University of Southern Queensland

Faculty of Engineering and Surveying

BALANCING A TWO WHEELED ROBOT

A dissertation submitted by

Kealeboga Mokonopi

In fulfilment of the requirements of

Courses ENG4111 and ENG4112 Research Project

Towards the degree of

Bachelor of Engineering and Bachelor of Business (Mechatronics and Operations management)

Submitted: November, 2006

ABSTRACT

INTRODUCTION

The two-wheeled balancing robot is a project that has become very popular of late, in the field of Mechatronics and Robotics. This project draws on the theoretical principles of the equally popular experiment of the inverted pendulum. The inverted pendulum system, unlike many other control systems is naturally unstable. The system therefore has to be controlled to reach stability in this unstable state.

1. BACKGROUND

The two-wheeled balancing robot operates on two wheels like the name suggests. The theory behind controlling this robot is moving the base of the robot towards the direction that the robot is falling and hence keeping the center of gravity of the robot vertically above the axis of the robot wheels at all times. This way the robot remains upright and does not topple over. To achieve this, the speed at which the center of gravity falls and it's displacement at every point in time should be known so that the base can be moved at a speed higher than the speed at which the center of gravity falls. Therefore the robot is mounted with sensors to measure both the tilt angle and the rate at which the angle changes. The robot is also mounted with sensors to measure the displacement of the wheels and speed. These are for both balancing the robot and controlling the horizontal movement of the robot.

2. OBJECTIVES

- To get the robot to settle at the upright position in the shortest settling time and smallest over shoot.
- To get the robot to move a predetermined distance along the horizontal whilst keeping its upright position
- To control the robot so that it goes around corners, if time permits.

3. METHODOLOGY

The robot was physically modeled as an inverted pendulum and the mathematical model was derived. Matlab control system toolbox was then used to analyze the system model and determine the system poles and stability region. The closed loop control system was then formulated. These were done using hypothetical parameters, the real robot parameters were then substituted in the model and the system balanced again.

4. CONCLUSION

The body of the robot has been completed and the wheels used are scooter wheels running on a belt system and 24VDC motors. The matlab code generation in C, for embedding in the Motorola HC12 microcontroller is in progress. The micro controller embedded system development is also in progress. Upon conclusion the robot should be able to balance and move on two wheels without falling over

Certification

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Kealeboga Mokonopi

Student Number: 0031234288

____
Signature

____
Date

ACKNOWLEDGEMENTS

I would like to express my gratitude to Mr Mark Phythian my project Supervisor for his unwavering support throughout the year as I was doing my project. I would also like to thank Professor John Billingsley for the support he gave me on Mark' absence, his help was so valuable.

I also want to thank my friends Dimpho, Maitseo and Ken in Brisbane who always made it possible for me to use books from UQ and QUT. Last but not least all the very important people who were there and the gentleman who helped me at the workshop for building my robot body.

Finally I want to thank God for everything.

TABLE OF CONTENTS

-

Chapter 1

1.1 Introduction

The dissertation is on the design of a two balancing wheel robot. A two wheeled robot is simply a robot that operates on two wheels. This is a topic that has attracted so much attention in the field of control engineering because of its nature as a natural unstably system. This particular project covers the modelling of the robot, investigation of a suitable control system techniques and methods and controller design and implementation. This dissertation starts with a literature review of the subject and continues to discuss important control essentials such as estimation and sensor fusion. The model of the system is then developed. Following the model building an investigation of suitable control techniques and controller design and implementation are covered. Lastly project's hardware implementation is covered.

1.2 Aim

The aim of the project is to balance the balance the robot and control it to a predetermined position.

1.3 Fundamental Control Principles

The control principle of the two wheel balancing robot is a simple and straight forward principle. It is simply driving the wheels of the robot or the base of the robot in the direction where the body is falling. It is the same principle as balancing a stick on the palm of the hand. When balancing a broom stick on the palm of a hand, a person balancing the stick moves the hand in the direction that the stick is falling. This serves to keep the centre of mass of the stick directly above the base of the stick. In like manner the centre of mass of the robot has to be kept vertically above the base of the robot, or above the axle of the robot wheels. Therefore when the robot tends to fall to the right the controller has to drive the wheels towards the right so as to keep the mass centre above the wheel axle. To move the robot to a pre determined position or a demanded target position, when it's on a balanced position, the motors turn slightly in the opposite direction to tilt the robot in the direction it must move. When the robot tilts the wheels starts to move in the direction that the robot is tilting. For the robot to keep moving the robot must remain in the slightly tilted position until the robot gets to the demanded target position. When the robot reaches the position, the wheels move to position the mass centre of the robot vertically above the axle. As long as the robot is upright the robot stays stationery.

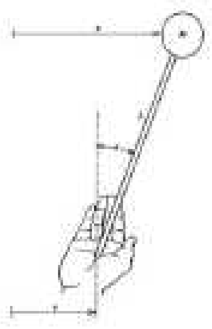


Figure 2.1: Free Body Diagram of inverted pendulum (Sophia University Japan 2005)

Chapter 2

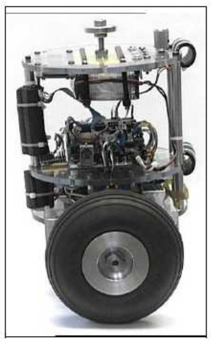
2.1 Literature Review

The two wheel balancing robot is a very popular project in the fields of robotics and control engineering. Therefore is a lot of work that has been done and more work is still been done on balancing a two wheeled robot. The following section is a literature review on this particular topic. A literature review is part of a research project where a researcher researches on similar work to his or hers. This very important part of the research helps the researcher to find out how other researchers have tackled the problem he/she is attempting to solve. It gives insight on how to go about solving the problem at hand and provides information on available technologies and tools for solving the problem.

2.1 Balancing Robots

Some of the work done on the two wheel balancing robot includes; Nbot by David Anderson, Joe le-Pendule by Felix Grasser et.al, Legway by Steve Hassenplug, Equibot by Dan Piponi and the Segway by Dean Kamen. There are many more projects that have been done on balancing a two wheeled robot that I have not covered in my literature review. The Nbot uses a total of four sensors to measure the states of the system. These sensors include the optical encoders on the motors to measure position of the robot and three other sensors to measure the tilt angle and it's rate of change. The three sensors include an accelerometer, rate gyroscope and tilt sensor. The accelerometer provides a measure of the tilt angle when the rate of change of the tilt angle is constant. This signal is obtained from twice integrating the raw signal from the sensor. The gyroscope gives a dynamic measure of the tilt angle. That is a measure when the rate of change of the angle is not constant. The signal from the rate gyro is integrated once to give the tilt angle. Finally the inclinometer or tilt sensor measures the tilt angle.

All these three sensors for the tilt angle and it's rate of change are in a single sensor called the FAS-G from Microstrain. Therefore there are three redundant sensors to measure the tilt angle. The signals from these sensors are fused together to provide a more accurate measure of the tilt angle. As mentioned above the accelerometer only gives the static measure of the angle and while the rate gyro gives the dynamic measure of the angle. The gyroscope is quite accurate however a drift problem, it's accuracy declines with time in operation. The inclinometer on the other hand has got slow dynamics, it reacts slowly and hence its measurement always lags the real tilt angle. The FAS-G uses a Weiner filter to fuse these three signals together to produce a signal of better quality. The Nbot uses an HC11 microcontroller to control the robot. Below is a picture of the Nbot



Joe Le-Pendule is another very exciting two wheel balancing robot. This particular robot has two decoupled control systems. It has a controller that balances the robot and controls its forward and backward movements. Another controller controls movements about its vertical axis. The robot can spin around its vertical axis and make u-turns. This robot is radio controlled. Joe Le-Pendule only uses an accelerometer and a rate gyro to measure the tilt angle of the robot. It uses filters to fuse the signals together and produce a tilt signal. It also has motor encoders to measure the position of the robot. Below is a picture of the Joe Le-Pendule robot.

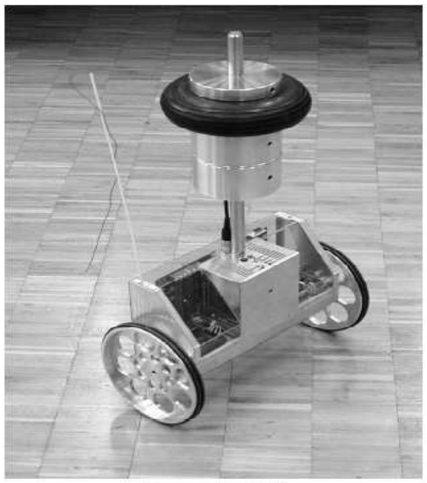
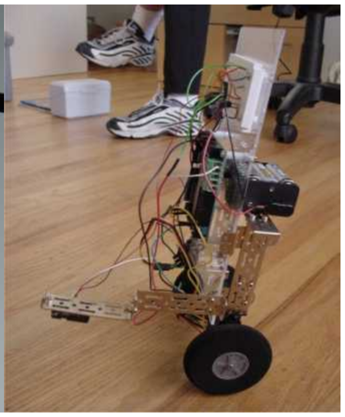


Figure 1. JOE

Another exciting two wheel balancing robot is the legway. The legway was built by Steve Hassenplug and he used Lego bricks to build the robot. This robot uses Infrared Proximity detectors to deduce the tilt angle of the robot. Another robot similar to the Legway is the Equibot by Dan Piponi. This one uses the Sharp GP2D120 Infrared ranger to measure the distance to the ground. From the distance to the ground the microcontroller deduces the tilt angle of the robot and where the robot is falling. Below are picture of both the Legway and the Equibot.



Legway Equibot

Lastly there is the Segway, the segway is the pinnacle of all these projects. The segway is a human transport system that has been produced by Dean Kamen. It is a two wheel balancing scooter as some call it. Its principle is similar to all the other two wheel balancing robots. According to the information on the website called howstuffworks the segway has got five gyroscopes and two more tilt sensors. These sensors are used to keep the segway balanced so that it doesn’t fall over. The segway only needs three gyroscopes to measure the forward and backward tilt angles and the corresponding rate of change angle. The other two gyroscopes are included for redundancy; this means that the signals from these sensors are fused with other sensor signals to produce a better and more reliable signal. The segway has got ten onboard microcontrollers to balance and control the segway. The segway can move forward, backwards, turn and spin around. To turn the Segway, the rider turns the handle bars in the direction they want to turn and the inner wheel is driven at a speed slower than the outer wheel to turn the segway. To spin around the wheels are driven in opposite directions.

2.2 Sensor Fusion

Sensor fusion is the act of combining signals from different sensors together to produce a better signal. The need for sensor fusion comes about because sensors are not reliable and they do not produce perfect results. A lot of things compromise the accuracy of a sensor. These include the sensor dynamics and noise. Furthermore different sensors have got different strengths and weaknesses. Combining multiple sensors improves the quality of the signal by using the strengths of one sensor to compensate for the weaknesses of the other sensor.

There are three classes of sensor interaction in a network of sensors. The first class is the complementary sensor class. In this class the sensors complement each other. They are not directly dependant on each other but they can be combined to give a more complete image of the environment. The next class of sensor interaction is the competitive class. Competitive sensors work independently of each other and they produce the same signal. These are called redundant sensors. When combined together the sensors produce a more reliable and accurate measure than their individual signals.

Lastly there is the cooperative class of sensor interaction. Cooperative sensors combine to produce a signal that can only be obtained from the sensors combined. The signal cannot be obtained from individual sensors.

2.3 Levels of Sensor fusion.

There are three levels of sensor fusion. The three levels are raw data level, state vector (feature) level and decision level. The raw data level is where the raw data from sensors is combined. The state vector level is where parameters concerning features are combined. The raw data from sensors is processed to produce the system parameters and then fusion follows. The last level is the decision level, here decisions are combined. Raw data is processed to produce parameters and the parameters are further processed to produce decisions then fusion follows.

2.4 Centralized and Decentralized fusion.

The last thing to cover on sensor fusion is centralized and decentralized fusion. In centralized fusion the information from sensors is combined in a single processor. Decentralized fusion on the other hand involves using multiple processors to process sensor information and perform fusion. Decentralized fusion gives more reliability and accuracy to the central system.

Chapter 3

3.1 Modelling

The robot has been modelled as an inverted pendulum on cart system. The principles behind controlling the inverted pendulum on cart system are the same as the principle that govern the control of the two wheel robot. Figure below is a picture of the inverted pendulum on cart system. The picture includes the external forces acting on the system.

Where:

- F is the driving force of the motors through the axle of the wheels
- B is the frictional force opposing the motion of the cart-pendulum system
- Mg is the force of gravity on the cart alone
- mg is the force of gravity on the pendulum alone.

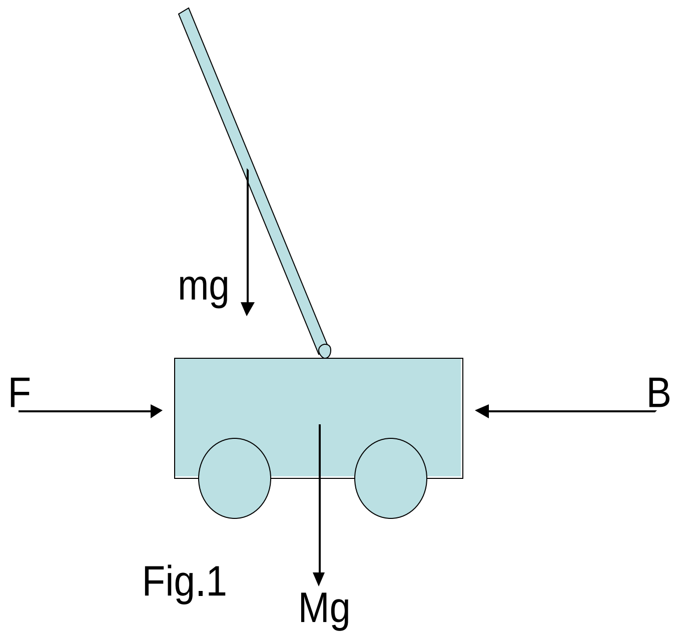


Fig.1

3.11 Free-body Diagram of the Cart

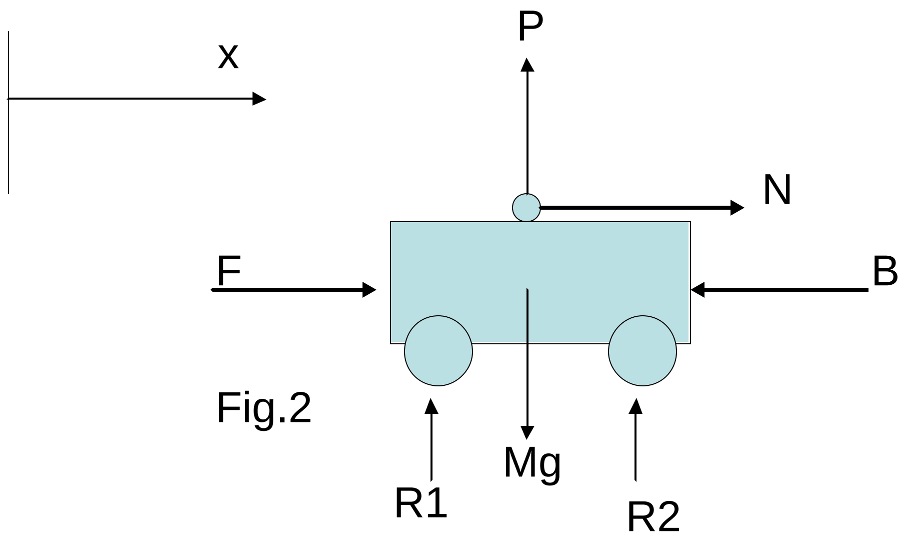


Fig.2

Where;

- P is the vertical force on the cart by the pendulum
- N is the horizontal force on the cart by the pendulum
- R1 and R2 are the reaction forces through the wheels.

From the free body diagram of then cart, we resolve forces in the x-direction, we could resolve forces in the y-direction but they do not give any useful equation towards the derivation of the system equations. Forces in the x-direction give the equation 3.1 below.

$$F + N - B = Ma_x \tag{3.1}$$

3.12 Free-Body Diagram of the Pendulum

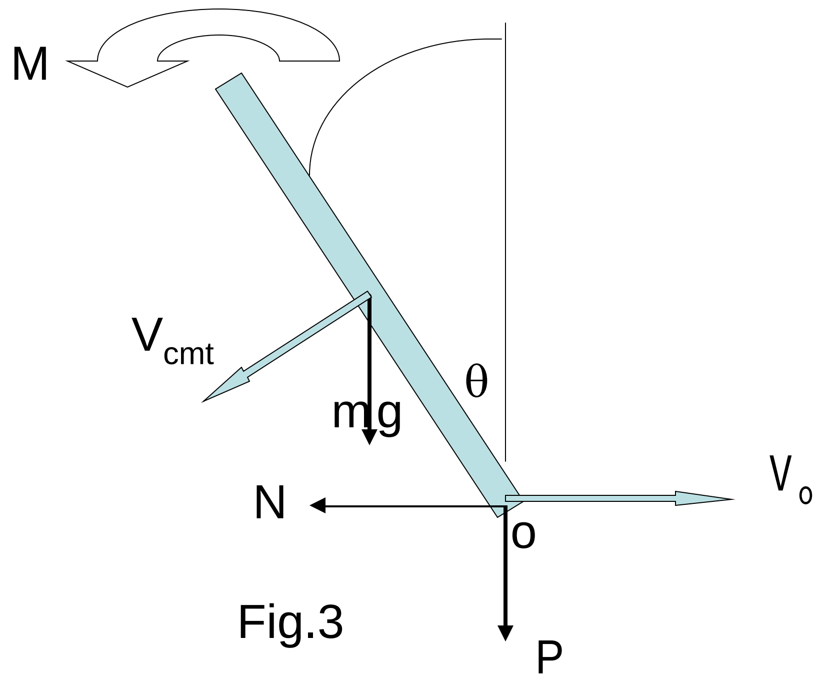


Fig.3

Where;

- M is the moment of force about point 'o' due to forces N and P
- Forces N and P are the force on the pendulum due to the cart resolved in the x and y directions
- V_{cmt} is the velocity of the mass center of the pendulum
- V_o is the velocity of point 'o' , which is in the x-direction
- θ is the displacement angle of the pendulum from the vertical

From the free-body diagram of the pendulum we derive the equations of motion of the pendulum. First of all the acceleration of the mass centre of the pendulum has to be derived. The acceleration of this point is derived from the velocities of point 'o' and that of the mass centre of the pendulum. We proceed by finding the velocity of the mass centre relative to the point 'o', this gives equation 3.2 below;

$$Vcmo = -\ell\dot{\theta}\cos\theta\vec{i} - \ell\dot{\theta}\sin\theta\vec{j} \tag{3.2}$$

From equation 3.2 we derive equation 3.3, which is the velocity of the mass centre relative to the inertial frame or the absolute velocity of the mass centre of the pendulum.

$$Vcm = (\dot{x} - \ell\dot{\theta}\cos\theta)\vec{i} - \ell\dot{\theta}\sin\theta\vec{j} \tag{3.3}$$

To get the absolute acceleration of the mass centre the absolute velocity derived in equation 3.3 above is differentiated. The resulting acceleration is given in equation 3.4 below;

$$a = (\ddot{x} - \ell\ddot{\theta}\cos\theta + \ell\dot{\theta}^2\sin\theta)\vec{i} - (\ell\ddot{\theta}\sin\theta + \ell\dot{\theta}^2\cos\theta)\vec{j} \tag{3.4}$$

The equation is given in vector notation as the other equations before it.

Now after finding the equation of acceleration of the mass centre of the pendulum, the summation of forces can be done. Summing the forces in the x-direction we get the next equation 3.5. To derive this equation the x-component if the acceleration above is used and the y-component is ignored.

$$N = ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta - m\ddot{x} \tag{3.5}$$

Summing forces in the y-direction give another equation, equation 3.6. Here the y-component of the acceleration is used and the x-component is ignored. Hence equation 3.6 below;

$$P = ml\ddot{\theta}\sin\theta + ml\dot{\theta}^2\cos\theta - mg \tag{3.6}$$

Finally summation of moments about the mass centre gives the last equation, equation 3.7

$$-Nl\cos\theta - Pl\sin\theta = Icm\ddot{\theta} \tag{3.7}$$

Substituting equations 3.5 and 3.6 in 3.7 gives equation 3.8 below.

$$ml\ddot{x}\cos\theta - (ml^2 + I)\ddot{\theta} = -mgl\sin\theta \qquad (3.8)$$

Substituting equation 3.5 in 3.1 and simplifying by putting like terms together give equation 3.9

$$(M + m)\ddot{x} - ml\ddot{\theta}\cos\theta = F - B\dot{x} - ml\dot{\theta}^2\sin\theta \qquad (3.9)$$

The next step is to solve equations 3.8 and 3.9 simultaneously for the second derivative of the tilt angle. The process involves a lot of algebra to simplify the equation. The final product is the equation 3.10 below;

$$\ddot{\theta} = ml/S\{[F - B\dot{x}]\cos\theta - ml\dot{\theta}^2\cos\theta\sin\theta + (M + m)g\sin\theta\} \qquad (3.10)$$

Where;

- $S = ml^2(M + m\sin^2\theta) + I(M + m)$

Equations 3.8 and 3.9 are solved simultaneously again for the second derivative of x. The process is quite long and involved like solving for the second derivative if the angle. The outcome is equation 3.11 below;

$$\ddot{x} = 1/S\{(I + ml^2)[F - B\dot{x} - ml\dot{\theta}^2\sin\theta] + (ml)^2 g\cos\theta\sin\theta\} \qquad (3.11)$$

Equations 3.10 and 3.11 are the equations that model the cart-pendulum system. These equations are not linear. The system is linearized about small deflections of theta, the tilt angle. It is linearized so that the methods of linear systems can be applied to analyze and control the system. Therefore in order to linearize the system theta is restricted to small deflections about the origin, which is the vertical position.

Hence if $|\theta|$ never exceeds 0.1 rads

Then: $\cos\theta \approx 1$

$\sin\theta \approx \theta$

$\dot{\theta}^2 \sin\theta \approx 0$

The linear equations are as follows;

$$\ddot{\theta} \approx ml / S[F - B\dot{x} + (M + m)g\theta]$$

$$\ddot{x} \approx 1/S\{(I + ml^2)[F - B\dot{x}] + m^2 l^2 g\theta\}$$

$$S \approx I(M + m) + mMl^2$$

From the system equations the state space model of the system is derived as below;

$$\begin{bmatrix} \dot{x}1 \\ \dot{x}2 \\ \dot{x}3 \\ \dot{x}4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -[(I+ml^2)B]/s & (m^2l^2g)/s & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -(mlB)/s & [mgl(M+m)]/s & 0 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} + \begin{bmatrix} 0 \\ (I+ml^2)/s \\ 0 \\ (ml)/s \end{bmatrix} * F$$

$$y = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} F$$

Chapter 4

4.1 Estimation theory

To successfully control a system, accurate information about the states of a system at every point in time is required. However to obtain accurate information about a system isn't an easily achievable task. For starters, the very model that represents the system is not the exact representation of the system; it's a close approximation of the system behaviour. When modelling a system only the most significant behaviours are modelled and therefore some behaviours which are not deemed important are not modelled. There is a trade off made between capturing most of the system's behaviour and simplifying the model. Secondly dynamic systems are not only driven by the control inputs, there are also some disturbances which alter the behaviour of the system but are not modelled. These are just two of the many reasons that make correct estimation of the system states a difficult task. To add on to these, sensors that are used to measure the output signals from the system are themselves not accurate and do not provide perfect information. Their signals are corrupted with noise and distortions. Having said all that, it is still imperative to retrieve system information that is as close to the actual information as possible. To obtain accurate data from noise corrupted observations and inaccurate models the estimation theory is used.

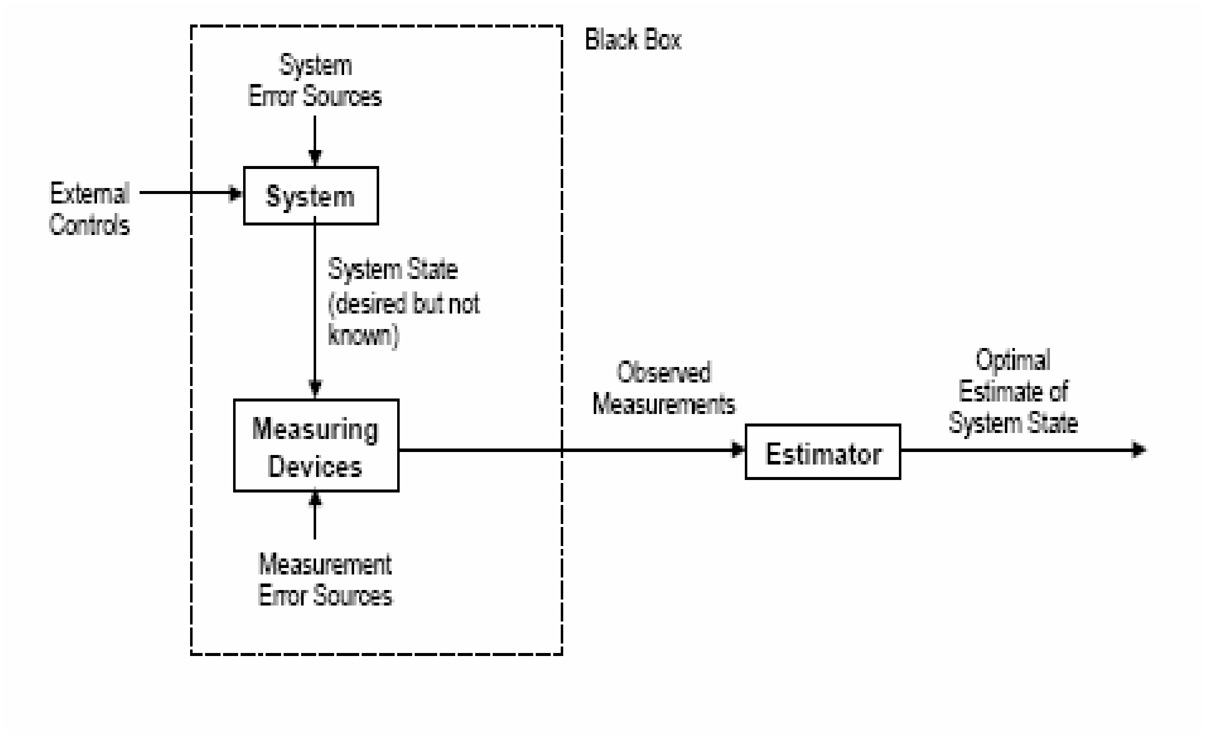
Estimation theory is the application of mathematical analysis to the problem of extracting information from observational data (George Siouris). Estimation is characterized as prediction, filtering and smoothing. George Siouris defines prediction, filtering and smoothing as the following:

> Prediction means extension in some manner of the domain of validity of the information. Filtering refers to the extraction of the true signal from the observation and smoothing usually refers to the elimination of some noisy or useless components of the data.

Filters use observations up to and including the time that the state of the dynamic system is to be estimated. Soothers use observations beyond the time that the state of the dynamic system is to be estimated. Lastly predictors use observations strictly prior to the time that the state of the dynamic system is to be estimated. There are different tools that are used to try and predict the actual states of the system to a high degree of certainty as possible. These include the Wiener filter and the Kalman filter. The Wiener filter was developed by N. Wiener in 1942. The wiener filter estimates the actual states of a system by minimizing the root mean square of the difference between the actual and the desired output. It is most suitable for stationary processes. Applying the wiener filter to time varying processes is very difficult. The Kalman filter is by far the best linear estimator there is. The Kalman filter estimates the correct states of the system in the presence of disturbances and measurement noise. It even has the ability to estimate the states of a system that cannot be fully modelled or precisely modelled. It can closely predict past, present and even future events

4.2 Kalman filter

The Kalman filter is a recursive solution to discrete-data linear filtering problem. It has been named after its developer Dr R E Kalman and it was developed in 1960. It is a set of mathematical equations that provides a recursive means of estimating the states of a process (Welsh and Bishop, 2006). There are two Kalman filters, the first one is a basic Kalman filter and the second one is an Extended Kalman filter. The Basic Kalman filter works with linear systems, and the Extended Kalman filter works with non-linear systems. Below is a picture of how the Kalman filter works.



The Kalman filter is a linear optimal observer; it uses all the information given to it to compute the best estimation of the state variables. The performance index of this optimal observer is the error covariance. The object is to minimize the error covariance, which is minimizing the mean squared error in the state estimates. The Kalman filter provides the best estimate of the states in the presence of measurement noise and process noise. It works as filter that filters off the noise from the sensors and the process inputs. The kalman filter can be steady-state or changing with time. The time varying kalman filter computes the optimum observer gains each time the filter is updated (Ledin, 2004). The result is an optimal estimate of the state at every step.

4.3 The Discrete Kalman Filter

A kalman filter generally works with a discrete-time time process that is governed by the linear stochastic difference equation;

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{4.1}$$

And a measurement equation;

$$z_k = Hx_k + v_k \tag{4.2}$$

The random variables w and v are the process and measurement noise respectively. These random variables are assumed to have a normal probability distribution with mean zero and covariances Q and R respectively. They are also assumed to be independent of each other or white noise.

The kalman algorithm works in two steps, in the first step the algorithm predicts the state estimates forward in time. That is the algorithm make a prediction of the state estimate of time t, before a measurement at time t is taken. The estimate from this first step is called the 'a priori' state estimate. The set of equations used in the first step are called the time update equations. The next step is to get feedback from the sensors and then update the 'a priori' state estimates with the feedback from the sensors. The updated state estimate is called the 'a posteriori' state estimate. The 'a posteriori' state estimate is a linear combination of the 'a priori' state estimate and the measurement update from the sensors. The following equation is the equation of the 'a posteriori state' estimate. The kalman filter goes through this cycle of predicting the state estimate forward in time and updating the state estimate with the measurement obtained from the sensors. The ongoing cycle of the algorithm is shown below,

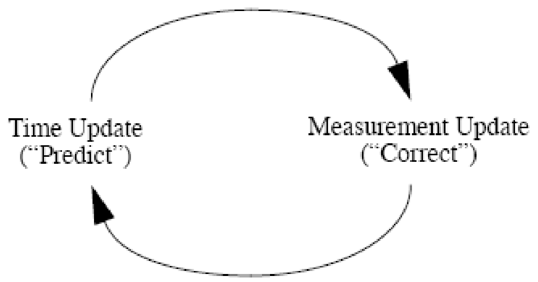


Fig 4.1

A summary of the steps involved in deriving the Kalman filter equations and the gain matrix K that minimizes the error covariance is as follows. Firstly the equations are separated into the time update and the measurement update equations as shown in the figure 4.1 above. The time update or predictor equations are as follows;

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{4.3}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{4.4}$$

Equation 4.3 calculates the state estimate of the state variable x_k. The A and B matrices are the states and the input matrices respectively. Equation 4.4 calculates an estimate of the error covariance matrix. This is the error between the true state variable x, and the estimate of x. The Q in equation 4.4 is the process noise covariance matrix. The next set of equations is the measurement update equations:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{4.5}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{4.6}$$

$$P_k = (I - K_k H)P_k^- \tag{4.7}$$

Equation 4.5 calculates the gain matrix that minimizes the error covariance P. The H and R matrices are the measurement matrix and the measurement covariance matrix respectively. Equation 4.6 is computes the 'a posteriori' state estimate or the measurement update estimate. The 'a posteriori' state estimate is a linear function of the 'a priori' state estimate and the weighted error between the measurement and the 'a priori' state estimate. The last equation 4.7, computes the 'a posteriori' covariance matrix.

4.4 The Kalman Filter and Sensor Fusion

The following section briefly discussed the Kalman filter as it is used for sensor fusion. The section attempt show how the Kaman filter performs sensor fusion in case of redundant sensors. The Kalman filter fuses measurement from sensors according to covariances. The measurement with a high covariance has little effect on the final state estimate. The equation for the kalman optimal observer gain K or L can also be represented in the form of equation 4.8 below.

$$L(t) = P(t)C^T R^{-1} \tag{4.8}$$

Where the P matrix is error covariance matrix, C is the measurement matrix and R is the measurement noise covariance matrix. Expanding out the above equation we get equation 4.9 below.

$$L(t) = P(t)C^T \begin{bmatrix} R_{11}^{-1} & & & \\ & \bullet & & \\ & & \bullet & \\ & & & R_{nn}^{-1} \end{bmatrix} \tag{4.9}$$

L(t) is a gain matrix, which is a column vector when R is diagonal. Each entry of the gain vector L is computed from the corresponding entry of the inverse matrix of the measurement covariance matrix. When a measurement covariance is big its inverse will be small and the resulting gain will also be small. Therefore measurements with large covariances are weighted less than those with small measurement covariance. From equation 4.6 the measurement update equation or the ‘a posteriori’ equation of the state estimate is as follows;

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-)$$

From this equation it can be seen that the state estimate correction for measurement is weighted by a gain K, which is calculated from equation 4.9. For noisy measurements, this gain will be small and the measurement correction will not have a big impact on ‘a priori’ state estimate. At the extreme, if the measurement is too noisy that the corresponding gain from equation 4.9 approaches zero, the state estimate will approach the ‘a priori’ state estimate.

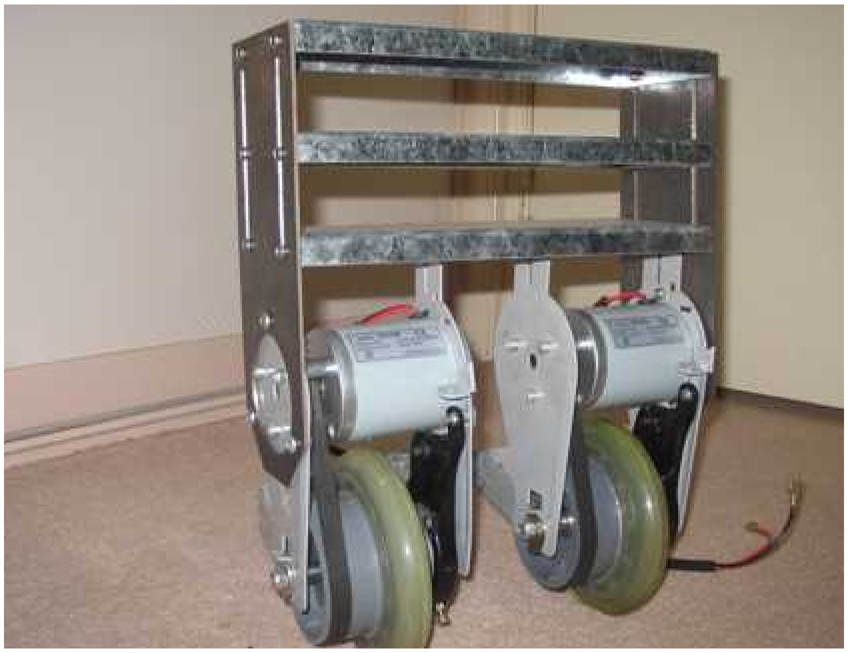
Chapter 5

5.1 The Robot Hardware

This chapter discusses the robot hardware which includes the robot chassis, the drive system, actuators, sensors and the controller.

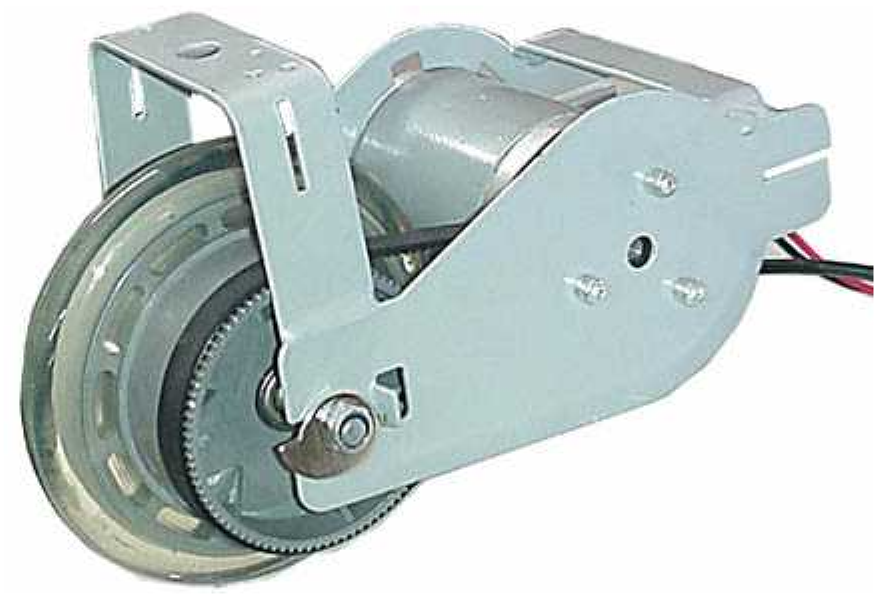
5.2 Chassis

The robot chassis is build of steel plates. There are two side plates which have slots where three more plates between the two side plates are held. The three middle plates form three platforms which hold the circuitry of the robot and actuators. The height of the platforms can be adjusted by moving the plates up and down the slots. This is down so as to adjust the height of the centre of mass of the robot and workout height for smoother control of the robot. A third small wheel was initially put in the robot to hold the robot up before the final control was implemented. The body of the robot has got a rectangular shape of length 29 cm and width of 10 cm. The robot has a height of 37 cm. The spacing between the platforms is adjustable and the platforms can be reduced or increased as required. The height of the robot is fixed to the length of the two side plates. The height can only be increased by replacing the side plates with longer plates. Below is a picture of the robot chassis;



5.3 Drive System

The robot uses a scooter rear drive assembly. The assembly consists of 100 W motor, a toothed belt and a wheel with an axle, bearings and pulley. The motor is a 24 V DC motor which runs happily and produce great torque at 12V DC. To of these assemblies are used and are held together by the side plates and the platforms of the chassis. Below is a picture of the drive assembly.



The assembly was bought from Oatley Electronics.

5.4 Actuators

As mention is the previous section the robot run on two l00 W motors. The motors have a rated speed of 2500 rpm and a rated current of 6A. The motors operate on 24V direct current but for the purpose of running the robot a 12Vdc battery is used. The battery used is a sealed lead acid battery. Below is a picture of the motor that run the robot;

5.6 Sensors

The robot uses three sensors which are mainly for balancing the robot. The sensors are the rate gyro, axis accelerometer and inclinometer. The first two sensors are for measuring the tilt rate while the third sensor is for measuring the tilt angle. Two sensors are used for the tilt rate mainly to provide redundancy and hence improved precision. Furthermore the accelerometer provides the static tilt information, when the robot is not accelerating and the gyroscope provides the dynamic tilt information. The gyro also has a drift problem and the accelerometer works to correct that.

5.61 Rate Gyroscope

The rate gyro used is an ADXRS300 single chip gyro. The output of the sensor is voltage proportional to angular rate about the z-axis. Clockwise rotation is positive and anticlockwise notation is negative. The sensor operates on 5V dc.

5.62 Accelerometer

The accelerometer used is an ADXL213 dual axis accelerometer with signal conditioned, duty cycle modulated outputs. The outputs are digital signals whose duty cycles are proportional to acceleration. The duty cycle outputs can be duty measured by a microcontroller without an A/D converter. The sensor operates on 5V DC.

5.63 Inclinometer

The tilt sensor used is an Accustar single axis sensor. This sensor is a capacitance based sensor, when rotated about its sensitive axis the sensor produce a linear variation in capacitance. The capacitance is electronically converted into angular data. The sensitive axis of the sensor is the z-axis, or the axis perpendicular to the sensor. The sensor has a range of ±60°. The sensor operates on 9Vdc and has a ratiometric output. The output is supply dependant. The midscale output, zero degrees, is half the supply voltage while the scale factor is also supply dependant. Clockwise rotations are positive and anticlockwise rotations are negative. Below is a picture of the tilt sensor.

5.7 Microcontroller

The microcontroller used in this project is the Motorola MC68HC912D60A. This microcontroller is a member of the 16 bit Motorola microprocessors family famously known as the HC12 microcontrollers. The microcontroller has 60k bytes of flash memory, 2k bytes of RAM, 1K byte of EEPROM, 2 asynchronous serial Communication interfaces (SCI) and a serial communication interface (SPI). Other peripherals include an enhanced capture timer, two 8 channel, 10-bit analogue-to-digital converters and a four channel pulse-width modulator (PWM). The two most important peripherals used in this project are the analogue-to-digital converter and the pulse-width modulator. The 16 bit CPU of this processor affords better processing power than the more common 8-bit processors. With this 16-bit controller implementation of floating point mathematics for the purpose of computations is used without the concern of depleting computing resources such as on-chip memory. A smaller 8-bit micro processor would restrict computations to fixed-point mathematics to try and reserve the small memory available in the microprocessor. This chip does not have an integrated digital-to-analogue converter and the pulse-width modulator is used for the purpose of producing analogue voltage to run the motor. Therefore the need for a digital-to-analogue converter is eliminated.

Chapter 6

6.1 Classical Control Methods

Classical control theory is the older of the linear control theories. This theory was developed in the early 19th century. The classical control methods are best suited for single input single output system and they include the root locus and the frequency response methods. Even though the root locus is suited for single input single output systems it can be used to great effect to analyse the multiple input multiple output systems. The root locus is one of the methods that have been used in this project to analyse the robot system

6.2 Modern Control Methods

The modern control methods of linear systems design are relatively new compared to the classical control methods. This class of methods include the state space design and the state space design method and the optimal control methods. There are specially suited for system of multiple input and outputs. The state space technique include the pole placement method, this method affords the designer the flexibility of being able to place the close loop poles anywhere that they want. The method is much easier to use than the classical control method. Choosing the best possible locations for the poles is not easy though, especially for higher order systems. The optimum design method is superior to the pole placement and is discussed in the next section.

6.3 Optimum Control

Optimal Control means developing the best controller according to a given performance specification. An optimum controller is the best controller that satisfies the given performance criteria. There are a number of different performance criteria that can be used. There is the minimum time performance criterion, where the best controller is the one that drives the states to zero or to a target position as fast as possible. Another performance criterion is the minimum energy criterion. According to this criterion the best controller is the one that drives the states of the system to the target as fast as possible but with minimal control effort. This criterion seeks for the best compromise or trade of between speed and control effort. Any linear feedback system is optimal in the sense that it minimizes the integral of a quadratic function of state and control variables whose weighting factors have been chosen appropriately (Greenside, 1970). The particular linear optimum controller is called the Linear Quadratic Regulator.

Chapter 7

7.1 Linear Quadratic Regulator

The linear quadratic regulator is the optimum controller that satisfies the following scalar cost function or the performance index.

$$J(u) = \int_0^\infty x^T Qx + u^T Rudt \tag{7.1}$$

The control method involves finding the closed loop gain matrix K that minimises the performance index. After finding the gain matrix K, the closed loop pole locations are found. The pole locations that results from this method are the best pole locations that could be found. This method involves finding the control law that drives the states of the system as fast as possible at the lowest control force possible. It finds the best compromise between the speed and the control force. This is a very important attribute of the linear quadratic regulator. It ensures that actuator saturation does not happen. It also ensures that the system is not driven too hard and out of the region where the linear approximation of the system holds. For the robot system, too much control force would tilt the robot too far and it wouldn’t be able to return it back to the balanced position.

The trade off between response speed and the control effort is determined by the weightings in the performance index. These are adjusted by the user as required. When the designer wants a bit more response speed he makes the weighting of the state variables small. Making these weightings big would slower the response speed. Similarly, to reduce the control effort the designer would weight the control variable a bit more heavily. A small weighting for the control effort would allow for more control effort.

Chapter 8

8.1 Control System Design

To analyse the system and design a control system matlab and matlab control system toolbox were used. Using matlab and matlab control system toolbox simplified the task of analysing the problem and designing a control system for the problem. This is mainly due to the fact that there are a lot of control systems commands both in matlab and the control system toolbox. Therefore a lot of problems can be solved by using a single command rather than having to write a program to solve the problem. Another reason is that there are a lot of software modules written in matlab available in the internet, which are useful in control systems design. With a bit of luck a relevant software module for the task at hand can be found in the internet and with a little bit of modification the module can be used. In the design and analysis stage two control methods were tried. The first method to be tried was the root locus method.

8.2 Root Locus Technique

The root locus technique falls within the classical domain of control system techniques. It is a classical presentation of the closed-loop poles as a system parameter is varied (Nise, 2004). To start the analysis of the problem with the root locus technique a plot of the open loop poles is made. Figure 4.1 shows the root locus of the open-loop system.

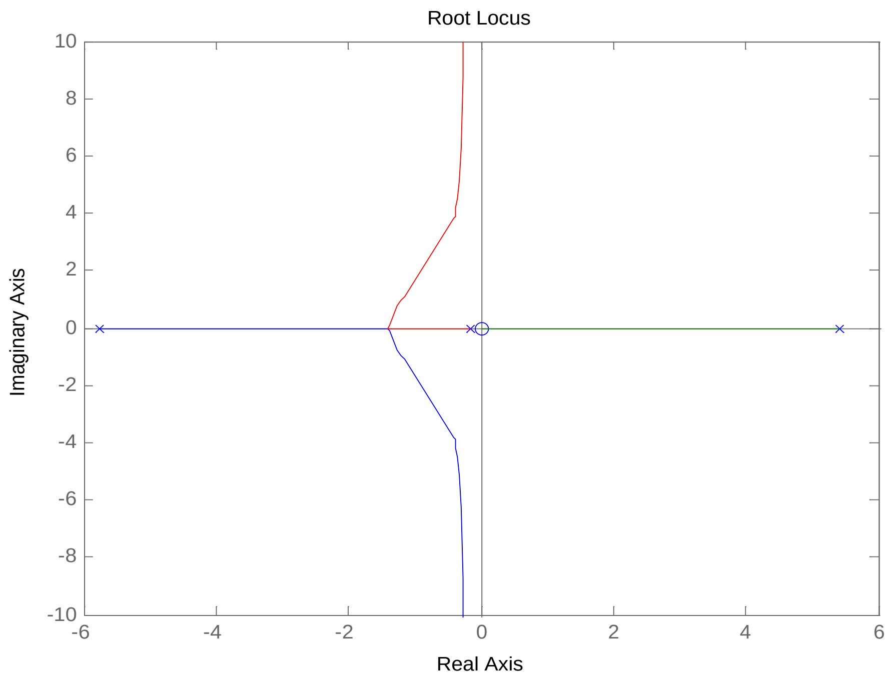


Fig 8.1

The system has four poles and two zero. One of the poles is next to -6 and the other is just next to the origin. Another pole is at the origin and the last one is next to 6. The zeros are at the origin. The system is not stable because it has a pole in the right pane of the complex plane.

Figure 8.2 below shows the open loop response of the unstable system

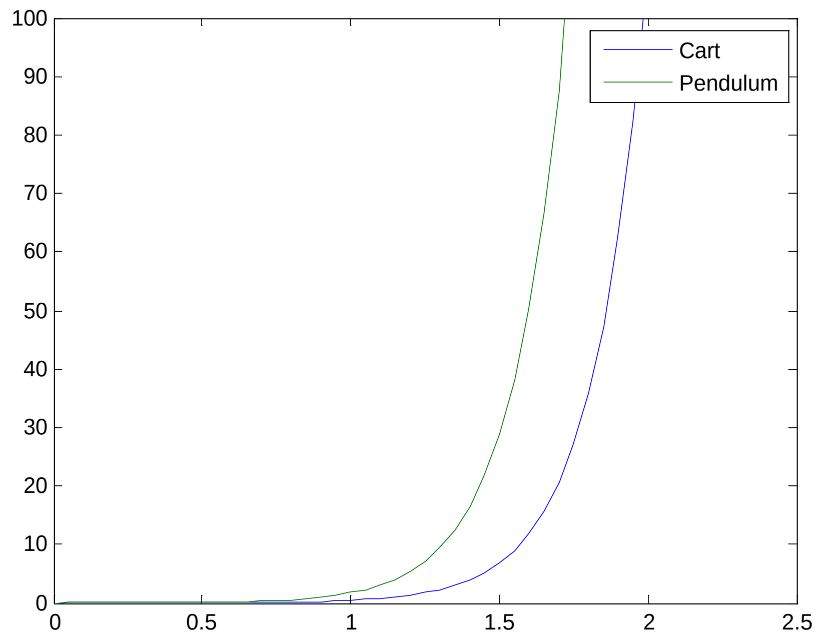


Fig 8.2

8.3 Controller Design with the Root locus

In controlling this particular system with the root locus, the objective is to pull the branch of the roof locus that is in the right side of the complex plane into the left side. This can be done is adding the correct mix of poles and zero to the system to pull the branch into the left side. However controller design with the root locus technique does not support the design of systems for SIMO systems. SIMO systems are single input, multiple output systems. Therefore to be able to design a controller for the robot system, the system would have to be a single input multiple output system. The robot system can be made a SIMO system by controlling just one output. Therefore the controller will only be concerned with balancing the robot and not controlling the movement of the robot on the floor. When the controller is concerned with just balancing the robot, the controller can be implemented using the root locus technique.

To produce the root locus shown in figure 8.1, one pole at the origin cancelled one zero at the origin leaving one zero, the pole out near -6 and the pole near the origin move toward each other and they meet somewhere next to -2. These poles then break away from the real axis and move in opposite directions, asymptotically to the complex axis. The pole on the right side of the complex plane moves towards the zero at the pole origin and terminates there. To pull the branch on the right side of the complex a number of steps are followed. The first step is to add a pole at zero. The pole at zero changes the root locus to the one shown I figure 8.3.

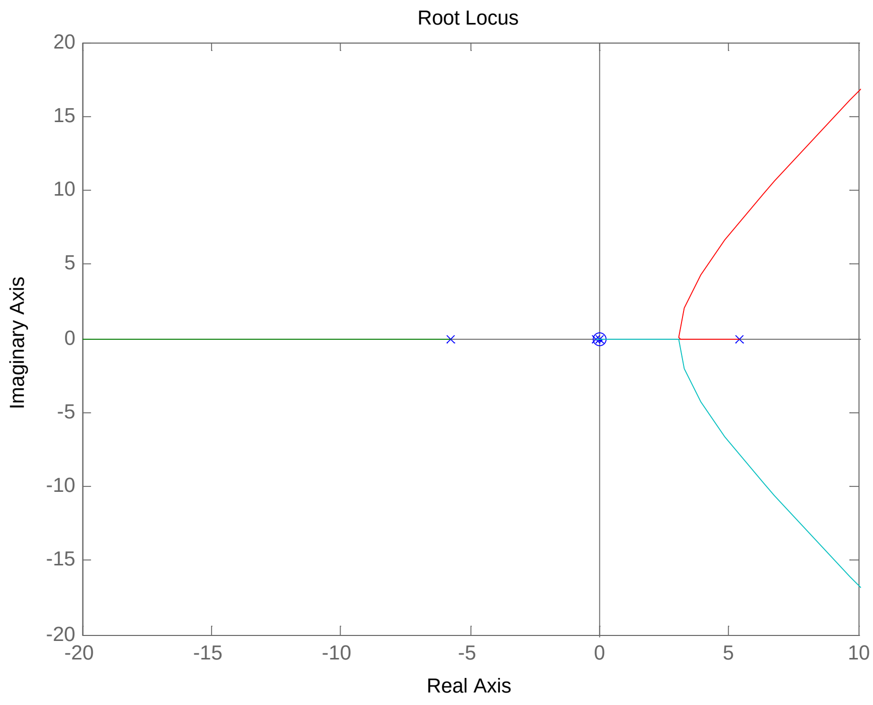


Fig 8.3

The added pole cancels the zero at the origin and new root locus is formed. The pole next to the origin moves to the right towards the pole in the right side of the complex plane, the pole on the right side moves inwards to meet it. These two poles meet somewhere around 3 and they break away as shown in the figure above. The pole in the far left moves out in the negative direction towards infinity. The next step is to pull the branches of the root locus to the left. To do this, a lead-lag compensator is implemented.

The compensator is implemented by adding a zero next to the pole at the origin but to the left of the pole. A pole is added together with this zero and is placed between the zero and the pole out at -6. Next, another zero is added between the recently added pole zero pair and a corresponding pole is also added but it is placed further out in the negative real axis. The result is a lead-lad compensator and the root locus is shown in figure 8.4 below.

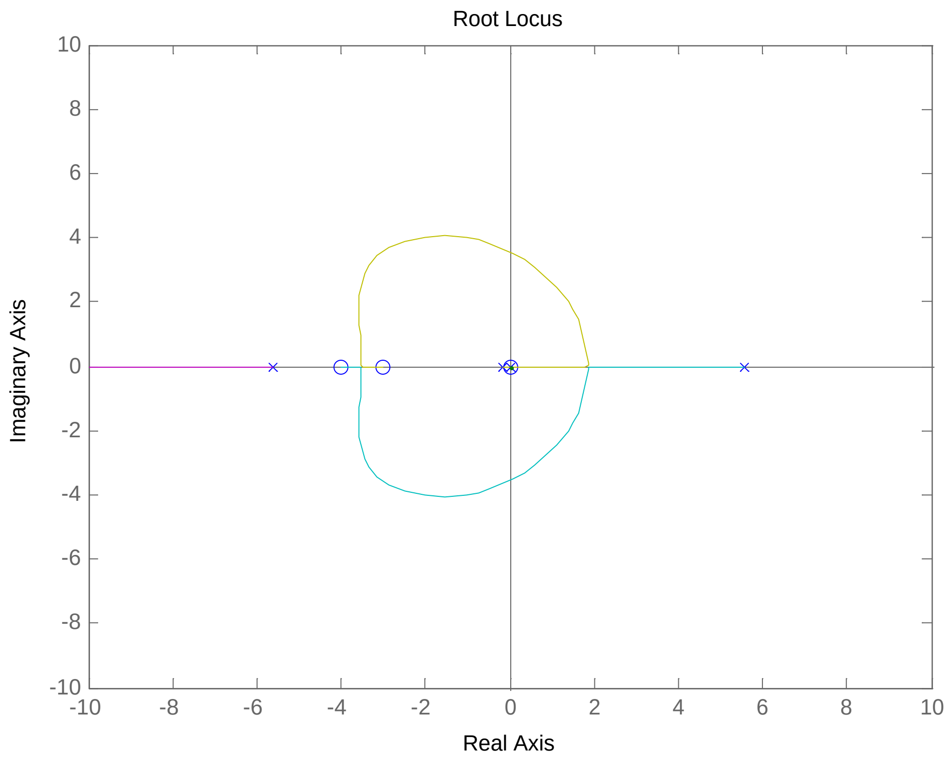


Fig 8.4

8.4 Linear Quadratic regulator Design

As already mentioned in the previous chapters, the linear quadratic regulator operates by working out closed loop pole locations that are the best pole locations that satisfy a given performance criterion. The performance criterion for the linear regulator is as follows;

$$J(u) = \int_0^\infty x^T Qx + u^T Rudt \qquad (8.1)$$

The objective is to find a control law that minimizes the performance criteria. To design a linear quadratic regulator in matlab, a matlab control system toolbox command lqr() is used for LQR design in continuous-time. With this command, the work is mainly in the selection of appropriate Q and R weighting matrices. Given the condition that these matrices should be positive semi-definite, it is best to set up these matrices so that only elements along the diagonal are none zero and nonnegative. The best starting point is to initialize both Q and R matrices as identity matrices. Setting up these matrices as identity matrices weighs each input and state variable equally. However, because there is only one input to the system, the R matrix reduces to a constant. After setting up the weighting matrices it is just a matter of calling the lqr() command with the linear plant model and the weighting matrices as the input arguments. The command computes the gain matrix K which is then used to compute the closed loop pole locations. When implementing the regulator, an assumption that all states are available for feedback is made. After the poles have been calculated it is essential to evaluate the performance of the resulting controller.

To evaluate the performance of the resulting controller another matlab function called plot poles was used. This function was written in 2003 by Jim ledin, and it simply plots the new poles of the system together with the performance constraints. Below is a plot of the closed loop poles potted using the plot poles function.

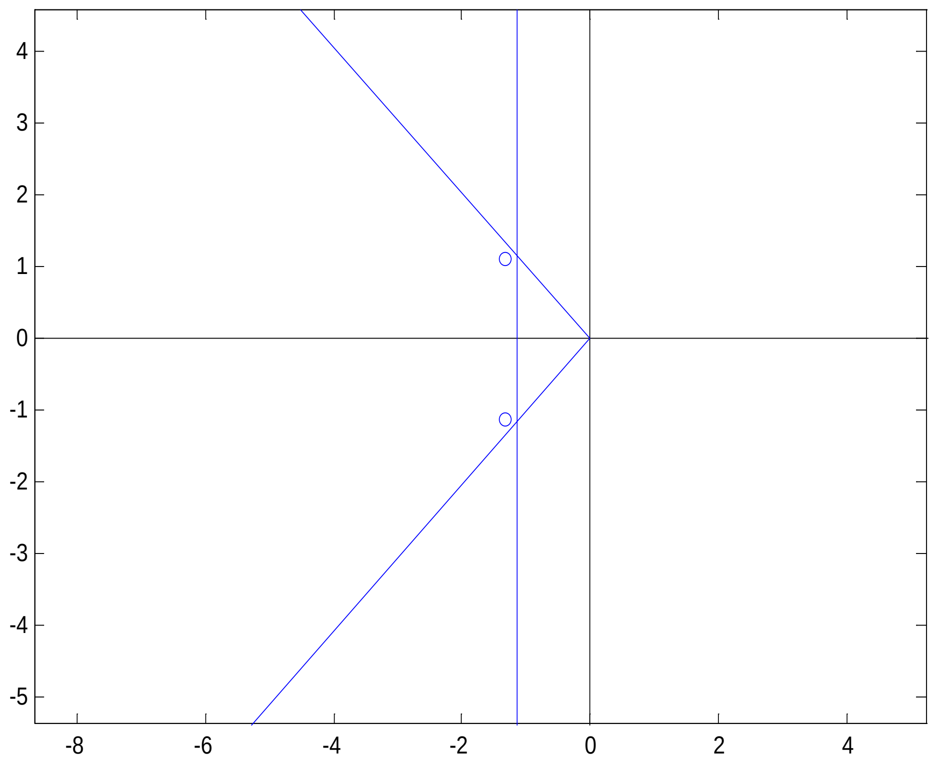


Fig 8.5

The diagonal lines on the plot are the settling time constraints and the vertical line is the damping ration constraint. The first plot shows all closed loop poles and the second plot shows close view of the two poles near the origin. The new closed loop poles are supposed to lie within the constraint lines on the plot, that way the closed loop system is meeting performance specifications of both the settling time and the damping ration. From the two plots it can be seen that there is a complex conjugate pair of poles next to negative one and there is another pair at minus sixty. To get the system to meet the performance specification, the diagonal element of the weighting matrix Q were iteratively adjusted until the desired performance was obtained. Different state variables were assigned different weights depending on which states were supposed to react faster than the other to get the system to stabilize. For example the wheels should move in the direction that the robot is falling faster than the rate at which the robot is falling so as to keep the robot balance. Therefore the robot's horizontal speed was given a higher weight than the robots tilt rate.

Plotting the poles to see whether they lied within the constraints of the performance specifications was not sufficient for evaluation of the controller. Another method was employed to balance the response speed and the control intensity used. The problem that arises is that when the state variables are made to response too fast, the actuator has to provide greater force to make that happen. However when the system is driven too hard to meet the response speed constraint there is a great likelihood that the system will be driven outside its linearity region. This means that the linear approximation of the system only works within the small region where the linearity condition holds. Outside this region the system becomes non-linear and the controller cannot control the system. Therefore it is imperative to avoid too much force so that the system is not driven outside the linear region. To accomplish that, as the weights of the matrix Q were adjusted, the response of the system was monitored. The limit of the tilt angle was set at $0.1\,rads$, therefore it was made sure that the amplitude of the tilt curve does not go beyond this limit. Beyond this limit the robot would topple over. The performance constraints of settling time and damping had to be relaxed a bit so that the control force is kept within allowable limits. To limit the amplitude of the tilt angle to less than or equal to 0.1 rads, the settling time constraint had to be set at 4s and the damping ratio at 0.7. The following diagram shows a plot of closed loop poles when the amplitude is limited to less than 0.1 but the settling time constraint is set lower at 3s and the damping ratio is increased to 0.8.

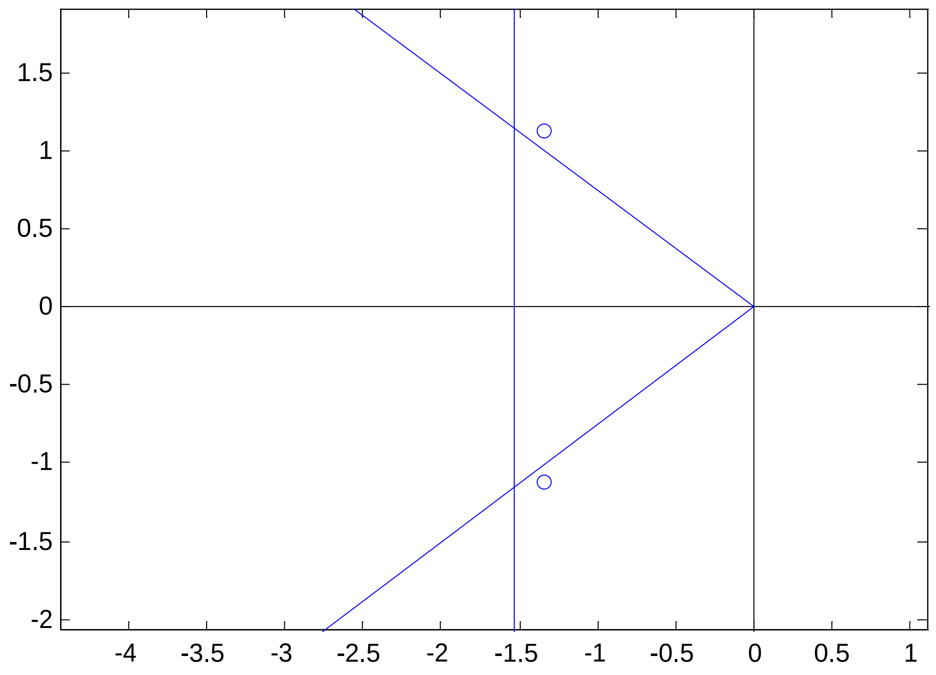


Fig 8.6

As can be seen from the plot the two closed loop poles nearer to the origin no longer fall within the specified constraints. Therefore the desired system response was obtained after a number of iterations to get a correct compromise between the response speed of the states and the control effort. This is a very desirable property of the linear regulator control design which helps to avoid saturation of the actuator or clipping of the control effort

8.6 Optimal Observer Designer

When designing the optimal controller which is the linear regulator, it was assumed that all states variables were measurable or measured. However not all states are measured and hence those that are not measured need to be estimated. The optimal observer used is the kalman filter. As stated in the chapter about estimation and estimation methods, the kalman filter works as both a filter and an estimator. It does not only estimate those states that are not measured or that we don’t have sensors for but it also estimates the true states of the measured variables. This is because even though thee variables are measured, there are sensor measurements are corrupted by noise and inaccuracies in the model of the system. To create the kalman filter, a matlab control toolbox command called kalman() is used. To use the kalman command the plant model has to be represented as linear stochastic model rather than the deterministic model. The stochastic model includes the process noise model and the measurement noise model together with their respective covariance matrices. Given the plant stochastic model as its input argument, the command computes the optimal observer gain matrix L.

8.61 Linear Stochastic Model

The plant linear stochastic model is shown below;

$$\dot{x} = Ax + Bu + Gw$$
$$y = Cx + Du + Hw + v$$

This model includes the G, H, w and v matrices. The G and H matrices represent the process noise model. The process noise model, models the disturbance inputs to the plant. These disturbance inputs are random inputs such as friction which have not been included in the plant model. Producing a realistic disturbance noise model is not always easy, as sometimes there is not enough information to model all the disturbance inputs to the plant. In such a case simplified model is used and this is the case with this system. The simplified model assumes that a disturbance input is added to the plant with every controlled in put to the plant. This is a crude assumption but one that produces the required results. Therefore under this assumption the G matrix equals the B matrix of the plant and the H matrix equals the B matrix of the plant. The measurement noise model, represented by matrix H, models the measurement noise. The w and v matrices represent the process noise covariance and the measurement noise covariance respectively. These matrices are represented as the Q and R matrices in matlab, the Q being the process noise and the R being the measurement noise. The Q matrix is an r x r matrix, where r is the number of plant inputs. The diagonal elements of Q represent the corresponding noise variance on each input. The matrix can be initialized as an identity matrix and the diagonal elements adjusted to get a satisfactory observer performance. For the robot system, the Q matrix is a constant since there is only one input to the plant. The measurement covariance matrix R is also a diagonal matrix, with each element along the diagonal containing the variance of the error in the corresponding sensor measurement. The error can be determined from the manufacturer-provided data describing the sensor error characteristics. The error variance is computed by squaring the root mean square jitter error specification from the sensor manufacturer's data sheet. The units of each diagonal must be the square of the units of the corresponding plant output.

The next stage in the design process is to implement the combined observer-controller for the plant. Below is a plot of the observer-controller poles of the system.

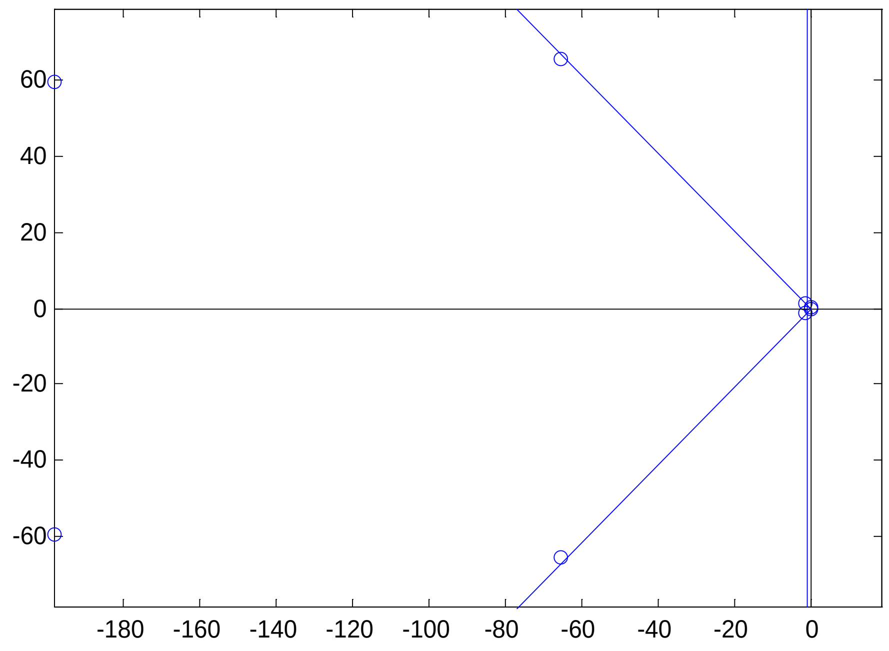


Fig 8.7

The left most poles in the diagram are the closed loop poles of the system and the next set of poles form the left are the observer poles. The other set of observer poles are near the origin but further right than the closed loop poles next to them. Below is a close view of the set of poles near the origin.

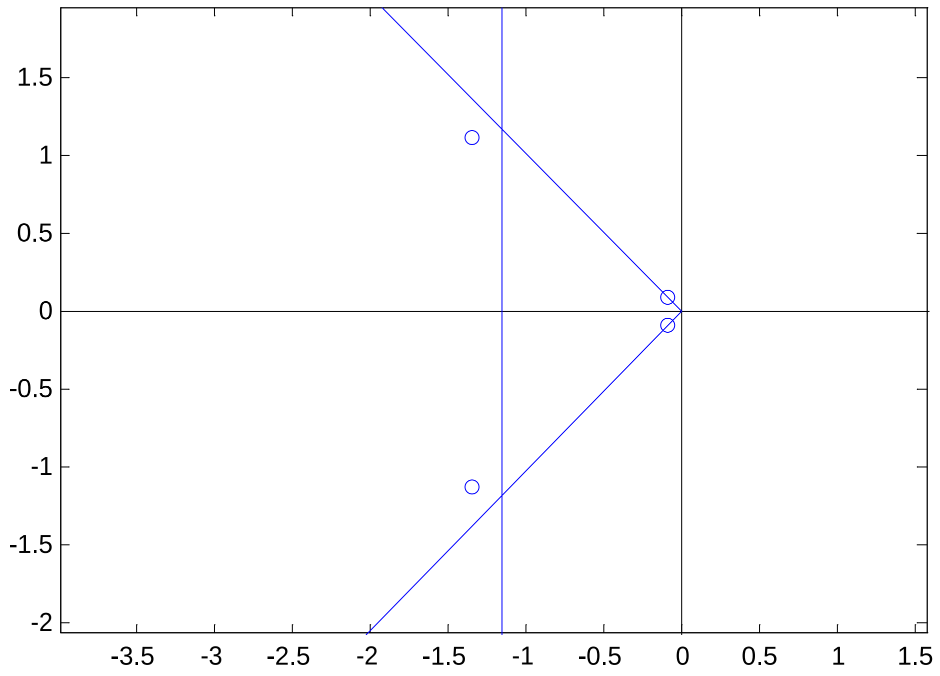


Fig 8.8

Next is a plot of the tilt response of the system with the observer-controller implemented. The system meets the specified performance specifications.

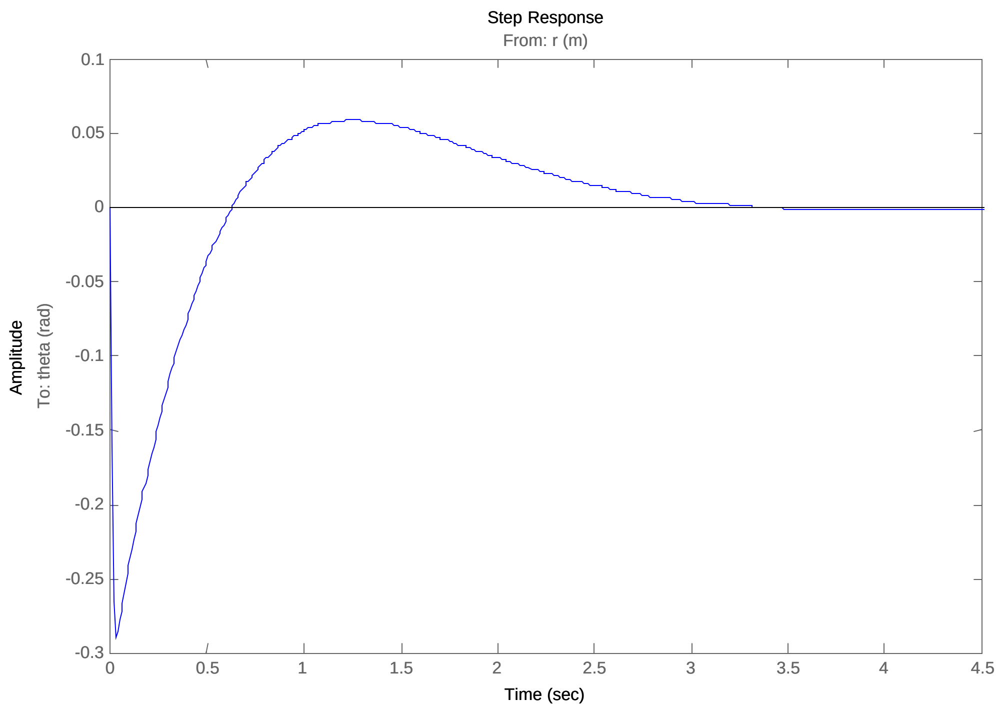


Fig 8.9

8.7 Feed-Forward Gain

The last part in the controller design process is to eliminate the steady-state error due to nonlinearities or modelling errors in the linear plant. There are two methods that can be used for this for this purpose. The first method is using the integral control method. This method is specifically useful in systems were the model is not very accurate or where there are modelling errors. The integral control here does exactly the same purpose as in the PID control method. The integral control portion of the PID eliminates the steady-state error of the system. The other method for eliminating the steady-state error of the system is using a feed forward gain. The feed forward gain is specifically useful in plants where the output is supposed to follow a reference input. However the method works well when there are no modelling errors or minimal modelling errors. The feed-forward gain method has been used for the robot system. This is due to the fact that the robot has to follow a commanded reference input in terms of the demanded robot position. The gain is selected to scale the reference inputs to produce steady-state error at the outputs. The following equation is used to compute the gain N;

$$N = N_u + KN_x \tag{8.2}$$

The K is the controller gain calculated from the regulator design method and the matrices N_u and N_x are determined form the plant model as shown below;

$$\begin{bmatrix} N_x \\ N_u \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ I \end{bmatrix} \tag{8.3}$$

The 0 matrix contains all zeros and I is an identity matrix. To calculate the feedforward gain in matlab I found a matlab function on the internet that and with a little bit of modification was able to get it work for me. The result is a system that meets both the settling time and damping specification and has zero steady-state error. The previous figure shows the amplitude response of the system with the feedforward gain implemented. Below is a figure that shows a plot of the response of the robot to a step input. The robot reaches its demanded position with minimal overshoot and it settles at the demanded position with zero settling error.

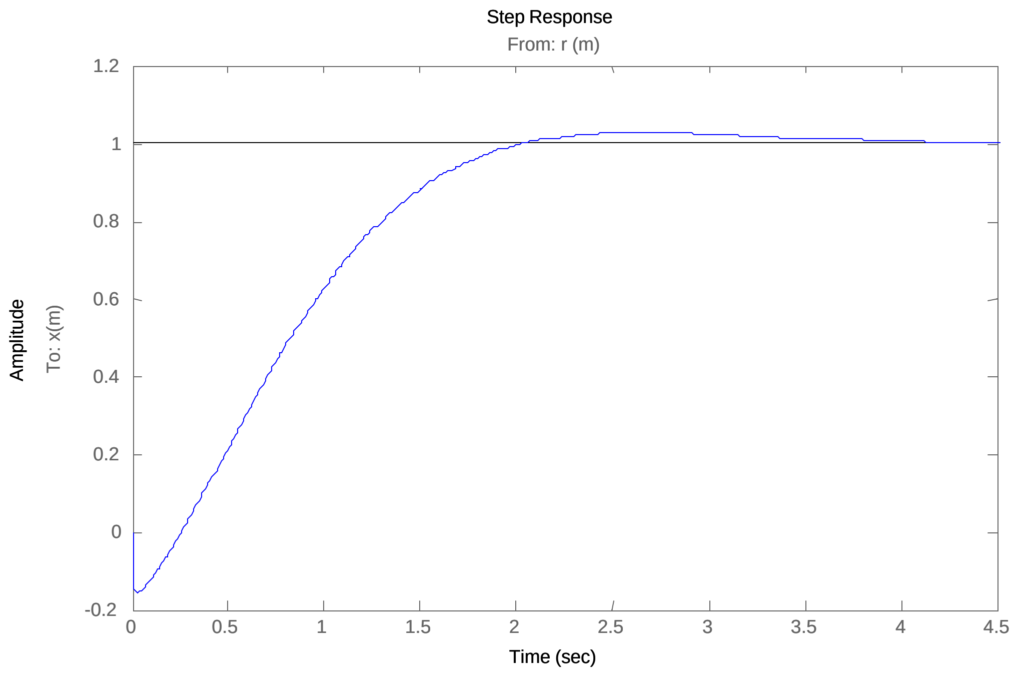


Fig 8.10

The optimum controller and observer have designed in the continuous-time domain. However the controller of the robot is going to be ran in a microcontroller which means the controller has to operate in discrete-time domain. Therefore the controller has to be transformed into the discrete-time domain. It is possible to change o the discrete time domain right at the start of the design process and then working in that domain all through to the finish. This however could present some problems in the design process. The problem could rise when a wrong sampling period is chosen for the implementation of the discrete controller. The thing is that all the computations that are made in the design process are dependant on the chosen sampling period and if it turns out that the sampling period chosen is not the right one then all the computations would have to be repeated for the correct sampling period. This obviously would be a tedious process. Therefore transforming the controller to the discrete-time domain saves a lot of design time.

From the above discussion it is dear that selection of the sampling frequency is an important part of the design process. When the sampling period is too small the discrete system closely follows the continuous system but the microcontroller would have to do a lot of computations. This would require a more powerful micro controller with a bigger memory. Naturally, this means a more expensive processor. On the other extreme, when the sampling period is too big the discrete time controller’s performance would diverge from the continuous time controller performance. When the sampling period is too big it can even lead to excessive overshoot, oscillation or even instability of the closed loop system.

8.9 Choosing A Sampling Period.

From the preceding discussion it is obvious that choosing of a sampling period is very critical for the success of the controller. A very important condition for the selection of a sampling period is the nyquist sampling theorem. By the sampling theorem, the sampling frequency must be at least twice the highest significant frequency in the controller input signal to enable processing by the controller. To help in the selection of a suitable sampling period a methodical approach to the problem has been adopted. The method was adopted from the book called embedded control systems in c/c++ by Jim Ledin. In the book, Jim proposes the following steps;

1. Plot the step response and the frequency response If the closed-loop system.
2. Choose a very short sampling period that provides a good discrete-time system approximation to the continuous-time system performance.
3. Discretize the control system with the c2d() command and appropriate discretization method.
4. Plot the step response and the frequency response of the closed-loop system using the discretized controller in place of the continuous-time controller.
5. Increase the sampling period and repeat steps 3 and 4. Continue until the step response of frequency response of the system with the discrete-time controller diverges unacceptably from that of the continuous-time system.

The above steps give an idea of how big a sampling period the system can tolerate. However a more conservatively smaller sampling period is chosen to accommodate the nonlinearities in the system. The results of using the above method are given below. However I used the open-loop system responses instead of the closed-loop system responses. This is because my closed-loop responses did not give a clear picture that would help me to arrive at meaningful conclusions. The figure below shows the open-loop continuous and discrete-time responses for a sampling period of 0.1 milliseconds.

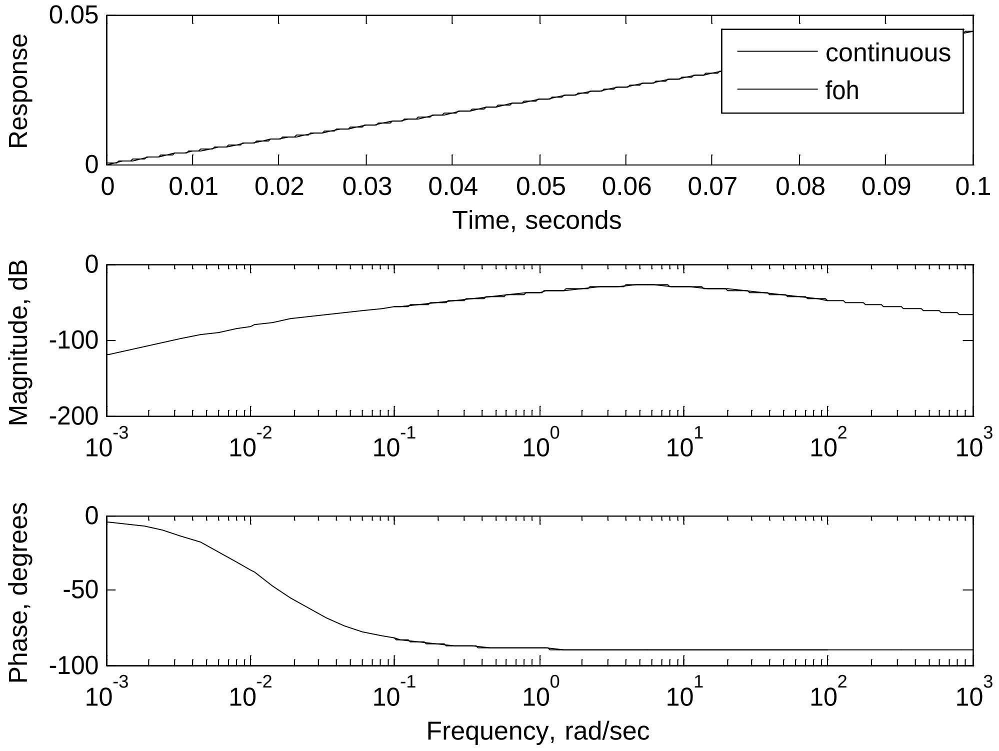


Fig 8.11

The discrete-system closely follows the continuous response and there is no divergence what so ever for the given frequency range.

The next figure shows both the frequency and step responses for a bigger sampling period of 1milliseconds. As can be seen from the figure, there still no divergence, the discrete-time system accurately follows the continuous- time system for the given frequency range.

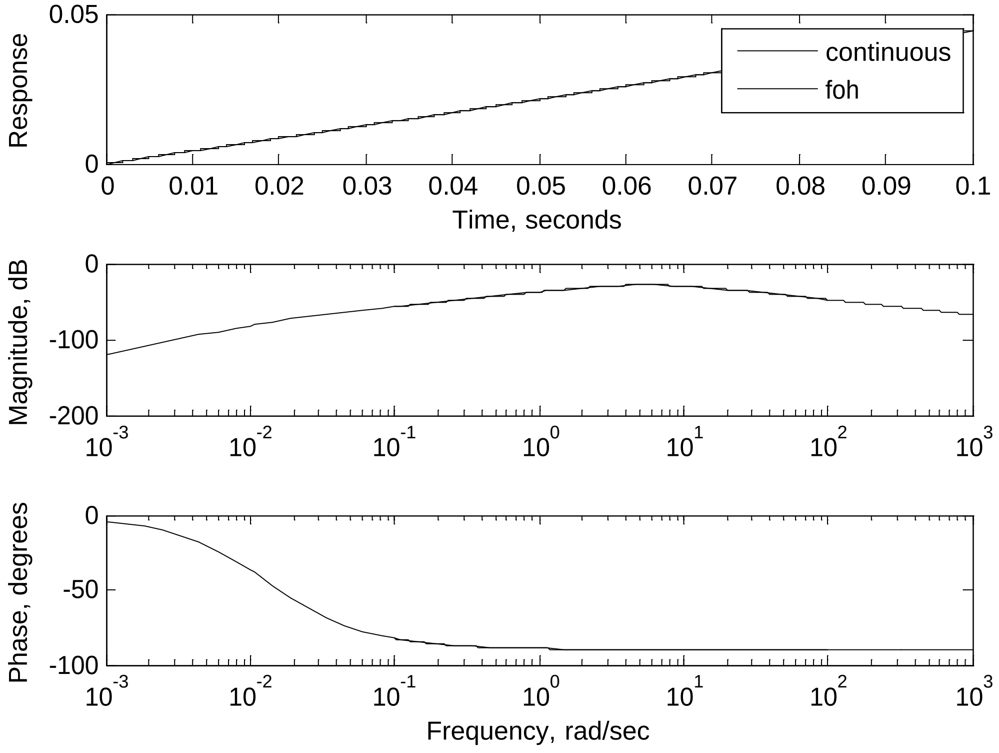


Fig 8.12

The last figure below show results for a 10 milliseconds sampling time. The results show a significant divergence of the discrete-time system from the continuous-time system. The frequency response shows a significant divergence even at low frequencies of a hundred hertz. The response dear shows that the sampling period has gone beyond allowable limits.

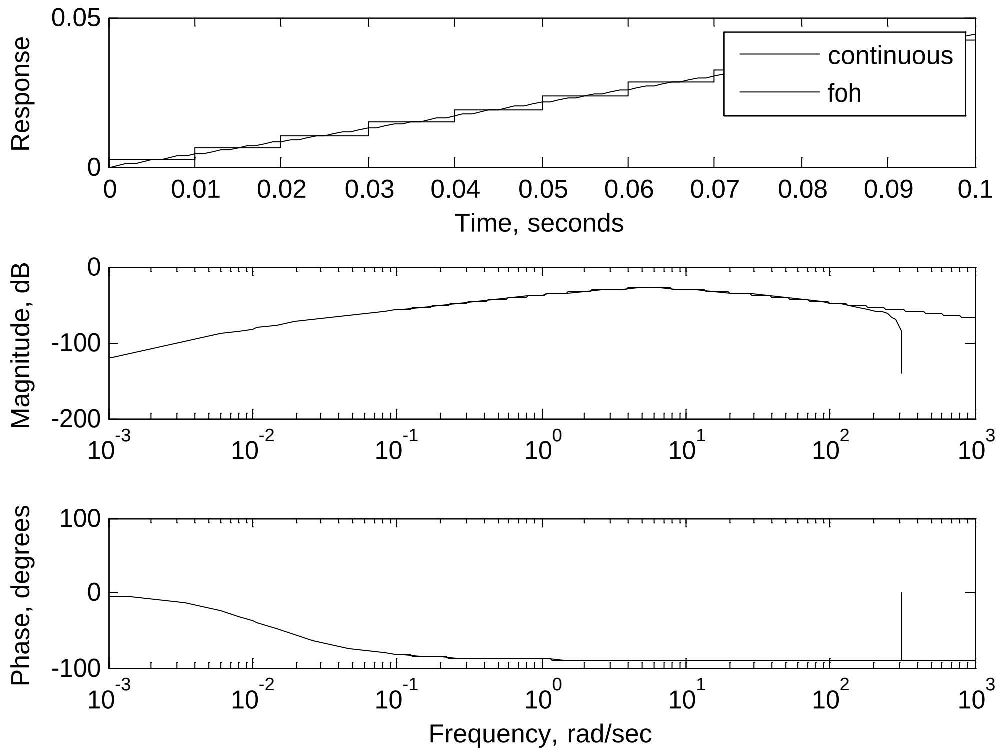


Fig 8.13

From the information given by these three figures, a sampling period of one 0.1ms was chosen. Even though a sampling time of 1ms appears to be fine, a more conservative option is chosen to try and accommodate for the nonlinearities in the system and hopefully to compensate for the fact that an open-loop system was used in the selection method instead of the close-loop system. Furthermore the 0.1 ms sampling time appears to be fine, given that the conversion time of the analogue-to-digital converter of the microprocessor is 10 microseconds. Hence the sampling period would not place an excessive performance requirement on the analogue-to-digital converter.

Chapter 9

9.1 Discretization Method

The last consideration in implementing a discrete-time controller is the discretization method used. There are a number of discretization methods available, each with its own advantages and disadvantages. The disretization methods available in matlab are the zero-order hold method, first-order hold method, the pulse-invariant discretization method, the bilinear approximation method, the bilinear approximation with frequency prewarping and lastly the matched pole zero method. The matched pole-zero method is only applicable for SISO systems as a result it us not considered for this problem. The following is a brief description of each method and their suitability.

The zero-order noted method is a general purpose method and the simplest of the remaining five methods. This method holds the input constant between sampling intervals. The method introduces a one-half sample time delay into the model. The frequency response of this method shows a significance divergence of the discrete-time response from the continuous-time delay even at low frequencies mainly because of the one-half sample time delay introduced into the model. The first order-hold method is also a general purpose method which is an improvement of the zero-order hold method. The sampling method uses a linear interpolation between samples instead of holding the input constant. This method does not introduce a half-sample time delay unlike the first method. As a result, the method exhibits an improved frequency response than the first method. The impulse-invariant method is used for single sample pulse inputs. The bilinear method uses an exponential function to relate the discrete continuous time domains (Ledin, 2004). The frequency response of this method is inferior to the frequency response of the first-order hold method. Lastly the bilinear method with frequency prewarping is for special cases where the there is a specific frequency where the linear and the discrete time frequency responses must match. Therefore the continuous and the discrete time frequency response match at the prewarp frequency.

From the above discussion the best two methods are the zero-order hold and the first-order hold methods because they are general purpose methods and do not need any special condition. Between the two methods the first-order hold method is superior and it is therefore the chosen method for the discretization process.

Chapter 10

10.1 Motor control

This section discusses pulse width modulation control of the motor and it also touches on the use of an H-Bridge amplifier for the bidirectional control of the motor.

10.2 Pulse Width Modulation.

Most microcontrollers do not come with integrated D/A converters in fact the hc12-D60A microcontroller used in this project does not have an integrated converter. Nevertheless, analogue output signals can be generated by tow-pass filtering a Pulse-Width Modulation signal. This is a very popular method of producing analogue signals in embedded systems. In fact a lot of microcontrollers come with integrated Pulse-Width Modulation units. The HC12D60A chip has four pulse width modulation units. The pulse-width modulation method produces an analogue output by setting the duty cycle of the pulse signal. The duty cycle is the percentage of the time t-high of the output waveform to the period of the output waveform. The produced analogue voltage is proportional to the duty cycle. For example if the source voltage is 12V and the duty cycle of the PWM is 50%, the produced output voltage would be 50% of the source voltage, which is 6V in this case. Therefore any analogue voltage level can be produced by setting an appropriate duty cycle. The PWM is able to adjust the speed of the motor by adjusting the duty cycle and hence the voltage supplied to the motor. Using the PWM method saves the cost of acquiring a Digital to Analogue Converter, especially that the microcontroller does not have it. Another benefit of using the PWM is that the signal remains digital and no digital-to-analogue conversion is necessary, by doing so the noise effects are minimized.

10.3 The H-Bridge Amplifier

The H-Bridge amplifier amplifies the signal from the pulse-width modulator channel to produce voltage that is sufficient enough to drive the motor. The amplifier use FET transistors for amplification. This amplifier also provides dedicational control of the motor. Under the command of the software the H-Bridge swaps the motor terminals to drive the motor in a different direction. Using the H-Bridge saves the cost of using two voltage sources for bidirectional control of the motor. The H-Bridge has got two logic inputs for the direction control of the motor.

Chapter 11

11.1 Hardware Configuration

To set up the hardware of the system for correct software control, the hardware needs to be configured. The two most important peripheral devices used in the system ore the analogue-to-digital converter and the pulse-width modulator.

The analogue to digital converter is used to interface the sensors of the system to the microcontroller. There are four sensors used for the robot system. All the sensors produce an analogue output proportional to the state measurement they are measuring. The analogue signals from the sensors need to be converted to digital signals for microprocessor processing. As a result the analogue-to-digital converter is used to interface the sensors and the microprocessor for this reason. For correct operation, the analogue-to-digital converter needs to be configured. The microprocessor has eight ADT channels and the four sensors are connected at channels 0 to 4. The ADT unit has a control register 2 associated to each channel. The most important bits in these registers are the AFFC, the DJM, the ASCIE and the ASCIF. The ASCIE is the Sequence Complete Interrupt Flag, this bit is set to enable an interrupt function to signal the MCU when the conversion is complete. This way the MCU can proceed with other operations and not have to keep checking the ADT to see whether a conversion is complete. The ASCIF is the Sequence Complete Interrupt Flag, this bit is set to enable the flag to be set when a conversion complete interrupt happens. This bit is cleared when the conversion complete interrupt is serviced. Last but not least the AFFC is the Fast Conversion Complete Flag Clear. This bit is set to enable the Sequence Complete Interrupt Flag to be automatically cleared by reading the conversion results from the data port. And finally the DJM is the Register data justification mode. The bit is set for either left justification or right justification of the data in the data register.

In control register 3 the analogue-to-digital Converter is set for normal operation by clearing bit 2 of the register. In normal operation the ADC conversion results a stored in data registers according to the sequence at which the conversions occur. The result of the first conversion appears in the first result register and so on. Bit 3 of the register, which is S1C controls the conversion sequence length together with bit 6 of control register 5. For the operation of the robot system these bit have been configured four conversions per sequence. In the control register 4, the resolution of the ADC has been set to 10 bit resolution by setting bit 7. The ADC clock is set to a quarter of the system clock by setting appropriates bits in register 4. The sample time is set to 2 A/D clock periods. The ADC is set for continuous conversion mode setting SCAN bit in control register 5. Lastly the MULT bit in control register 5 is set to allow the ADC sequence control to sample across many in a single sequence. Bits CC, CB and CA are all cleared to start sampling from channel 0.

The pulse-width modulation unit is also configured for correct operation. Channel 0 of the PWM is the one used in for driving the motors. Firstly bit 6 of the PWM clocks and concatenate register is cleared to operate channels 0 and 1 as separate 8-bit PWMs. Bits 3, 4 and 5 are set to 0, 1 and 1 respectively to prescale the channel clock to P/64. In the PWM clock select and polarity register, bit 4 is cleared to select clock A as the channel clock and bit 1 is also cleared to set the output low at the beginning of the period. Channel 0 is enabled in the PWM enable register by setting bit 0. Lastly the PWM control register bit 3 is set to operate the channel in centre aligned output mode. Bit 2 is cleared to allow port P pins to have normal drive capability and bit 0 is cleared to allow the PWM to continue in background mode.

Chapter 12

12.1 Encountered Difficulties

One of the biggest stumbling blocks that I encountered was scarce resources. The library here at USQ did not of a lot of the material that I needed for the methods that I choose to use for the robot. A lot of the books that I used were borrowed from the libraries of University of Queensland and Queensland University of Technology. Holding to these books for a substantial amount of time was not possible as most of them were just a week long loan books. Most of the books I borrowed were books on embedded systems and C programming. The other difficult that I met was implementing my controller in C programming language. Having not done C language programming before, I took a lot of time learning the language. In doing my project I decide to use a tilt sensor for measuring the tilt angle of the robot. Unfortunately there isn't a tilt sensor out there that is a cheap. I however decided to buy a tilt sensor but the sensor took long to arrive. I also intended to use an optical encoder to sense the speed of the robot but because of the housing of the motor and wheels it was impossible to mount the sensor. I lost a lot of valuable time a long the process and when it came time to integrate the software and the hardware I had to learn how to use the Introl C compiler. Not much information was available on the internet to help me with this. I ultimately failed to work the rest of the compiler operation mainly because there was not much time available to do an effective job of learning the compiler. As a result the final integration of the software and the hardware designs was not possible.

Chapter 13

13.1 Foreseeable Problems with my Robot System.

There are some few points about my system that I think may cause the robot not to operate well. The first problem is that I made an assumption that the dynamics of the motors would be faster than the dynamics of the robot body and hence respond quickly that the robot body. I therefore did not include the model of the motors into my system model. These could have serious drawbacks as I have mentioned.   To add on to that the motor constants were assumed. There was no information about the particular motors that I am using from the manufacture. I did find out an experiment of how to work out the motor constants but I couldn't get hold of the apparatus to perform it. The other problem could be the nonlinearities in the system. The motors run at 24V DC but I couldn't get hold of a lead acid battery of 24V. The last point that I think could present some problems is the weight of the robot. The chassis and the motors are heavy enough but the battery on its on weighs over 3kg.

Chapter 14

14.1 Risk Analysis

There are risks involved in the construction of the robot and its use. The first and maybe the biggest risk is in the building of the robot chassis. The chassis is constructed from steel plates. To machine these steel plates powerful machine with sharp cutting tools are used. These machines need skilled personnel to operate and they also need a lot of care. If not care is not taken when operating these machines, serious injuries may result such as loss of limbs. Another risk associated with robot construction is in dealing with a high voltage sources. The motors are operated from 24 V sources and even the lesser 12V sources are used, the risk is still high. The risk of damaging the electrical components, such as the H-Bridge and even the microprocessor itself is even higher. The FET-transistors used in the H-Bridge are particularly sensitive and easy to burn. Some of the electrical components used in the robot are very expensive, and the high cost makes the risk even higher.

During the operating stage of the robot, care needs to be taken as well. The complete robot and the battery weigh just below 10 kg and the body is made of sharp steel plates. The harm that this robot can cause on the people operating the robot if it accidentally falls on them could be great especially that the robot is heavy and the motors would have to use more power to stabilize the system. The robot could also damage the floor if it falls to the ground.

Lastly the robot must be disposed safely at the end of its life. The metals and the electronic components that make up the robot can be harmful to the environment.]

Chapter 15

15.1 Costs of Producing the Robot

The costs involve in the building of the robot are quite substantial. The greater costs are the costs of the Motorola MC68HC912D60A processor. The card12 costs around US$159. The other big cost was in acquiring the tilt sensor.   The tilt sensor cost about A$150. These are the two big costs, apart from that the drive system costs around A$100 and the battery was A$24. Other costs include the costs of building the body of the robot chassis, which include the labour costs of machining.

Chapter 16

References

Clark, R.N. 1996, Control System Dynamics, Cambridge University Press.

Dorf, R. C. 1989, Modern Control Systems, 5th edn, Addison-Wesley Publishing Company.

Nise, N.S, 2004, Control Systems Engineering, 4th edn, John Wiley & Sons

Stefani, S.T. et al, 1994, Design of Feedback Control Systems, 3rd edn, Sauinders College Publishers

http://www.tedlarson.com/robots/balancingbot.htm

http://www.engin.umich.edu/group/ctm/examples/pend/invpen.html.

http://www.geology.smu.edu/~dpa-www/robo/nbot/

Shinners S. M, 1998, Advanced modern control system theory and design, John Wiley and Sons

Ledin J, 2004, Embedded Control System in C/C++: An Introduction for Software Developers using Matlab, CMP Books

Zhou and Kemin, 1996, Robust and optimal control, Prentice Hall

Westphal L.C, 1995, Sourcebook of control systems engineering, 1st edition, chapman & Hall

Greenside L. A, 1970, Elements of Modern Control Theory, Mcmillian & Co

Lewis F. L, 1995, Optimal Control, 2nd,John Wiley and Sons

Siouris G.M, 1996, Optimal Control and estimation Theory; John Wiley and Sons

Friedland B, 1996, Advance Control System Design, Prentice Hall

Mohinder S & Andrews A, Kalman filtering: Theory and Practice using Matlab, 2nd, A Wiley-Interscience Publication

Welch G & Bishop G, 2006, An Introduction to the Kalman Filter

Bak T, 2006, Estimation and sensor Fusion.

Maybeck P.S, 1979, ‘Stochastic models, estimation, and control’, Academic Press

Freescale, Hc12 microcontrollers, technical data, 2005

APPENDICES

APPENDIX A: PROJECT SPECIFICATION

University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG 4111/4112 Research Project

FOR:	Kealeboga Mokonopi
TOPIC	Balancing a two wheeled robot
SUPERVISOR:	Mr Mark Phythian
SPONSORSHIP:	Faculty of Engineering and Surveying, USQ
PROJECT AIM:	The aim of this project is to design and building a two wheeled robot system,

PROGRAMME:

1. Literature Review
2. Producing a mathematical model of the system.
3. Suitable control system investigation
4. Control system implementation
5. Hardware design.

As time permits:

1. Control the robot to a predetermined position
2. Make the robot system to turn

AGREED
________________(Student), _______________(Supervisor)

___/___/___ ___/___/___ ___/___/___

APPENDIX B MATLAB CODES

```
====================================================================
% The code is used to determined the controllability of the system
%Produced by Kealeboga Mokonopi
%University of Southern Queens land
% 2006
%====================================================================

function controllabilitymatrix
% System Variables
 M = 0.5;
 m = 0.2;
 b = 0.1;
 i = 0.006;
 g = 9.8;
 l = 0.3;

 p = i*(M+m)+M*m*l^2;

 % Production of system amtrices
 A = [0      1              0        0;
      0  -(i+m*l^2)*b/p (m^2*g*l^2)/p  0;
      0      0              0        1;
      0 -(m*l*b)/p    m*g*l*(M+m)/p 0];
 B = [0; (i+m*l^2)/p; 0; m*l/p];

 Cm = ctrb(A,B)
Rank = rank(Cm)

====================================================================================================
==================================================================================================
% The code is used to determined the observability of the system
%Produced by Kealeboga Mokonopi
%University of Southern Queens land
% 2006
%====================================================================
function observabilitymatrix
%system variables and system amtrices
 M = 0.5;
 m = 0.2;
 b = 0.1;
 i = 0.006;
 g = 9.8;
 l = 0.3;

 p = i*(M+m)+M*m*l^2; %denominator
 A = [0      1              0        0;
      0  -(i+m*l^2)*b/p (m^2*g*l^2)/p  0;
      0      0              0        1;
      0 -(m*l*b)/p    m*g*l*(M+m)/p 0];
 B = [0; (i+m*l^2)/p; 0; m*l/p];
 C = [1 0 0 0;0 0 1 0];
 Om = obsv(A,C)
Rank = rank(Om)
```

```
==================================================================
% The function implements the continous time controller
% using the linear regulator and kalman observer
% Produced by Kealeboga Mokonpi
% USQ, 2006
function lqrcontrl
%=================================================================
% system variables and constants and system matrices
M = .5;
m = 7;
b = 0.1;
i = 0.523;
g = 9.8;
l = 0.25;

p = i*(M+m)+M*m*l^2;
A = [0      1              0            0;
     0 -(i+m*l^2)*b/p  (m^2*g*l^2)/p    0;
     0      0              0            1;
     0 -(m*l*b)/p       m*g*l*(M+m)/p  0];
B = [     0;
     (i+m*l^2)/p;
          0;
        m*l/p];
C = eye(4);
D = [0];
ssplant = ss(A,B,C,D);
%==================================================================

set(ssplant, 'InputName', 'Cart Force');
set(ssplant, 'OutputName', {'Cart Pos', 'Cart Vel', 'Pend Angle', 'Pend
Vel'});
set(ssplant, 'StateName', {'Cart Pos', 'Cart Vel', 'Pend Angle', 'Pend
Vel'});

% Design the controller gain
Q = diag([1e9 1e6 1e10 1e5]);
R = 1;
K = lqr(ssplant,Q,R);

%Compute the feedforward gain
Cn=[1 0 0 0];
N=feedforwardG(A,B,Cn,0,K)

cl_sys = feedback(ssplant, -K,+1)
% t_settle = 3;
% damp_ratio = 0.8;


% % Design the observer gain
a = ssplant.a; b = ssplant.b;
c = [1 0 0 0; 0 0 1 0; 0 0 0 1]; d = [0; 0; 0];

QN = 0.1^2;
RN = [0.01^2/12 0 0; 0 0.000017^2/12 0;0 0 0.0873^2/12];
```

```
g = b;
h = d;

obs_plant = ss(a, [b g], c, [d h])

[kest, L] = kalman(obs_plant, QN, RN)

% Create a state space observer-controller
ssobsctrl = ss(a-L*c, [L b-L*d], -K, 0)




% % Augment the plant model to pass the inputs as additional outputs
r = size(b, 2); % Number of inputs
n = size(a, 1); % Number of states
ssplant_aug = ss(a, b, [c; zeros(r, n)], [d; eye(r)]);


% Compute the feedforward gain
Cn=[1 0 0 0];
N=feedforwardG(A,B,Cn,0,K)

% Form the closed loop system with positive feedback
 sscl = N*feedback(ssplant_aug, cl_sys, +1);
 figure
plotpole(sscl, t_settle, damp_ratio);

%Plot the step response
set(sscl,'InputName','r (m)', 'OutputName', {'x(m)', 'theta (rad)',
'angularrate(rad/s)', 'F (N)'});
figure
step(sscl)
```

```
function plot_poles(sscl, t_settle, damp_ratio)
======================================================================
%PLOT_POLES  Plot system pole locations  with settling time and damping
ratio constraints.
%
%   PLOT_POLES(SSCL, T_SETTLE, DAMP_RATIO)
%
%   This function plots the pole locations for the closed loop
%   system SSCL along with the settling time constraint
%   T_SETTLE (in seconds) and damping ratio DAMP_RATIO.

%   By Jim Ledin, 2002.
======================================================================
plot(pole(sscl), 'o')

axis equal
a = axis;
x_min = a(1); x_max = a(2);
y_min = a(3); y_max = a(4);

settling_pct = 0.01; % If no settling percentage given, use 1%
settling_limit = -log(settling_pct) / t_settle;
if x_max < -settling_limit + 0.1*(x_max - x_min)
    x_max = -settling_limit + 0.1*(x_max - x_min);
    a(2) = x_max;
end

hold on
plot([x_min x_max], [0 0], '--k')
plot([0 0], [y_min y_max], '--k')

plot([-settling_limit -settling_limit], [y_min y_max]);

angle = acos(damp_ratio);
plot([x_min 0 x_min], [x_min*tan(angle) 0 -x_min*tan(angle)])

axis(a)
hold off
```