UNIVERSIDADE da MADEIRA

Math and Engineering Department

BALANCE-BOT

Presented to Universidade da Madeira for the Degree of Master

Victor Vicente Abreu nº 2031802

November 2009, Funchal

This Page Intentionally Left Blank

UNIVERSIDADE da MADEIRA

Math and Engineering Department

BALANCE-BOT

Presented to Universidade da Madeira for the Degree of Master

Victor Vicente Abreu nº 2031802

Supervisor: Ian Oakley

November 2009, Funchal

ABSTRACT

Research on inverted pendulum has gained momentum over the last decade on a number of robotic laboratories over the world; due to its unstable proprieties is a good example for control engineers to verify a control theory. To verify that the pendulum can balance we can make some simulations using a closed-loop controller method such as the linear quadratic regulator or the proportional–integral–derivative method.

Also the idea of robotic teleoperation is gaining ground. Controlling a robot at a distance and doing that precisely. However, designing the tool to takes the best benefit of the human skills while keeping the error minimal is interesting, and due to the fact that the inverted pendulum is an unstable system it makes a compelling test case for exploring dynamic teleoperation.

Therefore this thesis focuses on the construction of a two-wheel inverted pendulum robot, which sensor we can use to do that, how they must be integrated in the system and how we can use a human to control an inverted pendulum.

The inverted pendulum robot developed employs technology like sensors, actuators and controllers.

This Master thesis starts by presenting an introduction to inverted pendulums and some information about related areas such as control theory. It continues by describing related work in this area. Then we describe the mathematical model of a two-wheel inverted pendulum and a simulation made in Matlab. We also focus in the construction of this type of robot and its working theory. Because this is a mobile robot we address the theme of the teleoperation and finally this thesis finishes with a general conclusion and ideas of future work.

KEYWORDS: Two-wheel inverted pendulum, Arduino, accelerometer, gyroscope, kalman filter, PID, teleoperation.

ACKNOWLEDGMENTS

This Master Thesis has been done with technical and moral support from several persons. I would like to thank all of them.

I owe a great debt to Ian Oakley who has been very supportive throughout this thesis. And is help with all my questions and concerns.

My brother for the support, encouragement and company spent over the years.

Eng. Filipe Santos for all technical help and for the equipment.

My parents for their love and moral support when I most needed it.

My relatives for the moral support.

People who supported me when it was needed, it includes friends and colleagues.

Thanks to all of you!

CONTENT

FIGURES

CHAPTER 1 - INTRODUCTION

1.1. OVERVIEW

When we think of a pendulum we think of a ball at the end of a wire suspended from a pivot point, well an inverted pendulum is exactly the opposite. The inverted pendulum is a pendulum with is mass above the pivot point, by the simple fact that its mass is above the pivot point the system is unstable by nature. Due to the fact that it is unstable it's a very good example in control engineering to verify a control theory.

Inverted pendulum provides a good model example for a rocket or missile guidance, an automatic aircraft landing system, aircraft stabilization in the turbulent air-flow, stabilization of a cabin in a ship, and so forth.

A two-wheeled inverted pendulum robot is a robot that simulates the behavior of a inverted pendulum, i.e., we aim to build a robot that can self balance on two wheels only by reading and understanding the data provided by a sensor and act on the two-wheel according.

Due to the unstable nature of the pendulum it's an example of the use of control theory.

Control theory is a discipline that deals with the behavior of unstable systems. The desired output of a system is called reference, when a system as multiples inputs it need a controller to act on them so that its possible obtain the desire output of the system.

A controller, in control theory, is a device that observe and manipulates the inputs of a system so that we get the desire output. For example, the sensors of an automatic door (controller) that will open the door when a person approaches. In this case the controller is called an open-loop controller because isn't concerned about any unexpected forces that act on the system. To solve this problem, control theory introduces feedback. A close-loop controller uses feedback to control the output of a dynamical system.

Common closed-loop controller architectures include the PID controller and the LQR method.

PID controller means proportional–integral–derivative controller and attempts to correct the error between measured process variable, i.e., the variable that we need to control, and a desired setpoint, the value that the variable must reach or obtain. It's achieved through calculating a corrective action to adjust the process accordingly and rapidly in order to keep the error minimal. It’s the sum of the tree terms; the proportional term that changes the output value according to the current error, the integral term is proportional to the magnitude of the error and the duration of the error, and the derivate term of the PID calculate the error over the time.

The linear quadratic regulator (LQR) is a well know method to determine the feedback gains of a dynamic system. It’s assumed that we have an optimal full-state feedback, i.e. that we can measure all of our states.

When we talk about robotic we need to consider which type of function we need to have, what type of data are we interest in collect and to what purpose. In an inverted pendulum robot we are interested in measure the tilt of the pendulum, the tilt will tell us to which side and if the robot is falling.

To do that we can use different types of sensors such as; light sensor, infrared sensors, accelerometer or gyroscope.

Light sensors and infrared sensors have the disadvantage of being subject to the lighting conditions. In order to operate successfully we need to control those conditions precisely with little margin for error, in the other side the accelerometer and gyroscope don’t have that problem because they don’t act on the light but with the gravity of the earth.

An accelerometer is used to measure the acceleration that a body experiences relative to freefall, with that we can determine when and to which side the body is falling. Accelerometers are used to measure vehicle acceleration; machinery health monitoring; measure the motion and vibration of a structure; measure the depth of CPR chest compressions; in navigation to calculate the position, orientation, and velocity (direction and speed of movement) of a moving object; detect apogee in both professional and in amateur rocketry and more recently phones contain accelerometers for user interface control.

An angular rate sensor or gyroscope is a device for measure or maintaining direction based on the principles of angular momentum. Applications of gyroscope include navigation and stabilization of flying vehicles like Radio-controlled helicopters or

UAVs. Due to higher precision, gyroscopes are also used to maintain direction in tunnel mining.

To measure the tilt of the inverted pendulum system we need to use an accelerometer that can measure the acceleration in two-axis, with these data it’s possible to determine the pendulum tilt and more precisely the angle. However we can simply use the accelerometer but if we only use an accelerometer we are subject to that the information give by the accelerometer is not the most exact as possible, for that and to get the most precise information on the tilt we associate and gyroscope, so that the information is more precise.

Data from accelerometer and gyroscope are subject to noise. The noise can be from a bad wire contact, bad reading, etc. To correct the value and minimize the noise the value need to be filtered; the most well know filter algorithm to combine accelerometer and gyroscope data is the kalman filter.

A kalman filter is an efficient recursive filter that estimates the state of a linear dynamic system from a series of noisy measurements. It is used in a wide range of engineering applications from radar to computer vision, and is an important topic in control theory and control systems engineering. (Wikipedia, Kalman filter, 2009)

In a mobile inverted pendulum motors are an important part, it’s necessary to choose the more suitable motor to the robot. The Devantech RD01 12v Drive System is composed by a MD23 motor drive module, two EMG30 gearmotors with encoders to allow controlling each motor and respective speed. Only a single 12v battery capable of supplying peak current of 6 Amps is required to power the system. Power for the logic comes from an on-board 5v regulator which is also capable of supplying up to 300mA to other circuits. The board communicates with the Arduino trough the MD23 IC2.

Arduino is an open-source physical computing platform based on a simple microcontroller board and it has a specific development environment for writing software for the board, the “Arduino IDE”. Arduino BT is a microprocessor platform incorporating a Bluetooth module that can be use to control the robot at distance.

Teleoperation is controlling devices or machine at distance. Through a manipulator a person is capable of controlling a machine at a distance, that machine will act accordingly to the impulse receive. It’s becoming increasing important in domains such as tele-surgery, exploration, bomb disposal, and work in hazardous environments. Although the intelligence of devices is increasing, many of these situations still require that a person perform complex physical tasks with a high level of skill.

This project will involve building a simple robotic system, programming its controller to automatically balance it then developing a feedback system which allows a human operator to complete this skilled task.

The robot developed uses technologies like microcontrollers (Arduino), sensors (accelerometer, gyroscope), motors encoders (MD23), and Bluetooth communication. It provides experience with basic electronics.

1.2. THESIS OUTLINE

This thesis will cover the following chapters.

Chapter 2 consists of the references to existing works that relate to this thesis and that influenced us in making prototype.

In chapter 3 we describe the modeling and simulation of a two-wheel inverted pendulum with the linear quadratic regulator method using Matlab.

Chapter 4 describes the components and the construction of a two-wheel inverted pendulum. We also discuss the main circuit of the robot and the computing structure of the program.

At chapter 5 we will describe the tests that were made and their conclusion.

In chapter 6 we describe the teleoperation and its connection with the inverted pendulum problem.

At chapter 7 will describe an overview of the contribution of this thesis, a conclusion of the thesis and possible areas for future work.

CHAPTER 2 - RELATED WORK (STATE OF THE ART)

In this chapter we mention the different types of mobile inverted pendulum robots and the different components used in their construction. We will only review the most important, their similarity and differences.

2.1. INVERTED PENDULUM

An inverted pendulum is a pendulum which has its mass above its pivot point. It is often implemented with the pivot point mounted on a cart that can move horizontally and may be called a cart and pole. Whereas a normal pendulum is stable when hanging downwards, an inverted pendulum is inherently unstable, and must be actively balanced in order to remain upright, either by applying a torque at the pivot point or by moving the pivot point horizontally as part of a feedback system.

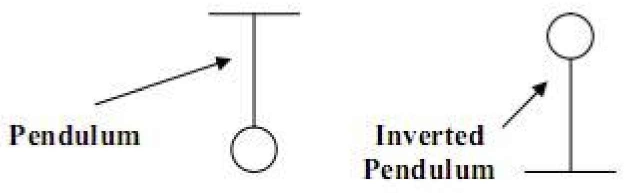


Figure 1 - Inverted Pendulum

2.2. TWO-WHEELED INVERTED PENDULUM ROBOTS

The idea of a robot that can self-balance has gained momentum over the last decade in various robot laboratories, for academic or research purpose worldwide and peoples who have robotic as a hobby.

Kazuo Yamafuji, Professor Emeritus at the University of Electro-Communications in Tokyo, built the first two-wheel inverted pendulum robot in 1986, according to an article in the Japan Times. (Times, 2001)

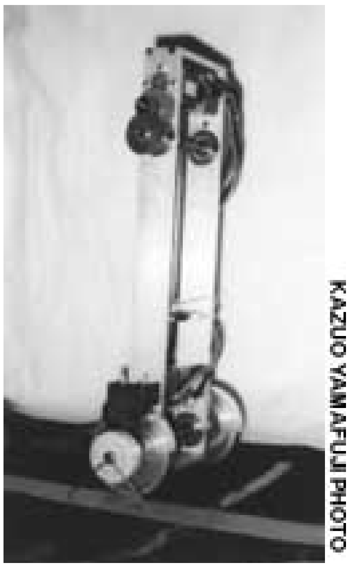


Figure 2 - Kazuo Yamafuji Robot

Basically, independent of how the robot was built, the idea is the same, built a robot that can simulate the behavior of an inverted pendulum. That can be made of different forms, we can use rolling carts attached to a platform or two wheels, different types of sensors from light sensor to accelerometers and gyroscopes and different type of controller.

2.2.1. SENSORS

Sensors are an important part of a mobile inverted pendulum robot. They will give us the angle of the pendulum.

2.2.1.1. LIGHT SENSORS

They must be positioned at front or back of the robot facing downwards so that it's possible to measure the wavelength of floor's color. With that value it's possible to measure the distance to the floor, and figure out which side the robot is falling.

Dani Piponi's Equibot uses a Sharp infrared sensor to measure the distance to the floor and uses that information to deduce tilt angle. Equibot is a balancing robot a bit like a small scale Segway. It is based around an ATMega32 RISC Microcontroller. (Piponi)

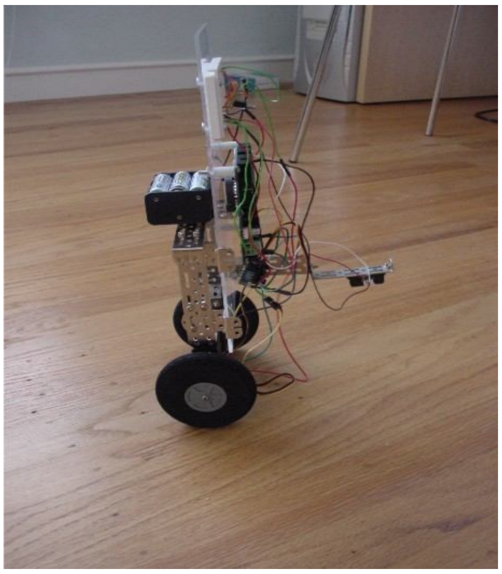


Figure 3 – Equibot

Steve Hassenplug’s LegWay is a two wheel robot that uses RCX Lego. It balances using the IR Lego sensor to detect the floor proximity. Can follow a black line and/or spin in place. (Hassenplug, 2007)

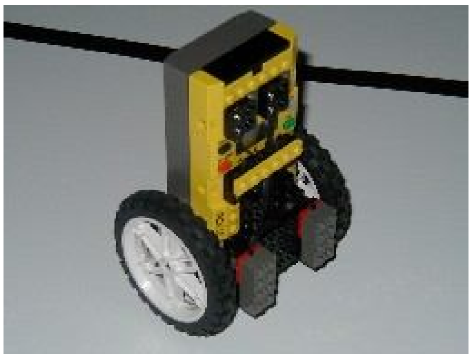


Figure 4 - Steve's LegWay

Philippe Hurbain’s NXTWay take inspiration on the LegWay, is the same idea but instead of using the RCX Lego uses NXT Lego. The basic principle is the same; it uses the light sensor to detect the proximity to the floor and balance. (Hurbain)

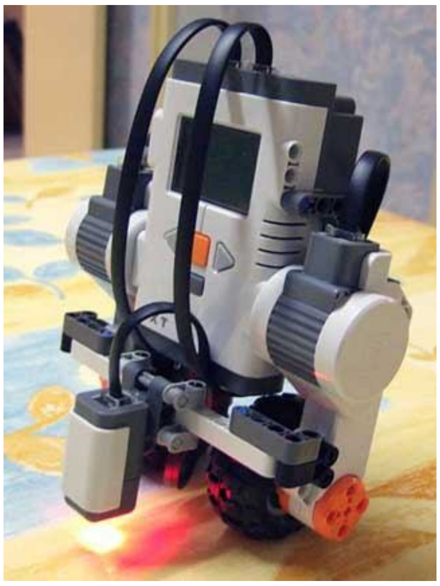


Figure 5 - Philippe Hurbain's NXTWay

2.2.1.2. ACCELEROMETER WITH GYROSCOPE

To sense the robot's angle we can use an accelerometer associated with a gyroscope. The accelerometer is responsible for measuring the proper acceleration the robot experiences relative to freefall, with this data we can determinate the angle of the robot and the gyroscope will measure the orientation that is used to correct accelerometer's angle. These sensors can be placed at any side of the robot, provided that they are aligned according to axis in which the robot falls.

The UMASS uBot is an inverted pendulum robot that uses an IMU board composed by an accelerometer and a gyroscope. The uBot is a platform that was custom build at UMASS, Amherst. The uBot was then modeled as an inverted pendulum using the State Space approach. A Linear Quadratic Regulator (LQR) was then used to solve the control problem. (Amherst, 2006)

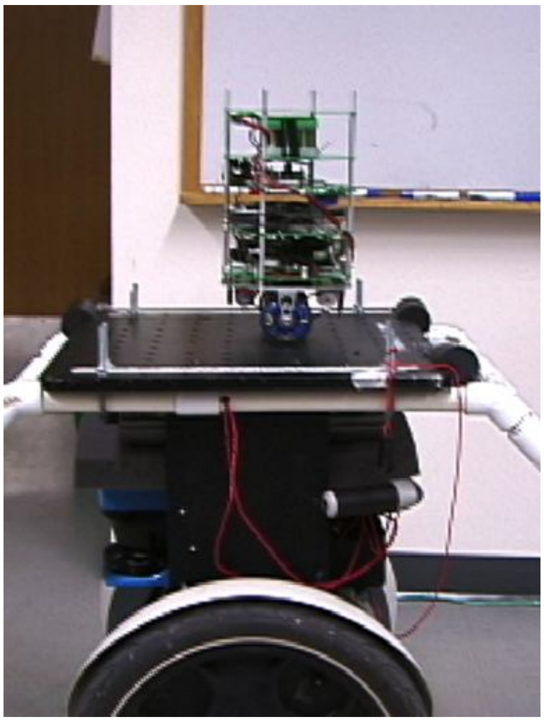


Figure 6 - UMASS uBot

Another example of an inverted pendulum that uses accelerometer and gyroscope is David P. Anderson’s nBot. The data collected from both sensors is processed by the FAS-G IMU that implements a "Weiner" filter to combine the two sensors into a single measurement.

The robot uses the HC11 robot controller developed for the M.I.T. 6.270 Robotics Course, the same robot controller used on the LegoBot and SR04. (Anderson, 2007)

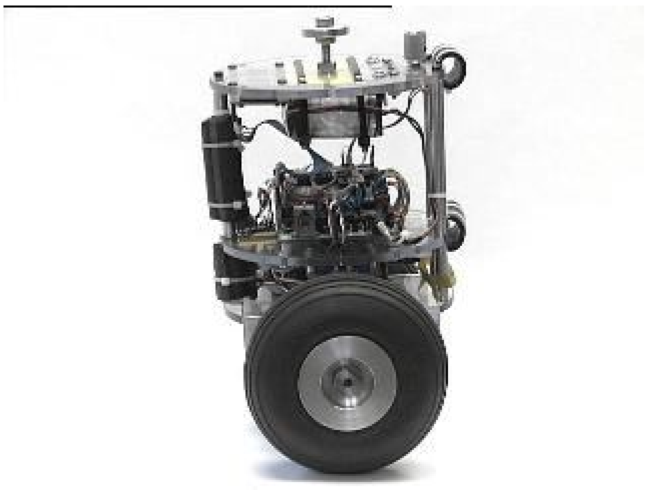


Figure 7 - David Anderson's nBot

Bender is another inverted pendulum robot to use both sensors to measure angle. Ted Larson and Bob Allen are the brains behind it. Bender is a robot that consists of several platforms to support its various components. Its “brain” is a Microchip PIC18F452.

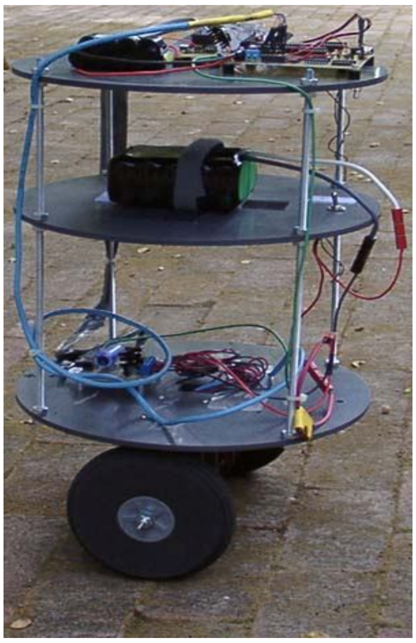


Figure 8 - Ted Larson's Bender

We also have JOE; JOE was developed by the Industrial Electronics Laboratory at the Swiss Federal Institute of Technology, Lausanne, Switzerland. Due to its configuration with two coaxial wheels, each of which is coupled to a DC motor, the vehicle is able to do stationary U-turns. A control system, made up of two decoupled state-space controllers, pilots the motors so as to keep the system in equilibrium. (Grasser, 2004)

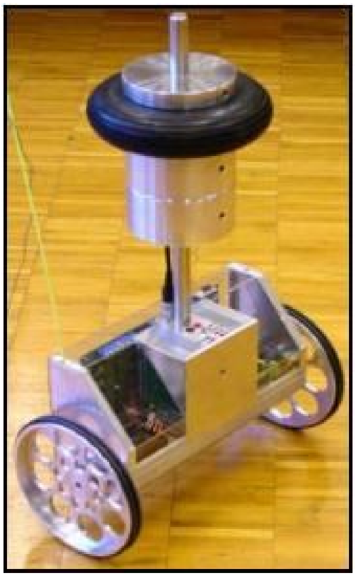


Figure 9 – JOE

Roy Watanabe’s NXTWay-G can be described like been a new version of the NXTWay but instead of using the NXT Light sensor it uses the NXT Gyro sensor. It means that instead of detect the proximity to the floor to balance; the balancing is made using the rotation detected by the gyro sensor, the number of degrees. By using this type of sensor it can eliminate the problem of the light sensor (must have a light controlled environment) and now use the body’s rotation angle. (Watanabe, 2007)

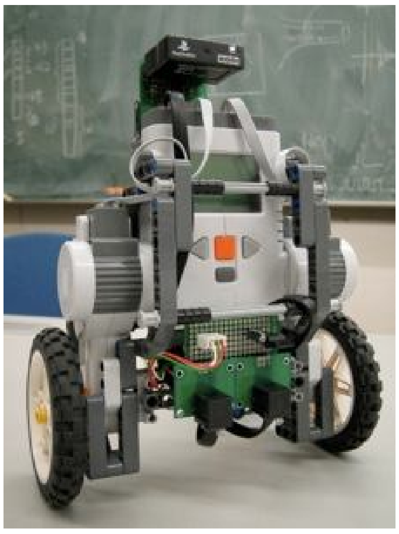


Figure 10 - Ryo Watanabe NXTWay-G

These were some examples of inverted pendulums. They serve to illustrate the different ways of construction and components of a robot capable of behaving like an inverted pendulum. Among these components we highlight the sensors used; robots that use light sensors have the drawback that it is necessary to control the lighting conditions of area the robot will operate. The smallest difference in light intensity in this spot will affect sensor’s reading and the controller may think that the robot is falling when in fact is in equilibrium position, or vice versa. Furthermore, the use of an accelerometer associated with a gyroscope, by not being associated with light but to gravity, resolves this problem and provides a better way to detect the tilt of the robot.

2.3. 2 DEGREE OF FREEDOM INVERTED PENDULUM ROBOTS

Another example of an inverted pendulum robot it’s Ballbot. The Ballbot is a mobile robot, an attempt to solve the robotic unicycle problem, and is designed to balance itself on its single spherical wheel while travelling around. It is the focus of the Ballbot Research Platform, a project conducted at Carnegie Mellon University, made possible by grants from the National Science Foundation. The robot is being developed by Ralph Hollis and George Kantor, with help from the graduate students Tom Lauwers, Anish Mampetta and Eric Shearer. The purpose of the Ballbot project is to discover how robots may maintain dynamic stability (that is, reliable balance), to enable designs with narrower bases for improved navigability (such as in a crowded room). This is a departure from the current paradigm in robot design, which relies on the static stability provided by having three or more wheels (and a much wider base). The project is titled "Dynamically-Stable Mobile Robots in Human Environments".

Ballbot balances with the aid of on-board sensors and computers and uses an "actuator mechanism based on an inverse mouse-ball drive" to move and change direction without needing to turn first.

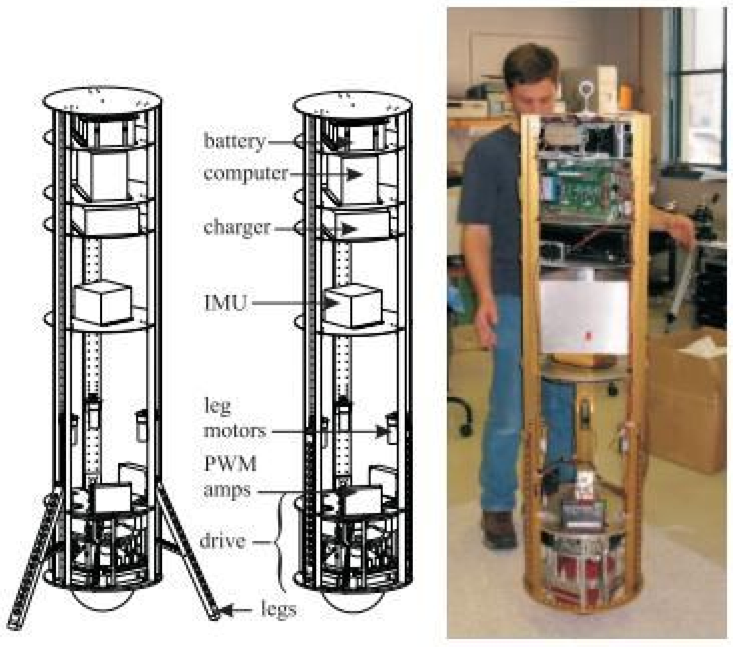


Figure 11 - Ballbot

2.4. TELEOPERATION

Teleoperation indicates operation of a machine at a distance. It is similar in meaning to the phrase "remote control" but is usually encountered in research, academic and technical environments. It is most commonly associated with robotics and mobile robots but can be applied to a whole range of circumstances in which a device or machine is operated by a person from a distance. (Wikipedia, 2009)

A telemanipulator (teleoperator or telerator) is a device that is controlled remotely by a human operator. If such a device has the ability to perform autonomous work, it is called a telerobot. If the device is completely autonomous, it is called a robot. Devices designed to allow the operator to control a robot at a distance are sometimes called telecheric robotics.

Telerobotics, is the technical term for the remote control of a robot called a telechir. In a telerobotic system, a human operator controls the movements of a telechir from some distance away. Signals are sent to the telechir to control it; other signals come back, telling the operator that the telechir has followed the instructions. These control and return signals are called telemetry.

Some teleoperated robots have a limited range of functions. A good example is a space probe, such as Voyager, hurtling past some remote planet. Earthbound scientists sent commands to Voyager based on the telemetry received from it, aiming its cameras and resolving minor problems.

Figure 12 - Voyager probe

In a more sophisticated form of teleoperation known as telepresence, the human operator has a sense of being on location, so that the experience resembles virtual reality (VR). For example, a telechir can be equipped with sensors that detect sensations of vision and sound, and in some cases pressure and texture. These sensations can then be reproduced at the operator location by means of specialized transducers. (techtarget.com, 2008)

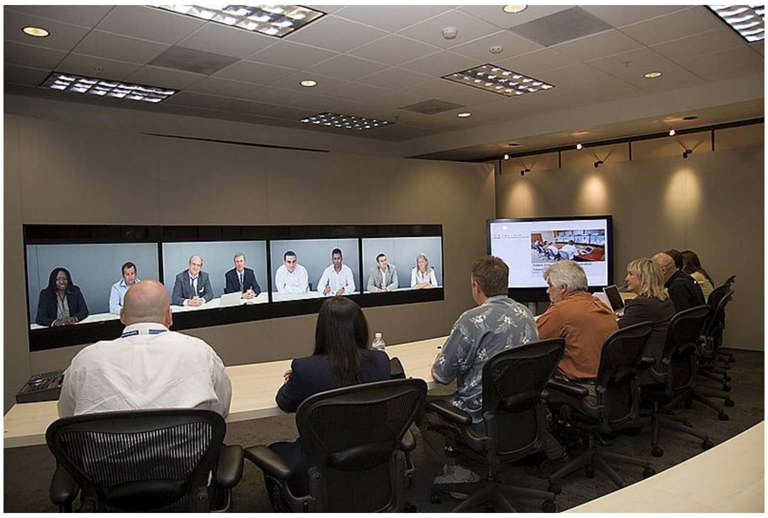


Figure 13 - Teliris Unified VirtuaLive telepresence room in use (2007)

2.4.1. TELEMANIPULATOR

The *IED detonator*, is a telemanipulator for investigating potentially explosive devices. Its goes investigate the presence of explosive and allows that the human operator is safe from any danger.

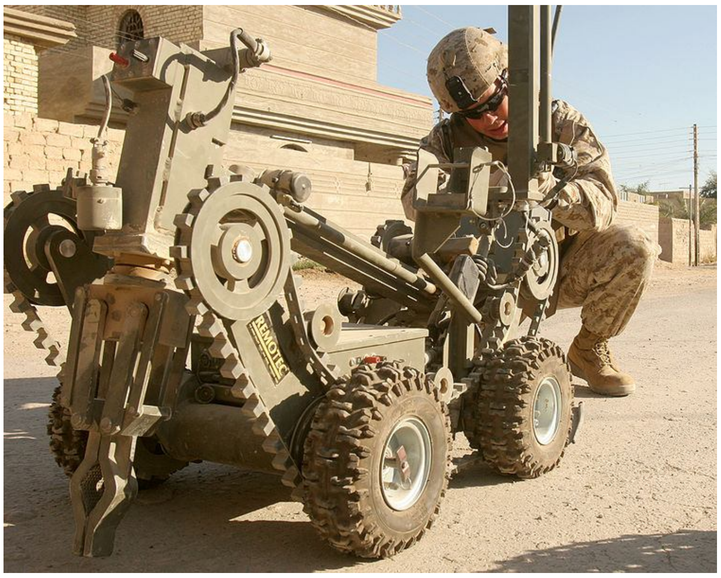


Figure 14 - IED detonator

The Da Vinci Telerobotic Surgical System permits the surgeon to perform an operation on a patient from a remote site. Currently, the FDA requires the surgeon to sit physically in the same room as the patient on whom he is operating.

The surgeon sits at the computer console. He views a virtual operative field through a binocular 3-dimensional imaging system. He sits in a comfortable ergonomically correct position with arms supported by a rest. His feet activate several peddles that control various aspects of the robot’s movements.

The surgeon’s console contains the binocular 3-dimensional imaging system. The surgeon immerses himself within a virtual operative field that is viewed through the binocular viewing finders.His arms are supported by the arm rest. Foot pedals control various adjustments of the robotic arms and instruments.

The surgeon inserts his hands into a “master” that translates motions of his hands into motions of the robotic arms and hand-like instruments. The surgeon acts as the “master” and the robot as the “slave” in this telerobotic “master-slave” system.Da Vinci only duplicate the motions of the surgeon. Da Vinci does not initiate any actions on its own volition. (Association)

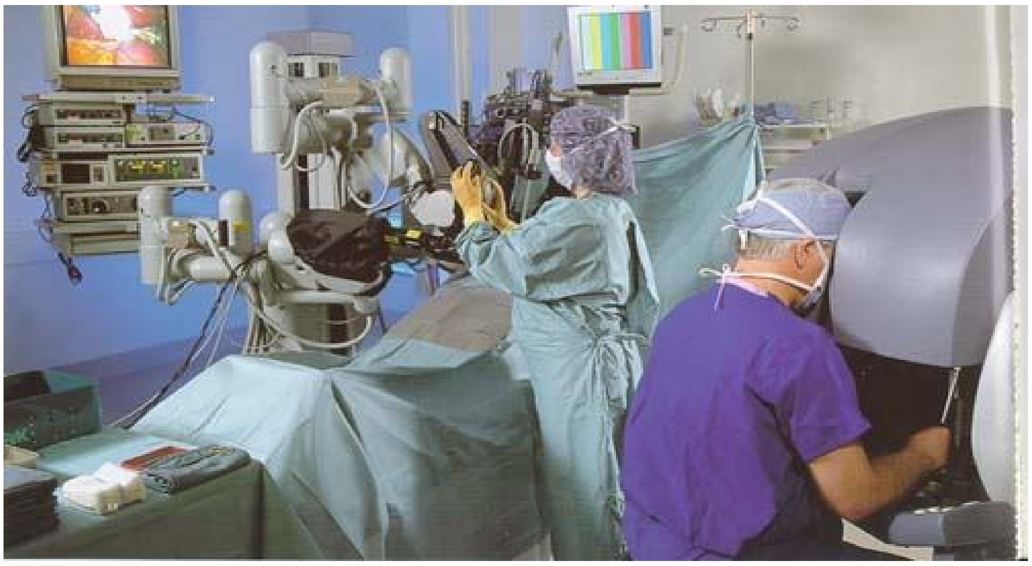


Figure 15 - DA VINCI TELEROBOTIC SURGICAL SYSTEM

Dextre (also known as the Special Purpose Dexterous Manipulator (SPDM)) is a two armed robot, or telemanipulator, which is part of the Mobile Servicing System on the International Space Station (ISS), and extends the function of this system to replace some activities otherwise requiring spacewalks. It was launched March 11, 2008 on mission STS-123. (Wikipedia, Dextre, 2009)

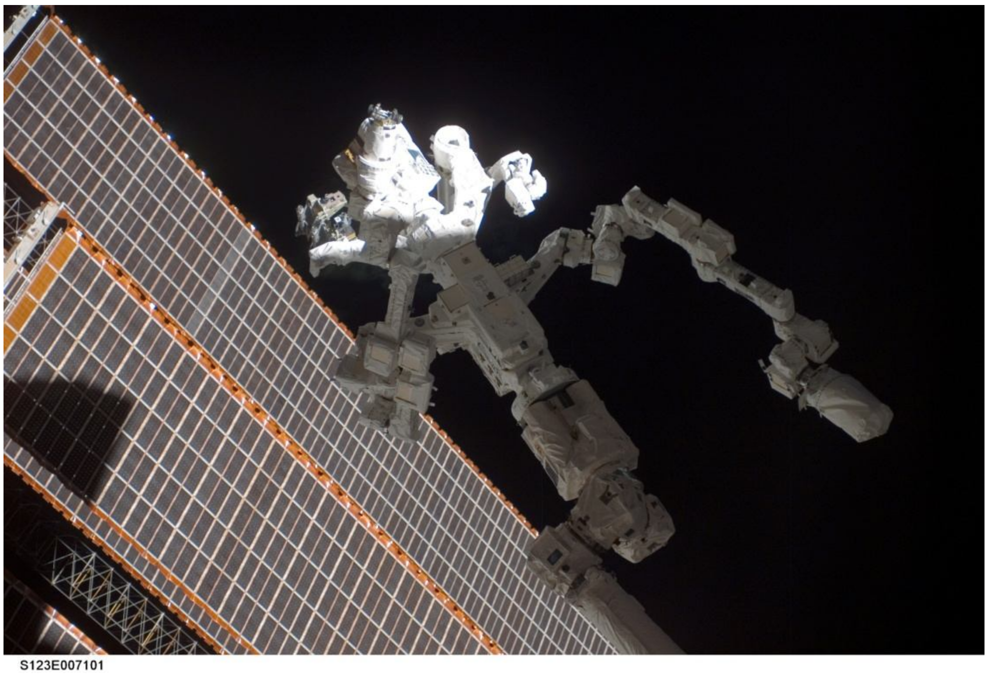


Figure 16 – Dextre

We can also make a difference between a remotely operated vehicle (ROV) and a remote control vehicle (RCV).

2.4.2. REMOTELY OPERATED VEHICLE

A remotely operated vehicle (ROV) is a tethered underwater robot. They are common in deepwater industries such as offshore hydrocarbon extraction. An ROV may sometimes be called a remotely operated underwater vehicle to distinguish it from remote control vehicles operating on land or in the air. ROVs are unoccupied, highly maneuverable and operated by a person aboard a vessel. They are linked to the ship by a tether (sometimes referred to as an umbilical cable), a group of cables that carry electrical power, video and data signals back and forth between the operator and the vehicle. High power applications will often use hydraulics in addition to electrical cabling. Most ROVs are equipped with at least a video camera and lights. Additional equipment is commonly added to expand the vehicle’s capabilities. These may include sonar, magnetometers, a still camera, a manipulator or cutting arm, water samplers, and instruments that measure water clarity, light penetration and temperature. (Wikipedia, Remotely operated underwater vehicle, 2009)

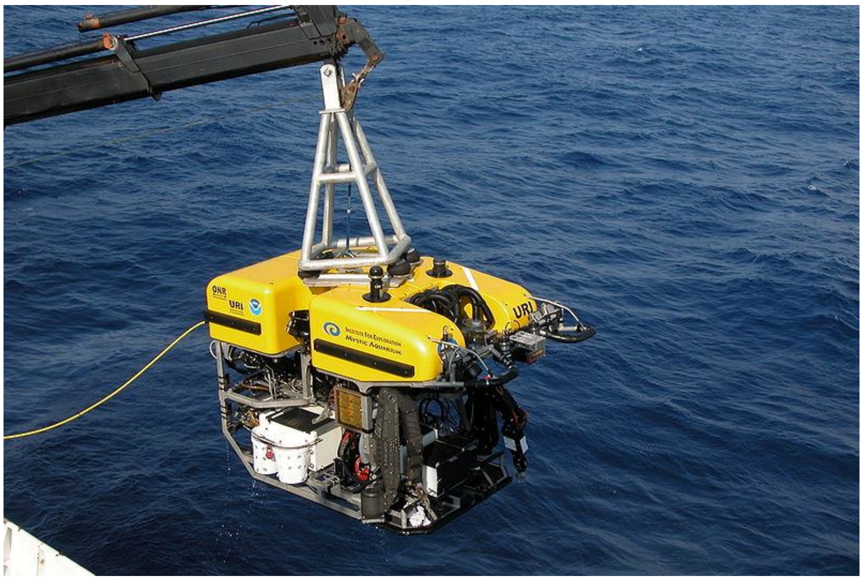


Figure 17 - ROV Hercules

2.4.2. REMOTE CONTROL VEHICLE

A remote control vehicle (RCV) is defined as any mobile device that is controlled by a means that does not restrict its motion with an origin external to the device. This is often a radio control device, cable between control and vehicle, or an infrared controller. A remote control vehicle or RCV differs from a robot in that the RCV is always controlled by a human and takes no positive action autonomously. (Wikipedia, Remote control vehicle, 2009)

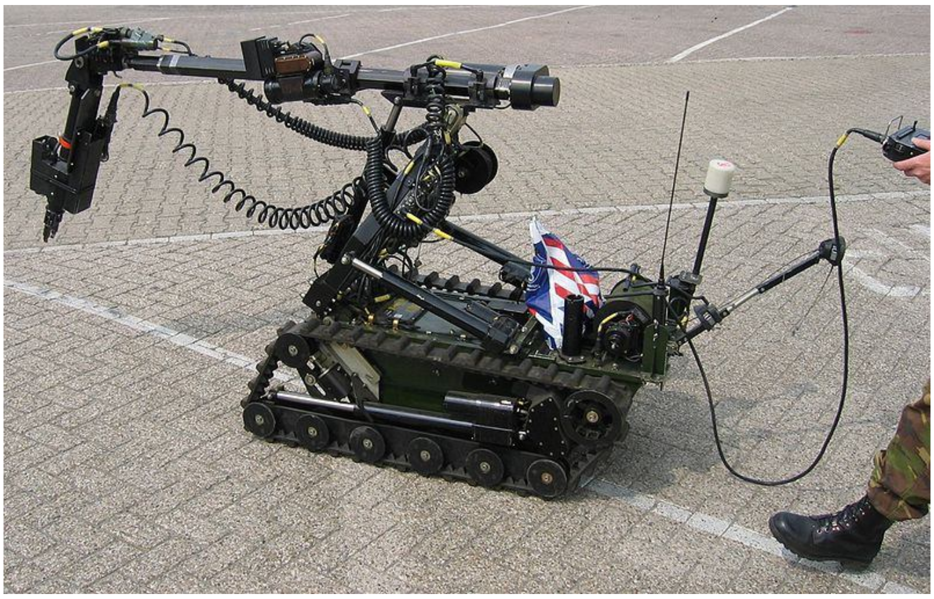


Figure 18 - Explosieven Opruimingsdienst

In this chapter we discussed the various mobile inverted pendulums; we found that we can use several types of sensors from light sensor to accelerometer combined with gyroscope, each one with its own advantage and disadvantage, the light condition influence the light sensor, and the accelerometers and gyroscope are affected by noise.

We also review a few example of robot operated by human at a distance, i.e. teleoperation. We discover that a telemanipulator is a device that is controlled remotely by a human operator.

CHAPTER 3 - ROBOT MODELING

The first important step in the construction of the robot is modeling the system. This means deducing the mathematical equations that concern to the movement of the system, the set of forces that act in the main body and in every single wheel.

3.1. NOMENCLATURE

Symbol	Parameter	Value[unit]
x_r	Straight line position	[m]
θ_p	Pitch angle	[rad]
δ_p	Yaw angle	[rad]
J_{RL}, J_{RR}	Moment of inertia of the rotating masses with respect to z axis	[kgm^2]
M_r	Mass of rotating masses connected to the left and rigth wheel. $M_{RL} = M_{RR} = M_r$	0.4[kg]
J_p	Moment of inertia of of the chassis with respect to z axis	[kgm^2]
J_δ	Moment of inertia of of the chassis with respect to y axis	[kgm^2]
M_p	Mass of body	2.1[kg]
R	Radius of wheel	0.05[m]
L	Distance between the z axis and the center of gravity of the vehicle	0.3[m]
D	Lateral distance between the contact patches of the wheel	[m]
y_r	Shift position of the wheel with respect to the y axis	[m]
x_p	Shift position of the chassis with respect to the x axis	[m]
g	Garvity constant	9.8[ms^{-2}]
C_L,C_R	Input torque for right and left wheels accordingly	0.147[Nm]

3.2. FORCE ANALISYS AND SYSTEM EQUATIONS

To develop an efficient control system for the vehicle, its dynamics have to be described by a mathematical model.

The free body diagram of the robot is described in Fig.19 that can be found at (Grasser, D'Arrigo, Colombi, & Rufer, February 2002).

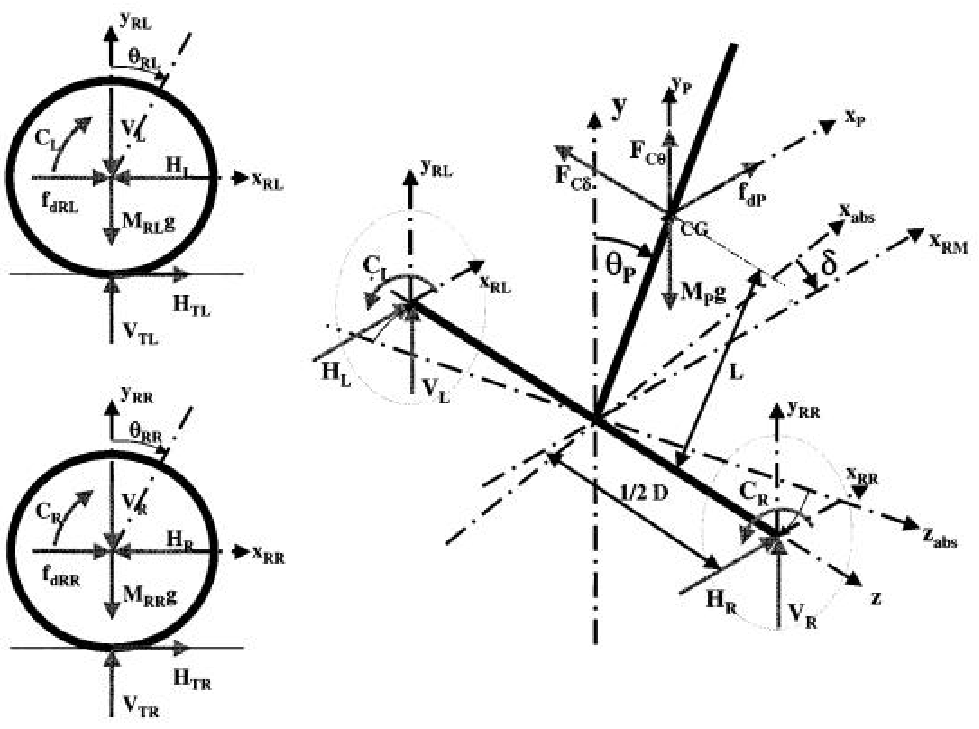


Figure 19 – Free body diagram of the robot

The following equations can be derivated from the model and can be found at (Grasser, D'Arrigo, Colombi, & Rufer, February 2002).

For left hand wheel (analog for right hand wheel).

$$\ddot{x}_{RL}M_r = H_{TL} - H_L + (f_{dRL} + f_{dRR}) \tag{1}$$

$$\ddot{y}_{RL}M_r = V_{TL} - V_L - M_r g \tag{2}$$

$$\ddot{\theta}_{RL}J_{RL} = C_L - H_{TL}R \tag{3}$$

$$\dot{x}_{RL} = R\dot{\theta}_{RL} \tag{4}$$

$$\dot{y}_p = -\dot{\theta}_p L \sin\dot{\theta}_p \tag{5}$$

$$\dot{x}_p = \dot{\theta}_L L \cos\dot{\theta}_p \tag{6}$$

$$\dot{\delta} = \frac{\dot{x}_{RL} - \dot{x}_{RR}}{2f} \tag{7}$$

For the chassis:

$$\ddot{x}_p M_p = (H_R + H_L) + f_{dP} \tag{8}$$

$$\ddot{y}_p M_p = V_R + V_L - M_p g + F_{C\theta} \tag{9}$$

$$\ddot{\theta}_p J_p = \left(V_R + V_L\right) L \sin\theta_p - \left(H_R + H_L\right) L \cos\theta_p - \left(C_L + C_R\right) \tag{10}$$

$$\ddot{\delta} J_\delta = \left(H_L - H_R\right) \frac{D}{2} \tag{11}$$

where H_L, H_R, H_{TL}, V_L, V_R, V_{TL}, and V_{TR} represent reaction forces between the different free bodies.

Equations (1)-(11) can be represented in the state-space form as:

$$\dot{x}(t) = f(x) + g(x)u \tag{12}$$

where $x \in IR^n$, $u \in IR^m$ are respectively the state and the control. f(x) is nonlinear dynamic function matrix and g(x) is nonlinear input function matrix. The state, x of the system is defined as:

$$x = \left[x_r, \dot{x}_r, \theta_p, \dot{\theta}_p, \delta, \dot{\delta}\right]' \tag{13}$$

Modifyng the equations above and then linearizing the result around the operating point (θ_p=0, x_r=0, δ=0) and de-coupling, the system's state space equations can be written in matrix form.

The state-space equations for the robot can be written as two different systems: 1) a system “pendulum” describing the rotation about the z axis and 2) a system “rotation” modeling the rotation about the y axis.

For the “pendulum” we have,

$$\begin{bmatrix} \dot{x}_r \\ \ddot{x}_r \\ \dot{\Theta}_p \\ \ddot{\Theta}_p \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & A_{23} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & A_{43} & 0 \end{bmatrix} \begin{bmatrix} x_r \\ \dot{x}_r \\ \Theta_p \\ \dot{\Theta}_p \end{bmatrix} + \begin{bmatrix} 0 \\ B_2 \\ 0 \\ B_4 \end{bmatrix} \begin{bmatrix} C_L & C_R \end{bmatrix} \tag{14}$$

and for the “rotation”,

$$\begin{bmatrix} \dot{\delta} \\ \ddot{\delta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \delta \\ \dot{\delta} \end{bmatrix} + \begin{bmatrix} 0 \\ B_6 \end{bmatrix} \left[C_L - C_R\right] \tag{15}$$

where,

$$A_{23} = g\left(1 - \frac{4}{3} L \frac{M_p}{X}\right), \quad A_{43} = \frac{g M_p}{X}$$

$$B_2 = \left(\frac{4LY}{3X} - \frac{1}{M_p L}\right), \quad B_4 = -\frac{Y}{X}, \quad B_6 = \frac{6}{(9M_r + M_p)RD}$$

and,

$$X = \frac{1}{3}\frac{M_p(M_p + 6M_r)L}{M_p + \frac{3}{2}M_r}, \qquad Y = \frac{M_p}{\left(M_p + \frac{3}{2}M_r\right)R} + \frac{1}{L}$$

The details for equations (14) and (15) are not shown here and can be found elsewhere (Nawawi, Ahmad, & Osman, Dec 2007).

We are now able to design an independent controller for each of these subsystems with the possibility of assigning different dynamics to each of them.

3.3. MATLAB ANALYSIS

In this section we will use the state-space equations, for the "pendulum" system, in the Matlab to run some simulations and see the response of the system to a determinate impulse.

We use the tutorial present at (CJC, 1997).

3.3.1. LINEAR QUADRATIC REGULATOR

The Linear Quadratic Regulator (LQR) is a well know method to determine a practical feedback gains of a system.

The main advantage of this method is that the optimal input signal u(t) is obtained from full state feedback; i.e. u = Kx for some K matrix. The feedback matrix K in question is obtained by solving the Ricatti equation associated with the particular LQR problem we have at hand.

One of the disadvantages of the LQR controller is that obtaining an analytical solution to the Ricatti equation is quite difficult in all but the simplest cases.

3.3.1.1. STATE-SPACE MODELS

The state space model represents a physical system as n first order coupled differential equations. This type of representation helps doing computer simulations and allows observe the behavior of the system to a set of inputs instead of a single input.

The general vector-matrix form of the state space model is:

$$\dot{x}(t) = A(t)x(t) + B(t)\boldsymbol{u}(\boldsymbol{t})$$

$$y(t) = C(t)x(t) + D(t)\boldsymbol{u}(\boldsymbol{t})$$

where **y** is the output equation, and **x** is the state vector.

The state-space equations for this problem are:

$$\begin{bmatrix} \dot{x}_r \\ \ddot{x}_r \\ \dot{\theta}_p \\ \ddot{\theta}_p \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & A_{23} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & A_{43} & 0 \end{bmatrix} \begin{bmatrix} x_r \\ \dot{x}_r \\ \theta_p \\ \dot{\theta}_p \end{bmatrix} + \begin{bmatrix} 0 \\ B_2 \\ 0 \\ B_4 \end{bmatrix} [C_L \quad C_R] u$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \Phi \\ \dot{\Phi} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} u$$

where,

$$A_{23} = g\left(1 - \frac{4}{3} L \frac{M_p}{X}\right), \quad A_{43} = \frac{gM_p}{X}$$

$$B_2 = \left(\frac{4LY}{3X} - \frac{1}{M_p L}\right), \quad B_4 = -\frac{Y}{X}$$

and

$$X = \frac{1}{3} \frac{M_p(M_p + 6M_r)L}{M_p + \frac{3}{2} M_r}, \qquad Y = \frac{M_p}{\left(M_p + \frac{3}{2} M_r\right)R} + \frac{1}{L}$$

This problem can be solved using full state feedback. The schematic for this type of control system is shown below:

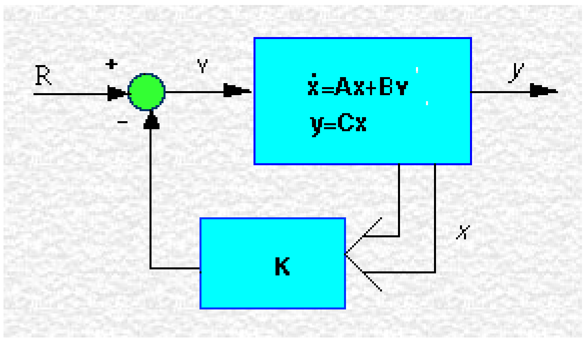


Figure 20 - Scheme for control system problems (CJC, 1997)

In this problem R represents the input that needs to be given to the wheels, i.e., the velocity of the wheels; the four states represent the position and velocity of the wheels and the angle and angular velocity of the pendulum. The output y contains the position of the wheels and the angular velocity of the pendulum.

We use it to design a controller that when an input is given to the system the pendulum should return to is balanced position and the wheel should move according.

3.3.1.2. OPEN-LOOP POLES

The first step to solve the problem is to determinate the open-loop poles of the system. For that we use the Maltab file (m-file) present in Appendix A according to the tutorial fount at (CJC, 1997).

The Matlab command window should output the following text as a result:

```
p =

          0
          0
     8.2825
    -8.2825
```

One of the poles is in the right-half plane, which means that the system is unstable in open-loop.

3.3.1.3. LINEAR QUADRATIC REGULATOR METHOD

First we need to assume that we have full-state feedback so that we can find the vector K which determines the feedback control law.

For that we enter the code from the m-file of Appendix B according to the tutorial fount at (CJC, 1997).

You should get the next value for K and a response plot:

```
K =

  316.2278   46.4863  -53.7311   -0.7813
```

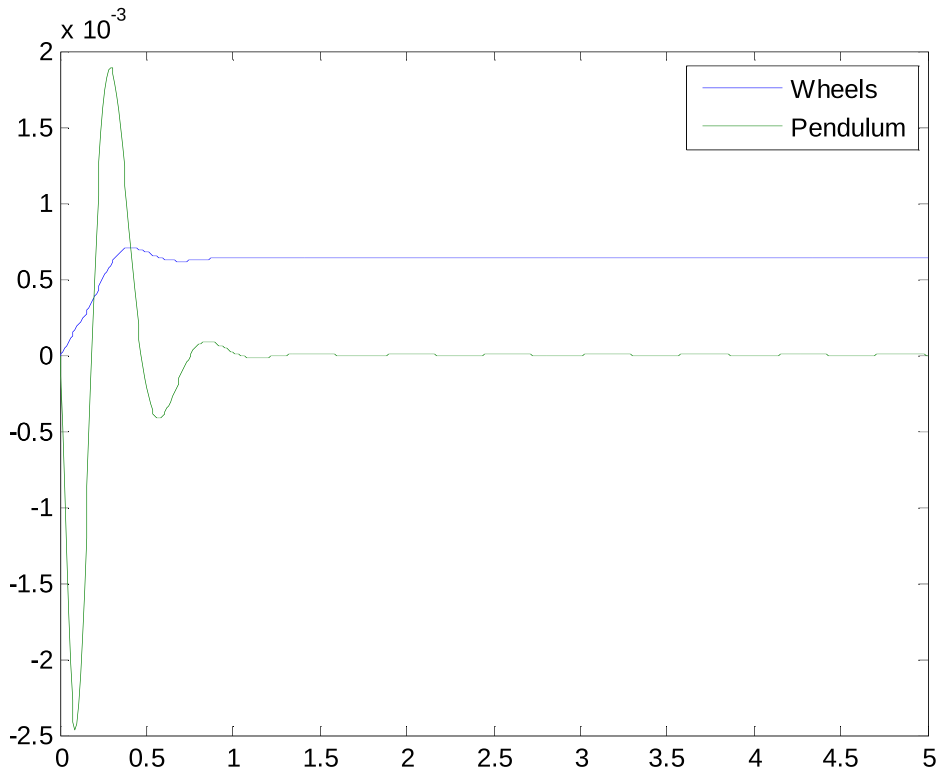


Figure 21 - Step response with LQR control

The curve in green represents the pendulum's angle, in radians and the curve in blue represents the wheel's position in meters. As you can see the wheels are not in the desire position but in the other direction.

3.3.1.4. REFERENCE INPUT

Now we want to get rid of the steady-state error. We need to compute what the steady-state value of the states should be, multiply that by the chosen gain K, and use a new value as our reference for computing the input. This can be done by adding a constant gain Nbar after the reference. The schematic below shows this relationship:

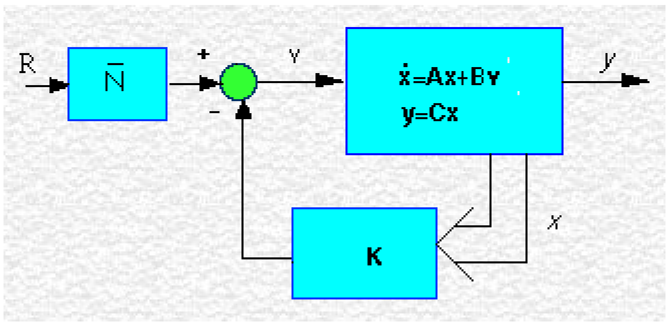


Figure 22 - Reference Input schematic (CJC, 1997)

After adding the m-file of Appendix C according to the tutorial fount at (CJC, 1997) to your Matlab simulations you should see the following response:

```
Nbar =

   316.2278
```

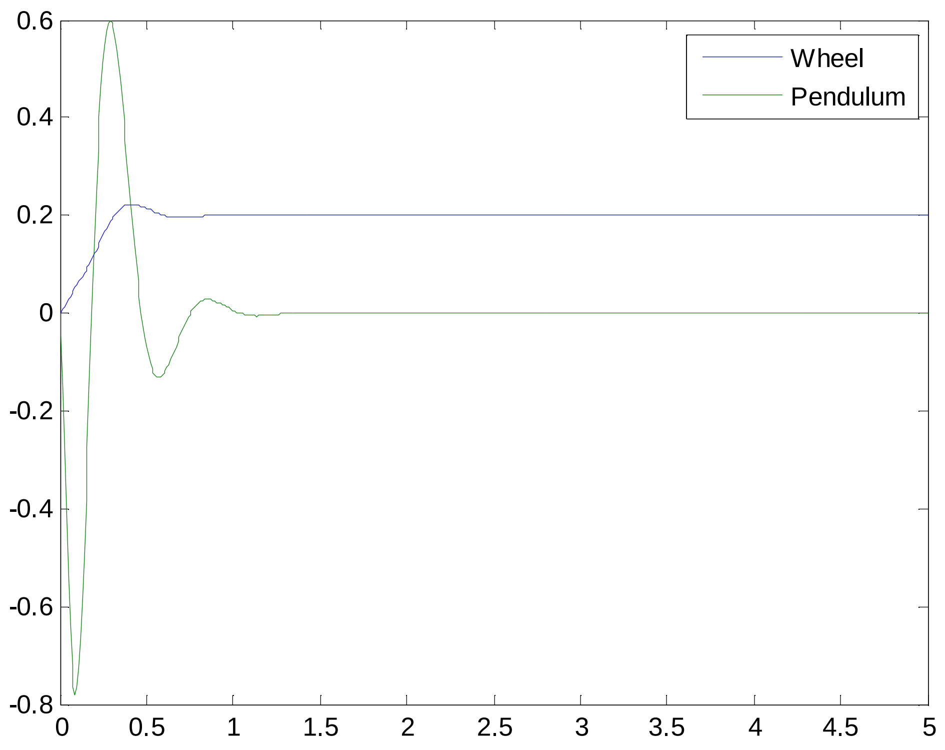


Figure 23 - Step response with LQR and Nbar control

Now, the steady-state error is within our limits, the rise and settling times are met and the pendulum's overshoot is within range of the design criteria.

3.3.1.5. OBSERVER DESIGN

This response is good, but was found assuming full-state feedback, which most likely will not be a valid assumption. To compensate for this, we will next design a full-order estimator to estimate those states that are not measured. A schematic of this kind of system is shown below, without Nbar:

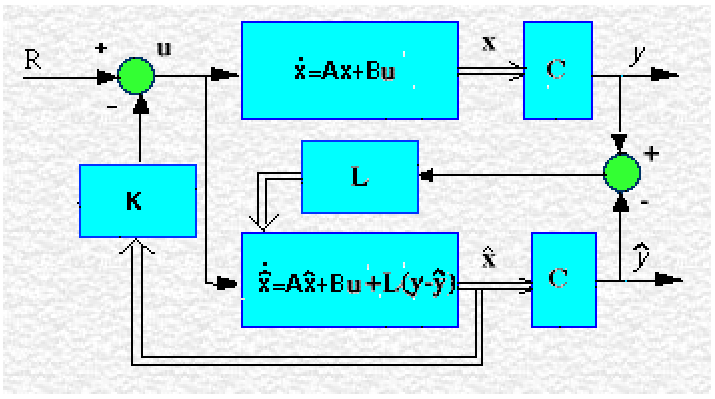


Figure 24 - Full-order estimator schematic (CJC, 1997)

After running the m-file of Appendix D according to the tutorial fount at (CJC, 1997), you should see the following step response simulation plot:

```
p =

 -17.7885 +12.4215i
 -17.7885 -12.4215i
  -5.5043 +11.1281i
  -5.5043 -11.1281i


L =

  1.0e+003  *

    0.0829     0.0011
    1.7161     0.0267
    0.0009     0.0831
    0.0380     1.7954
```

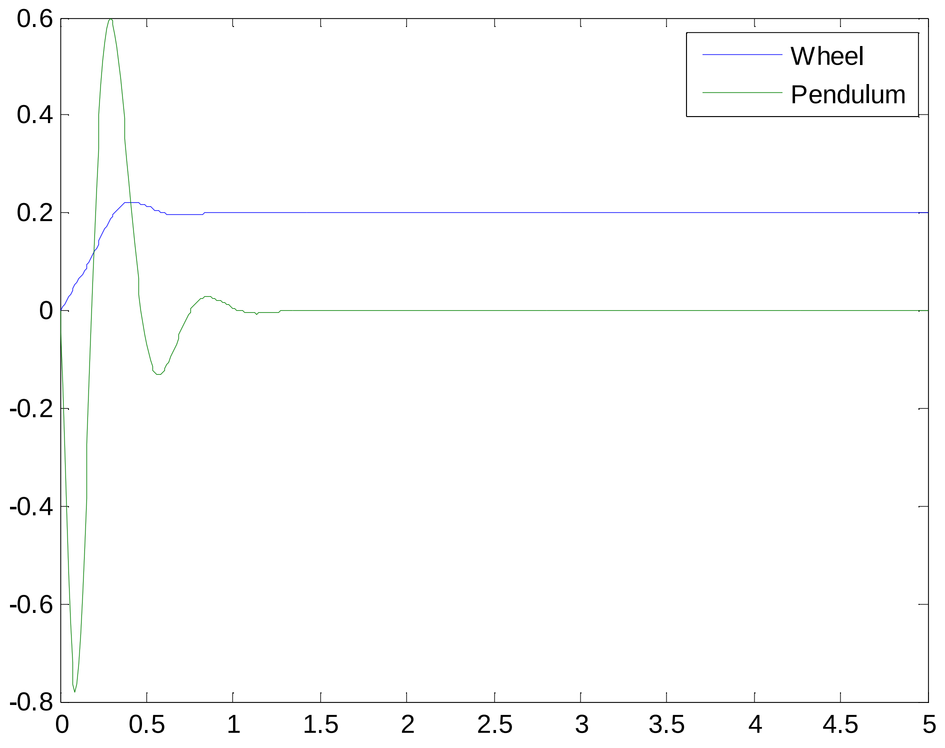


Figure 25 - Step response with estimator

This response is about the same as before. All of the design requirements have been met with the minimum amount of control effort, so no more iteration is needed.

With this simulation we can observe that our inverted pendulum can reach an equilibrium position. It helped us understand and realize the variables involved in the process and which we need to consider, in this case are the angle of the pendulum and the position of the wheels to reach that position at that time the wheels must move at a certain speed, that speed is one of the inputs of our final system.

3.4. MATLAB SIMULATION

In this section we present a few simulations that were done in Matlab so that we can understand better the behaviour of the mobile inverted pendulum robot.

3.4.1. WEIGHT

First we start by adding some extra weight to the body of the robot so that we can analyse its behaviour.

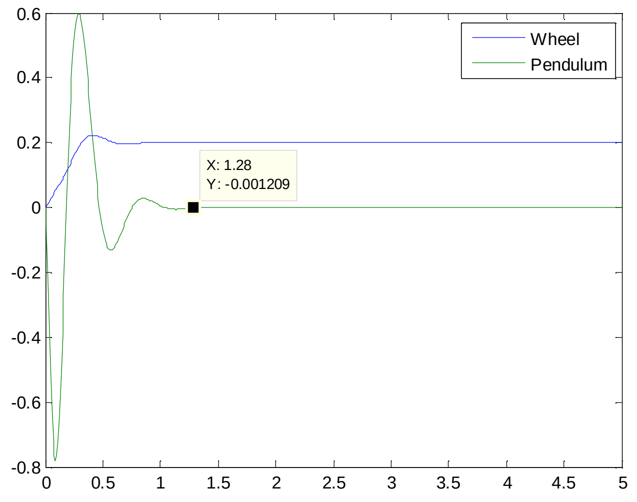


Figure 26 - Final Matlab simulation

This figure represents the final simulation of the LQR method applied to our mobile inverted pendulum robot.

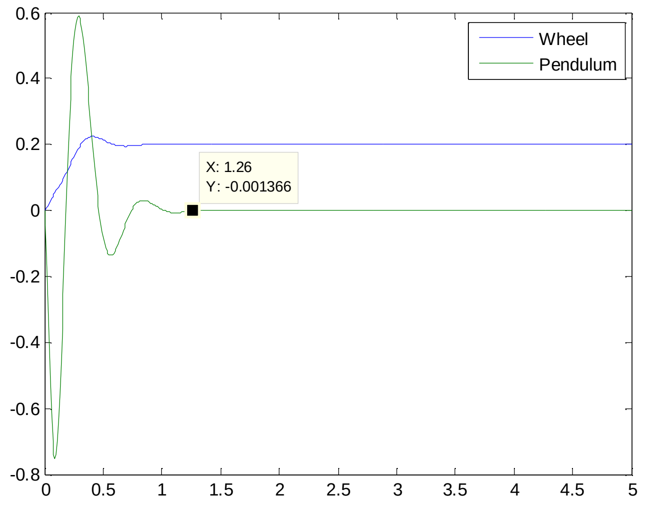


Figure 27 - Matlab simulation body mass +100g

In this simulation we add more 100g to the body. As we can see there is no big difference in pendulum rise time or in the wheel’s position. They are both around 1.26 seconds.

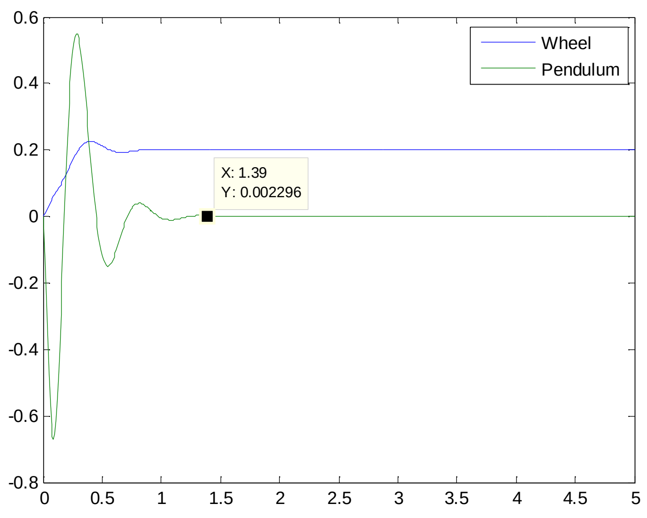


Figure 28 - Matlab simulation body mass +500g

In this simulation we add more 500g to the body. As we can see it took more time to settle the pendulum with the increase of weight. If compared with the initial graphic the difference is significative.

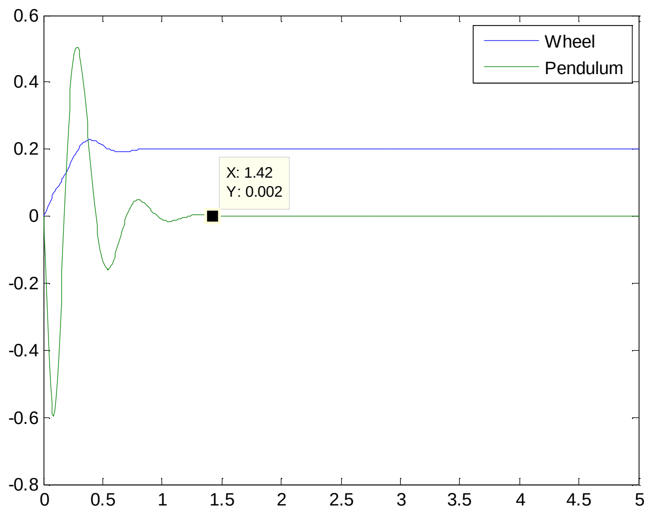


Figure 29 - Matlab simulation body mass +1000g

In this simulation we add more 1000g to the body. As we were excpecting it took more time to settle the pendulum.

In conclusion we can observe that a heavy robot will take more time to settle. By taking more time to settle the robot it means that the robot will tilt more making it more unstable. If it takes more time it will be necessary more acceleration to reach the equilibrium state.

Therefore when choosing materials for the pendulum we have to choose strong, lightweight materials.

4.3.2. CENTER OF GRAVITY

In this simulation we changed the center of gravity of the robot. By changing the robot center of gravity we are changing robot's mass distribution. With this we can see how the mass of the robot must be distributed along its body.

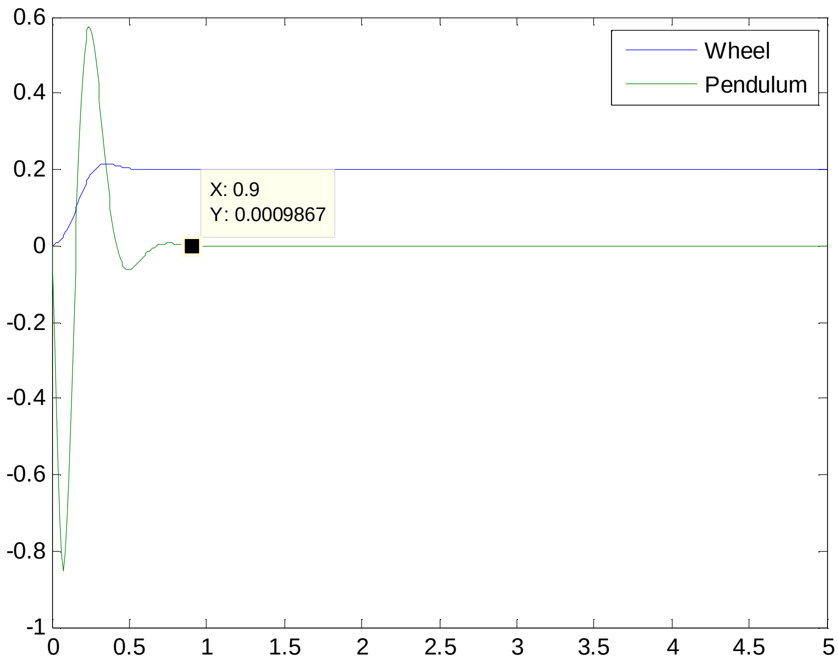


Figure 30 - 10cm lower Center of Gravity

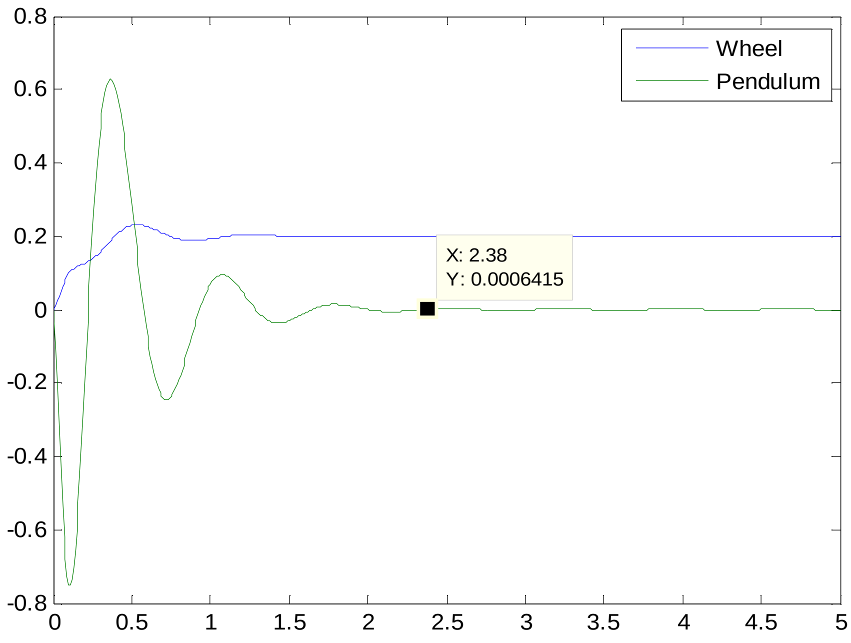


Figure 31 - 10cm upper Center of Gravity

As we can see in the figures if we lower the center of gravity of the robot we will decrease the setling time from 1.39s to 0.9s to settle the pendulum and when we increase the center of gravity the time to setle will be bigger and by the graphic we can also see that the pendulum doesn’t settle by complete.

Therefore, and by analising both tests, to build a mobile inverted pendulum robot we need to choose material not to heavy and resistant so that it doesn’t crack on impact. Also we need to optimze the tall of the robot with is center of mass. A taller robot is easier to balance but we need to place its center of gravity so that it doesn´t stay at the top of the robot, when balancing and so the robot can remain in a balanced position the wheels must be driven to stay under its center of gravity.

CHAPTER 4 - ROBOT CONSTRUCTION

In this chapter we discuss the physical construction of the robot as well as the components used in construction.

First we will discuss the components and their characteristics. And then we will talk about the real construction of the robot.

4.1. SENSORS

The sensors are responsable for given feedback to the system so that the robot can move according. Thre are two types os sensor in this projects gyroscope and accelerometer.

4.1.1. ADXL203 ACCELEROMETER

Accelerometers are used for measure the acceleration that a body suffers relatively to free fall on a determinate axis. In our case we will need an accelerometer capable of given that acceleration on two axes so that we can calculate the angle.

The ADXL103/ADXL203 are high precision, low power, complete single- and dual-axis accelerometers with signal conditioned voltage outputs, all on a single, monolithic IC. The ADXL103/ADXL203 measure acceleration with a full-scale range of ±1.7 *g*. The ADXL103/ADXL203 can measure both dynamic acceleration (for example, vibration) and static acceleration (for example, gravity). (Analog Devices, 2006)

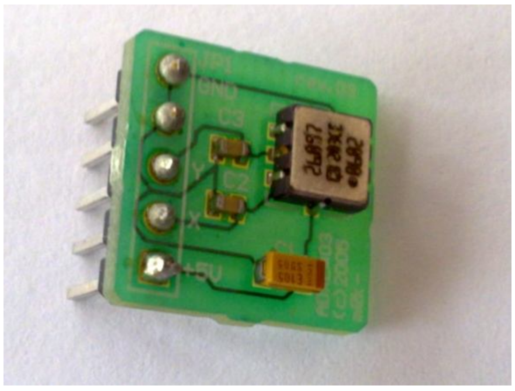


Figure 32 - Accelerometer

4.1.2. ADXRS610 - 300 DEGREE/SEC GYROSCOPE

The ADXRS610 is a complete angular rate sensor (gyroscope) that uses the Analog Devices, Inc. surface-micromachining process to create a functionally complete and low cost angular rate sensor integrated with all required electronics on one chip. (Electronics)

This gyro is meant to be use with the accelerometer. As the gyroscope is capable of measuring the rate of a single direction it's used to correct the angle given by the accelerometer.

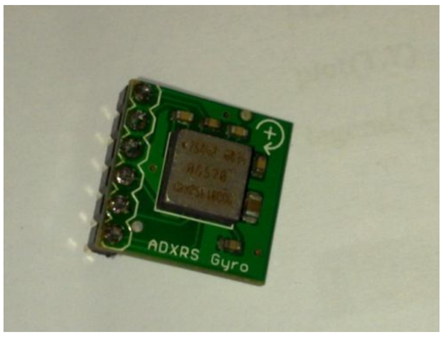


Figure 33 - Gyroscope

4.2. HARDWARE

4.2.1. ARDUINO NANO

Arduino Nano is a surface mount breadboard embedded version of the Arduino with integrated USB. It is small, complete, and breadboard friendly. The Nano was designed and is being produced by Gravitech. (A. Mellis, Arduino - ArduinoBoardNano, 2006)

Arduino is an open-source physical computing platform based on a simple microcontroller board and it has a specific development environment for writing software for the board, the "Arduino IDE".

Arduino can be used to develop interactive objects, taking inputs and controlling physical outputs. It can be connected via serial port to software on the computer or it can be stand-alone.

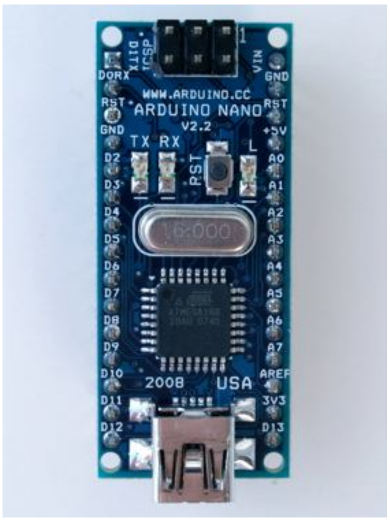


Figure 34 - Arduino Nano

4.2.2. ARDUINO BT

The Arduino BT is an Arduino board with built-in bluetooth module, allowing for wireless communication. (A. Mellis, Arduino - ArduinoBoardBluetooth, 2007)

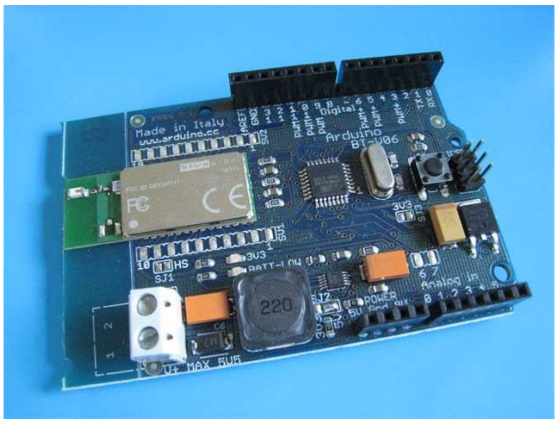


Figure 35 - Arduino BT-V06

4.2.3. DEVANTECH RD01 DRIVE SYSTEM

The Devantech RD01 Drive system is a complete Robot Drive system, ready to integrate into your robot, comprised of an MD23 motor drive module, two EMG30 gear motors with encoders, two mounting brackets and two 100mm wheels with hubs already fitted. Screws to fit the motors to the brackets and a hex key for the hub screw are included. (Robots Ltd., Active Robots - RD01 Robot Drive System - UK)

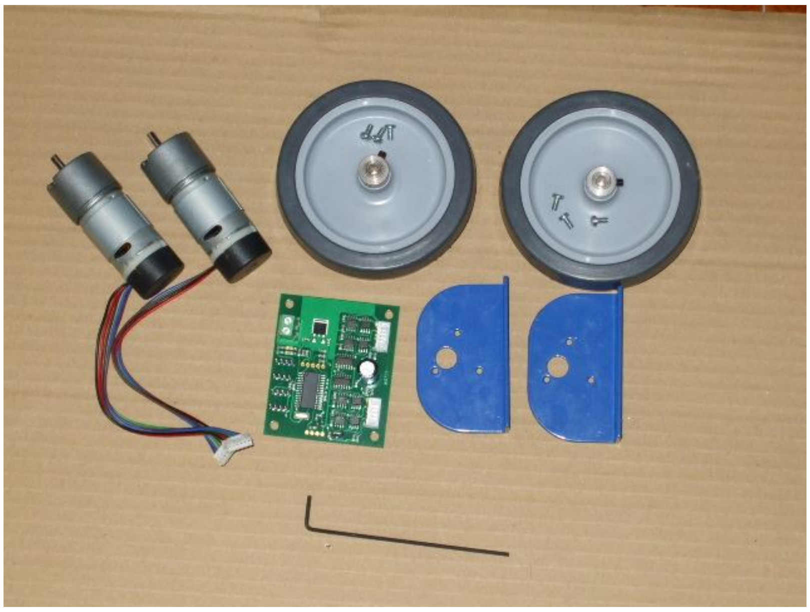


Figure 36 - Devantech RD01 Drive System set

4.2.4. BASES AND STRUCTURE

For the bases of the robot we used Perspex, this material was chosen because it's light and robust, precisely, five Perspex sheets of 15*20cm with 5mm thickness. For the main structure we used four metal rods of 50cm long and 6mm thickness.

Figure 37 - Perspex and metal rods

4.2.5. BATTERY

To power the wheels and the Arduino we used a 12V battery pack.

4.3. ROBOT’S CONSTRUCTION

From theory we know that a taller robot will be easier to balance, because it will fall slowly. However a taller robot will be more massive and there for it will need more acceleration to balance.

Therefore we build a 0.56m tall (including wheels) robot that weight 2.1kg.

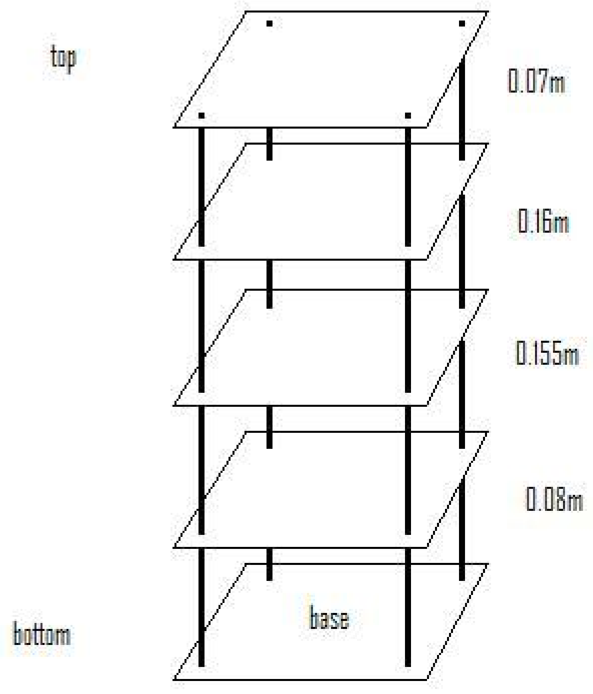


Figure 38 - Robot structure

Before the “real” construction of the robot begins, we drilled holes to hook up the metal rods in all the Perspex sheets before proceed further. We drilled 2cm away of each corner, with 6mm holes. Was also drilled a hole in the middle of the Perspex sheets for wiring.

Next we hook up the EMG30 Mounting Brackets to one of the Perspex sheets (some drilling will be necessary for the screws). This will be the base, were the motors and wheels will be hooked.

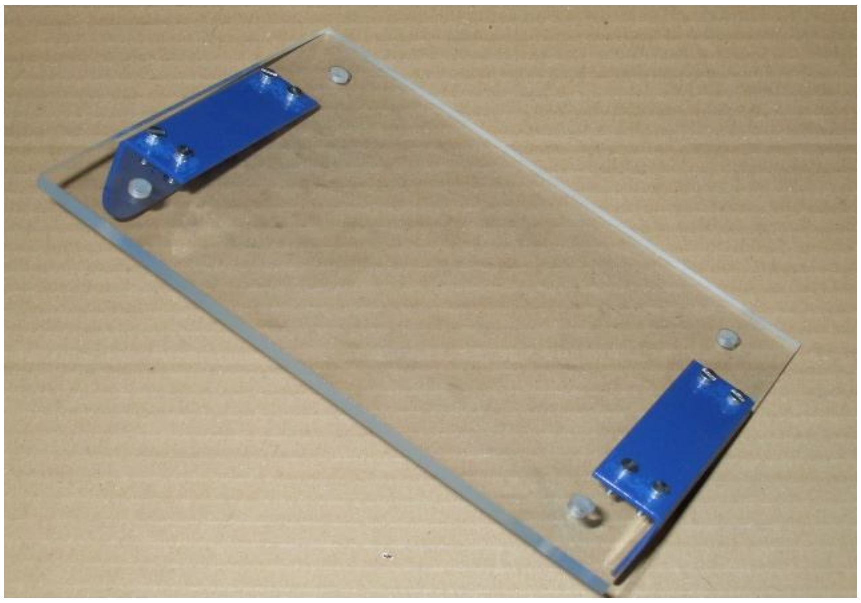


Figure 39 – Bracket montage

Then we hooked up the motors and respective wheels.

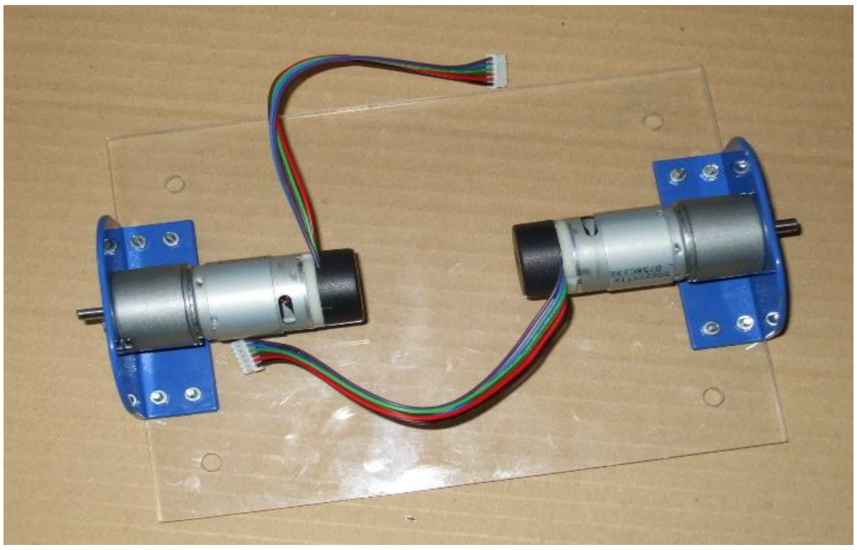


Figure 40 – Motors

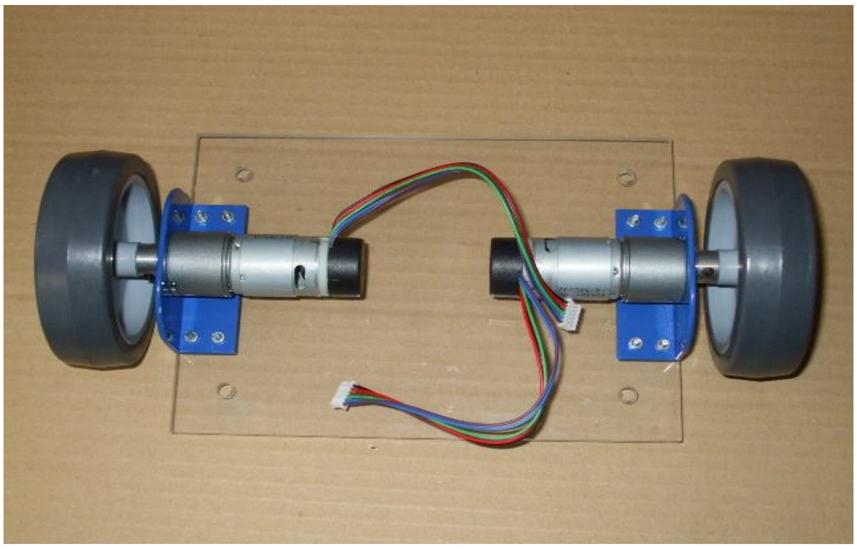


Figure 41 – Motors and wheels

After both motors and wheels are hooked up its time to build the main structure which will support the MD23 for the motors, the accelerometer, gyroscope and all the necessary battery and wires.

For that it will be necessary one bolt spanner and some nuts. Nuts of 6mm and bolt spanner of 10mm.

Figure 42 – Bolt spanner and nuts

We start by hooking up the metal rods to the "base", and then we repeat the same process for the others sheets of Perspex.

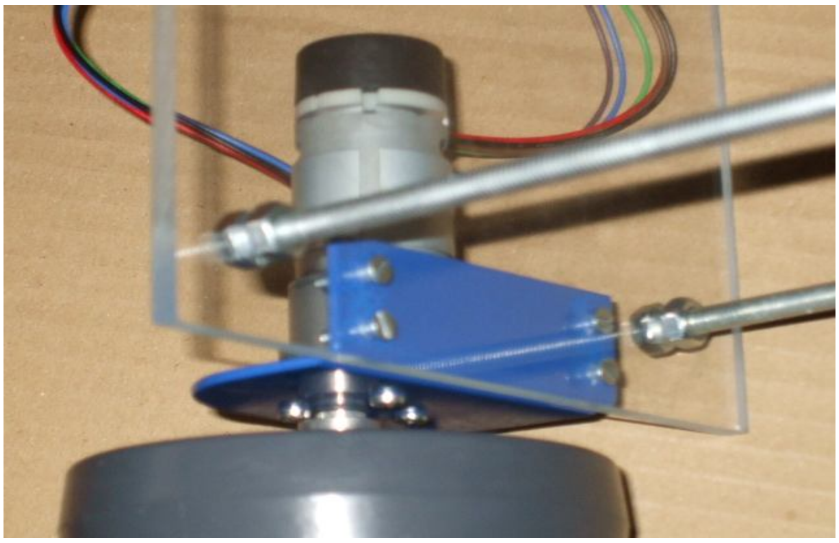


Figure 43 – First sheet of Perspex with metal rods.

After all four metal rods are attached to the main base it should be similar to the following image. We must ensure that all nuts used in the process are well tight to avoid that the robot wobbling while balancing.

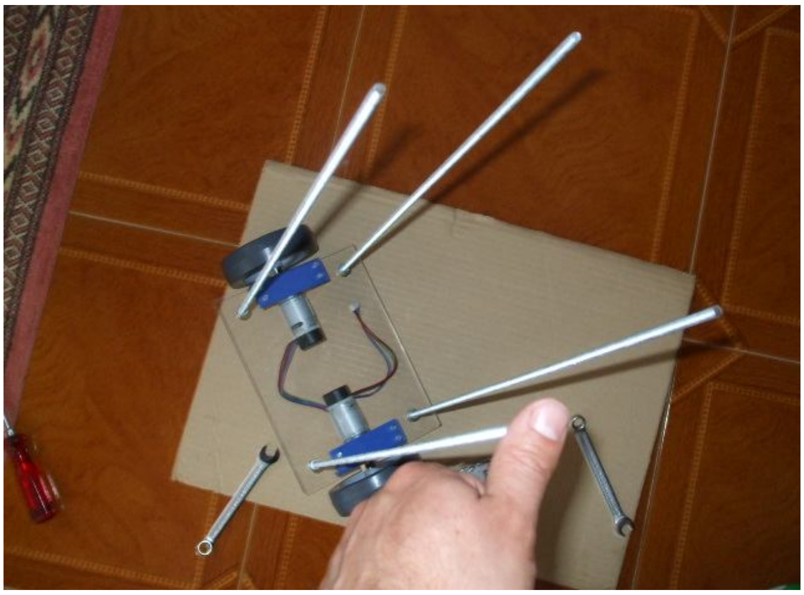


Figure 44 – Base and metal rods

The process is the same for the next four sheets. In the end the structure should be similar to the following. This will be our “body” were we will place the battery and the circuit (Arduino + accelerometer + gyroscope).

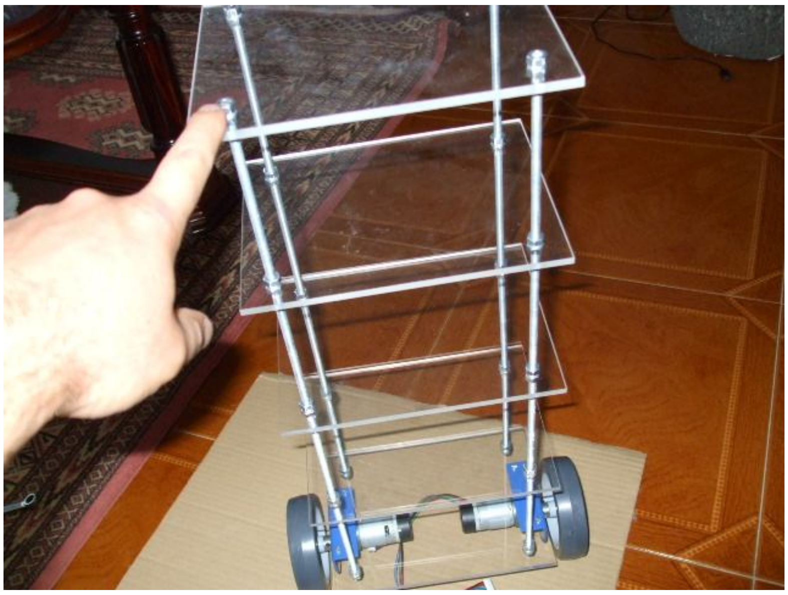


Figure 45 - Robot structure

4.4. ASSEMBLED CIRCUIT

The main circuit of the robot is composed by the sensors (accelerometer and gyroscope), the Arduino, the motors and battery.

When we assemble this circuit we need to make sure that the all GND (ground) pins are connected. And that we supply power to both accelerometer and gyroscope, it can be made throw the Arduino +5V output.

From the motors encoder we need to connect the SCL to pin 21, analog 5, of the Arduino and SDA is connected to two 1k8Ω parallel resistances and continues to pin 22, analog 4, of Arduino (the scheme is at figure 39). It's important that this pin are respected, otherwise the MD23 will not communicate with Arduino.

In the accelerometer we are interest in read the values of *x* and *y* and so we connect them to pins 23 and 24 respectively analog 3 and 2, from gyroscope we are only interested in the *rate* that we connected to pin 20, analog 6. They are connected to the analog pins of the Arduino because they are analog devices.

We used the +5v output (at orange in figure 41) of MD23 to power the Arduino Nano and its connected to pin 27, +5v of Arduino.

The following figure represents a schematic of the circuit.

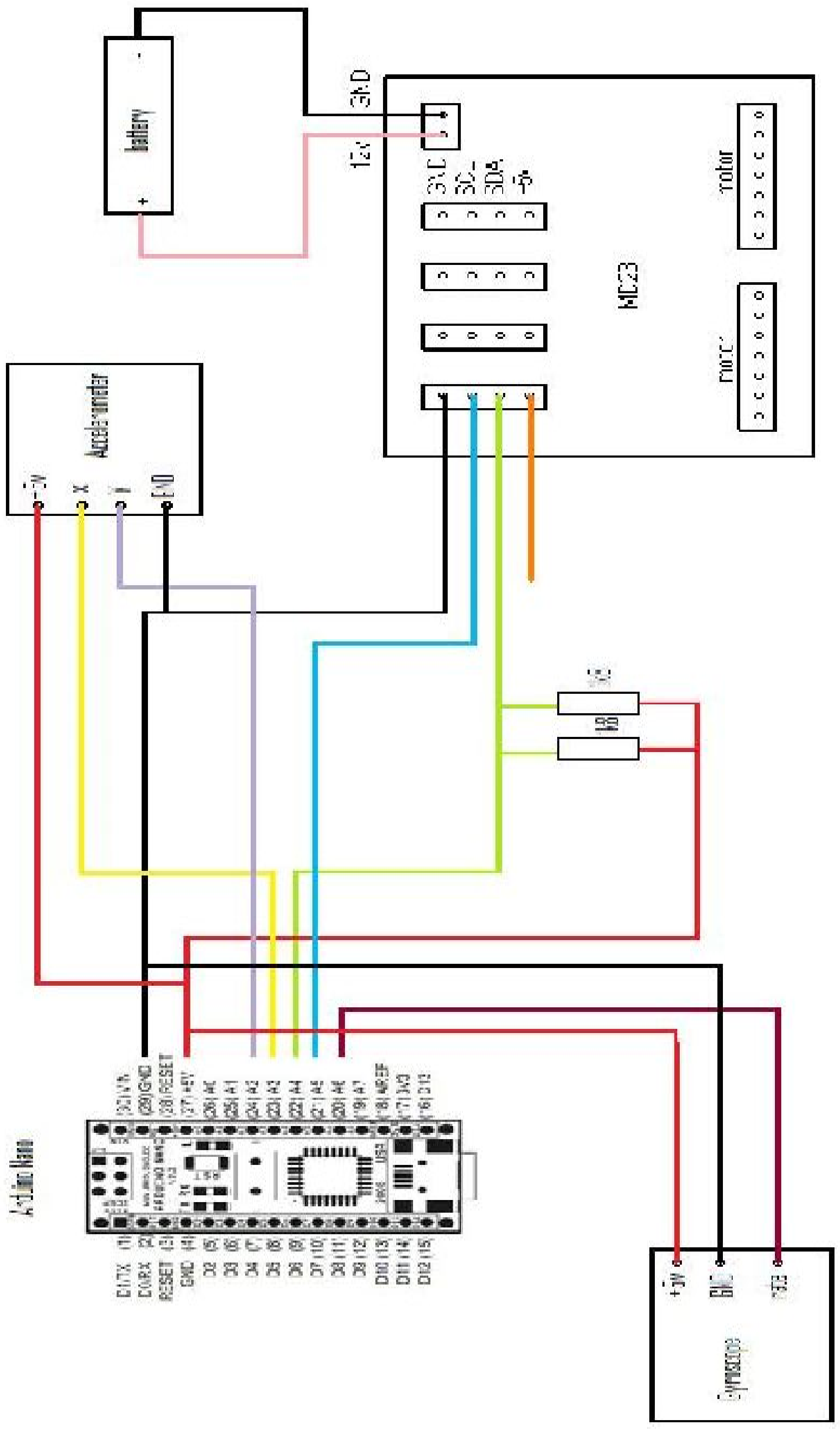


Figure 46 - Assembled circuit

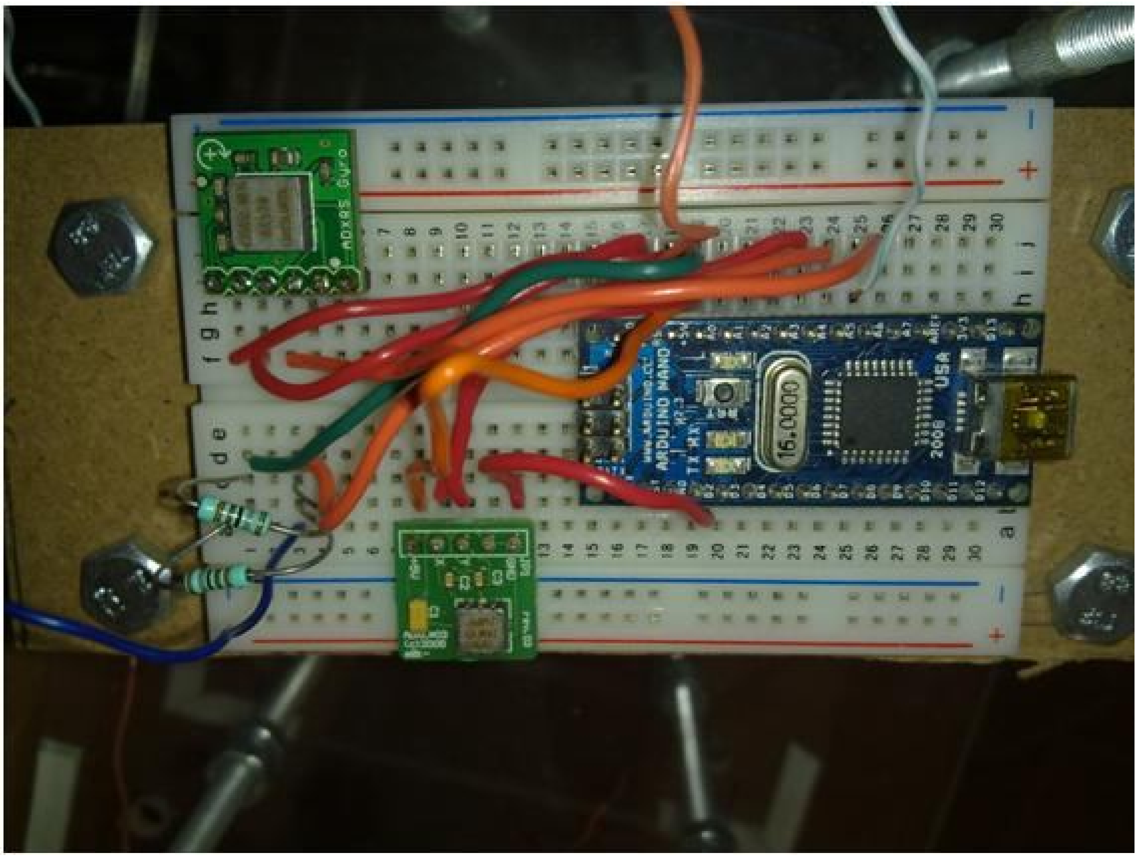


Figure 47 - Final Circuit

Now that the main structure is done and we know how to assemble the circuit we place the components in their place.

The sensors were placed at the top of the robot so that we can detect the angle the faster as possible.

The MD23 is placed in the base to be connected with the motors.

We have placed the 12v battery to power both motors and Arduino at the base just up the MD23.

The following figure represents the final version of the robot with all components placed.

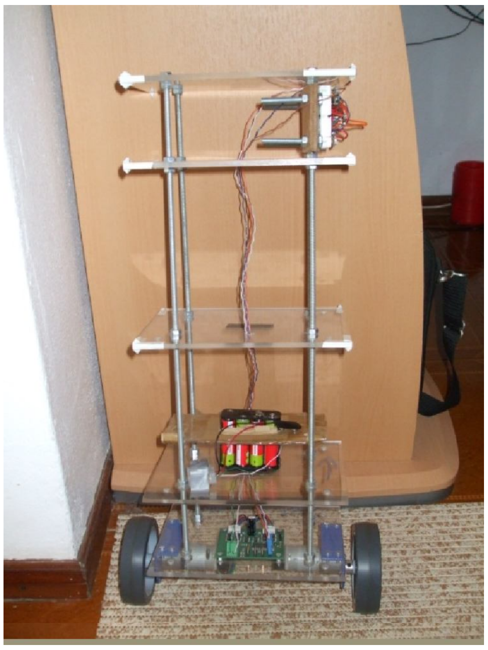


Figure 48 - Final robot front

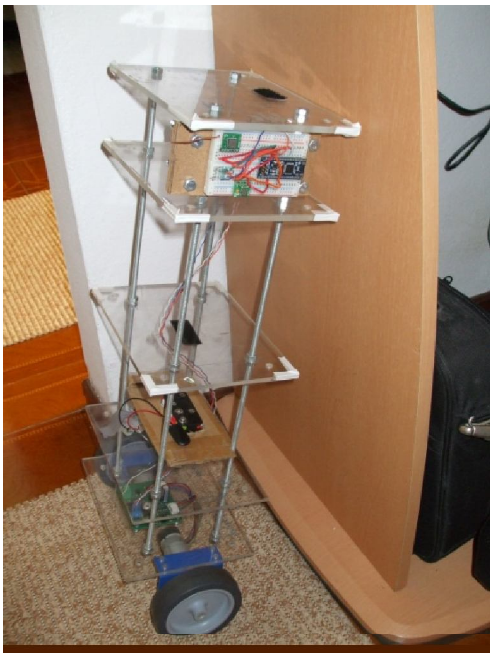


Figure 49 - Final robot side

4.5. SOFTWARE IMPLEMENTATION

Now that the circuit is assembled we need to know how to use sensor's information.

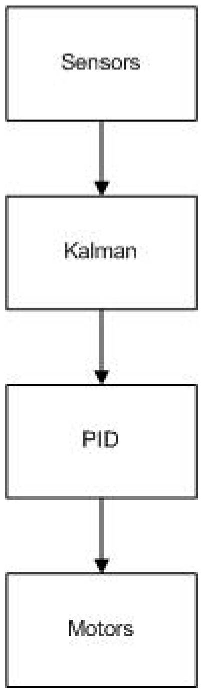


Figure 50 - Inverted Pendulum process

The inverted pendulum is an unstable system it can only reach a balance position if all its components acts according an in order. To make sure that they act in order the time scale must be precise for that we collect all the data and perform the kalman and pid calculation at 50Hz.

In an unstable system we must have mechanism that can help us measure that instability; in this case we need a way to measure the pendulum’s tilt for that we used an accelerometer and a gyroscope. We used a two-axis accelerometer because we need to measure the acceleration that the pendulum experiences in those two axes.

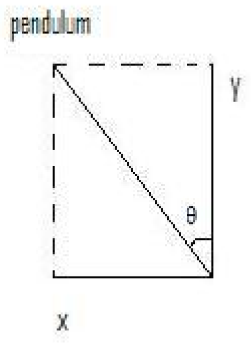


Figure 51 - Pendulum's Angle

Only with those values and some trigonometry it’s possible to determine the pendulum’s angle. If we only have one axis, that would not be possible.

After we have the angle (given by the accelerometer) and due to the fact that it is with noise we need to minimize that noise, this noise can be from a bad cable contact, bad readings from the pin. For estimate the real angle we need to use the accelerometer’s angle and gyroscope’s angular rate, which is noisy, and filter. We use a Kalman filter to achieve this.

Kalman filter is an efficient recursive filter that can estimate the state of a system from noisy values. It’s a powerful filter because supports estimations of past, present and even future states.

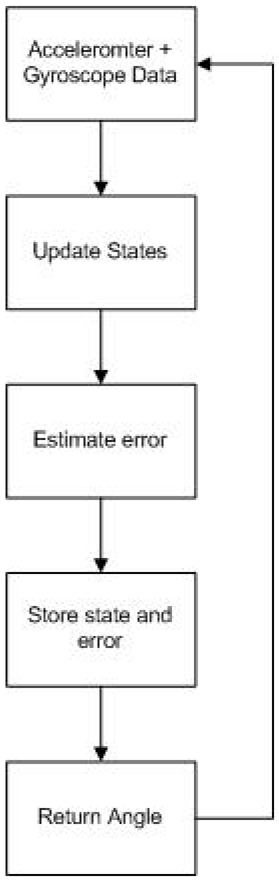


Figure 52 - Kalman process

After we get the final angle it's time to work with it. As we said before to make the robot balancing is necessary that the wheels move according to the angle, for this and even though the simulation was done using the LQR method, we use the PID method to balance the robot.

A PID controller attempts to correct the error between a measured process variable, which in our case is pendulum's angle, and a desired setpoint, for us the setpoint is the equilibrium angle, i.e., the angle when the robot is at an equilibrium position, by calculating and then instigating a corrective action that can adjust the process accordingly and rapidly, to keep the error minimal, by corrective action we understand the wheel's speed. We have tuned each of the PID values manually by trial and error.Then with PID's corrective action we feed the motors and if everything works well it should balance.

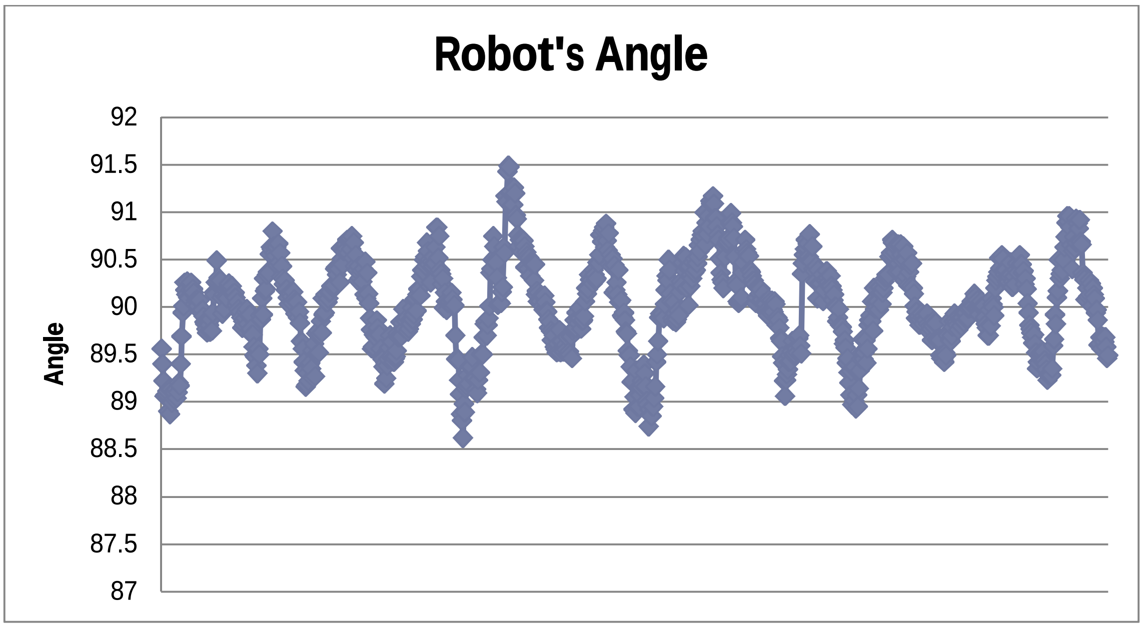
The next step is teleoperation. In this step we wanted to drive the robot with the help of a wiimote, for that we needed to receive the data of the wiimote to the Arduino, in this case we needed to use an Arduino BT.

CHAPTER 5 - ROBOT EVALUATION

In this chapter we will discuss the result of the tests that were made to the mobile inverted pendulum robot.

To test the dynamic of the mobile inverted pendulum we test its behavior when we change its unstable variable, i.e. when we changed the mass of the body. Due to the fact that we can't remove mass to the body we have test it by adding different mass to the body.

First we need to know the behavior of the pendulum as it is. More specifically, angle variation. For that we made the following graphic to illustrate it. It illustrates the angle variation in a 20 second time window.



Graphic 1 - Robot's angle

As we can see in the graphic the robot´s angle will tilt 1degree around the equilibrium angle, i.e. the angle when the robot is perfectly stable, in this case is 90degrees.

In the following test we add some mass to robot's chassis in different position and then we noticed the change in the angle.

The extra weights (20g, 100g and 400g) were added at the top and middle of the robot, in the center (black rectangle), positive (red rectangle) and negative (green rectangle), as showed in the figure 53, this has allowed us to notice the behavior of the pendulum to different variations of the weight in different parts of its body.

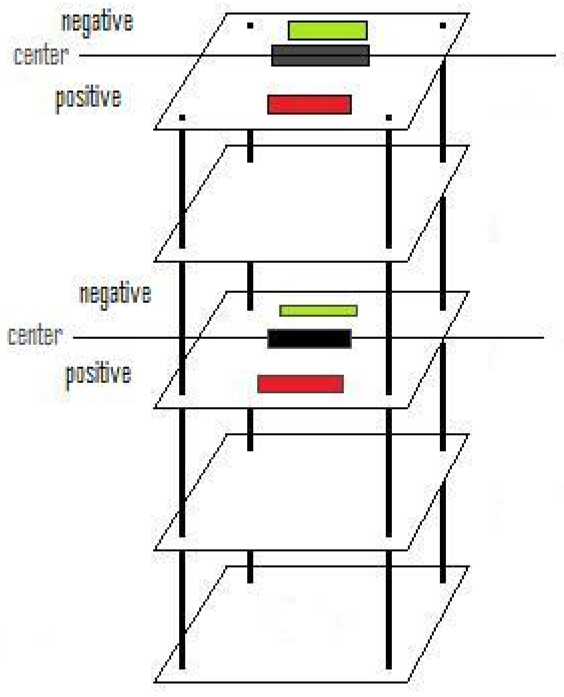
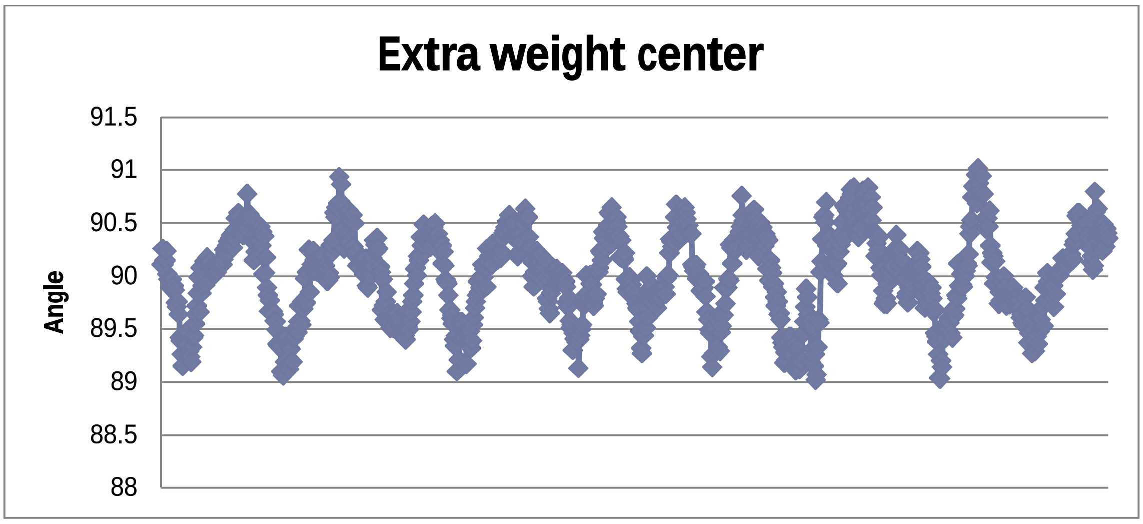


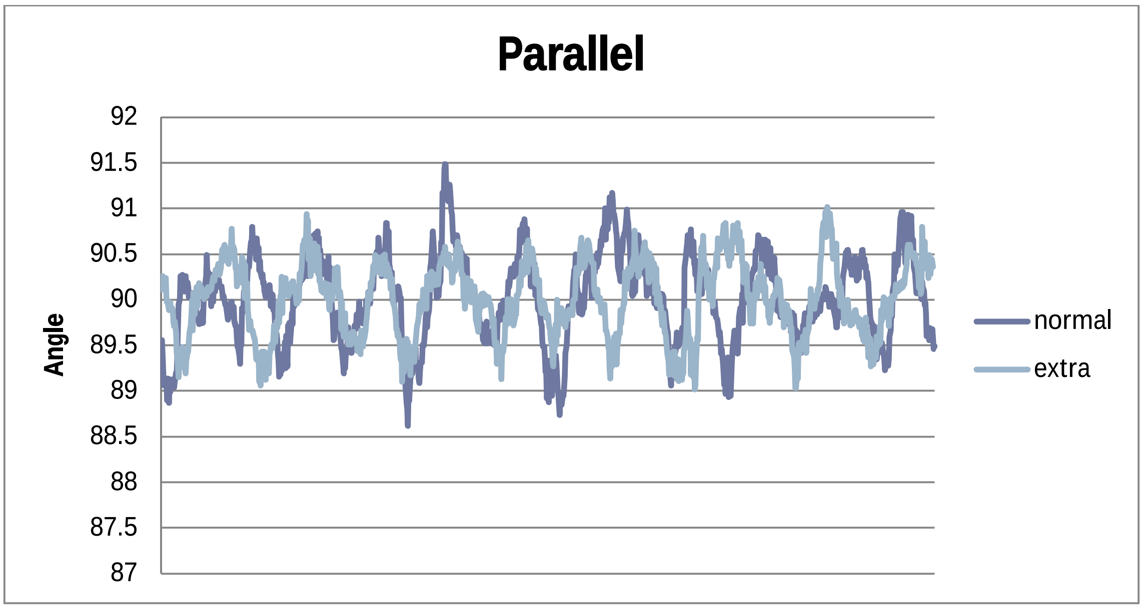
Figure 53 - Test structure

5.1. TEST WITH +20G OF EXTRA WEIGHT

In this test we add 20g of extra weight to the robot; it was made to observe the behavior of the robot to a slight increase in its mass.

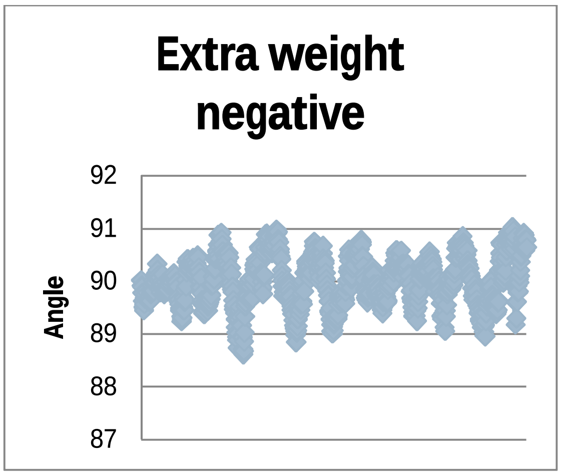


Graphic 2 - Extra 20g at top center

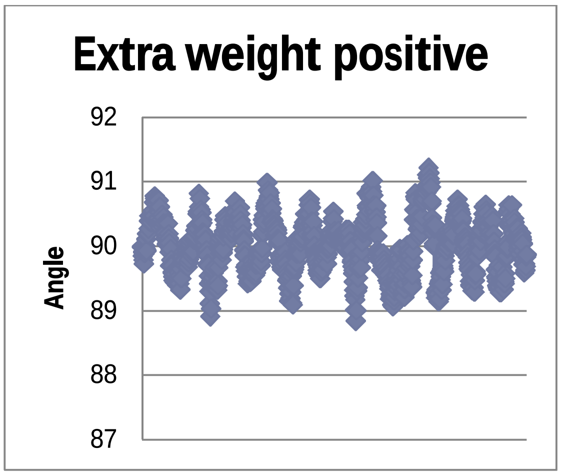


Graphic 3 - Parallel between Normal and Extra 20g at top center

As we can observe in the figure the robot's behavior was the same as before, the tilt of the robot was around 1 degree in relation to its setpoint. Therefore it's possible to add a little mass to the robot that it should continue balancing.



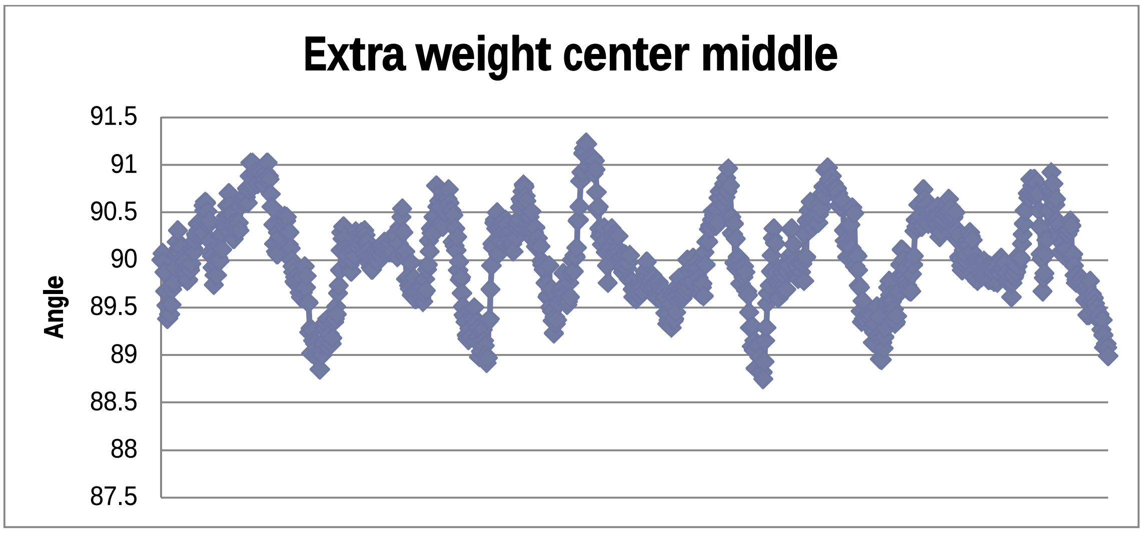
Graphic 4 - Extra 20g top negative



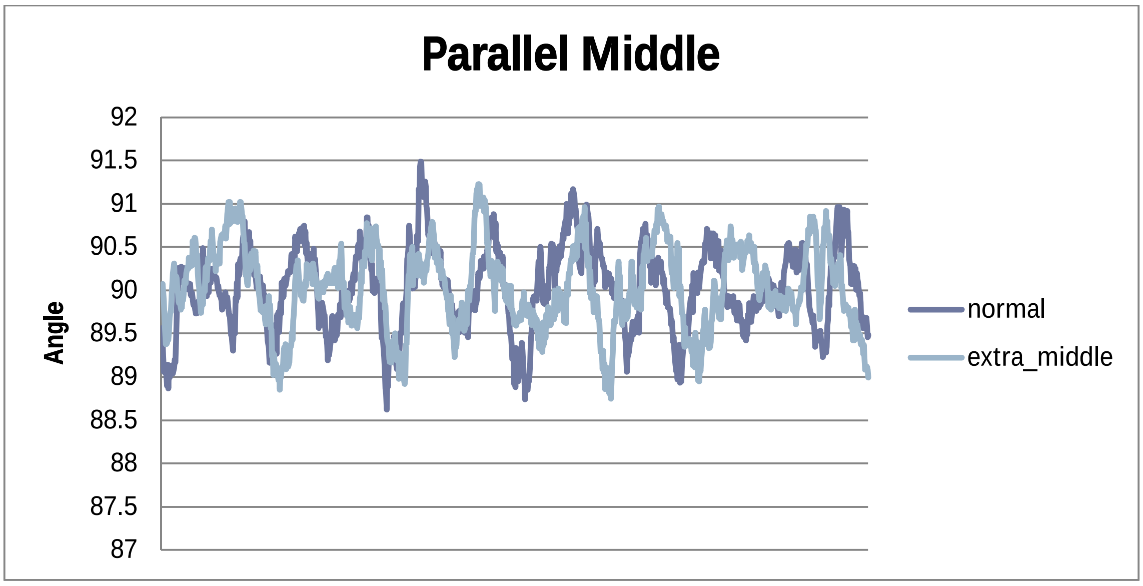
Graphic 5 - Extra 20g top positive

These two graphic represent the tilt of the angle when adding the extra weight in the negative or positive side of the robot. As we can see and for this mass there is no difference between them and between the original plot.

The next graphic represent adding the same extra weight but now in the middle of the body.

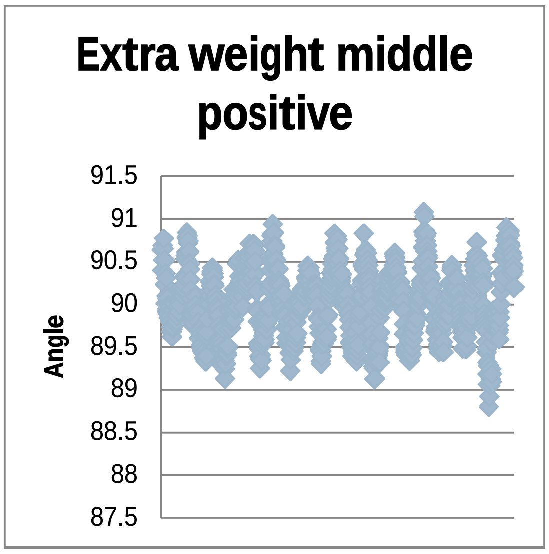


Graphic 6 - Extra 20g middle center

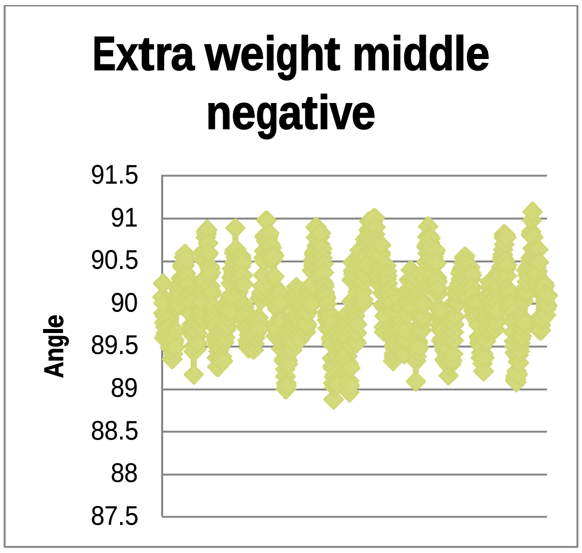


Graphic 7 - Parallel between normal and extra 20g middle center

As we can see the effects of that extra weight is practically none in the balancing of the robot even if that weight is added to the middle of the robot.



Graphic 8 - Extra 20g middle positive



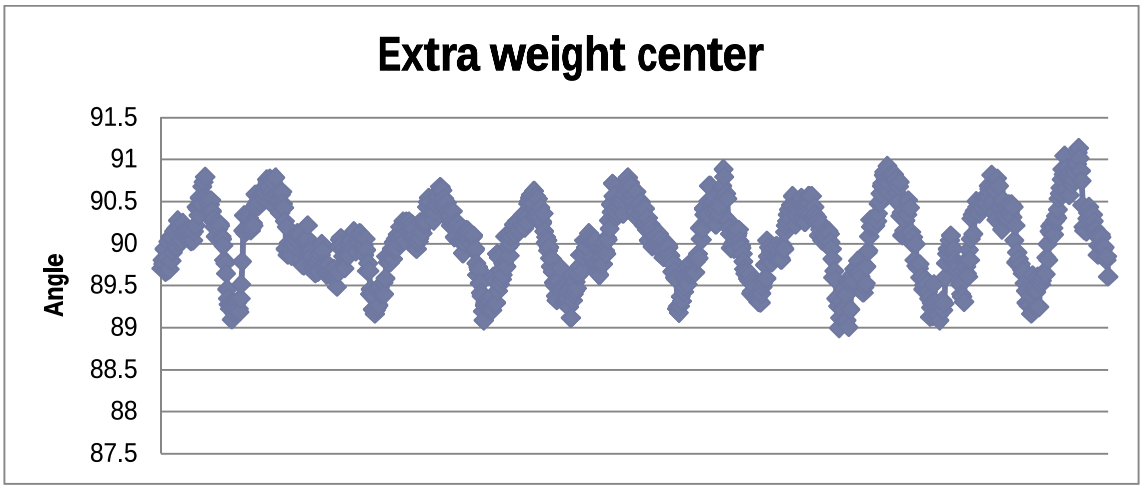
Graphic 9 - Extra 20g middle negative

These two graphic represent the tilt of the angle when adding the extra weight in the negative or positive side of the robot. As we can see and for this mass there is no difference between them and between the original plot.

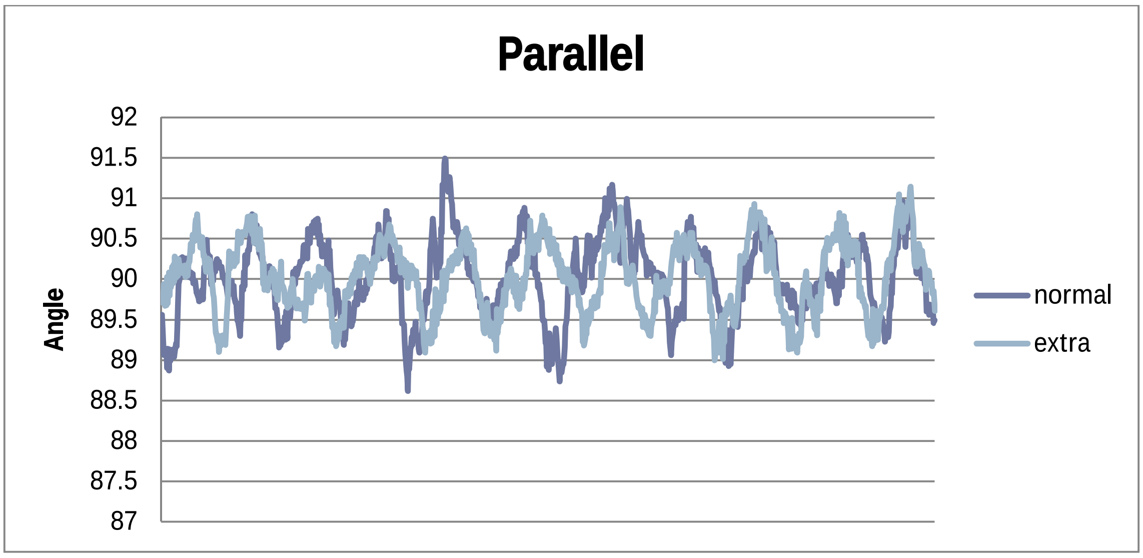
In conclusion we can observe that adding a small weight to the top or middle the robot will not influence the behavior of the robot. Regardless if that mass is added at the center, right or left. The robot should be able to balance.

5.2. TEST WITH +100G OF EXTRA WEIGHT

In this test we add 100g of extra weight to the robot; it was made to observe the behavior of the robot to a slight increase in its mass.



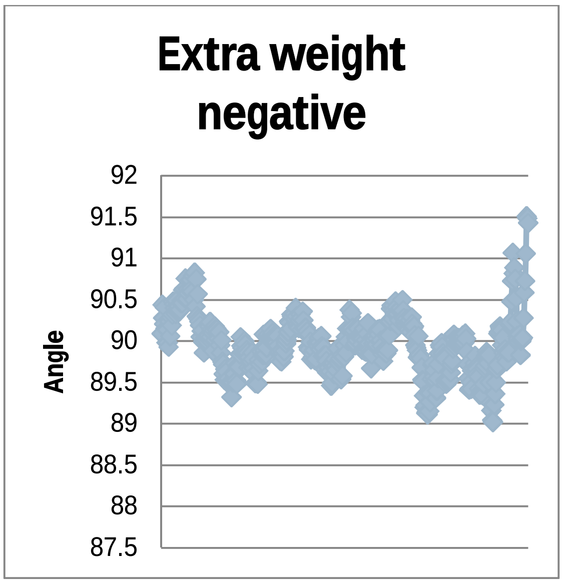
Graphic 10 - Extra 100g top center

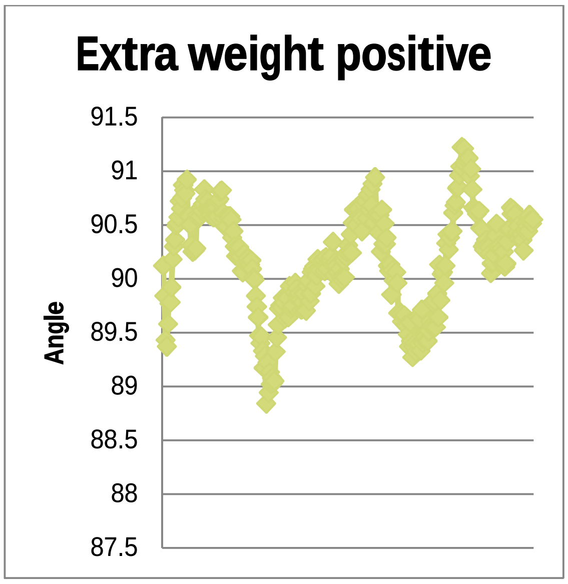


Graphic 11 - Parallel between normal and extra 100g top center

Even with 100g of extra weight the robot still can balance and the variation of the angle is the same, 1 degree around the setpoint, however the robot tilt more (frequency in the graphic).

The problem came when we add that extra weight to the left or right side of the robot, the robot can respond to that extra weight and it will fall. Both following test only have the duration of 10s, and as we can see the balance was not regular, it tilted a lot.

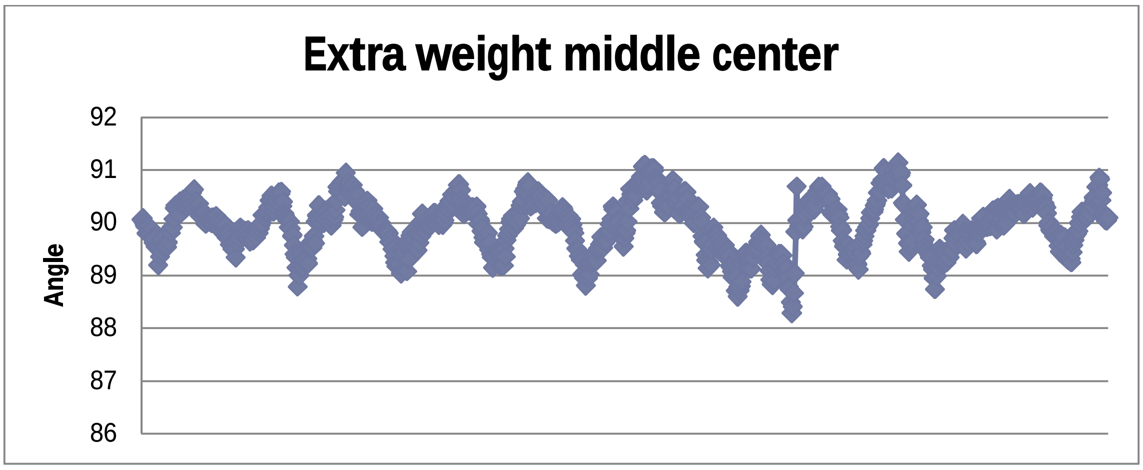




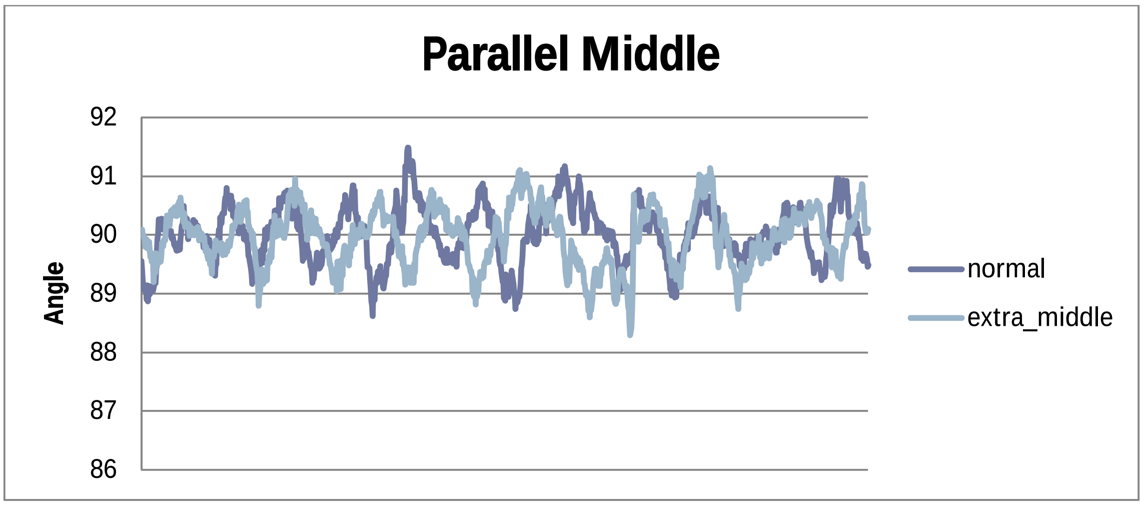
Graphic 12 - Extra 100g top negative

Graphic 13 - Extra 100g top negative

Now if we add the extra weight to the middle of the robot, the robot can balance as we can see in the following figures.

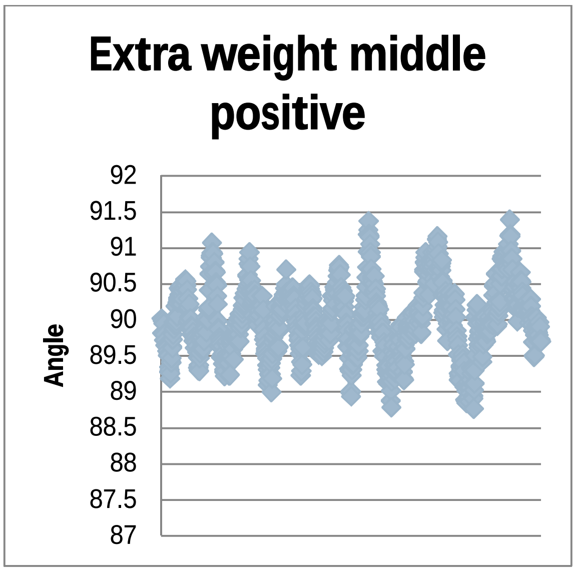


Graphic 14 - Extra 100g middle center

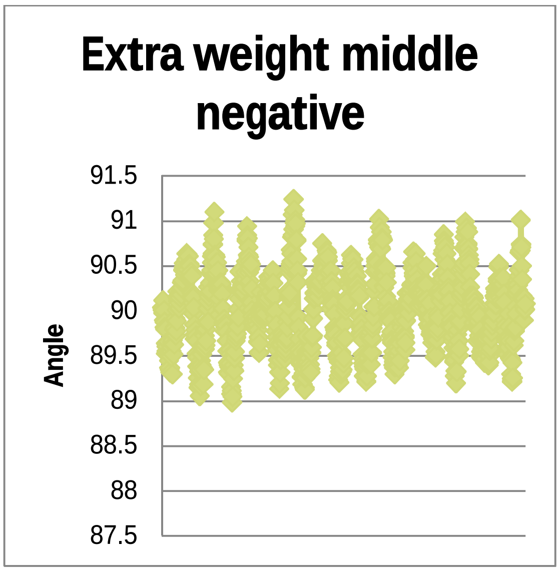


Graphic 15 - Parallel between normal and extra 100g middle center

However the robot can respond if we add that extra weight in the middle but in the left or in the right side. It falls about 10 seconds before the start.



Graphic 16 - Extra 100g middle positive

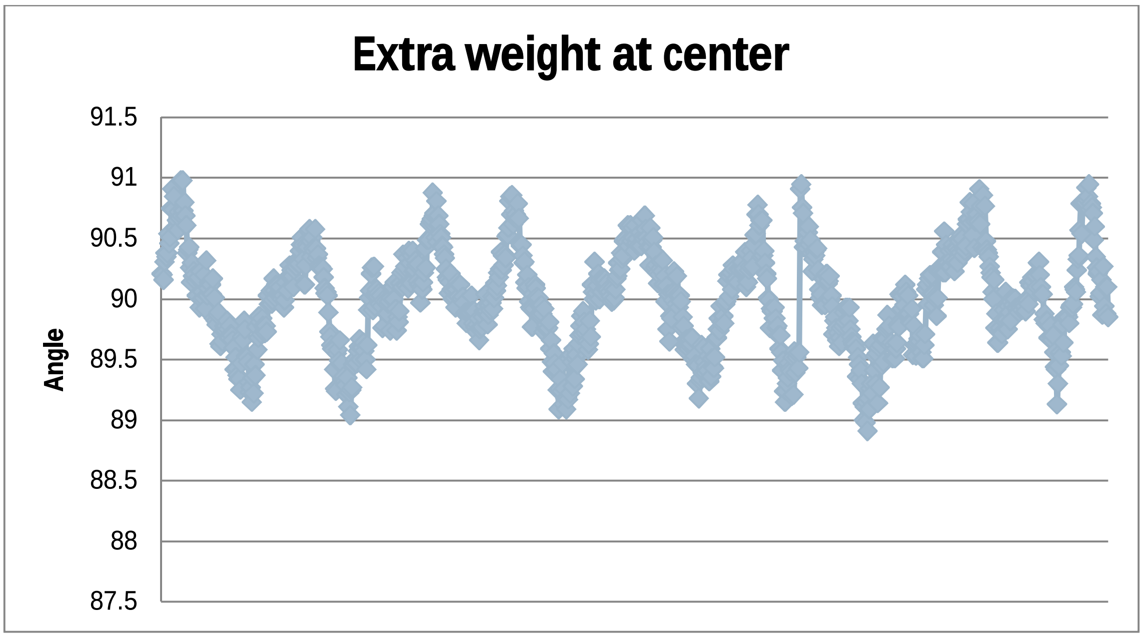


Graphic 17 - Extra 100g middle negative

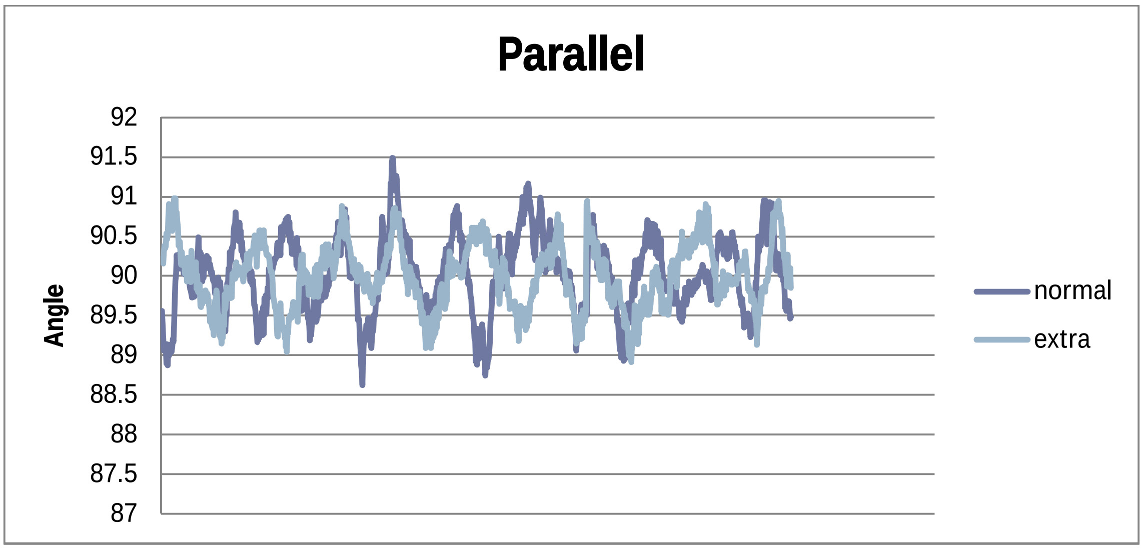
5.3. TEST WITH +400G OF EXTRA WEIGHT

In this test we add 400g of extra weight to the robot; it was made to observe the behavior of the robot to a slight increase in its mass.

The robot still can perform a shy balance, but cannot maintain balance for a big period of time, it eventually will fall faster that for the others extra weight.

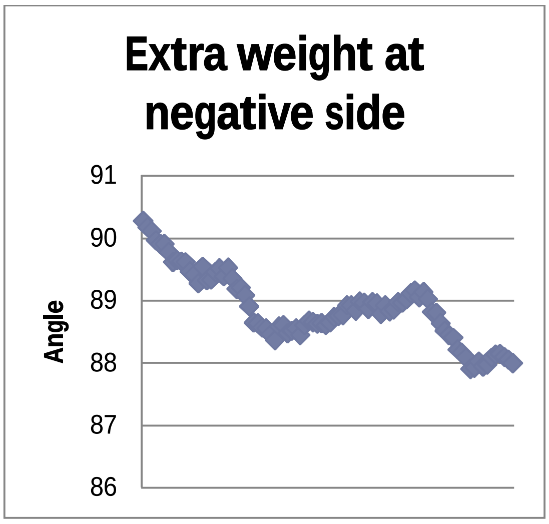


Graphic 18 - Extra 400g top center

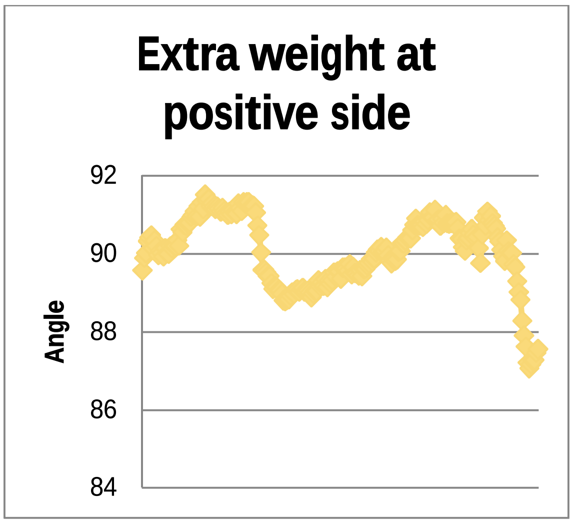


Graphic 19 - Parallel between normal and extra 400g top center

If we add that weight to the negative or positive side of the robot it will be chaotic. The robot can perform any kind of balance. It falls about 4second before it begins.

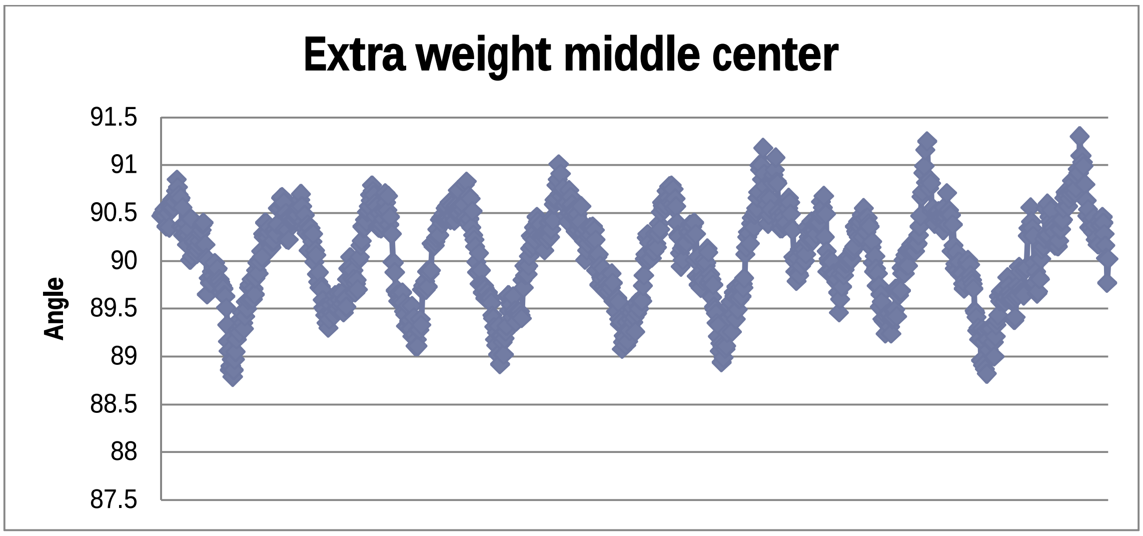


Graphic 20 - Extra 400g top negative

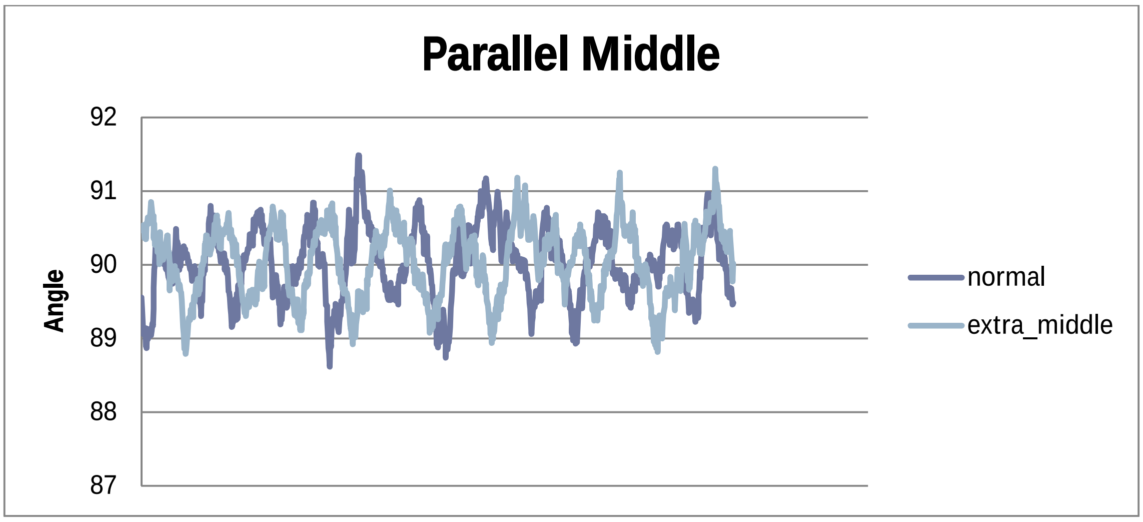


Graphic 21 - Extra 400g top positive

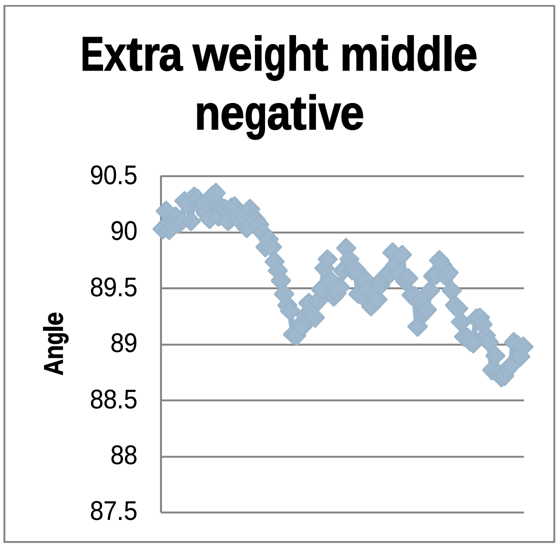
When adding the extra weight of 400g to the middle of the robot it behaves more erratically. It as a bigger tilts that in the normal case. If that weight is added to the positive or negative side of the robot, it has the same irregular behavior and falls.



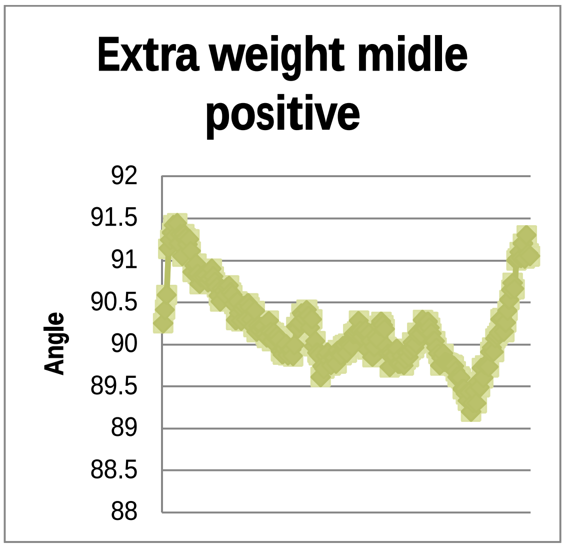
Graphic 22 - Extra 400g middle center



Graphic 23 - Parallel between normal and extra 400g middle center



Graphic 24 - Extra 400g middle negative



Graphic 25 - Extra 400g middle positive

In general if we add weight to the center of the robot it can still balance regardless if it is in the top or middle of the robot. The major difference in them is, if the weight is added to the top we are moving the center of gravity up and therefore it will take more time to settle the robot and will be necessary more acceleration for it.

When we added a heavy extra weight to the positive or negative side of the robot we observe an erratic behavior, this is due to the fact that the robot's center of gravity is shifted of the axis.

CHAPTER 6 – CONCLUSION

In control engineering we deal with dynamic systems, an example of one of those systems is the inverted pendulum system, in this thesis we have model and build a mobile inverted pendulum robot.

The inverted pendulum represents a fine example of an unstable system; it's widely used in research labs to demonstrate the capacities of feedback systems due to its unstable nature. It's considered a simplified representation of a rocket flying into space.

We have simulated the pendulum behavior in Matalb in which we could observe the behavior of our mobile inverted pendulum. In the simulations we used the Linear Quadratic Regulator method because it can provide us a rapid way to simulate an unstable system. This was a difficult part of the thesis because we didn't have any experience with Matlab and with simulations of this type.

In the simulation we assume that we have predefined a mass, maintaining all other variables, and then we add some extra weight to observe the settling time. From this we could see that a massive robot will take more time to settle in comparison with a lighter robot and therefore will origin a more unstable system. With this we can conclude that our robot will need to be as light as possible.

That's why we used Perspex and metal rods for the structure, they are both strong, to avoid that the robot break during the test and case it falls, and these components are not heavy.

In others simulations we changed the place of system center of gravity. This allowed us to see how the weight should be distributed in the robot and we have concluded that the center of gravity of the robot should be as low as possible.

We build a mobile inverted pendulum of 56cm tall because a taller robot is more easily balanced, we can observe that by balancing a broom for example, a taller broom is easily balanced than a shorter.

We initially start by using an IMU Combo board that included an accelerometer and a gyroscope but because of problems with that board we switched to a two-axis accelerometer breakout board and a gyroscope breakout board.

We choose the ADXL203 accelerometer because it’s a two-axis accelerometer with high precision and low cost. We needed for a two-axis accelerometer because we were interested in measuring the acceleration that the pendulum suffers in two axes so that we can determinate the angle.

We used a single axis gyroscope to correct the angle given by the accelerometer, it was only necessary a single axis gyroscope because we were interested in measuring the angular rate in the vertical direction, that is the direction were the pendulum balance.

The Arduino board used in this prototype has provided us a way to read the analog values from the accelerometer and gyroscope, and to communicate with the computer. It is the “brain” of our robot.

The developed prototype can balance using the implemented PID (proportional-integral-derivate) controller; we implemented a PID controller instead of the LQR (Linear Quadratic Regulator) controller used in the Matlab simulation because the PID controller is easier to implement. The values founded for the terms of the PID method were found by trial and error and that as caused that they may not be the more accurate as possible, also the process of trial and error isn’t the most academic process to find those values.

Due to the fact that the motors don’t have an integrated PID and because we didn’t have the opportunity to implement a PID in the motors the robot can only balance in a particular surface. The robot was developed and tested in that surface. Although we think that the major problem is in the wheels.

In the test that we made we can conclude that a small increase in the mass of the robot don’t influence robot’s behavior when they were placed in center to robot’s vertical axis. However we notice that when we add a heavy mass the robot tilted more and behave more erratically. Although we could observe a big change in the behavior of our prototype when we added the mass in the center, the same didn’t append when we added the mass on the sides of the robot. In those cases the robot can maintain the balance process and falls, it occurred faster when the weight was added to the top of the robot instead of in the middle, this is due to the fact that the center of gravity of the robot is shifted away of the vertical axis.

Therefore, future work for this thesis should involve improving the balance process of the robot so that it can be more stable, this could be by implementing a PID algorithm for the motors or by eventually changing the wheels or the motors.

It also should be considered for future work associating this robot with a wiimote to work on the teleoperation. The balancing implementation should be similar, there is only necessary to change the Arduino to a Bluetooth microcontroller and we can for example use the Arduino BT and work in the communication between both equipments. The idea to make the robot walk is changing the setpoint according to the desire direction. Teleoperation of system that exhibit fast dynamic behavior is a challenging application.

REFERENCES

A. Mellis, D. (2007, January 27). *Arduino - ArduinoBoardBluetooth*. Retrieved June 19, 2009, from Arduino: http://www.arduino.cc/en/Main/ArduinoBoardBluetooth

A. Mellis, D. (2006, March 25). *Arduino - ArduinoBoardNano*. Retrieved June 19, 2009, from Arduino: http://www.arduino.cc/en/Main/ArduinoBoardNano

Amherst, U. o. (2006, August 24). *Robotics at UMASS Amherst - Robots - UBot-2*. Retrieved June 18, 2009, from uBot-2: http://www-robotics.cs.umass.edu/Robots/UBot-2

Analog Devices, I. (2006, March). *ADXL203: Precision ±1.7g Dual-Axis iMEMS® Accelerometer*. Retrieved August 15, 2009, from Analog Devices: http://www.analog.com/en/sensors/inertial-sensors/adxl203/products/product.html

Anderson, D. P. (2007, July 1). *nBot, a two wheel balancing robot*. Retrieved June 18, 2009, from nBot Balancing Robot: http://www.geology.smu.edu/~dpa-www/robo/nbot/

Association, M. I. (n.d.). *Da Vinci*. Retrieved August 25, 2009, from Telerobotic Surgery: http://www.teleroboticsurgeons.com/davinci.htm

CJC. (1997, August 8). *CTM Example: State-space design for the inverted pendulum*. Retrieved April 15, 2009, from Control Tutorials for Matlab: http://www.engin.umich.edu/group/ctm/examples/pend/invSS.html

Electronics, S. (n.d.). *SparkFun Electronics - Gyro Breakout Board - ADXRS610 - 300°*. Retrieved August 15, 2009, from SparkFun Electronics: http://www.sparkfun.com/commerce/product_info.php?products_id=9058

Grasser, F. (2004, May 06). *JOE le pendule - Home*. Retrieved June 18, 2009, from The homepage of JOE le pendule, a mobile inverted pendulum: http://leiwww.epfl.ch/joe/index.html

Grasser, F., D'Arrigo, A., Colombi, S., & Rufer, A. C. (February 2002). JOE: A Mobile, Inverted Pendulum. *IEEE Transactions On Industrial Electronics, Vol.49, NO. 1* , 107-114.

Hassenplug, S. (2007, July 2). *Steve's LegWay*. Retrieved June 18, 2009, from Steve'sLegWay: http://www.teamhassenplug.org/robots/legway/

Hurbain, P. (n.d.). *Get up, NXTway!* Retrieved June 18, 2009, from Get up, NXTway!: http://www.philohome.com/nxtway/nxtway.htm

Larson, T. (2008). *Balancing Robot*. Retrieved June 18, 2009, from TedLarson.com: http://www.tedlarson.com/robots/balancingbot.htm

Nawawi, S., Ahmad, M., & Osman, J. (Dec 2007). Development of Two-wheeled Inveter Pendulum Mobile Robot. In *SCOReD07* (pp. 153-158). Malaysia, Malaysia.

Piponi, D. (n.d.). *Equibot the Balancing Robot*. Retrieved November 10, 2009, from Equibot the Balancing Robot: http://homepage.mac.com/sigfpe/Robotics/equibot.html

Robots Ltd., A. (n.d.). *Active Robots - RD01 Robot Drive System - UK*. Retrieved June 19, 2009, from Robot Electronics - RD01 12Vdc Robot Drive System: http://www.active-robots.com/products/motorsandwheels/rd01-drive.shtml

Robots Ltd., A. (2003). *SparkFun - Sensors - IMU/GYROS*. Retrieved June 19, 2009, from Active Robots - SparkFun - Sensors - IMU/Gyros - UK:: http://www.active-robots.com/products/sensors/sparkfun/5degreesoffreedom.shtml

techtarget.com. (2008, March 18). *Teleoperation*. Retrieved August 25, 2009, from What is Teleoperation?: http://whatis.techtarget.com/definition/0,,sid9_gci1196650,00.html

Times, T. J. (2001, December 31). *Scientist claims he made Segway predecessor in '86 | The Japan Times Online*. Retrieved June 18, 2009, from The Japan Times Online: http://search.japantimes.co.jp/member/member.html?nn20011231a9.htm

Watanabe, R. (2007, March 23). *NXTway-G*. Retrieved June 18, 2009, from NXTway-G: http://web.mac.com/ryo_watanabe/iWeb/Ryo%27s%20Holiday/NXTway-G.html

Wikipedia. (2009, September 17). *Dextre*. Retrieved September 22, 2009, from Dextre: http://en.wikipedia.org/wiki/Dextre

Wikipedia. (2009, November 16). *Kalman filter*. Retrieved November 23, 2009, from Wikipedia: http://en.wikipedia.org/wiki/Kalman_filter

Wikipedia. (2009, September 13). *Remote control vehicle*. Retrieved September 15, 2009, from Remote control vehicle: http://en.wikipedia.org/wiki/Remote_control_vehicle

Wikipedia. (2009, September 5). *Remotely operated underwater vehicle*. Retrieved September 15, 2009, from Wikipédia: http://en.wikipedia.org/wiki/Remotely_operated_vehicle

Wikipedia. (2009, July 21). *Teleoperation*. Retrieved 09 15, 2009, from Teleoperation: http://en.wikipedia.org/wiki/Teleoperation

Wikipedia. (2009, August 12). *Telerobotics*. Retrieved August 25, 2009, from Telerobotics: http://en.wikipedia.org/wiki/Telerobotics

APPENDIX

APPENDIX A – OPEN-LOOP POLES M-FILE

```
%data
M = 2.1;
m = 0.4;
%Hr = 0.25;
%Hfr = 0.6;
C = 1.5*0.0980;
%i = 0.006;
g = 9.8;
l = 0.3;
r = 0.05;

%variables
X=(1/3)*((M*(M+6*m)*l)/(M+(3/2)*m));
Y=(M/((M+(3/2)*m)*r))+1/l;
A23=g*(1-(4/3)*l*(M/l));
A43=g*M/l;
B2=((4*l*Y/3*X)-(1/M*l));
B4=-(Y/X);

%A, B, C and D matrices
A=[0 1 0 0;
0 0 A23 0;
0 0 0 1;
0 0 A43 0];
B_1=[0;
B2;
0;
B4];
B_2=[C+C];
B=B_1*B_2;
C=[1 0 0 0;
0 0 1 0];
D=[0;
0];

%open-loop poles
p=eig(A)
```

APPENDIX B - LQR DESIGN

```
C'*C

x=100000;
```

```
y=500;
Q=[x 0 0 0;
   0 0 0 0;
   0 0 y 0;
   0 0 0 0];
R = 1;
K = lqr(A,B,Q,R)
Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];

T=0:0.01:5;
U=0.2*ones(size(T));
[Y,X]=lsim(Ac,Bc,Cc,Dc,U,T);
plot(T,Y)
legend('Wheels','Pendulum')
```

APPENDIX C – NBAR CONTROL

```
Cn=[1 0 0 0];
Nbar=rscale(A,B,Cn,0,K)
Bcn=[Nbar*B];
[Y,X]=lsim(Ac,Bcn,Cc,Dc,U,T);
plot(T,Y)
legend('Cart','Pendulum')
```

APPENDIX D – ESTIMATOR

```
p = eig(Ac)

P = [-40 -41 -42 -43];
L = place(A',C',P)'

Ace = [A-B*K           B*K;
       zeros(size(A)) (A-L*C)];
Bce = [      B*Nbar;
     zeros(size(B))];
Cce = [Cc zeros(size(Cc))];
Dce = [0;0];
T = 0:0.01:5;
U = 0.2*ones(size(T));
[Y,X] = lsim(Ace,Bce,Cce,Dce,U,T);
plot(T,Y)
legend('Wheel','Pendulum')
```

APPENDIX E – KALMAN CODE

```
#include "math.h"

/*
 *
 * (c) 2003 Trammell Hudson <hudson@rotomotion.com>
 *
 * Converted to Java (c) 2008 Andy Shaw
 *
 *  This file is part of the autopilot onboard code package.
 *
 *  Autopilot is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  Autopilot is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with Autopilot; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 *
 */

unsigned long nextKCompTime = 0;

/*
    * Our update rate.  This is how often our state is updated with
 * gyro rate measurements.  For now, we do it every time an
 * 8 bit counter running at CLK/1024 expires.  You will have to
 * change this value if you update at a different rate.
 */
int             dtMS        = 20;   // 50 hz
float      dt       = ((float)dtMS/1000.0);         // 50 hz


/*
    * Our covariance matrix.  This is updated at every time step to
 * determine how well the sensors are tracking the actual state.
 */
float P[2][2] = {
  {
    1, 0  }
  ,
  {
    0, 1  }
 ,};

/*
    * Our two states, the angle and the gyro bias.  As a byproduct of computing
 * the angle, we also have an unbiased angular rate available.   These are
 * read-only to the user of the module.
 */
float angle = 0.0f;
float q_bias = 0.0f;
float rate = 0.0f;
/*
    * R represents the measurement covariance noise.  In this case,
 * it is a 1x1 matrix that says that we expect 0.3 rad jitter
 * from the accelerometer.
 */
static float R_angle = 0.3f;
/*
    * Q is a 2x2 matrix that represents the process covariance noise.
 * In this case, it indicates how much we trust the acceleromter
 * relative to the gyros.
 */
static float Q_angle = 0.001f;
static float Q_gyro = 0.003f;


/*
 * state_update is called every dt with a biased gyro measurement
 * by the user of the module.  It updates the current angle and
 * rate estimate.
 *
 * The pitch gyro measurement should be scaled into real units, but
 * does not need any bias removal.  The filter will track the bias.
 *
 */
boolean state_update(
float q_m, /* Pitch gyro measurement */
float dt)
{
  /* Unbias our gyro */
  float q = q_m - q_bias;

  /*
   * Compute the derivative of the covariance matrix
   *
   *      Pdot = A*P + P*A' + Q
   *
   * We've hand computed the expansion of A = [ 0 -1, 0 0 ] multiplied
   * by P and P multiplied by A' = [ 0 0, -1, 0 ].  This is then added
   * to the diagonal elements of Q, which are Q_angle and Q_gyro.
   */
  float Pdot[] = {
    Q_angle - P[0][1] - P[1][0], /* 0,0 */
    -P[1][1], /* 0,1 */
    -P[1][1], /* 1,0 */
    Q_gyro                          /* 1,1 */

  };

  /* Store our unbias gyro estimate */
  rate = q;

  /*
        * Update our angle estimate
   * angle += angle_dot * dt
   *      += (gyro - gyro_bias) * dt
   *      += q * dt
   */
  angle += q * dt;

  /* Update the covariance matrix */
  P[0][0] += Pdot[0] * dt;
  P[0][1] += Pdot[1] * dt;
  P[1][0] += Pdot[2] * dt;
  P[1][1] += Pdot[3] * dt;

  return true;
}


/*
 * kalman_update is called by a user of the module when a new
 * accelerometer measurement is available.  ax_m and az_m do not
 * need to be scaled into actual units, but must be zeroed and have
 * the same scale.
 *
 * This does not need to be called every time step, but can be if
 * the accelerometer data are available at the same rate as the
 * rate gyro measurement.
 *
 */
boolean kalman_update(
//const float           ax_m,  /* X acceleration */
//const float           az_m   /* Z acceleration */
float angle_m /* angle in rad, determined from accel */)
{
  /* Compute our measured angle and the error in our estimate */
  //const float         angle_m = atan2( -az_m, ax_m );
  //const float         angle_m = atan2( ax_m, az_m );      //bk
  float angle_err = angle_m - angle;
```

```
/*
      * C_0 shows how the state measurement directly relates to
 * the state estimate.
 *
 * The C_1 shows that the state measurement does not relate
 * to the gyro bias estimate.  We don't actually use this, so
 * we comment it out.
 */
float C_0 = 1;
/* const float          C_1 = 0; */

/*
      * PCt<2,1> = P<2,2> * C'<2,1>, which we use twice.  This makes
 * it worthwhile to precompute and store the two values.
 * Note that C[0,1] = C_1 is zero, so we do not compute that
 * term.
 */
float PCt_0 = C_0 * P[0][0]; /* + C_1 * P[0][1] = 0 */
float PCt_1 = C_0 * P[1][0]; /* + C_1 * P[1][1] = 0 */

/*
      * Compute the error estimate.  From the Kalman filter paper:
 *
 *      E = C P C' + R
 *
 * Dimensionally,
 *
 *      E<1,1> = C<1,2> P<2,2> C'<2,1> + R<1,1>
 *
 * Again, note that C_1 is zero, so we do not compute the term.
 */
float E =
  R_angle + C_0 * PCt_0 /* + C_1 * PCt_1 = 0 */;

/*
      * Compute the Kalman filter gains.  From the Kalman paper:
 *
 *      K = P C' inv(E)
 *
 * Dimensionally:
 *
 *      K<2,1> = P<2,2> C'<2,1> inv(E)<1,1>
 *
 * Luckilly, E is <1,1>, so the inverse of E is just 1/E.
 */
float K_0 = PCt_0 / E;
float K_1 = PCt_1 / E;

/*
      * Update covariance matrix.  Again, from the Kalman filter paper:
```

```
   *
   *      P = P - K C P
   *
   * Dimensionally:
   *
   *      P<2,2> -= K<2,1> C<1,2> P<2,2>
   *
   * We first compute t<1,2> = C P.  Note that:
   *
   *      t[0,0] = C[0,0] * P[0,0] + C[0,1] * P[1,0]
   *
   * But, since C_1 is zero, we have:
   *
   *      t[0,0] = C[0,0] * P[0,0] = PCt[0,0]
   *
   * This saves us a floating point multiply.
   */
  float t_0 = PCt_0; /* C_0 * P[0][0] + C_1 * P[1][0] */
  float t_1 = C_0 * P[0][1]; /* + C_1 * P[1][1]  = 0 */

  P[0][0] -= K_0 * t_0;
  P[0][1] -= K_0 * t_1;
  P[1][0] -= K_1 * t_0;
  P[1][1] -= K_1 * t_1;

  /*
       * Update our state estimate.  Again, from the Kalman paper:
   *
   *      X += K * err
   *
   * And, dimensionally,
   *
   *      X<2> = X<2> + K<2,1> * err<1,1>
   *
   * err is a measurement of the difference in the measured state
   * and the estimate state.  In our case, it is just the difference
   * between the two accelerometer measured angle and our estimated
   * angle.
   */
  angle += K_0 * angle_err;
  q_bias += K_1 * angle_err;
  return true;
}

float setDT(float newDT) //input in ms
{
  dt = newDT/1000.0;
  dtMS = newDT;
}
```

```
float getKalmanAngle()
{
  return angle;
}

boolean calculateKalman(float acc_angle_rads, float gyro_rads_sec)
{
  // do the timing
  unsigned long now = millis();
  if (now<nextKCompTime)
    return false;

  state_update(gyro_rads_sec, dt);
  kalman_update(acc_angle_rads);

  nextKCompTime = now + dtMS;
  return true;
}
```

APPENDIX F – MAIN CODE

```
// the time to update most significant parts of the loop (Kalman, PID)
int timeParam = 20;

//////////////////////////////////////////////////////////////////////////////////////
// Sensor pins and variables
//////////////////////////////////////////////////////////////////////////////////////
//define pins
int xaxis = 3;                  // x-axis of the accelerometer
int yaxis = 2;                  // y-axis
int xrate = 6;                  // x-rate of gyro

// define variables for the above
float xAcc = 0;
float yAcc = 0;
float xGyro = 0;


//////////////////////////////////////////////////////////////////////////////////////
// Kalman vars
//////////////////////////////////////////////////////////////////////////////////////
float gyroMult=1.1;
float acc_angle_rads=0;
float gyro_rads_sec=0;
float kalman_angle=0;

// timing monitor, mainly for Kalman
unsigned long t_start = 0;
```

```
////////////////////////////////////////////////////////////////////////////////////////
// MD23 and Motors stuff
////////////////////////////////////////////////////////////////////////////////////////
#include <Wire.h>
#include <MD23.h>


MD23 md23;  // create an MD23 object with default address


////////////////////////////////////////////////////////////////////////////////////////
// PID Stuff
////////////////////////////////////////////////////////////////////////////////////////
#include <PID_Beta6.h>

double PID_setpoint=90;
double PID_input=90;
double PID_output=128;
double inputRange = 7;

////////////////////////////////////////////////////////////
// define tuning parameters


// the strength of the response
double PID_proportional=0.83;
double PID_integral=0.40;
double PID_derivative=0.48;

////////////////////////////////////////////////////////////

//Specify the links and initial tuning parameters
PID myPID(&PID_input, &PID_output, &PID_setpoint, PID_proportional,
PID_integral, PID_derivative);

////////////////////////////////////////////////////////////////////////////////////////
// Main program
////////////////////////////////////////////////////////////////////////////////////////
void setup()
{

  Serial.begin(9600);

  setDT(timeParam);

  Wire.begin()
  md23.setMode(0);
  md23.resetEncoders();
  md23.setAccelerationRate(0x10);
  md23.disableSpeedRegulation();
  // Initialise the PID model
  PID_input = 90;
  PID_setpoint = 90;
  myPID.SetInputLimits(PID_setpoint-inputRange, PID_setpoint+inputRange);
myPID.SetOutputLimits(5, 250);
  myPID.SetSampleTime(timeParam);
  myPID.SetMode(AUTO); // the PID is active

  t_start = millis();

}

void enforceCycleTime(unsigned long cycleTime)
{
  // aiming for cycleTime ms loop
  unsigned long lastStart = t_start;
  t_start = millis();

  unsigned long minDiff = cycleTime;
  unsigned long diff = t_start - lastStart;
  if (diff < minDiff)
    delay (minDiff - diff - 1);
  t_start = millis();
}


void loop()
{
  // we may enforce the time to timeParam ms
  enforceCycleTime(timeParam);


  xAcc = (analogRead(xaxis)-506);
  xAcc = xAcc *= 0.0048544;
  yAcc = (analogRead(yaxis)-506);
  yAcc = yAcc *= 0.0048544;
  xGyro = (analogRead(xrate) - 508);

  acc_angle_rads = atan2(yAcc, xAcc);
  gyro_rads_sec = radians(xGyro*gyroMult);

  if (calculateKalman(acc_angle_rads, gyro_rads_sec))
  {
    kalman_angle = getKalmanAngle();
  }

  PID_input = degrees(kalman_angle);
  if (myPID.Compute())
  {
    int motors = ((int)PID_output);

    if (motors<128 && motors>5)
      motors = motors - 9;
    else if (motors>128 && motors <250)
      motors = motors + 9;

    md23.setMotor1Speed(motors);
    md23.setMotor2Speed(motors);
  }

  return;
}
```