SELF-BALANCING BOT USING CONCEPT OF INVERTED PENDULUM

Pratyusa kumar Tripathy (109EC0427)

Department of Electronics and Communication Engineering
National Institute of Technology Rourkela
Rourkela- 769008,India

SELF-BALANCING BOT USING CONCEPT OF INVERTED PENDULUM

A project submitted in partial fulfilment of the requirements for the degree of

Bachelor of Technology
in

Electronics and Communication Engineering

by

Pratyusa kumar Tripathy
(Roll 109EC0427)

under the supervision of

Prof. S. K. Das

Department of Electronics and Communication Engineering
National Institute of Technology Rourkela
Rourkela -769008, India

Electronics and Communication Engineering
National Institute of Technology Rourkela
Rourkela-769008, India.www.nitrkl.ac.in

Prof S. K. Das May 13^{th}, 2013

Certificate

This is to certify that the work in the Project entitled self-balancing robot using concept of inverted pendulum by Pratyusa kumar Triparthy, is a record of an original research work carried out by him under my supervision and guidance in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof S. K. Das

Acknowledgment

I would like to express my sincere gratitude and thanks to my supervisor Prof. S. K. Das for his constant guidance, encouragement and extreme support throughout the course of this project. I am thankful to Electronics and Communication Engineering Department to provide us with the unparalleled facilities throughout the project. I am thankful to Ph.D Scholars for their help and guidance in completion of the project. I am thankful to our batch mates and friends specifically Debabrata Mahapatra and my super senior Subhranshu Mishra for their support and being such a good company. I extend my gratitude to researchers and scholars whose papers and thesis have been utilized in our project. Finally, I dedicate my thesis to our parents for their love, support and encouragement without which this would not have been possible.

Pratyusa Kumar Tripathy

Abstract

Self-balancing robot is based on the principle of Inverted pendulum, which is a two wheel vehicle balances itself up in the vertical position with reference to the ground. It consist both hardware and software implementation. Mechanical model based on the state space design of the cart, pendulum system. To find its stable inverted position, I used a generic feedback controller (i.e. PID controller). According to the situation we have to control both angel of pendulum and position of cart. Mechanical design consist of two dc gear motor with encoder, one arduino microcontroller, IMU (inertial mass unit) sensor and motor driver as a basic need. IMU sensor which consists of accelerometer and gyroscope gives the reference acceleration and angle with respect to ground (vertical), When encoder which is attached with the motor gives the speed of the motor. These parameters are taken as the system parameter and determine the external force needed to balance the robot up.

It will be prevented from falling by giving acceleration to the wheels according to its inclination from the vertical. If the bot gets tilts by an angle, than in the frame of the wheels; the centre of mass of the bot will experience a pseudo force which will apply a torque opposite to the direction of tilt.

CONTENT

List of figures

List of Tables

CHAPTER 1

Introduction

Chapter 1

Introduction

To make a self-balancing robot, it is essential to solve the inverted pendulum problem or an inverted pendulum on cart. While the calculation and expressions are very complex, the goal is quite simple: the goal of the project is to adjust the wheels' position so that the inclination angle remains stable within a pre-determined value (e.g. the angle when the robot is not outside the premeasured angel boundary). When the robot starts to fall in one direction, the wheels should move in the inclined direction with a speed proportional to angle and acceleration of falling to correct the inclination angle. So I get an idea that when the deviation from equilibrium is small, we should move "gently" and when the deviation is large we should move more quickly.

To simplify things a little bit, I take a simple assumption; the robot's movement should be confined on one axis (e.g. only move forward and backward) and thus both wheels will move at the same speed in the same direction. Under this assumption the mathematics become much simpler as we only need to worry about sensor readings on a single plane. If we want to allow the robot to move sidewise, then you will have to control each wheel independently. The general idea remains the same with a less complexity since the falling direction of the robot is still restricted to a single axis.

Basic Aim:

- To demonstrate the methods and techniques involved in balancing an unstable robotic platform on two wheels.
- To design a complete digital control system with the state space model that will provide the needed.
- To complete the basic signal processing which will be required in making a unicycle.

A concept to start with:

Ultimate aim to start this project is the unicycle. The basics of the project basically lie on the inclination angle. Also at the very first I thought of making a self-balancing platform, which used data from accelerometer and gyroscope. A self-balancing bot is an advanced version of this platform. Self-balancing bot includes the basic signal processing part (which uses kalman filter and compensatory filter) of the unicycle.

Objective:

- ANALYSIS OF SYSTEM DYNAMICS AND DESIGN OF AN APPROXIMATE MODEL
- System synthesis
- Simulation
- Optimization
- Hardware implementation

CHAPTER 2

ANALYSIS OF SYSTEM DYNAMICS AND DESIGN OF AN APPROXIMATE MODEL

Chapter 2

2.1 Requirement Analysis

The system in my project consists of an inverted pendulum mounted to a cart which is stable by its own wheel. The inverted pendulum system is already an exclusive example that u can commonly find it in control system reference and research literature. Its idea concludes in part from the fact that it is not controllable for all values of system parameter, that is, the pendulum will simply can't control itself on upright position. Its dynamics of the system equation are nonlinear. The main fundamentals of the control system are to balance the inverted pendulum by applying a force to the hinge point. A real-world example that relates directly to this inverted pendulum system is the Segway vehicle.

In this case we will consider a two-dimensional problem where the pendulum is constrained to move in the vertical plane shown in the figure 2.1. For this system, the control input was the force F that moves the cart horizontally and the outputs are the angular position of the pendulum θ and the it at a distance x from the origin..

By taking this example we have to take a system that contains all the experimentals components with all there charcterstics. After analysing the system we will get the eqautions which helps to derive the state space model.

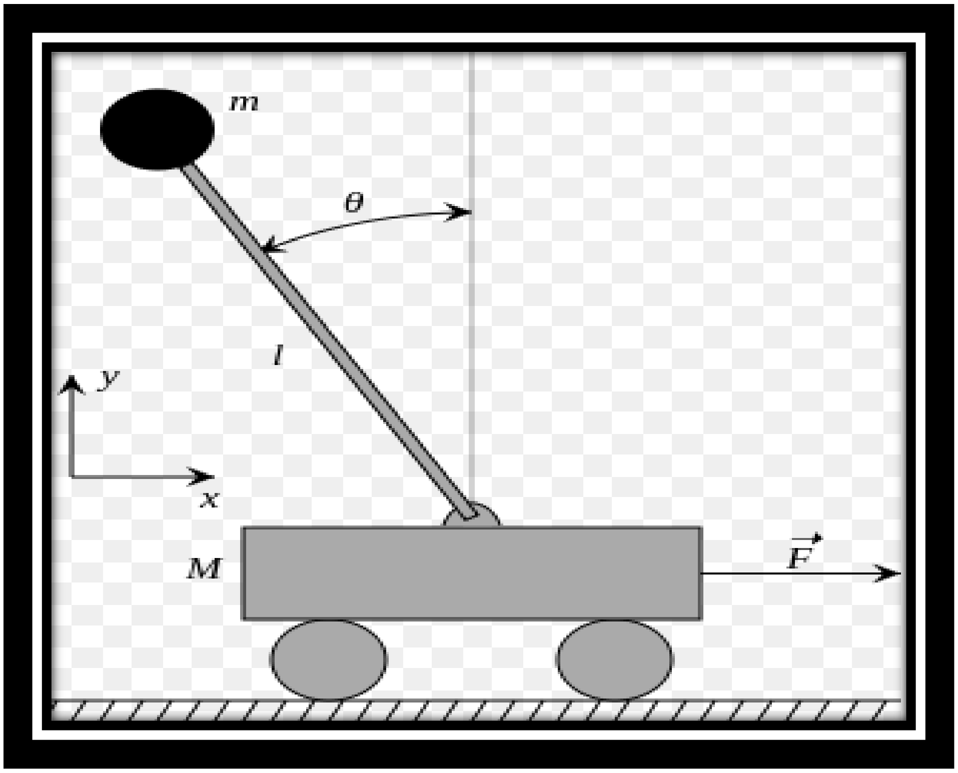


FIG 2.1 A model of an cart-pendulum system

Component	Abbreviation	Value
mass of cart	M	1 K.G.
Mass of pendulum	m	1 K.G.
coefficient of friction on wheel	b	0.1 N/M/sec
length to pendulum from the hinge point	l	0.3 M
pendulum angle from vertical (down)	Φ	To be calculated
moment of inertia of the pendulum	I	0.025 K.G.*m^2
force applied to the cart	F	Will be given

Table 2.1: cart pendulum system parameters

For the PID, frequency response and root locus sections of this problem, we are interested in the control of the pendulum angel from vertical and pendulum's position. Using transfer function which is best-suited for single-input, single-output (SISO) systems you can't control both the system output. Therefore, the design criteria deal with the cart's position and pendulum angle simultaneously. We can, however, assume the controller's effect on the cart's position after the controller has been designed by hit and trial method. In the next sections, we will design a controller to keep the pendulum to a vertically upward position when it will undergo a sudden force. Specifically, the design criteria are that the pendulum restores its upright position within 2 seconds and that the pendulum never inclined more than 0.08 radians away from vertical(that is controllable) after being disturbed by a force of magnitude 2 Nsec. The pendulum will initially begin from any controllable position

But employing state-space design techniques, we are more confined and eager to choose a multi-output system. In our case, the inverted_ pendulum system on a cart is single-input (only external), multi-output (SIMO). Therefore, for the state-space model of the Inverted Pendulum on a cart, we will control both the cart's position and pendulum's angle. To make the design more accurate and well defined in this section, we will check the parameters a 0.2-meter step in the cart's desired position. Under these conditions, it is expected that the cart achieve its stable position within 5 seconds and have a rise time under 0.2 seconds. It is also desired that the pendulum archive to its vertical position in under 2 seconds, and further, that the pendulum angle not travel more than 30 degrees (0.35 radians) way from the vertically upward.

In summary, the design requirements for the inverted pendulum state-space example are:

- Rise time for x of less than 0.5 seconds

- Settling time for x and θ of system equation less than 2 seconds
- Steady-state error of less than 2% for x and θ
- Pendulum angle θ never more than 20 degrees (0.35 radians) from the vertical

2.2 Force Analysis and Equation

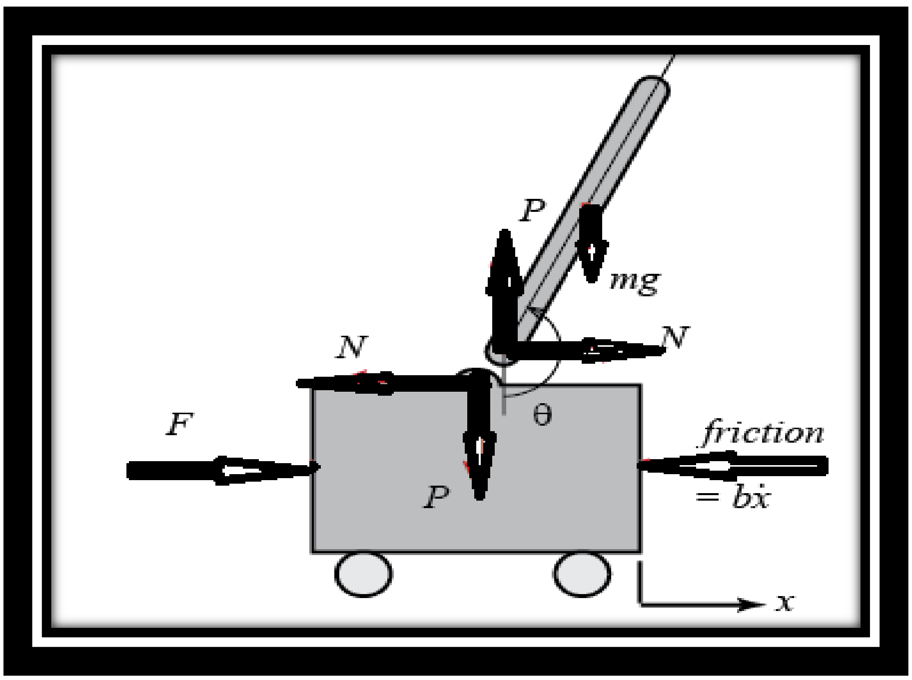


Figure 2.2:force analysis

By taking the total force along the horizontal axis of the cart, we got the equation:

$$F = H + b\dot{x} + M\ddot{x} \qquad 2.1$$

Equation of Motion:

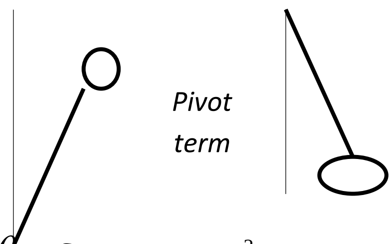
$$\tau_{\text{total}} = \tau_{\text{grav}} + \tau_{\text{pivot}} = I\frac{d^2\theta}{dt^2} = ml^2\frac{d^2\theta}{dt^2} \qquad 2.2$$

Angular acceleration:

$$ml^2\frac{d^2\theta}{dt^2} = mgl\sin\theta - mg\omega^2 A\cos(\omega t)\sin\theta \qquad 2.3$$

Total angular force must be ‘0’:

$$\boxed{\frac{d^2\theta}{dt^2} - \left[\frac{g}{l} - \frac{\omega^2 A}{l}\cos(\omega t)\right]\sin\theta = 0} \qquad 2.4$$



$$\frac{d^2\theta}{dt^2} - \frac{g}{l}\theta = 0 \qquad \frac{d^2\theta}{dt^2} + \frac{g}{l}\theta = 0$$

$$\theta(t) = \exp(\omega_0 t) \qquad \theta(t) = \sin(\omega_0 t)$$

$$\boxed{\omega_0^2 = \frac{g}{l}} \qquad g = 9.81\,\mathrm{m.s^{-2}} \qquad l = 19\,\mathrm{cm}$$

$$\boxed{\omega_0 = 7.19\ \mathrm{rad.s^{-1}}}$$

Note: summing the forces in the vertical direction for the cart, but we can’t get any useful information.

By taking the Sum of the forces in the free-body diagram of the pendulum cart problem in the horizontal direction, we got the following expression for the reaction force N.

$$N = m\ddot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta \qquad 2.5$$

By substituting equation 2.5 in the equation 2.1, we will get one of the two governing equations of motion for the system.

$$(M+m)\ddot{x} + b\dot{x} + ml\ddot{\theta}\cos\theta - ml\dot{\theta}^2\sin\theta = F \qquad 2.6$$

To find the second equation of motion for this system, we have to take take the sum of the forces perpendicular to the pendulum original position. By solving the equation along pendulum axis greatly simplifies the mathematics. Then the equation is:

$$P \sin\theta + N \cos\theta - mg \sin\theta = ml\ddot{\theta} + m\ddot{x}\cos\theta \qquad 2.7$$

To solve for the P and N terms in the equation2.5 and 2.6, sum the angular moments about the centroid of the pendulum. The equation is found out as:

$$-Pl \sin\theta - Nl \cos\theta = I\ddot{\theta} \qquad 2.8$$

Combining equation 2.7 and 2.8, we get the second governing equation.

$$(I + ml^2)\ddot{\theta} + mgl \sin\theta = -ml\ddot{x}\cos\theta \qquad 2.9$$

The analysis and control design techniques only can be applied in the above problem statement apply only to linear systems. So to solve these equations this set of equations needs to be linearized. Simply, we will assume that the system stays within a small neighborhood of this equilibrium; therefore, we will linearize the equations about the vertically upward equilibrium position, $\theta = \pi$. This assumption is one type of reasonably valid since under control we have to monitor that the pendulum should not deviate more than 20 degrees from the vertically upward position (uncontrollable). Let ϕ defined as the deviation of the pendulum from vertically upward position, that is, $\theta = \pi + \phi$. Again assuming a small deviation (ϕ) from vertical equilibrium, we can use the following small angle approximations of the nonlinear functions in our system equations:

$$\cos\theta = \cos(\pi + \phi) \approx -1 \qquad 2.10$$

$$\sin\theta = \sin(\pi + \phi) \approx -\phi \qquad 2.11$$

$$\dot{\theta}^2 = \dot{\phi}^2 \approx 0 \qquad 2.12$$

By applying the above equation 2.11 and 2.12 into nonlinear equations 2.6 and 2.9, we come to the end with two linearized equations of motion. Note $u$has been substituted for the input$F$

.

$$(I + ml^2)\ddot{\phi} - mgl\phi = ml\ddot{x} \qquad 2.13$$

$$(M + m)\ddot{x} + b\dot{x} - ml\ddot{\phi} = u \qquad 2.14$$

Final equation of motion and torque

- F = H +b$\dot{x}$ + M$\ddot{x}$
- V = mg + ml cosØ$\dot{Ø}^2$+ mlsinØ.$\ddot{Ø}$
- H = m$\ddot{x}$ + mlsinØ$\dot{Ø}^2$ − mlcosØ.$\ddot{Ø}$
- I.$\ddot{Ø}$ = mglcosØ

2.3 Transfer function

To find the transfer functions of the system equations, we have to first take the Laplace transform of the system equations assuming zero initial conditions. The Laplace's equations used are,

The Laplace transform $\mathcal{L}$ is defined by

$$\mathcal{L}_t\,[f(t)]\,(s) \equiv \int_0^\infty f(t)\,e^{-s\,t}\,d\,t,$$

Then the equation becomes,

$$(I + ml^2)\Phi(s)s^2 - mgl\Phi(s) = mlX(s)s^2 \qquad 2.15$$

$$(M + m)X(s)s^2 + bX(s)s - ml\Phi(s)s^2 = U(s) \qquad 2.16$$

That a transfer function of a system represents the relationship between a single input and a single output for every equation. To find the first transfer function for the pendulum angel and an input of U(s) we have to eliminate X(s) from the equations 2.15 and 2.16.Solving the equation for X(s).

$$X(s) = \left[\frac{I + ml^2}{ml} - \frac{g}{s^2}\right]\Phi(s) \qquad 2.17$$

Then substitute the equation 2.17 into the second equation.

$$(M + m)\left[\frac{I + ml^2}{ml} - \frac{g}{s^2}\right]\Phi(s)s^2 + b\left[\frac{I + ml^2}{ml} - \frac{g}{s^2}\right]\Phi(s)s - ml\Phi(s)s^2 = U(s) \qquad 2.18$$

Manipulating, the transfer function is then the following

$$\frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s^2}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \qquad 2.19$$

Where,

$$q = [(M + m)(I + ml^2) - (ml)^2] \qquad 2.20$$

From the transfer function shown in the equation 2.18 and 2.19 it can be seen that there is both a pole and a zero at the origin. These can be canceled and the transfer function becomes the following.

$$P_{pend}(s) = \frac{\Phi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(I+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \qquad [\frac{rad}{N}] \qquad 2.21$$

Second, the transfer function with the cart's position will be derived in a same way to reach at the following.

$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(I+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \qquad [\frac{m}{N}] \qquad 2.22$$

2.4 State-space model

The transfer function represented as equations of motion derivation of commanding equations can also be represented in state-space form if they can arranged into a series of first order and second order differential equations. They can then be put into the standard matrix, because the equations are linear.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u$$

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

The control matrix has 2 columns because both the pendulum's position and the cart's position are part of the system output. Basically, the cart's position is the first necessary of the output y and the pendulum's angle from vertically upward position is the second element of Y.

CHAPTER 3

PID CONTROLLER AND OPTIMISATION

CHAPTER 3

3.1 PID controller Overview

A **proportional-integral-derivative controller** is a generic feedback controller. PID controller processes the "error" as the difference between a measured output and a desired given references and tries to minimize the error by adjusting the control parameters…

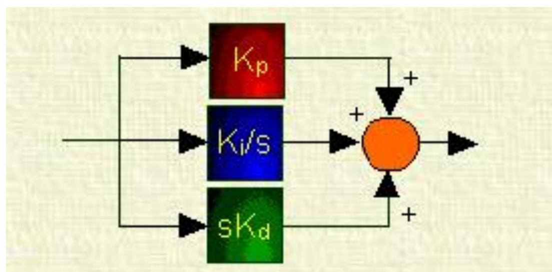


Figure 3.1:PID control parameters

In the time-domain analysis, the output of a PID controller, which is proportional to control input is given by:

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_p \frac{de}{dt} \qquad 3.1$$

Giving look towards how the PID controller works in a closed-loop system using the system variable. 'e' represents the system error due to both system noise and measurement noise, the difference between the desired output value and the actual output produced. This error signal is given to the PID controller, and the controller determines both the derivative and the integral of this error. The input to the plant should be the summation of derivative constant multiplied by derivative error, proportional constant times the proportional error and integral times the integral error.

When control signal (u) will sent to the plant as the only input, and the output (y) is obtained according to the given input.. The new output (y) is then given back and subtracted to the desired reference to find error signal (e) again and the loop continues. The PID takes this error and calculates its control constants.

The transfer function of a PID controller is found by taking the Laplace transform of Eq.(1).

$$K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} \qquad 3.2$$

Where

K_p = Proportional gain

K_i= Integral gain

K_d= Derivative gain

The structure of a plant:

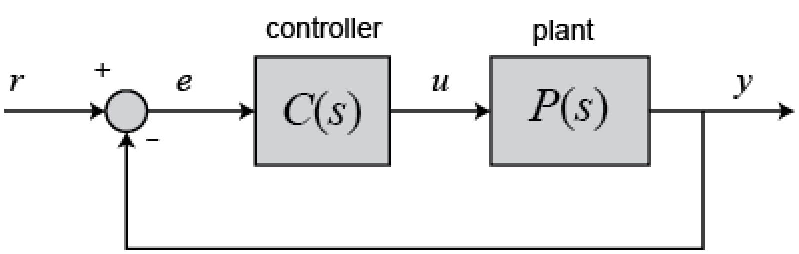


Fig 3.2: system with PID controller

3.2 Effects of control parameters on the close loop system

Due to proportional controller, we will have reduced the rise time but no effect on steady state error. An integral control (K_i) reduces the steady-state error for step input, but negative effect on rise time. A derivative increases the stability of the system as well as reduces the overshoot.

Closed loop responce	Rise time	Over shoot	Settling time	Steady state error
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	eliminate
Kd	Small Change	Decrease	Decrease	No change

Table 3.1:PID parameter effect comparision.

3.3 PID controller design with state space

The easiest method one should attempt to make the pendulum balanced is to rotate the wheels in the inclined direction until the inclination angle approaches to zero where the pendulum is in balance. Basically, the rotation speed of the wheel must be proportional to the inclination angle (e.g. move faster when the inclination is more and vice versa) so that the robot move with a greater settle time. This is called the simplest PID control with neglecting both the I and the D terms.

- Proceeding to the next step in the design process, we have to find state-feedback control gains represented in a vector assuming that we are well aware (i.e. can measure) all the state variables (four state variables are there). There is various methods to do it. If you know the desired closed-loop pole locations, we can use the advanced control theory. We can also use the “lqr” command which returns the optimal controller gain by heat and trial method for a linear plant, cost function must be power of 2 at most and initial conditions must equal to zero .

- We have to check that the system is **controllable before design a controller**. By meeting all of this property of controllability means that we can set the state of the system anywhere in the controllable region (under the physical constraints of the system). The system to meet all the conditions to be completely state controllable, the rank of the controllability is the number of independent rows (or columns).

$$\mathcal{C} = [A|AB|A^2B|\cdots|A^{n-1}B] \qquad 3.3$$

- The controllability matrix of the system is shown by equation 3.3. The number of power indicates to the number of state variables of the system. Addition of terms to the controllability matrix with higher powers of the matrix cannot increase the rank of the matrix because they are linear combination of each other.

- Controllability matrix is consisting of four variables; the rank of the matrix should be 4 to be controllable. By using the command ctrb in MATLAB to generate the controllability matrix. Likewise using rank command we can find the rank. So we will test in simulation chapter.

3.4 Pre-compensation

The designed controller meets our transient requirements so, but now we should focus upon the steady-state error. With respect to the other design methods, where we feedback the output and compare it to the reference input to compute an error, with a full-state feedback controller we are feeding back all of the states. We need to compute what the steady-state value of the states should be, multiply that by the chosen gain, and use a new value as our "reference" for computing the input. We can do it by adding a constant gain after the reference input.

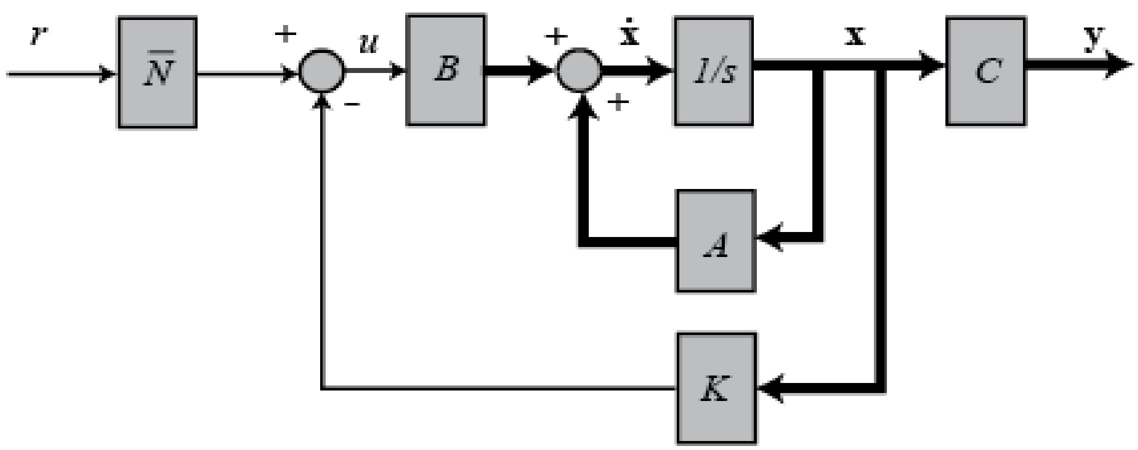


Fig 3.3: the cart pendulum system

CHAPTER 4

Simulation

CHAPTER 4

4.1:
Time response

The **time response** shows how the state of a dynamic system changes with respect to time when a particular input is applied. Our system consists of differential equations, so we must perform some integration in order to determine the time response of this dynamic system. For most systems, especially nonlinear systems or those subject to different input parameters, we have to carry out integrations numerically. But in case of linear systems MATLAB provides many useful commands for calculating time responses for many types of inputs.

The time response of a linear dynamic system consists of the sum of the **transient response** and **steady state response**. Transient response depends on the initial conditions while the steady-state response which depends on the system input. So in the differential equation contains two terms, one is due to free parameters and other is due to forced parameters.

Frequency Response

Linear time invariant systems are most important because its response linear for state variables. The property of LTI that the input to the system is sinusoidal, therefore the steady-state output will also be sinusoidal at the same frequency. They only differs in phase and magnitude. These magnitude and phase differences as a function of frequency and called as the **frequency response** of the system.

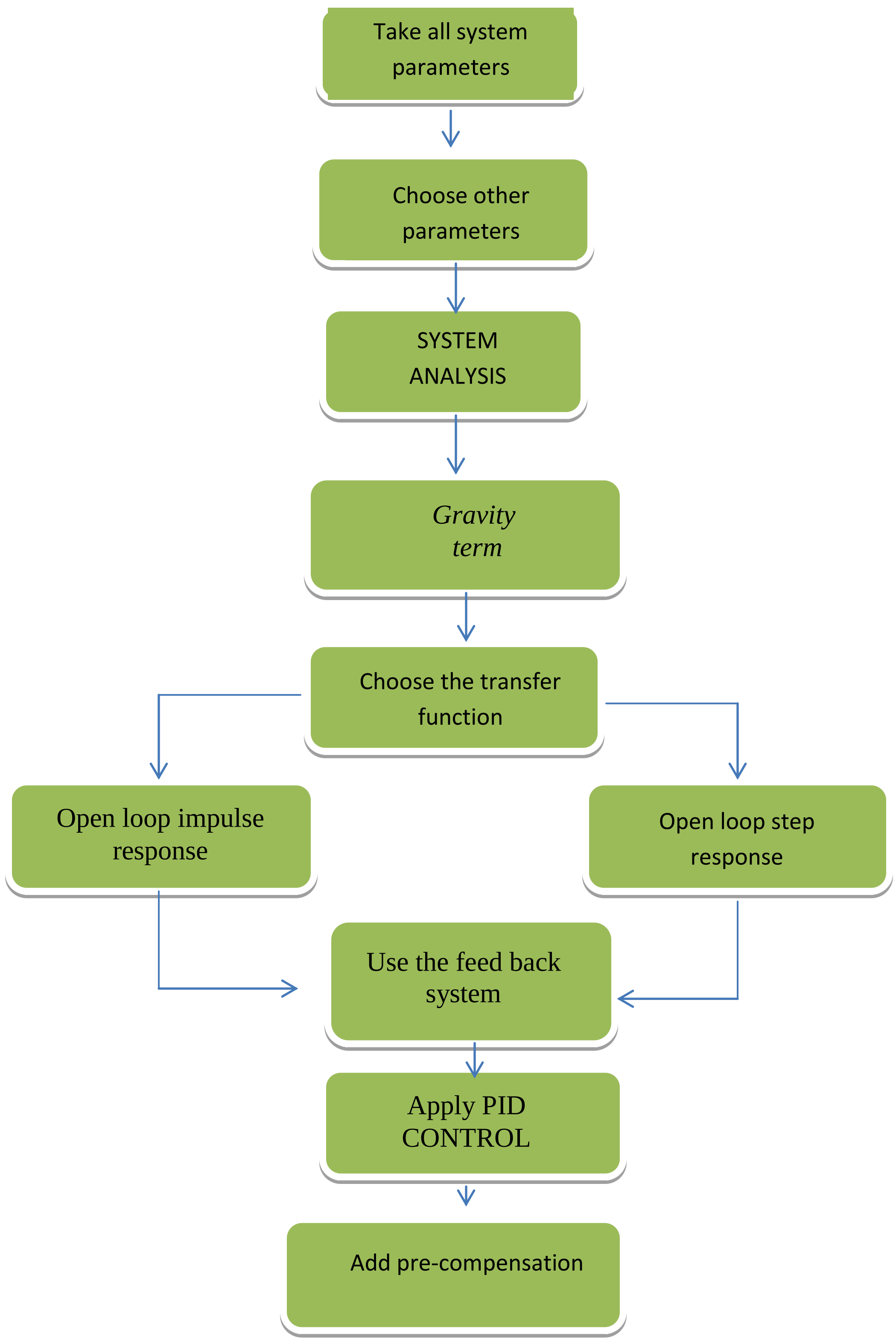
In various ways we can get the frequency response analysis (varying between zero or "DC" to infinity). We will start with computing the value of the plant transfer function at known frequencies. If G(s) is the open-loop transfer function of a system and ω is the desired frequency, we then plot $G(j\omega)$ versus ω. Then we can use **bode plot** and Nyquist plot to get rid of complex frequency values.

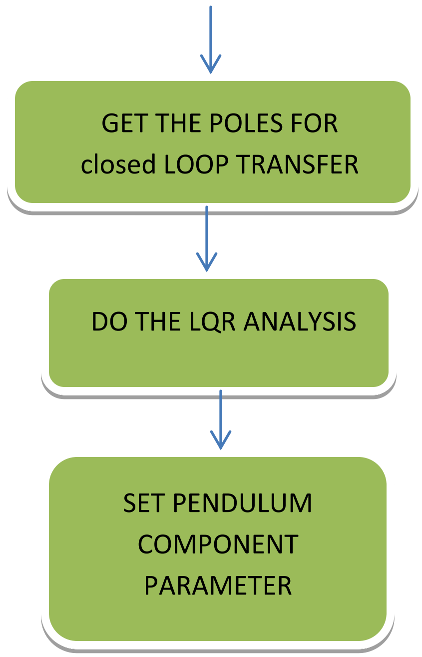
Stability

In the pendulum cart system we will use the **Bounded Input Bounded Output (BIBO)** definition of stability which states that a system is stable if the output remains bounded in a linear region for all inputs in the controllable region. Means this system will not diverges for any input to the systems while working.

To check the system stability the transfer function representation must needed. If all poles of the transfer function (values of s at which the denominator equals zero) lies in the –VE X-axis then system is called stable. If any pole has a positive real part, then the system is unstable. If any pair of poles is on the imaginary axis, then the system is called marginally stable and the system will oscillate infinite time which is not possible for a practical case. The poles of a LTI system model can easily be found in MATLAB using the command “POLE”.

4.2 Controller design:





4.3 Simulator Design:

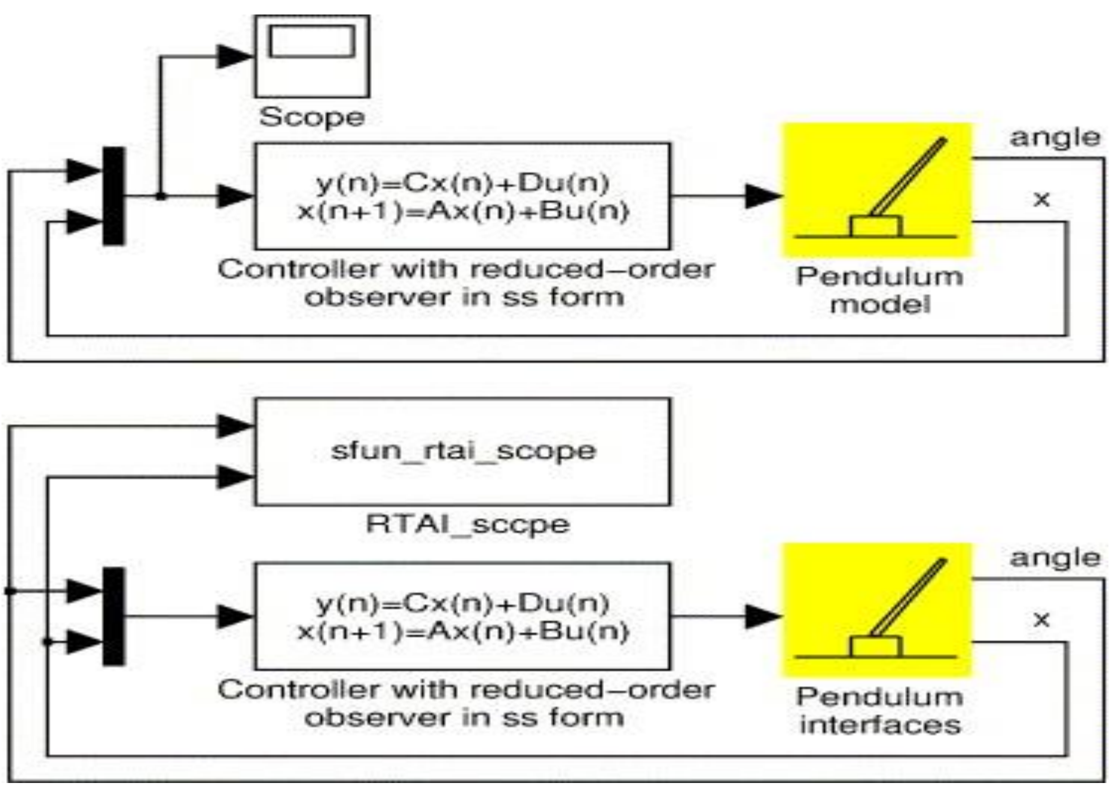


Figure 4.1: simulation of open loop

Another recursive approach:

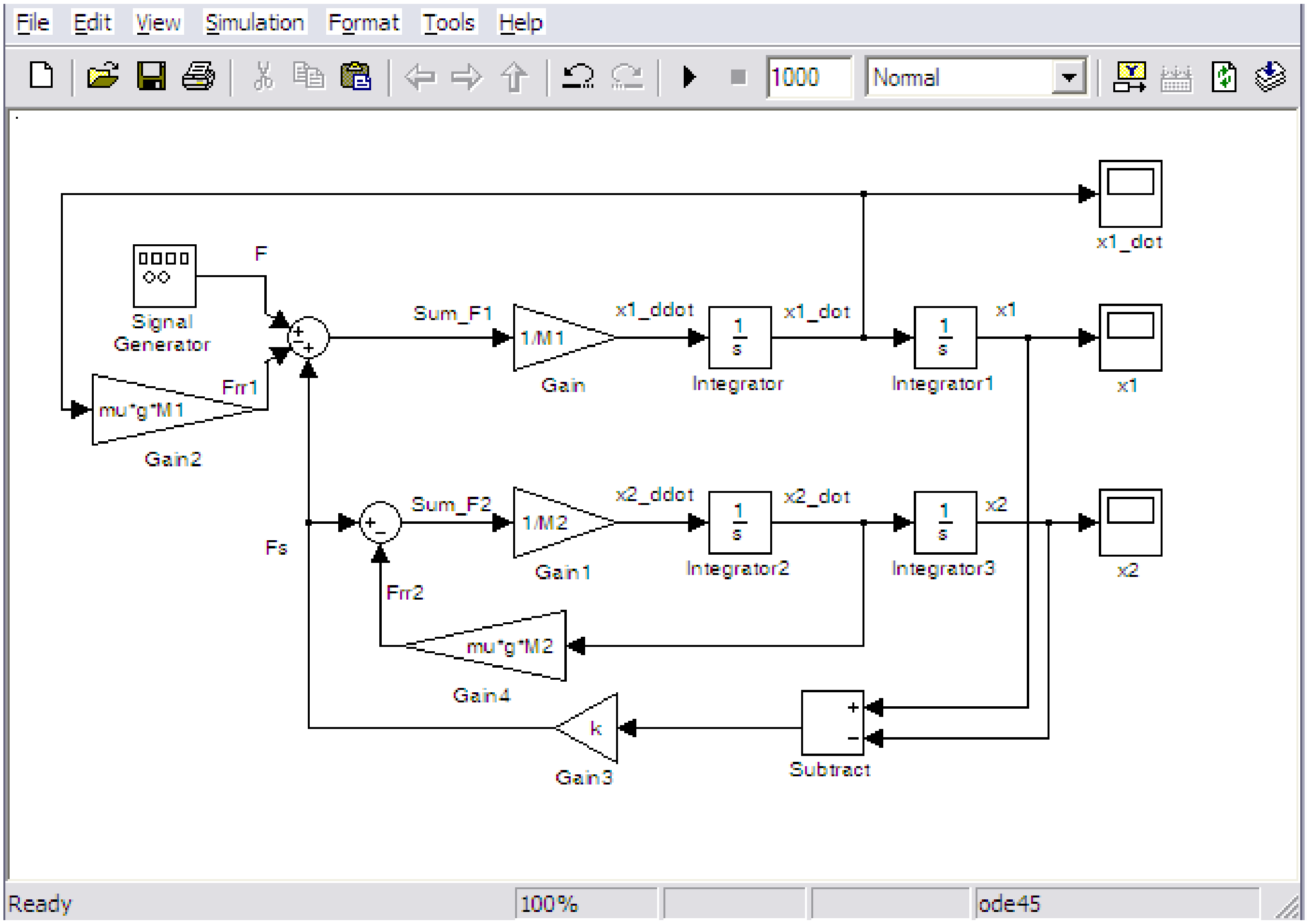


Figure 4.2: simulation of closed loop transfer function

CHAPTER 5

KALMAN FILTER
(THE ESTIMATOR AND PREDICTOR)

CHAPTER 5

4.1 Introduction to kalman filter

Discrete time linear systems basically represented as

$$x_j = a x_{j-1} + b u_j \quad \textbf{5.1}$$

Where;

x_j, represent the jth state variables,

a and b are constants

u_j represents control input in jth state ;

j is the time variable.

Note that many type of kalman filter don't include the input term (zero initial conditions), and k is also used to represent time. I have chosen to use j to represent the time variable because we use the variable k for the Kalman filter gain later in the text. The extension of kalman filter represented next.

Important properties of kalman filter:

- Data processed through recursive algorithm.
- For a given the set of measurements, it generates optimal estimate of desired outputs.
- Optimal outputs:
  - It is the best filter to get the minimal error according to the previous state.
  - For non-linear system optimality is very possible using unsent kalman filtering.
- Recursive algorithm:

– It does not store any previous data, it recursively calculates all every time.

$$x(k+1) = F(k)x(k) + G(k)u(k) + v(k)$$
$$y(k) = H(k)x(k) + w(k)$$

Where,

$x(k)$ is the n-dimensional state vector (unknown)
$u(k)$ is the m-dimensional input vector (known)
$y(k)$ is the p-dimensional output vector (known, measured)
$F(k), G(k), H(k)$ are appropriately dimensioned system matrices (known)
$v(k), w(k)$ are zero-mean, white Gaussian noise with (known)
covariance matrices $Q(k), R(k)$

4.2 Discrete Kalman Filter

By sampling in the time domain the sample at unit time is stored in the state variables and then the operations will be performed:

- Estimation the state variables $\mathbf{x} \in \Re^n$ which is a linear stochastic difference equation

$$\mathbf{x}_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_{k-1} \qquad 5.3$$

  – Process noise **w** will obtained from $N(0,\mathbf{Q})$, where covariance matrix represented as **Q**.

- with all measurement parameter which is also real ($\mathbf{z} \in \Re^m$)

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \qquad 5.4$$

– Measurement noise **v** is obtained from *N*(0,**R**), where R is the covariance matrix.

A, Q are n dimension square matrix, B is *having* n*l configuration and '**R'** is *m*x*m* and '**H'** is *m*x*n*

Time Update (Predictor step):

In the predictor step time will be updated with respect to both control variable and state variable-

- Expected Update in the value of **x in time domain**

$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1} + \mathbf{B}\mathbf{u}_k \qquad 5.5$$

- Expected update for the covariance matrix:

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q} \qquad 5.6$$

- The simplest method to show the equation 5.5 and 5.6

$$\hat{x}(t_3^-) = \hat{x}(t_2) + u[t_3 - t_2] \qquad 5.7$$

$$\sigma^2(t_3^-) = \sigma^2(t_2) + \sigma_\varepsilon^2[t_3 - t_2]$$

Measurement Update (Corrector step)

According to the predicted values we have to update the next time domain quantities

- Value Updated in corrector step:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-) \qquad 5.8$$

– The real error update is $\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-$

- Updated error for the covariance matrix

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H})\mathbf{P}_k^- \qquad 5.9$$

- Compare the equation 5.8 and 5.9

$$\hat{x}(t_3) = \hat{x}(t_3^-) + K(t_3)(z_3 - \hat{x}(t_3^-)) \qquad 5.10$$

$$\sigma^2(t_3) = (1 - K(t_3))\sigma^2(t_3^-)$$

The Kalman Gain

- Then Kalman gain $\mathbf{K}_k$ for the optimal state:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T (\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R})^{-1} \qquad 5.12$$

$$= \frac{\mathbf{P}_k^- \mathbf{H}^T}{\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R}} \qquad 5.13$$

- The equation 5.12 can be comparable with simple kalman filter as

$$K(t_3) = \frac{\sigma^2(t_3^-)}{\sigma^2(t_3^-) + \sigma_3^2} \qquad 5.14$$

The Jacobian Matrix

To simplify a system equation to use the nonlinear kalman filter we use the Jacobean matrix

- For implementation of a scalar function $y=f(x)$,

$$\Delta y = f'(x)\Delta x \qquad 5.15$$

- For implementation of a vector function $\mathbf{y}=f(\mathbf{x})$,

$$\Delta \mathbf{y} = \mathbf{J}\Delta \mathbf{x} = \begin{bmatrix} \Delta y_1 \\ \vdots \\ \Delta y_n \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_n}{\partial x_n}(\mathbf{x}) \end{bmatrix} \cdot \begin{bmatrix} \Delta x_1 \\ \vdots \\ \Delta x_n \end{bmatrix} \quad 5.16$$

EKF Update Equations

By using the nonlinear kalman filter, the filter operation know as extended kalman filter (EKF) whose steps are derived above and only the equation given below:

- Predictor step:

$$\hat{\mathbf{x}}_k^- = f(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k) \quad 5.17$$

$$\mathbf{P}_k^- = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^T + \mathbf{Q} \quad 5.18$$

- Kalman gain:

$$\mathbf{K}_k = \mathbf{P}_k^- \mathbf{H}^T (\mathbf{H}\mathbf{P}_k^- \mathbf{H}^T + \mathbf{R})^{-1} \quad 5.19$$

- Corrector step:

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k (\mathbf{z}_k - h(\hat{\mathbf{x}}_k^-)) \quad 5.20$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H})\mathbf{P}_k^- \quad 5.21$$

4.3 EKF ALL STEP AS A GRAPH:

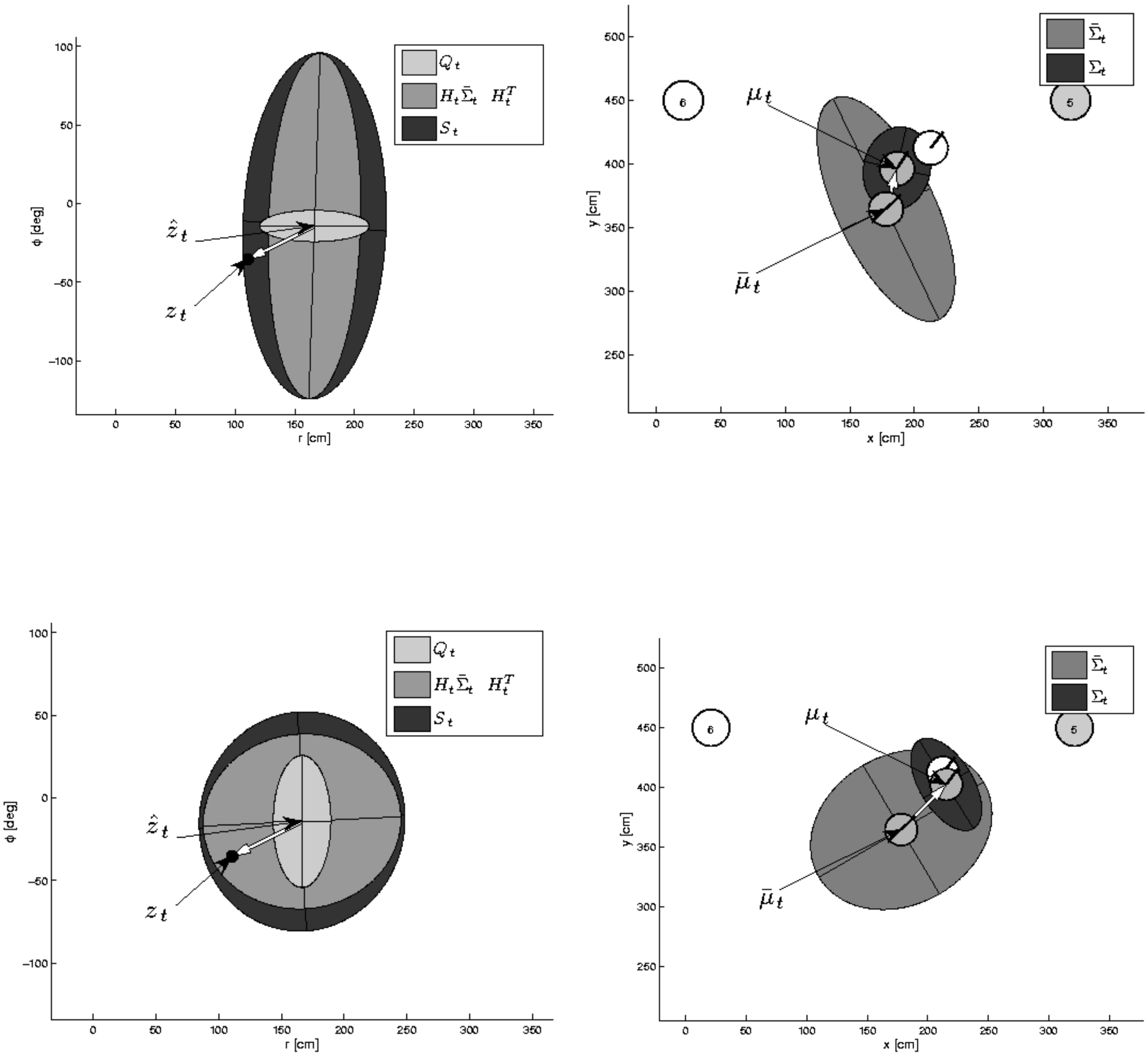


Fig 4.1 Extended kalman filter in graph representation

CHAPTER 6

HARDWARE IMPLIMENTATION

CHAPTER 6

6.1 The components used to build a working model

1. Arduino mega 2560
2. Inertial measurement unit
3. X-bee
4. Dc motor and shaft encoder.
5. High current Dc motor
6. Lcd
7. Power circuit.

6.2 Arduino mega 2560:

The Arduino Mega 2560 is a arduino based microcontroller board based on the ATmega microcontroller 2560. It contains 16 analog inputs, 4 UARTs (hardware serial ports), 54 digital input/output pins (of which 14 can be used as PWM outputs), a 16 MHz crystal oscillator, a power jack, an ICSP header, a USB connection, as well as a reset button. It contains all on chip peripherals to support the microcontroller. By connecting it simply to a computer with a USB cable or with an AC-to-DC adapter or battery (for power purpose), to get started. Most shields designed for the Arduino Duemilanove or Diecimila is compatible with arduino mega 2560.

The new feature added in Mega2560 is the FTDI USB-to-serial driver chip which differs from all preceding boards. At last, it uses the ATmega16U2 (ATmega8U2 is used in revision 1 and revision 2 boards) programmed as a USB-to-serial converter.

- The data lines for I2C added SDA and SCL pins that are near to the AREF pin and two other new pins AREF and ground placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage provided from the board. In future, shields will be compatible both with the board that use the AVR, which operate with 5V and with the Arduino Due that operate with 3.3V.

Specification:

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB (8 KB used by bootloader)
SRAM	8 KB

EEPROM	4 KB
Clock Speed	16 MHz

Table 6.1:arduino mega 2560 specification

Memory:

The ATmega2560 is capable to store code in 256 KB of flash memory (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which is used for store variable at run time).

Input and Output:

All the 54 digital pins on arduino Mega can be used as an digital input or output, using some functions like digitalWrite(), pinMode(), and digitalRead() functions. They operate at

A input voltage of 5 volts. Maximum current flow in these digital pins is of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. Like atmega some pins are having one or more specialized functions:

- **Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. UART 0 is also connected the ATmega16U2 USB-to-TTL Serial chip used to program the board.
- **External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2).** These pins can be configured to trigger an interrupt on a rising or falling edge on a low value, or a change in value. **Attach Interrupt()** function gives all details about interrupts

- **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library.
- **LED: 13.** A led is connected to digital pin 13. When the pin goes HIGH, the LED is on, when the pin goes LOW, it's off.
- **PWM: 2 to 13 and 44 to 46. It** uses analogWrite() function to Provide 8-bit PWM output.

Pin Diagram:

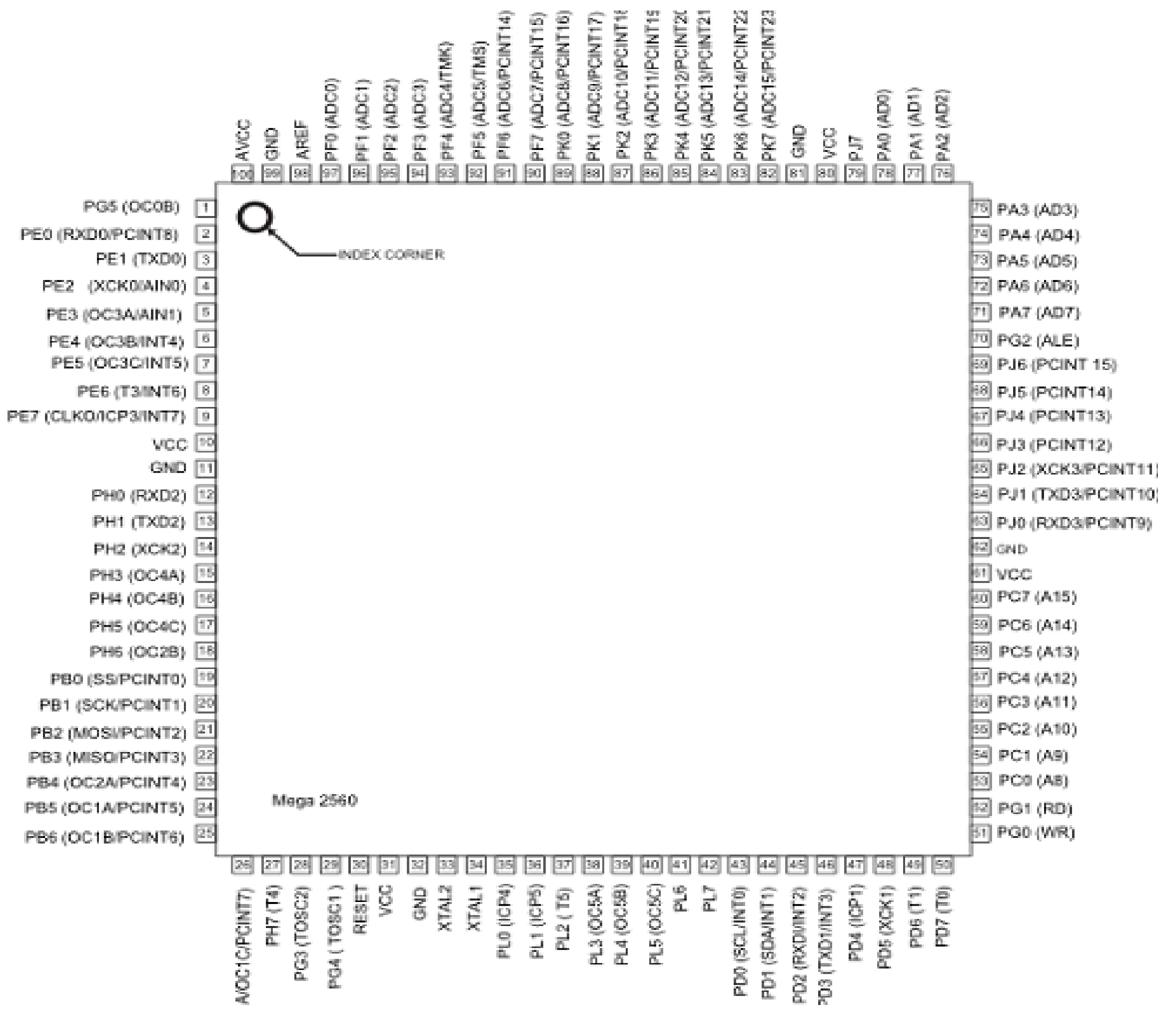


Fig 6.1: PIN diagram of arduino mega 2560

Programming

The Arduino Mega can be programmed with the Arduino software which is a open source software. Arduino is so famoused due to its open and long library. The ATmega2560 on the Arduino Mega comes pre-burned with a boot loader which helps to upload new code to it without the using an external hardware programmer. It communicates using the STK500 protocol which is also used by the atmega.

6.3 Inertial measurement Unit

The IMU is an electronics module consist of more than one module in a single unit, which takes angular velocity and linear acceleration data as a input and sent to the main processor. The IMU sensor actually contains three separate sensors. The first one is the accelerometer. To describe the acceleration about three axes it generates three analog signals and acting on the planes and vehicle. Because of the physical limitations and thruster system, the significant output sensed of these accelerations is for gravity. The second sensor is the gyroscope. It also gives three analog signals. These signals describe the vehicle angular velocities about each of the sensor axes. It not necessary to place IMU at the vehicle centre of mass, because the angular rate is not affected by linear or angular accelerations. The data from these sensors is collected by the microprocessor attached to the IMU sensor through a 12 bit ADC board. The sensor information communicates via a RS422 serial communications (UART) interface at a rate of about 10 Hz.

The accelerometer and gyroscope within the IMU are mounted such that coordinate axes of their sensor are not aligned with self-balancing bot. This is the real fact that the two sensors in the IMU are mounted in two different orientations according to its orientation of the axis needed..

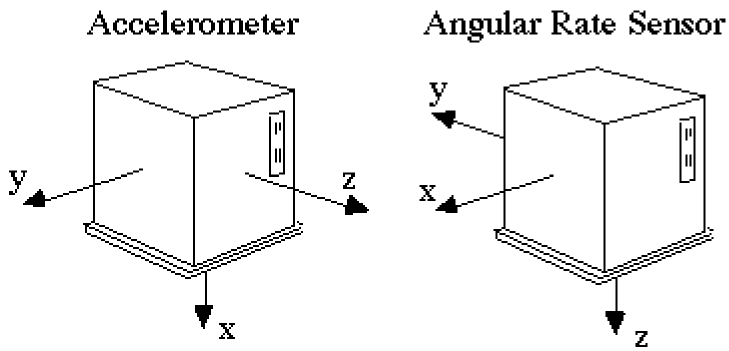


Fig 6.2:IMU sensor angle

The accelerometer is manufactured by using a left handed coordinate system. The transformation algorithm first uses fig 6.2 to align the coordinate axes of the two sensors.

$$
\begin{aligned}
x_{comp} &= -x \\
y_{comp} &= y \\
z_{comp} &= z \\
\omega_{x_{comp}} &= -\omega_z \\
\omega_{y_{comp}} &= \omega_x \\
\omega_{z_{comp}} &= -\omega_y
\end{aligned}
$$

Notice that the gyroscope are now aligned and right handed according to the IMU axis.

Once the accelerometer and gyroscope axes are aligned with the axes of the IMU, then it should be aligned to vehicle reference frame. Let’s take an example, the unit is mounted on the wall of the electronics module, and we have to rotate it 45° with respect to the horizontal. So according to the Euler’s axis transformation, the angle it deviates

$$\tan^{-1}\left(\frac{4}{30}\right) = 7.59 \pm .04^\circ = 0.1325 \pm .0007(\text{Radians})$$

from the vehicle axes. This was calculated by taking the assumption that over the 30 inches from the back to the front of the of the IMU sensor the reference move inward 4 inches.

Using information of angel transformation, along with the orientation with which the IMU is mounted on the vehicle of the euler angle transformation allows to form a direction cosine matrix that is used to convert the IMU coordinate measurement frame to the vehicle coordinate frame. Figure 6.3 illustrates the orientation with which the IMU is mounted with the three sensors.

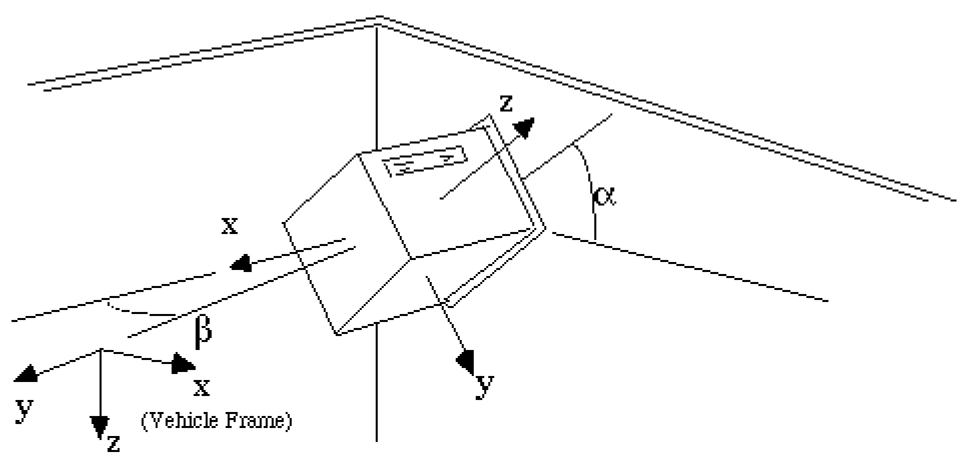


Fig 6.3: Euler angle transformtation.

Transformed IMU coordinate axes compared to vehicle coordinate axes

Following is the order of transformations:

1) Rotating alpha + 90° about the x-axis to align the IMU z-axis with the Vehicle frame(Z-axis).

$$\mathbf{T_1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \qquad 6.1$$

2) Rotate beta + 90° about the z-axis to align IMU and vehicle coordinate frames.

$$\mathbf{T_2} = \begin{bmatrix} \cos\beta & -\sin\beta & 0 \\ \sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad 6.2$$

3) So the complete transformation from IMU to vehicle coordinates is given by

$$\mathbf{T} = \begin{bmatrix} -\sin\beta & \cos\beta\sin\alpha & \cos\alpha\cos\beta \\ \cos\beta & \sin\alpha\sin\beta & \cos\alpha\sin\beta \\ 0 & \cos\alpha & -\sin\alpha \end{bmatrix} \qquad 6.3$$

4) Inserting numerical values for the angles ([alpha] = 45°, [beta] = 7.5946°), produces:

$$\mathbf{T} = \begin{bmatrix} -0.132162 & 0.700904 & 0.700904 \\ 0.991228 & 0.0934528 & 0.0934528 \\ 0 & 0.707107 & -0.707107 \end{bmatrix} \qquad 6.4$$

In order to transform the sensor data from the individual sensor frames into the vehicle frame, first use the simple sign/axis transformations shown in (6.1), and then pre-multiply the modified sensor data by the transformation matrix given in (6.3). It would also be possible to combine the initial transformations from (6.1) into the transformation in (6.3), but the initial axes of the two sensors in the IMU are different on their own axes frame, there would be a different transformation matrix for each sensor. To simplify the derivation, the program first aligns the sensor axes using (6.1), and then uses the second transformation matrix from (2.6) to transform the data of the accelerometer and angular rate sensors from IMU to vehicle coordinates.

6.4 Dc motor and Shaft encoder

Figure 6.4 High torque motor

Here I am using the 85RPM 37DL Gear Motor with shaft Encoder, which is a high performance DC gear motor and contains magnetic Hall Effect quadrature shaft encoder. Shaft Encoder gives 90 pulses per revolution of the output shaft per channel. Motor can run from 4V to 12V supply. For shaft encoder signals, Sturdy 2510 relimate connector is used and Shaft encoders are used in applications that requires motor's angle of rotation such as robotics, CNC etc.

Specifications:

Supply: 12V (Motor runs smoothly from 4V to12V)

Shaft encoder resolution: 90 quadrature pulses per revolution of the output shaft

RPM: 85RPM @ 12V; 42RPM @6V

Stall Current: 1.8A @ 12V; 0.8A @ 6V

Stall Torque: 21kg/cm @ 12V; 10kg/cm @ 6V

Shaft encoder supply: 3.3V or 5V

Shaft encoder current requirement: 5 to 10mA

Shaft diameter: 6mm

Gear ratio: 1:30

Table 6.2:motor specification

Interfacing

We are using Sturdy 2510 relimate to give power to motor and take the output signal of shaft encoder. Shaft encoder needs a supply voltage of 3.3V or 5V DC. 1K to 2.2K pull-up resistor is connected between Vcc and Channel A and Channel B.

Black	Motor power 1
Brown	Motor power 2
Red	Shaft encoder Vcc (5V / 3.3V)
Orange	Shaft encoder ground
Yellow	Channel A open collector output
Green	Channel B open collector output

table 6.3:interfacing of motor

CHAPTER 7

Real Time implementation

CHAPTER 7

7.1 Algorithm to control SBB vertically upward

- **Take the necessary header files and global variables to support the main software.**
- **Configure PID angle and speed of the motor as well as PID constants.**
- **Take the read values of the IMU sensor and store it in a matrix up to 100 samples**
- **Configure the start button, stop button and device configure button.**
- **Configure PID module to update it.**
- **Set digital pins, serial pins and PWM pins on the arduino board**
- **Set the kalman filter and update its weight matrix and coefficient.**
- **Initialize PID speed, PID angle and Time action module.**
- **Update the IMU sensor and update the IMU matrix.**
- **Then we will face three conditions**

  1) **When debug->**

     a) **calibrate the previous values and update the previous declared variables.**

  2) **When started->**

A. **calculate the PID angle and motor speed.**

B. **According to the angel check motor speed regularly and update the IMU sensor.**

C. **If steering is not applied then set the external offset to zero**

D. **Else refer to SBB console program.**

3) **When stoped-> do the power off. Fall automatically.**

7.2 Algorithm to steer the SBB by remote control

- **Take the necessary header files and global variables to support the main software.**
- **Configure PID angle and speed of the motor as well as PID constants.**
- **Take the read values of the IMU sensor and store it in a matrix up to 100 samples**
- **Configure the start button, stop button and device configure button.**
- **Configure PID module to update it.**
- **Set digital pins, serial pins and PWM pins on the arduino board**
- **Set the kalman filter and update its weight matrix and coefficient.**
- **Set the remote of the buttons as well as the UART 1.**
- **If any button is pressed make its coefficient to 100.**
- **Add it to the next time action.**
- **Make the PID mode aggressive.**

- **Display it on the LCD**
- **Check the control parameter according to the change in PID angle.**

7.3 Other files used in the implementation

I. Remote_communication(using serial communication)

II. Eeprom.h (to store the update values)

III. Serial_communication

IV. Motor.h

V. Shaft_encoder.h

CHAPTER 8

Result and Graph

CHAPTER 8

8.1 result for controller design

According to the system model the open loop response of the system is not controllable and the graph is giving peak means the angle and the amplitude goes on increasing which is not BIBO stable. Fig 8.1 describes the same.

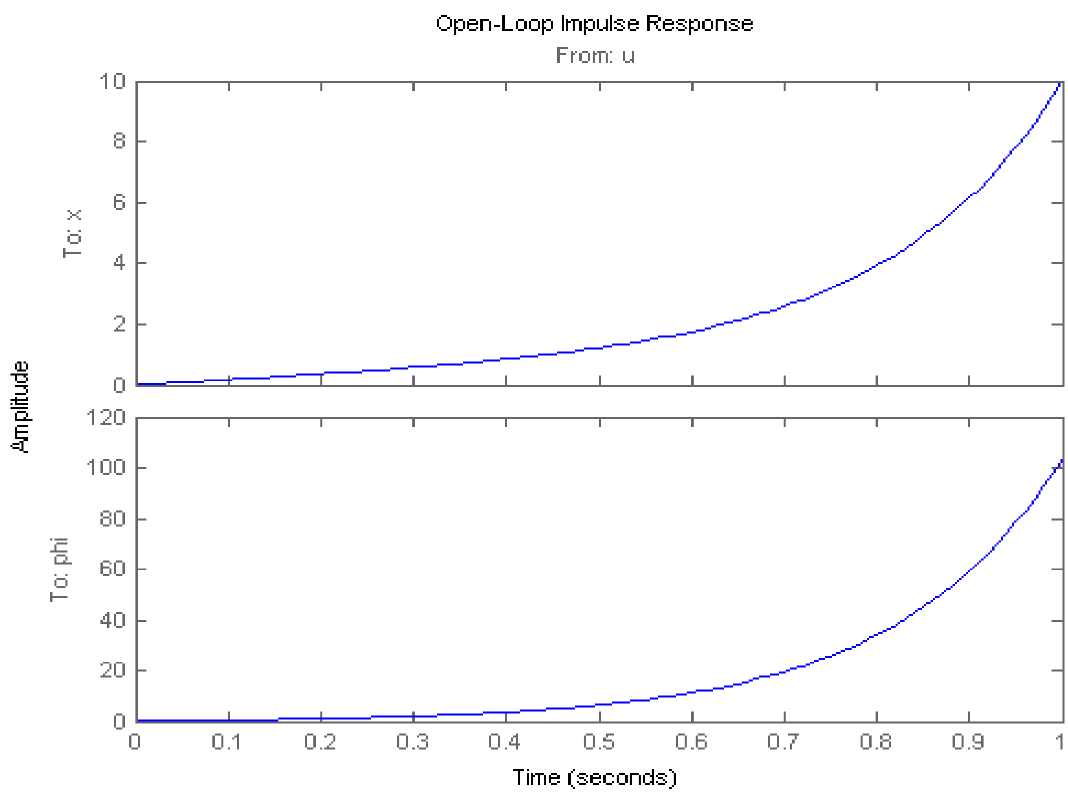


Fig 8.1 open loop impulse response and open loop step response

To find the root locus of the open loop transfer function we used the specific MATLAB command. The graph shows that the pole are on the +VE half of the X-axis which leads to system is unstable.

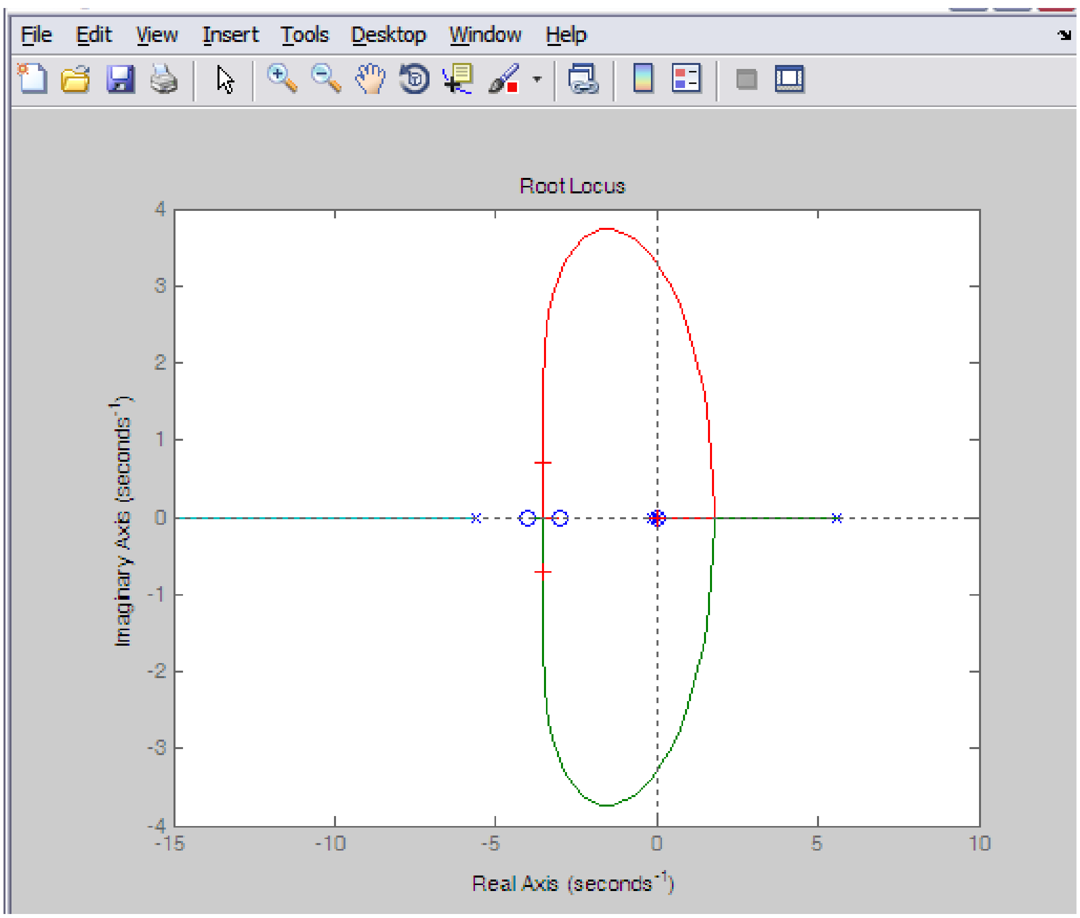


Fig 8.2 pole location and root locus

Design of a closed loop transfer function using the state space model can be done in two ways. By using LQR algorithm and also by placing the closed loop pole location on the desired position. The next two figures describe the same. Figure 8.3 describes the LQR model while in figure 8.4 shows LQR model with PID tuning.

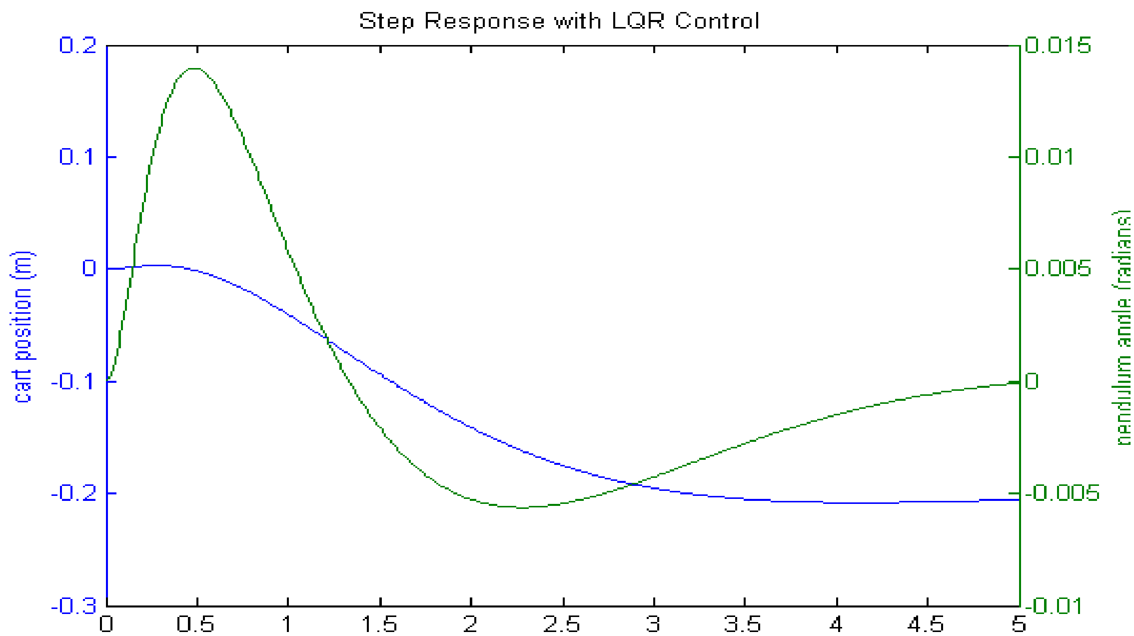


Fig 8.3 Control using LQR model

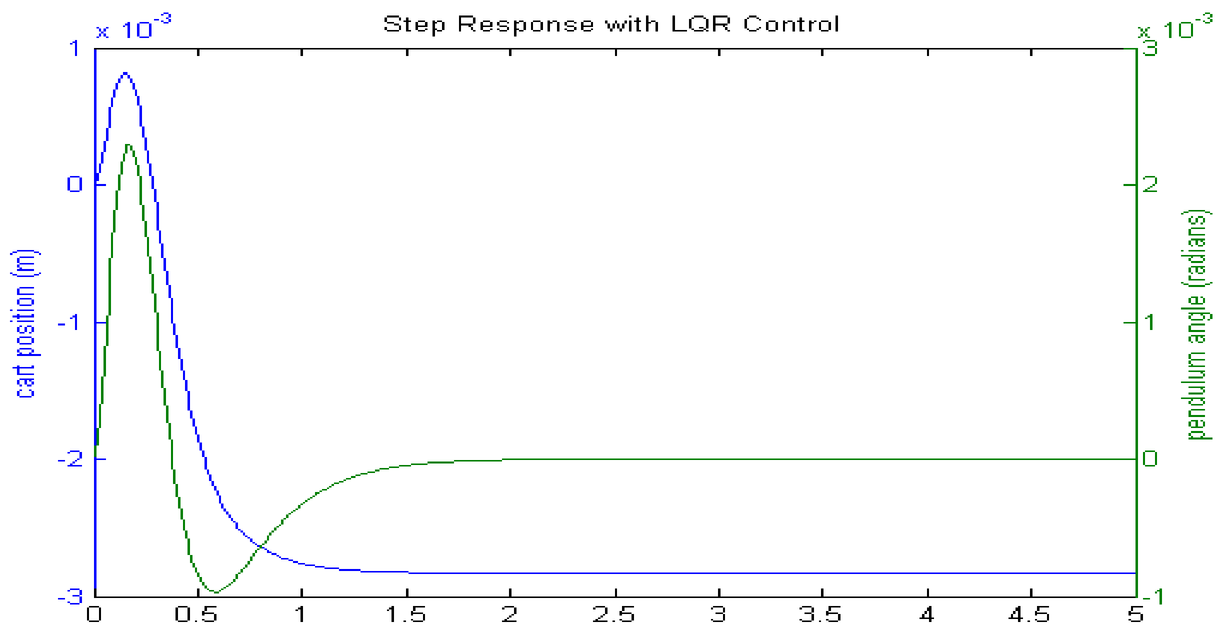


Fig 8.4 Control using LQR model with tuning parameters

As we discussed in chapter two though it is a feedback system, two take the error with respect to same reference we have to add a pre-compensation before the control input. The effect of pre-compensation is described in the following graph

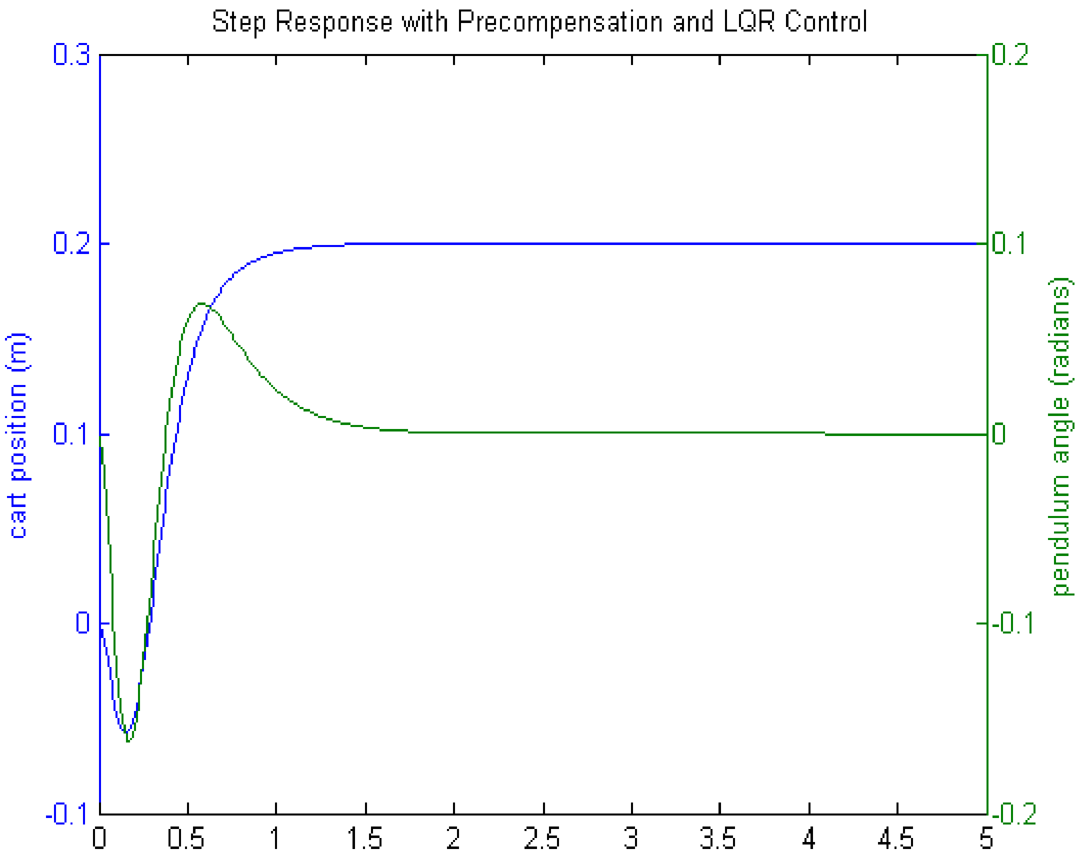


Fig 8.5 control in addition of pre-compensation

8.2 simulation results

By doing the simulation using MATLAB we get into the conclusion that the system function and parameter we derived as a reference for self-balancing bot work well. So in the next step we can proceed through the design. With open loop transfer function the animated system is unstable as shown in fig 8.6 whereas for a close loop transfer function the system is stable.

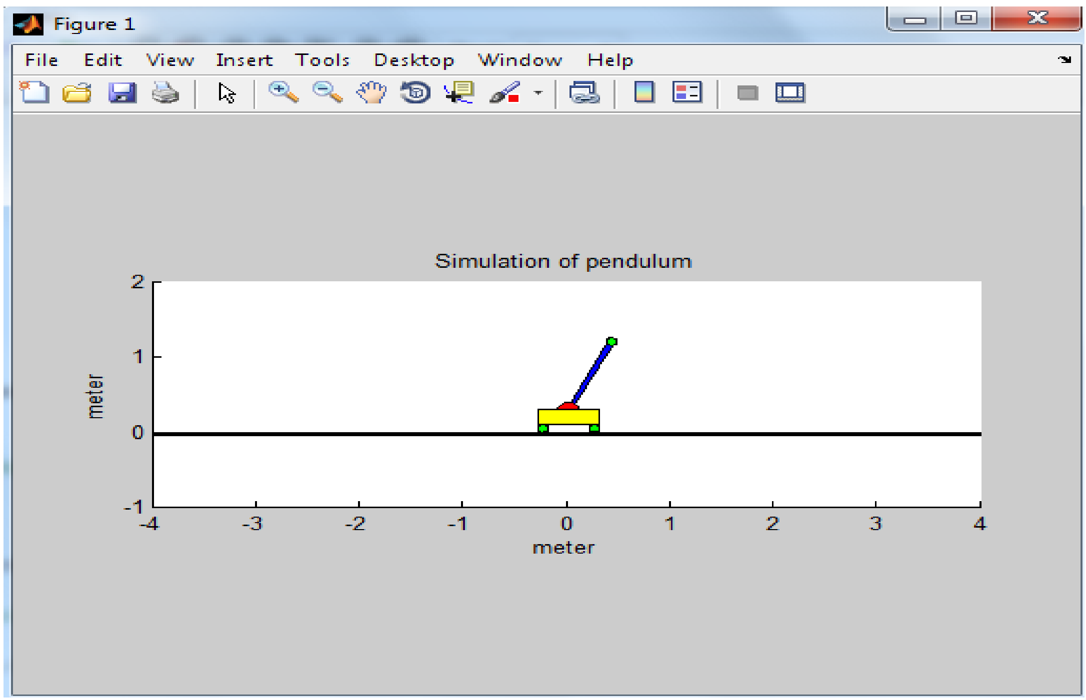


Fig 8.6 cart pendulum system in unstable condition

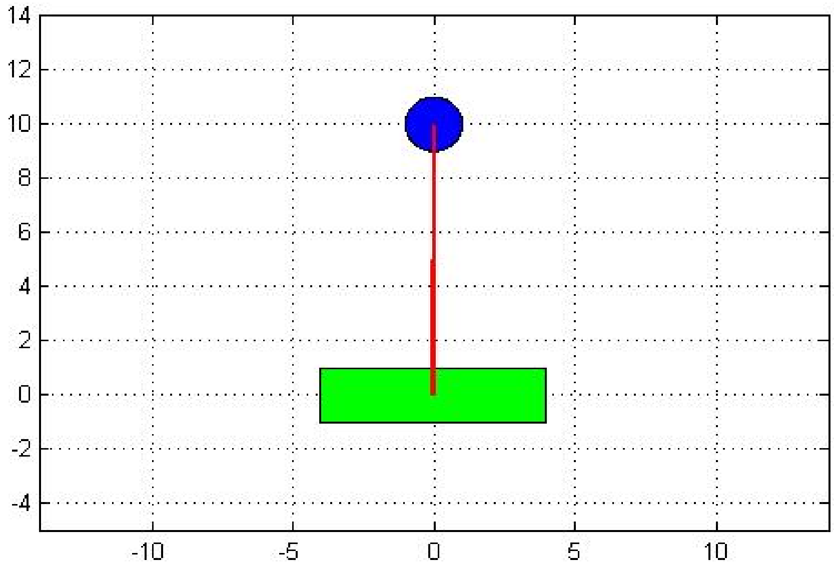


Fig 8.7 self-balanced Pendulum cart system

8.3. Hardware simulation result.

The following figure is taken from XCTU which is an interface to communicate with computer from the arduino board with the help of serial communication. It shows the output of the IMU sensor in form of yaw roll and pitch.

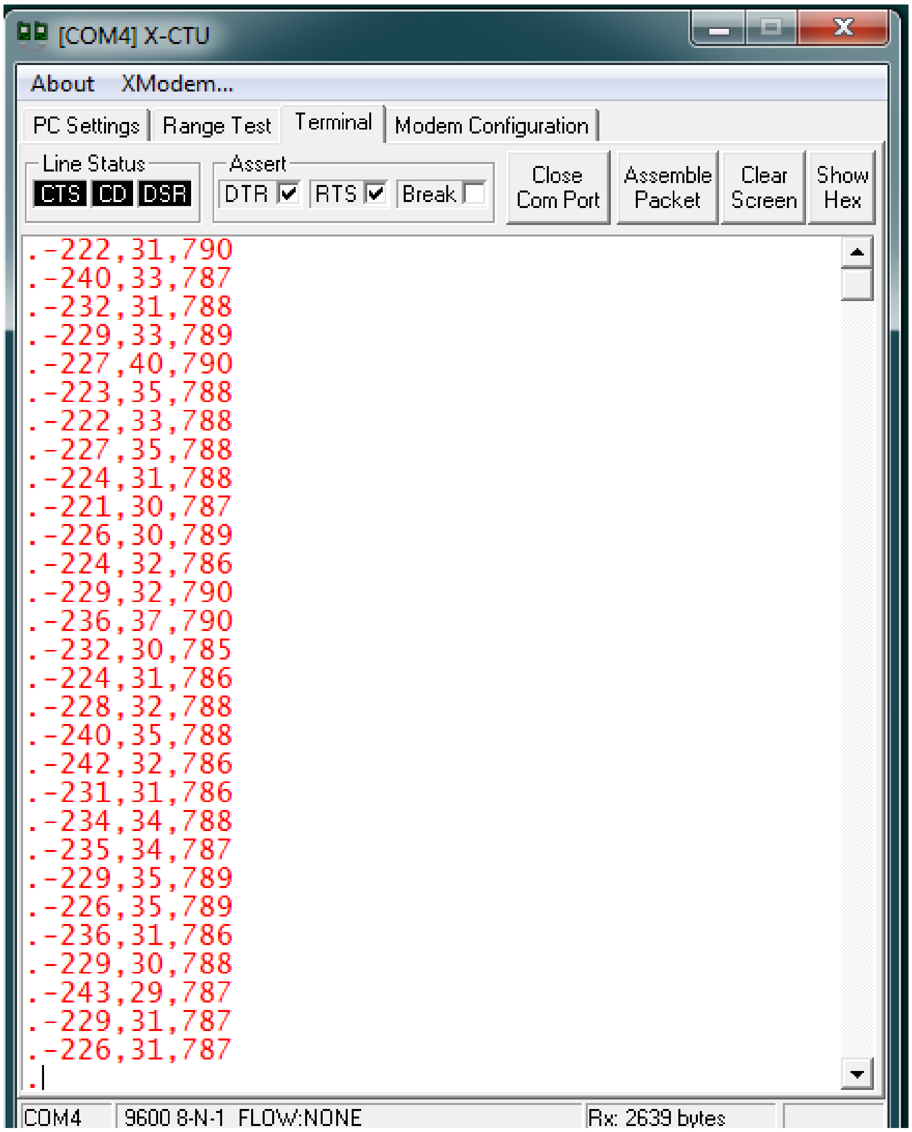
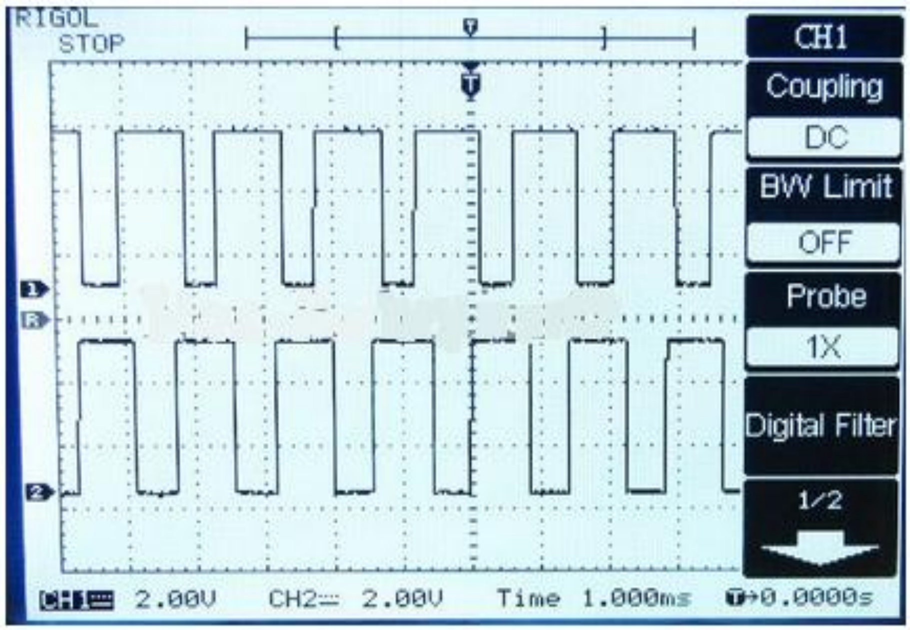


Fig 8.8 output of IMU sensor

Shaft encoder calculates the rotation of the wheel according to the values obtain from channel and channel b.both of them with a 90 degree phase shift each giving 90 count per 1 revolution making total of 180 counts



Channel A and Channel B quadrature outputs observed on CRO

Fig 8.9 output of shaft encoder

After all the derivations and simulations the final working model is derived which consist of all the components. fig 8.10 shows the working model on the upward direction.

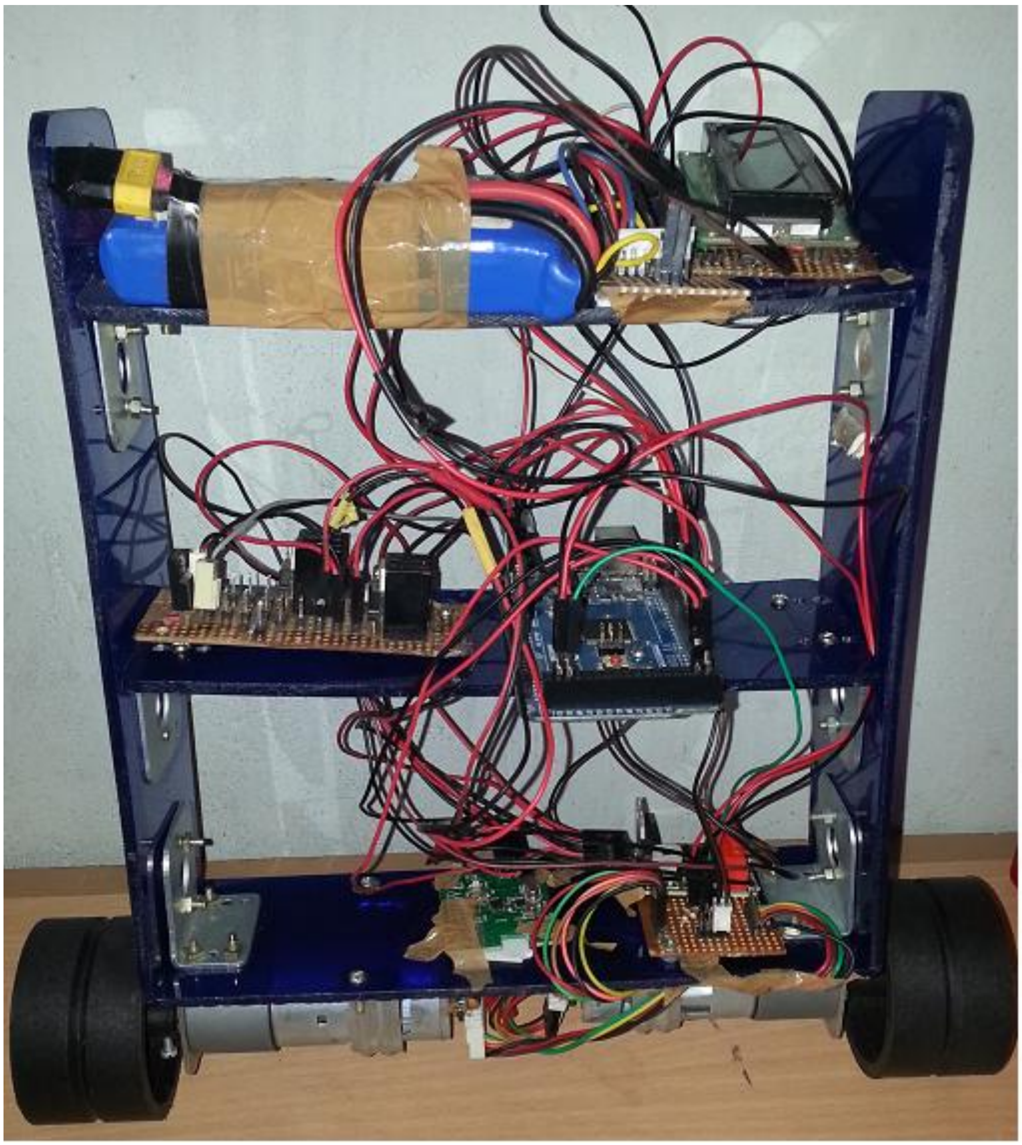


Fig 8.10 working model of SBB

CHAPTER 9

CONCLUSION

CHAPTER 9

Conclusion:

To build a self-balancing robot we first derived the system equation then check its real time response (both time and frequency). Then we designed a PID controller to control the close loop function. We checked the controllability and set the pole location. Then we used kalman filter as estimator and predictor. Then by choosing the appropriate components we analyse their simulation sucessfully

The above test steps are successful, then we are near to build a SBB. The easiest way to tune a PID controller is to tune the *P*, *I* and *D* parameters one at a time. It was done successfully. The stability of the SBB may be improved if you use a properly designed gearbox that is having negligible gear backlash.

So by implementation all of these concepts and avoid the errors that we came across the self-balancing bot is completely build. We can make Segway and ball bot as a application of self-balancing bot

Bibliography

[1]http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SimulinkModeling

[2]http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=SystemModeling

[3] http://playground.arduino.cc/Main/RotaryEncoders

[4] http://blog.wolfram.com/2011/01/19/stabilized-inverted-pendulum/

[5] Kalman, R. E. 1960. "A New Approach to Linear Filtering and Prediction Problems", Transaction of the ASME--Journal of Basic Engineering, pp. 35-45 (March 1960).

[6] Welch, G and Bishop, G. 2001. "An introduction to the Kalman Filter", http://www.cs.unc.edu/~welch/kalman/

[7] Maybeck, P. S. 1979. "Stochastic Models, Estimation, and Control, Volume 1", Academic Press, Inc.

[8] http://www.kerrywong.com/2012/03/08/a-self-balancing-robot-i

[9] http://diydrones.com/profiles/blog/show?id=705844%3ABlogPost%3A23188

[10] Lei Guo , Hong Wang , "PID controller design for output PDFs of stochastic systems using linear matrix inequalities", IEEE Transactions on Systems, Man, and Cybernetics, Part B 01/2005; 35:65-71. pp.65-71.