International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 07 Issue: 09 | Sep 2020 www.irjet.net p-ISSN: 2395-0072

Design and Implementation of a Self-Balancing Two-Wheeled Robot Driven by a Feed-Forward Backpropagation Neural Network

Arunit Maity[1], Sarthak Bhargava[2]

[1]*B. Tech Student, School of Electronics Engineering (SENSE), VIT Vellore 632014, Tamil Nadu, India*
[2]*B. Tech Student, School of Electronics Engineering (SENSE), VIT Vellore 632014, Tamil Nadu, India*

Abstract – *This research paper reports the design, construction and a novel control system for a two-wheeled self-balancing robot. The system architecture comprises of a pair of DC motors and an Arduino Uno R3 microcontroller board, a 3-axis gyroscope and a 3-axis accelerometer (employed for altitude determination). The constructed two-wheeled robot is essentially the real-time model of an inverted pendulum open-loop system which is inherently highly unstable without feedback control and has nonlinear dynamics. The proposed balancing algorithm consists of training a feedforward neural network using the backpropagation algorithm to learn to balance the bot through supervised learning on the basis of a training routine which runs at startup. In addition, the linear quadratic estimation algorithm (LQE) along with a complementary filter are used to compensate for gyro drifts. Our experimental results show that the proposed feedforward neural network balancing technique can learn to balance the bot within a short span of time and that too with considerably lesser oscillations about the equilibrium point as compared to the standard PID controller, thereby increasing the system's elegance. Our designed bot is a compact and cost-effective protype that showcases the efficiency and complex-learning capability of artificial neural networks (ANNs).*

Key Words: **Two-Wheeled robot; Self-Balance control; Proportional-Integral-Derivative Controller; Kalman Filter; Arduino Uno R3; Digital Motion Processing (DMP); Backpropagation Algorithm; Feedforward Neural Network.**

1. INTRODUCTION AND RELATED WORKS

Over the past decade, robots capable of motion have stepped out of military and industrial avenues, and entered civilian spaces such as hospitals, schools and ordinary homes. although many of these robots for civil applications are somewhat mechanically stable, like 'Aibo' the Sony robotic dog, or four-wheeled intelligent vacuum cleaners, one that every day on-lookers would find awe-inspiring is the Segway personal transport. It is a mechanically unstable, two-wheel self-balancing mode of transport that has seen deployment for law-enforcement, tourism, and personal use. This vehicle can be appropriately called a robot because in the absence of the sensory capabilities and intelligent controls that accompany each robot, the Segway can never stay upright. While the Segway may have been a famous commercial product, research into the control of such a mechanical system has been quite divergent.

Besides the development of Segway, studies of two-wheel self-balancing robots have been widely reported. For example, JOE [1] and nBot [2] are both early versions complete with inertia sensors, motor encoders and on-vehicle microcontrollers. Since then, there has been active research on the control design for such platforms, including classical and linear multivariable control methods nonlinear backstepping controls, PID controller, application of discrete Kalman filters, fuzzy-neural control, Linear-quadratic regulators (LQR) and combinations of the above [3][4]. Many intelligence algorithms have been implemented to tackle the equilibrium problem posed by wheeled inverted pendulums such as fuzzy control [5][6], control schemes based on support vector machines [7], operant conditioning theory [8], etc.

The two-wheeled robot is an amalgamation of the wheeled mobile robot and the inverted pendulum system [9]. It also incorporates the concept of creating a vehicle for humans. The inverted pendulum is a dynamically unstable and non-linear open-loop system with a single-input, multi-output system setting (SIMO) where the center of mass is above the pivot point of the system. This system is therefore a classic problem in dynamics and control theory and is commonly used as a benchmark for testing control system techniques. The wheeled inverted pendulum is not actuated on its own and must be actively balanced in order to remain upright. It uses gyroscopes and accelerometers to detect the inclination of the vertical axis and in order to overcome the inclination, the controller generates torque signals to each motor in order to prevent the system from falling. It is a control system model in which the object can be manipulated only by adding additional load to it. Wheeled inverted pendulums have thus become a novel challenge and implementing balancing algorithms to solve their equilibrium problem attract the interest of many researchers.

Although there are many research-works in this field, only a minority have been directed to make clear comparisons between standard control system techniques such as PID controllers and LQR feedback controllers and the relatively new artificial-intelligence based neural network controllers. We attempt to do just that. Based on the dynamic theory, we will analyze the self-balancing system, establish a mathematical model, and explain the working of the neural

net in the most detailed and lucid fashion. Moreover, a simulation will be performed, which demonstrates the working and efficiency of the same.

2. CONSTRUCTION OF THE ROBOT

1. *CHASIS*: Our frame involves the use of acrylic sheets. We have used metallic rods of diameter 3mm to hold our 3-tiered structure in place. The top portion houses the gyroscope sensor along with the 9V batteries. The middle sheet contains the Arduino Uno R3 microcontroller board and another 9V battery. Finally, the bottom layer contains the driver motor controller and 2 DC geared motors.

2. *MOTORS*: The motors we have used are standard DC geared motors which operate at 12V to give 300 rpm and generate 1.5 kg-cm of torque.

3. *MOTOR CONTROLLER*: We have used the L298N motor controller which is dual H-bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.

4. *GY-521 MODULE*: This measures the angle of the tilt of the robot. A function is used to call the values of the same. The GY- 521 module is a breakout board for the MPU-6050 MEMS (Microelectromechanical system) that features a 3-axis gyroscope, a 3-axis accelerometer, a digital motion processor (DMP), and a temperature sensor. The digital motion processor can be used to process complex algorithms directly on the board. Usually, the DMP processes algorithms that turn the raw values from the sensors into stable position data. The sensor values are retrieved by using the I2C serial data bus, which requires only two wires (SCL and SDA). Here, we used the MPU6050 Arduino library which consists of reverse-engineered functions to utilize the DMP present on-board the MPU-6050 to calculate yaw, pitch and roll.

5. *ARDUINO UNO R3*: Arduino Uno R3 is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button.

6. JUMPER WIRES: Male to male and female to male wires.

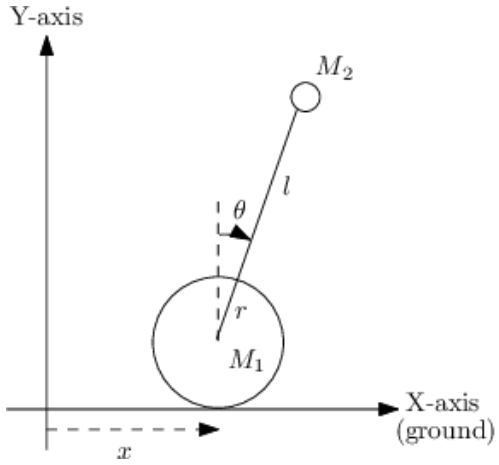

Fig. 1: Schematic model for a wheeled inverted pendulum

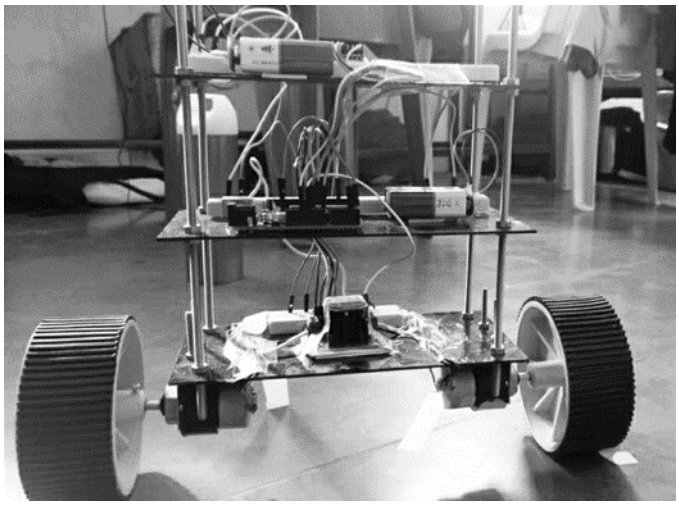

Fig. 2: Front view of constructed robot

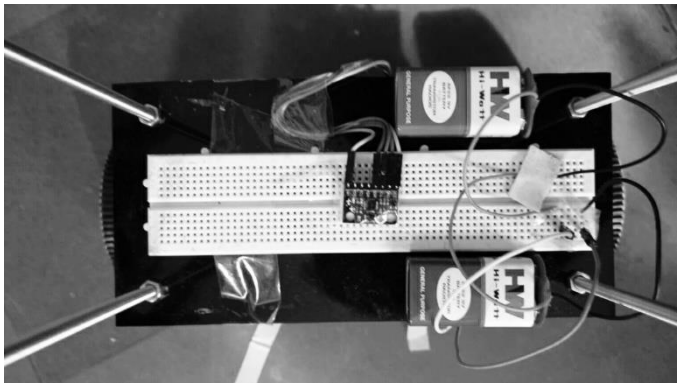

Fig. 3: Top view of constructed robot

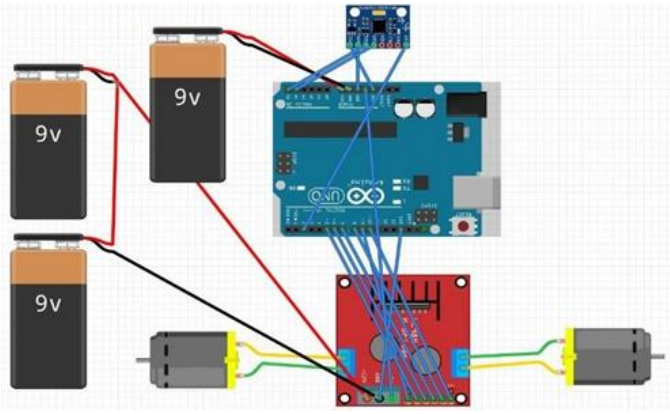

Fig 4: Circuit Schematic

3. ALGORITHMS USED

3.1 PID CONTROLLER APPROACH

We explore two approaches to maintaining the upright nature of the robot. The first method is using a PID controller. The Proportional Integral Derivative controller (PID) is a control loop feedback mechanism which is now widely used in industrial control systems and a plethora of other applications that require continuously modulated control [10]. The PID controller continuously computes an error value as the disparity between the desired setpoint (SP) and the measured process variable (PV) and initiates a correction on the basis of proportional, integral, and derivative terms, denoted by P, I, and D respectively which gives the controller its name. Technically, it maps accurate and responsive correction to a control function. PID comprises of three parts: (1). The proportional part is used to directly control the magnitude of the response. If the value of Kp is too high or too low, the system becomes unstable. (2). The integral part is used to average past error and accelerate the movement of the process towards the setpoint. (3). The derivative part is to predict the error in the future through previous error variations. The final control value is calculated by simply adding these three terms together. The formula is as follows:

$$u=k_p e(t)+k_i \int e(t)dt+k_d \frac{de(t)}{dt} \quad (1)$$

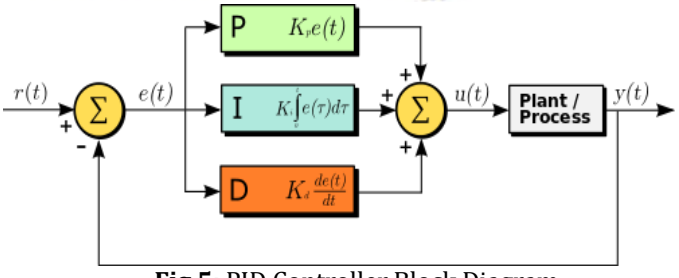

Fig 5: PID Controller Block Diagram

Algorithm to adjust K_p, K_i and K_d values: -

- After setting the I and D terms to 0, we adjust the value of P so that the robot begins to oscillate about the mean position. P should be big enough for the robot to move but not too large which would hinder smooth movement.
- With P appropriately identified, I is increased so that the robot accelerates faster when off balance. With P and I properly adjusted, the robot should be able to balance itself for at least a few seconds.
- Next, the value of D is decided so that the robot will be able to move about its balanced position in a gentler fashion. Also, there should not be any significant overshoots.
- If the first attempt does not lead to satisfying results, one should reset the PID values and repeat the process above until you arrive at a satisfactory result.
- While fine tuning, the PID values are confined to neighboring values and its effects are observed in practical situations

3.2 FEEDFORWARD NEURAL NET APPROACH

In our proposed approach, we employ a feedforward neural network in order to control a two wheeled self-balancing robot. The robot does not do anything additional to self-balancing, there exists no method for directional control. Keeping in mind the Arduino Uno's 2K SRAM constraint, the program uses a neural net with a single input node, two hidden nodes and a single output node. The program first uses a training routine which takes only a few seconds to finish so it can be run every time the robot is started. Here, the bot needs to be held upright while the neural network routine gets trained using the backpropagation algorithm. The input sensor values (tilt angle of the robot) is mapped to a value between 0 and 1. This value is sent to the neural network model which then returns an output from the sigmoid activation layer which is between 0 and 1. Finally, we map these output values to a useful value for the DC motors, between -255 to 255 in this case.

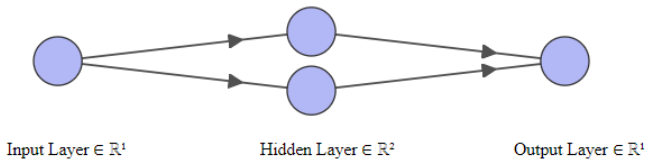

Fig 6: Feedforward Neural Network Architecture

4. METHODOLOGY

In order to obtain information regarding the spatial orientation of the robot, we make use of the Kalman filter, which is also known as the Linear Quadratic Estimation (LQR) algorithm in conjugation with the Complementary filter in order to reverse-engineer the DMP (Digital Motion Processing) algorithm of the GY-521 Breakout Board which uses the InvenSense MPU-6050 6-Axis Accelerometer and

Gyroscope Module. The DMP functions are called and used in order to get precise and accurate values for the Yaw, Pitch and Roll of our mechanically unstable system.

The Pitch value is then used in order to understand and calculate the vertical tilt angle of the robot so that the robot can then rectify its orientation. This is done by providing Pulse-Width Modulated Signals (PWM) in order to make the motors run slower or faster, depending on the tilt angle. The PWM signals can be sent within a range of -255 to 255 and thus a PID controller is used in order to compute the value to be sent as PWM. Now, in order to calculate the PWM value we need to give an input to the PID controller and a Setpoint or that it can calculate the error (Setpoint – Input) and then give an output which will ultimately reduce the error. Here we provide the tilt angle of robot when held at vertical as the input. The input tilt angle keeps changing depending on the surface on which the robot is kept and thus we need to change the value in the code.
However, in our neural net application none of these parameters need to be set. We simply use the back-propagation algorithm and incorporate the neural net with a single input node, two hidden nodes and a single output node.

Instead of tinkering with all the values of K_p, K_i and K_d required for the PID Controller, we simply call a training routine during initialization or the robot during which the bot needs to be held upright and steady. The bot feeds this vertical tilt angle as the input angle to the training set and computes the error.

In this case, we are using the Sigmoid Activation Function for the neural net. The graph for the function is as follows-

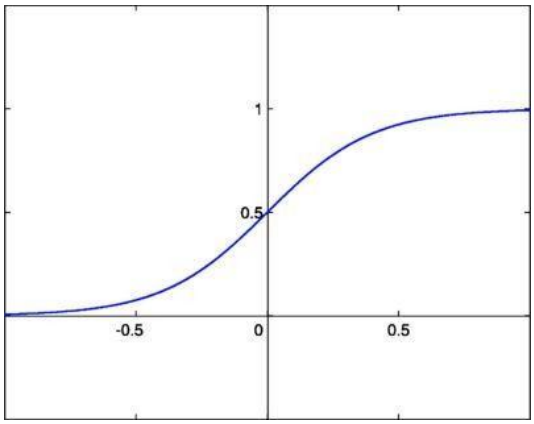

Fig. 7: Sigmoid Activation Function

The hyperparameters set for the neural net are as follows-

Parameter	Value
Learning Rate	0.3
Momentum	0.9
Initial Weights	0.5
Success	0.0015
Sample Time (mS)	0.005
Loop Timer (mS)	4
Input Node	1
Hidden Nodes	3
Output Node	1

Table 1: Optimized Hyperparameter values of FNN

The parameter 'Success' has been given a value of 0.0015. The error calculated as the difference in the required Set Point (Vertical tilt angle) and the Current Angle (Real-Time Tilt Angle) is supposed to be less than the value of success in order to keep the robot upright and the neural net provides the output required in order to do that. This output is then linearly mapped onto the range of -255 to 255. This value is then used as the magnitude of the PWM signal.

So greater the value given to the PWM signal, more is the torque provided by the motor and thus the bot is able to maintain its upright position.

5. RESULTS AND DISCUSSION

We employed both approaches on our constructed robot, in order to compare and infer as to which algorithm worked better.

The first approach was the PID Controller approach where the following steps were taken-

- The desired setpoint was set as 6.10 (degrees from the vertical).
- The value of Kp after extensive experimenting, was set to 90 as it was optimal and made the bot oscillate about the balance position.
- The value of Ki was set to 250 as it made the balancing more efficient and provided more torque to the motors at higher tilt angles.
- The value of Kd was set at 2.15 since any value more than that made the robot jitter and any value less than that made the robot fall.

The next approach was the Neural Net approach where the following steps were taken-

- The desired setpoint was set as 6.10 (degrees from the vertical).
- The value of Learning Rate was tinkered with, and it was found that a value of 0.3 made the bot balance perfectly without oscillating at high frequencies and causing large deviations.
- A high learning rate made the output change frequently and this caused the motors to frequently change speed, thus making the robot oscillate and jitter.
- The momentum and initial weights were set as 0.9 and 0.5 respectively based on hit and trial.

- The Success value was kept as 0.0015 since all values higher or lower to that caused a significant change in the error calculation and caused the DMP function to be called frequently, thus making changes to the output more than required making the robot jitter a lot.

Furthermore, we used the serial graph plotter present in the Arduino IDE Software in order to visualize and analyze the motor control signals under wheel synchronization in Fig. 8. It can be clearly seen that the wheel synchronization is effective.

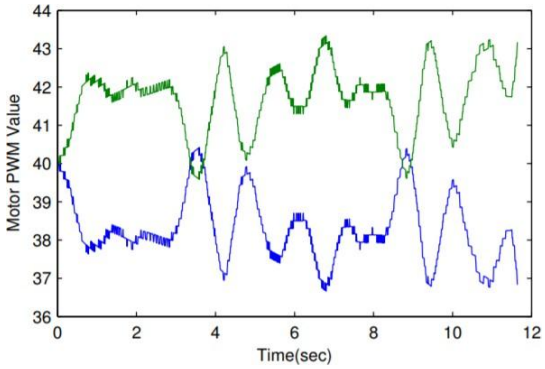

Fig. 8: PWM Motor Control Signals

The Arduino IDE Software's Serial Graph was also used in order to plot a graph between the tilt angle and time passed. These graphs were plotted for both the approaches and it can be seen in Fig. 9 that stability with PID control is marginal. Meanwhile, much more improved stability is obtained using the Neural Net and the tilt angle deviates way less using the Neural net.

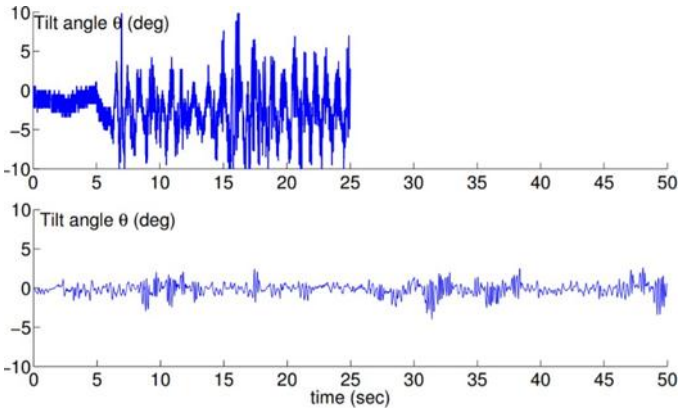

Fig. 9: Experimentally obtained history of tilt angle: PID controller (top), Neural Net (Bottom)

6. CONCLUSION

Neural Net approaches for self-balancing are not as widely used as PID controllers, however based on our extensive research we can conclude that the Feedforward Neural Network was able to balance the robot much better than the PID Controller. As seen from the results, the PID Controller requires extensive tuning of the K_p, K_i and K_d values. These values are specific to a system and are different for every system. Thus, the same values cannot be used for another robot. However, the parameters set in the neural net are universal and thus this approach does not require any tuning whatsoever. The neural net code simply needs to be uploaded on to the Arduino Uno Microcontroller and the robot starts balancing itself. In conclusion, the Neural Net approach is an interesting and reliable way to balance the robot since it makes the bot oscillate less around the balance point and thus keeps the bot steady for a longer period of time. Meanwhile, the PID controller works better when it is subjected to manual disturbances/imbalances. When the bot is provided with a slight push and caused to imbalance, the PID controller is able to provide the higher torque required and balances the bot but the Neural Net often fails to do so, causing the robot to fall. In conclusion, we have successfully constructed a compact and cost-effective two-wheeled self-balancing robot using low-cost components. The self-balancing challenge was tackled using two different approaches, a fine-tuned PID controller and a 3-layered feedforward neural network trained using backpropagation algorithm and suitable comparisons were drawn between them.

REFERENCES

[1] F. Grasser, A. D. Arrigo, and S. Colombi, "JOE: A mobile, inverted pendulum," IEEE Trans. Ind. Electron., vol. 49, no. 1, pp. 107–114, Feb. 2002.

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

[3] Nguyen Gia Minh Thao, Duong Hoai Nghia and Nguyen Huu Phuc, "A PID backstepping controller for two-wheeled self-balancing robot," International Forum on Strategic Technology 2010, Ulsan, 2010, pp. 76-81, doi: 10.1109/IFOST.2010.5668001.

[4] W. An and Y. Li, "Simulation and control of a two-wheeled self-balancing robot," 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), Shenzhen, 2013, pp. 456-461, doi: 10.1109/ROBIO.2013.6739501.

[5] Q. Yong, L Yanlong, Z Xizhe, and L. Ji, "Balance control of two-wheeled self-balancing mobile robot based on TS fuzzy model," 2011 6th International Forum on Strategic Technology (IFOST), pp.406,409, Aug 22–24, 2011.

[6] J. Wu, S. Jia, "T-S adaptive neural network fuzzy control applied in two-wheeled self-balancing robot," 2011 6th International Forum on Strategic Technology (IFOST), pp.1023, 1026, Aug 22–24, 2011.

[7] L. Jiang, M. Deng, and A. Inoue, "Support vector machine-based two wheeled mobile robot motion control in noisy environment", J. of Systems and Control Engineering, vol. 222, no. 7, pp. 733–743, 2008.

[8] J. Cai and X. Ruan, "Bionic autonomous learning control of a two-wheeled self-balancing flexible robot," J. of Control Theory and Applications, vol. 9, no. 4, pp.521–528, 2011..

[9] Kim, Y., Kim, S.H. & Kwak, Y.K. Dynamic Analysis of a Nonholonomic Two-Wheeled Inverted Pendulum Robot. *J Intell Robot Syst* **44,** 25–46 (2005). https://doi.org/10.1007/s10846-005-9022-4

[10] S. Cong and Y. Liang, "PID-like neural network nonlinear adaptive control for uncertain multivariable motion control system," IEEE Trans. Ind. Electron., vol. 56, no. 10, pp. 3872–3879, Oct. 2009.