Research Internship in System Engineering

DESIGN AND CONTROL OF TWO-WHEELED SELF BALANCING ROBOT

Harish Swaminathan Gopal
swamin01@ads.uni-passau.de

Rutvik Bhatt
bhatt02@ads.uni-passau.de

Ram Prasanth Udhaya Baskar
udhaya01@ads.uni-passau.de

Shrinivas Iyengar
iyenga01@ads.uni-passau.de

Prof. Dr. Fabian Wirth
Ms. Roxanne Jackson
Chair of Mathematics with focus on Dynamic Systems

Contents

ABSTRACT

This report entails the modelling, design and simulation of a two-wheeled self-balancing robot followed by the hardware implementation on an embedded system. Linear Quadratic Regulator (LQR) is implemented to control the linearized dynamic system. Kalman filter has been implemented for state estimation.The system architecture comprises of a pair of DC stepper motors, a Raspberry Pi development board and an Inertial Measurement Unit for attitude determination.

1 Introduction

A self-balancing robot, based on the principle of inverted pendulum, is a two-wheeled robot that balances itself up in the vertical position with reference to the ground. The self-balancing robot is an unstable system. These problems serve as a platform to witness and verify control concepts such as system stability and compare various controlling techniques. As a result, the two wheeled self-balancing robot has become a widely studied system used to observe and verify various control design strategies.

The most appropriate example of this principle in commercial use is the Segway (originaly, Human transporters), which uses stability results from the study of two-wheeled self-balancing robots. Besides the Segway transporter, an in-depth study on two-wheeled self balancing robots and the use of various control algorithms have been widely published. Self-balancing robot using Arduino micro-controller [1] uses PID and LQR based PI-PD controller; JOE [2] is an example of the use of two decoupled state-space controllers and a Digital Signal Processor (DSP); the use of Linear Quadratic Regulator (LQR) and Proportional-Integral-Derivative (PID) to control the non linear dynamical system considering the disturbances is shown in [3]. Another interesting early version of a self-balancing robot with the configuration of inertial sensors, actuators, on-board controllers and the ability to navigate and drive is the nBot [4]. Other related works include fuzzy-neural control [5], nonlinear backstepping control [6], the combination of fuzzy-backstepping controllers [7] and Model Predictive Control (MPC) [8].

To control the dynamics of a robot, we need a way to describe how the robot behaves. Mathematical modelling of a system allows us to describe the dynamic behaviour of the system in a tractable mathematical formulation. We use the Euler-Lagrange equations to derive the equations that govern the motion of our robot.

The physical structure of the robot contains two stepper motors powered by two 11.1 V LiPo batteries, a Raspberry Pi 3B+ along with the MPU 6050, an inertial measurement unit (IMU), powered by a 8.4V NiMH battery, all enclosed in a 3D-printed chassis. The Kalman Filter has been implemented for state estimation. Based on the linearized equations of motion, LQR is adopted for this system.

This report is organised as follows: Section 2 outlines the derivation of the mathematical model for the robot and introduces the model parameters, Section 3 gives an insight into understanding the control design and implementation, Section 4 details the Observer and Kalman Filter implementation, Section 5 presents the physical structure and the components used in the robot. Section 6 describes the practical problems occurred during hardware implementation and potential solutions that were tried to mitigate them. Finally, Section 7 summarizes the project.

2 Mathematical Modelling

We derive the dynamic equations by using the Euler-Lagrange energy methods. Lagrangian mechanics makes modelling easier compared to Newtonian mechanics as it facilitates the usage of generalized coordinates.

The model parameters of the free standing robot are shown in Figure 1 and described in Table 1.

The parameters and variables of wheeled inverted pendulum robot		
Variable	Unit	Description
θ	[radians]	Angle of chassis from the vertical position
ϕ	[radians]	Angle between chassis and wheel initial position
α	[radians]	Angle between inclined and horizontal
M	[kg]	Mass of robot chassis
m	[kg]	Combined mass of both wheels and motors
J	[kg.m^2]	Mass moment of inertia for m about the axle
I	[kg.m^2]	Mass moment of inertia for M about the centre of mass
τ	[N.m]	Driving torque
g	[m.s^{-2}]	Acceleration due to gravity (defined to be negative)
r	[m]	Radius of wheels
l	[m]	Distance to center of gravity from the axle

Table 1: Parameters used to describe kinetics and kinematics of a free-standing robot on an inclined plane

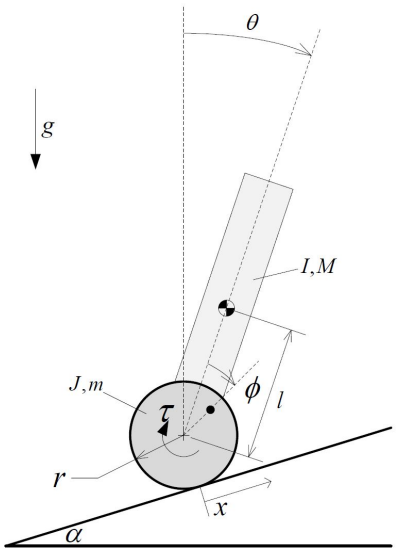

Figure 1: Model of a wheeled inverted pendulum robot

2.1 Euler-Lagrange Model

We can describe all physical, dynamic systems using simple energy-based approaches from analytical mechanics. The Lagrangian $\mathcal{L}$ is defined by the kinetic co-energy, $\mathcal{T}^*$, and the potential energy, $\mathcal{V}$, to be $\mathcal{L}(q, \dot{q}) = \mathcal{T}^*(q, \dot{q}) - \mathcal{V}(q)$. Derived from Newton's second law, the generalised Euler-Lagrange equation is given as:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i - \frac{\partial \mathcal{D}}{\partial \dot{q}_i}, \tag{1}$$

where $\mathcal{D}$ is the dissipation function, q_i is the i-th generalised coordinate, and τ_i is the generalized force. First, we find the Lagrangian, $\mathcal{L}$, by calculating the energy of the system, $\mathcal{T}^*$ and $\mathcal{V}$, and then evaluate Eq. 1 by substituting ϕ and θ, the generalized co-ordinates of our system, for q_i.

The Kinetic energy of the system is given by,

$$\mathcal{T}^* = \frac{1}{2}mv^2 + \frac{1}{2}MV^2, \tag{2}$$

where v^2 and V^2 are made up of the linear and inertial components of the velocity of the wheels and the chassis respectively. The velocities can be further expanded as.

$$v^2 = \dot{x}^2 + \frac{r^2}{2}(\dot{\theta} + \dot{\phi})^2,$$

where $x = r(\theta + \phi)$, using the arc length formula, and

$$V^2 = V_x^2 + V_y^2 + \frac{1}{12}l^2\dot{\theta}^2,$$

where, $V_x = \dot{x} + l\dot{\theta}\cos(\theta + \alpha)$, and, $V_y = l\dot{\theta}\sin(\theta + \alpha)$.

The potential energy of the system is given as follows:

$$\mathcal{V} = mgh_1 + Mgh_2 + (m + M)gh_0, \tag{3}$$

where, $h_1 = r(\theta + \phi)\sin\alpha$, is the vertical height of the wheel axle, $h_2 = h_1 + l\cos\theta$, is the vertical height of the chassis' center of mass and h_0 is some initial vertical height of the robot on the on the inclined plane. Substituting the

individual components into Eq. 2 and Eq. 3, we derive the Lagrangian [12] as:

$$\begin{aligned}\mathcal{L} = &\frac{1}{2}(J + (m+M)r^2)(\dot{\theta}+\dot{\phi})^2 + Mlr(\dot{\theta}+\dot{\phi})\dot{\theta}\cos(\theta+\alpha) + \frac{1}{2}(I+Ml^2)\dot{\theta}^2 \\ &- mg(r(\theta+\phi)\sin\alpha) - Mg(r(\theta+\phi)\sin\alpha - l\cos\theta) - (m+M)gh_0.\end{aligned} \tag{4}$$

We substitute the Lagrangian and the two co-ordinates of our system, ϕ and θ, one at a time, into Eq. 1 to derive two equations. We get the following equation of motion with respect to the ϕ co-ordinate:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\phi}}\right) - \frac{\partial \mathcal{L}}{\partial \phi} = \tau_{ext} + \tau_D,$$

and with respect to the θ co-ordinate:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}}\right) - \frac{\partial \mathcal{L}}{\partial \theta} = \tau_{ext} + \tau_D,$$

where τ_{ext} is the external torque applied to the system, and τ_D is the torque of the disturbances, which can be neglected in this case. After solving for the above two equations, we arrive at the following two equations of motion:

$$(J + (m+M)r^2)(\ddot{\phi}+\ddot{\theta}) + Mlr\ddot{\theta}\cos(\theta+\alpha) - Mlr\dot{\theta}^2\sin(\theta+\alpha) + (M+m)gr\sin\alpha = \tau \tag{5}$$

$$\begin{aligned}&(J + (m+M)r^2)(\ddot{\phi}+\ddot{\theta}) + Mlr\ddot{\theta}\cos(\theta+\alpha) - Mlr\dot{\theta}^2\sin(\theta+\alpha) \\ &+ Mlr(\ddot{\phi}+\ddot{\theta})\cos(\theta+\alpha) + (I+Ml^2)\ddot{\theta} + (M+m)gr\sin\alpha - Mgl\sin\theta = 0\end{aligned} \tag{6}$$

It should be noted that the external torque applied to the θ co-ordinate is zero as we are not applying any force directly to the chassis. The only force that we apply is through the wheels (whose angle of rotation is represented by ϕ) which in-turn corrects the chassis' tilt-angle. Thus, Eq. 5 contains the τ term and Eq. 6 doesn't. As this two co-ordinate system has only one input[1], this is an underactuated system.
These second-order equations are the dynamic equations of our self-balancing robot which describe the inter-dependencies between our variables of interest, $\theta, \dot{\theta}, \phi$, and, $\dot{\phi}$.

2.2 Notation

To simplify the equations and make them easier to read, a few repeating constant terms have been substituted with the following :

$$\begin{aligned}a_1 &= J + (M+m)r^2, \\ a_2 &= I + Ml^2, \\ a_3 &= (M+m)gr\sin\alpha, \\ a_4 &= Mlr\cos\alpha.\end{aligned}$$

2.3 Non-linear model

From Eq. 5 and 6, we obtain the following non-linear model, while also accounting for damping on the system:

$$J\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\end{bmatrix} + B\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\end{bmatrix} + D\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\end{bmatrix} + C = \begin{bmatrix}\tau\\ 0\end{bmatrix}, \tag{7}$$

where J, B, D and C matrices are defined as follows:

$$J = \begin{bmatrix} a_1 & a_1 + Mlr\cos(\theta+\alpha) \\ a_1 + Mlr\cos(\theta+\alpha) & a_1 + 2Mlr\cos(\theta+\alpha) + a_2 \end{bmatrix},$$

$$B = -Mlr\dot{\theta}\sin(\theta+\alpha)\begin{bmatrix}0 & 1\\ 0 & 1\end{bmatrix},$$

[1]Though the robot contains two motors, the torque is the only force which is distributed to both the motors.

$$D = \begin{bmatrix} b/r & b/r \\ 0 & c \end{bmatrix},$$
$$C = \begin{bmatrix} a_3 \\ a_3 + mgl\sin(\theta) \end{bmatrix}.$$

From Eq. 7, the final equations for $\ddot{\phi}$ and $\ddot{\theta}$ can be written as:

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \end{bmatrix} = J^{-1}\left[-B\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \end{bmatrix} - D\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \end{bmatrix} - C + \begin{bmatrix} \tau \\ 0 \end{bmatrix}\right]. \tag{8}$$

Eq. 8 represents the non-linear equations of motion for our robot. The nonlinear equations could be used to design a nonlinear controller. But, using a linear controller is less complex to implement and is sufficient to balance the robot. In order to apply a linear controller to the system, we need to linearise these equations, which will be explained in the next section.

2.4 Linearisation and State Space

As Eq. 8 is a second order differential equation, we convert these equations into a first order system and then linearize around a fixed point. For converting into a first order system, the following convention was adopted. Consider the state space variables:

$$x_1 = \theta; \quad x_2 = \dot{\theta} = \dot{x}_1;$$
$$x_3 = \phi; \quad x_4 = \dot{\phi} = \dot{x}_3.$$

So, the state of the system is given by $x = [\theta\ \dot{\theta}\ \phi\ \dot{\phi}]^T$

The state space equation for the system may be written in the form of:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix},$$

where, $f_1 = x_2, f_3 = x_4$,

$$f_2 = \frac{1}{D_J}\left[(Mlrx_2^2\sin(x_1+\alpha))(-Mlr\cos(x_1+\alpha)) + a_3(Mlr\cos(x_1+\alpha)) - a_1Mgl\sin x_1 \right.$$
$$\left. - \tau(a_1 + Mlr\cos(x_1+\alpha))\right],$$

$$f_4 = \frac{1}{D_J}[(Mlr)^2x_2^2\sin(x_1+\alpha)\cos(x_1+\alpha) + a_2Mlrx_2^2\sin(x_1+\alpha) - a_3Mlr\cos(x_1+\alpha) - a_3a_2$$
$$+ a_1Mgl\sin x_1 + (Mlr)(Mgl)\sin x_1\cos(x_1+\alpha) + \tau(a_1 + 2Mlr\cos(x_1+\alpha) + a_2)],$$

and

$$D_J = a_1a_2 - (Mlr\cos(x_1+\alpha))^2.$$

To linearise the equations of motion, we use the Jacobian Matrix, which is a matrix of partial derivatives of the functions with respect to each of the variables and the control input. The Jacobian is defined as:

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{(x^*,u^*)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_4} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_4}{\partial x_1} & \cdots & \frac{\partial f_4}{\partial x_4} \end{bmatrix}_{(x^*,u^*)},$$

and,

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{u}}\right)_{(x^*,u^*)} = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \vdots \\ \frac{\partial f_4}{\partial u} \end{bmatrix}_{(x^*,u^*)}.$$

To obtain the linearised state-space equations, we evaluate the above Jacobian at some point of interest, (x^*, u^*), known as the fixed point. In the next section, we will take a look at our desired fixed point and evaluate the Jacobian matrix at the fixed point.

2.5 Linear model

Since we want this system to stand in an upright position, we want to linearise the system around its equilibrium point. This system has an unstable equilibrium at $\theta = 0$. So, as a starting point we select our fixed points as $x^* = [0 \ 0 \ 0 \ 0]'$ and $u^* = 0$. We evaluate the Jacobian at the fixed points which gives us the matrices A and B, respectively, as

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{24} \\ 0 & 0 & 0 & 1 \\ a_{41} & a_{42} & 0 & a_{44} \end{bmatrix},$$

where,

$$a_{21} = \frac{\left(a_1 a_2 - M^2 l^2 r^2 \cos^2\alpha\right)\left(-a_3 Mlr \sin\alpha - a_1 Mgl\right) - \left(a_3 Mlr\cos\alpha\right)\left(2M^2 l^2 r^2 \cos\alpha \sin\alpha\right)}{\left(a_1 a_2 - (Mlr\cos\alpha)^2\right)^2},$$

$$a_{22} = \frac{-ba_1 - bMlr\cos\alpha + cra_1}{r\left(a_1 a_2 - (Mlr\cos\alpha)^2\right)^2},$$

$$a_{24} = \frac{-a_1 b - bMlr\cos\alpha}{r\left(a_1 a_2 - (Mlr\cos\alpha)^2\right)^2},$$

$$a_{41} = \frac{\left(a_1 a_2 - M^2 l^2 r^2 \cos^2\alpha\right)\left(a_3 Mlr \sin\alpha + a_1 Mgl + (Mlr)(Mgl)\cos\alpha\right) + \left(a_2 a_3 + a_3 Mlr\cos\alpha\right)\left(2M^2 l^2 r^2 \cos\alpha \sin\alpha\right)}{\left(a_1 a_2 - (Mlr\cos\alpha)^2\right)^2}$$

$$a_{42} = \frac{(2b - cr)Mlr\cos\alpha + b(a_1 + a_2) - cra_1}{r\left(a_1 a_2 - (Mlr\cos\alpha)^2\right)^2},$$

$$a_{44} = \frac{b(a_1 + a_2) + 2bMlr\cos\alpha}{r\left(a_1 a_2 - (Mlr\cos\alpha)^2\right)^2},$$

and

$$B = \begin{bmatrix} 0 \\ b_2 \\ 0 \\ b4 \end{bmatrix}$$

where,

$$b_2 = \frac{-a_1 - Mlr\cos\alpha}{a_1 a_2 - (Mlr\cos\alpha)^2}, \qquad b_4 = \frac{a_1 + a_2 + 2Mlr\cos\alpha}{a_1 a_2 - (Mlr\cos\alpha)^2}.$$

The matrix A is called the state-transition matrix, and the matrix B is called the control matrix. Together, they make up the process model, which defines the evolution of the state along with the effect of the control input, given by:

$$\dot{x} = Ax + Bu. \tag{9}$$

The measurement model, which describes the variables that we measure/observe, is given by:

$$\dot{y} = Cx. \tag{10}$$

The matrix C is called the measurement matrix and is a map from the state-space to the measurement space. Since we are observing all the states the measurement matrix is given by the identity matrix:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

The linearised system can be described by the equations 9 and 10. To apply the linear controller we check if the system is controllable, by the Kalman Rank condition:

$$rank(B|AB|A^2B|A^3B) = n.$$

By substituting the matrices A and B, it can be easily verified that the above property is true. Thus, the system is completely controllable.

3 Control Design and Implementation

This system has a scalar control input, u, which is the torque, τ, applied to the motors. We need to compute and apply the input at regular time steps to correct the orientation of the robot. We use the Linear Quadratic Regulator (LQR) to compute the control input. Controlling in a state-feedback formulation, for a system given by:

$$\dot{x} = Ax + Bu,$$

the control input can be calculated such that

$$u = -Kx.$$

This results in the closed-loop system:

$$\begin{aligned} \dot{x} &= Ax - BKx \\ &= (A - BK)x \end{aligned}$$

To balance the robot, we need to place the closed loop poles of the system at desirable positions. For a continuous-time linear system, it is desirable that the real part of the closed-loop poles are negative. The closed-loop poles of a system correspond to the eigenvalues of the system after the feedback has been applied, that is:

$$det[sI - (A + BK)] = 0.$$

3.1 Linear Quadratic Regulator(LQR)

The linear quadratic regulator (LQR) finds an optimal feedback to stabilize the system with minimal cost in a given time period. Dynamics of the system are defined by the linearized equations and cost is described by quadratic function. For a linear system as defined in Eq. 9, with a cost function defined as:

$$J_c = \frac{1}{2}\int_0^T (x^T Q x + u^T R u + 2x^T N u)dt, \tag{11}$$

the control law that minimizes the cost function is

$$u = -Kx,$$

where, the feedback gain matrix, K, is given by

$$K = R^{-1}(B^T P + N^T),$$

and P is calculated by solving the Algebraic Ricacati equation

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

Q and R are both positive semi-definite matrices and N is an optional cross term matrix. Here, N has been set to zero. As this system has only one control input, R is a scalar value. The states' deviation from zero is penalized by the matrix Q. The entries of Q are multiplied by each of the state variables in Eq. 11:

$$\begin{bmatrix} \theta & \dot{\theta} & \phi & \dot{\phi} \end{bmatrix} \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 \\ 0 & 0 & q_3 & 0 \\ 0 & 0 & 0 & q_4 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix}.$$

This means that each q_i affects one of the state variables. The higher the value of q_i, the faster the corresponding state converges to zero. However, we also need to take care of variables' dependence on each other. For an instance, setting q_2 as a relatively large number results in $\dot{\theta}$ converging relatively quickly. As $\dot{\theta}$ is the rate of change of θ, this would result in θ not converging to zero quickly. Similarly, setting a higher value to R penalizes the value of u. The feedback gain matrix, when multiplied by the state determines the torque, τ, required to stabilize the system.

3.2 Controller Design

The equations described in Section 3.1 are the theoretical foundations of the working of LQR. However, during implementation, the feedback matrix K is computed using the `lqr` function in MATLAB. This function takes in, as arguments, the matrices A, B, Q and R. A desirable value of K would result the value of θ converging to zero. To

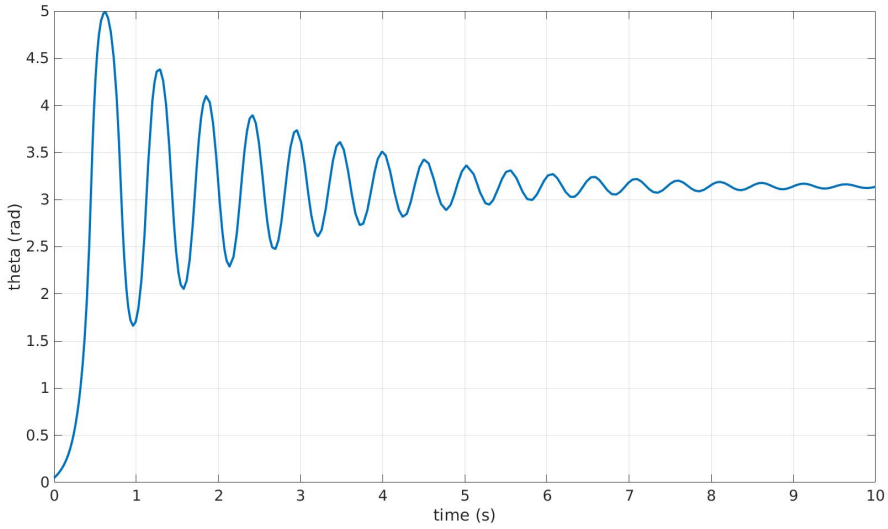

Figure 2: Behaviour of θ without adding a feedback

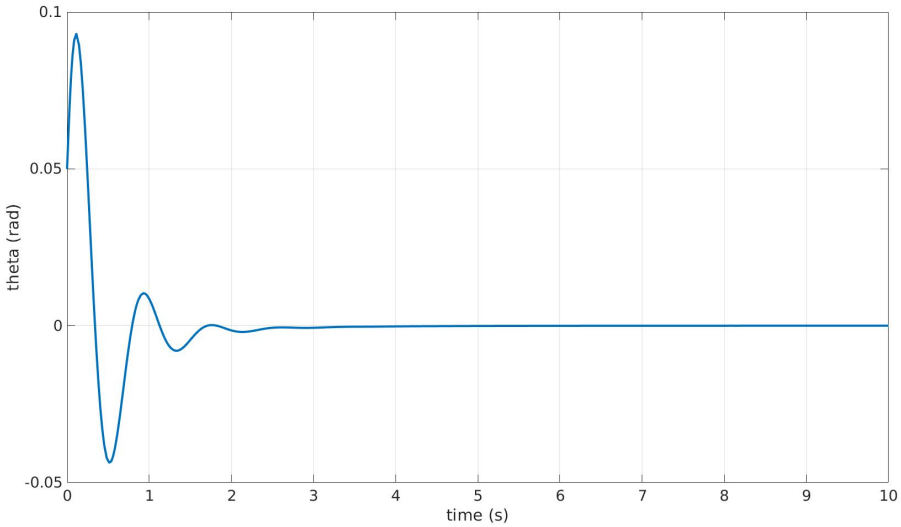

Figure 3: theta plot

examine the effect that K has on our model, we first simulate the non-linear model, Eq. 8, using Simulink. The behaviour of the system, without adding a feedback input is shown in Figure 2.

After multiple iterations of trying with different values for Q and R, we finally arrived at the following values:

$$Q = \begin{bmatrix} 10 & 0 & 0 & 0 \\ 0 & 50 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix}, \qquad R = 100.$$

This results in the feedback matrix $K = [0.3162 \quad 1.1003 \quad 0.0617 \quad 0.3112]$, and the corresponding behaviour of θ is shown in Figure 3.

3.3 Control Allocation

The control allocation problem is that of distributing a desired total control effort among a redundant set of actuators [10]. The feedback calculated by LQR, when multiplied with the state, gives us the torque, τ. We apply the same torque to both the motors. However, the input to the motors is different from the torque calculated by the controller. The motor can move precisely by accurately synchronizing with the pulse signal from the micro-controller. The speed of the motor can be controlled by modifying the time period, T, of the square wave pulses. To apply a specific torque, we need to determine the equivalent angular velocity to operate the motors, and then convert it in the form of pulse width modulated (PWM) signal. The angular velocity of the wheels have been modelled by the state variable $\dot{\phi}$. In Section 4.2 we explain in more detail to get the formulation:

$$\dot{\phi}(t+h) = \dot{\phi}(t) + \left[hA \begin{bmatrix} \theta & \dot{\theta} & \phi & \dot{\phi} \end{bmatrix}^T\right]_4 + \left[hB\tau\right]_4 .$$

To rotate the motor by one step (i.e. 1.8°) a single pulse of 50% duty cycle is required. An angular velocity of 2π rad/sec or 1 rev/sec implies 200 steps in one second. We use the motor at 16 micro-step configuration, which implies 3200 steps/sec. This requires the time period of the PWM signal to be:

$$T = \frac{1}{3200}\ s.$$

Following the same logic, the equivalent time period for any required angular velocity, $\dot{\phi}$ can be calculated by this relation:

$$T = \frac{\dot{\phi}}{2\pi} \times 16 \times 200\ s.$$

We divide by 2π to convert the unit from rad/s to rev/s.

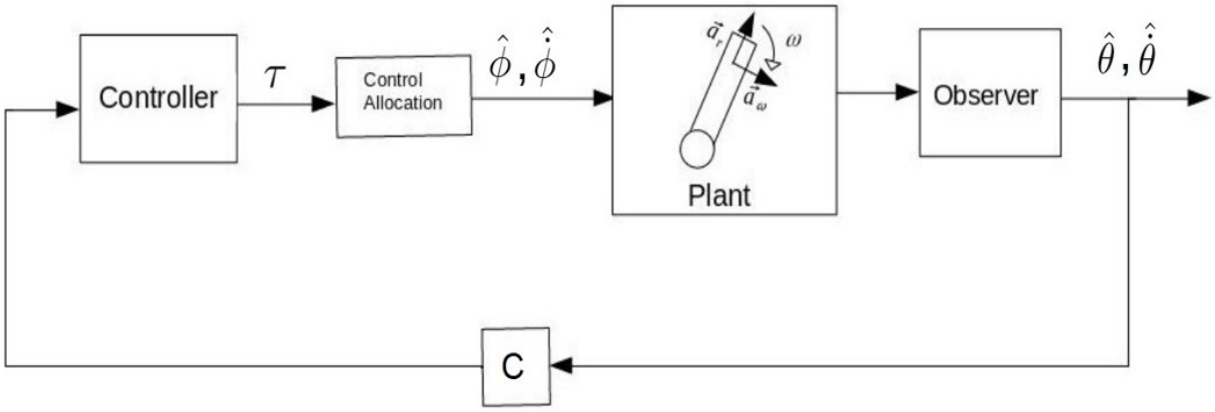

Figure 4: Block Diagram

Figure 4 summarizes the overall flow of the entire system. The controller calculates a torque to be applied in order to stabilize the system, which is then converted to an angular velocity to operate the motors which changes the plant dynamics. Then, we need to measure or estimate the variables of interest to track the changes.

4 Observer and Kalman Filter

To stabilize any system, it is necessary to measure the internal state of the robot. In Section 2.4 we defined the state of our robot to be:

$$x = \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} .$$

The only sensor on-board the robot is the MPU-6050 inertial measurement unit, which provides the angular velocity measurements from the gyroscope and the acceleration on the body measured by the accelerometer. As a result of this

limitation, we need a way to *observe* or *estimate* all the internal states of the robot. Additionally, gyroscopes are prone to bias instabilities which cause a small offset in the readings to drift over time. So, the constant bias error must be estimated and subtracted at every measurement. To estimate θ and $\dot{\theta}$, we use the Kalman Filter, and we estimate the other two variables from the mathematical model.

4.1 Kalman Filter

The Kalman Filter (KF) is a recursive algorithm and addresses the general problem of estimating the state, when we can not directly measure the states. For the Kalman Filter to be the optimal estimator, it is required that the state-transition and the observation models are linear and have zero mean Gaussian noise.

Applying the KF on measurements from the IMU, we want to estimate θ, $\dot{\theta} =: \omega$ and the bias, b. We use a discrete-time Kalman Filter here as the the data output rate from the IMU is at 1 KHz. The discrete time process model is given by:

$$\begin{bmatrix} \dot{\omega}_k \\ \dot{\theta}_k \\ \dot{b}_k \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ T & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\omega}_{k-1} \\ \dot{\theta}_{k-1} \\ \dot{b}_{k-1} \end{bmatrix} + \begin{bmatrix} \dot{w}_{\omega k} \\ 0 \\ \dot{w}_{bk} \end{bmatrix}, \tag{12}$$

with the associated co-variance matrix:

$$Q = \begin{bmatrix} Tq_\omega & \frac{1}{2}T^2 q_\omega & 0 \\ \frac{1}{2}T^2 q_\omega & \frac{1}{3}T^3 q_\omega & 0 \\ 0 & 0 & Tq_b \end{bmatrix},$$

where $T = 10^{-3}s$ is the sampling rate of the IMU. The parameters for the matrix Q and P have been described in Table 2.

It should be noted that we are not estimating all four state variables with the KF. When talking about the state in the context of the KF, we are referring to estimating the three variables θ, $\dot{\theta}$ and b.
We denote the estimate of the state by μ_k, where:

$$\mu_k = \begin{bmatrix} \mu_\omega \\ \mu_\theta \\ \mu_b \end{bmatrix}_k,$$

with the associated co-variance matrix:

$$P = \begin{bmatrix} Tp_\omega & \frac{1}{2}T^2 p_\omega & 0 \\ \frac{1}{2}T^2 p_\omega & \frac{1}{3}T^3 p_\omega & 0 \\ 0 & 0 & Tp_b \end{bmatrix}.$$

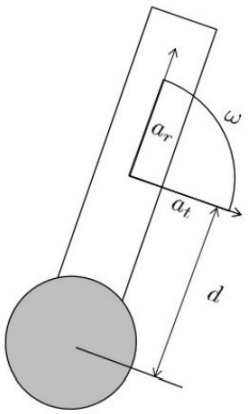

Figure 5: Directions of a_r, a_t and ω

The measurement model uses accelerometer measurements in the radial and tangential directions as shown in Figure 5 to estimate θ, where translational accelerations are neglected and zero-mean white noise is added as measurement noise. This results in the non-linear approximation [11]:

$$y_\theta(t) = arctan2\left(\frac{-a_r(t) + \dot{v}_r(t)}{a_t(t) + \dot{v}_t(t}\right) \approx \theta(t) + \dot{v}_\theta(t).$$

Tuning constant	Unit	Description	Relative magnitude
p_ω	[rad^2.sec^{-2}]	Initial ω state uncertainty	Large
p_b	[rad^2.sec^{-2}]	Initial b state uncertainty	Large
q_ω	[rad^2.sec^{-2}]	ω state uncertainty	Large
q_b	[rad^2.sec^{-2}]	b state uncertainty	Small
r_θ	[rad^2.sec]	θ measurement noise	Large
r_ω	[rad^2.sec^{-1}]	ω measurement noise	Small

Table 2: Kalman filter tuning Parameters

Gyroscopic measurements include the angular velocity and bias states with zero-mean white noise:

$$y_\omega(t) = \omega(t) + b(t) + \dot{v}_\omega(t).$$

Finally, the discrete time measurement model is:

$$\begin{bmatrix} y_{\theta k} \\ y_{\omega k} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix} + \begin{bmatrix} \omega_k \\ \theta_k \\ b_k \end{bmatrix} + \begin{bmatrix} \dot{v}_{\theta k} \\ \dot{v}_{\omega k} \end{bmatrix}, \tag{13}$$

with the associated uncertainty co-variance matrix

$$R = \begin{bmatrix} r_\theta & 0 \\ 0 & r_\omega \end{bmatrix}.$$

The Kalman Filter recursively iterates between the Prediction and the Correction steps to make an estimation, as described below.

4.1.1 Prediction

Predict the mean:

$$\mu_{k|k-1} = A_d \mu_{k-1} + B_d u_k \tag{14}$$

Predict the co-variance:

$$P_{k|k-1} = A_d P_{k-1} A_d^T + Q \tag{15}$$

4.1.2 Correction

Calculate the Kalman Gain:

$$K_k = P_{k|k-1} C_d^T \left(C_d P_{k|k-1} C_d^T + R\right)^{-1} \tag{16}$$

Correct the mean:

$$\mu_{k|k} = \mu_{k|k-1} + K_k(Y_k - C_d \mu_{k|k-1}) \tag{17}$$

Correct the co-variance:

$$P_{k|k} = (I - K_k C_d) P_{k|k-1} (I - K_k C_d)^T + K_k R K_k^T \tag{18}$$

Firstly, we initialise μ and P. We set the mean estimate to zero and set the parameters of P according to Table 2. Then, in Eq. 14 we compute the predicted mean estimate of the state based on the process model, Eq. 12, and the best estimate of the state from the previous time step, μ_{k-1}. The matrix B_d has been set to zero in our case. Eq. 15 computes the co-variance based on the state-transition matrix, A_d, and the process uncertainty, Q. This equation gives us the predicted change in uncertainty of the state.

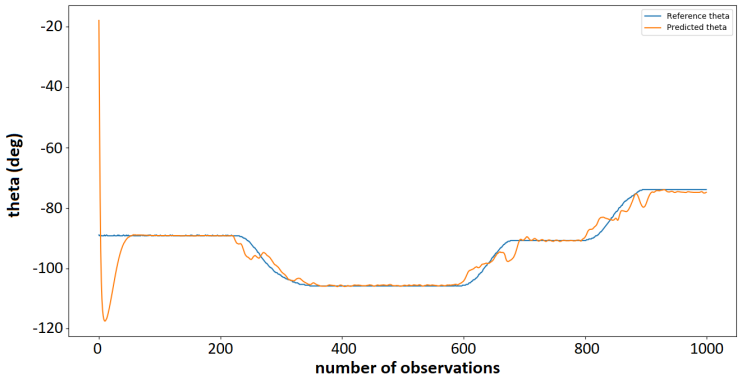

Figure 6: Comparison of predicted and reference θ values

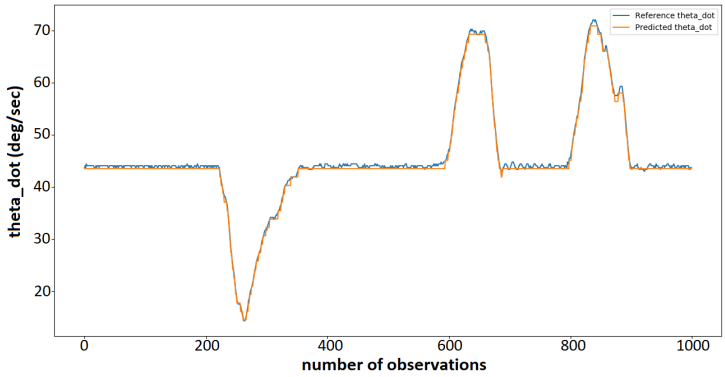

Figure 7: Comparison of predicted and reference $\dot{\theta}$ values

In Eq. 17, the difference between Y_k and $C_d\mu_{k|k-1}$ gives us the difference between the measurement from the sensor and what we expected to measure based on the measurement model, Eq. 13, and the predicted estimate, $\mu_{k|k-1}$. This difference is multiplied by the Kalman Gain, K.

The Kalman Gain is a relative weighting matrix between the predicted mean and the measurement. If there is a large uncertainty associated with the measurements then K would be comparatively smaller and the final mean estimate, $\mu_{k|k}$ would be much closer to the predicted mean. Similarly, Eq. 18 updates the co-variance matrix. Figures 6 and 7 show the plot of θ and $\dot{\theta}$ validated against measurements from an unbiased gyroscope and encoder. Figure 8 shows a plot of raw accelerometer values with the estimated θ.

Using the Kalman Filter, we estimate the chassis' tilt-angle, θ, and angular velocity, $\dot{\theta}$. As there is no sensor on the wheels to measure the change of the wheel angle, ϕ, and its angular velocity, $\dot{\phi}$, we must estimate these values directly from the process model, which will be explained in the next section.

4.2 Estimating Wheel Parameters

We estimate the wheel parameters, ϕ and $\dot{\phi}$, using the method of Variation of Constants. We can trust the estimate of these parameters from the model as stepper motors have the ability to make very precise movements. Naturally, this is a strong assumption that might be invalid in case the wheel slips a little. For a system given by the equation $\dot{x} = Ax + Bu$, the solution of the system at any given time is given by:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-s)}Bu(s)ds. \tag{19}$$

Eq. 19 computes the solution for the entire state x. But, we are interested in the values of ϕ and $\dot{\phi}$, for every time step. As explained earlier our control loop runs every 10^{-3} seconds. So, we want to estimate these values in the time that a control action has been applied and the controller is ready with a new value of τ. And thus, we consider a small window from t to $t + h$ for $h = 10^{-3}s$. The value of ϕ and $\dot{\phi}$ at $t + h$ would be dependent on the value at time t.

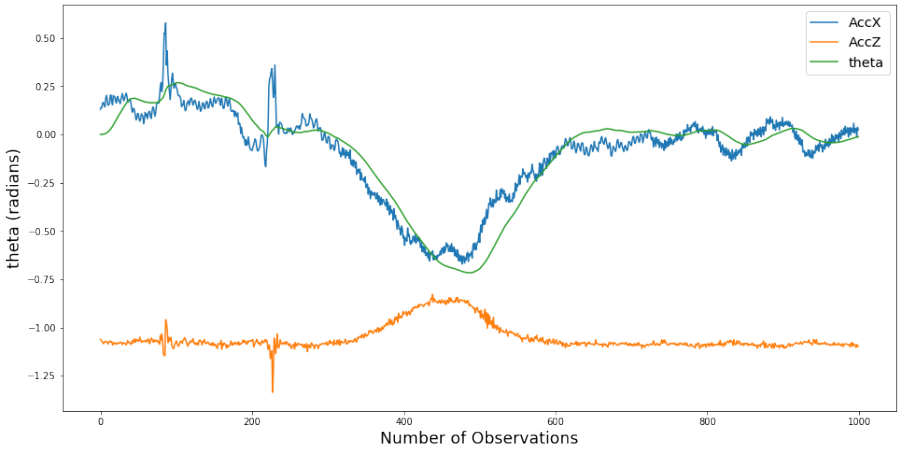

Figure 8: Estimated θ from KF on IMU data

$$\dot{\phi}(t+h) = \left[e^{Ah} \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} \right]_4 + u \int_t^{t+h} e^{A(t-s)} B ds \tag{20}$$

Expanding e^{Ah}:

$$\begin{aligned} e^{Ah} &= I + hA + \frac{1}{2}h^2A^2 + \frac{1}{3}h^3A^3 + ... \\ &\approx I + hA, \end{aligned}$$

as h is very small, and higher powers of h tend to 0. Substituting e^{Ah} in Eq. 20:

$$\begin{aligned} \dot{\phi}(t+h) &\approx \dot{\phi}(t) + \left[hA \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} \right]_4 + \int_t^{t+h} (I + (t-s)A)\, ds Bu \\ &\approx \dot{\phi}(t) + \left[hA \begin{bmatrix} \theta \\ \dot{\theta} \\ \phi \\ \dot{\phi} \end{bmatrix} \right]_4 + (hBu)_4 . \end{aligned} \tag{21}$$

Computing $\dot{\phi}$ and applying it to the motor for the time step h we obtain ϕ as:

$$\phi(t+h) \approx \phi(t) + h\dot{\phi}(t). \tag{22}$$

The matrix multiplication of A with the state, and B with the control input, in Eq. 21 yields a 4×1 vector. We are interested in the 4^{th} element of this column vector as that is the the element corresponding to $\dot{\phi}$.

Having estimated all four state variables, it can be fed back into the controller, thus creating a loop.

5 System Structure

This section provides a brief description on how the components have been arranged. Figure 9 shows the front view of the self-balancing robot with all the components. The chassis is 3D printed and is made of Polylactic Acid (PLA), which is strong enough to carry the components of the robot. Raspberry Pi 3B+ and Motor drivers are mounted on the middle and bottom section of the chassis respectively. A pair of stepper motors are encased at the bottom, with wheels attached to the shaft. The IMU is placed on the top of the robot in order to accurately sense the change in the motion and tilt angle. Batteries are positioned at an appropriate height to reasonably increase the distance of the center of gravity (COG) from the shaft of the motor.

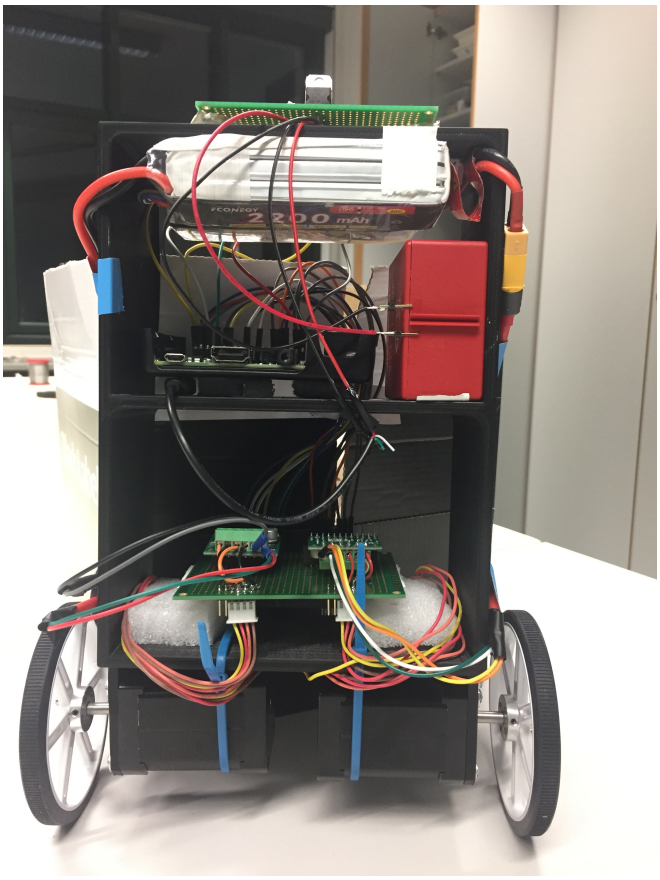

Figure 9: Front view of the self-balancing robot

5.1 Micro Controller

Raspberry Pi 3B+ is used as a development board mainly because of its high performance and functionality. It offers a processing speed of 1.4 GHz which is more than enough to compute the processing tasks such as acquisition of the data, control computation and sending the PWM signals to the motor drivers. In this project SPI and I2C communication protocols have been used for the communication with the motor drivers and the IMU respectively.

5.2 Sensor

MPU 6050 an inertial measurement unit used to determine the angular velocity and total external acceleration of the robot. MPU 6050 is a 6-axis (Gyro + Accelerometer) motion device. The gyroscope measures the angular velocity and accelerometer measures the gravitational and motion acceleration and these measurements are sent to the development board by using I2C communication protocol. In Section 4 the estimation of the angle of inclination θ and the angular velocity of the chasis $\dot{\theta}$ is explained in detail.

5.3 Stepper Motor and Motor Drivers

Stepper motors are widely used in applications which require quick positioning in short distances. Stepper motors provide high torque and low vibrations at low-speed which is important for our self balancing robot. In this project, NEMA-17 2-phase bipolar stepper motors are used with a holding torque of $0.8Nm$. It can move with a step angle of $1.8°$and requires a continuous current of 1.8 $Ampere$ per phase.

The motor operates by accurately synchronizing with the pulse signal output from the controller to the driver, achieving highly accurate positioning and speed control [9]. In this project, Pololu A4988 and AMIS-30543 micro stepping motor-drivers are used. The A4988 can supply a continuous current of 1 A per phase without external cooling and 2 A per phase with cooling. The AMIS-30543, on the other hand, can provide a continuous current of 3 A per phase with

and $1.8\ A$ per phase without cooling. The AMIS-30543 can be interfaced through the SPI protocol with a development board. It has an on-chip voltage regulator and SPI-adjustable current limiting. The A4988 motor driver allows a configuration of upto 16 micro-steps while AMIS-30543 allows for upto 128 micro-steps per step angle for a smoother movement.

5.4 Power supply

The motors operate at $5.4\ V$ each and draw a current of $1.8\ A$. The motor drivers have a minimum operating voltage of $6\ V$ and the Raspberry Pi operates on $5\ V$. Considering the power requirements of all the components, two $11.1\ V$ Li-Po batteries, and a $8.4\ V$ NiMH battery is used.

5.5 Center of Gravity

Section 2 defined the parameter l, which is the distance of the Center of Gravity (CoG) from the wheel axle. The components were placed such that the overall CoG of the robot moves away from the axle. This causes a delay in the fall of the robot. Naturally, the CoG should not be too high up for our motors to not be able to provide enough torque to balance it. To calculate the position of the CoG from the axle, we used the following equation:

$$l = \frac{x_1 m_1 + x_2 m_2 + \ldots + x_n m_n}{m_1 + m_2 + \ldots + m_n},$$

where $m_1, m_2, ...$ are the masses of the individual components and $x_1, x_2, ...$ are the distances of those components from the axle.

After assembling the components of the robot, we experiment how all the theoretical concepts described in this report actually work in the real world. This also introduces chance for errors as mechanical and electrical components are prone to damage/break down during implementation.

6 Experimental Evaluation

As it is naturally the case with all hardware implementations, many times the components do not function as expected. Additionally, it is only during implementation that some faults are discovered. In this section we address all the experimental issues that we faced during the implementation and possible solutions.

Initially, the motor driver that was chosen for this project was the Pololu A4988. Some time later it was discovered that the A4988 is not suitable as the maximum continuous current per phase (without external cooling) is only $1\ A$. Even with external cooling the A4988 could only supply a current of $2\ A$ and the motors would vibrate vigorously as we increased the angular velocity of the wheels. The motors need a current of $1.8\ A$ per phase. Following this limitation, the A4988 was switched out in favour of the AMIS-30543 motor driver. Additionally, as a result of intensive mechanical vibrations, the wheels would often fall off from the shaft. To counter this, the motor shaft was glued to the wheels.

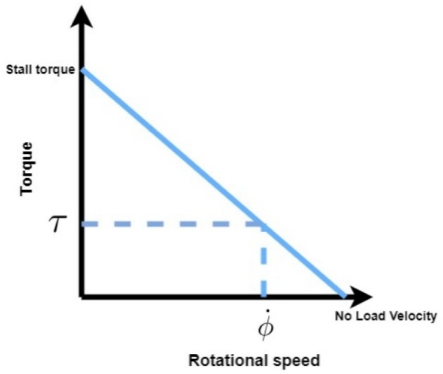

Figure 10: Torque-speed relation

The initial estimation of the parameters ϕ and $\dot{\phi}$ was done using the torque-speed characteristics of the NEMA-17 motor. Ideally, the torque of the motor is inversely proportional to the speed of the shaft. The torque was calculated by

the controller and the corresponding angular velocity of the wheel, $\dot{\phi}$, was calculated following the curve, as shown in Figure 10. Though, this method did not yield the desired result. $\dot{\phi}$ diverged quickly and the motor would vibrate intensely. Later, we estimated these parameters from the existing mathematical model, as described in Section 4.

7 Conclusion

In this report, a two-wheeled self balancing robot has been developed using an LQR controller. The Kalman filter has been used for state estimation. This report presented a general mathematical approach to building a self-balancing robot. Though, as of the end of the project the robot could not be stabilized in an upright position. Here are a few changes and future scope to improve the performance:

1. Relocating the IMU to the position of the Center of Gravity to better estimate the parameters as opposed to placing.
2. During testing, it was observed that the robot never achieved enough velocity to counter its fall. The LQR could be tuned further to achieve a balance.
3. The components were interconnected on separate general purpose boards, with many wires going around the robot. This is a possible source of errors, and a printed circuit board should be used to limit the interconnections to within the board.
4. A possible extension is to combine the Raspberry Pi with other sensors to implement a Simultaneous Localization and Mapping System.
5. Furthermore, as Raspberry Pi provides support for a camera, Object Detection and Tracking could also be implemented.

In conclusion, the hardware implementation worked partially due to above possible reasons and also there are chances of some undetected flaws. However, theoretically, as well as during simulation and prototyping the individual components of the robot, the outputs were as expected and successful.

References

[1] H. Juang and K. Lurrr. "Design and control of a two-wheel self-balancing robot using the arduino microcontroller board". In: (June 2013), pp. 634–639. ISSN: 1948-3449. DOI: 10.1109/ICCA.2013.6565146.

[2] F. Grasser et al. "JOE: a mobile, inverted pendulum". In: *IEEE Transactions on Industrial Electronics* 49.1 (Feb. 2002), pp. 107–114. DOI: 10.1109/41.982254.

[3] Lal Bahadur Prasad, Barjeev Tyagi, and Hari Om Gupta. "Optimal Control of Nonlinear Inverted Pendulum System Using PID Controller and LQR: Performance Analysis Without and With Disturbance Input". In: *International Journal of Automation and Computing* 11.6 (Dec. 2014), pp. 661–670. ISSN: 1751-8520. DOI: 10.1007/s11633-014-0818-1. URL: https://doi.org/10.1007/s11633-014-0818-1.

[4] David P. Anderson. *nBot Balancing Robot*. 2003. URL: http://www.geology.smu.edu/~dpa-www/robo/nbot/.

[5] K. Su and Y. Chen. "Balance control for two-wheeled robot via neural-fuzzy technique". In: *Proceedings of SICE Annual Conference 2010*. Aug. 2010, pp. 2838–2842.

[6] Tatsuya Nomura et al. "Adaptive backstepping control for a two-wheeled autonomous robot". In: *2009 ICCAS-SICE*. Aug. 2009, pp. 4687–4692.

[7] X. Ruan and J. Cai. "Fuzzy Backstepping Controllers for Two-Wheeled Self-Balancing Robot". In: *2009 International Asia Conference on Informatics in Control, Automation and Robotics*. Feb. 2009, pp. 166–169. DOI: 10.1109/CAR.2009.42.

[8] N. Dini and V. J. Majd. "Model predictive control of a wheeled inverted pendulum robot". In: *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*. Oct. 2015, pp. 152–157. DOI: 10.1109/ICRoM.2015.7367776.

[9] 28. *Oriental motor*. URL: https://www.orientalmotor.com/stepper-motors/technology/everything-about-stepper-motors.html.

[10] *Mathworks*. URL: https://www.mathworks.com/matlabcentral/mlc-downloads/downloads/submissions/4609/versions/1/previews/QCAT/doc/intro.html#_alloc.

[11] Roxanne Jackson. "Kalman Filter using IMU Measurement Data (unpublished)". In: (2019).

[12] Roxanne Jackson. "Euler-Lagrangian Model of Self Balancing Robot (unpublished)". In: (2019).