teórico en el que se ha basado el proyecto se ha utilizado la herramienta Matlab. El control del sistema físico se ha desarrollado en Arduino. Para la elaboración de la memoria se ha usado el sistema de composición de textos  $\text{\LaTeX}$ , específicamente a través de la plataforma online Overleaf.

El proyecto físico se ha desarrollado con recursos e instalaciones de Boston University, incluyendo los laboratorios en EMA (Engineering Manufacturing Annex), y siguiendo la tutela e instrucciones del director de proyecto Enrique Gutiérrez-Wing.

Tras haber realizado el proyecto, podemos decir que entre los controles basados en la representación de estados, el regulador LQR es el método de control más cómodo y sencillo de usar, y tiene más ventajas que el posicionamiento de polos. De todas formas, nos decantamos por utilizar el control PID por temas educativos. Este control requería un esfuerzo un poco mayor en cuanto a programación y los pasos que se tenían que seguir para implementarlo tenían una estructura lógica y fácilmente entendible.

Durante la realización del proyecto, nos encontramos con varios contratiempos. Aunque las pruebas preliminares fueron sencillas, la implementación en el sistema real fue más complicada. El mayor problema al que tuvimos que hacer frente fue en relación a los acelerómetros que colocamos en el péndulo. Estos tenían más ruido del esperado, por lo que fue imposible usar la información que proveían para calcular el ángulo. Aunque podríamos haber resuelto los problemas de precisión añadiendo un giroscopio (podríamos haber usado directamente un IMU, Inertial Measurement Unit), nos decantamos por una solución más sencilla y cambiamos de dispositivos de

lectura pasando a usar un encoder.

# **Inverted pendulum test rig for closed-loop control demonstrations**

**Author: Mora-Figueroa Cruz-Guzmán, Carmen**

Director: Gutiérrez-Wing, Enrique S.

## **SUMMARY OF THE PROJECT**

Achieving the equilibrium of an inverted pendulum is a control problem complete and exemplifying, which is very helpful when trying to explain concepts such as the controllability and the observability of a system or the imposition of poles. The physical system we are going to study in this project, in a simplified way, will be formed by a base and a pendulum that, through a control applied to the system, will try to have the pendulum stay in equilibrium following the vertical axis, being the center of mass of the pendulum above the center of gravity of the system. We want to therefore turn an unstable system into a stable one. The problem can be reduced to a plane, with the base moving across the x axis and the pendulum following a circular movement with radius L (length of the pendulum).

In this project, we will study the application of a control system to an inverted pendulum. This control will be later applied to a real mechanical system, formed in a first instance by a base, a motor, a pendulum and an Arduino control board. This project will serve as a basis for the subject Senior Design I, imparted in the university Boston University (Boston, USA). Different types of control will be studied with their corresponding advantages and limitations, such as a Proportionate Control (control P), a Proportionate Derivative Control (control PD), a Proportionate Integrate control (PI control), a Proportionate, Integrate and Derivative Control (control PID) and a Linear Quadratic Regulator (LQR regulator). The physical system will be considered in this project, but the actual building of the system will be carried out as part of a different final degree project.

The aim of this final degree project is to obtain a simple, exemplifying and replicable model. We want to obtain a prototype that covers several theoretical concepts and that shows different methods of control and their efficiency (in particular the design of a PID control, a method for its adjustment and the obtainment of the parameters for the design), and helping the student following this project learn how to deal with the problems that arise from working with an imperfect system that follows a perfect theoretical model.

With this project we are therefore trying to achieve the following objectives:

- Apply a form of control to an unstable system
- Facilitate the comparison of the use of different types of control in a mechanical system
- Test the reliability of a theoretical model compared to the physical system it is trying to represent
- Evaluate the results in a methodical and scientific way, and write a report that is legible, rigorous and that contributes with new information that can be of interest to the reader
- Procure a project that can be replicable in an official subject of the university Boston University by other students (the aim is to reduce the time needed to design and build the system presented in this report)

To reach these goals, we have followed an exhaustive work model:

Firstly, a thorough study of the system has been carried out. The equations that govern the behaviour of the system have been obtained and the state space model of the system defined. The equations were differential, and we linearized them to be able to apply the different controls.

Secondly, we studied the different controls that could help us reach our goal of balancing the pendulum: P control, PD control, PI control, PID control, LQR regulator and pole placement. After a study of the different controls we settled for the most adequate control, following a criterion of efficiency and best ratio power used versus damping. We also considered the educational value of each one. After deciding what the best control was, we implemented it in Arduino. Regarding Arduino, we had to take into account the *interrupts* for the correct translation of orders from the board to the physical system, the calculation of the appropriate state variable through the measurements available and the actual control programming.

Lastly, we studied the functioning of the physical system. We chose the most adequate elements to obtain a simple but functional physical system, one to which we could apply the control chosen previously. We had to design the system from the very beginning, having to decide the type of motor (DC motor or a steps motor) or the types of transistors (BJT or MOSFET) we were going to use, and designing and

building the necessary circuits. This design has taken into account the torque of the motor, response speed, inertia of the base and pendulum and the contact between the bands and the motor. All these elements we took into account empirically. This step included preliminary tests to check the efficiency of the control through simple tests with a DC motor which we wanted to have reaching a certain constant velocity in the least time possible, looking at details like the damping of the system.

The theoretical background of the project has been based on information found on the internet and on books in the library, physical and virtual, of Boston University, as well as notes taken in classes taken all through the engineering degree. To check the works consulted for the elaboration of this project, go to the section *References*. To develop the model we used the tool Matlab. The control system was developed for Arduino. For the elaboration of the report we used the text composition tool L<sup>A</sup>T<sub>E</sub>X, more specifically through the online platform Overleaf.

The physical project will be developed through resources and facilities belonging to Boston University including the laboratories that can be found in EMA (Engineering Manufacturing Annex), and following the instructions of director Enrique Gutiérrez-Wing.

After doing the project, we can conclude that out of the controls based on the state-space modeling of the system, the easiest and most convenient to use is the LQR regulator. Its benefit compared to pole placement are also higher, as explained in the report. Nonetheless, we chose to work with the PID control for educational reasons. This control required to put more effort in its programming, and the steps that had to be followed had a very coherent and easy to understand structure.

During the carrying out of the project we had to face several challenges. Although the preliminary run tests were fairly easy, the implementation of the control on the actual system was more complicated. The biggest problem we had to solve was regarding the accelerometers. The noise we had to deal with was higher than expected, and it made calculating the angle with the information provided by them impossible. We could have solved the problem adding a gyroscope, or directly using an IMU device, but we chose to look for an easier solution and used an encoder instead.





# Contents

<b>1</b>	<b>Introduction</b>	<b>19</b>
<b>2</b>	<b>Theoretical Study of the System</b>	<b>19</b>
2.1	State Space Model . . . . .	19
2.2	Poles and Stability of the System . . . . .	26
2.3	Controllability and Observability of the System . . . . .	27
<b>3</b>	<b>Control Systems</b>	<b>29</b>
3.1	Proportionate Control (P) . . . . .	31
3.2	Proportionate Integral Control (PI) . . . . .	31
3.3	Proportionate Derivative Control (PD) . . . . .	32
3.4	Proportionate Integral Derivative Control (PID) . . . . .	32
3.5	Pole placement . . . . .	32
3.6	LQR Regulator . . . . .	34
<b>4</b>	<b>Arduino</b>	<b>36</b>
4.1	Interrupts . . . . .	36
4.2	Run tests . . . . .	37
<b>5</b>	<b>Physical Implementation</b>	<b>41</b>
5.1	Accelerometers ADXL 335 . . . . .	42
5.2	Encoder . . . . .	45
<b>6</b>	<b>Conclusions</b>	<b>46</b>
<b>7</b>	<b>Annex 1</b>	<b>49</b>
<b>8</b>	<b>Matlab Codes</b>	<b>51</b>
8.1	Code for the differential equations defining the system . . . . .	51
8.2	Code for original functions and linearized functions . . . . .	53
8.3	Study of the observability and controllability of a system . . . . .	57
8.4	Control through pole placement and a LQR regulator . . . . .	58
<b>9</b>	<b>Arduino Codes</b>	<b>61</b>
9.1	Code for setting a constant velocity in brushed DC motor . . . . .	61
9.2	Code for implementing a constant velocity in brushed DC motor through a PID control . . . . .	63
9.3	Code for calibrating the accelerometers . . . . .	65
9.4	Code for reading from the accelerometers and calculating $\theta$ . . . . .	67



# 1 Introduction

In this project, we will analyze how to achieve the equilibrium of an inverted pendulum using a closed loop control system. This is a very clarifying example, that will allow us to study, analyze and compare different types of control such as Proportionate, Integral and Derivative control (PID control), Linear Quadratic Regulator (LQR regulator) or pole placement. It is also useful when trying to explain more abstract concepts such as the controllability or observability of a system.

Our system can be reduced to a two dimensional problem. We will study the behaviour of the base and pendulum in the XY plane, with rectilinear movement in the x axis for the base and circular movement of the pendulum over the XY plane. Our output will depend on three main variables we have control over: the type of control system we apply, the parameter  $K$  ( $K_p$ ,  $K_i$  and  $K_d$  when applying a PID control), and the sample time. This last variable is not very flexible though, since we need to take into consideration the reaction time of the microprocessor we use and will therefore be limited by it. A small sample time will make the digital response more similar to the analog one we are looking to implement. If it is too small though, the computational load might be too big for the microprocessor, and calculations of small numbers could be problematic (for example, the integrate action could be cancelled) [Aut19a].

The stabilization of an inverted pendulum has already been faced in other studies, but the approach followed has been mostly theoretical. We want this project to also focus on the physical system, with all its imperfections and problems and its corresponding solutions. The physical set up of the system will be carried out by Miguel Rodríguez, as part of his Senior Final Project.

## 2 Theoretical Study of the System

We will define the equations that govern the movement of the system formed by the base (or cart) and pendulum through Newton's Second Law ( $F = ma$ ). After linearizing those equations, we will put them in matrix form and build the state space model of the system ( $\dot{X} = AX + BU$ ;  $Y = CX + DU$ ). With our system defined, we will be able to study its poles and stability and its observability and controllability.

### 2.1 State Space Model

We will define the following variables:  $M$  for the mass of the base,  $m$  for the mass of the pendulum, and  $L$  for the length of the pendulum. We will establish the angle  $\theta$  as positive following what is expressed in Figure 1: counterclockwise, with  $\theta = 0^\circ$

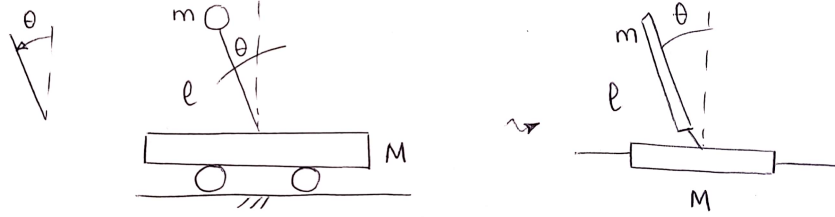


Figure 1: System formed by a cart and inverted pendulum

for pendulum coordinates  $x(t)=x$ ,  $y(t)=+L$ .  $F$  is the force applied to the cart,  $T$  the tension that appears across the length of the pendulum,  $N$  the normal force that appears between floor and base and  $g$  the force of gravity.

To apply Newton's Second Law, we will construct the free body diagrams (FBD) of the pendulum and cart (shown in figures 2 and 3) and apply it separately to the system divided by axis  $x$  and  $y$ . By doing force equilibrium and taking the acceleration of the body as the second derivative of its displacement, we can obtain the following equations for the base and the pendulum respectively:

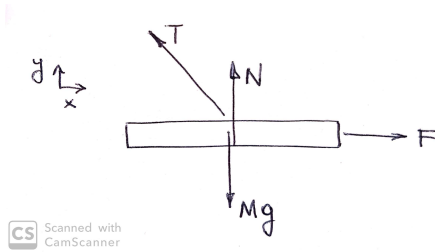


Figure 2: FBD of the base

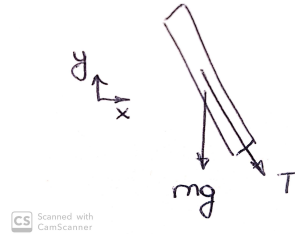


Figure 3: FBD of the pendulum

$$\begin{cases} \vec{i} \equiv F - T \sin(\theta) = M\ddot{x} \\ \vec{j} \equiv N + T \cos(\theta) - Mg = 0 \end{cases} \quad \begin{cases} \vec{i} \equiv T \sin(\theta) = m \frac{d^2}{dt^2}(x - L \sin \theta) \\ \vec{j} \equiv T \cos(\theta) + mg = -m \frac{d^2}{dt^2}(L \cos \theta) \end{cases}$$

Calculating the second derivative of the displacement of the pendulum in  $x$  and  $y$  we obtain:

$$\frac{d^2}{dt^2}(x - L \sin \theta) = \ddot{x} - L\ddot{\theta} \cos \theta + L\dot{\theta}^2 \sin \theta$$

$$\frac{d^2}{dt^2}(L \cos \theta) = -L\ddot{\theta} \sin \theta - L\dot{\theta}^2 \cos \theta$$

Replacing these derivatives in the original equations, we have the following equations:

$$\begin{cases} T \sin(\theta) = m(\ddot{x} - L\ddot{\theta} \cos(\theta) + L\dot{\theta}^2 \sin(\theta)) \\ T \cos(\theta) + mg = m(L\ddot{\theta} \sin(\theta) + L\dot{\theta}^2 \cos(\theta)) \end{cases}$$

The next step is to get rid of T. To do so we will multiply by  $\cos(\theta)$  and by  $\sin(\theta)$  the pendulum equations for  $\vec{i}$  and  $\vec{j}$  respectively and then add them. We are left with the next equation:

$$-mg \sin(\theta) = m\ddot{x} \cos(\theta) - L\ddot{\theta}m$$

To have a second equation in our system of equations we will replace  $T \sin \theta$  in the base  $\hat{i}$  equation by that found in the pendulum  $\hat{i}$  equation. We obtain the following system of two equations with two variables (x and  $\theta$ ):

$$\begin{cases} -mg \sin(\theta) = m\ddot{x} \cos(\theta) - L\ddot{\theta}m \\ F + mL\dot{\theta} \cos(\theta) - mL\dot{\theta}^2 \sin(\theta) = (m + M)\ddot{x} \end{cases}$$

To solve for  $\ddot{x}$  and  $\ddot{\theta}$ , we can put the equations in matrix form and do as follows:

$$\begin{pmatrix} m \cos(\theta) & -Lm \\ m + M & -mL \cos(\theta) \end{pmatrix} \begin{pmatrix} \ddot{x} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} -mg \sin(\theta) \\ F - mL\dot{\theta}^2 \sin(\theta) \end{pmatrix}$$

Now we can do a simple inversion ( $M_1 M_2 = M_3 \Rightarrow M_2 = M_1^{-1} M_3$ ). We are left with the following equations for  $\ddot{x}$  and  $\ddot{\theta}$ :

$$\ddot{x} = \frac{F - mL\dot{\theta}^2 \sin(\theta) + mg \sin(\theta) \cos(\theta)}{m + M - m \cos^2(\theta)}$$

$$\ddot{\theta} = \frac{mL\dot{\theta}^2 \sin(\theta) \cos(\theta) - F \cos(\theta) - (m + M)g \sin(\theta)}{mL \cos^2(\theta) - L(m + M)}$$

To apply our controls, we will need to describe the system in matrix form and define the state space model. To be able to obtain the matrices describing the movement of the system, we have to linearize the equations. We can do so considering

the variation of  $\theta$  very small around the equilibrium points ( $\theta = 0^\circ$ ,  $\theta = 180^\circ$ ). A definition for equilibrium point is to say that, if the system started at said point, it would not leave it [Dja18]. If we linearize these equations taking the first element of the Taylor series for the sines and cosines ( $\sin(\theta) = \theta$ ,  $\cos(\theta) = 1$ ), and establishing that squared values are small enough to be considered infinitesimals ( $\theta^2 = 0$ ), we obtain:

$$\ddot{x} = \frac{F + mg\theta}{M}$$

$$\ddot{\theta} = \frac{F + g\theta(m + M)}{LM}$$

After taking these steps, the system can be defined as follows:

$$\begin{cases} \dot{X} = AX + BU \\ Y = CX + DU \end{cases}$$

$$\begin{pmatrix} \dot{x} \\ \dot{\theta} \\ \ddot{x} \\ \ddot{\theta} \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{mg}{M} & 0 & 0 \\ 0 & \frac{g(M+m)}{LM} & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \frac{1}{M} \\ \frac{1}{LM} \end{pmatrix} F$$

$$Y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{pmatrix} + \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} F = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{pmatrix}$$

In Matlab, we can solve the differential equations through the functions `dsolve` and `diff` (subsection 8.1 "Code for the differential equations defining the system"). This code can be used as an example of how to write differential equations in Matlab. In our case they are not especially useful though, since the system is unstable and thus Matlab will not be able to find an explicit solution. The solution for this system in any case would be a trivial one, being the point of operation the equilibrium point. In subsection 8.2 "Code for original functions and linearized functions", we use the function `ode45` which is based on the Runge-Kutta formulas of 4th and 5th orders to resolve ordinary differential equations [Sir+04]. We also apply a simple PD control to the system, using the angle for the proportionate action and the angular velocity of the pendulum for the derivative action. We can see that although the control can be applied to the linearized system (figures 4, 5 6), applying it to the non linear system does not stabilize the system (figures 7, 8, 9).

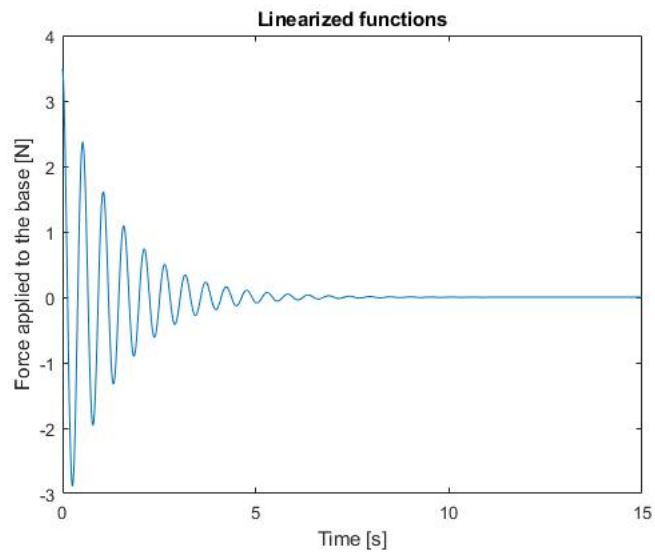


Figure 4: Force  $F$ , linear system

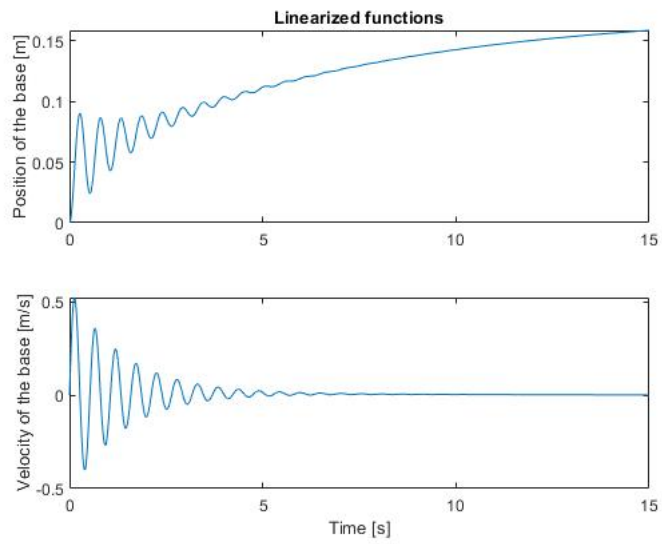


Figure 5: Position  $x$  and velocity  $v$ , linear system

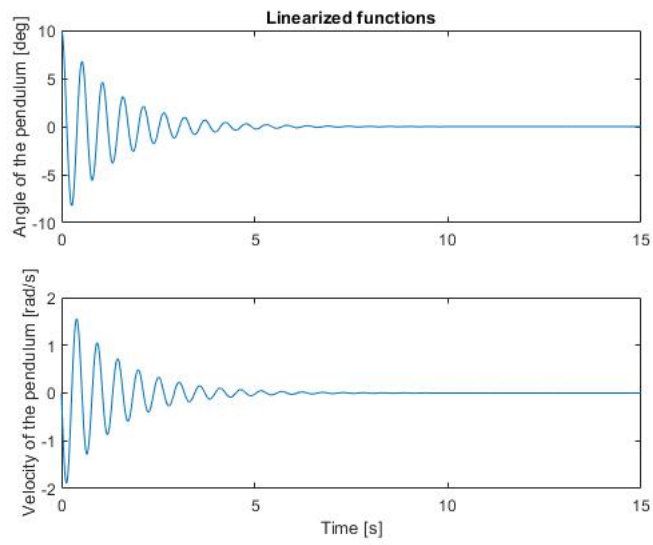


Figure 6: Angle  $\theta$  and angular velocity  $w$ , linear system

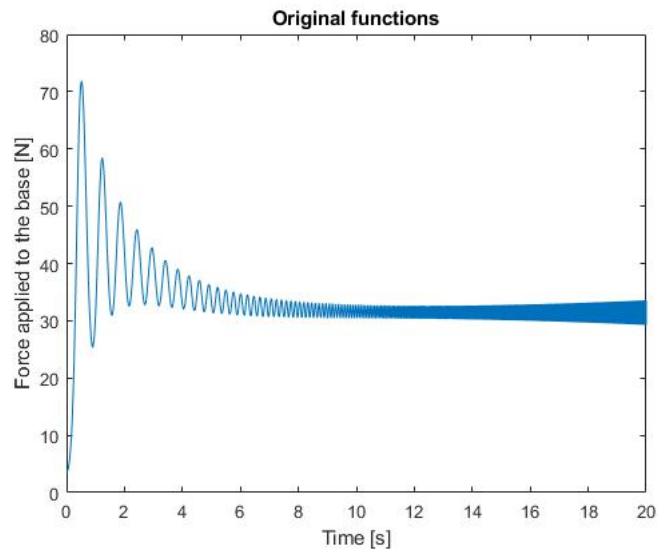


Figure 7: Force  $F$ , original system



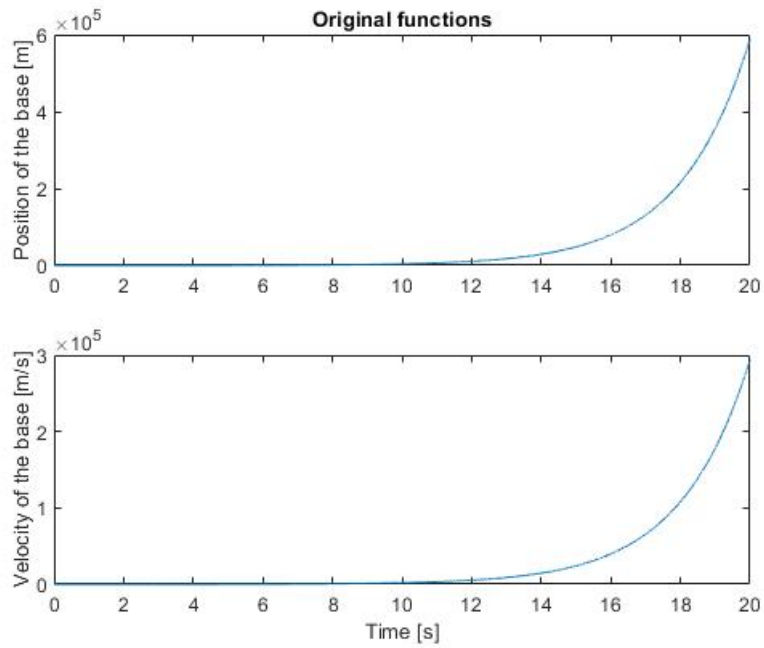


Figure 8: Position  $x$  and velocity  $v$ , original system

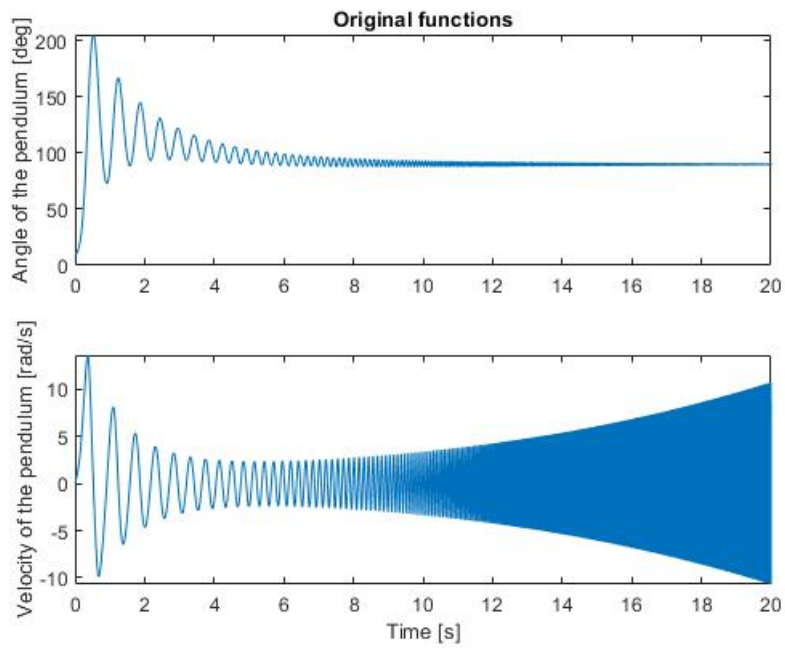


Figure 9: Angle  $\theta$  and angular velocity  $w$ , original system

## 2.2 Poles and Stability of the System

The stability of a given system depends on the eigenvalues of its matrix  $A$  ( $\dot{X}(t) = AX(t) + BU(t)$ ), which will define the poles of the system. These eigenvalues define the envelope function that describes the fading of a function, which will set the damping and natural frequency of the system, which means that for a system to be stable, it has to have negative eigenvalues so that it fades with time and reaches a constant value.

To obtain the eigenvalues of the open loop system, we need to solve the characteristic polynomial:

$$| A - \lambda I | = \begin{vmatrix} -\lambda & 0 & 1 & 0 \\ 0 & -\lambda & 0 & 1 \\ 0 & \frac{mg}{M} & -\lambda & 0 \\ 0 & \frac{g(m+M)}{LM} & 0 & -\lambda \end{vmatrix} = 0$$

$$| A - \lambda I | = -\lambda \begin{vmatrix} -\lambda & 0 & 1 \\ \frac{mg}{M} & -\lambda & 0 \\ \frac{g(m+M)}{LM} & 0 & -\lambda \end{vmatrix} = -\lambda((- \lambda)^3 - \frac{-\lambda g(M+m)}{LM}) = 0$$

$$| A - \lambda I | = -\lambda^2(-\lambda^2 + \frac{g(M+m)}{LM}) = 0$$

We obtain then the following eigenvalues:

$$\lambda_1 = 0(2)$$

$$\lambda_2 = \pm \sqrt{\frac{g(M+m)}{LM}}$$

As we can observe, this system has the eigenvalue 0 with multiplicity 2, and one positive eigenvalue. The system is clearly unstable and it is with the application of our control that we will be able to stabilize the system. We will apply a negative feed loop control. If we take for example  $u = -Kx$  as our control, we will have different eigenvalues than before:

$$\dot{x} = Ax + Bu = Ax - BKx = (A - BK)x$$

Now the poles of the system are defined by the eigenvalues of  $A - BK$ , which we can make negative. Different controls will give us different  $K$ s, as we will be able to see further on in this report.

## 2.3 Controllability and Observability of the System

When we say that a system is controllable, it means that it can be transferred from an initial state  $x(t = t_0) = x_0$  to *any* other state  $x(t = t_1) = x_1$  in a *finite* time  $t_1 - t_0 \geq 0$  [Fri86]. We can determine the controllability of the system through the controllability gramian  $W(t_0, t_1)$ :

$$W(t_0, t_1) = \int_{t_0}^{t_1} \phi(t_0, t)B(t)B^T(t)\phi^T(t_0, t)dt$$

Where  $\phi(t_0, t)$  is the state transition matrix of the system. A system is controllable if the controllability gramian is non singular or invertible for any  $t > 0$ . More information on this matrix and the controllability gramian in Annex I.

Another way of determining whether a system is controllable is through the matrix  $C$  (not to confuse with the matrix  $C$  from the state space model). For a system to be controllable, the matrix  $C$  has to be full ranked.

$$C = (B \quad AB \quad A^2B \quad \dots \quad A^{n-1}B)$$

$$B = \begin{pmatrix} 0 \\ 0 \\ \frac{1}{M} \\ \frac{1}{LM} \end{pmatrix}$$

$$AB = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_{32} & 0 & 0 \\ 0 & a_{42} & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \frac{1}{M} \\ \frac{1}{LM} \end{pmatrix} = \begin{pmatrix} \frac{1}{M} \\ \frac{1}{LM} \\ 0 \\ 0 \end{pmatrix}$$

$$A^2B = \begin{pmatrix} 0 & a_{32} & 0 & 0 \\ 0 & a_{42} & 0 & 0 \\ 0 & 0 & 0 & a_{32} \\ 0 & 0 & 0 & a_{42} \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \frac{1}{M} \\ \frac{1}{LM} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \frac{a_{32}}{LM} \\ \frac{a_{42}}{LM} \end{pmatrix}$$

$$A^3B = \begin{pmatrix} 0 & 0 & 0 & a_{32} \\ 0 & 0 & 0 & a_{42} \\ 0 & a_{32}a_{42} & 0 & 0 \\ 0 & a_{42}^2 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \frac{1}{M} \\ \frac{1}{LM} \end{pmatrix} = \begin{pmatrix} \frac{a_{32}}{LM} \\ \frac{a_{42}}{LM} \\ 0 \\ 0 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & \frac{1}{M} & 0 & \frac{a_{32}}{LM} \\ 0 & \frac{1}{LM} & 0 & \frac{a_{42}}{LM} \\ \frac{1}{M} & 0 & \frac{a_{32}}{LM} & 0 \\ \frac{1}{LM} & 0 & \frac{a_{42}}{LM} & 0 \end{pmatrix}$$

Since the matrix  $C$  is rank 4, the system is controllable. We can also check the controllability of a system directly in Matlab through the command `contr=rank(ctrb(sys_ss))`, which gives you the rank of the  $C$  matrix [Aut].

*Note:*  $a_{32} = \frac{mg}{M}$ ,  $a_{42} = \frac{g(M+m)}{LM}$

When referring to a system as observable, we mean that a state  $x_t$  can be inferred by having a finite record of the output  $y(\tau)$  ( $t_0 \leq \tau \leq t_1$ ) [Fri86]. The observability of a system can be determined through the observability gramian  $M(t_0, t_1)$ :

$$M(t_0, t_1) = \int_{t_0}^{t_1} \phi^\top(t_0, t) C^\top(t) C(t) \phi(t_0, t) dt$$

The system is observable if the observability gramian is non singular. More information on said gramian in Annex I.

Another easy way of checking whether a system is observable is establishing if the matrix  $O$  is fully ranked.

$$O = \begin{pmatrix} C \\ CA \\ CA^2 \\ \dots \\ CA^{n-1} \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

$$CA = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_{32} & 0 & 0 \\ 0 & a_{42} & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$CA^2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & a_{32} & 0 & 0 \\ 0 & a_{42} & 0 & 0 \\ 0 & 0 & 0 & a_{32} \\ 0 & 0 & 0 & a_{42} \end{pmatrix} = \begin{pmatrix} 0 & a_{32} & 0 & 0 \\ 0 & a_{42} & 0 & 0 \end{pmatrix}$$

$$CA^3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & a_{32} \\ 0 & 0 & 0 & a_{42} \\ 0 & a_{32}a_{42} & 0 & 0 \\ 0 & a_{42}^2 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & a_{32} \\ 0 & 0 & 0 & a_{42} \end{pmatrix}$$

$$O = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & a_{32} & 0 & 0 \\ 0 & a_{42} & 0 & 0 \\ 0 & 0 & 0 & a_{32} \\ 0 & 0 & 0 & a_{42} \end{pmatrix}$$

The range of the matrix  $O$  is 4, and therefore the system is observable. We can also check the observability of a system directly in Matlab through the command `obsrv=rank(obsv(sys_ss))`, which gives you the rank of the matrix  $O$  directly [Aut].

*Note:*  $a_{32} = \frac{mg}{M}$ ,  $a_{42} = \frac{g(M+m)}{LM}$

In section 8.3 "Study of the observability and controllability of a system" we can see the application of the before mentioned Matlab functions to check the observability and controllability of our system.

### 3 Control Systems

Physical systems are normally flawed, with issues stemming from retards in signal flows to the existence of perturbations. We define as perturbation conditions that make the theoretical model differ from the real system. Although perturbations can be considered when designing a control, it is not in the scope of this project. For example, a perturbation in our system will be friction, a variable we did not take into account while defining our theoretical model. A control system allows the designer to reduce those errors that occur because of physical imperfections.

There are two main types of systems, open loop and closed loop systems. Open loop systems receive an input and exit an output which depends on the model of the system or plant. The output of the system depends only on the plant and it does not take into account physical imperfections. Thus, there will always be present an error, a difference between the desired output and the real output. In the case of closed

loop systems, information about the output is fed back constantly to the beginning of the control system and compared to the input to make the necessary changes that will make the error as small as possible [Mac+18].

We will use the following nomenclature to define the variables in the controller:

- $u(t)$ : input, control signal (voltage)
- $r(t)$ : reference, desired value of the output (voltage equivalent to  $\theta = 0^\circ$ )
- $x(t)$ : state variable (angle  $\theta$ , position  $x$ )
- $y(t)$ : output of the system
- $y_m(t)$ : the measured output (voltage coming from the encoder)
- $e(t)$ : error, the difference between the desired output and the real one ( $e(t) = r(t) - y_m(t)$ )

There are different types of control systems, and these are implemented through the programs that govern the actions of the physical system. In the case explored in this report, an inverted pendulum, the control will be applied to the voltage that controls the motor that administers the force on the base and which will allow us to balance the system. We will be constantly measuring the position of the base and the angle formed between the arm of the pendulum and the vertical axis. The input of the system will be the voltage applied to the motor and the output will be a voltage provided by the device used to measure the position of the pendulum, through which we will calculate the angle between the vertical axis and the pendulum.

To visualize the controls it is useful to use block diagrams. Instead of using the space model to create the diagram, which makes it messier and more complicated to understand than necessary, we can derive the transfer function that models the plant from the space model matrices and apply the control directly to it. We can obtain this transfer function through the Matlab function `sys=tf(ss(A, B, C, D))`. For  $M=0.5$  kg,  $m=0.1$  kg and  $L=0.2$  m, we obtain the following transfer function:

$$P(s) = \begin{pmatrix} \frac{2s^2+1.95*10^{-14}-98.1}{s^4-8.88*10^{-16}s^3-58.86s^2} \\ \frac{10s^2}{s^4-8.88*10^{-16}s^3-58.86s^2} \end{pmatrix}$$

This matrix has two terms, one for each output ( $x, \theta$ ). Depending on what state we are controlling, we will use one term or the other in our block diagram.

To have a first contact with the system's controls we will use Matlab. We will simulate each control and evaluate their efficiency. We will then use our findings to program the best control on Arduino. We will decide which is the best control taking into consideration the rapidness and damping of the output.

### 3.1 Proportionate Control (P)

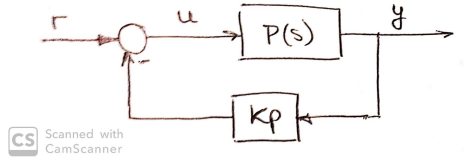


Figure 10: Control P State Space Model

This is the most simple control we can apply to a system. The input varies depending on the output: if the output measured deviates a lot from the desired output, the change in response will be bigger than if the deviation is small. As shown in figure 10, the change of input follows the next equation:

$$u(t) = K_p e(t)$$

As we can observe in this equation, if the pendulum is falling to the left with a positive angle, the control will apply a negative force to the left, since  $r(t) = 0$  and  $y_m(t) > 0$ , resulting in a negative control signal ( $u(t) < 0$ ).

### 3.2 Proportionate Integral Control (PI)

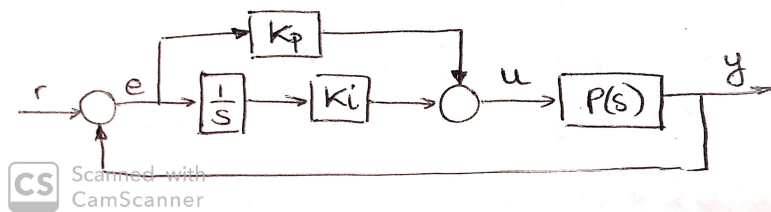


Figure 11: Control PI State Space Model

For this type of control we will apply an integral component to the control, to help the proportionate control eliminate the signal following and the perturbation errors. We can see in figure 11 how the change of input follows the next equation for the PI control:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt$$

### 3.3 Proportionate Derivative Control (PD)

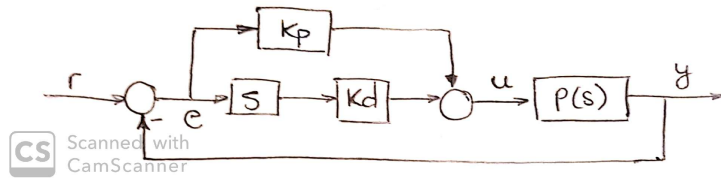


Figure 12: Control PD State Space Model

To make the control faster, improve the damping coefficient or make the output more precise, we add the derivative component to the proportionate control. It can be defined as the following:

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt}$$

### 3.4 Proportionate Integral Derivative Control (PID)

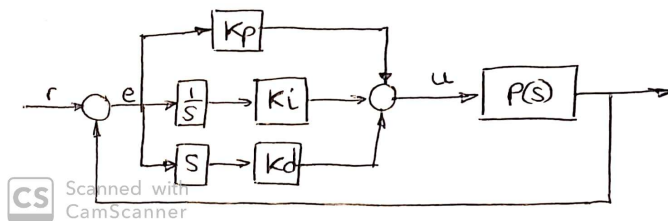


Figure 13: Control PID State Space Model

We can add all three controls for the best results:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt}$$

The values for  $K_p$ ,  $K_i$  and  $K_d$  are chosen arbitrarily through trial and error, as we will see further on in subsection 4.2 "Run tests".

### 3.5 Pole placement

As explained before, the stability of a given system depends on the eigenvalues of its matrix A, which will define the poles of the system. These eigenvalues define the



envelope function that describes the fading of a function, which will set the damping and natural frequency of the system. If a system is controllable, we can force a system to have any pole values that we want with the function for pole placement in Matlab. We will set real, negative values, so that the function fades with time and doesn't grow to infinite (14). The more negative the poles, the more stable the system, but also the higher the overshoot, the bigger the control signal that is needed and the higher the chance of saturation (figure 15).

For pole placement we will use the function  $K = \text{place}(A, B, p)$ , where  $A$  and  $B$  are the state matrices that define the system and  $p$  is a vector with the location of the poles that want to be imposed. The solution to this function is the  $K$  that can be used in the close loop system, and can be defined as a type of proportional control [Gar+16]. This type of control would have a feedback  $u = -Kx$  such that the values in the vector  $p$  are as close as possible as the eigenvalues of the closed loop  $A - BK$  (section 2.2).

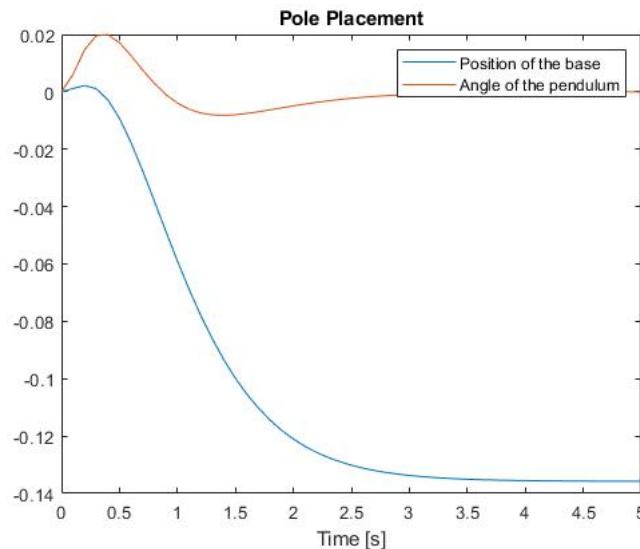


Figure 14: Pole placement

Pole placement, although being very convenient at times, can also pose some problems. For example, the case might be when you do not know where the poles have to be placed. It can also happen that you saturate the signals if you place the poles too far away from the origin, or you might find that your power source is not able to create a signal big enough to place the poles where you specified [Fri86]. It is then useful to look at other types of control like the LQR regulator.

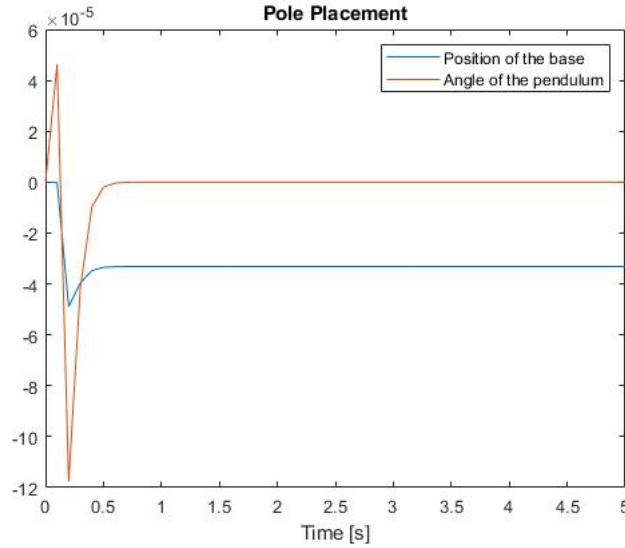


Figure 15: Pole placement with more negative values

### 3.6 LQR Regulator

A Linear Quadratic Regulator (LQR regulator) is based on the minimization of the cost function  $J$  for a given linear control law  $u(t) = -Kx(t)$  [Alo10]. This cost function is defined as:

$$J = \int_0^{\infty} (x(t)^T Q x(t) + u(t)^T R u(t) + 2x(t)^T N u(t)) dt$$

We will use the Matlab expression  $[K, S, e] = \text{lqr}(A, B, Q, R, N)$  to minimize said function, where  $A$  and  $B$  define the system,  $Q$  is a *weighing matrix* and has to be a symmetric positive semi-definite matrix,  $R$  is another *weighing matrix* and has to be symmetric and positive definite and  $N$  is almost always 0. Positive semi-definite matrix means that the product  $z^T A z$  (scalar) has to be zero or positive, while for positive definite, that same product has to be strictly positive.  $z$  is every non zero column vector of size  $n$  ( $A$  being of size  $n \times n$ ). The term  $x(t)^T Q(t) x(t)$  represents the penalty for the state  $x$  deviating from the origin and  $Q$  estates how much each component of the state vector matters in the control relative to each other, and the term  $u(t)^T R(t) u(t)$  is intended to limit the magnitude of the control signal  $u(t)$  [Fri86].

For choosing  $Q$  we will follow a simple reasoning. Since we are interested in our input related to how it affects our output ( $y = Cx$ ), we will establish the following criterion:

$$y = Cx$$

$$y^2 = x^T C C^T x$$

$$Q = C C^T$$

We can follow a similar approach for choosing R. We have to consider that although sometimes the designer wants the signal to saturate because it is the fastest way to get a response, this is not our case, since a saturated signal can make the system deviate from the predicted model and become unstable. We will then choose a large R.

The solution that minimizes J is  $u = -Kx$ , where  $K = R^{-1}(B^T P + N^T)$  and P is the solution to the Riccati equation

$$A^T P + P A - (P B + N) R^{-1} (B^T P + N^T) + Q = 0$$

[Aut19b]. The solution given by the Matlab expression is K as defined previously, S is equal to the previously defined P (solution to the associated Riccati equation) and e are the eigenvalues associated to A-BK.

We can solve a problem with a LQR regulator on Matlab, as seen in section 8.4 "Control through pole placement and a LQR regulator". We can also see the difference in efficiency depending on the values given to Q: the bigger the matrix, the more quickly the system responds:

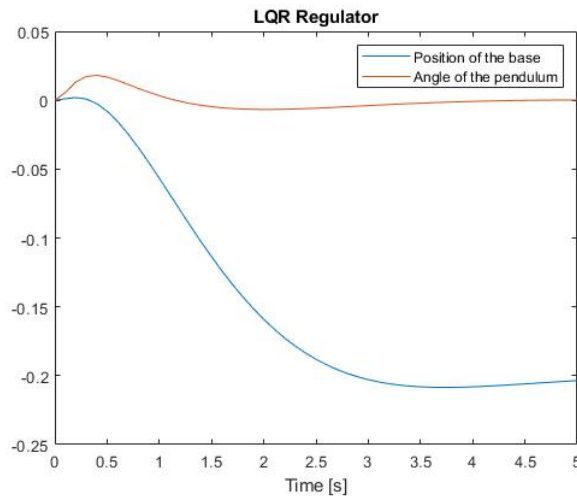


Figure 16:  $Q=CC'$

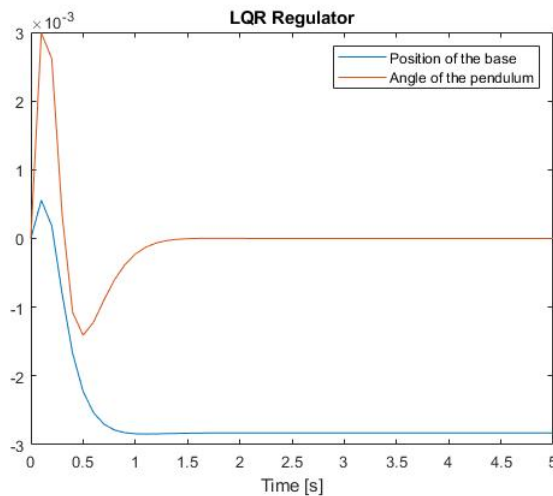


Figure 17: Higher values in Q

## 4 Arduino

Arduino is a company that distributes open sourced hardware and software. We chose to use its components because of how easy it is to learn about it and interact with it, and because of the great amount of information available online provided by its wide community of users. In this project we have used the board Arduino UNO to apply the control systems to our inverted pendulum. In this section we will discuss the use of interrupts, the preliminary tests runs we did to test our controls and the final control.

### 4.1 Interrupts

Interesting characteristics that this project allows us to explore are the interrupts. In Arduino, interrupts are used to pause whatever action the microprocessor is carrying out at that moment and execute a different action. After this later action has been fulfilled, the microprocessor goes back to whatever action it was taking on before the interrupt. We will use interrupts in this project for reading the input from the encoder, so we do not miss any pulse readings. If we did not use interrupts for reading the pulses coming from the encoder, the program could be carrying out a different function at the moment of an incoming pulse and miss its reading.

We will not use directly the function `interrupt()`. Instead, we will use the functions `millis()` and `micros()`, which use interrupts to work. Their aim is to count the milliseconds or microseconds (respectively) that have passed since the function they find themselves in started working. We will also use `delayMicroseconds(us)`,

which pauses the program for the amount of `us` specified.

## 4.2 Run tests

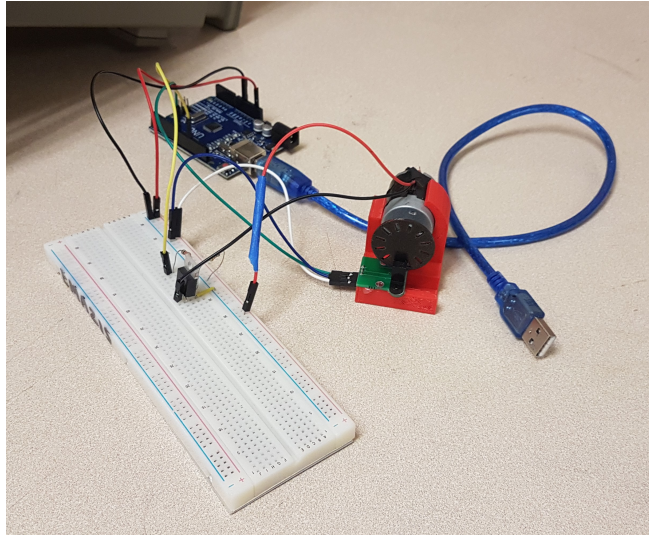
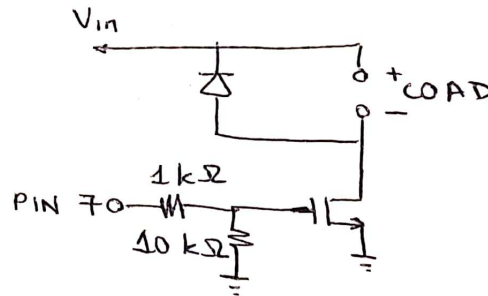


Figure 18: Physical system composed by Arduino UNO board, brushed DC motor, encoder and corresponding circuit

Before applying the control to the final system, we first did some preliminary runs with a simple brushed DC motor. We used an encoder to measure pulses and used that pulse count to calculate the velocity of the motor. As seen in section 9.1 "Code for setting a constant velocity in brushed DC motor", a constant velocity can be reached easily without a control, augmenting the velocity if the current one is below the target, and reducing it if it goes above. This is not very efficient though, and the velocity is constantly fluctuating. In section 9.2 "Code for implementing a constant velocity in brushed DC motor through a PID control" we can find the applied control to the system (figure 18). Our aim was to reach a steady velocity of 2200 rpm in the minimum time possible.

For connecting the Arduino board and the physical system, we had to use a simple circuit that would allow current and signal flows. We followed the one proposed by Professor Enrique Gutiérrez in class ME360. As shown in figure 19, we used a P30N06LE MOSFET, a 1k resistor and a 10k resistor.

Regarding the control, the most critical part was deciding in the values for the different  $K$ s. As mentioned before, the values for  $K_p$ ,  $K_i$  and  $K_d$  are chosen arbitrarily. While varying the different  $K$  values we observed how much each component of the



CS Scanned with CamScanner

Figure 19: Circuit connecting Arduino and motor

control (proportionate, integral and derivative) affected the output. To decide which were the best values, we took all our trials and entered the values for the different  $K_s$ , the maximum velocity reached ( $w_{max}$ ) and the time it took for the system to first reach the target velocity ( $t_{rise}$ ) in Excel. We then ordered the information following several criteria, including which were the fastest controls (lowest  $t_{rise}$ ) and the ones with less dumping (highest  $w_{max}$ ). We then chose the most representative trials to plot in Matlab and later analyze them. We chose to plot the trials with the following K values, as shown in table 1:

Control	$K_p$	$K_i$	$K_d$
PID	0.003	0.03	0.015
PID	0.015	0.03	0.015
PI	0.003	0.03	0
PD	0.003	0	0.015
P	0.015	0	0

Table 1: Most representative trials, with  $K_p$ ,  $K_d$  and  $K_i$  values

Arduino allows you to plot the data it reads from the Arduino board through its tool Serial Plotter. The precision of these graphs is not very high though, so to treat the data we decided to copy and paste the values produced by the tool Monitor Serie, and then process them on Matlab. We used a simple code to treat them, taking off the initial zero values (kick off values):

```

1 thresh=0;
2
3 indb=find(datos.b(:,2)<=thresh);

```

```

4 indb=max(indb);
5 datos_b=datos_b((indb:end),:);
6 t0b=datos_b(1,1);
7 datos_b(:,1)=datos_b(:,1)-t0b;
8
9 indc=find(datos_c(:,2)<=thresh);
10 indc=max(indc);
11 datos_c=datos_c((indc:end),:);
12 t0c=datos_c(1,1);
13 datos_c(:,1)=datos_c(:,1)-t0c;
14
15 figure(1),plot(datos_b(:,1),datos_b(:,2),datos_c(:,1),datos_c(:,2))

```

Listing 1: Treating Arduino data

After processing the data, we were able to easily compare the different controls by looking at the graphs we obtained on Matlab.

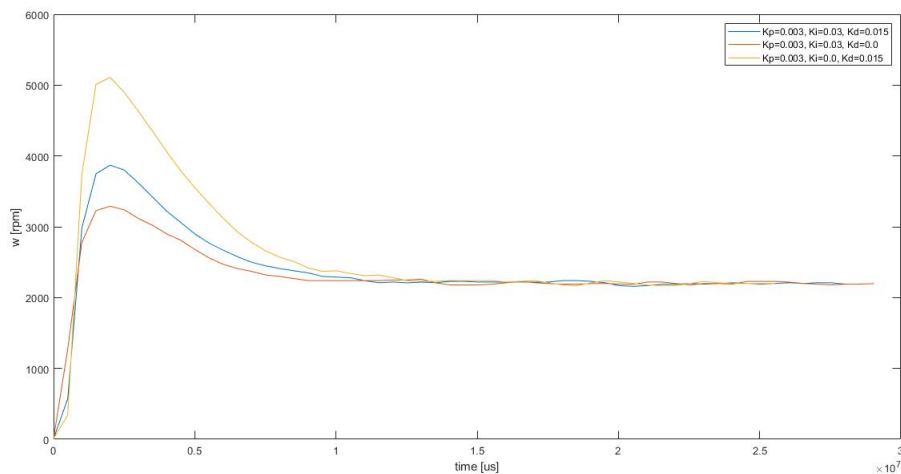


Figure 20: Graph obtained through Matlab with Arduino data (PD, PI, PID)

Analyzing graph 20, we can conclude that, for a given  $K_p$ , the controls can be ordered following the lowest  $t_{rise}$  criteria as PD, PID and PI. We can conclude that the derivative action makes the control faster.

As seen in figure 21 we can observe that raising  $K_p$  makes the control faster (PID control with  $K_p = 0.015$  is faster than PID control with  $K_p = 0.003$ ), but it also lowers the damping ( $w_{max}$  is higher for the first control than the second one).

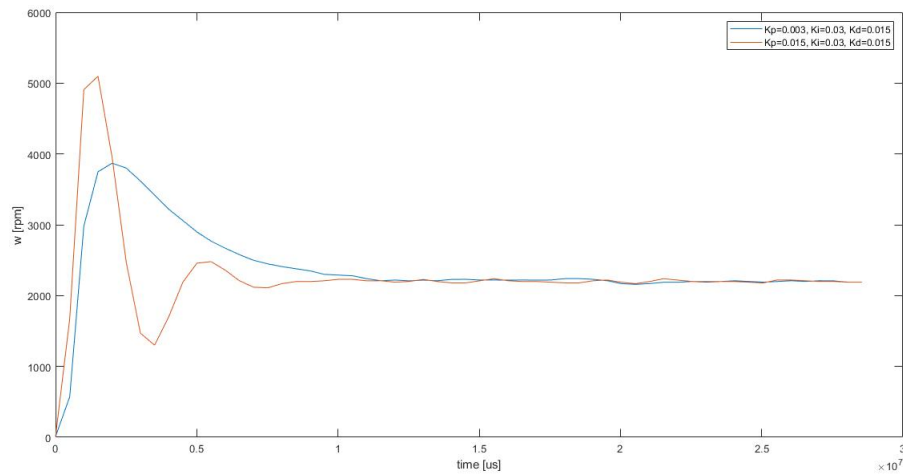


Figure 21: Graph obtained through Matlab with Arduino data (P)

When designing our control, we have to decide what to prioritize, whether rapidness of reaction of the control or lower damping (figure 22).

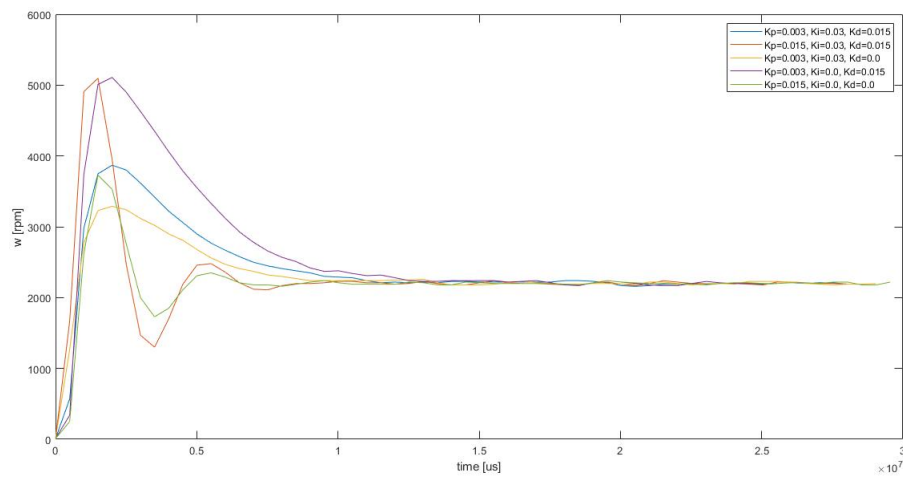


Figure 22: Graph obtained through Matlab with Arduino data (all controls)



## 5 Physical Implementation

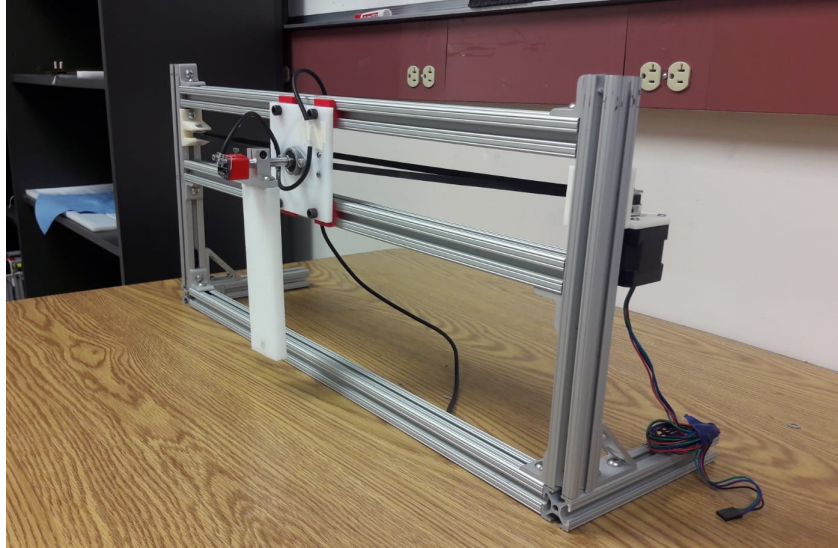


Figure 23: Physical system

For the physical system, we used the system put together by Miguel Rodríguez as his Senior Project. The Arduino board used was the same as the one used in the trial runs, an Arduino UNO board, and we changed the brushed DC motor to a stepper motor. For calculating the angle of the pendulum we first tried using accelerometers, model ADXL 335. After some problems including inaccurate calculations of the angle  $\theta$  we had to change devices and use a digital encoder instead.

It is important to note that in a general case where we do not have an initial position for the base and where using a DC motor (for example in the case of a segway), the position of the cart  $x$  cannot be given a value for each set moment and should therefore not be included in the state space model. When we look at the point of operation  $\dot{X} = 0$ , we set the velocity of the cart to 0. Since the position of the cart is calculated as the integration of the velocity, we obtain an indefinite cart position for that point of operation (integral of 0 is an undefined constant), and when we do have a velocity, the position keeps changing, never having a definite value. In our case though, we are going to use a stepper motor (figure 24, center bottom), which will allow us to calculate the position and velocity of the cart at every moment.

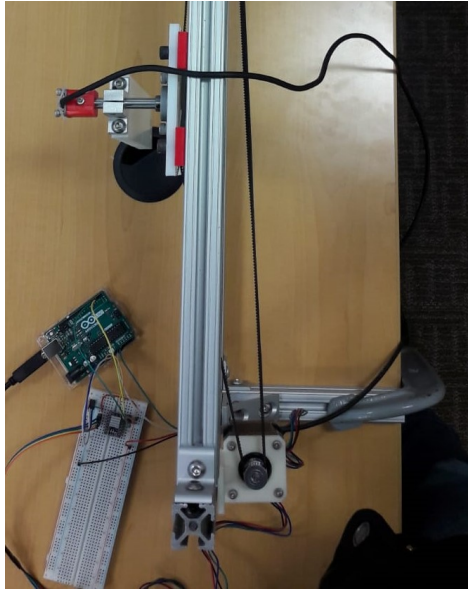


Figure 24: Stepper motor

## 5.1 Accelerometers ADXL 335

To calculate the angle  $\theta$  and have the state of our system defined, we first tried using the accelerometers ADXL 335 to obtain the accelerations of the pendulum in x and y and with them calculate the angle. Because of a problem of accuracy in the readings of the accelerations, we had to change tactics later on, but we will explain the procedure followed in any case.

The accelerators were connected to analogical input pins A0 and A1 of the Arduino board. They were placed on the upper side of the pendulum, as shown in figure 25:

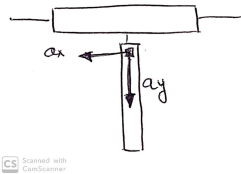


Figure 25: Accelerometers on System

To calculate the angle  $\theta$  we had to use the lectures provided by the two accelerometers, which provided voltages proportional to accelerations in x and y of the pendulum,  $a_x$  and  $a_y$ , following the directions showed in figure 26:

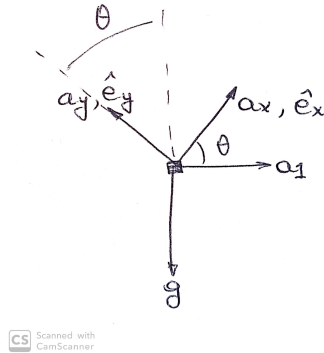


Figure 26: Accelerations given by accelerometers

From diagram 26 we can obtain the equations that govern the movement of the pendulum:

$$\begin{aligned} a_1 \vec{i} - g \vec{j} &= a_x \vec{e}_x + a_y \vec{e}_y \\ a_1^2 + g^2 &= a_x^2 + a_y^2 \end{aligned}$$

With the unitary vectors  $\vec{e}_x$  and  $\vec{e}_y$  defined as follows:

$$\begin{aligned} \vec{e}_x &= \cos \theta \vec{i} + \sin \theta \vec{j} \\ \vec{e}_y &= -\sin \theta \vec{i} + \cos \theta \vec{j} \end{aligned}$$

From these equations we wanted to obtain the angle  $\theta$ . We could do so simplifying the equations to obtain the solution for  $\cos \theta$  and  $\sin \theta$ , and doing the quotient to obtain  $\tan \theta$  and later on calculating  $\arctan \theta$  to obtain  $\theta$ :

$$\begin{aligned} a_x &= g \sin \theta + a_1 \cos \theta \\ a_y &= g \cos \theta - a_1 \sin \theta \end{aligned}$$

Which translated into matrix form is:

$$\begin{pmatrix} a_x \\ a_y \end{pmatrix} = \begin{pmatrix} g & a_1 \\ -a_1 & g \end{pmatrix} \begin{pmatrix} \sin \theta \\ \cos \theta \end{pmatrix}$$

After solving for  $\begin{pmatrix} \sin \theta \\ \cos \theta \end{pmatrix}$ , we obtained the following expression for  $\tan \theta$ :

$$\tan \theta = \frac{\sin \theta}{\cos \theta} = \frac{ga_x - a_1a_y}{ga_y + a_1a_x}$$

To have  $\theta$  fully defined, apart from the readings for  $a_x$  and  $a_y$ , we were only missing  $a_1$ , which could be easily deduced from an earlier equation:

$$a_1 = \sqrt{a_x^2 + a_y^2 - g^2}$$

To obtain  $a_x$  and  $a_y$ , we had to convert the readings provided by the accelerometers, for which we did the following. When placing the accelerometers on the pendulum, we had to be careful with the direction of the coordinates. As we can see in figure 27, the accelerometers have a device on the inside that measure pressure. If the spring in x suffers a force of tension while on the first and second quadrants ( $a_x$  is perpendicular to the length of the pendulum, as seen in figure 25), the accelerometer in x will read positive. If on the other hand it gets compressed while on the third and fourth quadrants, the reading will be negative. It is the same way the accelerometer in y works. Another way to visualize  $a_x$  and  $a_y$  is by projecting them on a vector of value  $g$ , in the positive y direction.



Figure 27: Inner workings of the accelerometers

To procure accurate readings, we had to calibrate the accelerometers. Standard values measured by the accelerometers are numbers between 0 and 1023. This is not always the case. After obtaining the range of values measured by the accelerometers, we had to transform these values to an interval between  $-g$  and  $g$  ( $-9.81$  and  $9.81$ ).

To do this we used a simple program to obtain the minimum and maximum values read by the accelerometers and transform it to the  $2g$  interval. We show this code in the subsection 9.3, "Code for calibrating the accelerometers and scaling the readings". We followed a simple rule of three, where  $a_{xmin}$  is to  $-g$  and  $a_{xmax}$  is to  $g$  as the acceleration we looked for is to the value read by the accelerometers. We followed the same process for the y accelerometer.

We then used the accelerations to calculate the angle  $\theta$ , as presented in subsection 9.4, "Code for reading from the accelerometers and calculating  $\theta$ ".

This method, although apparently very simple, gave us more problems than expected. Because of the noise, the accuracy of the accelerations read by the accelerometers was not high enough to give proper readings of  $a_x$  and  $a_y$ , so when calculating  $a_1$  to insert in the formula for acquiring  $\tan(\theta)$ , it resulted in negative numbers inside a square root, which gave us imaginary numbers and made the angle impossible to calculate.

We could have used an Inertial Measurement Unit (IMU) to avoid the problems encountered when calculating  $a_1$ , since this device can measure the velocity, orientation and acceleration of a system simultaneously (combining a 3 axis gyroscope and a 3 axis accelerometer). After using a complementary filter (no need to use the more complicated, statistically based Kalman filter in our case), we would have been able to calculate our angle accurately.

A complementary filter works as a low pass filter for the measurements coming from the accelerometer (taking out the excessive noise) and as a high pass filter for the measurements coming from the gyroscope (taking out the *drift*, error accumulated from the calculation of the angle in the device). The filter, in its most basic application, looks like this:

$$\theta = a(\theta_{prev} + \theta_{gyro}) + b\theta_{accel}$$

Where  $a$  and  $b$  are two constants that work as weighs, with the condition that the sum of both constants has to be 1. Initially you can give them the values 0.98 and 0.02 respectively, and calibrate them accordingly [Lla16].

We did not have any IMUs available though, so we had to look for another, more simple solution: an encoder.

## 5.2 Encoder

An encoder is a device that can translate movement into an electric signal that can be used in the system's control. The encoder has different uses, such as determining the direction of rotation, the velocity, or the position or angle of the device's axis it is set on.

With an accurate reading of the angle, we can apply the control chosen, which would have two phases: first, we would have the pendulum swing at the oscillating angular speed, raise the pendulum above a certain limit angle ( $\theta_{limit}$  in figure 30), and then switch controls to have the pendulum stay inverted vertically.

Although lack of time prevented us from trying out the final control, we believe it would be efficient and not complicated to implement, given the results of the run tests. One of the key aspects would be to have the pendulum reach the critical angle above the horizontal line and not just oscilate at the set angular velocity. A *push*, a

sudden stop of the motor and change of direction, would be critical to obtain such result. After reaching the critical angle we would switch controls and instead of having as the main goal to reach a certain angular velocity, we would control the angle and have it reach  $\theta = 0^\circ$ .

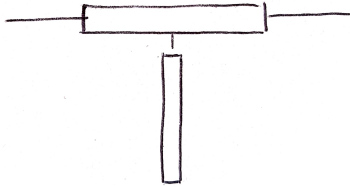


Figure 28: Phase 1

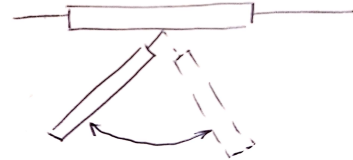


Figure 29: Phase 2

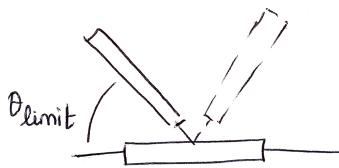


Figure 30: Phase 3

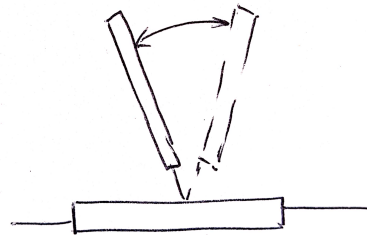


Figure 31: Phase 4

## 6 Conclusions

From this project we can reach a series of conclusions, which we will briefly summarize in this section.

An unstable system, such as an inverted pendulum, can be stabilized through the application of a control. We can know whether a system is unstable through the previous analysis of the eigenvalues of its defining matrix  $A$  ( $\dot{X} = AX + BU$ ). The eigenvalues are the roots of its characteristic polynomial ( $|A - \lambda s| = 0$ ). If the real part of any of its eigenvalues is positive, the response of the system will not fade in time and will never reach a specific value.

To know whether we can apply a control to a system we must first check whether it is controllable. A system will be controllable if its matrix  $C$  ( $C = (B \ AB \ A^2B \ \dots \ A^{n-1}B)$ ) is full ranked. In the case of an inverted pendulum, the system is indeed controllable.

After linearizing the equations that define the behaviour of the system, there are different controls we can apply to it. Some have to be applied to the system in state space form, such as pole placement or LQR regulator, but this is not always the case, such as with the PID controller. Although the most convenient control is the LQR regulator, for educative purposes we have chosen to implement the PID control to our inverted pendulum.

Before implementing a control, it is very useful to do running tests to see how different  $K$ s affect the response of the system. Although  $K$ s vary from system to system therefore making the  $K$ s obtained in the running tests fairly useless, it is important to understand how each parameter affects the system's response.

When it comes to finally applying a control to a physical system, it has to be noted that many problems will arise along the way. In our case, these problems mostly stemmed from the use of measuring devices. Although accelerometers can seem a good way to get measures that can be later used to calculate the angle we want to apply the control to, it is not the case because of the amount of noise perceived by them. The problem could be solved through the use of a gyroscope, but as it is most always the case in engineering, the best solution is the most simple one. In our case, the use of an encoder.





## 7 Annex 1

In this Annex we will discuss the transition matrix, the Peano Beaker series and the variation of constants formula.

We can find the solution to a difference equation  $x(k+1) = A(k)x(k) + B(k)u(k)$  following a recursive method:

$$\begin{aligned}
 x(k) &= A(k-1)x(k-1) + B(k-1)u(k-1) = \\
 &= A(k-1)A(k-2)x(k-2) + A(k-1)B(k-2)u(k-2) + B(k-1)u(k-1) = \\
 &= A(k-1)A(k-2)[A(k-3)x(k-3) + B(k-3)u(k-3)] + \\
 &+ A(k-1)B(k-2)u(k-2) + B(k-1)u(k-1) = \\
 &= A(k-1)A(k-2)A(k-3)x(k-3) + A(k-1)A(k-2)B(k-3)u(k-3) + \\
 &+ A(k-1)B(k-2)u(k-2) + B(k-1)u(k-1) = \\
 &= \dots
 \end{aligned}$$

We can simplify this method in the following equation:

$$x(k) = \phi(k, 0)x(0) + \sum_{l=1}^k \phi(k, l)B(l-1)u(l-1)$$

Where we define as the transition state function:

$$\phi(k, l) = A(k-1)A(k-2)\dots A(l)$$

With  $k > l$ ,  $\phi(k, k) = I$

We can find this useful for a special continuous time case, where  $\dot{x}(t) = a(t)x(t)$ ,  $x(0) = x_0$  and  $a(t)$  is a scalar:

$$\phi(t, 0) = 1 + \int_{\sigma_1=0}^t a(\sigma_1)d\sigma_1 + \int_{\sigma_1=0}^t \int_{\sigma_2=0}^{\sigma_1} a(\sigma_1)a(\sigma_2)d\sigma_2d\sigma_1 + \dots$$

This is known as the Peano Beaker series.

If we define the following  $\gamma(t)$ :

$$\gamma(t) = \int_{\sigma_1=0}^t a(\sigma_1)d\sigma_1$$

We can make further useful observations:

$$\phi(t, 0) = 1 + \gamma(t) + \frac{1}{2!}\gamma^2(t) + \frac{1}{3!}\gamma^3(t) + \dots = e^{\gamma(t)}$$

And thus we find that the solution to the differential equation  $\dot{x}(t) = a(t)x(t)$  is:

$$x(t) = e^{\int_{\sigma_1=0}^t a(\sigma_1)d\sigma_1} x_0$$

If we want to solve a general case, we can do so:

$$\dot{x} = A(t)x(t) + f(t)$$

Changing the coordinates of A so that it is 0, and integrating both sides, we will have as a solution:

$$x(t) = \phi(t, t_0)x_0 + \int_{\sigma=t_0}^t \phi(t, \sigma)f(\sigma)d\sigma$$

Which is also known as the variation of constants formula. This formula will be the basis for obtaining the controllability and observability gramians, which will help us establish whether a system is controllable and observable respectively.

## 8 Matlab Codes

### 8.1 Code for the differential equations defining the system

Coauthor: Enrique Gutiérrez

With this code we pretend to solve the differential equations that govern the behavior of our inverted pendulum with the functions `dsolve` and `diff`. It is important to note though that the system is unstable, so we will not obtain an explicit solution. It is through the application of a control that we will stabilize the system and be able to reach a solution.

```
1 %% Variables
2
3 F=0.1; % Force applied to the base [N]
4 mc=0.5; % Mass of the base [kg]
5 mp=0.1; % Mass of the pendulum [kg]
6 l=0.2; % Length of the pendulum [m]
7 g=9.81; % Gravity
8
9 % Initial conditions for the ODEs
10 pos_carro_ini=0;
11 Dpos_carro_ini=0;
12 ang_ini=0;
13 Dang_ini=0;
14
15 %% Equations related to free body diagrams (FBD)
16
17 % FBD base
18 % F-Tsin(ang)=mc*der2pos_carro
19 % N + T*cos(ang) - mc*g=0
20
21 % FBD pendulum
22 % T*sin(ang)=mp*a_pend_x
23 % -T*cos(ang) - mp*g= mp*a_pend_y
24
25 %% Equations that define the movement
26
27 % -mp*g*sin(ang)=mp*der2pos_carro*cos(ang) - l*der2ang*mp
28 % F + mp*l*der2ang*cos(ang) - mp*l*(der1ang)^2*sin(ang)=(mp+mc)*
    der2pos_carro
29
30 syms pos_carro(t) ang(t)
31 Dpos_carro=diff(pos_carro)
32 Dang=diff(ang)
33
```

```

34 ode1 = diff(pos_carro, t, 2) == (F+mp*l*Dang^2*sin(ang)-mp*g*cos(ang)*
    sin(ang))/(mp+mc-mp*(cos(ang))^2);
35 ode2 = diff(ang, t, 2) == (g*sin(ang)*(mp+mc)-mp*l*Dang^2*sin(ang)*cos(
    ang)-F*cos(ang))/(l*(mp+mc)-mp*l*(cos(ang))^2);
36 odes = [ode1; ode2]
37
38 cond1 = pos_carro(0)== pos_carro_ini;
39 cond2 = Dpos_carro(0)== Dpos_carro_ini;
40 cond3 = ang(pi)== ang_ini;
41 cond4 = Dang(0)== Dang_ini;
42 conds = [cond1 cond2 cond3 cond4];
43
44 S(t)=dsolve(odes, conds);

```

Listing 2: System Ecuations

## 8.2 Code for original functions and linearized functions

Coauthor: Enrique Gutiérrez

Simple PD control applied to the system to stabilize it, both to the system after linearizing it and to the original non linear system. Figures displaying the results of this program can be found in subsection 2.1 "State Space Model".

```
1  clc
2  clear all
3  global Fvec
4  Fvec=[];
5
6  %% Initial conditions for the ODEs
7  x_ini=0;
8  v_ini=0;
9  ang_ini=10;
10 ang_ini=ang_ini*pi/180;
11 w_ini=0;
12
13 %% Ecuaciones
14 tspan = [0 20];
15
16 cond1 = x_ini;
17 cond2 = ang_ini;
18 cond3 = v_ini;
19 cond4 = w_ini;
20 conds = [cond1 cond2 cond3 cond4];
21
22 %% odefunlin , with linearized equations
23
24 [t1, y1]=ode45(@odefunlin, tspan, conds);
25 x1=y1(:,1);
26 ang1=y1(:,2);
27 v1=y1(:,3);
28 w1=y1(:,4);
29
30 figure(1)
31 subplot(211),plot(t1,ang1*180/pi), ylabel('Angle of the pendulum [deg]')
    ,title('Linearized functions')
32 subplot(212),plot(t1,w1), ylabel('Velocity of the pendulum [rad/s]')
33 xlabel('Time [s]')
34
35 figure(2)
36 subplot(211),plot(t1,x1), ylabel('Position of the base [m]') ,title('
    Linearized functions')
```

```

37 subplot(212), plot(t1, v1), ylabel('Velocity of the base [m/s]')
38 xlabel('Time [s]')
39
40 figure(3)
41 th_target=0;
42 w_target=0;
43 v_target=0;
44 kp=2;
45 kd=0.3;
46 F1=-kp*(th_target-ang1)-kd*(w_target-w1)-kd*(v_target-v1);
47 plot(t1, F1), title('Linearized functions'), ylabel('Force applied to the
    base [N]'), xlabel('Time [s]');
48
49 %% odefun, with original equations
50
51 [t, y]=ode45(@odefun, tspan, conds);
52 x=y(:,1);
53 ang=y(:,2);
54 v=y(:,3);
55 w=y(:,4);
56
57 figure(4)
58 subplot(211), plot(t, ang*180/pi), ylabel('Angle of the pendulum [deg]'),
    title('Original functions')
59 subplot(212), plot(t, w), ylabel('Velocity of the pendulum [rad/s]')
60 xlabel('Time [s]')
61
62 figure(5)
63 subplot(211), plot(t, x), ylabel('Position of the base [m]'), title('
    Original functions')
64 subplot(212), plot(t, v), ylabel('Velocity of the base [m/s]')
65 xlabel('Time [s]')
66
67 figure(6)
68 th_target=0;
69 w_target=0;
70 kp=20;
71 kd=0.2;
72 F=-kp*(th_target-y(:,2))-kd*(w_target-y(:,4))-kd*(v_target-y(3));
73 plot(t, F), title('Original functions'), ylabel('Force applied to the
    base [N]'), xlabel('Time [s]');

```

Listing 3: Original VS Linearized functions

Helper function @odefunlin used in the code found in subsection 7.2 "Code for original functions and linearized functions".

```
1 function dydt = odefunlin(t,y)
2     global Fvec
3
4     M=0.5; % Mass of the base [kg]
5     m=0.1; % Mass of the pendulum [kg]
6     l=0.2; % Length of the pendulum [m]
7     g=9.81; % Gravity
8
9     % y(1) = position of the base
10    % y(2) = angle of the pendulum
11    % y(3) = velocity of the base
12    % y(4) = angular velocity of the pendulum
13
14    th_target=0;
15    w_target=0;
16    x_target=0;
17    v_target=0;
18    kp=20;
19    kd=0.2;
20    F=-kp*(th_target-y(2))-kd*(w_target-y(4))-kd*(v_target-y(3));
21    Fvec=[Fvec;F];
22
23    dydt = zeros(4,1); % Initialize the vector
24    dydt(1) = y(3);
25    dydt(2) = y(4);
26    dydt(3) = (F-m*g*l*y(2))/(M);
27    dydt(4) = (-F+g*y(2)*(m+M))/(l*M);
28 end
```

Listing 4: Helper function for linearized functions

Helper function @odefun used in the code found in subsection 7.2 "Code for original functions and linearized functions"

```

1 function dydt = odefun(t,y)
2     global Fvec
3
4     M=0.5; % Mass of the base [kg]
5     m=0.1; % Mass of the pendulum [kg]
6     l=0.2; % Length of the pendulum [m]
7     g=9.81; % Gravity
8
9     % y(1) = position of the base
10    % y(2) = angle of the pendulum
11    % y(3) = velocity of the base
12    % y(4) = angular velocity of the pendulum
13
14    th_target=0;
15    w_target=0;
16    x_target=0;
17    v_target=0;
18    kp=2;
19    kd=0.3;
20    F=-kp*(th_target-y(2))-kd*(w_target-y(4))-kd*(v_target-y(3));
21    Fvec=[Fvec;F];
22
23    dydt = zeros(4,1);
24    dydt(1) = y(3);
25    dydt(2) = y(4);
26    dydt(3) = (F-m*l*y(4)^2*sin(y(2))+m*g*cos(y(2))*sin(y(2)))/(m+M-m*(
27    cos(y(2)))^2);
28    dydt(4) = (m*l*y(4)^2*sin(y(2))*cos(y(2))-F*cos(y(2))-g*sin(y(2))*(m
29    +M))/(m*l*(cos(y(2)))^2-l*(m+M));
30
31 end

```

Listing 5: Helper function for original functions



### 8.3 Study of the observability and controllability of a system

Simple application of Matlab built-in functions to declare whether a system is controllable and observable.

```
1 clear all
2 clc
3
4 M=0.5; % Mass of the base [kg]
5 m=0.1; % Mass of the pendulum [kg]
6 L=0.2; % Length of the pendulum [m]
7 g=9.81; % Gravity
8
9 % X=[x ; ang; v; w];
10 % dX=[v; w; ac_base; ac_ang];
11
12 % dX=AX+Bu
13 % y=CX+Du
14
15 A=[0 0 1 0; 0 0 0 1; 0 m*g/M 0 0; 0 g*(m+M)/(L*M) 0 0];
16 B=[0 0 1/M 1/(L*M)].';
17 C=[1 0 0 0;0 1 0 0];
18 D=[0: 0];
19
20 sys_ss=ss(A, B, C, D)
21 tf_sys=tf(sys_ss)
22
23 %% Controllability
24 Co=[B A*B A^2*B A^3*B]
25 contr=rank(ctrb(sys_ss))
26
27 %% Observability
28 O=[C; C*A; C*A^2; C*A^3]
29 obser=rank(observ(sys_ss))
```

Listing 6: Observability and controllability of a system

## 8.4 Control through pole placement and a LQR regulator

Application of pole placement technique and LQR regulator to the inverted pendulum system.

```
1 clear all
2 clc
3
4 M=0.5; % Mass of the base [kg]
5 m=0.1; % Mass of the pendulum [kg]
6 L=0.2; % Length of the pendulum [m]
7 g=9.81; % Gravity
8
9 % X=[x ; ang; v; w];
10 % dX=[v; w; ac_base; ac_ang];
11
12 % dX=AX+Bu
13 % y=CX+Du
14
15 A=[0 0 1 0; 0 0 0 1; 0 m*g/M 0 0; 0 g*(m+M)/(L*M) 0 0];
16 B=[0 0 1/M 1/(L*M)].';
17 C=[1 0 0 0; 0 1 0 0];
18 D=[0: 0];
19
20 %% LQR REGULATOR
21
22 %Q=C'*C;
23 Q=[5000 0 0 0; 0 300 0 0; 0 0 0 0; 0 0 0 0];
24 N=0;
25 R=1;
26
27 % LQR Regulator
28 [K, S, e]=lqr(A,B,Q,R,N);
29 Ac=A-B*K;
30 t=0:0.1:5;
31 r=0.2*ones(size(t));
32 [y, x]=lsim(Ac, B, C, D, r, t);
33 figure(1)
34 plot(t, y)
35 title('LQR Regulator')
36 xlabel('Time [s]')
37 legend('Position of the base', 'Angle of the pendulum')
38
39 %% POLE PLACEMENT
40
41 % p=polos deseados
42 %p=[-5, -2.1, -3.2, -4.3];
```

```
43 p=[-40, -16.8, -25.6, -34.4];
44
45 K=place(A, B, p)
46 Ac=A-B*K;
47 t=0:0.1:5;
48 r=0.2*ones(size(t));
49 [y, x]=lsim(Ac, B, C, D, r, t);
50 figure(2)
51 plot(t, y)
52 title('Pole Placement')
53 xlabel('Time [s]')
54 legend('Position of the base', 'Angle of the pendulum')
```

Listing 7: Pole Placement, LQR regulator



## 9 Arduino Codes

### 9.1 Code for setting a constant velocity in brushed DC motor

Coauthor: Enrique

In this program we use a basic strategy of augmenting power if the velocity of the motor is below the target velocity and reducing power if it is above the target.

```
1
2 double w=0;
3
4 void setup() {
5     pinMode(2,INPUT); // Pin to the encoder
6     pinMode(7,OUTPUT); // Pin to the motor
7     Serial.begin(9600);
8 }
9
10 int ct=0, pp=80;
11 int target=2200, one_sided_margin=100;
12
13 void loop() {
14     power(7,pp,1000000);
15     if (w>target+one_sided_margin)
16         pp=pp-1;
17     if (w<target-one_sided_margin)
18         pp=pp+1;
19 }
20
21 int factor=3;
22
23 // void function, no devuelve nada
24 void power(int pin, int p1, unsigned long t){
25
26     char printout[80];
27     // a = encoder lecture
28     // a0 = value of reference
29     // ct = count
30     int a=0, a0=0, ct=0;
31     int p;
32
33     if (p1>0)
34         p=int(p1/4+10);
35     else
36         p=0;
37     unsigned long t0=micros();
```

```

38  while((micros()-t0)<t){
39      digitalWrite(pin,1);
40      delayMicroseconds(p*factor);
41      digitalWrite(pin,0);
42      delayMicroseconds((100-p)*factor);
43      a=digitalRead(2); // read pin 2, either HIGH or LOW
44      if (a>a0) // a changes depending on the lecture from pin 2
45          ct++; // counts rising flanks
46      a0=a; // save new value
47  }
48
49  unsigned long t1=micros();
50  w=double(ct)/12.0*60000000.0/double(t1-t0); //12 slots in the
    encoder, 60k miliseconds in 1 minute; w=r/t
51  Serial.println(ct);
52  sprintf(printout,"Power: %d   Speed: %d   Target: [%d, %d] ",p
    *2,int(w),target-one_sided_margin, target+one_sided_margin);
53  Serial.println(printout);
54  }

```

Listing 8: Code for setting a constant velocity in a brushed DC motor

## 9.2 Code for implementing a constant velocity in brushed DC motor through a PID control

Coauthor: Enrique Gutiérrez

Application of a PID control to the system. The proportionate action acts with the error only. The integral action works with the sum of the errors calculated during the running of the program. The derivative action works with the difference of errors between loop and loop.

```
1 #define PIN_MOTOR 7
2 #define PIN_ENCODER 2
3
4 void setup() {
5     pinMode(PIN_ENCODER,INPUT);
6     pinMode(PIN_MOTOR,OUTPUT);
7     Serial.begin(9600);
8 }
9
10 float w=0;
11 int ct=0;
12 float pp=40.0;
13 float w_target=2200;
14 float cambio_pp=0, err=0, errSum=0;
15
16 float Input, lastInput;
17 float dErr=0, lastErr=0;
18 unsigned long now=0, lastTime=0;
19 int sampleTime=1000;
20 int i=0;
21
22 float Kp=0.015, Ki=0.03, Kd=0.015;
23
24 void loop() {
25
26     power(PIN_MOTOR, pp, 25000);
27     now=millis();
28     float timeChange=float(now-lastTime)/1000.0;
29
30     err=w_target-w;
31     errSum+=err;
32
33     //float dInput=(Input-lastInput)/timeChange;
34     float dErr=(err-lastErr)/timeChange;
35
36     if (i > 1) {
37         cambio_pp= err*Kp - Ki*errSum*timeChange + Kd*dErr;
```

```

38     //cambio_pp=err*Kp - Ki*errSum*timeChange - Kd*dInput;
39     pp+=cambio_pp;
40 }
41
42 i+=1;
43 //lastInput=Input;
44 lastErr=err;
45 lastTime=now;
46 }
47
48 int factor=3.0;
49
50 void power(int pin, float p1, unsigned long t){
51
52     char printout[80];
53     int a=0, a0=0, ct=0;
54     float p;
55
56     if (p1>0)
57         p=float(p1/4.0+10.0);
58     else
59         p=0;
60     unsigned long t0=micros();
61     while((micros()-t0)<t){
62         digitalWrite(pin,1);
63         delayMicroseconds(p*float(factor));
64         digitalWrite(pin,0);
65         delayMicroseconds((100.0-p)*float(factor));
66         a=digitalRead(PIN_ENCODER);
67         if (a>a0)
68             ct++;
69         a0=a;
70     }
71
72     unsigned long t1=micros();
73     float timeChange=t1-t0;
74
75     w=float(ct)/12.0*60000000.0/timeChange;
76
77     Serial.println(String(t1)+' '+String(w));
78     //Serial.println(String(t1)+' '+String(p*2)+' '+String(w)+' '+
79     String(w_target)+' '+String(1.1*w_target)+' '+String(0.9*
80     w_target));//-one_sided_margin)+' '+String(w_target+
81     one_sided_margin));
82 }

```

Listing 9: Code for setting a constant velocity in a brushed DC motor through a PID control



### 9.3 Code for calibrating the accelerometers

This program reads the range of the accelerometers, the maximum and minimum values the accelerometers can read. Those values should be equivalent to g and -g depending on whether they are looking down or up respectively.

```
1 #define PIN_ACCELEROMETER_X A0
2 #define PIN_ACCELEROMETER_Y A1
3
4 void setup() {
5     //analogReference(EXTERNAL);
6     Serial.begin(9600);
7 }
8
9 float xRawMin = 300;
10 float xRawMax = 300;
11 float yRawMin = 300;
12 float yRawMax = 300;
13
14 // Take multiple samples to reduce noise
15 const int sampleSize = 10;
16
17 void loop() {
18
19     int xRaw = ReadAxis(PIN_ACCELEROMETER_X);
20     int yRaw = ReadAxis(PIN_ACCELEROMETER_Y);
21     Calibrar(xRaw, yRaw);
22
23     Serial.print("Raw Ranges: X: ");
24     Serial.print(xRawMin);
25     Serial.print(" -");
26     Serial.print(xRawMax);
27
28     Serial.print(", Y: ");
29     Serial.print(yRawMin);
30     Serial.print(" -");
31     Serial.print(yRawMax);
32 }
33
34 int ReadAxis(int axisPin)
35 {
36
37     long reading = 0;
38     analogRead(axisPin);
39     delay(1);
40     for (int i = 0; i < sampleSize; i++)
41     {
```

```
42     reading += analogRead(axisPin);
43   }
44   return reading/sampleSize;
45 }
46
47 void Calibrar(int xRaw, int yRaw){
48
49   if (xRaw<xRawMin){
50     xRawMin=xRaw;
51   }
52   if (xRaw>xRawMax){
53     xRawMax=xRaw;
54   }
55   if (yRaw<yRawMin){
56     yRawMin=yRaw;
57   }
58   if (yRaw>yRawMax){
59     yRawMax=yRaw;
60   }
61 }
62 }
```

Listing 10: Code for calibrating the accelerometers

## 9.4 Code for reading from the accelerometers and calculating $\theta$

Code for transforming the readings from the accelerometers into accelerations, and use these values to calculate the angle  $\theta$ .

```
1
2 #define xmax 410
3 #define xmin 271
4 #define ymax 409
5 #define ymin 269
6 #define g 9.81
7 #define pi 3.1415
8
9 float x0, y0;
10 void setup() {
11     // put your setup code here, to run once:
12     Serial.begin(9600);
13     x0=(xmax+xmin)/2;
14     y0=(ymax+ymin)/2;
15 }
16
17 String cad, cad2;
18 float ax, ay;
19 float ang;
20
21 void loop() {
22     // put your main code here, to run repeatedly:
23     ax=float(ReadAxis(0)-x0)/float(xmax-xmin)*2*g;
24     ay=float(ReadAxis(1)-y0)/float(ymax-ymin)*2*g;
25     cad=String(ax)+' '+String(ay);
26     //Serial.println(cad);
27     delay(10);
28     ang=angulo();
29     //Serial.println(angulo()*180/pi);
30
31 }
32
33 float mod_a;
34 float a1;
35 float cos_ang;
36 float sin_ang;
37 float tan_ang;
38 int sampleSize = 10;
39
40 int ReadAxis(int axisPin)
```

```

41 {
42
43     long reading = 0;
44     analogRead(axisPin);
45     delay(1);
46     for (int i = 0; i < sampleSize; i++)
47     {
48         reading += analogRead(axisPin);
49     }
50     return float(reading/sampleSize);
51 }
52
53 float angulo(){
54     float ang;
55     mod_a=sq(ax)+sq(ay);
56     a1=(mod_a-sq(9.75));
57
58     cad2= "mod_a: "+String(sqrt(mod_a))+" ax: "+String(ax)+" ay: "+
59         String(ay)+" g: "+String(g)+" ahor: "+String(a1);
60     Serial.println(cad2);
61
62     float cos_ang=(ax*a1+ay*g)/mod_a;
63     float sin_ang=(-ay*a1+ax*g)/mod_a;
64     float tan_ang=sin_ang/cos_ang;
65     ang=atan(tan_ang);
66
67     // char printout[80];
68
69     return ang;
70 }

```

Listing 11: Code for reading from the accelerometers and calculating  $\theta$

## References

- [Alo10] José Luis Beltrán Alonso. *Simulación de un péndulo invertido*. 2010.
- [Aut19a] Several Authors. “Control Digital notes (Cap. 1: Introducción al control digital)”. In: (2019). ICAI.
- [Aut19b] Several Authors. “Tutorial 8 - Week 13: Cart-Pole Inverted Pendulum”. In: (2019).
- [Aut] Several Authors. *Inverted Pendulum: State-Space Methods for Controller Design*. Control Tutorials for Matlab and Simulink. URL: <http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlStateSpace> (visited on 06/01/2019).
- [Dja18] Theodore Djaferis. “Dynamic System Theory notes”. In: (2018). Boston University.
- [Fri86] Bernard Friedland. *Control System Design: An Introduction to State-Space Methods*. Dover, 1986.
- [Gar+16] José García et al. *Las leyes de Newton en el modelado y control del péndulo invertido sobre un carro*. 2016.
- [Lla16] Luis Llamas. *Medir la inclinación con IMU, Arduino y filtro complementario*. 2016. URL: <https://www.luisllamas.es/medir-la-inclinacion-imu-arduino-filtro-complementario/> (visited on 06/01/2019).
- [Mac+18] Juan Luis Zamora Macho et al. “Conceptos Básicos y Acciones de Control PID”. In: (2018).
- [Sir+04] Hebertt Sira-Ramírez et al. *Control de Sistemas No Lineales*. 2004. URL: [https://www.researchgate.net/profile/Hebertt\\_Sira-Ramirez/publication/327929276\\_Control\\_de\\_Sistemas\\_No\\_Lineales\\_Linealizacion\\_aproximada\\_extendida\\_exacta/links/5c009b67a6fdcc1b8d4a9371/Control-de-Sistemas-No-Lineales-Linealizacion-aproximada-extendida-exacta.pdf](https://www.researchgate.net/profile/Hebertt_Sira-Ramirez/publication/327929276_Control_de_Sistemas_No_Lineales_Linealizacion_aproximada_extendida_exacta/links/5c009b67a6fdcc1b8d4a9371/Control-de-Sistemas-No-Lineales-Linealizacion-aproximada-extendida-exacta.pdf) (visited on 06/10/2019).



## Listings

1	Treating Arduino data . . . . .	38
2	System Ecuations . . . . .	51
3	Original VS Linearized functions . . . . .	53
4	Helper function for linearized functions . . . . .	55
5	Helper function for original functions . . . . .	56
6	Observability and controllability of a system . . . . .	57
7	Pole Placement, LQR regulator . . . . .	58
8	Code for setting a constant velocity in a brushed DC motor . . . . .	61
9	Code for setting a constant velocity in a brushed DC motor through a PID control . . . . .	63
10	Code for calibrating the accelerometers . . . . .	65
11	Code for reading from the accelerometers and calculating $\theta$ . . . . .	67