

Design and Control of Two-wheeled Self-Balancing Robot using Arduino

Anmol Singh Shekhawat
Electrical and Electronics Engineering
IET, JK Lakshmipat University
Jaipur, India
anmolshkhawat@jklu.edu.in

Yogesh Rohilla
Electrical and Electronics Engineering
IET, JK Lakshmipat University
Jaipur, India
0000000276332226

Abstract—This paper discusses the design, construction, and control of a two-wheel study level, small self-balancing robot. Two-wheel robots are compact and require less number of motors and other equipment in comparison to conventional four-wheel robots. Consequently, they offer a complex control system problem of balancing on two-wheel. The system architecture comprises of Arduino Nano microcontroller board, DC stepper motor, gyroscope, and accelerometer sensor. PID controller controls it through the program written in Arduino IDE. A detailed description of constituting materials in making the robot has been provided. Flow chart and closed loop control diagrams have been given for understanding the controlling of the robot. A step-by-step process of connecting the hardware components has been listed, and finally, hardware assembly is shown.

Keywords—MPU-6050, stepper motor, PID controller, Arduino Nano, self-balancing robot

I. INTRODUCTION

In today's world, we get to see Segway or Hoverboard in movies, TV shows, YouTube videos and in similar video platforms. These devices work on the same principle that of a self-balancing robot. A self-balancing robot is a robot, balances itself on two wheels, by constantly correcting its position.

In the '80s, a Japanese Professor, Kazuo Yamafuji, built the first model to simulate the behaviour of an inverted pendulum [1]. Since then, many different prototypes have been built by many researchers. The widespread accessibility of economical electronic components has made it a fascinating project for makers and students.

A self-balancing robot is an inverted pendulum. In the case of an ordinary pendulum, it just swings around a point (centre) as it has a centre of gravity (COG) below the centre point which is the centre of the circle. When it comes to a self-balancing robot, it has COG above the centre point, which makes it unstable. By balancing this unstable inverted pendulum, a self-balancing robot is made. By minimising the amplitude of this inverted pendulum, a very stable self-balancing robot can be made which can stand still.

Inverted pendulum based devices can be used in many applications, such as automated transportation. Following are the applications of a self-balancing robot:

1. Segway: Segway is a two-wheeled, personalised and simple self-balancing robot. It can handle the weight of single person suitably and used for short distance outdoor transportation, indoor transportation, in tourism, and medical purposes [2].
2. Rockets and missiles: Rocket and missiles are also lifted on the principle of an inverted pendulum. They are

designed to reach a specific target and to do so; they must maintain a certain angle which is like a self-balancing robot [3].

3. Hoverboard: Hoverboards are almost the same as Segway, but they don't have a handle and have small wheels. They are getting quite popular in recent years [4].
4. Toys: Many semi-autonomous toys are available in the market, which can react to gestures, voice commands, remote controller and even smartphone. These have many inbuilt playing modes which are safe and exciting to play [5].

This paper provides the design, construction, and control of a study-level, two-wheeled, small self-balancing robot. Many papers have provided the design and procedure to make the self-balancing robot. Some of them used MATLAB for simulation [6, 7]; some used fuzzy algorithm for controlling it [8, 9]; some used complex mathematical modelling approach [10]; and some used control system concepts for controlling the robot [11-13]. Fig. 1 shows the block diagram of the proposed self-balancing robot. Here Arduino Nano microcontroller board with MPU-6050 (gyroscope + accelerometer), stepper motors, and proportional-integral-derivative (PID) controller has been used to make the self-balancing robot.

The things which make this work unique than the existing literature are the detailed description of components and step by step explanation of connections. A detailed description of components with the technical specification has been provided which help any other interested person in making the robot. Similarly, working and construction are given in a step by step manner for easy understanding.

The paper arrangement is as follows: after the introduction section, the paper provides detailed descriptions of the components in section II. Sections III discusses the robot's

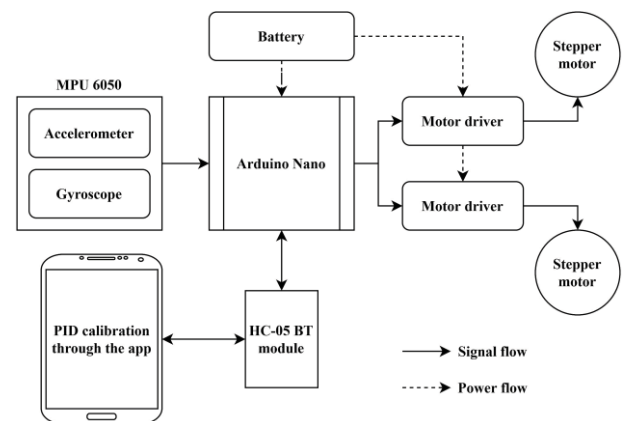


Fig. 1. Block diagram of the self-balancing robot.



Fig. 2. A default Arduino IDE window.

working and balancing control. Section IV lists the step by step process of hardware connections. Section V concludes the paper with the results.

II. CONSTITUTING MATERIALS

Software and hardware components have been used in making the proposed robot. Only one software is used which is Arduino IDE, and different hardware is used such as Arduino Nano as the brain, stepper motor to provide the motion, MPU-6050 to get the orientations, and HC05 as communicating device. Explanation of these constituting materials are as follows:

A. Arduino IDE

Arduino IDE is simple to use software yet the best one to code, compile and upload the code in Arduino microcontroller boards. It is developed by the Arduino team [14]. One can code on other platforms too, such as visual code studio, but Arduino IDE is official and accepted worldwide. It is the first one to receive all the updates of the libraries and the boards. Many libraries are available which make it convenient to code in the Arduino IDE. A default window of the Arduino IDE is shown in Fig. 2.

B. Arduino Nano

The Arduino Nano is used in this paper as the brain of the robot. This microcontroller is the smallest member of the Arduino family. It is based on the ATmega328 microcontroller, which is capable for several projects. It is a 30 pin breadboard-friendly board with 22 I/O pins. It can work as same as Arduino UNO in almost all projects, but due to its small size, it is the perfect choice to work with. The main difference between UNO and Nano is very minute. Nano has a different architecture than UNO and has 8 analog pins, whereas UNO has only 6 analog pins. It works with a Mini-B USB cable instead of a power jack. An Arduino Nano is shown in Fig. 3. Pin description of Arduino Nano is provided in TABLE I.

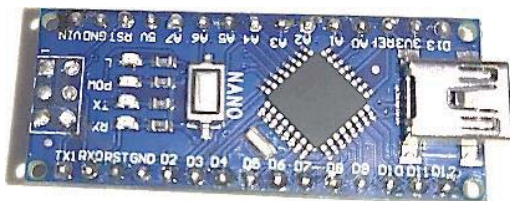


Fig. 3. The Arduino Nano.

TABLE I. PIN DESCRIPTION OF ARDUINO NANO

Type	Pins	Description
Power	Vin, 3.3 V, 5 V, GND	Vin is used for giving supply to the Arduino board, which can vary from 6 to 12 V. 5 V and 3.3 V give output voltages as stated. GND refers to the ground pin.
Reset	RST	This is reset pin in Arduino which resets the Arduino.
Analog	A0-A7	These are analog input pins. Arduino has an inbuilt ADC (analog to digital converter) which helps to measure the input voltage from 0 to 5V. Arduino doesn't have a DAC (digital to analog converter) but it can do PWM (pulse width modulation) which are locate at in digital pins and can be used to give some of the functions of an analog output as well. A4(SDA), A5(SCL): These pins are used for TWI (Two-Wire Interface) communication. (SCL: serial clock line, SDA: serial data)
Digital Pins	D0-D13	These are digital input-output pins These pins give logic high (5V) or logic low (0V). They can also be used as input logic high and logic low. 3,5,6,9,11: These are some 8-bit PWM output present on the board. 13: Pin 13 is for inbuilt LED.
Serial	Rx, Tx	Rx-Receiver Tx-Transmitter These are used for serial communication.
SPI	10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK)	Used for SPI communication. SS, MOSI, MISO, and SCK stand for Slave Select, Master out Slave in, Master in Slave out, and Serial clock.
AREF	AREF	To provide reference input voltage.

C. MPU-6050

The MPU-6050 is a motion-tracking device. Here, this sensor gets the alignment of the bot, which is further processed to take the control actions. The MPU-6050, as shown in Fig. 4, constitutes a Micro-Electro-Mechanical Systems (MEMS) 3-axis accelerometer and a MEMS 3-axis gyroscope on a single chip. It also has a temperature sensor inbuilt on it. MPU-6050 delivers accurate results and can provide hundreds of measurements per second. It captures the x, y, and z orientation at the same time. It uses the I2C-bus to interact with the Arduino. Pin diagram and pin description of MPU-6050 is given in Fig. 5 and TABLE II, respectively.

D. HC-05

HC-05 is the Bluetooth module used to get the PID values from the Android smartphone for calibration. It is versatile, reliable and easy to work. HC-05 Bluetooth module has 6 pins, and pin description is given in TABLE III.

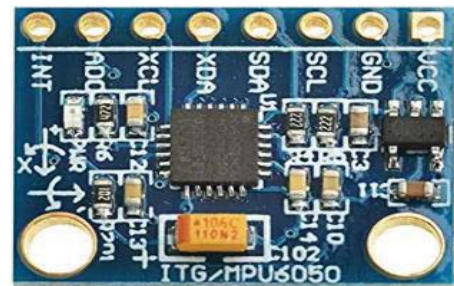


Fig. 4. The MPU-6050.

TABLE II. PIN DESCRIPTION OF MPU-6050.

Type	Pins	Description
Power	VCC, GND	VCC is 5 V input. GND is for grounding.
Communication	SDA (Serial Data), SCL (Serial Clock)	SDA transfers data and SCL provides clock pulse for communication.
Auxiliary Communication	XDA (Auxiliary Serial Data), XCL (Auxiliary Serial Clock)	Can be used to interface other modules as Arduino have limited SDA and XCL ports.
Other	AD0, INT	AD0 (Slave address at 0 th bit, i.e. this is 0 th number bit of a 7-bit slave address of the module) is used when more than 1 MPU 6050 modules are used to communicate in synchronous. INT is interrupt pin to indicate that data is available for MCU to read.

TABLE III. PIN DESCRIPTION OF HC-05 BLUETOOTH MODULE.

Type	Pins	Description
Power	VCC, GND	VCC is 5 V input. GND is for grounding.
Key	EN (Enable)	It is used to toggle between data mode and AT (Attention) command mode. Set it high to get into AT command mode.
Serial	RX, TX	RX is the receiver and TX is the transmitter. Used for serial communication.
State	State	This pin is connected to the inbuilt LED of the module and is used as a feedback to check the working of the module by the microcontroller.

E. Stepper motor

Stepper motors work as the legs of the bot providing movement to it. Stepper motor provides accurate movements making it much stable than traditional geared motors used in projects. Stepper motor works in steps. They have multiple coils groups forming pairs when they rotate. By supplying current to different pairs, the motor moves one step at a time. By providing a synchronised supply to consecutive coils, precise movements can be achieved. When controlling through a microcontroller, stepping can be excellent means in achieving precise positioning and speed control. The stepper motor was chosen in this paper, and its technical specifications are given in Fig. 6 and TABLE IV, respectively.

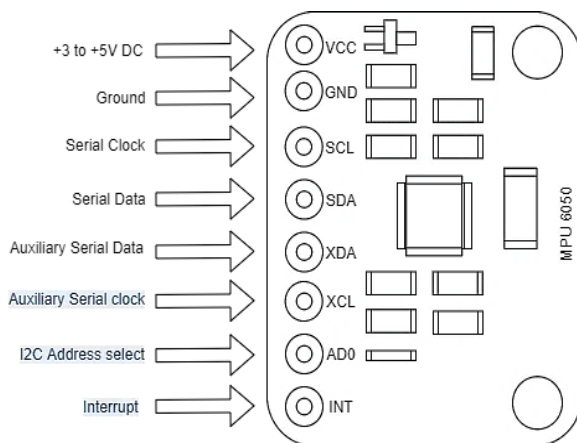


Fig. 5. Pin arrangement of MPU-6050.

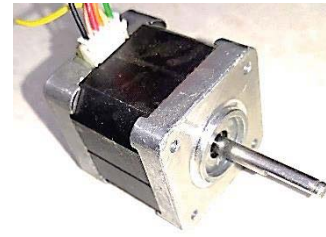


Fig. 6. The stepper motor.

As stated earlier, the stepper motor provides precision movement. It can be as precise as 0.9 degrees per half step. It has enough torque to stabilise to the bot. Conventional DC motor achieve required torque by using gears which make them slightly unsuitable for this project. DC motor has some unnecessary motion due to gaps in gears making the bot unstable and also their reaction time is high. Due to these reasons, the stepper motor is preferred over DC motor.

F. Stepper Motor Driver (A4988)

This is a very versatile driver which can be operated up to 35 V with around $\pm 2A$. This driver contains the allegro A4988 chip which works as a translator, i.e. there is no need to provide step signals instead just need to provide step size and directions. The A4988 stepper motor driver module and its pinout diagram are shown in Fig. 7 (a) and (b). This driver is capable of providing rotation to stepper motor from a full step to as low as one by sixteenth of a step. It contains a reset (*RST*) and a sleep pin (*SLP*) which need to be shortened to start the motor. To operate the motor in full-step, there is no need to connect any pin to *MS1* (micro-step), *MS2*, and *MS3*. Fractional step rotation can be provided using pins *MS1*, *MS2*, and *MS3* as per TABLE V.

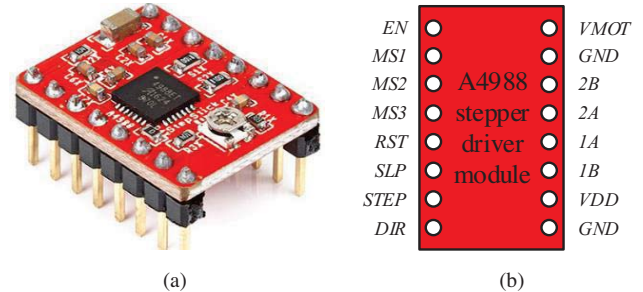


Fig. 7. (a) A4988 Stepper motor driver module, and (b) its pinout diagram.

G. Miscellaneous

Other than the components mentioned above, some general things which used in the making of this robot are acrylic sheet, screws, and connecting wires. Connecting wires are required to connect all the hardware to supply power; acrylic sheets to make the base for all the components; and long M5 screws to join multiple levels of the bot on acrylic sheets.

TABLE V. ROTATION SEQUENCE INPUT IN STEPPER MOTOR DRIVE

Resolution	MS1	MS2	MS3
Full Step	0	0	0
Half Step	1	0	0
Quarter step	0	1	0
Eight step	1	1	0
Sixteenth step	1	1	1

III. THE SELF-BALANCING ROBOT

Block diagram of the self-balancing robot is shown in Fig. 1. The robot gets the PID values from the mobile phone using the HC-05 Bluetooth module. Values can be put manually in the phone using an application. The bot is then calibrated by placing it on the horizontal surface.

After these preliminary steps, the bot comes to action. It gets the acceleration values and gyro values, i.e. the angles. The microcontroller processes this data applying PID to get the right amount of movement given to the motors. Based on the data received by the controller from MPU-6050, the controller commands the motor to run forward or reverse to maintain the position of the robot. The robot should make 0° angle with the y-axis, Fig. 8 (a). When bot body tilts in a direction, clockwise or anti-clockwise, then there is a change in the angle between y-axis and body, Fig. 8 (b) and (c). The MPU-6050 gyro sensor detects this angle, and the microcontroller initiates actions to maintain it to the desired angle. The bot can do this whole cycle for around 400 times in second to maintain a stable position at its place.

As stated earlier, to put the PID values in the bot, a mobile application is needed. Similarly, to make the Arduino function, it needed to code. These three things, i.e., mobile application, PID values, and microcontroller programming, are important and discussed in detail here.

A. PID controller

The closed loop control system of the self-balancing robot is shown in Fig. 9. The output shows the actual position of the robot. The MPU-6050 sensor senses the actual position of the robot. Sensor feedbacks the data to compare with the reference input. Reference input or the aim of the robot is to stay in the 90° vertical position with reference to ground, as shown in Fig. 8 (a). The comparison of output and reference input provides the tilt angle. This tilt is the error which needs to be reduced to zero. It requires a fast controller to handle this task of error reduction. PID controller has been used for this purpose. PID controller sends the command to the motor driver, and in result, it controls the motion of the stepper motors precisely. PID controller is also shown in Fig. 9, and mathematically can be written as:

$$u(t) = K_p e(t) + K_i \int e(t) dt + K_d \frac{d}{dt} e(t) \quad (1)$$

where $u(t)$ is the response of the controller. K_p , K_i , and K_d are the proportional constant, integral constant, and derivative constant respectively, which are corresponds to proportional, integral, and derivative controllers,

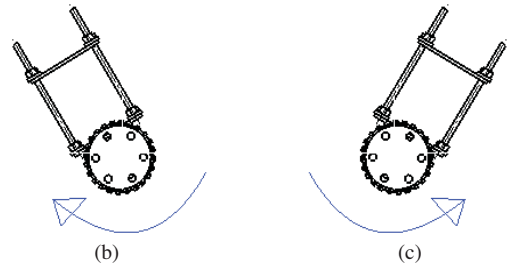
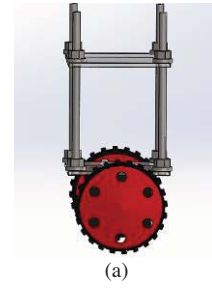


Fig. 8. Self-balancing positions of robot. (a) Mean position of robot, (b) correction of the robot when tilted at an angle in anticlockwise side, and (c) correction of the robot when tilted at an angle in clockwise side.

respectively. $e(t)$ is the error signal which shows the difference between y-axis and the actual position of the bot.

The angle set points where the bot has to maintain its default position and PID values are fed to the Arduino using the mobile application through the Bluetooth device. K_p is the proportional term, which multiplies with the error and speeds-up or slows-down the response. Any change in K_p makes the bot to balance at another set-point. K_i is the integrating parameter. This element reduces the steady-state error and smoothens the motion of the bot. Being an integrating factor, even a small change affects the bot very much. K_d is the derivative term. It increases the reaction time of the bot. Increasing it too much may make the bot unstable. The values of PID is set with the help of Ziegler–Nichols tuning method [15].

B. Microcontroller Programming

To program the microcontroller, an open-source platform, known as balancingwii, was used [16]. This firmware is based on multiwii firmware which is also an open-source firmware but used for aerial projects, such as quadcopter. It had inbuilt compatibility to use the mobile application. A flowchart explaining the function of microcontroller programming is shown in Fig. 10. It can be seen in the initiation of the flowchart; first Arduino verifies whether I2C communication has been established or not (this means that

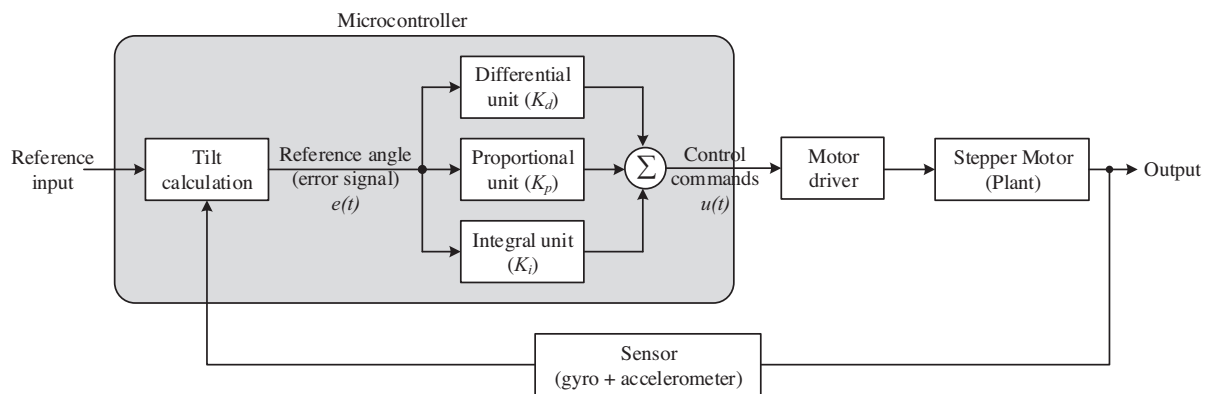


Fig. 9. The closed loop control system of the self-balancing robot.

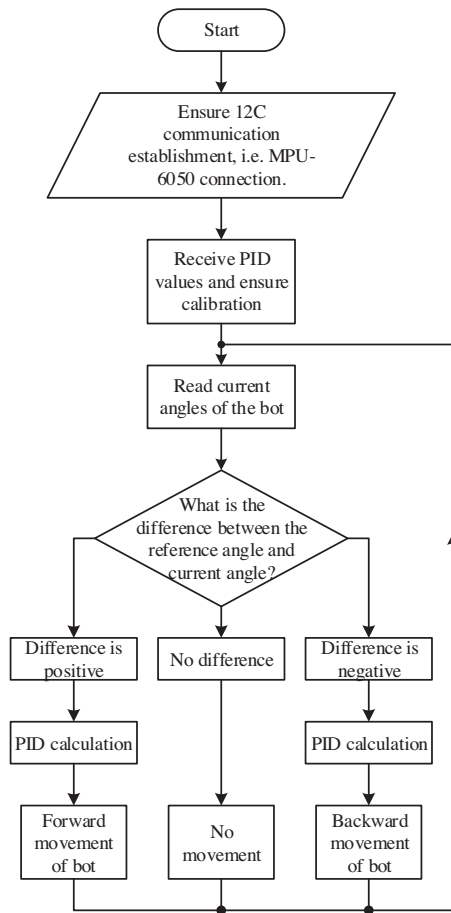


Fig. 10. Flowchart of microcontroller programming.

it check for the availability of the gyro sensor, i.e. MPU-6050 is connected or not). After that, set-point is calibrated, and PID values are received through the Bluetooth module HC-05. After receiving all the values, the bot comes to action by taking the difference between the set-point (or reference point) and the current angles. Then calculations are performed, and specific acceleration value and direction are

provided to the motors, accordingly. After repeating this process several times, the bot tries to be stable after some to and fro motions.

C. Mobile application

An Android platform-based mobile application EZ-GUI has been used to feed PID values to the microcontroller through the Bluetooth module. The mobile application removes the need for a laptop or computer system again and again in sending and receiving the values, set calibration point, and changing modes. This app was initially developed for Cleanflight, Betaflight, iNav, and Multiwii, which are the flight systems. It allowed the user to quickly change the PID values, which were very hectic when using a computer or laptop to do such things. The best part of using this app was that it could share the required information through many communication means, such as Bluetooth, WiFi, radio transmission, and USB. Any other compatible application can also be used.

IV. HARDWARE IMPLEMENTATION

This section discusses the hardware connections of the self-balancing robot. A connection diagram is shown in Fig. 11. The step by step process of connections are as follows:

- Step 1. Place Arduino Nano, MPU-6050, two stepper motors and their A4988 drivers, HC-05 Bluetooth module, LD11173 voltage regulator, 100 nF ceramic capacitor, 10 μ F electrolytic capacitor, and battery as shown in Fig. 11.
- Step 2. Make common positive bus (bus +ve) and connect Arduino V_{in} to it. Similarly, make common ground bus (bus GND) and connect Arduino GND to it.
- Step 3. Connect VCC of MPU-6050 to positive of 10 μ F capacitor and V_{out} of the voltage regulator.
- Step 4. Connect MPU-6050 GND to ground bus, its SCL to Arduino A5, and its SCA to Arduino A4.
- Step 5. Connect first motor driver OUT2A, OUT2B, OUT1A, and OUT1B to first stepper motor D, B, C, and A respectively.

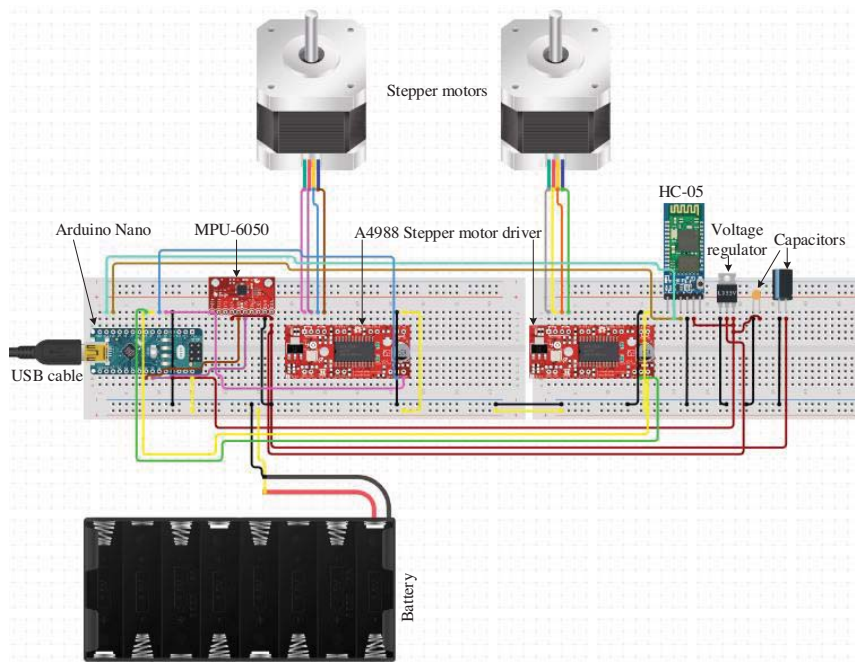


Fig. 11. Connection diagram for the self-balancing robot.

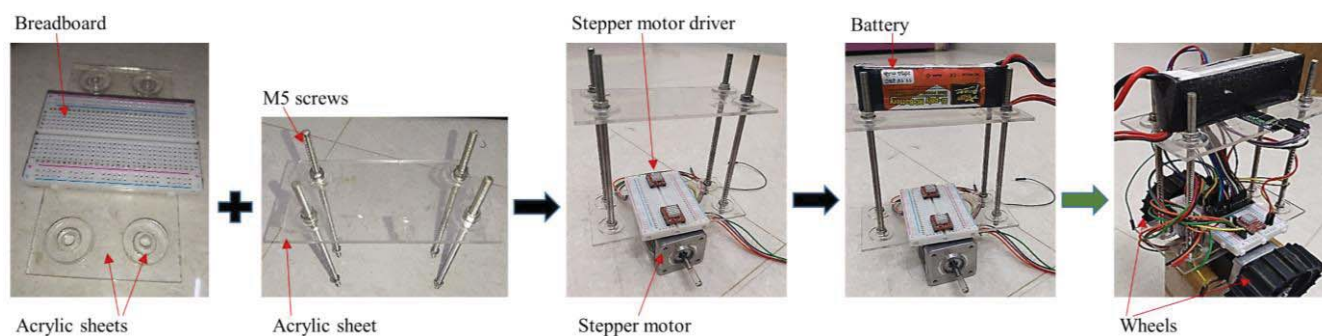


Fig. 12. Hardware assembly of the self-balancing robot.

- Step 6. Connect first motor driver *DIR* to Arduino *D2*, its *GND* to bus *GND*, *STEP* to Arduino *D3*, and *V* + to bus +*ve*.
- Step 7. Connect second motor driver *OUT2A*, *OUT2B*, *OUT1A*, and *OUT1B* to second stepper motor *D*, *B*, *C*, and *A* respectively. Connect its *DIR* to Arduino *D4*, its *GND* to bus *GND*, *STEP* to Arduino *D3*, and *V* + to bus +*ve*.
- Step 8. Connect HC-05 *GND* to bus *GND*, *TX* to Arduino *D10*, and *RX* to Arduino *D11*.
- Step 9. Connect voltage regulator *V_{in}* to Arduino 5 *V*, HC-05 *VCC*, and 100 *nF con0*. Connect its 0 to bus *GND*.
- Step 10. Connect 100 *nF con1* and 10 μF *neg* to bus *GND*.
- Step 11. Connect Arduino Nano to the computer using a USB cable.
- Step 12. Connect battery +*ve* to bus +*ve* and battery - *ve* to bus *GND*.
- Step 13. Upload code in Arduino Nano using the USB power mini. It will test all the wiring.
- Step 14. Test the MPU-6050, stepper motors, and HC-05. Send PID values through the mobile application and test again.

Hardware assembly of all the components is shown in Fig. 12. It contains a two-level partition. The lower one contains the motor driver, Arduino Nano, and motors attached at the lower side. The top one contains the battery, HC-05 Bluetooth module, and the MPU-6050 gyro sensor. The reason to place the battery at the top of the arrangement is to keep the center of gravity above the horizon of motors.

The partitions are made out of the laser cut acrylic sheets. These sheets are held together by the long M5 screws, nuts, and bolts. All the modules are attached using double-sided glue tapes.

V. RESULT AND CONCLUSION

After some testing and calibrating, the two-wheeled robot balanced itself independently. It required appropriate PID values fed to the microcontroller and accurate measurement from the gyro sensor. Using a CAD designed, laser-cut and 3D printed structure, the bot stood straight with good stability.

There are many possibilities to extend this work. In place of a breadboard, a printed PCB can be used, which would make it look more aesthetic. By adding more sensor and features, this bot can be used in many places and can be a great learning tool. For instance, standard line following robot, obstacle avoiding robot, and maze solving robot can be made using this self-balancing robot. These applications

would make the learning process more exciting and mainly a differentiating factor from those custom robots which uses four wheels in these tasks.

REFERENCES

- [1] K. Yamafuji and A. Koshiyama, "Postural and Driving Control of a Variable-Configuration-Type Parallel Bicycle," *J. Robotics Mechatronics*, vol. 3, no. 5, pp. 365-372, 1991.
- [2] D. L. Kamen *et al.*, "Personal mobility vehicles and methods," 2001.
- [3] N. J. Mathew, K. K. Rao, and N. Sivakumaran, "Swing up and stabilization control of a rotary inverted pendulum," *IFAC Proceedings Volumes*, vol. 46, no. 32, pp. 654-659, 2013.
- [4] A. Mehrvarz, M. J. Khodaei, W. Clark, and N. Jalili, "Modeling and dynamics analysis of a beam-hoverboard self-transportation system," in *Dynamic Systems and Control Conference*, 2018, vol. 51913, p. V003T32A008: American Society of Mechanical Engineers.
- [5] (2020). *WowWee®- MiP*. Available: <https://wowwee.com/mip>
- [6] W. An and Y. Li, "Simulation and Control of a Two-wheeled Self-balancing Robot," in *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2013, pp. 456-461: IEEE.
- [7] V. Kongratana, S. Gulphanich, V. Tipsuwanporn, and P. Huantham, "Servo state feedback control of the self balancing robot using matlab," in *2012 12th International Conference on Control, Automation and Systems*, 2012, pp. 414-417: IEEE.
- [8] Q. Yong, L. Yanlong, Z. Xizhe, and L. Ji, "Balance control of two-wheeled self-balancing mobile robot based on TS fuzzy model," in *Proceedings of 2011 6th International Forum on Strategic Technology*, 2011, vol. 1, pp. 406-409: IEEE.
- [9] J. Wu and W. Zhang, "Design of fuzzy logic controller for two-wheeled self-balancing robot," in *Proceedings of 2011 6th international forum on strategic technology*, 2011, vol. 2, pp. 1266-1270: IEEE.
- [10] N. Razmjoooy and M. Ramezani, "Optimal control of two-wheeled self-balancing robot with interval uncertainties using Chebyshev inclusion method," *Majlesi Journal of Electrical Engineering*, vol. 12, no. 1, pp. 13-21, 2018.
- [11] R. S. Martins and F. Nunes, "Control system for a self-balancing robot," in *2017 4th Experiment@ International Conference (exp. at'17)*, 2017, pp. 297-302: IEEE.
- [12] S. Yuan, G. Lei, and X. Bin, "Dynamic modeling and sliding mode controller design of a two-wheeled self-balancing robot," in *2016 IEEE International Conference on Mechatronics and Automation*, 2016, pp. 2437-2442: IEEE.
- [13] J. Qiu *et al.*, "Two-wheeled self-balancing robot modeling and nonlinear control," in *2017 14th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2017, pp. 734-739: IEEE.
- [14] (2020). *Arduino - Software*. Available: <https://www.arduino.cc/en/Main/Software>
- [15] J. G. Ziegler and N. B. Nichols, "Optimum settings for automatic controllers," *trans. ASME*, vol. 64, no. 11, 1942.
- [16] Mahowik. (2020). *BalancingWii*. Available: <https://github.com/mahowik/BalancingWii>