ABSTRACT

Two-wheeled self-balancing robot is a popular model in control system experiments which is more widely known as inverted pendulum and cart model. This is a multi-input and multi-output system which is theoretical and has been applied in many systems in daily use. Most research just focus on balancing this model through try-on experiments or by using simple form of mathematical model. There were few researches that focus on complete mathematical modeling and designing a controller for the system.

In this project we aim to design, construct and implement the control of a two-wheel self-balancing robot using LQR Controller. The system consist a pair of DC motor, motor controller and Arduino 101 microcontroller board with inbuilt 3-axis gyroscope and a 3-axis accelerometer, employed for vertical angle determination and encoders for drift and linear velocity determination. In addition, a complementary filter is implemented to compensate for accelerometer noise and gyro drifts.

The controller was tested with different case of system condition practically and through simulation on Simulink and Matlab programming.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

SYMBOL	DESCRIPTION
$\dot{\theta}$	Linear velocity
$\ddot{\theta}$	Linear acceleration
θ_l	Roll angle of left wheel
θ_r	Roll angle of right wheel
$\dot{\psi}$	Pitch velocity
$\ddot{\psi}$	Pitch acceleration
$\dot{\phi}$	Yaw (drift) velocity
$\ddot{\phi}$	Yaw (drift) acceleration
γ	Vertical angle of the pendulum
θ	Roll angle
ψ	Pitch angle
ϕ	Yaw (drift) angle
D	Body depth
G	Gravitational acceleration at the surface of the earth
H	Body height
f_m	Friction coefficient between body and DC motor
f_w	Friction coefficient between wheel and floor
J_W	Wheel moment of inertia
J_Φ	Body yaw inertia moment
J_Ψ	Body pitch inertia moment
K_b	DC motor back EMF constant
K_t	DC motor torque constant
L	Distance of Center of mass from wheel axle

M	Wheel weight
M	Body weight including wheel and battery
n	Gear ratio
R	Wheel radius
W	Body width

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The Two Wheeled Self-Balancing Robot works on the principle of inverted pendulum, which is a classical control system that is developed to balance itself in the vertical position. The system is inherently unstable and even a slight disturbance would cause the system to fall. Hence a suitable controller has to be designed to maintain the stability of the system.

Self - Balancing Robots have been a topic of interest of many researchers, students and hobbyist worldwide. The primary practical application of a Self-Balancing Robot is the Segway Personal Transporter. It is used in many industries such as inside factory floors or for tourism in the park. The concept of self-balancing robot finds its application in Segway, as in Fig. 1.1 and Attitude control of Quadcopter, Rockets and Space shuttles.

Fig. 1.1 Application of inverted pendulum

1.2 LITERATURE REVIEW

Grasser and Alonso D'Arrigo, (2002). presented a paper on 'JOE: A Mobile, Inverted Pendulum', IEEE Transactions on Industrial Electronics, Vol 49. The

overview of the paper was studied and the mathematical model of self balancing robot was analyzed.

Ogata, K. (2009). authored a book named 'Modern Control theory, University of Minnesota, United States of America: Person Publication. Inc.' The algorithm of LQR and its analysis gives the theoretical background and mathematical analysis of self balancing robot.

Mikael Arvidsson. And Jonas Karlsson. (2012). presented a thesis on 'Design, construction and verification of a self-balancing vehicle. Chalmers University of Technology.' The software modelling is implemented using the proposed method.

Sundin, C. and Thorstensson, F. (2013), presented a thesis on 'Autonomous balancing robot, Chalmers University of Technology.' The overview of the thesis was reviewed with help of the procedure proposed and analyzed the mathematical model of self balancing robot.

Ooi, R. (2013). presented a thesis 'Balancing a Two-Wheeled Autonomous Robot. Undergraduate. The University of Western Australia.' The overall implementation in software uses sensor calibration, filtering and Arduino coding.

Amir A. Bature and Mohamed. N.Ahmad (2014). published a journal 'A Comparison of Controllers for Balancing Two Wheeled Inverted Pendulum Robot'. International Journal of Mechanical & Mechatronics Engineering IJMME-IJENS Vol:14 No:03. The comparison of various controllers like PID, Linear Quadratic Regulator and LQG were analyzed with the help of this paper.

Osama Jamil et al (2014). Presented a paper 'Modeling, Control of a Two Wheeled Self-Balancing Robot'. International Conference on Robotics and Emerging Allied Technologies in Engineering, Islamabad, Pakistan, April 22-24, 2014. Algorithm implementation using Arduino was referred from this paper.

1.3 PROBLEM STATEMENT

The Two wheel Self Balancing Robot has to be controlled so that the body remains balanced and upright, and is resistant to a step disturbance. The output of inertial sensors used is noisy and filtering of raw output of sensors is required before it is fed to the microcontroller for reliable performance. Another major problem is that the computation time required by microcontroller to run control algorithm. Excessive computation delay leads to more time for correction of the tilt angle of inverted pendulum and leaves the system out of vertical equilibrium position. As compared to conventional robots, the power required for driving motors to maintain equilibrium position is high in case of auto balancing robots.

1.4 PROBLEM SOLVING

The goal of project is to investigate the use of digital control algorithm, implemented on a microcontroller. The digital control algorithm uses Linear Quadratic Regulator (LQR), which provides optimum control gain to attain the vertical equilibrium position of the inverted pendulum, to make the system stable. To filter the noisy outputs of inertial sensors, the outputs of accelerometer and gyroscope are fused using complementary filter algorithm to give reliable tilt angle information. Over the period of time, microcontrollers are providing versatile, quicker, cheaper and reliable options. The project is built with Arduino 101, powerful and cost effective development board having Intel Curie module. The Complementary Filter algorithm and LQR controller is implemented in Arduino 101. This provides the required cost effectiveness along with low power consumption. High torque DC motors is used to drive the inverted pendulum to its vertical equilibrium and balance entire physical structure. Keeping track of speed is important because the information is fed back to microcontroller in a feedback loop from wheel encoders, to minimize the error to achieve equilibrium.

1.5 OBJECTIVES

To design controller for a two wheel self balancing Robot using Arduino.

- To achieve vertical stability of the Robot in two wheels.
- To make the Robot robust to external disturbances.

1.6 METHODOLOGY

The objective is fulfilled using the following method

1. Derive dynamical equations based on theory of the inverted pendulum.
2. Form state space model for the angle of deviation and position.
3. Find a controller that can control these two conditions.
4. Modeling and simulation using Simulink and Sketch up.
5. Design controller gain in MATLAB.
6. Hardware construction and physical modeling.
7. Calibration of sensors and actuators of robot.
8. Implementation of controller using Arduino
9. Testing the Final Design.

1.7 OUTLINE OF THE REPORT

Chapter 1 of the report deals with Introduction. It includes the problem statement, Literature Review objectives of the project.

Chapter 2 of the report deals with Theoretical Background. It includes study about control techniques namely, PID and LQR.

Chapter 3 of the report deals with Mathematical modeling of the system (plant) and design of controller. It includes the Kinematics equation, dynamic equations and the controller.

Chapter 4 of the report deals with Hardware Description. It includes the Hardware specification for the stated problem and solution to it.

Chapter 5 of the report deals with MATLAB Simulation. It includes 3D model design by Google SketchUp, Interfacing the 3D model with MATLAB and the simulation results with and without controller.

Chapter 6 of this report deals with Conclusion.

Chapter 7 of this report deals with Future Improvements.

CHAPTER 2

THEORTICAL BACKGROUND

2.1 PENDULUM MODEL

2.1.1 Simple Pendulum Model

To understand the system better, analysis of a simple pendulum is crucial. According to Fig. 2.1, a mass, M and moment of inertia, J is joined by a massless rod of length L to a frictionless pivot. The angular velocity, ω and the rate of change of angular velocity, $d\omega/dt$ are given by the equation 2.1 and 2.2.

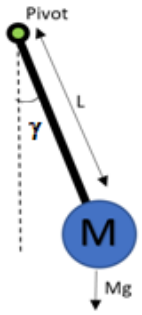


Fig. 2.1: Simple Pendulum

$$\frac{d\gamma}{dx} = \omega \tag{2.1}$$

$$\frac{d\omega}{dt} = \frac{-\mathrm{MgL}\sin\gamma}{J} \tag{2.2}$$

2.1.2 Inverted Pendulum Model

The Inverted pendulum is a classical problem in control systems to explore the unstable dynamics. The common examples are given in Fig. 2.2.

(a) (b)

Fig. 2.2(a) Inverted Pendulum on a linear cart; (b) Self-Balancing Robot

2.2 CONTROLLERS

To maintain the robot upright, commonly used controllers are Proportional – Integral - Derivative (PID) and the Linear Quadratic Regulator (LQR). Other research works have also explored the use of Linear - Gaussian Control (LQG), Fuzzy Logic and Pole-Placement method. In our project, the robot stability is controlled by LQR.

2.2.1 PID

A proportional–integral–derivative (PID) controller is a control loop feedback widely used in industrial control systems and in a wide range of applications requiring continuously modulated control. The controller, continuously calculates an error value, e(t), as the difference between a desired set point SP = r(t), measured process variable PV = y(t), and applies a correction based on proportional, integral, and derivative terms. The controller attempts to minimize the error over time by adjustment of a control variable u(t), determined by a weighted sum of the control terms. The Fig. 2.3 shows the block diagram of the PID controller in a feedback loop.

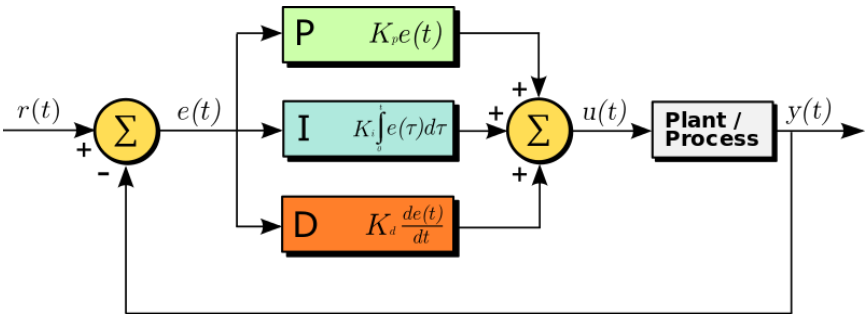


Fig. 2.3 Block diagram of PID controller

- Term P is proportional to the current value of the error, e(t).
- Term I accounts for past values of the error and integrates them over time to produce I term. The integral term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error.
- Term D is the estimate of future trend of the error, based on its current rate of change. It is sometimes called anticipatory control, as it is effectively

seeking to reduce the effect of the error by exerting a control influence generated by the rate of error change. The more rapid the change, the greater the controlling or dampening effect.

The overall control function can be expressed mathematically as,

$$u(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{d}{dt} e(t) \quad (2.3)$$

Where, u(t) is the output of the controller and K_p, K_i and K_d are non-negative coefficients for proportional, integral and derivative terms respectively.

2.2.2 LQR

To control a linear system, LQR-control can be applied. The states of the dynamical system are described by a state space model including the matrices A, B and C, as seen in Fig. 2.4.

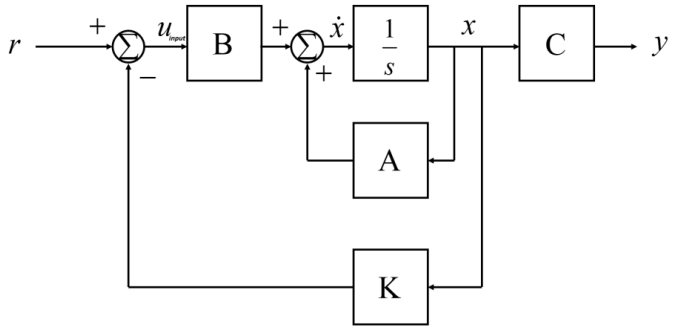


Fig. 2.4 Block representation of state space system.

The linear quadratic regulator problem deals with minimization of the cost function. In the equation Q is positive definite (or positive semi definite) symmetric matrix and R is positive definite symmetric matrix. The two cost matrices Q and R, are known as weight matrices are introduced in order to weigh how fast each corresponding element in the state vector need to reach their desired end values.

$$J = \int (x^T Q x + u^T R u)\, dt \quad (2.4)$$

The control effort, u_{input} is given by,

$$u_{input} = -K x \quad (2.5)$$

where, K is the gain matrix, defined as,

$$K = R^{-1}B^{T}P \tag{2.6}$$

and P is solved from the Riccatti Equation, given in 2.6,

$$Q + A^{T}P + PA - PBB^{T}P = 0 \tag{2.7}$$

For a state space system that is observable and controllable, P has a unique Solution and the closed loop system poles are strictly in the right half plane. LQR is a form of optimal control that aims to minimize the performance index whist taking into account the control effort, as often, higher input effort would imply higher energy consumption. LQR control requires derivation of the state-space model of the system, thus it is more challenging to implement. The mathematical description of LQR is given in Appendix, A1.

Table 2.1 Comparison between PID and LQR

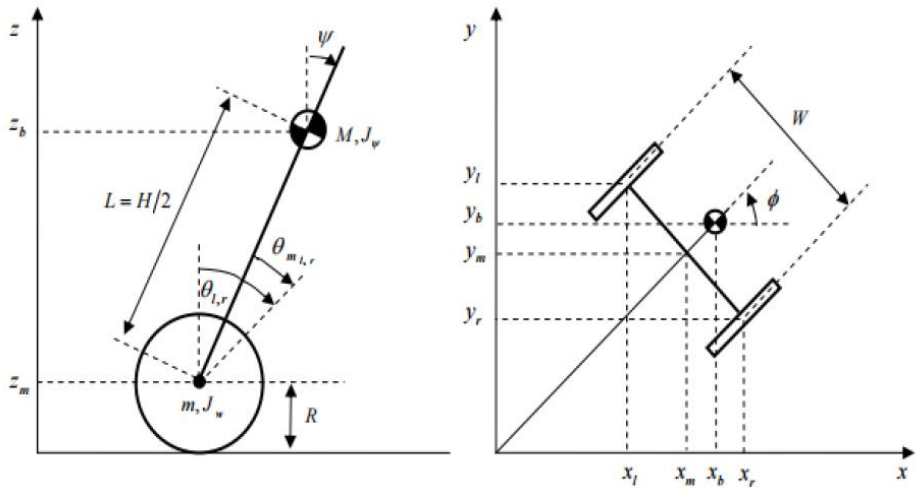
Controller	**Advantages**	**Disadvantages**
PID	- Easy to implement - Does not require the state space model	- Applicable to Single Input, Single Output System(SISO) - Does not perform as well as LQR
LQR	- Applicable to MIMO systems - Performs better	- More challenging - Requires derivation of State-Space Model

CHAPTER 3

MATHEMATICAL MODEL

3.1 TWO WHEELED INVERTED PENDULUM MODEL

The robot's mathematical description was divided in three parts, one for the inverted pendulum, one for the wheels and one for the electrical motor system. The pendulum and the wheel have three equations each, one for rotational direction and two for x and y - direction. The pendulum has ψ as its vertical angle and $\dot{\psi}$ as its angular velocity and the wheels have θ and $\dot{\theta}$ for the angle and angular velocity and the system has yaw, ϕ(drift) and $\dot{\phi}$ drift velocity. Fig. 3.1 shows the orientation of the Robot along the three axes.



Ψ : Body Pitch Angle $\theta_{l,r}$: Wheel Angle(Left, Right) $\theta_{m\,l,r}$: DC Motor Angle

Fig. 3.1 Inverted Pendulum Model

3.2 SYSTEM PARAMETERS

The parameters that were used in the mathematical model are:

Table 3.1 System parameters

Symbol	**Units**	**Description**
g = 9.81	m/s^2	Gravitational acceleration on earth

M = 0.03328	Kg	Wheel weight
R = 0.0325	M	Wheel radius
$J_W = 1.75x10^{-5}$	kg m^2	Wheel moment of inertia
M = 0.663	Kg	Body weight including wheel and battery
W = 0.15	m	Body width
D = 0.9	m	Body depth
H= 0.118	m	Body height
L=H/2	m	Distance of centre of mass from wheel axle
$J_\Psi = 2.304x10^{-3}$	kg m^2	Body pitch inertia moment
$J_\Phi = 1.21x10^{-3}$	kg m^2	Body yaw inertia moment
$Jm = 1.44x10^{-5}$	kg m^2	DC motor inertia moment
Rm = 6.00	Ohms	DC motor resistance
K_b=0.29	V s/rad	DC motor back EMF constant
K_t = 0.29	Nm/A	DC motor torque constant
N = 21		Gear ratio
fm = 0.0022	Nm/rad/s	Friction coefficient between body and DC motor
fw=0	N/m/s	Friction coefficient between wheel and floor

3.3 SYSTEM KINEMATICS

Kinematic equation describes the system trajectory without considering the forces that caused the motion. The diagrammatic representation of the system and its trajectory is shown in Fig. 3.2.

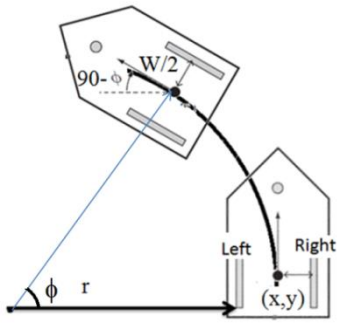


Fig. 3.2 System and its trajectory

From the geometry of motion (as shown in Fig. 3.3), we have

$$\phi = \frac{R}{r}\theta_l = \frac{R}{r+\frac{W}{2}}\theta = \frac{R}{r+W}\theta_r \tag{3.1}$$

where, $R\theta_l$ and $R\theta_r$ are the distance covered by left wheel and right wheel

$R\theta$ is the distance covered by centre of mass

Thus by solving 3.1 and 3.2, we get the roll and drift angle to be,

$$\phi = \frac{R}{W}(\theta_r - \theta_l) \tag{3.2}$$

$$\theta = \frac{1}{2}(\theta_r + \theta_l) \tag{3.3}$$

Thus the left and right wheel position is given by,

$$x_m = R\theta cos\phi \tag{3.4}$$

$$y_m = R\theta sin\phi \tag{3.5}$$

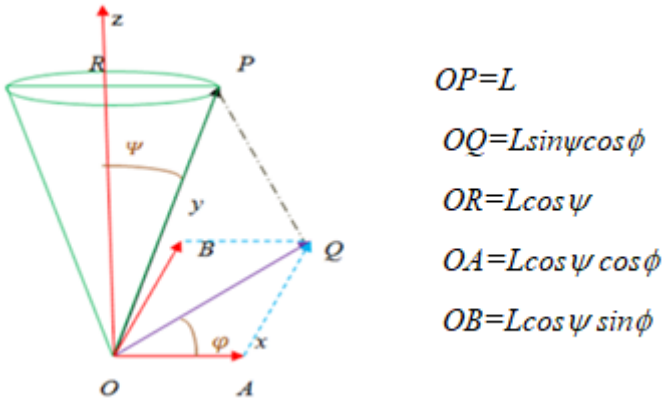


Fig. 3.3 Geometry of the body

$$x_l = x_m - \frac{W}{2}sin\phi \tag{3.6}$$

$$x_r = x_m + \frac{W}{2}sin\phi \tag{3.7}$$

$$y_l = y_m + \frac{W}{2}cos\phi \tag{3.8}$$

$$y_r = y_m - \frac{W}{2}cos\phi \tag{3.9}$$

Next, we have for the coordinates of the centre of mass to be,

$$x_b = x_m + L\,sin\psi\,cos\phi \tag{3.10}$$

$$y_b = y_m + L\,sin\psi\,cos\phi \tag{3.11}$$

3.4 MOTOR DYNAMICS

The Electrical equivalent model of the motor is shown in the Fig. 3.4.

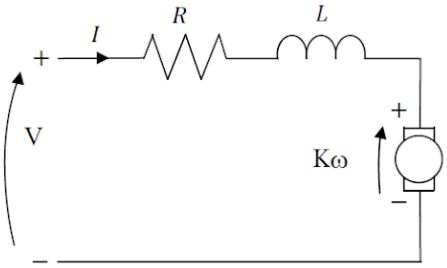


Fig. 3.4 - Electrical Motor Model

From the circuit, the terminal Voltage is given by,

$$V = IR + K_e\omega + L\frac{dI}{dt} \tag{3.12}$$

Since the electrical time constant is much smaller than the mechanical the inductance can be neglected which gives equation

$$V \approx IR + K_e\omega \tag{3.13}$$

Shaft Torque is given by,

$$T = nK_tI \tag{3.14}$$

The shaft torque needs to overcome the inertia of the motor as well as the viscous damping and motor friction.

3.5 SYSTEM DYNAMICS

Dynamics is concerned with the study of forces and torques and their effect on motion.

A. Newtonian mechanics

Dynamics is governed by Newton's laws of motion. This is done by drawing free body diagram as shown in Fig. 3.5.

The force on cart along horizontal axis is given by,

$$MR\ddot{\theta} + bR\dot{\theta} + N = F \tag{3.15}$$

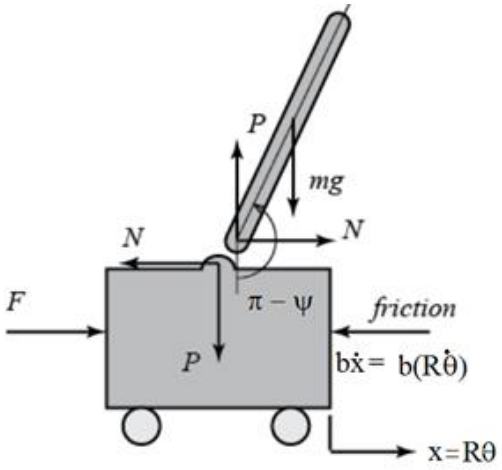


Fig. 3.5 Forces acting on the system

The force on pendulum along horizontal axis is given by,

$$N = mR\ddot{\theta} - ml\ddot{\psi}cos\psi - ml\dot{\psi}^2 sin\psi \tag{3.16}$$

Equating the above equations, we get,

$$(M+m)R\ddot{\theta} + bR\dot{\theta} - ml\ddot{\psi}\cos\psi - ml\dot{\psi}^2 sin\psi = F \tag{3.17}$$

The force on pendulum along its perpendicular axis is given by,

$$Psin\psi - Ncos\psi - mgsin\psi = ml\ddot{\psi} - mR\ddot{\theta}cos\psi \tag{3.18}$$

The torque about the center of mass of the pendulum is,

$$-Plsin\psi + Nlcos\psi = I\ddot{\psi} \tag{3.19}$$

Thus,

$$(I+ml^2)\ddot{\psi} + mglsin\psi = mlR\ddot{\theta}cos\psi \tag{3.20}$$

Since the analysis will be employed only to linear systems,

$$cos\psi = 1;\ \ sin\psi = \psi;\ \ \dot{\psi}^2 = 0 \tag{3.21}$$

The final equations of motion are,

$$-(I+ml^2)\psi + mgl\ddot{\psi} = mlR\ddot{\theta} \tag{3.22}$$

$$(M+m)R\ddot{\theta} + bR\dot{\theta} + ml\ddot{\psi} = u \tag{3.23}$$

Taking Laplace transform,

$$(I+ml^2)\psi(s)s^2 - mgl\psi(s) = mlX(s)s^2 \tag{3.24}$$

$$(M+m)X(s)s^2 + bX(s)s - ml\psi(s)s^2 = U(s) \tag{3.25}$$

Solving the equations,

$$\frac{X(s)}{\psi(s)} = \left[\frac{I+ml^2}{ml} - \frac{g}{s^2}\right] \tag{3.26}$$

$$(M+m)\left[\frac{I+ml^2}{ml} - \frac{g}{s^2}\right]\psi(s)s^2 + b\left[\frac{I+ml^2}{ml} - \frac{g}{s^2}\right]\psi(s)s - ml\psi(s)s^2 = U(s) \tag{3.27}$$

$$\frac{\psi(s)}{U(s)} = \frac{\frac{ml}{q}s^2}{s^4 + \frac{b(l+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \tag{3.28}$$

Where,

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

$$P_{pend}(s) = \frac{\psi(s)}{U(s)} = \frac{\frac{ml}{q}s}{s^3 + \frac{b(l+ml^2)}{q}s^2 - \frac{(M+m)mgl}{q}s - \frac{bmgl}{q}} \tag{3.29}$$

The transfer function with the cart position X(s) as the output can be derived as,

$$P_{cart}(s) = \frac{X(s)}{U(s)} = \frac{\frac{(I+ml^2)s^2 - gml}{q}}{s^4 + \frac{b(l+ml^2)}{q}s^3 - \frac{(M+m)mgl}{q}s^2 - \frac{bmgl}{q}s} \tag{3.30}$$

The equations can be represented in state space form as,

$$\begin{bmatrix}\dot{\theta}\\ \ddot{\theta}\\ \dot{\psi}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2gl^2}{I(M+m)+Mml^2} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Mml^2} & 0\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ \psi\\ \dot{\psi}\end{bmatrix} + \begin{bmatrix}0\\ \frac{(I+ml^2)}{R[I(M+m)+Mml^2]}\\ 0\\ \frac{ml}{I(M+m)+Mml^2}\end{bmatrix}u \tag{3.31}$$

$$y = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ \psi\\ \dot{\psi}\end{bmatrix} + \begin{bmatrix}0\\ 0\end{bmatrix}u \tag{3.32}$$

Where,

$$u = \frac{K_b}{Rr}(V - K_e\omega)$$

Substituting we get,

$$\begin{bmatrix}\dot{\theta}\\ \ddot{\theta}\\ \dot{\psi}\\ \ddot{\psi}\end{bmatrix}=\begin{bmatrix}0 & 1 & 0 & 0\\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Ml^2} & -\frac{K_bK_e(I+ml^2)}{Rr[I(M+m)+Ml^2]} & \frac{m^2gl^2}{I(M+m)+Ml^2} & 0\\ 0 & 0 & 0 & 1\\ 0 & \frac{-mlb}{I(M+m)+Ml^2} & -\frac{mlK_bK_e}{Rr[I(M+m)+Ml^2]} & \frac{mgl(M+m)}{I(M+m)+Ml^2} & 0\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ \psi\\ \dot{\psi}\end{bmatrix}+\begin{bmatrix}0\\ \frac{(I+ml^2)K_bK_e}{Rr[I(M+m)+Ml^2]}\\ 0\\ \frac{mlK_bK_e}{Rr[I(M+m)+Ml^2]}\end{bmatrix}V \quad (3.33)$$

The equation represents the state space model with supply voltage as input.

B. Lagrangian mechanics.

Newtonian system analysis is generally not followed for complex systems because,

- The presence of motion constraints equations by invoking forces like Coriolis force complicates the analysis.
- It involves vector manipulations and it is easy to make mistakes in drawing the free body diagram.

The Lagrangian formulation of mechanics overcomes all these drawbacks and provides an elegant frame work for mechanical modeling:

For a system of N particles with P constraints on its coordinates (holonomic constraints), let (q_1,q_2,……q_{3N-P}) be 3N-P independent coordinates that are independent and let the real 3N Cartesian Coordinates of N particles be a function of these generalized coordinates.

$$\boldsymbol{r_i}=(q_{1,2},\ldots\ldots\ldots q_{3N-P})$$

$r_i(x_i,y_i,z_i)$, position vector of the i^{th} particle (i=1,2,.....N)

Then the equations of motion for those generalized coordinates is given by the celebrated Euler Lagrange Equations as

$$\frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial \dot{\mathrm{q}}_\mathrm{i}}\right)-\frac{\partial \mathrm{L}}{\partial \dot{\mathrm{q}}_\mathrm{i}}=F_{qi} \quad (3.34)$$

for i=1,2.....(3N-P) for all the generalized coordinates.

'L' is called the Lagrangian or Action and is a scalar and is defined as

$$L=T\text{-}V \quad (3.35)$$

T: Total Kinetic Energy of all the particles (in the generalized velocities)

V: Total Potential Energy of all the particles (in the generalized coordinates)

F_{qi} : Generalized force (Force along the i^{th} generalized coordinate)

The translational Kinetic Energy (T_1), Rotational Kinectic Energy (T_2) and Potential Energy (U) is given by,

$$T_1 = \frac{1}{2}m\left(\dot{x_l}^2 + \dot{y_l}^2 + \dot{z_l}^2\right) + \frac{1}{2}\left(\dot{x_r}^2 + \dot{y_r}^2 + \dot{z_r}^2\right) + \frac{1}{2}M\left(\dot{x_b}^2 + \dot{y_b}^2 + \dot{z_b}^2\right) \quad (3.36)$$

$$T_2 = \frac{1}{2}J_w\left(\dot{\theta_l}^2 + \dot{\theta_r}^2\right) + \frac{1}{2}J_\psi\left(\dot{\psi_l}^2\right) + \frac{1}{2}J_\phi\left(\dot{\phi_l}^2\right) + \frac{1}{2}n^2J_m\left(\dot{\theta_l} - \dot{\psi}\right)^2 + \frac{1}{2}n^2J_m\left(\dot{\theta_r} - \dot{\psi}\right)^2 \quad (3.37)$$

$$U = mgz_l + mgz_r + Mgz_b \quad (3.38)$$

T_1 consists of left wheel, right wheel and body's translational kinectic energy.

T_2 consists of wheel, body's rotational kinectic energy along Psi, Phi axes and Motor's rotational energy where the effective angle moved by shaft is θ – ψ.

Using Euler Lagrange Equations for the generalized coordinates are

$$\frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial \dot{\theta}}\right) - \frac{\partial \mathrm{L}}{\partial \theta} = F_\theta \quad (3.39)$$

$$\frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial \dot{\psi}}\right) - \frac{\partial \mathrm{L}}{\partial \psi} = F_\psi \quad (3.40)$$

$$\frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial \dot{\phi}}\right) - \frac{\partial \mathrm{L}}{\partial \phi} = F_\phi \quad (3.41)$$

The generalised coordinate forces are given by,

$$[(2m+m)R^2 + 2J_w + 2n^2J_m]\ddot{\theta} + (MLRcos\psi - 2n^2J_m)\ddot{\psi} - MLR\dot{\psi}^2sin\psi = F_\theta$$

$$(MLRcos\psi - 2n^2J_m)\ddot{\theta} + \left(ML^2 + 2n^2J_m + J_\psi\right)\ddot{\psi} - MgLsin\psi - MLR\dot{\phi}^2sin\psi cos\psi = F_\psi$$

$$\left[\frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}(J_w + n^2J_m) + ML^2sin^2\psi\right]\ddot{\phi} + 2ML^2\dot{\psi}\dot{\phi}sin\psi cos\psi = F_\phi \quad (3.42)$$

To transform the generalised force to the applied force, D'Alembert's Principle is used.

$$Q_a = \sum_{i=1}^{N} F_i^{(applied)} \frac{\partial \mathrm{r_i}}{\partial \mathrm{q_a}} \tag{3.43}$$

Q_a includes F_θ, $F_\emptyset$, F_ψ and F_i applied consists of left and right. Substituting in the above equation,

$$F_\theta = F_l + F_r \tag{3.44}$$

$$F_\phi = \frac{W}{2R}(F_r - F_l) \tag{3.45}$$

And the Applied force produces opposite effects on generalised coordinates i.e., θ and ψ.

$$F_\psi = -F_l - F_r \tag{3.46}$$

The applied torque on Left and Right wheel is given as the sum of Load and friction torque.

$$\begin{aligned} F_l &= nK_t i_r + f_m(\dot{\psi} - \dot{\theta}_l) \\ F_r &= nK_t i_r + f_m(\dot{\psi} - \dot{\theta}_r) \end{aligned} \tag{3.47}$$

The Motor equation is given by,

$$V = R_m + K_b(\dot{\theta} - \dot{\psi}) \tag{3.48}$$

For small angles of ψ, the following approximations are carried out,

$$\begin{aligned} &sin\psi = \psi \qquad cos\psi = 1 \\ &\dot{\psi}^2 = 0 \qquad \dot{\phi}^2 = 0 \\ &sin^2\psi = 0 \end{aligned} \tag{3.49}$$

The above equations in matrix form can be represented as,

$$E\begin{bmatrix}\ddot{\theta}\\ \ddot{\psi}\end{bmatrix} + F\begin{bmatrix}\dot{\theta}\\ \dot{\psi}\end{bmatrix} + G\begin{bmatrix}\theta\\ \psi\end{bmatrix} = H\begin{bmatrix}v_l\\ v_r\end{bmatrix} \tag{3.50}$$

Where,

$$E = \begin{bmatrix}(2m+M)R^2 + 2J_w + 2n^2J_m & MLR - 2n^2J_m \\ MLR - 2n^2J_m & ML^2 + J_\psi + 2n^2J_m\end{bmatrix} \tag{3.51}$$

$$F = 2\begin{bmatrix}\beta & -\beta \\ -\beta & \beta\end{bmatrix} \quad G = \begin{bmatrix}0 & 0 \\ 0 & -gLM\end{bmatrix} \quad H = \begin{bmatrix}\alpha & \alpha \\ -\alpha & -\alpha\end{bmatrix} \tag{3.52}$$

Where β and α are given by,

$$\alpha = \frac{nK_t}{R_m} \qquad \beta = \frac{nK_tK_b}{R_m}$$

$$I\left[\ddot{\phi}\right] + J\left[\dot{\phi}\right] = k[v_r - v_l] \tag{3.53}$$

$$I = \frac{1}{2}mW^2 + J_\phi + \frac{W^2}{2R^2}(J_w + n^2J_m) \tag{3.54}$$

$$J = \frac{W^2}{2R^2}(\beta + f_w) \tag{3.55}$$

$$K = \frac{W}{2R}\alpha \tag{3.56}$$

$$\dot{x} = Ax + Bu \tag{3.57}$$

Thus the state space representation with States and Outputs is as follows,

$$x_1 = \begin{bmatrix}\theta\\ \psi\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \qquad y_1 = \begin{bmatrix}\theta\\ \psi\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix}$$

$$x_1 = \begin{bmatrix}\phi\\ \dot{\phi}\end{bmatrix} \qquad y_1 = \begin{bmatrix}\phi\\ \dot{\phi}\end{bmatrix} \tag{3.58}$$

Taking Inputs as,

$$u = \begin{bmatrix}v_l\\ v_r\end{bmatrix} \tag{3.59}$$

And State Matrix as,

$$A_1 = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & gLM(MLR - 2n^2J_m) & 2\beta\left(ML^2 + J_\psi + MLR\right) & -2\beta(ML^2 + J_\Psi + MLR) \\ 0 & -glM[(M')R^2 + 2J_w + 2n^2J_m] & -2\beta[(M')R^2 - 2J_w - MLR] & 2\beta[(M')R^2 + 2J_w + MLR] \end{bmatrix}$$

Where, (2m + M) =M' (3.60)

$$B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \alpha(ML^2 + J_\psi - MLR) & \alpha(ML^2 + J_\psi - MLR) \\ -(MR^2 + 2mR^2 + MLR) & -(MR^2 + 2mR^2 + MLR) \end{bmatrix} \tag{3.61}$$

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.62}$$

$$A_2 = \begin{bmatrix} 0 & 1 \\ -\frac{K}{I} & -\frac{J}{I} \end{bmatrix} \tag{3.63}$$

$$B_2 = \begin{bmatrix} 0 & 1 \\ -\frac{K}{I} & -\frac{K}{I} \end{bmatrix} \quad (3.64)$$

$$C_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad (3.65)$$

$(v_l + v_r)$ affects only the dynamics of x_1 and $(v_l - v_r)$ affects only the yaw dynamics. Let, $u_1 = (v_l + v_r)$ and $u_2 = (v_l - v_r)$.

$$v_l = \frac{1}{2}(u_1 + u_2)$$
$$v_r = \frac{1}{2}(u_1 - u_2) \quad (3.66)$$

Thus,

$$\dot{x}_1 = A_1 x_1 + B'_1 u_1$$
$$\dot{x}_2 = A_2 x_2 + B'_2 u_2 \quad (3.67)$$

Where, $B1'$: First column of $B1$

$B2'$: First column of $B2$

3.6 LQR CONTROLLER

LQR controller design for the system was formulated. The linear quadratic regulator problem deals with minimization of the cost function.

$$J = \int_0^\infty (x^T Q x + u^T R u)\, dt$$

The control law that minimizes the cost function is linear, given by,

$$u(t) = -R^{-1} B^T P x(t)$$

The control law guarantees that, $x(t)$ and $u(t) \to 0$ as $t \to \infty$ at a rate that is an optimal tradeoff between control effort and performance, demanded by Q and R.

The crucial and difficult task in the LQR controller design is a choice of the weighting matrices. We generally select weighting matrices Q and R to satisfy expected performance criterion. The different Q and R values give a different system response. The system will be more robust to disturbance and the settling time will be shorter if Q is larger (in a certain range). But there is no straightforward way to select these weighting matrices and it is usually done through an iterative simulation process.

For various values of Q and R matrix, the controller gain is calculated by solving Ricatti's Equation in MATLAB. An advantage of using the LQR optimal control scheme is that system designed will be stable and robust, except in the case where the system is not controllable.

LQR controller for the models obtained from both mathematical analyses and state space matrix namely, Lagrangian and Newtonian Mechanics are implemented.

CHAPTER 4

MECHATRONIC SYSTEM

The section aims to provide an overview of the design considerations and implementation of the robot physically.

4.1 MECHANICAL SYSTEM

The general design of the robot was a rectangular body on two wheels. The wheels are placed parallel to each other. The battery was placed as on the body.

4.1.1 Main Frame

The main frame of the body consists of three shelves and four steel rods that can represent the four legs of the shelf. As in the Fig. 4.1, two shelves are made of Acrylic to support the battery and fixing the motor wheel set up. The other shelf is a PCB layer that is mounted with Arduino 101 and motor driver circuit.

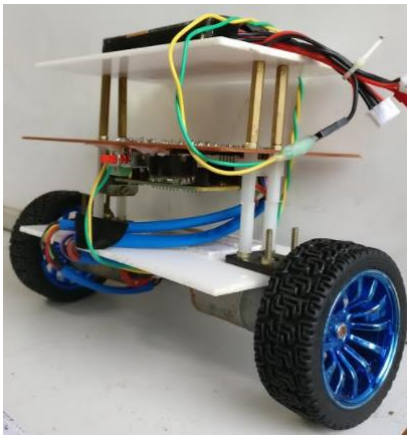


Fig. 4.1 Two Wheeled Self Balancing Robot

4.1.2 Wheel

The wheels are of dimensions, 65×26.5 mm. The wheels are selected by considering the angular velocity of the motor and the desired speed of the robot to keep the robot upright. The wheels are made of light plastic with a rubber tire and attached to the motor axis hub.

4.2 ELECTRICAL SYSTEM

4.2.1 Microcontroller

The microcontroller used in the robot is Arduino 101. It contains the Intel Curie Module, designed to integrate the core's low power consumption and high performance with the Arduino ease-of-use. The 101 adds Bluetooth Low Energy capabilities and has on-board 6-axis accelerometer/gyroscope. Arduino is programmed using the Arduino Software (IDE) which enables quick development. The module contains two tiny cores, an x86(Quark) and a 32-bit ARC architecture core, both clocked at 32MHz. It requires an input voltage between 7-12V for optimal functioning and is powered and programmed through a USB-cable.

4.2.2 Motor

To determine the appropriate motors for the robot, the first consideration is the minimum torque required. To estimate the torque, the model is considered,

$$\tau = \|\boldsymbol{r}\|\|\boldsymbol{F}\|\sin\theta, \quad (4.1)$$

where, $\boldsymbol{r}$ is the position vector and θ is angle between force and position vectors. For distance between the pivot point and the centre of mass (L) is 12cm, the maximum tilt angle (θ_{max}) is 40° and the mass of the robot (m) is 0.7kg, the minimum torque, $\tau = mgLsin\theta = 0.530\ Nm$.

Table 4.1 Motor Specification

Rated Voltage	6 V
No-Load Speed	210 rpm
Max efficiency	2.0 Kg.cm/170rpm/2.0W/0.60A
Max power	5.2 Kg.cm/110rpm/3.1W/1.10A
Stall torque	10 Kg.cm 3.2A
Retarder reduction ratio	1 : 21

To begin with, the torque required by the motor is estimated and the compromise between torque and RPM is done. The Motor used is DC Shunt Gear Motor, as in the Fig. 4.2, with specification, as in the Table 4.1.

Fig. 4.2 Motor with Encoder

4.2.3 Motor Driver

To power the motors a motor driver, L298N is needed for two 6V DC-motors. The motor chosen is designed to operate at 6V and has a stall current of 2A. The microcontroller cannot supply this power, thus a Full bridge driver is required to allow the motor to be controller in both directions. The motor driver circuit consists of L298N and diodes to protect the microcontroller and battery from back EMF. The L298N chips can operate with a supply voltage up to 42V, and can supply an RMS current of 4A (peak 5A).

The motors is controlled by setting up PWM on the direction pin for deciding which direction the motor would go and the speed of the motor, where 255 was full power of supply voltage Vin and 0 is completely OFF.

4.2.4 Encoder

The encoders in the motors are quadrature encoders. There are two signals and the rising edges are counted to establish the displacement and angular velocity. The direction in which the shaft is rotating is determined by the phase difference between the two signals. This is shown in the Fig. 4.3. To count the edges and determine the phase different, rising edge interrupts were setup on channel A.

$$Rotor\ Displacement\ [degress] = edgecount \times \frac{360}{\text{NPPR}} \qquad (4.2)$$

Where, N_{ppr} is the number of pulses per revolution of the encoder and T is the time window between readings. The Encoder specification is given in Table 4.2.

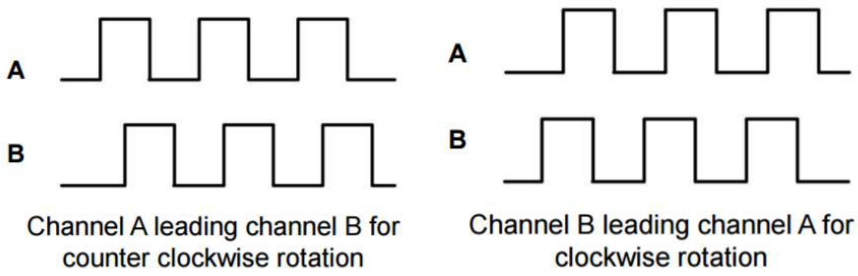


Fig. 4.3 Quadrature pulse of Encoder

Table 4.2 Encoder Specification

Encoder motor end	11 signals
Hall Resolution (N_{ppr})	234

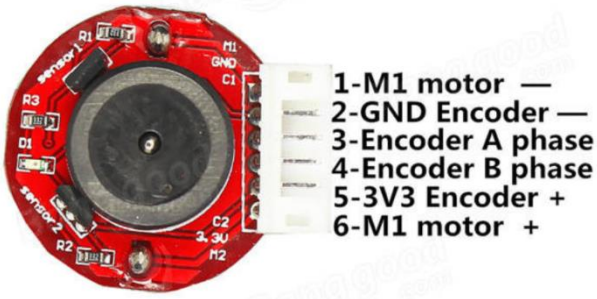


Fig. 4.4 Motor Encoder

4.2.5 PCB

The PCB is designed and routed using Eagle software. Refer appendix, A8 for

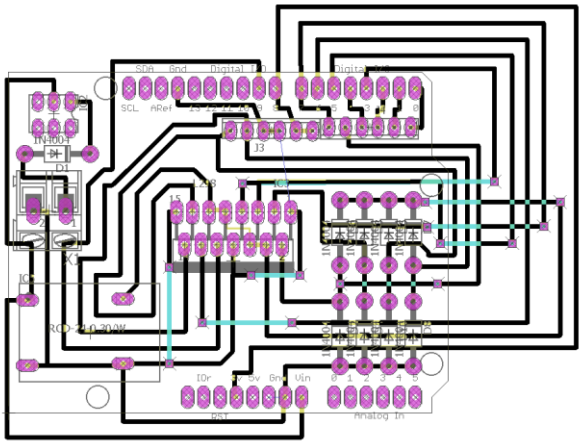


Fig. 4.5 PCB Layout

schematic design of the circuit. The PCB board is made by copper coated plastic material. The PCB Layout is shown in the Fig. 4.5. The prepared layer is printed on the copper clad by a laminator and then etched using $FeCl_3$ solution. Now the tracks are obtained and the PCB is drilled by using PCB drilling machine. After drilling, the motor driver IC and other components are soldered on the PCB. The PCB is designed by hands and used after short circuit and voltage check.

4.2.6 Inertial Sensor

The BMI l60 is a highly integrated, low power inertial measurement unit (IMU) that provides precise acceleration (angle) and angular rate (gyroscopic) measurement. The data flow is shown in the Fig. 4.6.

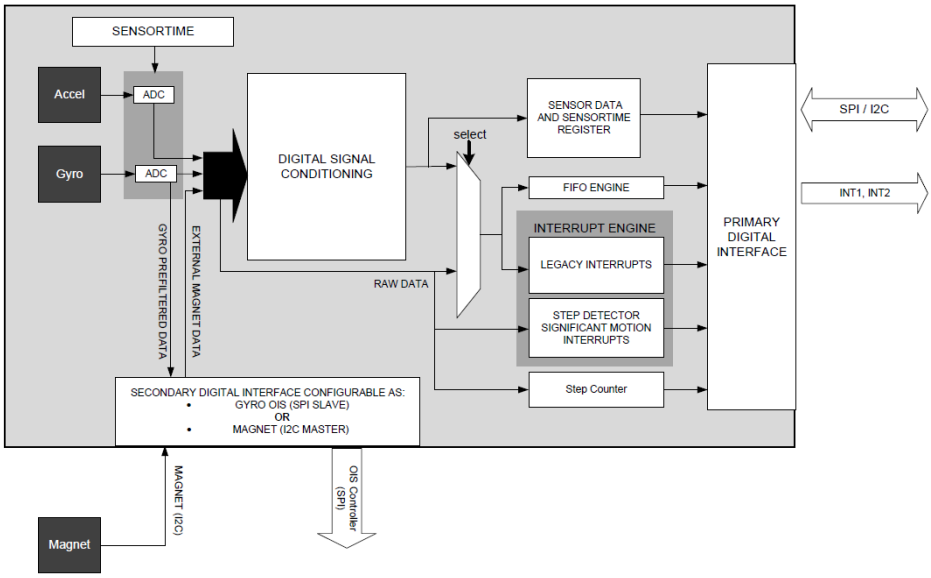


Fig. 4.6 Block Diagram for Dataflow

The BMI l60 integrates:

- 16 bit digital, triaxial accelerometer
- 16 bit digital, triaxial gyroscope

a. Accelerometer

The force of gravity always acts perpendicular to the earth's surface. The accelerometer measures the total external acceleration of the balancing robot, which includes the gravitational and motion accelerations. From

the Fig. 4.7, Following, the direction cosine method, one can use the X, Y and Z-axis gravitational acceleration measurements to calculate the tilt angle. Although this method gives quickly, it is easily influenced by external forces and noise. Table 4.3 shows the 3db cut off frequency for accelerometer.

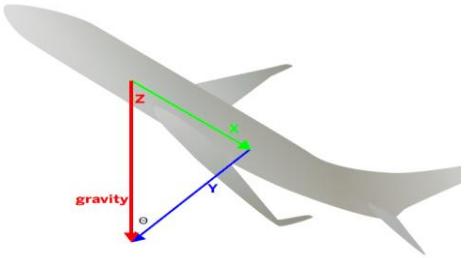


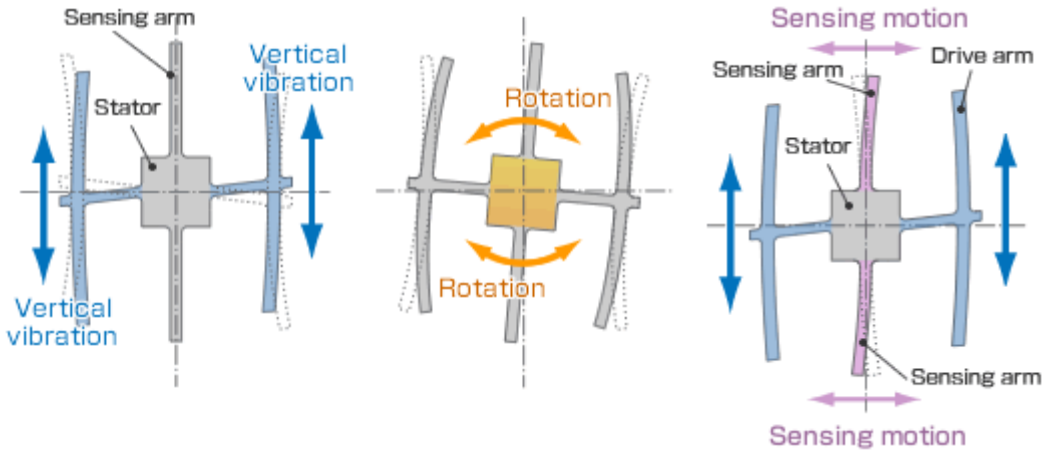
Fig. 4.7 Vector Representation

Table 4.3 3-db Cut-off Frequency for Accelerometer

Accelerometer ODR(Hz)	12.5	25	50	100	200	400	800	1600
3dB Cutoff Frequency	5.06	10.12	20.25	40.5	80	162	324	684

b. Gyroscope

A gyroscope measures angular velocity, the rate of change in orientation angle. The motion of a pair of sensing arms produces a potential difference from which angular velocity is sensed. The angular velocity is converted to, and output as, an electrical signal. Fig 4.3, shows the working of gyroscope. Table 4.4 shows the 3db cut off frequency for gyroscope.



(a) (b) (c)

Fig. 4.8 Working of Gyroscope

a) When the gyro is rotated, the Coriolis force acts on the drive arms, producing vertical vibration.
b) Direction of rotation
c) The stationary part bends due to vertical drive arm vibration, producing a sensing motion in the sensing arms.

Table 4.4 3-dB cut-off Frequency for Gyroscope

Gyroscope ODR(Hz)	25	50	100	200	400	800	1600	3200
3dB Cutoff Frequency	10.7	20.8	39.9	74.6	136.6	254.6	523.9	890

c. Sensor Fusion

The gyro measurements can only give us angles relative to initial orientation, as we simply integrate the angular velocity and it is prone to accumulate error. In addition to this, the sensor has bias errors, the angle estimate drift. But, it is less sensitive to linear mechanical movements.

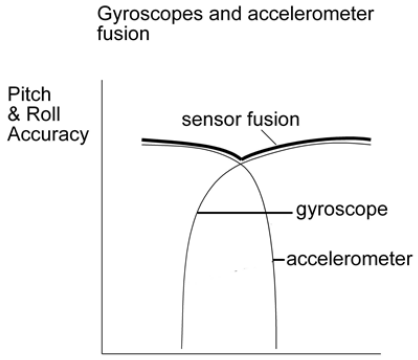


Fig. 4.9 Sensor Fusion

On the other hand, the accelerometer provides an absolute measure of inclination, but the output signal is often corrupted with mechanical noise and vibration.

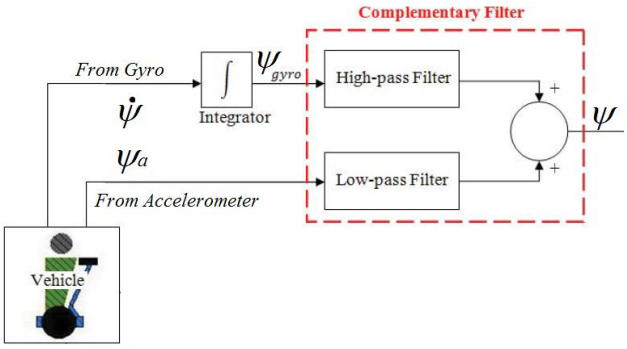


Fig. 4.10 Complementary Filter

As in the Fig. 4.9, averaging the data that comes from accelerometers and gyroscope can produce a better estimate of orientation than obtained using a single sensor data alone. We use complementary filter which fuses the accelerometer and gyro data by passing through a 1st order low pass and high pass filter respectively and adding their outputs, as shown in Fig. 4.10

The filter is represented as,

$$\psi_k = \alpha\psi_{k-1} + (1-\alpha)(\psi_a)_k + \alpha\dot{\psi}_k\Delta t \quad (4.3)$$

where, $\alpha = \frac{T}{\Delta t}/(1+\frac{T}{\Delta t})$

The Control Loop and Sampling rate of gyro and accelerometer: 200 Hz ($1/\Delta t$) and the Cut off frequency for low pass and high filter for raw data: 74.6 Hz (1/T) The mathematical description of the filter is done in Appendix, A2. The implementation is described in Appendix, A6.

4.2.7 Power Source

To provide power while maintaining the robot mobile, a Lithium Polymer (Li-Po) 3 – cell, 12 V, 2200mAh - 8C Battery was used. The battery can supply a current of up to 24 A. Given that the stall current of each of the motors is 2A and the remaining components (Arduino101 and encoders) have an estimated current draw of 500 mA, thus the battery can power up. The motor operates with 6V which is fed from a battery through buck converter.

4.3 HARDWARE MODEL

The hardware model that was used is shown in Fig. 4.11.

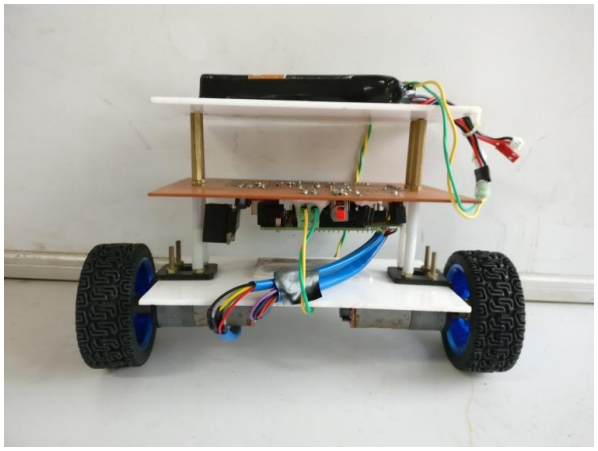


Fig. 4.11 Hardware model

4.4 OVERALL DESIGN

This section explains the sequence of events in the robot.

4.4.1 Algorithm

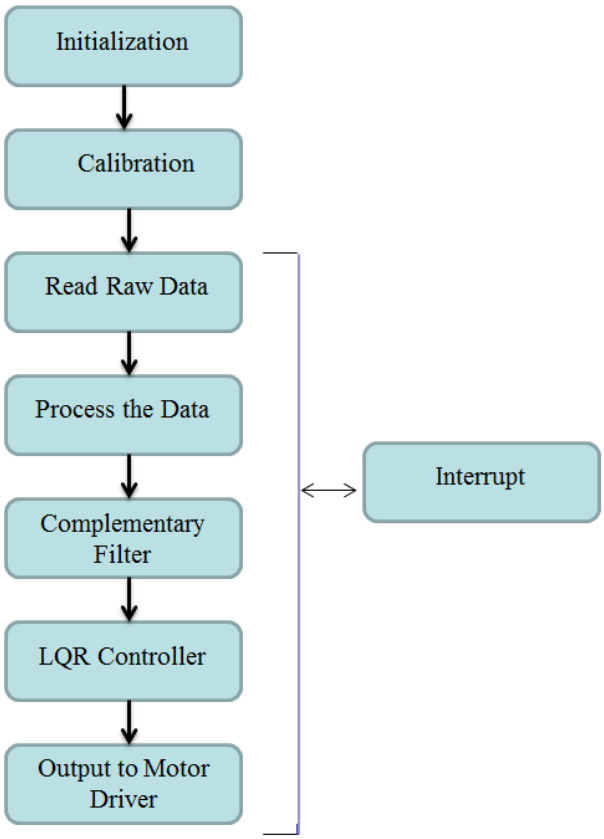


Fig. 4.12 Work Flow

4.4.2 Layout

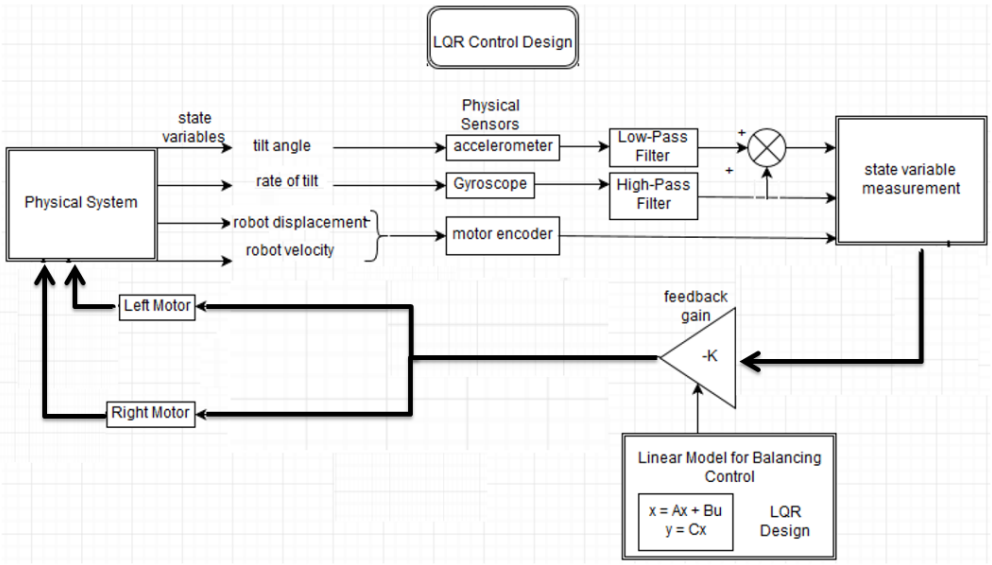


Fig. 4.13 Working Layout

4.4.3 Working

- The robot undergoes initialization and the sensor calibration takes place to find the offset and bias error.
- The sensor data is acquired and the data is processed after filtering and the encoder data, acquired using interrupts are the states
- Using LQR, the gain is computed by Ricatti Equation.
- The Pitch (ψ) is obtained by sensor fusion using complementary filter. The Roll angle and velocity of wheels ($\theta_{l,}\theta_{r}$) and Yaw (ϕ) is obtained from encoder.
- In Real time the states are obtained and the actuation fed to the motor is calculated as the product of states and gain.

  Refer appendix for Arduino codes of the controller.

CHAPTER 5

SIMULATION

5.1 SOFTWARE MODEL

5.1.1 3D Sketchup Model Design

The 3D model of the Self balancing robot as shown in Fig. 5.1, is designed using Google SketchUp software. SketchUp is a 3D Modelling computer program used to draw 3D models of automobiles, buildings and hardware set up for various engineering projects. In the 3D SketchUp model, the right wheel along with the right motor form one group, left wheel with left motor form another and the chassis forms a separate group. Once the model is done, it needs to be interfaced with MATLAB Simulink for further simulation. This is done with the help of VR Sink tool (Virtual Reality Toolbox). Once it is interfaced with MATLAB, the system behaviour can be observed for different values of parameters.

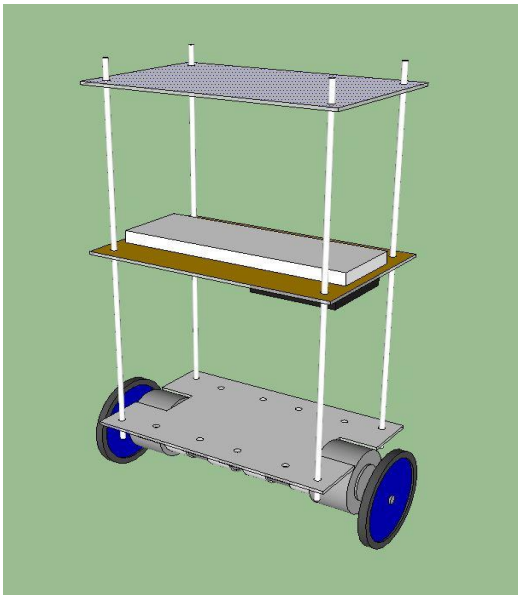


Fig 5.1 3D SketchUp Model

5.1.2 Simulink Model

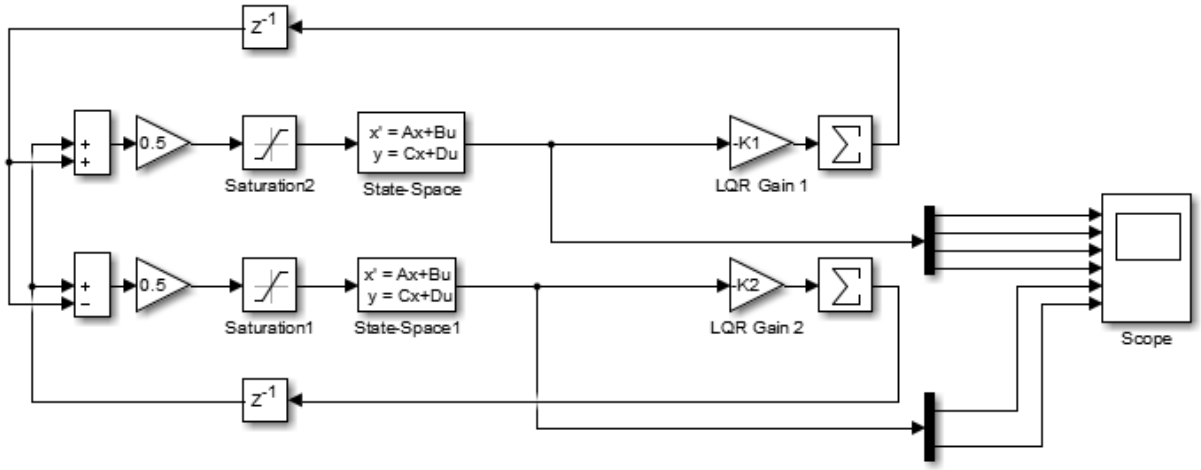


Fig. 5.2 Simulink Model with Controller

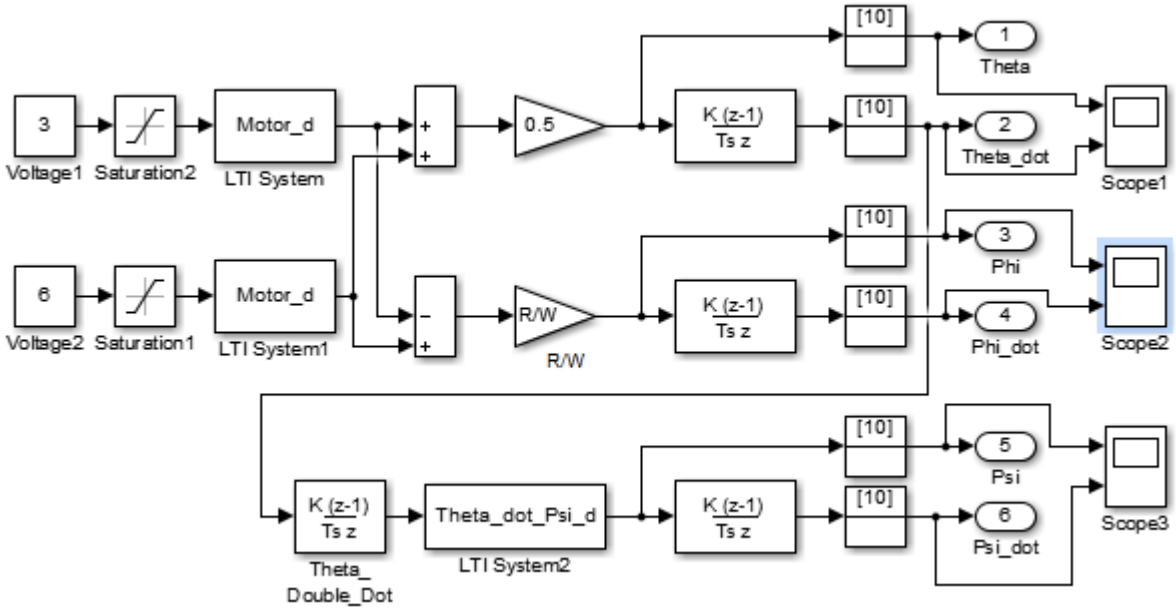


Fig. 5.3 Simulink Model without Controller

5.1.3 MATLAB Interfacing

Once the 3D Model is generated with the help of SketchUp, in order to understand the system behaviour, it is interfaced with MATLAB. To perform this, the model is exported as VRML File in to the MATLAB environment. Once this is done, the VRML File is linked with the VR Sink Block of the 3D Animation Toolbox. The system behavior is viewed as a simulation.

5.2 SIMULATION OUTPUT

5.2.1 Analytical Results

The output obtained from MATLAB simulation with and without controller is described in this section. The Matlab codes are given in given in the Appendix.

(i) Using Newtonian Method

Without controller,

$$A = \begin{bmatrix} 0 \\ -5.6041 \\ -0.1428 \\ 5.5651 \end{bmatrix}$$

Since all the Eigen values are not negative, the system is not completely stable

With controller,

$$(A - BK) = \begin{bmatrix} -5.5978 + 0.4070\text{i} \\ -5.5978 - 0.4070\text{i} \\ -0.8494 + 0.8323\text{i} \\ -0.8494 - 0.8323\text{i} \end{bmatrix}$$

Since all the eigen values are negative, the system is stable

(ii) Using Lagrangian Method

Without controller,

$$A_1 = \begin{bmatrix} 0.0000 + 0.0000\text{i} \\ -0.1055 + 6.9580\text{i} \\ -0.1055 - 6.9580\text{i} \\ -37.9582 + 0.0000\text{i} \end{bmatrix}$$

$$A_2 = \begin{bmatrix} -0.7343 \\ -36.6045 \end{bmatrix}$$

Since all the Eigen values are not negative, the system is not completely stable

With controller,

$$(A_1 - B_1K_1) = \begin{bmatrix} -37.9699 + 0.0000\text{i} \\ -0.0021 + 0.0000\text{i} \\ -0.1059 + 6.9584\text{i} \\ -0.1059 - 6.9584\text{i} \end{bmatrix}$$

$$(A_2 - B_2K_2) = \begin{bmatrix} -0.7352 \\ -36.6304 \end{bmatrix}$$

Since all the Eigen values are negative, the system is stable

5.2.2 Graphical Results

(i) Using Newtonian Method

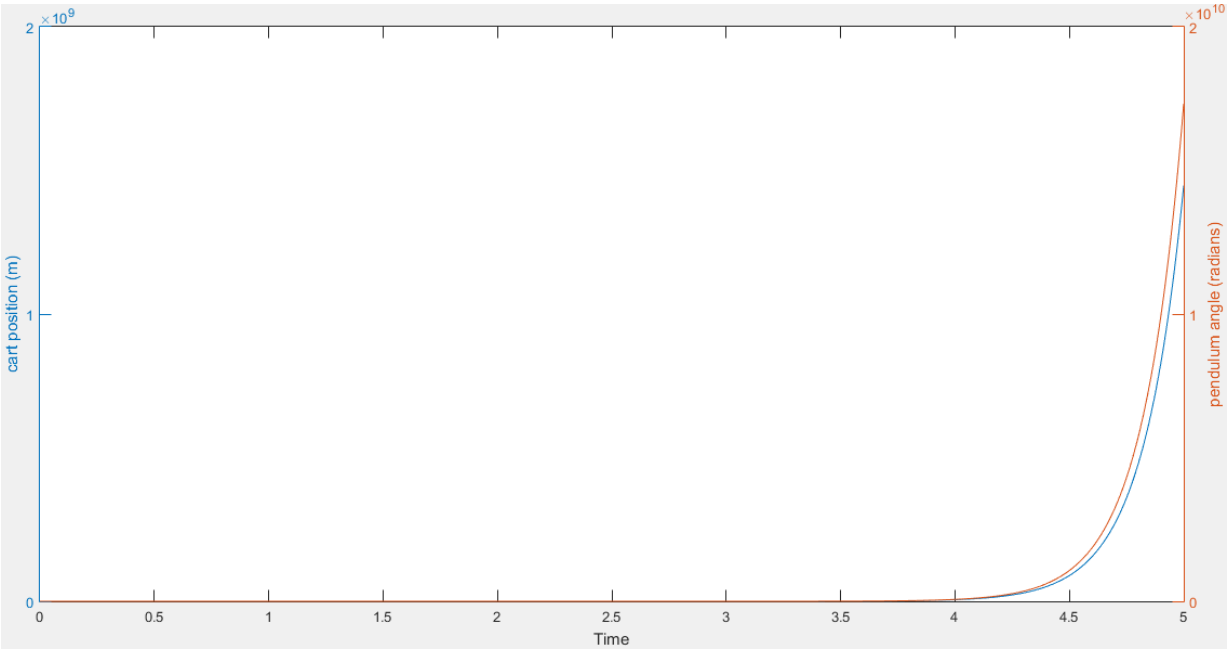


Fig. 5.4 Cart and pendulum position vs. time (Without Controller)

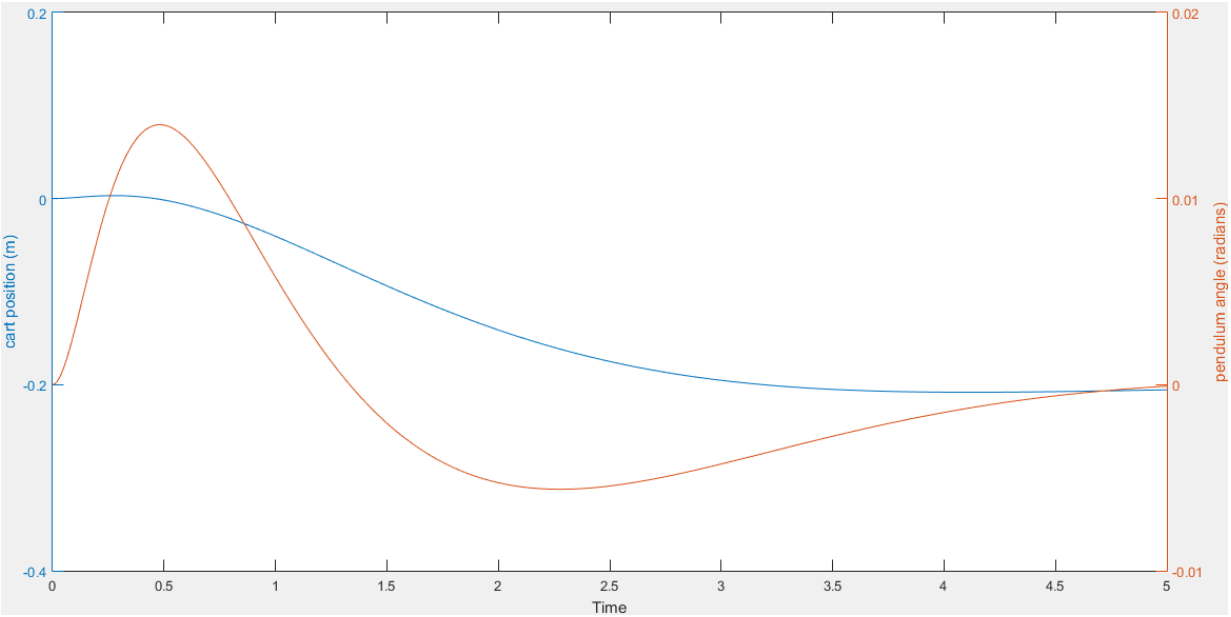


Fig. 5.5 Cart and pendulum position vs. time (With Controller)

(ii) Using Lagrangian Method

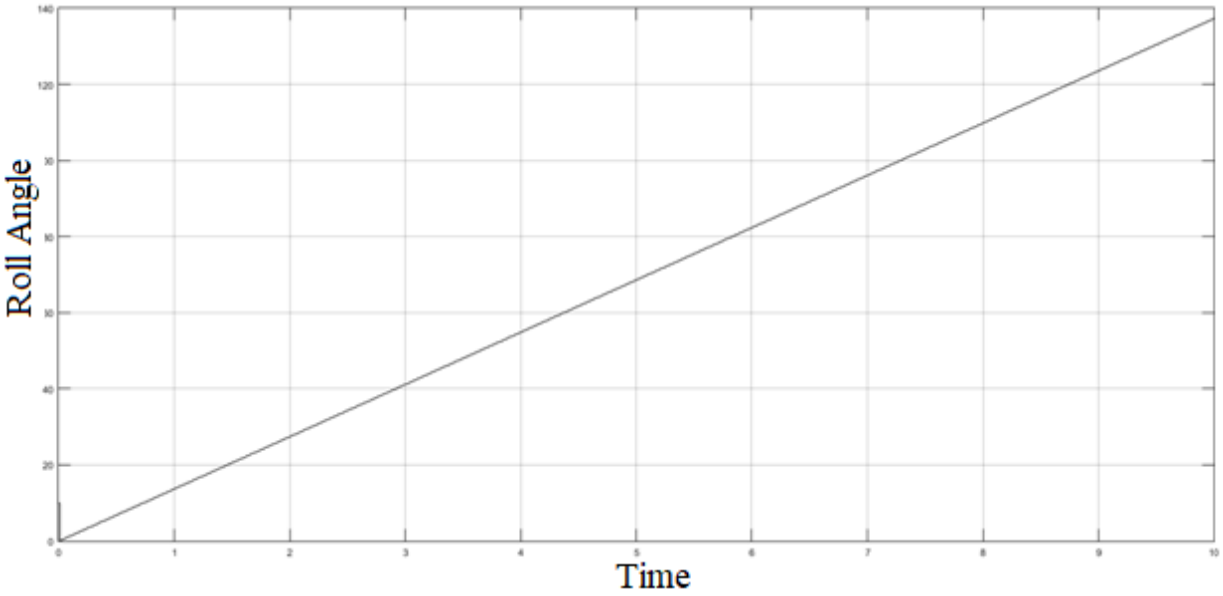


Fig. 5.6(a) Roll Angle vs. time (Without controller)

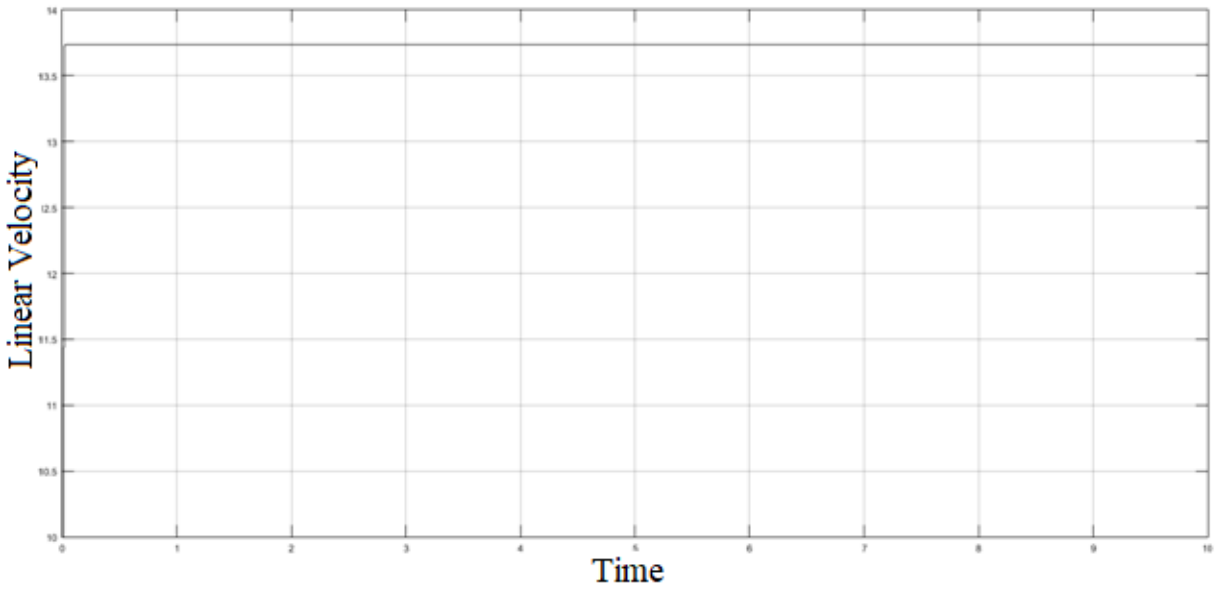


Fig. 5.6(b) Linear velocity vs. time (Without controller)

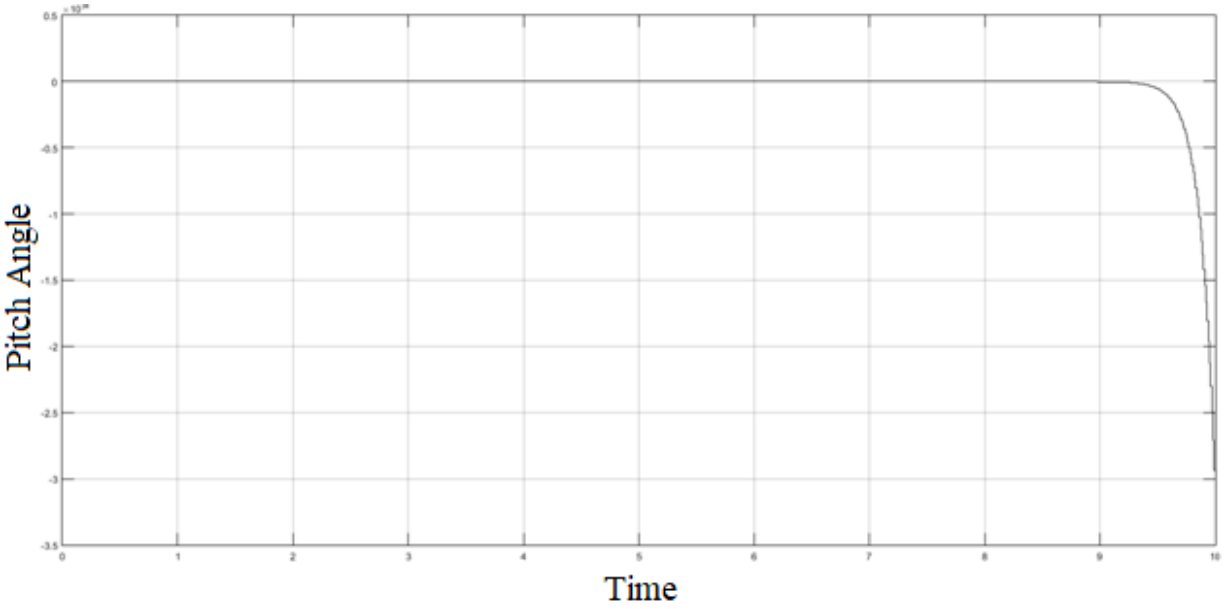


Fig. 5.7(a) Pitch Angle vs. time (Without controller)

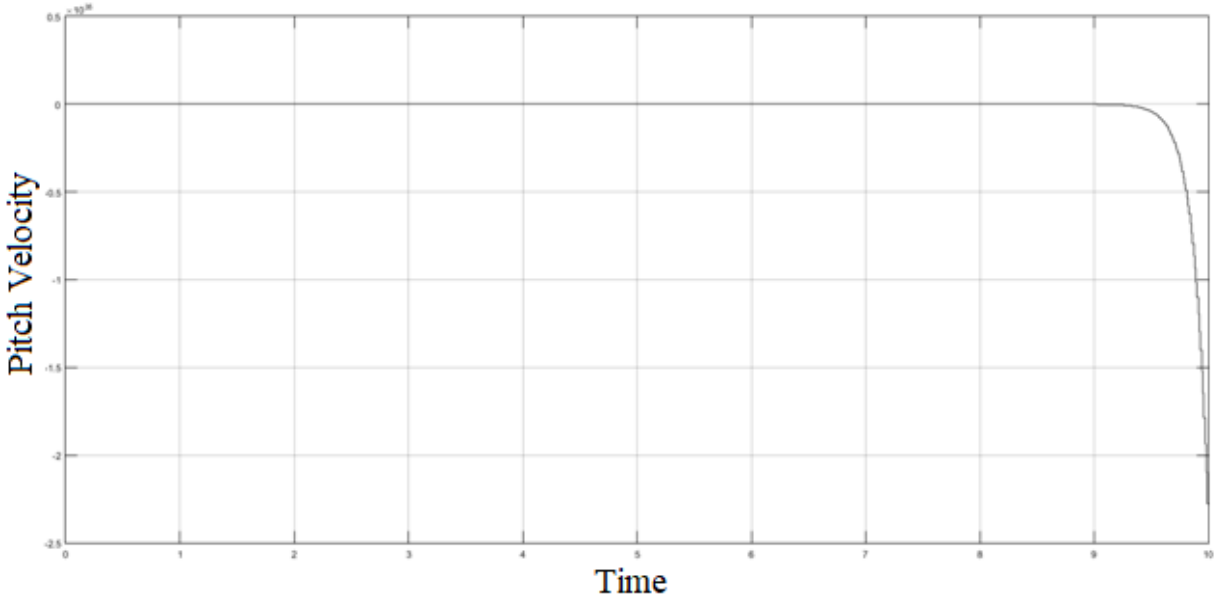


Fig. 5.7(b) Pitch velocity vs. time (Without controller)

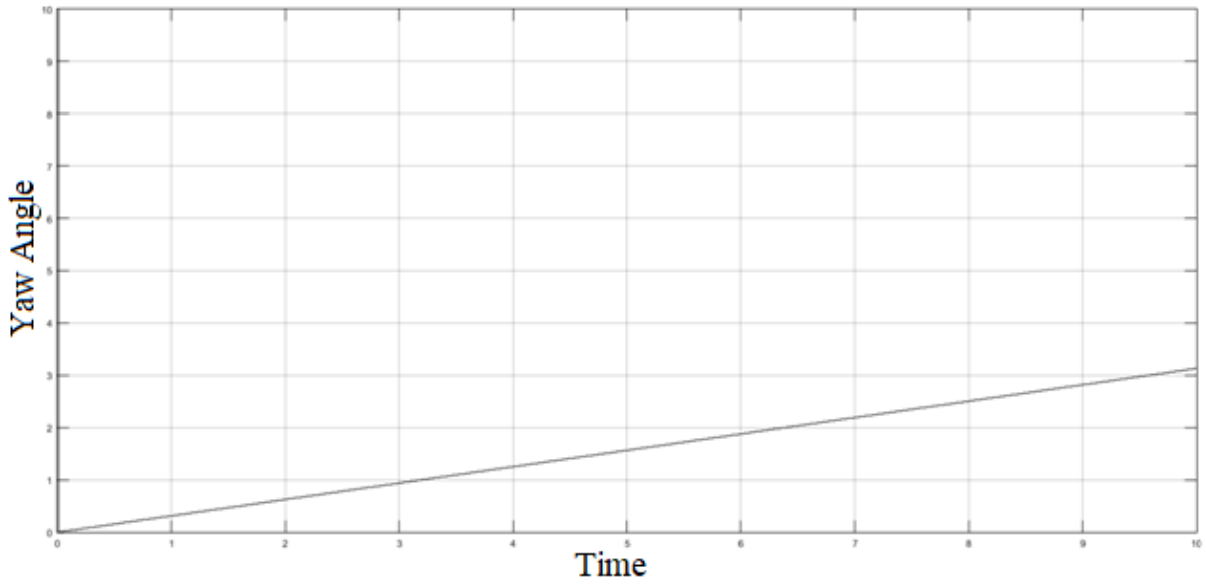


Fig. 5.8(a) Yaw Angle vs. time (Without controller)

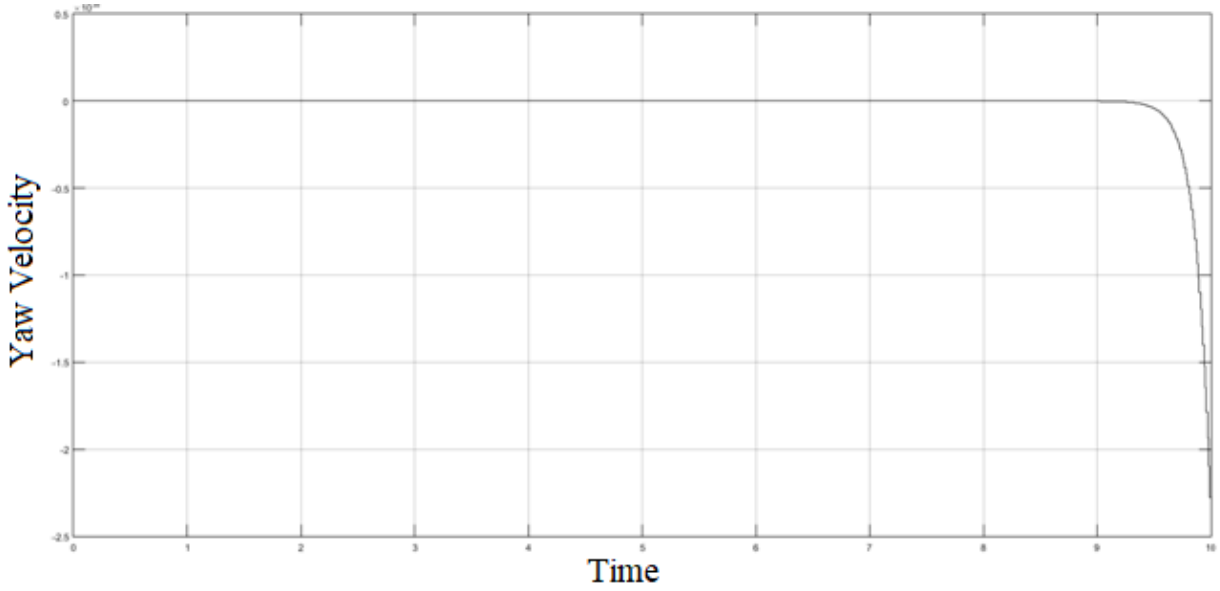


Fig. 5.8(b) Yaw velocity vs. time (Without controller)

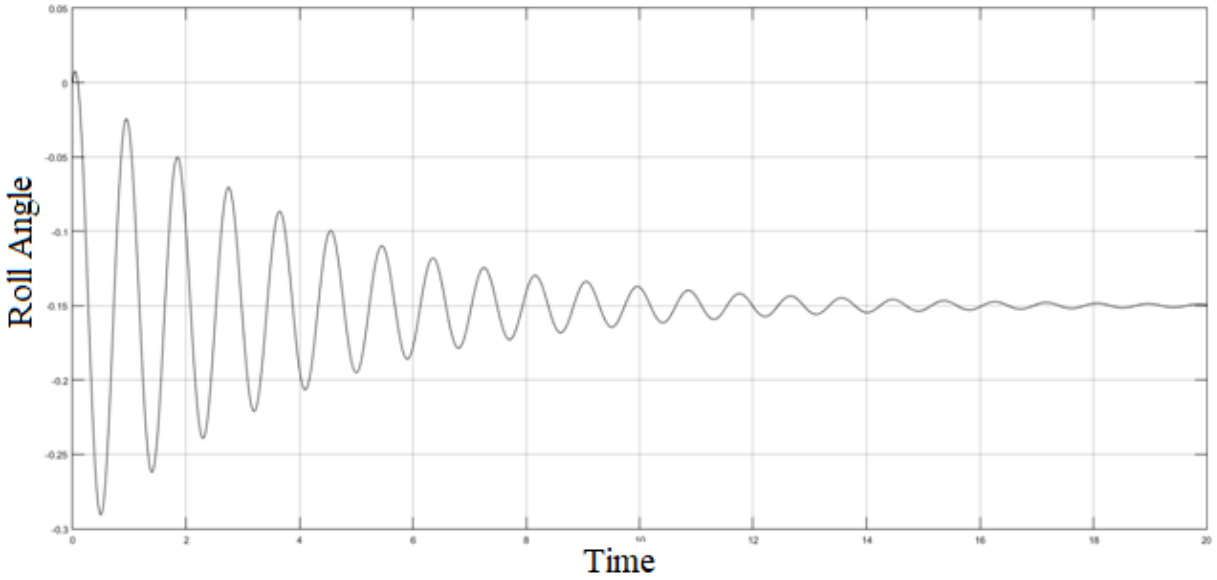


Fig. 5.9(a) Roll Angle vs. time (With controller)

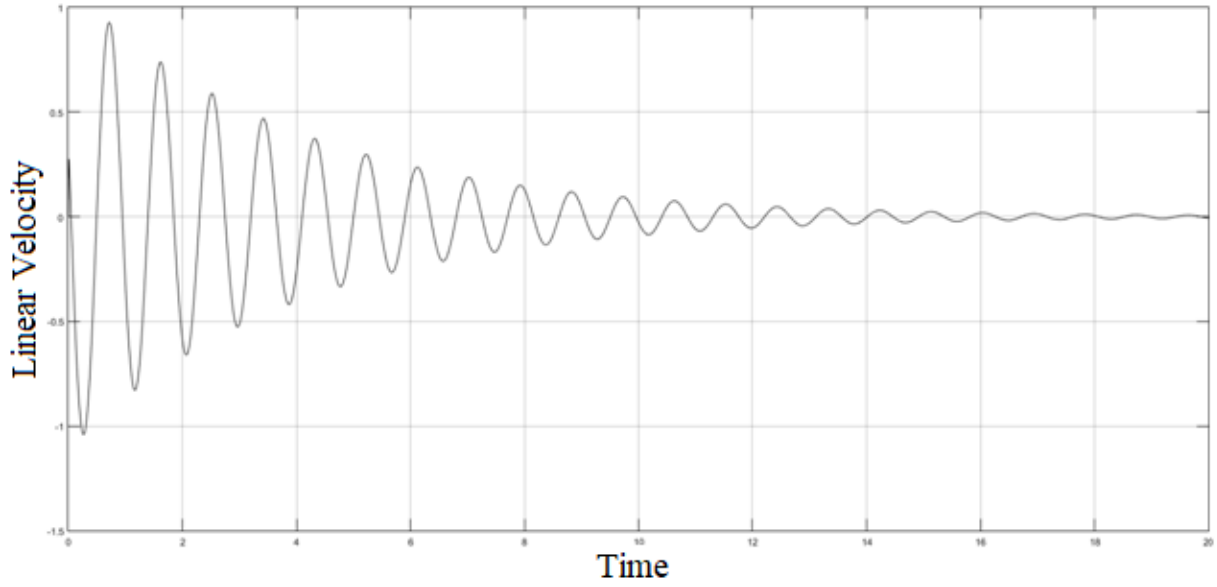


Fig. 5.9(b) Linear velocity vs. time (With controller)

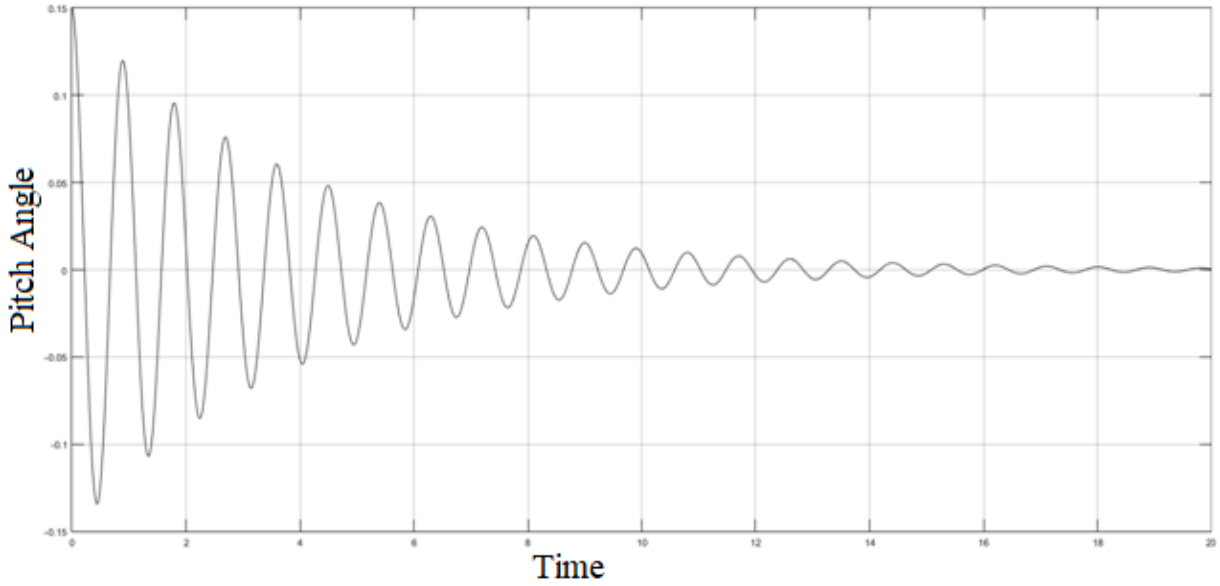


Fig. 5.10(a) Pitch Angle vs. time (With controller)

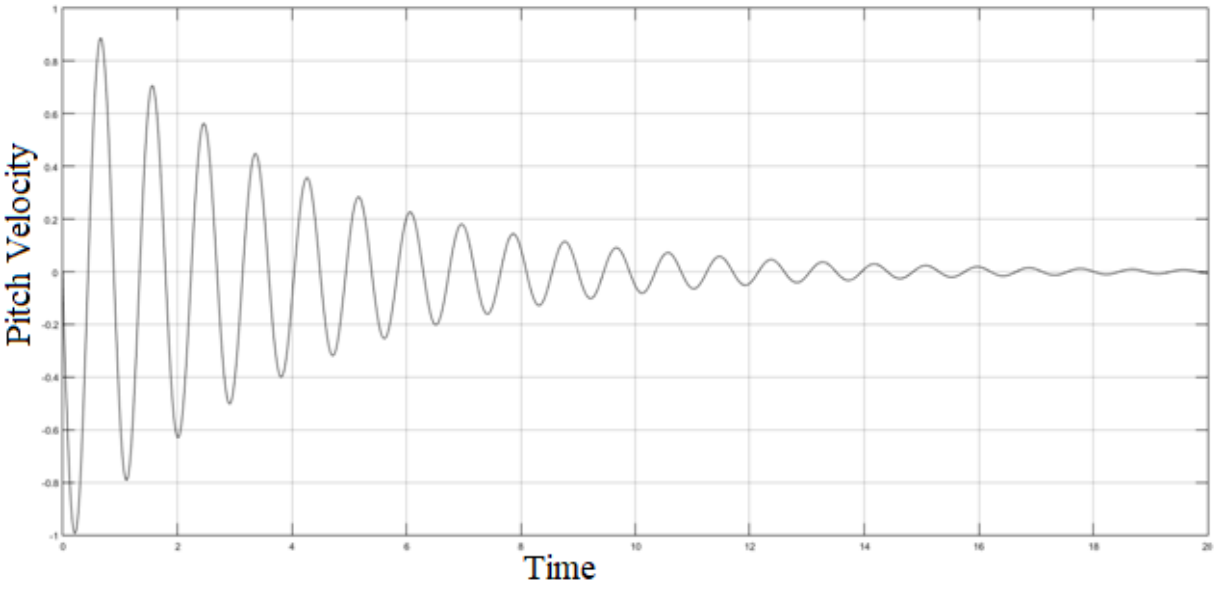


Fig. 5.10(b) Pitch velocity vs. time (With controller)

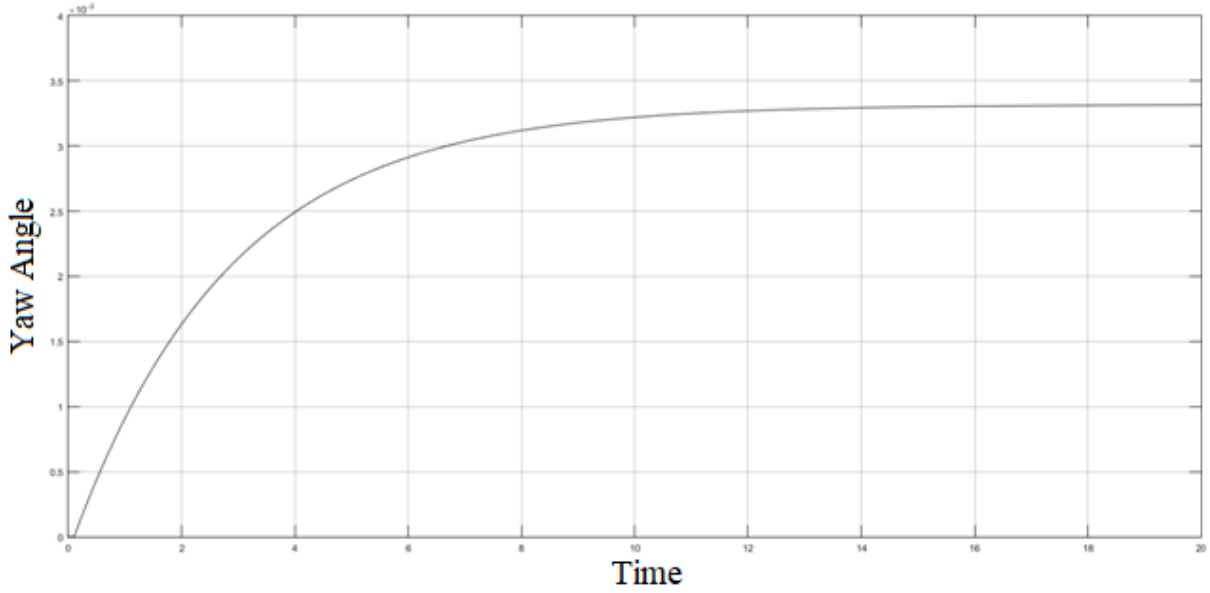


Fig. 5.11 Yaw Angle vs. time (With controller)

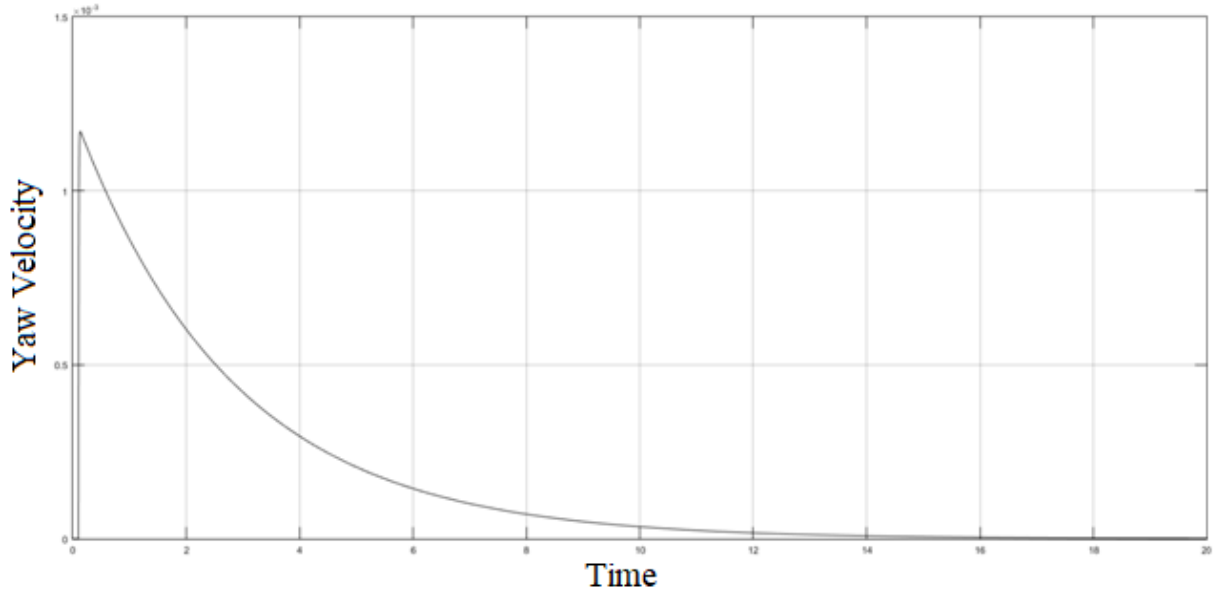


Fig. 5.11 Yaw velocity vs. time (With controller)

CHAPTER 6

CONCLUSION

The two wheeled Self balancing robot is implemented using a LQR with Lagrangian and Newton Analysis on the self - balancing robot coded in Arduino, by calculating gain from MATLAB. The robot stays upright and tracks required trajectories as commanded. Before the hardware, the simulations were performed for various controllers using MATLAB coding. The 3D model was designed using Google SketchUp and integrated into the MATLAB environment with the help of its 3D animation facility.

The project has been carried out from the design and production of specific parts to the integration of electronic, mechanical and software sections. Due to the need to use the knowledge in the fields of mechanics, electronics, programming and control, this project is extremely interdisciplinary and as such one of the most representative mechatronic problems.

CHAPTER 7

FUTURE IMPROVEMENTS

Further work will include increasing the level of autonomy of the robot by adding a vision system, thus allowing the robot to avoid obstacles. Also, by improving the components of the robot we hope to achieve higher speeds.

Another possible extension is to combine the Arduino system and other sensors such ultrasonic and IR senses, GPS, digital compass and Camera to address other advanced applications, e.g. obstacle avoidance and perimeter following.

This system serves as a benchmark for testing controllers. So,

- Non-linear controllers can be designed and their performance can be tested.
- Intelligent controllers can be designed and tested.
- This system can act as a model to perform comparative study on performance for various controllers.

Also, Stability analysis of the linearized control system on actual non-linear system can be quantitatively investigated.

A. APPENDICIES

A1. LQR Mathematical Analysis

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

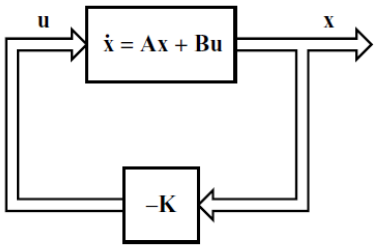
determines the matrix $\mathbf{K}$ of the optimal control vector

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t)$$

so as to minimize the performance index

$$J = \int_0^\infty (\mathbf{x}^*\mathbf{Q}\mathbf{x} + \mathbf{u}^*\mathbf{R}\mathbf{u})\, dt$$

where $\mathbf{Q}$ is a positive-definite (or positive-semidefinite) Hermitian or real symmetric matrix and $\mathbf{R}$ is a positive-definite Hermitian or real symmetric matrix.
The matrices $\mathbf{Q}$ and $\mathbf{R}$ determine the relative importance of the error and the expenditure of this energy.



$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} - \mathbf{B}\mathbf{K}\mathbf{x} = (\mathbf{A} - \mathbf{B}\mathbf{K})\mathbf{x}$$

In the following derivations, we assume that the matrix $\mathbf{A} - \mathbf{B}\mathbf{K}$ is stable, or that the eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$ have negative real parts.

$$J = \int_0^\infty (\mathbf{x}^*\mathbf{Q}\mathbf{x} + \mathbf{x}^*\mathbf{K}^*\mathbf{R}\mathbf{K}\mathbf{x})\, dt$$

$$= \int_0^\infty \mathbf{x}^*(\mathbf{Q} + \mathbf{K}^*\mathbf{R}\mathbf{K})\mathbf{x}\, dt$$

Let us set

$$\mathbf{x}^*(\mathbf{Q} + \mathbf{K}^*\mathbf{R}\mathbf{K})\mathbf{x} = -\frac{d}{dt}(\mathbf{x}^*\mathbf{P}\mathbf{x})$$

where $\mathbf{P}$ is a positive-definite Hermitian or real symmetric matrix. Then we obtain

$$\mathbf{x}^*(\mathbf{Q} + \mathbf{K}^*\mathbf{R}\mathbf{K})\mathbf{x} = -\dot{\mathbf{x}}^*\mathbf{P}\mathbf{x} - \mathbf{x}^*\mathbf{P}\dot{\mathbf{x}} = -\mathbf{x}^*\big[(\mathbf{A} - \mathbf{B}\mathbf{K})^*\mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{B}\mathbf{K})\big]\mathbf{x}$$

Comparing both sides of this last equation and noting that this equation must hold true for any $\mathbf{x}$, we require that

$$(\mathbf{A} - \mathbf{B}\mathbf{K})^*\mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{B}\mathbf{K}) = -(\mathbf{Q} + \mathbf{K}^*\mathbf{R}\mathbf{K})$$

The performance index J can be evaluated as

$$J = \int_0^\infty \mathbf{x}^*(\mathbf{Q} + \mathbf{K}^*\mathbf{R}\mathbf{K})\mathbf{x}\, dt = -\mathbf{x}^*\mathbf{P}\mathbf{x}\Big|_0^\infty = -\mathbf{x}^*(\infty)\mathbf{P}\mathbf{x}(\infty) + \mathbf{x}^*(0)\mathbf{P}\mathbf{x}(0)$$

Since all eigenvalues of $\mathbf{A} - \mathbf{B}\mathbf{K}$ are assumed to have negative real parts, we have $\mathbf{x}(\infty) \to \mathbf{0}$. Therefore, we obtain

$$J = \mathbf{x}^*(0)\mathbf{P}\mathbf{x}(0)$$

Thus, the performance index J can be obtained in terms of the initial condition $\mathbf{x}(0)$ and $\mathbf{P}$.

To obtain the solution to the quadratic optimal control problem, we proceed as follows: Since $\mathbf{R}$ has been assumed to be a positive-definite Hermitian or real symmetric matrix, we can write

$$\mathbf{R} = \mathbf{T}^*\mathbf{T}$$

it is zero, or when

$$\mathbf{TK} = (\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P}$$

Hence,

$$\mathbf{K} = \mathbf{T}^{-1}(\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P} = \mathbf{R}^{-1}\mathbf{B}^*\mathbf{P}$$

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t) = -\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P}\mathbf{x}(t)$$

$$(\mathbf{A}^* - \mathbf{K}^*\mathbf{B}^*)\mathbf{P} + \mathbf{P}(\mathbf{A} - \mathbf{B}\mathbf{K}) + \mathbf{Q} + \mathbf{K}^*\mathbf{T}^*\mathbf{T}\mathbf{K} = \mathbf{0}$$

which can be rewritten as

$$\mathbf{A}^*\mathbf{P} + \mathbf{P}\mathbf{A} + \left[\mathbf{TK} - (\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P}\right]^*\left[\mathbf{TK} - (\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P}\right] - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P} + \mathbf{Q} = \mathbf{0}$$

The minimization of J with respect to $\mathbf{K}$ requires the minimization of

$$\mathbf{x}^*\left[\mathbf{TK} - (\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P}\right]^*\left[\mathbf{TK} - (\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P}\right]\mathbf{x}$$

with respect to $\mathbf{K}$. minimum occurs when it is zero, or when

$$\mathbf{TK} = (\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P}$$

Hence,

$$\mathbf{K} = \mathbf{T}^{-1}(\mathbf{T}^*)^{-1}\mathbf{B}^*\mathbf{P} = \mathbf{R}^{-1}\mathbf{B}^*\mathbf{P}$$

$$\mathbf{u}(t) = -\mathbf{K}\mathbf{x}(t) = -\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P}\mathbf{x}(t)$$

$$\mathbf{A}^*\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^*\mathbf{P} + \mathbf{Q} = \mathbf{0}$$

A2. Complementary Filter Mathematical Analysis

The angle is determined by, passing the accelerometer and gyro signal to High pass and Low pass filter.

$$\psi = \frac{1}{1+Ts}\psi_{\mathrm{a}} + \frac{Ts}{1+Ts}\frac{1}{s}\dot{\psi} = \frac{\psi_a + T\dot{\psi}}{1+Ts} = \mathrm{j}$$

According to bilinear transformation,

$$1 + Ts = \left(1 + \frac{T}{\Delta t}\right) - \frac{T}{\Delta t}z^{-1} \text{ and s}$$

$$\psi_k = \alpha(\psi_{k-1} + \dot{\psi}_k \Delta \mathrm{t}) + (1 - \alpha)(\psi_{\mathrm{a}})_{\mathrm{k}}$$

Where, $\alpha = \frac{T}{\Delta t}/(1 + \frac{T}{\Delta t})$

$$\psi_k = \alpha\psi_{k-1} + (1 - \alpha)(\psi_{\mathrm{a}})_{\mathrm{k}} + \alpha\dot{\psi}_k \Delta t$$

A3. Matlab Code Using Lagrangian Mechanics

```
m = 0.03328; % Mass of Wheel
M = 0.663; % Mass of Body including Wheel, Motor, Battery.
R = 0.0325; % Radius of Wheel
Jw = 1.757 * 10 ^ (-5); % Moment of Inertia of Wheel
Jm = 1.4453* 10 ^ (-5); % Moment of Inertia of Motor
H = 0.118; % Bot Height
L = H/2;
Jpsi = 2.304 * 10 ^ (-3);
r = 6.00; % Resistance
Kt = 0.12; % Torque Constant
Kb = 0.29; % EMF Constant
n= 21.3; % Gear Ratio
fm = 0.0022; % Friction coefficient between body and DC motor
b = ((n*Kb*Kt)/r)+fm;
g = 9.8;
a = (n*Kt/r)
W = 0.15; % Width
D = 0.09; % Depth
Jphi = 1.2168 * 10 ^ (-3);
T_ = 1/100;
E = [ ((2*m+M)*R*R+2*Jw+2*n*n*Jm)  (M*L*R-2*n*n*Jm); (M*L*R-2*n*n*Jm)
(M*L*L + Jpsi + 2*n*n*Jm) ];
F =  2* [b -b; -b b ]
G = [0 0; 0 -M*g*L ]
H = [a a; -a -a ]
I = 0.5*m*W*W + Jphi + (W*W)*0.5*(Jw+n*n*Jm)/(R*R)
J = ((W*W)*0.5*b)/(R*R)
K = W*a*0.5/R
Z = zeros([2,2]);
A1 =  [Z eye([2,2]); inv(E)*G -inv(E)*F];
B1 =  [0; 0; a*(M*L*L + Jpsi + M*L*R); -a*(2*m*R*R + M*R*L + M*R*R)]
A2 =  [0 1; (-K/I) (-J/I)]
B2 =  [0; (-K/I)]
Q1 = ([0.02 0 0 0; 0 100 0 0; 0 0 0.3 0; 0 0 0 1])
Q2 =([0.1 0;0 0.1]);
R1=10;
R2=10;
K1= ([1.0000  -100.3022  10.3845   -1.0666])
K2 = ([-0.002 -0.0010])
s = tf('s');
Rm = 1/(((r*Jm/Kt) * s^2)  + ((Kb  + (r * b/Kt))* s));
Lm = 1/(((r*Jm/Kt) * s^2)  + ((Kb  + (r * b/Kt))* s));
Lm_d = c2d(Lm,T_);
Rm_d = c2d(Rm,T_);
```

A4. Matlab Code Using Newtonian Mechanics

```
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 for the A and B matrices
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];
D = [0;
    0];
states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'u'};
outputs = {'x'; 'phi'};
sys_ss =
ss(A,B,C,D,'statename',states,'inputname',inputs,'outputname',outputs);
poles = eig(A);
co = ctrb(sys_ss);
controllability = rank(co);
Q = C'*C;
R = 1;
K = lqr(A,B,Q,R)
Ac = [(A-B*K)];
Bc = [B];
Cc = [C];
Dc = [D];
states = {'x' 'x_dot' 'phi' 'phi_dot'};
inputs = {'r'};
outputs = {'x'; 'phi'};
sys_cl =
ss(Ac,Bc,Cc,Dc,'statename',states,'inputname',inputs,'outputname',outputs);
t = 0:0.01:5;
r =0.2*ones(size(t));
[y,t,x]=lsim(sys_cl,r,t);
[AX,H1,H2] = plotyy(t,y(:,1),t,y(:,2),'plot');
set(get(AX(1),'Ylabel'),'String','cart position (m)')
set(get(AX(2),'Ylabel'),'String','pendulum angle (radians)')
title('Step Response with LQR Control')
```

A5. Arduino code for obtaining values from Encoder

```
#define encoder0PinA  2
#define encoder0PinB  4
#define encoder1PinA  3
#define encoder1PinB  5
volatile long encoder0Pos=0, encoder1Pos=0;
float newposition_0=0, newposition_1=0;
float oldposition_0 =0, oldposition_1 = 0;
unsigned long newtime, oldtime = 0;
float vel_0, vel_1, vel, drift;
float R = 1, W = 1;

void setup()
{
  pinMode(encoder0PinA, INPUT);
  digitalWrite(encoder0PinA, HIGH);       // turn on pullup resistor
  pinMode(encoder0PinB, INPUT);
  digitalWrite(encoder0PinB, HIGH);       // turn on pullup resistor
  attachInterrupt(2, doEncoder0, RISING);  // encoDER ON PIN 2
  pinMode(encoder1PinA, INPUT);
  digitalWrite(encoder1PinA, HIGH);       // turn on pullup resistor
  pinMode(encoder1PinB, INPUT);
  digitalWrite(encoder1PinB, HIGH);       // turn on pullup resistor
  attachInterrupt(3, doEncoder1, RISING);  // encoDER ON PIN 3
  Serial.begin (9600);
```

```
  Serial.println("start");                // a personal quirk
}
void loop()
{
newposition_0 = encoder0Pos;
newposition_1 = encoder1Pos;
newtime = millis();
vel_0 = 360* abs((newposition_0-oldposition_0)/235) * 1000 /(newtime-
oldtime);
vel_1 = 360* abs((newposition_1-oldposition_1)/235) * 1000 /(newtime-
oldtime);
vel = (vel_0 + vel_1)/2;
drift = (R*(vel_0 - vel_1))/W;
Serial.print ("Theta Dot = ");
Serial.println (vel);
Serial.print ("Theta = ");
Serial.println (vel * (newtime-oldtime) / 1000);
Serial.print ("Drift_Velocity = ");
Serial.println (drift);
Serial.print ("Drift = ");
Serial.println (drift * (newtime-oldtime) / 1000);
 oldposition_0 = newposition_0;
 oldposition_1 = newposition_1;
 oldtime = newtime;
 delay(1000);
 }

 void doEncoder0()
 {  { if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB))
     {  encoder0Pos++; }
     else { encoder0Pos--; }}
   { if (encoder0Pos > 235)
     { encoder0Pos = encoder0Pos - 235;}
     else if (encoder0Pos < -235)
      { encoder0Pos = encoder0Pos + 235;}  }  }

 void doEncoder1()
 {  if (digitalRead(encoder1PinA) == digitalRead(encoder1PinB))
          { encoder1Pos++; }
   else { encoder1Pos--; }
   { if (encoder1Pos > 235)
          { encoder1Pos = encoder1Pos - 235;}
     else if (encoder1Pos < -235)
          { encoder1Pos = encoder1Pos + 235;}  }  }
```

A6. Arduino code to obtain values from Gyroscope

```
#include "CurieIMU.h"
unsigned long microsPerReading, microsPrevious;
int gx, gy, gz, ax, ay, az, Psi_a;
float Total_angle, ga; //scaled Gyro values
void setup()
{
  microsPerReading = 1000000 / 25;
  microsPrevious = micros();
  Serial.begin(9600); // initialize Serial communication
  while (!Serial);    // wait for the serial port to open
  Serial.println("Initializing IMU device...");
  CurieIMU.begin();
  CurieIMU.setGyroRange(1000);
  CurieIMU.setAccelerometerRange(2);
```

```
  CurieIMU.setGyroRate(25);
  CurieIMU.setAccelerometerRate(25);
}
  if(e == 0)
  {
    for(int i=0; i<100; i++)
    {
     CurieIMU.readGyroScaled(gx, gy, gz);
     CurieIMU.readAccelerometer(ax, ay, az);
      Psia = atan2 (sqrt(ay*ay + ax*ax), az);
      offset = offset + Psia;
        if(i==99)
        {  offset = offset/100;
           e = 1;
           Totalangle = 0;
           Serial.print("OFFSET IS SET");
           Serial.println(offset); }
           delay(25);   }  }   }
void loop()
 {
    unsigned long microsNow;
    microsNow = micros();
   if (microsNow - microsPrevious >= microsPerReading)
    {
     CurieIMU.readMotionSensor(ax, ay, az, gx, gy, gz);
     Psi_a = (180/3.14) * atan2 (sqrt(ay*ay + ax*ax), az);
     ga = (1000*gx)/(32768*25);
     Total_angle = 0.70 *(Total_angle + ga) + 0.30 * Psi_a;
            { if(Psia > 0)
          { Psi = (offset - Totalangle); }
        else
          { Psi = -(Totalangle + offset); } }
    Serial.print(" Angle : ");
    Serial.println(Total_angle);
    Serial.print(" Angular Velocity : ");
    Serial.println(ga*25);
    microsPrevious = microsPrevious + microsPerReading;
    }
 }
```

A7. Arduino Controller Code

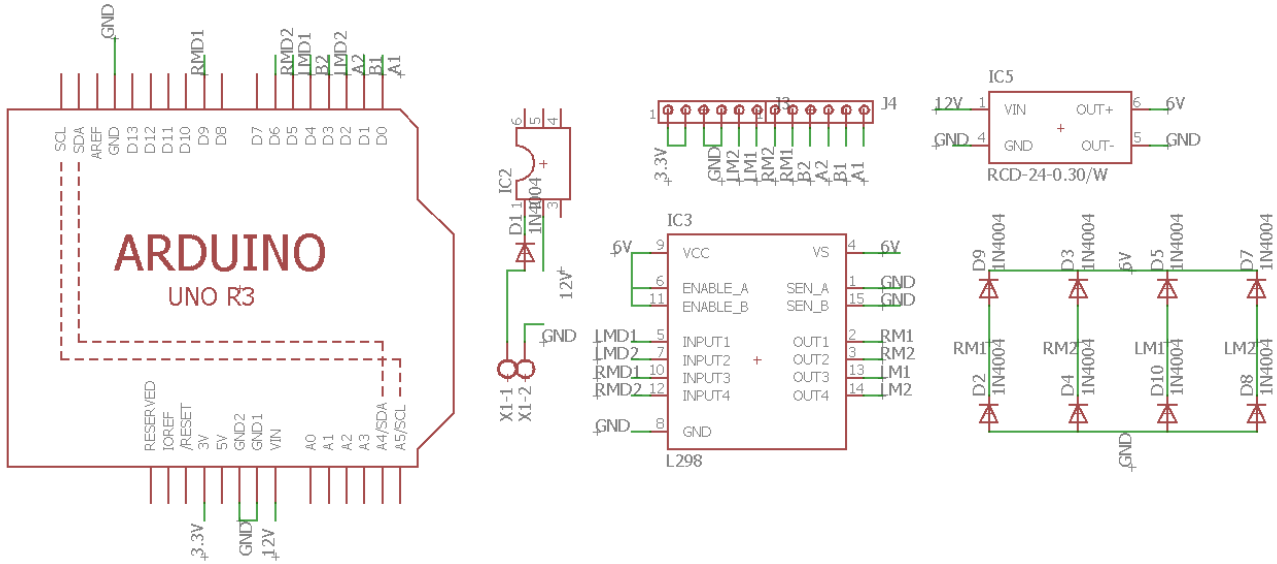
```
 #include "CurieIMU.h"
 #define encoder0PinA  0
 #define encoder0PinB  1
 #define encoder1PinA  2
 #define encoder1PinB  4
int ax, ay, az;
float vel = 0, vela = 0, drift = 0, drifta = 0, Vl = 0, Vr = 0, Vlp = 0,
Vrp=0, u1=0, u2=0, offset = 0, Psia = 0, gx, gy, gz, e = 0;
float encoder0Pos = 0, encoder1Pos = 0, newposition0 = 0, newposition1 = 0,
oldposition0 = 0, oldposition1 = 0, Totalangle = 0, v0 = 0, v1 = 0, Psi = 0;
unsigned long microsPerReading, microsPrevious = 0, microsNow;

// R = 0.0325, W = 0.202

void setup()
 {
   pinMode(encoder0PinA, INPUT);
   pinMode(encoder0PinB, INPUT);
   attachInterrupt(0, doEncoder0, RISING);
```

```
   pinMode(encoder1PinA, INPUT);
   pinMode(encoder1PinB, INPUT);
   attachInterrupt(2, doEncoder1, RISING);
   pinMode(3, OUTPUT);
   pinMode(5, OUTPUT);
   pinMode(6, OUTPUT);
   pinMode(9, OUTPUT);
// Motor Max Speed is 4 RPS so Sampling Frequency is > 4. Taken as 25
   microsPerReading = 1000000 / 200;
   microsPrevious = micros();
   CurieIMU.begin();
   CurieIMU.setGyroRange(250);
   CurieIMU.setGyroRate(200);
   CurieIMU.setAccelerometerRate(200);
   Serial.print(CurieIMU.getGyroOffset(X_AXIS));
   Serial.print("\t");
   Serial.print(CurieIMU.getGyroOffset(Y_AXIS));
   Serial.print("\t");
   Serial.println(CurieIMU.getGyroOffset(Z_AXIS));
   Serial.println("About to calibrate. Make sure your board is stable and
upright");
   delay(5000);
   Serial.println("Starting Gyroscope calibration and enabling offset
compensation...");
   CurieIMU.autoCalibrateGyroOffset();
   Serial.println("Starting Acceleration calibration and enabling offset
compensation...");
   CurieIMU.autoCalibrateAccelerometerOffset(X_AXIS, 0);
   CurieIMU.autoCalibrateAccelerometerOffset(Y_AXIS, 0);
   CurieIMU.autoCalibrateAccelerometerOffset(Z_AXIS, 1);
   Serial.println("Internal sensor offsets AFTER calibration...");
   Serial.print(CurieIMU.getAccelerometerOffset(X_AXIS));
   Serial.print("\t"); //
   Serial.print(CurieIMU.getAccelerometerOffset(Y_AXIS));
   Serial.print("\t"); //
   Serial.print(CurieIMU.getAccelerometerOffset(Z_AXIS));
   Serial.print("\t"); //
   Serial.println(CurieIMU.getGyroOffset(X_AXIS));
   Serial.println("CHANGE position");
   delay(5000);
   if(e == 0)
   {
    for(int i=0; i<100; i++)
    {
     CurieIMU.readGyroScaled(gx, gy, gz);
     CurieIMU.readAccelerometer(ax, ay, az);
      Psia = atan2 (sqrt(ay*ay + ax*ax), az);
      offset = offset + Psia;
        if(i==99)
        {  offset = offset/100;
           e = 1;
           Totalangle = 0;
           Serial.print("OFFSET IS SET");
           Serial.println(offset); }
           delay(25);   }  }   }
 void loop()
 {  microsNow = micros();
    if (microsNow - microsPrevious >= microsPerReading)
     {
      CurieIMU.readGyroScaled(gx, gy, gz);
      CurieIMU.readAccelerometer(ax, ay, az);
```

```
  gx = gx*(3.14/180);
  Psia = (ay/abs(ay))*(atan2 (sqrt(ay*ay + ax*ax), az));
  Totalangle = 0.70 *(Totalangle - 0.005*int(gx)) + 0.30 * Psia;
   { if(Psia > 0)
      { Psi = (offset - Totalangle); }
     else
      { Psi = -(Totalangle + offset); } }
 newposition0 = encoder0Pos;
 newposition1 = encoder1Pos;
// 200*2*pi/235 = 5.347
v0 = ((5.347)*(newposition0-oldposition0));
v1 = ((5.347)*(newposition1-oldposition1));
vel = (v0 + v1)*(0.5);
drift = (0.16089)*(-v0 + v1);
vela = vela + (vel*(0.04));
    { if (vela > 6.28)
       { vela = vela - 6.28;}
      else if (vela < -6.28)
       { vela = vela + 6.28;}  }
drifta = drifta + (drift*(0.04));
    { if (drifta > 6.28)
       { drifta = drifta - 6.28;}
      else if (drifta < -6.28)
       { drifta = drifta + 6.28;}  }
Serial.print("Angle : ");
Serial.println(Psi);
Serial.print("Angular Velocity : ");
Serial.println(-gx);
Serial.print("Drift_Velocity = ");
Serial.println(drift);
Serial.print("Drift = ");
Serial.println(drifta);
Serial.print("Theta Dot = ");
Serial.println(vel);
Serial.print("Theta = ");
Serial.println(vela);
u1 = -1*( (vela)* 0.1594  + (Psi)*(-0.9974) + (vel)*(9.9941) + (-gx)*(-
38.7041) );
u2 = -1*( (drifta) * (-0.0494) + (drift) * (-6.7489) );
Vlp = (constrain((abs(Vl)) + 50, 0, 255));
Vrp = (constrain((abs(Vr)) + 90, 0, 255));
Vl = 0.5*(u1 + u2);
Vr = 0.5*(u1 - u2);
Serial.print ("Voltage Left= ");
Serial.println (Vl);
Serial.print ("Voltage Right= ");
Serial.println (Vr);
Serial.print ("Voltage Left Pulse= ");
Serial.println (Vlp);
Serial.print ("Voltage Right Pulse= ");
Serial.println (Vrp);
Serial.println ();
{ if(Vl>0)
   {      analogWrite(3, Vlp);
          analogWrite(5, 0);     }
  else
   {      analogWrite(3, 0);
          analogWrite(5, Vlp); }  }

{ if(Vr>0)
   {      analogWrite(6, Vrp);
```

```
              analogWrite(9, 0);      }
      else
       {      analogWrite(6, 0);
              analogWrite(9, Vrp); }   }

    oldposition0 = newposition0;
    oldposition1 = newposition1;
    microsPrevious = microsPrevious + microsPerReading;   }    }

void doEncoder0()
{   { if (digitalRead(encoder0PinA) == digitalRead(encoder0PinB))
    {  encoder0Pos++; }
    else { encoder0Pos--; }}
  { if (encoder0Pos > 235)
    { encoder0Pos = encoder0Pos - 235;}
    else if (encoder0Pos < -235)
     { encoder0Pos = encoder0Pos + 235;}   }   }

void doEncoder1()
{   if (digitalRead(encoder1PinA) == digitalRead(encoder1PinB))
          { encoder1Pos++; }
  else { encoder1Pos--; }

  { if (encoder1Pos > 235)
          { encoder1Pos = encoder1Pos - 235;}
    else if (encoder1Pos < -235)
          { encoder1Pos = encoder1Pos + 235;}   }   }
```

A8. PCB schematic circuit



REFERENCE

[1] Amir A. Bature, Salinda Buyamin, Mohamed. N.Ahmad, Mustaphpa Muhammad (2014) A Comparison of Controllers for Balancing Two Wheeled Inverted Pendulum Robot. International Journal of Mechanical & Mechatronics Engineering IJMME-IJENS Vol:14 No:03

[2] Dorf, Richard & Robert H. Bishop. 2001 'Modern Control Systems', Prentice-Hall, United States of America.

[3] Grasser, Felix and Alonso D'Arrigo, 2002 'JOE: A Mobile, Inverted Pendulum', IEEE Transactions on Industrial Electronics, Vol 49.

[4] Kuo-Chun Wu (2007) Analysis and comparison of PID and state feedback controller in inverted pendulum system. 2007 Master's thesis, National Kaohsiung University Department of Electrical Engineering, Kaohsiung, Taiwan

[5] Ogata, K. (2009). 'Modern Control theory, University of Minnesota, United States of America: Person Publication. Inc.'

[6] Ooi, R. (2013). 'Balancing a Two-Wheeled Autonomous Robot. Undergraduate. The University of Western Australia.'

[7] Osama Jamil, Mohsin Jamil, Yasar Ayaz, Khubab Ahmad (2014) Modeling, Control of a Two Wheeled Self-Balancing Robot. 2014 International Conference on Robotics and Emerging Allied

Technologies in Engineering (iCREATE)Islamabad, Pakistan, April 22-24, 2014

[8] Sundin, C. and Thorstensson, F. (2013), ‘Autonomous balancing robot, Masters of Science, Chalmers University of Technology.’ Mikael Arvidsson. and Jonas Karlsson. (2012). ‘Design, construction and verification of a self-balancing vehicle. Chalmers University of Technology.’