# Design of a Two-Wheel Self-Balancing Robot with the Implementation of a Novel State Feedback for PID Controller using On-Board State Estimation Algorithm

**Article** · December 2018

DOI: 10.24247/ijrrddec20181

CITATION
1

READS
7,774

**2 authors:**

Chinmay Samak
SRM Institute of Science and Technology
**11** PUBLICATIONS **5** CITATIONS
SEE PROFILE

Tanmay Samak
SRM Institute of Science and Technology
**11** PUBLICATIONS **5** CITATIONS
SEE PROFILE

International Journal of Robotics
Research and Development (IJRRD)
ISSN(P): 2250-1592; ISSN(E): 2278-9421
Vol. 8, Issue 2, Dec 2018, 1-10


TRANS STELLAR
•Journal Publications •Research Consultancy

# DESIGN OF A TWO-WHEEL SELF-BALANCING ROBOT WITH THE IMPLEMENTATION OF A NOVEL STATE FEEDBACK FOR PID CONTROLLER USING ON-BOARD STATE ESTIMATION ALGORITHM

**CHINMAY VILAS SAMAK & TANMAY VILAS SAMAK**

*Department of Mechatronics, SRM Institute of Science and Technology, Tamil Nadu, India*

***ABSTRACT***

***This research was aimed at designing, developing and analyzing a self-balancing robot with the implementation of a novel state feedback for the PID controller using on-board state estimation. Unlike most of the previous works on self-balancing robots, which involved state estimation using IMU and feedback from the actuator side using encoders, potentiometers or tachometers; this research predominantly focused at implementing an on-board state estimation algorithm to generate the feedback, thus eliminating the requirement of sophisticated actuators. Such a system would not only be cost-efficient but would also be more robust in terms of state estimation, since the actuator feedbacks have a recurrently bad impression of misguiding the controller. Subsequently, the robot was designed and a prototype of it was tested for its performance. The performance analysis of the robot exhibited the expected behavior in terms of stability and robustness of the control design.***

***KEYWORDS*: *Robotics, Inverted Pendulum, PID Controller, Feedback, Accelerometer & Gyroscope***

**Received:** Sep 22, 2018**; Accepted:** Oct 12, 2018**; Published:** Nov 01, 2018**; Paper Id.:** IJRRDDEC20181

Original Article

## 1. INTRODUCTION

Various designs of self-balancing robots have been proposed to date, but only a handful of them have achieved success in actually balancing the robot with high stability and robustness. Some of the previous works include the famous Segway Personal Transporters (Segway Inc.); a Self-Balancing Vehicle (Grepl, R., et al., 2016) by Brno University of Technology, which is low-cost design of a self-balancing personal transporter; LegWay (Steve, 2002), which is a self-balancing robot that can be controlled remotely or made to follow a line; Balance Bot (The Art of Invention), which has built-in obstacle avoidance algorithm; Balanduino (TKJ Electronics), which is an open source self-balancing robot; a Two-Wheel Self-Balancing Robot (Hellman, et al., 2015) by KTH; another Self-Balancing Robot (Juang & Lum, 2013); and many more.

The design of a self-balancing robot is based upon the fundamentals of the “Inverted Pendulum” model, which describes a system, wherein an inverted pendulum has to be maintained upright by moving the wagon it is attached to under its center of mass. This idea was extended to design a robot that had to balance itself on two wheels by driving its base of support under the center of mass and thus align itself perpendicular to the floor.

The robot was inherently unstable and needed to be constantly controlled to correct itself and balance. There were two parameters to be controlled in this scenario: direction and acceleration. While the direction of the

motion depended upon the direction of rotation of the wheels, the acceleration was dependent on the angular acceleration of the wheels (since, $\ddot{x} = r * \ddot{\theta}$). The tilt of the robot had to be sensed in order to provide the necessary feedback to the controller and effectively drive the correction elements. This was achieved by implementing a closed-loop control system, the PID controller.

## 2. SYSTEM MODELLING

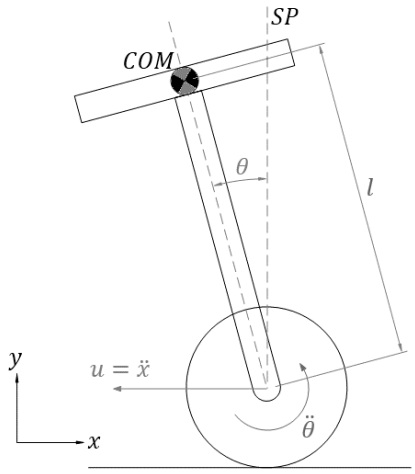


**Figure 1: Dynamics of a Self-Balancing Robot**

Since this research was aimed at developing a self-balancing robotic system, the prime objective was only to maintain the robot erect (i.e. control the pitch angle) and not to control its position; hence, the unicycle model was ignored and the robot was essentially considered to be an inverted pendulum on a cart model. Thus, the equation of motion of the self-balancing robot was derived using Lagrange's equations (Inverted Pendulum, Wikipedia) by analyzing the dynamics of the robot (Figure 1).

$$l\ddot{\theta} - gsin\theta = \ddot{x}cos\theta$$

Here, $l$ is the distance of the center of mass of the robot from the axis of rotation (i.e. wheel center), $g$ is the acceleration due to gravity, $\theta$ is the pitch angle, $\ddot{\theta}$ is the angular acceleration and $\ddot{x}$ is the linear acceleration (fed as control input $u$).

Dynamics of the system were therefore written as:

$$\ddot{\theta} = \frac{g}{l}sin\theta + \frac{u}{l}cos\theta$$

The state of the system was represented as follows:

$$State: \begin{cases} x_1 = \theta \\ x_2 = \dot{\theta} \end{cases} \Rightarrow x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

Since we were measuring the pitch angle $\theta$ of the robot,

$$Output: y = \theta$$

As discussed earlier, the control signal influenced the linear acceleration of the base of the robot. Therefore,

$$Control\ Signal: u = \ddot{x}$$

Thus, orders of matrices $x$, $u$ and $y$ were described as:

$$x \in \Re^2, u \in \Re^1, y \in \Re^1$$

Hence, we obtained the open-loop system dynamics as follows:

$$\dot{x} = f(x,u) = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{g}{l} sin\theta + \frac{u}{l} cos\theta \end{bmatrix}; y = h(x) = x_1$$

Since the dynamics of the robot were non-linear, it was required to linearize them around an operating point $(x_0, y_0)$, which was chosen to be $(0,0)$; i.e. when the robot was upright and there was no control input. The state space model of the linear time invariant (LTI) system was therefore derived to be:

$$\dot{x} = Ax + Bu; y = Cx + Du$$

Here, $A$, $B$, $C$ and $D$ are Jacobians that were computed as follows:

$$A = \frac{\partial f}{\partial x} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix}_{(0,0)} = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix}; B = \frac{\partial f}{\partial u} = \begin{bmatrix} \frac{\partial f_1}{\partial u} \\ \frac{\partial f_2}{\partial u} \end{bmatrix}_{(0,0)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; C = \frac{\partial h}{\partial x} = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} \end{bmatrix}_{(0,0)} = [1 \quad 0]; D = [0]$$

Now, since both $g$ and $l$ were positive real values, either of the eigen values of matrix $A$ would have its real part greater than zero, i.e. the system would be unstable in nature. Thus, a state feedback (i.e. $u = -Kx$) was essential to regulate the system about the setpoint. Since the on-board state estimation algorithm provided the necessary feedback, we obtained the closed-loop system dynamics as follows:

$$\dot{x} = [A - BK]x = \left( \begin{bmatrix} 0 & 1 \\ \frac{g}{l} & 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix} * [k_1 \quad k_2] \right) x = \begin{bmatrix} 0 & 1 \\ \frac{g}{l} - k_1 & -k_2 \end{bmatrix} x$$

The eigen values of $[A - BK]$ were determined from its characteristic equation given by:

$$\chi_{[A-BK]}(\lambda) = \det(\lambda I - [A - BK]) = \begin{vmatrix} \lambda & -1 \\ k_1 - \left(\frac{g}{l}\right) & \lambda + k_2 \end{vmatrix} = \lambda^2 + k_2 * \lambda + \left(k_1 - \frac{g}{l}\right)$$

Consequently, using the concept of pole placement, desirable eigen values (i.e. poles with the strictly negative real part) were placed in order to determine the values of $k_1$ and $k_2$, which defined the $K$ matrix. Once the $K$ matrix was defined, the system achieved asymptotic stability. Following the separation principle, the on-board state-estimation results (observer design) were then combined with the controller design (with state feedback) so as to effectively control the plant (self-balancing robot).

## 3. CONTROL SYSTEM

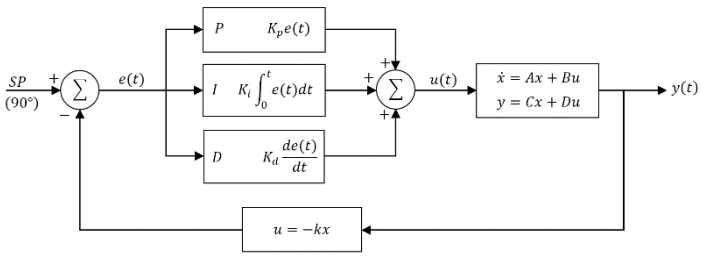


**Figure 2: PID Controller for the Self-Balancing Robot along with the State Feedback**

Since the robotic system had highly unstable dynamics, utilization of a control system was extremely necessary in order to stabilize it about the setpoint (upright position). Thus, the PID controller (Figure 2) was implemented in order to maintain the verticality of the robot.

The prime objective of the PID controller was to maintain the value of pitch angle $\theta$ of the robot close to the setpoint (SP) value, which in our case was 90˚ since the IMU was mounted perpendicular to the base of the robot. The setpoint value was fed to the controller as the reference signal and the error $e(t)$ was calculated by means of negative feedback (i.e. by subtracting estimated state from the setpoint). The error was then used to compute the correction signal or the input signal $u(t)$ as follows:

$$u(t) = Kp * e(t) + Ki * \int_0^t e(t)dt + Kd * \frac{de(t)}{dt} \quad (1)$$

Here $Kp$, $Ki$, and $Kd$ are the controller gains for the PID controller. It was noted that the integral term in the above equation was the accumulation of error over time and the derivative term was the rate of change of error. Hence, the individual terms were replaced as follows:

$$e(t) = e;\ \int_0^t e(t)dt = E * dt;\ \frac{de(t)}{dt} = \frac{e - e_o}{dt}$$

Here, $e$ is the present error, $E$ is the summation of all the errors over time, $e_o$ is the previous error and $dt$ is the sampling time (which in our case was 12.5 ms). Therefore, equation (1) was re-written as:

$$u(t) = Kp * e + Ki * E * dt + Kd * \frac{e - e_o}{dt}$$

Physically, the proportional term corrected the system linearly with respect to the error and therefore often built up an error offset. The integral term was particularly helpful when a persistent error offset had been developed. The derivative term resisted any sudden change in the error and therefore soothed the system response. Thus, the proportional term corrected the present error, the integral term compensated for the past errors and the derivative term predicted and avoided any future errors.

## 4. PROTOTYPING AND TESTING

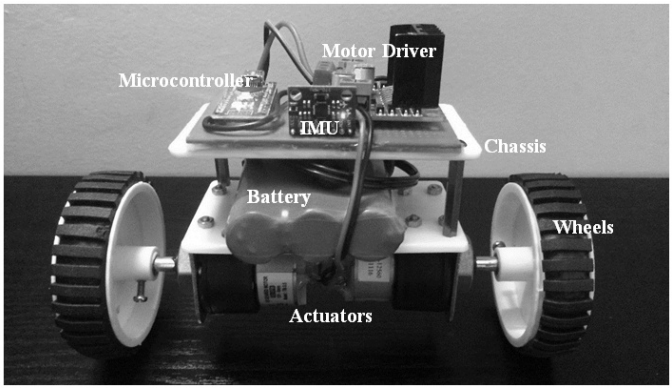


**Figure 3: Prototype of the Self-Balancing Robot Exhibiting the Most Unstable Structure with a Reduced Height and Lower Center of Mass**

### 4.1 Mechanical Systems

The mechanical chassis of the robot comprised of two layers. One layer housed the battery while the other housed the on-board electronics. The two layers were joined using brass hex spacers so that the length of the robot could be varied during experimental analysis. The motors were fastened to the lower layer using an L-clamp, each, and wheels (with grip) were installed onto the motor shafts using screws. The grips provided adequate friction to the robot while balancing as the robot had a fair chance of slipping during high-velocity responses.

### 4.2 Electrical Systems

#### 4.2.1 Power Supply

The entire on-board electronics and actuators were powered using an 11.1 V 2200 mAh Li-ion battery pack. The battery capacity was chosen to be high enough so as to provide both the motors with adequate current and at the same time, last for a prolonged duration (to analyze the robot over a period of time).

#### 4.2.2 Inertial Measurement Unit (IMU)

GY-521, a 6 DOF IMU, was used for state estimation of the robot. The breakout board features a combined MEMS-based accelerometer and gyroscope, MPU-6050 (Arduino Playground) as well as a Digital Motion Processor (DMP). The accelerometer measures the acceleration in all the three axes (x, y, z) but it is susceptible to noise. On the other hand, the gyroscope, which is stable towards the noise, measures the angular rate about all the three axes (x, y, z); however, its values tend to drift away along with time. It therefore, became necessary to consolidate both the individual sensors (termed as “sensor fusion”) as neither of them was reliable, individually. The output of these sensors was stored in FIFO buffer packets of 1024 bytes; however, utilizing this raw data was cumbersome. Therefore, the DMP was used to perform complex calculations on the raw sensor data on-chip itself, thereby eliminating the need of computing it separately on the microcontroller side. Lastly, the IMU’s 16-bit analog-to-digital converter (ADC) was used to convert the analog signals from the sensors into readable digital signals for the microcontroller.

#### 4.2.3 Microcontroller

Arduino Nano (ATmega328) was chosen as the microcontroller due to its availability and the fact that it is an open source hardware. It supports the I2C communication protocol and is therefore compatible with the IMU chosen. Furthermore, it can be easily interfaced with different types of actuators (DC motors, servo motors, stepper motors, etc.) and is compact in size. Although its computational resolution is less as compared to other sophisticated microcontrollers, the fact that it is cost effective made it an ideal candidate for prototyping our robot.

#### 4.2.4 Motor Driver and Actuators

L298N, a dual H-Bridge motor driver, was chosen for managing power distribution to the actuators due to its support for pulse-width modulation (PWM). PWM was used to regulate the speed of the motors, which was very essential so as to implement the controller. The choice of two 500 rpm, 2 kg-cm DC motors was made as the actuators due to their simplicity and cost-effectiveness. Although sophisticated actuators such as stepper motors and servo motors provide better feedback, since the position of the robot was not to be controlled, they were not explicitly required.

### 4.3 Block Diagram and Algorithms

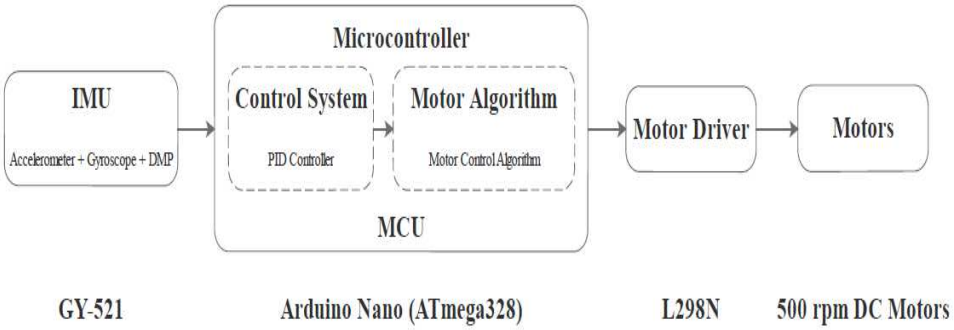


**Figure 4: Block Diagram of the Self-Balancing Robot**

The fundamental stage in the working of the self-balancing robot is the on-board state estimation of its tilt angle. This was accomplished with the help of IMU, more specifically, MPU-6050. This raw sensor data was then passed onto the DMP wherein the individual sensor errors were compensated, the values were auto-calibrated and complex calculations were performed upon them in order to compute the tilt angles with respect to all the three axes (roll, pitch, yaw). Since the robot had only 1 DOF, only pitch value was considered. This tilt angle was then communicated to the microcontroller unit (MCU) through a 16-bit ADC via I2C bus.

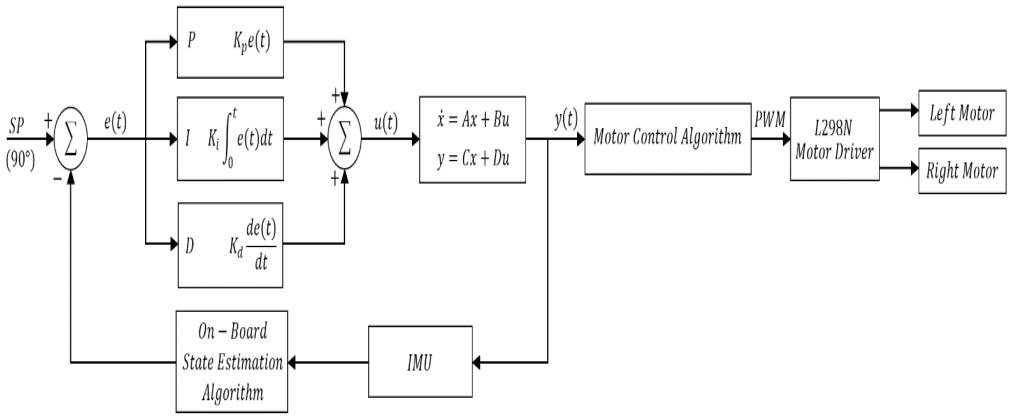


**Figure 5: System Representation of the Self-Balancing Robot**

The tilt angle was fed to the PID controller algorithm and was compared against the setpoint. The error was computed and a corresponding correction signal was generated in the MCU, which was then mapped in the range of [-255, 255] and fed to the motor control algorithm. This algorithm determined the speed and direction of rotation of both the motors. The mechanical offsets amongst the motors were also compensated with the help of this algorithm. The output signals from the motor control algorithm were fed to the motor driver module, which accordingly controlled the motors at the rated current to maintain the robot in an upright position.

## 5. RESULTS

The data from the actual robot was communicated with the laptop at a baud rate of 115200 bits per second, and live plots were generated with respect to time. The parameters studied include pitch angle of the robot, controller output, and motor PWM values. It is to be noted that the three parameters were not analyzed simultaneously.

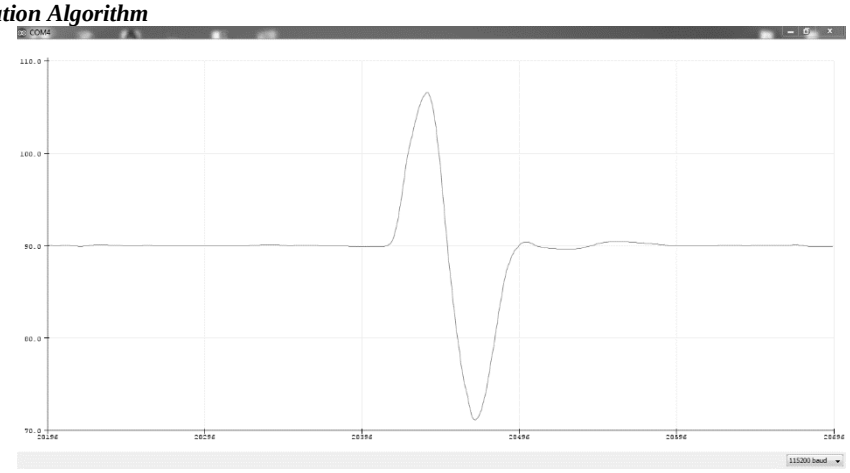


**Figure 6: Robot Response with Time – Angle. The x-Axis Represents Time in Milliseconds and the y-Axis Represents Pitch Angle in Degrees**

When the pitch angle of the robot was analyzed with respect to time (Figure 6), it was seen that the angle was 90˚ (setpoint) when the robot was balanced. Nevertheless, when the robot was disturbed by hitting it slightly, the angle changed to a value of around 106˚. The controller then tried to correct the robot and bring it back to the setpoint; however, due to inertia, the robot got tilted in the opposite direction at about 72˚. The controller again corrected the error and brought the robot back to the setpoint. This time, it predicted the future error and damped the oscillations (derivative control) and the robot stabilized about the setpoint.

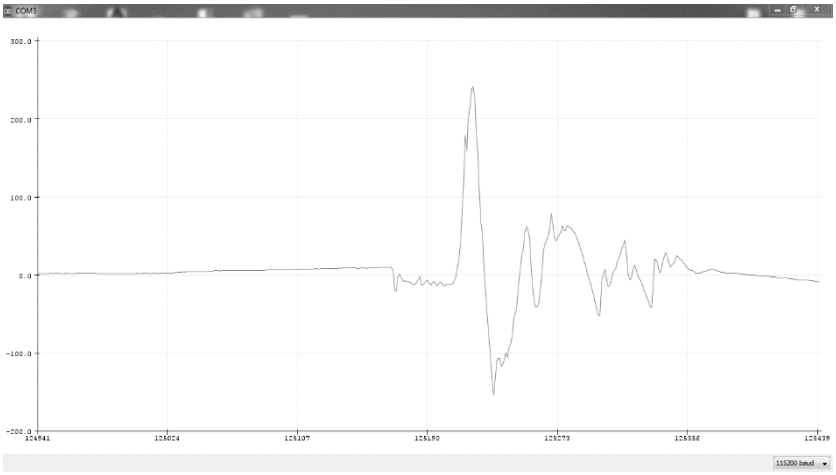


**Figure 7: Robot Response with Time – Controller Output. The x-Axis Represents Time in Milliseconds and the y-Axis Represents the Output from the PID Controller**

When the controller output was analyzed with respect to time (Figure 7), it was observed that as the robot was stable, the controller did not generate any output to correct it. Nevertheless, once the robot was disturbed from the setpoint, the controller generated a sudden peak value in order correct the robot; however, when the robot tried to correct itself, due to inertia, it got tilted in the opposite direction. This caused the controller to generate a negative peak in response to the robot's tilt. Note that this value was lesser than the previous positive peak, which indicates that the effect was damped by the D-regulator. Eventually, the controller brought the robot back to the setpoint by nullifying the error. It is worth noting, here, that the entire operation took only about 166 milliseconds.

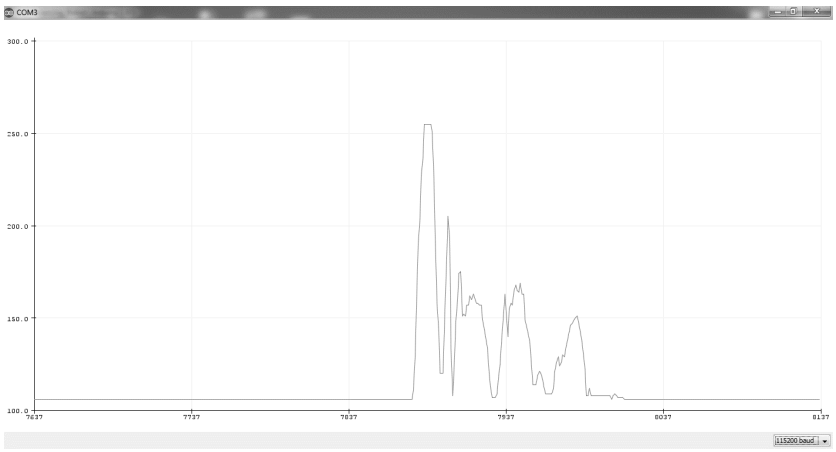


**Figure 8: Robot Response with Time – Left Motor PWM. The x-Axis Represents Time in Milliseconds and the y-Axis Represents the PWM Values**

When the left motor PWM values were analyzed with respect to time (Figure 8), it was observed that when the robot was balanced, the motors were off (please note that motors have a threshold value of PWM below which they stop completely due to inertial effect. Also note that both the motors need not have the same threshold value, which is termed as "motor offset". In our case, the threshold PWM values for left and right motors were experimentally found out to be 105 and 55 respectively, at which the motors just impended to start). Once the robot was disturbed, the PWM was shot to maxima (i.e. 255). As the robot corrected itself, the PWM was reduced but since the robot got tilted in the opposite direction, it increased again, although this time the direction of rotation was reversed. Eventually, the controller damped the oscillations and balanced the robot, and the PWM value was dropped back to 105, within 120 milliseconds. Similar was the case of right motor PWM.

## 6. DISCUSSIONS

From the performance analysis of the robot, it was understood that the following parameters were most significantly responsible for achieving stability and robustness in the control design of the self-balancing robot.

### 6.1 Controller Gains

The controller gains $K_p$, $K_i$ and $K_d$ determined the response of the PID controller to the error (a difference between the set point and estimated state of the robot). It was observed that the controller gains were specific for a particular system and needed to be tuned experimentally until stability in the control design was achieved. As an example, for the most unstable prototype of our robot (Figure 3), the stability was achieved at $K_p = 10.5$, $K_i = 70$ and $K_d = 0.35$; the values varied for different designs that were tested (by varying height and mass distribution).

### 6.2 Moment of Inertia of the Robot

The moment of inertia of the robot was varied by changing its height (using hex spacers) and the center of mass (mass distribution). For a robot with a lower value of the moment of inertia, it was effortless to correct the pitch angle; however, at the same time, the robot would also get disturbed easily. This was because the angular acceleration is inversely proportional to the moment of inertia of a body.

$$\tau = I * \alpha \ \Rightarrow \alpha = \frac{\tau}{I}$$

Here, $\tau$ is the torque, $I$ is the moment of inertia and $\alpha$ is the angular acceleration. Therefore, a body with a lower value of the moment of inertia would accelerate faster, and hence, fall faster.

In contrast, for a robot with a higher value of the moment of inertia, it took more effort to correct the pitch angle; but once corrected, it tried not to deviate from this value. This was because the angular acceleration, in this case, was lower, meaning slower fall. Hence, the controller had enough time to correct the robot. It was therefore inferred that a taller robotic system with a higher center of mass is preferable in terms of achieving control stability.

### 6.3 Motor Parameters

There are two fundamental motor parameters viz. rpm and torque. The motor rpm governed the responsiveness of the robot to correct itself and thus, a higher rpm motor would correct the robot more quickly but would tend to slip. Furthermore, a motor with an excessively high rpm value would not be able to produce enough torque required to erect the robot. On the other hand, a lower rpm motor would be able to produce enough torque but it would take too long to correct the robot, making it fall eventually. Hence it was necessary to calculate the torque of the motor (Trivedi, 2017) and choose the motor of maximum rpm corresponding to this torque value.

$$\tau = \frac{l*m*(a+g*sin\theta)}{N}$$

Here,$\tau$ is the torque, $l$ is the perpendicular distance between the axis of rotation and point of application of force, $m$ is the mass, $a$ is the linear acceleration, $g$ is the acceleration due to gravity, $\theta$ is the tilt angle and $N$ is the number of actuators used.

Apart from these two, another motor parameter that affected the robotic system was the gears. Gear backlash was a major problem in controlling the robot precisely. Poorly manufactured gears tended to lock or slip at certain points making motors unresponsive for that period of time. Even though this seemed to be an insignificant time interval, it did affect the dynamics of the robot.

### 6.4 IMU Parameters

There were two fundamental parameters that affected the system; one being the IMU positioning and the other being the sensor update rate.

Since the accelerometer is susceptible to noise and vibrations, it was required to be mounted away from the motors. Although the gyroscope does not have such problems, since it was also mounted on the same breakout board, it was advisable to mount the IMU away from the motors so as to accurately measure inertial data of the robot.

The sensor update rate, if too low, reduced the accuracy of state estimation of the robot. This drastically affected the controller, which could not generate the correction signals quick enough. As a result, the robot was not able to balance itself and fell over within no time. On the contrary, if the update rate was too high, it called for very high computational requirements, and since the microcontroller used could not perform the algorithms fast enough, it tended to calculate incorrect values, which again, lead to an inefficient control system. Tweaking with the sensor update rate, we were able to narrow down to a value of “12.5 ms” for Arduino Nano (ATmega328). This simply means that the IMU would estimate the state of the robot after every 12.5 milliseconds.

## 7. CONCLUSIONS

The mechatronic system of a self-balancing robot was designed and constructed successfully by applying the key concepts in engineering mechanics, electronics, programming, and control theory, and it performed at a considerably high level of accuracy. Performance analysis of the robot exhibited expected behavior, along with some small undesirable oscillations and minute errors in state estimation that were caused due to the physical limitations of the sensors (update rate) and actuators (predominantly gear backlash). These imperfections can be easily avoided by selecting precisely manufactured sensors and actuators.

Future considerations of this research include analysis of different control systems, position control of the robot and implementation of autonomous features such as obstacle avoidance, perimeter following, path planning, etc.

***REFERENCES***

1. *Segway Inc. [Online]. Retrieved from: http://www.segway.com/*
2. *Grepl, R., et al. (2016). The Development of a Self-Balancing Vehicle: A Platform for Education in Mechatronics. Brno University of Technology. Retrieved from: http://dsp.vscht.cz/konference_matlab/MATLAB11/prispevky/040_grepl.pdf*
3. *Steve. (2002). Steve's LegWay. Retrieved from: http://www.teamhassenplug.org/robots/legway/*
4. *The Art of Invention. (2018). Retrieved from: http://www.art-of-invention.com/robotics/*
5. *TKJ Electronics. (2013-2018). Retrieved from: http://www.balanduino.net/*
6. *Hellman, et al. (2015). Two-Wheeled Self-Balancing Robot. Design and control based on the concept of an inverted pendulum. Retrieved from: https://kth.diva-portal.org/smash/get/diva2:916184/FULLTEXT01.pdf*
7. *Hau-Shiue, Juang & Lum, Kai-Yew. (2013). Design and Control of a Two-Wheel Self-Balancing Robot using the Arduino Microcontroller Board. 10th IEEE International Conference on Control and Automation (ICCA). 634-639. Retrieved from: https://www.researchgate.net/publication/261126156_Design_and_control_of_a_two-wheel_self-balancing_robot_using_the_arduino_microcontroller_board*
8. *Wikipedia. (2018). Inverted Pendulum. Retrieved from: https://en.wikipedia.org/wiki/Inverted_pendulum*
9. *Arduino Playground. (2018). MPU-6050 Accelerometer + Gyro. Retrieved from: https://playground.arduino.cc/Main/MPU-6050*
10. *Gunjan Trivedi. (2017). Design and Development of Self Balancing Robot. Retrieved from: https://www.slideshare.net/gunjantrivedi2006/design-and-development-of-self-balancing-robot*