Self- balacning robot with Bluetooth remote control

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Electrical and Automation Engineering

Valkeakoski, Autumn 2018

Duc Nguyen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Electrical and Automation engineering
Valkeakoski

Author Duc Nguyen **Year** 2018

Subject Self- balancing robot with Bluetooth remote control

Supervisor(s) Antti Aimo

ABSTRACT

The target of this project was to create a self- balancing robot with remote control via Bluetooth signal which would be utilized as a conductive household assistant. With a self- balancing ability and mobilizing on different types of surface, the robot could be the most fitting choice in family uses such as carrying goods then delivering to designated points.

The main purpose of this idea was to design a two wheels robot which was able to stabilize even lifting items on it and remotely control in a medium range. Basically, the system itself is not balance, which means it keeps falling off away from the vertical axis because of gravity. Therefore, a MPU-6050 was embedded to sense angle positions of the robot then input signals was sent into the Arduino microcontroller. The microcontroller provided a type of feedback signal to the Dual H-Bridge L298N motor drive to operated motors clockwise or anticlockwise for maintaining robot's position. Additionally, HC-05 Bluetooth module was also connected for controlling remotely as well as making an ease to debug programming.

The robot was completed and tested successfully through the meeting with supervisor. Practically, this project could be developed more tasks such as streaming camera or detecting face devices, etc.

Keywords Arduino, MPU-6050, Dual H-Bridge L298N motor drive, HC-05 Bluetooth module.

Pages **48 pages**

Contents

1 INTRODUCTION

1.1 Introduction

In today's scenario, the field of robotics has been attractive the minds of people worldwide. It was actually the dream of human being to create such as machines that replicate them in every aspect of daily life as well as reflect their thoughts, gestures, postures then perform as human's activities. Development in this field for a last couple of decades has transformed dreams into reality. With the uses of efficient microcontrollers and sensitive sensors has contributed a lot in achieving this milestone. In practice, robots are employed on production lines in factories, appeared as intelligent machines to do specific tasks or used as

commercial products. For instances, it could be utilized in several practical applications with various perspectives such as intelligent gardener in agricultural fields, an autonomous trolley in hospitals, shopping malls, offices, airports or an intelligent robot to guide blind or disable people,etc.

Two wheels self-balancing robot is also a development in the field of robotics. Based on the inverted pendulum theory, two wheels self-balancing robot was designed for this project. With an utilization of a control system that is used to stabilize an unstable system via efficient microcontrollers and specific sensors, this type of robot extremely has gained interest for researchers and engineers.

The aim of the project was to creat and implement a two wheels self-balancing robot that would bring many attributes and aspects of robots in it for daily life. A suitable microcontroller for maintaining robot's position was implemented. Two type of sensors which were used to provide tilt information and encoders with motors were also used to measure wheel's rotation.

1.2 **Problem Statement**

The project was based upon one of the principal problems of mobile robot experiences: self- balancing. Fundamentally, robot was built based on an electro-mechanical structure which can balance itself onto a pair of wheels while standing verticallly. In the situation that the robot kept falling away from the vertical axis, the gyro chip was embedded to specify angular positions then conducted input into the microcontroller, which was programmed in a balancing algorithm. After that, the PID controller provided feedback signal via PWM control signal to turn DC-motors clockwise or anticlockwise, thus balancing the platform. Those measurements were summed and fed- back to the actuator which produced counter torque required to maintain the position of robot.

The robot was created to mobile upon different sorts of surface, stable or rough, based on the maximum torque that wheels were able to supply with the help of dedicated DC/ Servo motors. Other problems were locomotion, autonomy and risk assesment for an autonomous self- balanced mobile robot in order to manage a sense of position and to navigate without human intervention was a paramount.

1.3 **Project Objectives**

Through all the experiments and testing stages of the project, there were objectives:

- To practice deeply in Arduino programming via IDE 1.8.5
- To apply relevant libraries into the project
- To use MIT App Inventor for Bluetooth remote control

- To select products that are suitable for the project
- To perceive knowledge of hardware assembling and its electronic learning.
- To gain mechanical skills in building mobile robot

1.4 **Project Scopes**

The scopes were given as follows:

- Using MPU-6050 Inertial Measurement Unit for stability
- Using DC- motors with RPM 255 for the cause of stability and linear acceleration
- Using Dual H-Bridge L298N as DC motor driver
- Using Arduino Uno R3 as the main board of the robot
- Using PID as the flarness controller
- Using HC-05 Bluetooth Module for remote control
- Using MIT App Inventor 2 for creating control interface on mobile phone as well as monitor

2 THEORETICAL BACKGROUND

2.1 **Inverted Pendulum Theory**

2.1.1 Typical Inverted Pendulum Systems

The inverted pendulum is a classical problem in control systems, and to explore the unstable dynamics, different platforms have been developed. These platforms are similar in many ways, leading to many of the behaviours being comparable. The most common types are the self-balancing robot, inverted pendulum on a cart and an inverted pendulum on a linear track as pictures below

Figure 1. Typical self-balancing robot designed and fabricated in North Carolina, USA by SuperDroid Robots

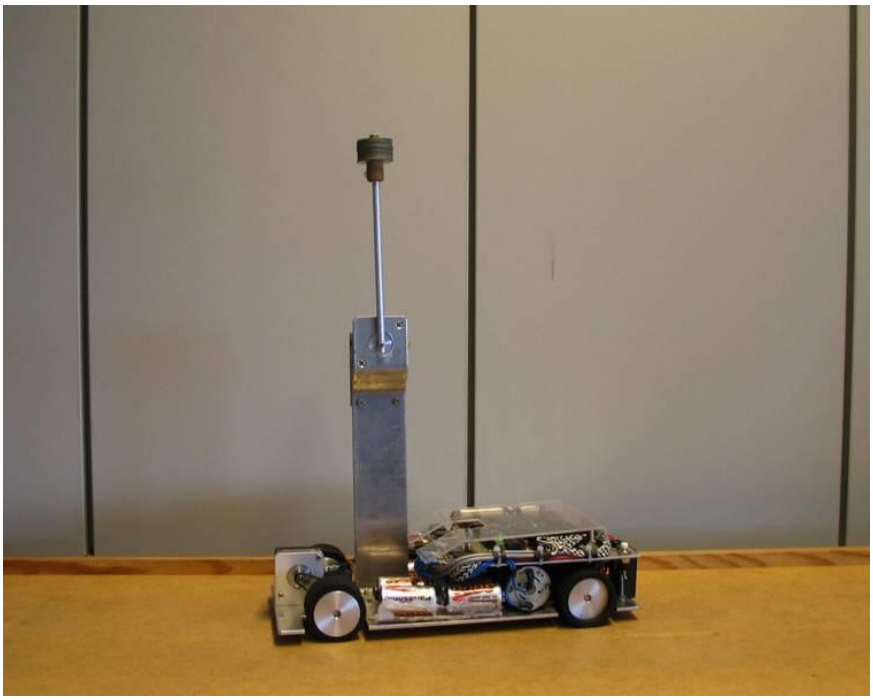


Figure 2. Typical inverted pendulum on a cart designed by Institute of Computer Science Universität Osnabrück

Figure 3. Typical inverted pendulum on a linear designed by ASTI Automation

2.1.2 Simple Pendulum

To better understand these systems, analysis of the dynamics of a simple pendulum is crucial. Afterward, picture below shows a simple pendulum system

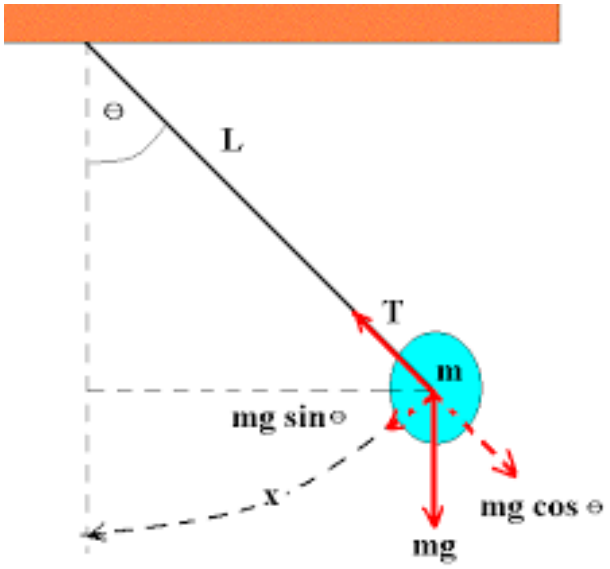


Figure 4. Simple pendulum

Assuming the system on the left, where a mass m is connected by a massless rod of length L to a frictionless pivot. The angular velocity ω and the rate of change of angular velocity $\frac{\mathrm{d}\omega}{\mathrm{dt}}$ are given by the following formulars

$$\frac{d\theta}{dt} = \omega \quad (1)$$

$$\frac{d\omega}{dt} = -\frac{mgL}{I}\sin\theta \quad (2)$$ where I is the moment of inertia and g is the gravitational force.

Intuitively, the system will remain at rest when ω = 0 and $\frac{\mathrm{d}\omega}{\mathrm{dt}}$ = 0. When equation (1) and (2) are solved to fit the criteria, the equilibrium points are found to be $\dot{\theta}$ = 0, and $\theta 1$ = 0 or $\theta 2$ = 180°. $\theta 1$ is stable, but not applicable to self-balancing robots, $\theta 2$ is the target position for inverted pendulum systems. However, at this position, the system is unstable. Any external disturbance will cause the pendulum to move indefinitely away from that specific equilibrium point, hence the need for it to be actively balanced.

2.1.3 Physical Model

Basically, in this project, the robot was built to has the ability to rotate around the z-axis (pitch) by an angle with a corresponding angular velocity shown in the picture below

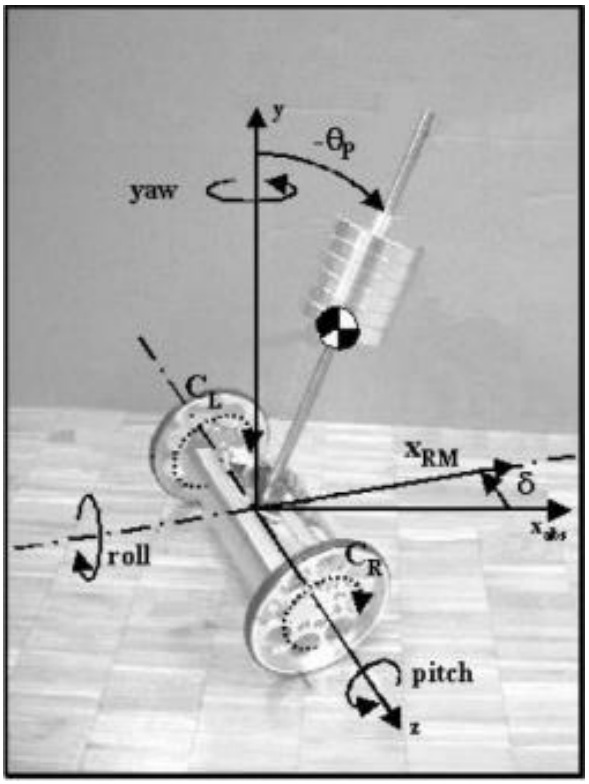


Figure 5. The 3 degrees of freedom of the system (T. Anderson, 2003)

The linear movement of the robot is characterized by the position xRM and the velocity vRM. Furthermore, the robot can rotate around the vertical axis (yaw) by an angle δ with a correspondingangular velocity $\dot{\delta}$.

The control system of the robot was based on two state controls implemented through software

- Controlling the stability around the lateral axis (pitch) for moving forward and backward
- Controlling the stability for rotating around the vertical axis (yaw) for turning left and right.

2.2 **Embedded Sytem**

2.2.1 Definition of Embedded System

Basically, embedded system is a special purpose computer system which is designed to perform a small number of dedicated functions for a specific application (Sachitanand, 2002; Kamal, 2003). Additionally, embedded system is also a combination of hardware and software where software is usually known as firmware that is embedded into the hardware. There are applications utilizing embedded systems such as microwave ovens, TVs, VCRs, DVDs, mobile phones, MP3 players, washing machines, air conditioners, handheld calculators, printers, digital watches, digital

cameras, automatic teller machines and medical equipments (Barr, 1999; Bolton, 2000; Fisher et al., 2004; Pop et al., 2004). Besides these applications, which can be viewed as “noncritical” systems, embedded technology has also been used to develop “safety-critical” systems where failures can have very serious impacts on human safety. For instance, aerospace, automotive, railway, military and medical applications (Redmill, 1992; Profeta et al., 1996; Storey, 1996; Konrad et al., 2004).

Embedded systems engineers are concerned with all aspects of the system development including hardware and software engineering. Therefore, activities such as specification, design, implementation, validation, deployment and maintenance will all be involved in the development of an embedded application. A design of any system usually starts with ideas in people’s mind. These ideas need to be captured in requirements specification documents that specify the basic functions and the desirable features of the system. The system design process then determines how these functions can be provided by the system components.

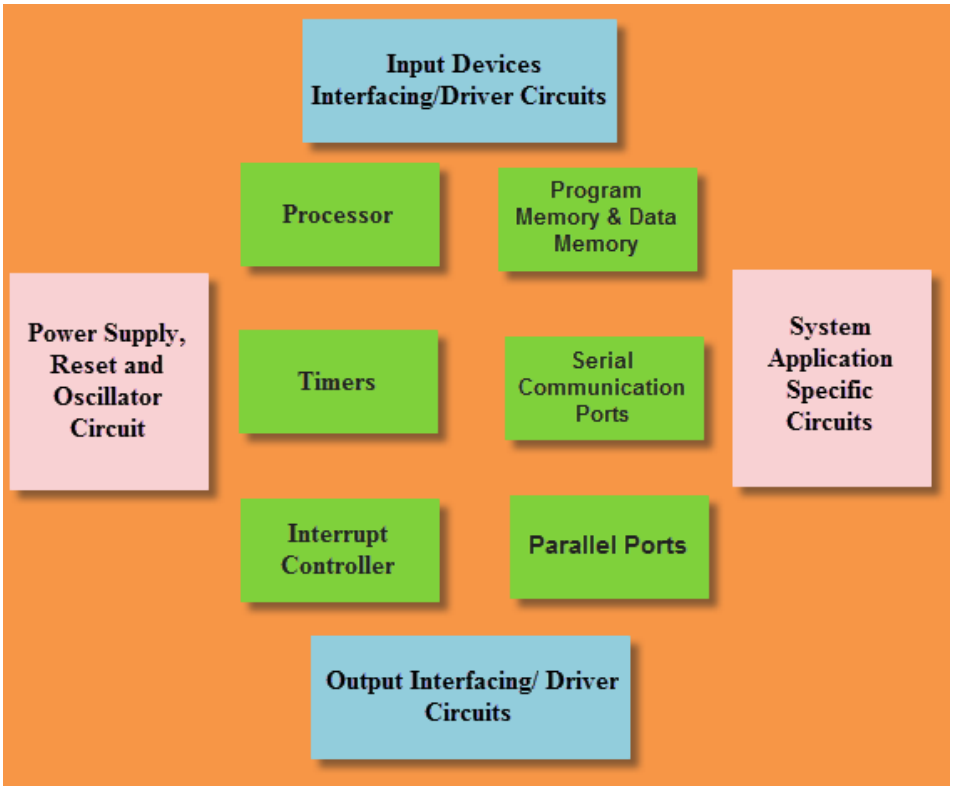


Figure 6. Embedded System block diagram (Elprocus. 2013-2018. Basics Of Embedded System and Applications)

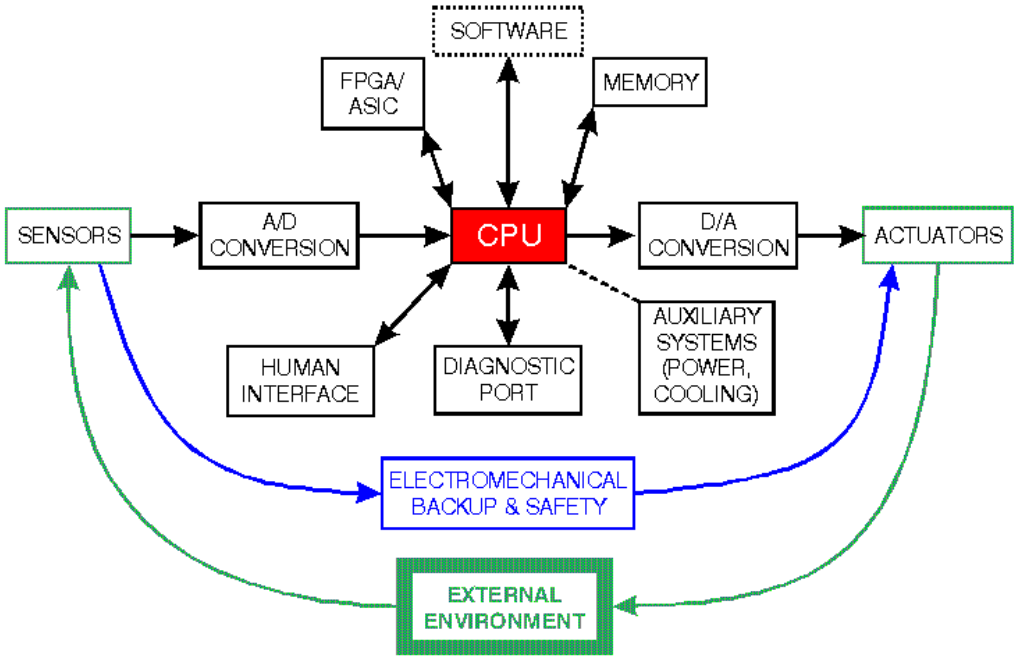


Figure 7. Block description of Embedded System (P. Koopman, "Perils of the PC Cache", Embedded Systems Programming, , 1993)

2.2.2 Embedded System Hardware

Embedded system uses a hardware platform to perform its operation. Typically, hardware of the embedded system is assembled with a microprocessor/microcontroller. Specifically, it has aspects such as input/output interfaces, memory, user interface and the display unit. Generally, an embedded system comprises of the following

- Power Supply
- Memory
- Processor
- Timers
- Input/Output circuits
- Serial communication ports
- System application specific circuits

2.2.3 Embedded System Software

Software of the embedded system is programming to execute a particular function. It is normally written in a high-level setup and then compiled down to offer code that can be stuck within a non-volatile memory in the hardware. An embedded system software is intended to keep in view of the following three limits

- Convenience of system memory

- Convenience of processor's speed
- Limit power dissipation for actions such as run, stop and wake up is necessary while the embedded system runs constantly

2.2.4 Real Time Operating System

The utilization of embedded systems in safety-critical applications requires that the system should have real-time operations to achieve correct functionality and/or avoid any possibility for detrimental consequences. Real-time behavior can only be achieved if the system is able to perform predictable and deterministic processing (Stankovic, 1988; Pont, 2001; Buttazzo, 2005; Phatrapornnant, 2007). As a result, the correct behavior of a real-time system depends on the time at which these results are produced as well as the logical correctness of the output results (Avrunin et al., 1998; Kopetz, 1997). In real-time embedded applications, it is important to predict the timing behavior of the system to guarantee that the system will behave correctly and consequently the life of the people using the system will be saved. Hence, predictability is the key characteristic in real-time embedded systems.

Therefore, this operating system is specially designed to run various applications with an exact timing and a huge amount of consistency. Particularly, this can be significant in measurement and industrial automation systems where a delay of a program could cause a safety hazard.

2.2.5 Classification of Embedded System

2.2.5.1. Small Scale Embedded System

Typically, small scale embedded system is designed by using an 8 bit microcontroller that may even be activated by a battery. For developing embedded software for such system, an editor, assembler or cross assembler are used for specific microcontroller or processor used in the system.

Figure 8. Small Scale Embedded System

2.2.5.2. Medium Scale Embedded System

The medium scale embedded systems are designed using single or multiple 16 bit or 32 bit microcontroller or digital signal processors (DSP's) or reduced instruction set computer (RISC's). These types of embedded systems have both hardware and software complexities. The development tools such as real time operating system shorten as RTOS, source code engineering tools, simulator, debugger and integrated development tools are required for such complex software design system. These software tools also provide the solution for the hardware complexities, so assembler is used very rarely.

Figure 9. Medium Scale Embedded System

2.2.5.3. Sophisticated Embedded System

Sophisticated embedded systems consist of large quantity of hardware and software complexities hence they may required scalable processors or configurable processors and programmable logic arrays.They are used for cutting-edge applications that need hardware and software Co-design and components which have to assemble in the final system.

Figure 10. Sophisticated Embedded System

2.3 **Components**

In this section, specific components that were utilized during experimentation and testing process were described with their principle operations as well as the configuration.

2.3.1 Arduino Uno R3

Arduino is an open-source computer hardware and software company, project and user community that designs and manufactures microcontroller-based kits for building digital devices and interactive objects that can sense and control objects in the physical world.

Arduino comes in variety of sizes and IO pins with multiple modes, the type we are using in this project is Arduino Uno R3 which is based on Atmel's Atmega 328p microcontroller and have a UART communication port alongside 6 analog IO pins, 13 digital IO pins and 6 PWM pins with on-board supply regulators for 5VDC and 3.3VDC as output for various small projects.

Table 1. General details for ATmega328 (Component101. 2018. Arduino Uno Datasheet)

Operating Voltage	5V
Supply Voltage (recommended)	7-12V
Supply Voltage (limited)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328)
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

The reason Arduino Uno R3 was selected because of the affordable price with reusability and market trust as major factors. Although there are several boards available such as Raspberry Pi, STM development kits, TI Cortex M3/M4 boards having more I/O's than Arduino Uno board but it was not necessary to require a large I/O's in this project so Arduino was the foremost choice.

Figure 11. Arduino Uno R3 Board

Next is the upside down view of the Arduino Uno board outline

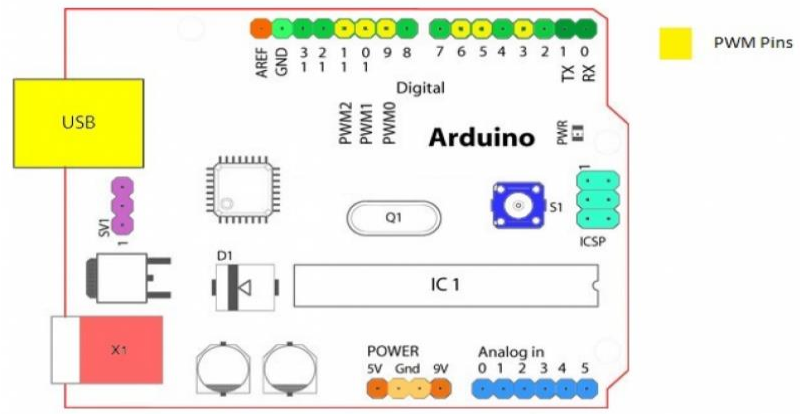


Figure 12. Arduino Board elaborative model

In the figure above, there are several details as follow:

- Analog Reference Pin AREF
- Digital Ground GND
- Digital Pins 2-13
- Digital Pin 0-1 Serial In/Out – TX/RX: For serial communication
- Reset Button S1
- In-circuit Serial Programmer ICSP
- Analog Input Pins 0-5
- Power and Ground Pins
- External Power Supply (9-12VDC) X1
- Toggles External Power and USB Power SV1
- USB Port

2.3.2 Inertial Measurement Unit MPU-6050

Theoretically, the MPU-60X0 is the world's first integrated 6 axis MotionTracking device that combines a 3 axis gyroscope, 3 axis accelerometer, and a Digital Motion Processor™ abbreviated as DMP all in a small 4x4x0.9 mm package. With its dedicated I2C sensor bus, it directly accepts inputs from an external 3 axis compass to provide a complete 9 axis MotionFusion™ output. The MPU-60X0 MotionTracking device, with its 6 axis integration, on-board MotionFusion™, and run time calibration firmware, enables manufacturers to eliminate the costly and complex selection, qualification, and system level integration of discret devices, guaranteeing optimal motion performance for consumers. The MPU-60X0 is also designed to interface with multiple no inertial digital sensors, such as pressure sensors, on its auxiliary I2C port.

The MPU-60X0 features three 16 bit analog-to-digital converters known as ADC for digitizing the gyroscope outputs and three 16 bit ADCs for digitizing the accelerometer outputs. For precision tracking of both fast and slow motions, the parts feature a user-programmable gyroscope full scale range of ±250, ±500, ±1000, and ±2000°/sec and a user-programmable accelerometer full-scale range of ±2g, ±4g, ±8g, and ±16g.

An on-chip 1024 Byte FIFO buffer helps lower system power consumption by allowing the system processor to read the sensor data in bursts and then enter a low-power mode as the MPU collects more data. With all the necessary on-chip processing and sensor components required to support many motion-based use cases, the MPU-60X0 uniquely enables low-power Motion Interface applications in portable applications with reduced processing requirements for the system processor. By providing an integrated MotionFusion output, the DMP in the MPU-60X0 offloads the intensive MotionProcessing computation requirements from the system processor, minimizing the need for frequent polling of the motion sensor output.

Communication with all registers of the device is performed using either I2C at 400kHz. Additional features include an embedded temperature sensor and an on-chip oscillator with ±1% variation over the operating temperature range. The part features a robust 10.000 g shock tolerance, and has programmable low-pass filters for the gyroscopes, accelerometers, and the on-chip temperature sensor.

For power supply flexibility, the MPU-60X0 operates from VDD power supply voltage range of 2.375V-3.46V.

Additionally, the MPU-6050 provides a VLOGIC reference pin, which sets the logic levels of its I2C interface. The VLOGIC voltage may be 1.8V±5% or VDD.

The MPU-6000 and MPU-6050 are identical, except that the MPU-6050 supports the I2C serial interface only, and has a separate VLOGIC reference pin so that was the reason why chip MPU-6050 was selected to the project.

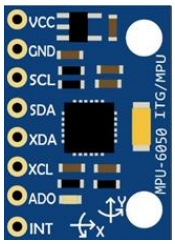


Figure 13. IMU MPU-6050 Circuit Board

By using Fritzing software, the configuration of the MPU-6050 chip to the Arduino board was sketched as below

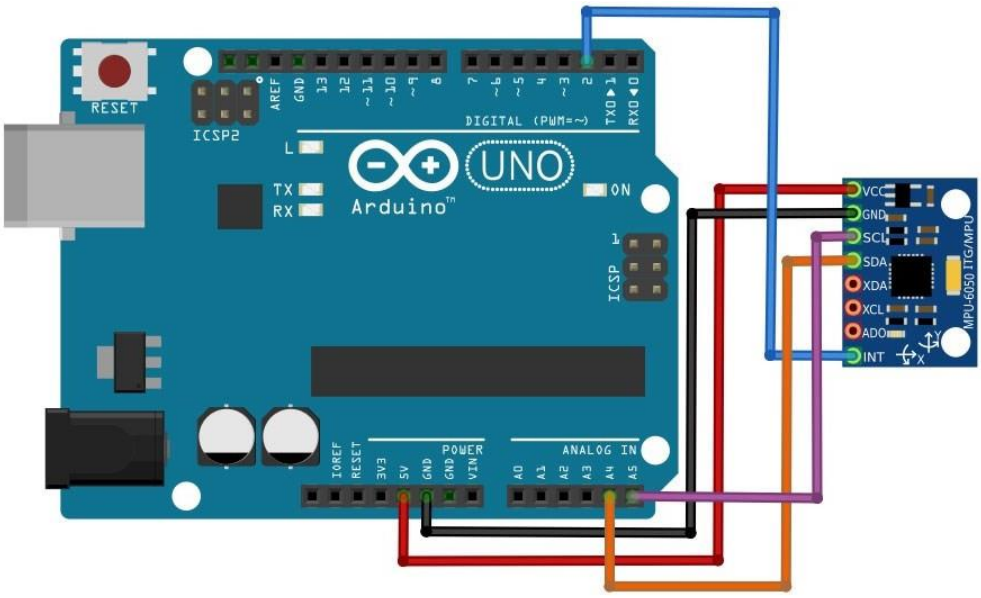


Figure 14. Configuring IMU with Arduino Uno

2.3.3 DC Gear Motors

The DC motors used in this project were DC 3-6V gear motor with plastic wheel tires for smart car Arduino. Additionally, there are features mentioned in the datasheet as magnetic interference motor and high quality rubber wheel, non-slip.

Figure 15. Plastic Robot Wheel Tire with DC 3-6V Gear Motor

Table 2. Specification of DC gear motor

Voltage	DC 3V	DC 6V	DC 7.2V
Current	160mA	220mA	250mA
No load Speed	120 rpm/min	200 rpm/min	250 rpm/min
Load Speed	100 rpm/min	175 rpm/min	210 rpm/min
Torque	0.45 kg.cm	1.0 kg.cm	1.5 kg.cm
Reduction ratio		1:48	
Noise		≤65dB	
Tire diameter		65mm	
Motor size		65mm x 18mm x 22mm (L*W*H)	

2.3.4 Dual H-Bridge L298N DC Motor Drive

The L298 is an integrated monolithic circuit in a 15 lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full bridge driver designed to accept standard logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the

connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

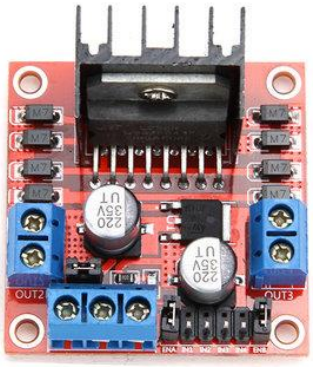


Figure 16. Dual H-Bridge L298N DC Motor Drive

The term H-Bridge is derived from the typical graphical representation of such a circuit. An H-bridge is built with four switches. Since the IC packaging contains two set of H- Bridges, hence the name as Dual H-Bridge. When the switches S1 and S4 are closed while S2 and S3 are opened, a positive voltage will be applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

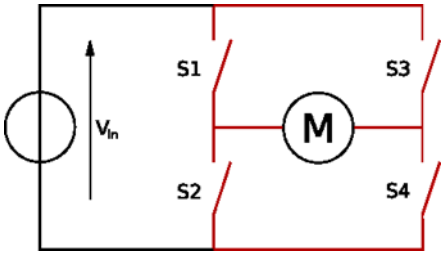


Figure 17. Switches based H-Bridge for driving a DC motor both ways (forward/ backward)

With the use of the nomenclature above, switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. Similarly, switches S3 and S4 operate following that principle. This condition is known as shoot-through.

Moreover, a Dual H-Bridge L298N DC motor drive was chosen because H-Bridge was capable of providing +5 and -5 volts with a maximum of 2.5 A as load current and able to drive two motors simultaneous with a single module based upon L298 IC.

Table 3. The absolute maximum ratings of L298N motor drive

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_i,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

Below is the pins connection information of the motor drive

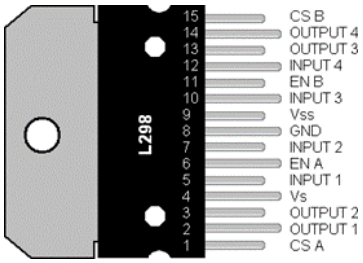


Figure 18. L298N IC with pins description

Following all the instruction, the configuration of the L298N motor drive to the Arduino board was designed for a specific pins wiring in practice with the help of Fritzing software.

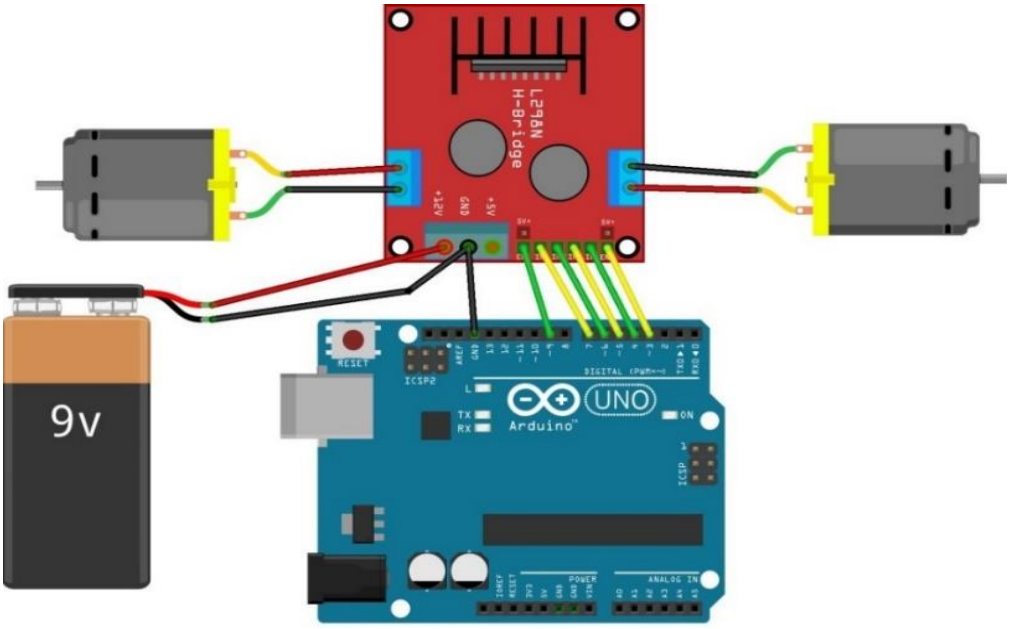


Figure 19. Configuring L298N with Arduino and DC Motors

2.3.5 Bluetooth Module HC-05

Bluetooth Module HC-05 is a typical Bluetooth SPP (Serial Port Protocol) module, designed for wireless serial connection set up which is normally utilized for remotely communication projects. Generally, it is based upon a Bluetooth TM 2.0 technology. For more information, serial port Bluetooth Module is totally qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps modulation with complete 2.4GHz radio transceiver and baseband. It also uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH (Adaptive Frequenc Hopping Feature). Beside that, a footprint with the size as 12.7mmx27mm is implemented inside the module. Specifically, the Bluetooth Module HC-05 is created by sevaral detail in hardware and software characteristics.

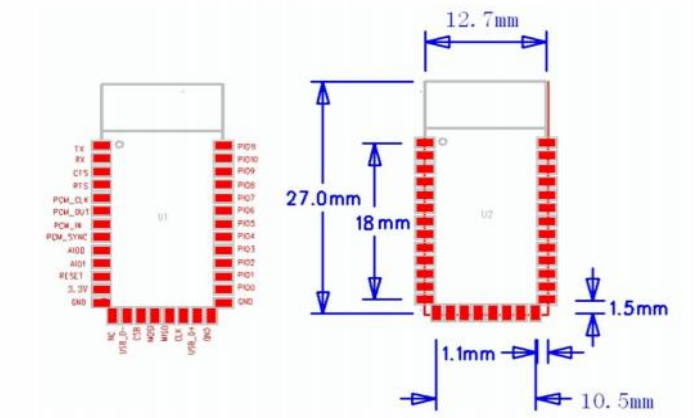


Figure 20. Hardware description

Hardware features are illustrated as follow

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation, 3.3 to 5 V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector

There are assorted software features illustrated below

- Slave default Baud rate: 9600, Data bits:8, Stop bit:1,Parity:No parity
- PIO9 and PIO8 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s
- Auto-connect to the last device on power as default
- Permit pairing device to connect as default
- Auto-pairing PINCODE:"1234" as defaultAuto-reconnect in 30 min when disconnected as a result of beyond the range of connection

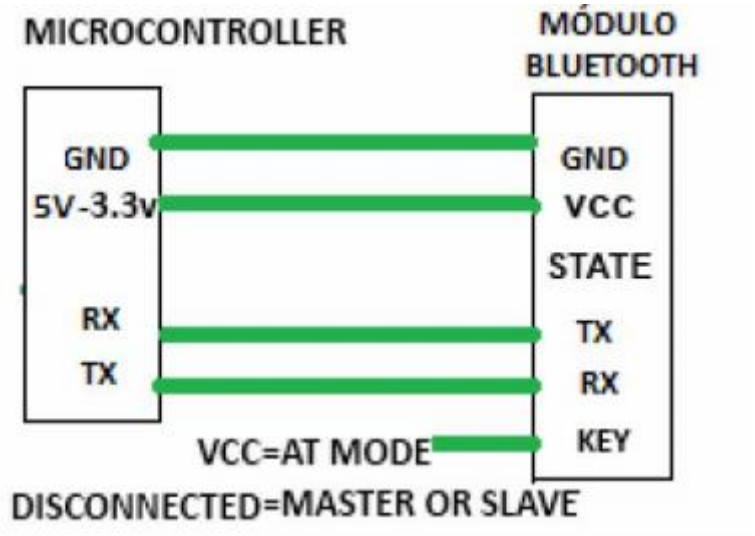


Figure 21. Typical application circuit

HC-05 Bluetooth Module was chosen in a self-balancing robot project to provide a remote monitoring of the behaviour of the robot's pitch angles and for the controlled locomotion of the robot.

Figure 22. HC-05 Bluetooth Module

With the instruction and guidance of the HC-05 Bluetooth Module, the configuration to the Arduino board was sketched via Fritzing software.

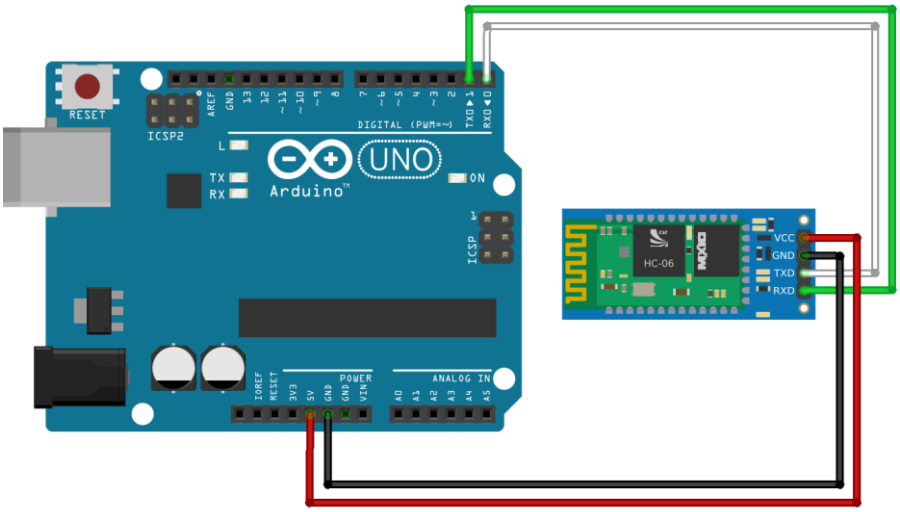


Figure 23. Configuring HC-05 with Arduino

2.3.6 Batteries

Power supply plays an essential role in the robot system. If the power supply is not able to supply enough energy, the robot can not sustain for a long time and other devices can be affected through testing, which causes some errors of those devices.

Initially, the sample of a self-balancing robot was supplied by the battery 9 V type PP3 but the system was not able to be stable and rapidly ran out of energy and caused many disadvantages in testing the system.To correspond to the structural stability in the mechanical robot's design, the rechargeable battery 10.8 V 1.3 AH Li-ion was selected.

Figure 24. Rechargeable battery 10.8 V 1.3 AH Li-ion

Table 4. Basic characteristics of the rechargeable battery 10.8 V 1.3 AH Li-ion

Capacity（25±5℃）	Nominal Capacity：2600mAh (0.52A Discharge， 2.75V) Typical Capacity：2550mAh（0.52A Discharge，2.75V） Minimum Capacity：2500mAh (0.52A Discharge， 2.75V)
Nominal Voltage	3.7V
Internal Impedance	≤ 70mΩ
Discharge Cut-off Voltage	3.0V
Max Charge Voltage	4.20±0.05V
Standard Charge Current	0.52A
Rapid Charge Current	1.3A
Standard Discharge Current	0.52A
Rapid Discharge Current	1.3A
Max Pulse Discharge Current	2.6A
Weight	46.5±1g
Max. Dimension Diameter	Diameter(Ø): 18.4mm Height (H): 65.2mm
Operating Temperature	Charge: 0 ~ 45℃ Discharge: -20 ~ 60℃
Storage Temperature	During 1 month: -5 ~ 35℃ During 6 months: 0 ~ 35℃

2.4 **Software**

2.4.1 Fritzing Sortware

Fritzing is an open source software initiative to support designers and artists ready to move from physical prototyping to actual product. It was developed at the University Of Applied Sciences Of Potsdam.

Figure 25. Fritzing trademark logo

The software is created in the spirit of the processing programming language and the Arduino microcontroller and allows a designer, artist, researcher, or hobbyist to document their Arduino-based prototype and create a PCB layout for manufacturing. The associated website helps users share and discuss drafts and experiences as well as to reduce manufacturing costs.

Following the instruction of Fritzing design, specifically the basic configuration system was buit for clarifying the view of wiring pins and making an ease to connect components.

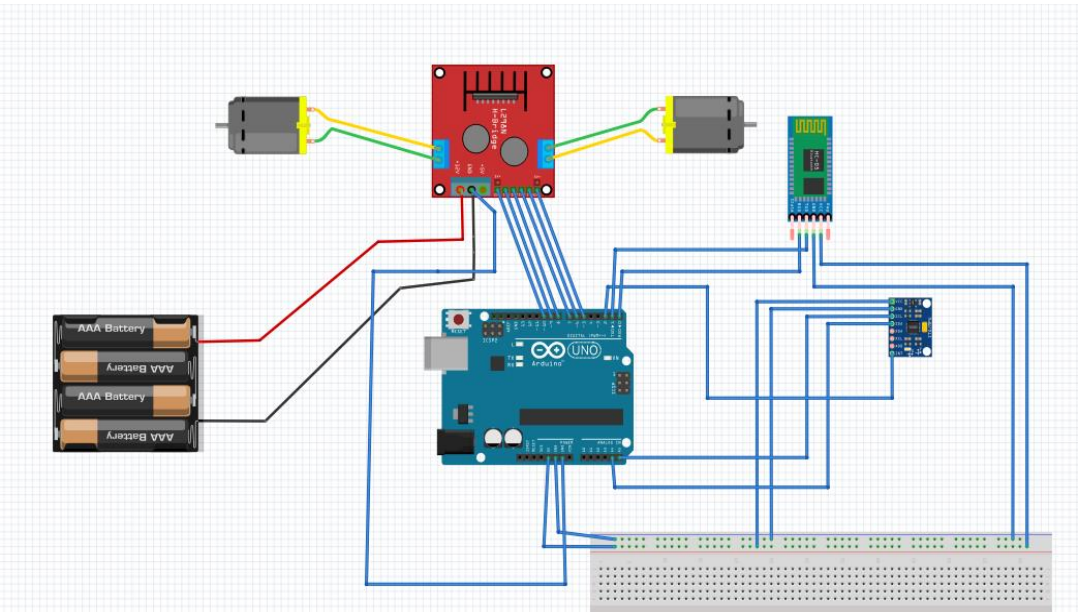


Figure 26. Circuit representation in Fritzing, a screenshot image

2.4.2 IDE 1.8.5 Arduino Software

The Arduino integrated development environment (IDE) is a cross platform application for Windows, macOS, Linux that is written in the programming language Java. It is used to write and upload programs to Arduino board.

Figure 27. IDE 1.8.5 trademark logo

The source code for the IDE is released under the GNU General Public License, version 2. Beside that, the Arduino IDE also supports the languages C and C++ using special rules of code structuring and supplies a software library from the wiring project, which provides many common input and output procedures. Users are only required two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main() into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution.

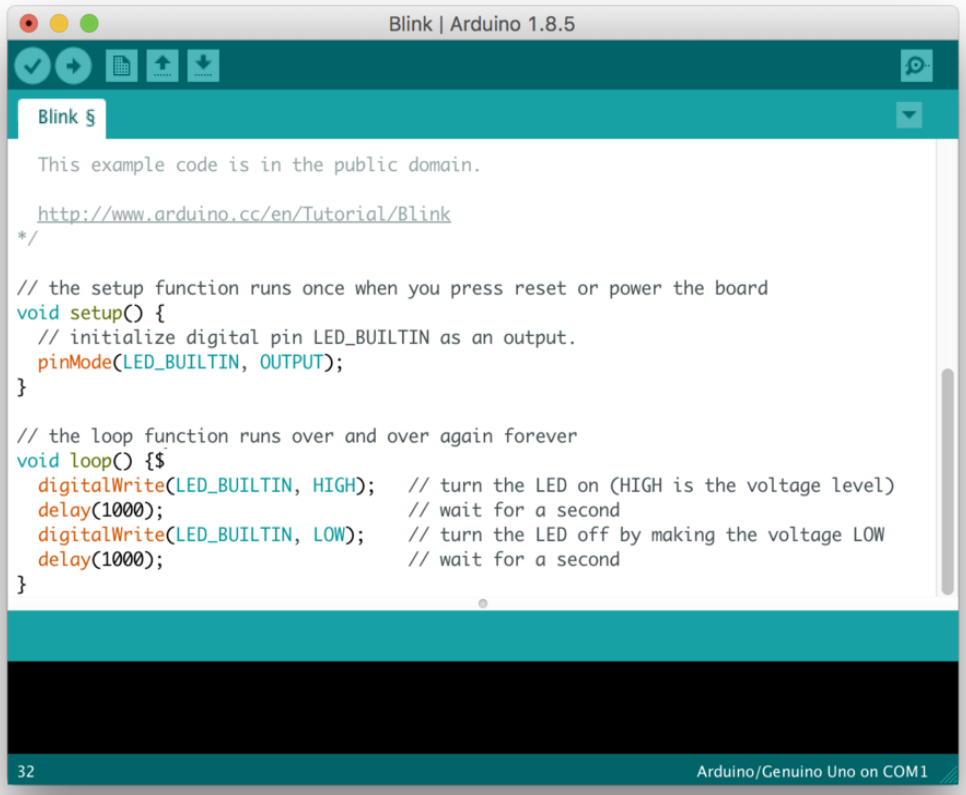


Figure 28. Screenshot of Arduino IDE showing example blink led program

The Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware.

2.4.3 MIT App Inventor 2

App Inventor for Android is an open source web application originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT).

Figure 29. MIT App trademark logo

It allows newcomers to computer programming to create software applications for the Android operating system. It uses a graphical interface, very similar to Scratch and the StarLogo TNG user interface, which allows users to drag-and-drop visual objects to create an application that can run on Android devices. In creating App Inventor, Google drew upon significant prior research in educational computing, as well as work done within Google on online development environments.

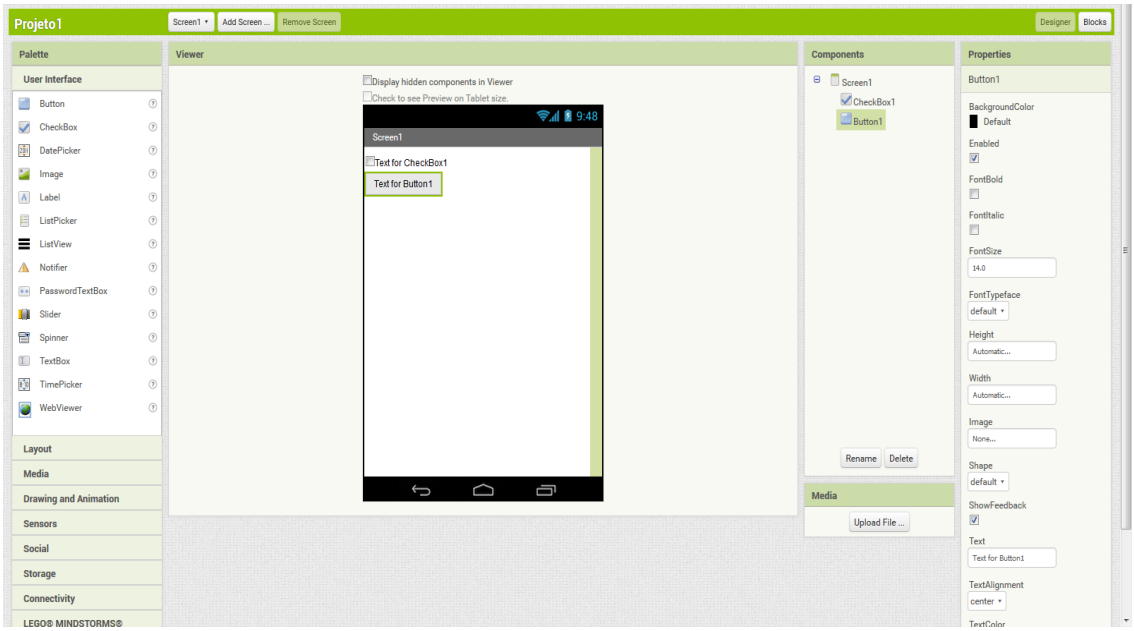


Figure 30. Designing robot's control interface in MIT App Inventor 2

App Inventor and the projects on which it is based are informed by constructionist learning theories, which emphasizes that programming can be a vehicle for engaging powerful ideas through active learning.

2.4.4 Autodesk Inventor

Autodesk Inventor is a computer-aided design application for 3D mechanical design, simulation, visualization, and documentation developed by Autodesk.

Figure 31. Autodesk Inventor desktop icon

Inventor allows 2D and 3D data integration in a single environment, creating a virtual representation of the final product that enables users to validate the form, fit, and function of the product before it is ever built. Furthermore, Autodesk Inventor includes powerful parametric, direct edit and freeform modeling tools as well as multi CAD translation capabilities and in their standard DWG™ drawings.

By using Autodesk Inventor, the final main robot's body version 2.2 was designed as following

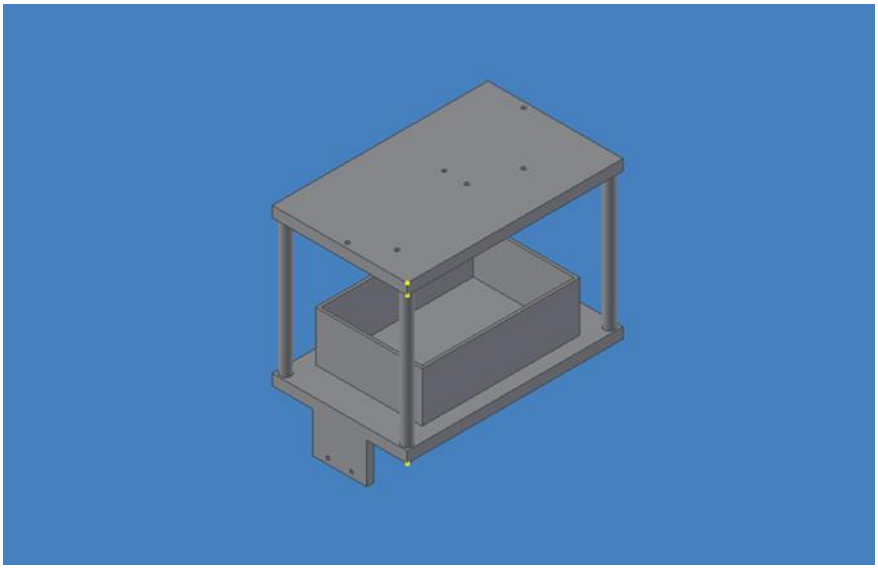


Figure 32. Sketching 3D robot's frame in Autodesk Inventor software

2.5 **Mechanical Structure**

2.5.1 Primary Design Stages

The self-balancing robot was required to be more efficient, more compact as well as convenient to be configured and easier to reassemble while transporting to exhibitions. Beside that, the basic qualification were also demanded to be economical, balanced weight and stable structure as well. In an initial stage of the project, three proposed mechanical structures were designed and tested.

2.5.1.1. Concept Design Version 1.0

The first robot with flats close to the ground simple to assemble marked as a sample version 1.0 which could have provide an efficient balancing ability and it was suitable for carrying outside. Overally, the robot was implemented using two circle acrylic layers with a limited surface capacity and screw holes which has been drilled. That were a reason why it was difficult for assembling and arranging other components, particularly the size of battery as well as wiring. Furthermore, the lack of high tech mechanical equipment and balanced struture due to a significant adjustment in fabricating new version of the robot.

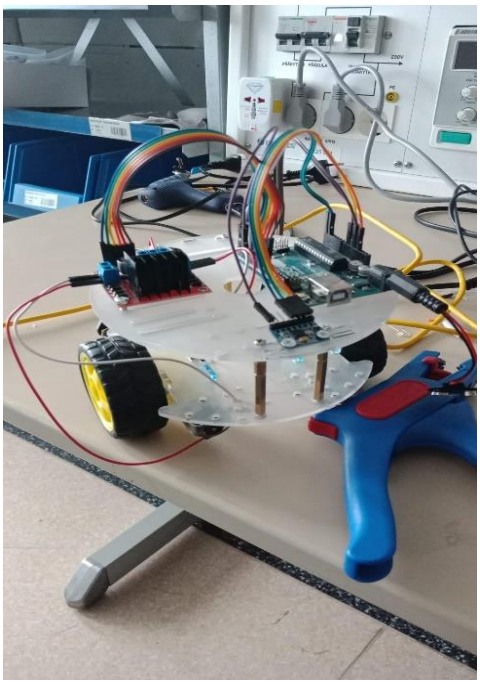


Figure 33. Sample version 1.0

2.5.1.2. Concept Design Version 2.0

The second version was formed by assembling features which were machined through dimensional measurements, cutting, drilling and grinding. The obstacles were that it had got more height level with layer added, fluctuated in DC-motor wheels at the bottom and the disproportionate weight on the top of the robot, which caused stability issues as moment of inertia increases with the height while the lower end motors acting as pivots. Therefore, the frame of the robot was analyzed and rebuilt repeatedly.

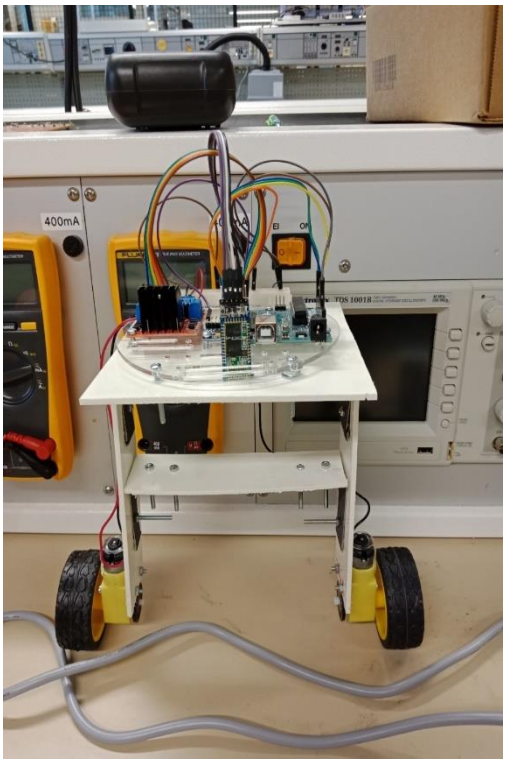


Figure 34. Sample version 2.0

2.5.1.3. Concept Design Version 2.1

In the next experiment based on the last sample, for avoiding the vibration of the system while mobilizing and relatively lower center of gravity, the auxiliary thin plastic flat even was compiled at the bottom end between those DC-motor wheels. However, the system was not constant as expectation because it kept being out-centered from the vertical axis and DC-motor wheels were not able to bear whole robot's body. The sample version 2.1 was arranged as the following figure

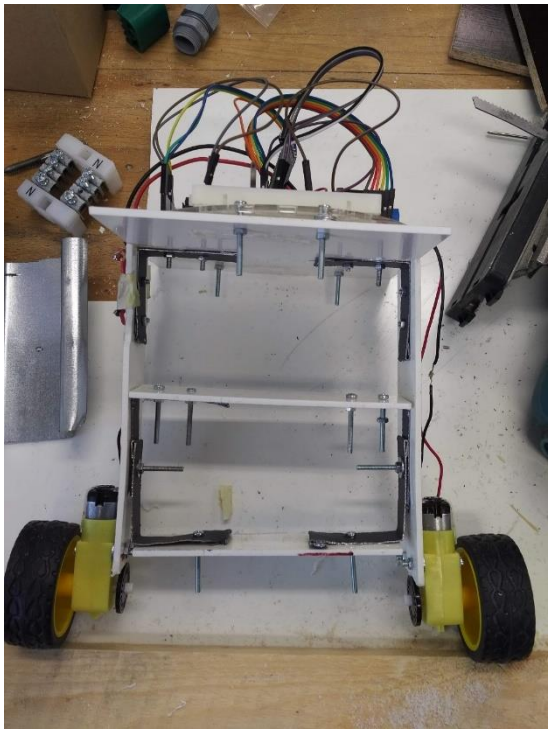
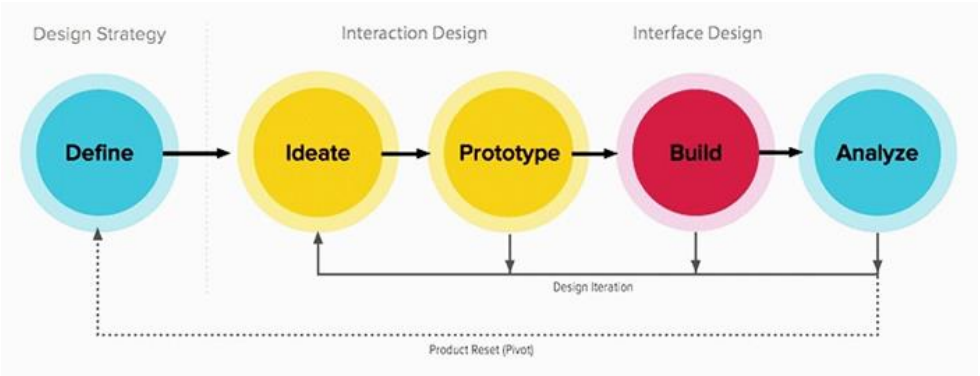


Figure 35. Sample version 2.1

Through all the problems by testing and examining in previous stages, a fourth effort was meticulously planned to reconsider and remodel the whole structure with high durability and valuable quality, working on the following block diagram of the process.



`

Figure 36. Principle for the structure designing process

2.5.2 Final Design Stage

This robot version was reformed by considering all the failures and set backs faced during the implementation of previous versions such as suitable frame, efficient center of gravity, restrained disturbance immunity, scale down a weight handling capacity and out-centered mass, etc.

2.5.2.1. Design Consideration

Following points were the necessary consideration that were incorporated while designing the new structure

- To buid a structure which was demanded a center of gravity as low as possible to the line of action of torque so that the natural stability of the robot is increased.
- The mass of robot's body and materials.
- The supporting frame was strong enough to handle the weight of the base and top plates.
- Appropriately distributing components on the surface of flats for corresponding the balance factor.
- Reasonably adjusting locations of each device for fitting the area on the surface of two plates.

After analyzing the above considerations, the final version was prepared and simulated for the durability under specific loads and turbulent frequencies.

2.5.2.2. Concept Design Version 2.2

The final structure was built by using 3D laser printer. Firstly, parts of robot's frame cautiously were sketched via Autodesk Inventor software based on specifications from datasheets of each required component such as dimension measurements. The main robot's body was determined to have two thin rectangular plates which were designed to fit all devices onto surfaces of robot's layers. Those layers were sustained by four cylindrical rods which were joined into four corners of each layer. Beside that, there was a fixed frame for locating the rechargeable battery which was set onto the surface of bottom plate. Additionally, two small scale pieces for assembling two DC-motors were also merged into small sides of this bottom layer. Because of special plastic material which was utilized in 3D laser printer, the mass of robot's frame was significantly reduced which means the out-centered of gravity problem reasonably was handled.

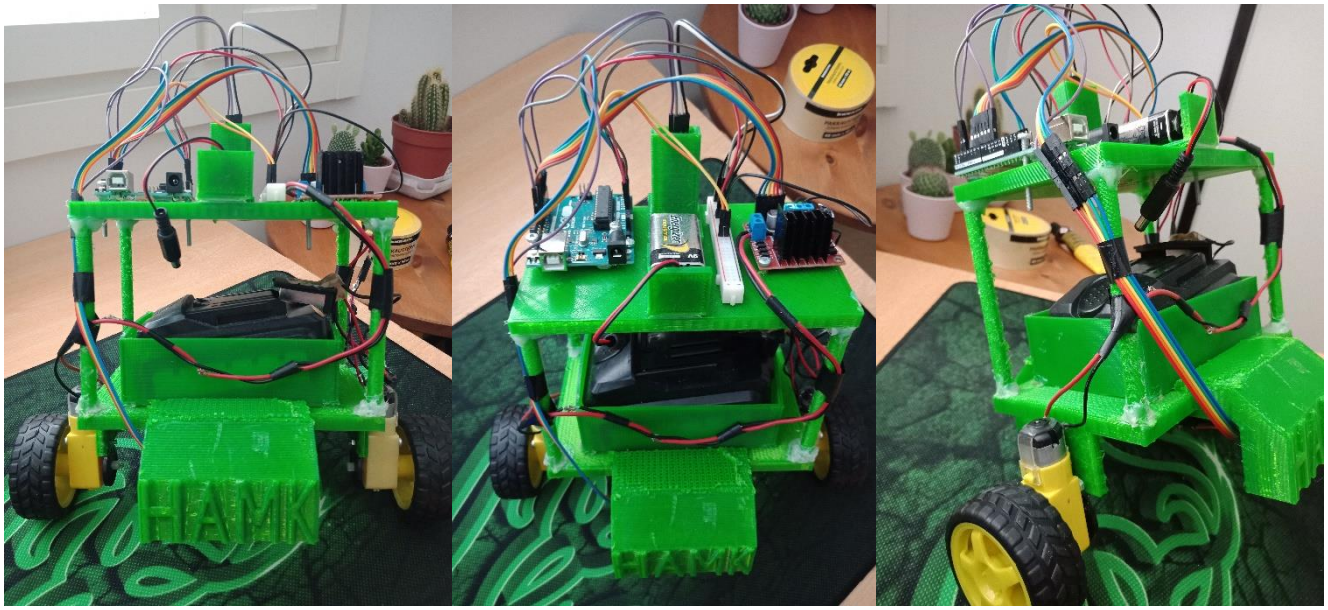


Figure 37. Samples version 2.2

3 HARDWARE IMPLEMENTATION

3.1 Implemented Designs

The following couple of designs was implemented until the current standing structure of the Self-Balancing Robot.

3.1.1 Classic Structure of Self-Balancing Robot

Basically, the robot's structural model plays an essential role in the stability of the robot from material, weight and height perspective. There were two implementations made, first one was created using the ready-to-go structure with acrylic small scale frames. That model had two plates which was located at different levels with making the net height of approximately 12 cm. Due to the lack of assembling area on the surface of flats and the impossibility to carry the mass of battery, this current robot type was restructured.

3.1.2 Robot Version 2.2 Design

After several experiments and reformation of the frame type, the final sample was created. With the structure built by sketching in the Autodesk Inventor software then formed by using 3D laser printer, the robot's body was upgraded to new level such as more compact, less weight, more stable, easier for assembling,etc. The main parts had two layers and four sustainable rods. On the below plate, there was a frame for holding the

rechargeable battery and two pieces for mounting DC-motors. This structure has brought more stability to the robot.

3.1.3 Structure Features Version 2.2

Robot version 2.2 design contains the following features

- Elegant appearance
- Light weight
- Low mass top plate
- Low center of gravity system
- High durability and dependability
- More stable than its predecessor
- Pitch angle measurement using Digital Motion Processor unit built in the IMU
- Autonomous unit with a battery last over 1 hour
- Less jerk sensitive
- Recovering smoothly from oscillation
- Controlling remotely using Bluetooth Module HC-05 via MIT App Inventor 2

3.1.4 Working of the Robot Version 2.2

The working of the robot version 2.2 consists of the following stages

- Sensing the coordinates
- Communicating coordinates changes with the Central Processing Unit
- Generating controlled PWM in response of the observed coordinate changings
- Distributing the smoothened PWM over the two motors using the L298 Module
- Re-monitoring the coordinates changes as compared to the reference state, in order to get stabilized after cancellation of the effect of external turbulence or disturbance

The basic of working details are briefly described as below

When the robot is activated, the first step taken by the Arduino is that the initial coordinates are fetched to match with the reference coordinates set inside the controller. This communication is done using I2C protocol which is operated at 400 kHz and gives a rush of coordinates with as much as 6 digit precision after the decimal point.

The bulk of data is difficult to respond to, as the decisions taken by the controller are not that rapid and if it is assumed to be possible even then it would have raised the net work load of motor to 100%, which is not at all feasible. So a filter is used in order to take the running sum of these values

at a rate of 100 kHz and then take their average to impose one decision for the trend observed in data on the motoring system.

The motor driver is provided with a PWM signal from the Arduino as a result of detection of the direction of motion from the difference of coordinates obtained from the IMU. This PWM signal contains the information of on-time and off-times of motor in the form of varying duty. According to this variation motor is turned ON/ OFF with the help of L298N dual-H-Bridge module in forward or backward direction as per required.

When the rotation has taken place, the coordinates of the IMU, in the meanwhile, have also been updated, so the Arduino re-processes upon those newly obtained coordinates as previously explained, and slowly slows down the duty in the PWM signal as over-ruled by the PID Controller, implemented to the pitch angle and the position of the robot separately, until it completely vanishes down to 0 duty.

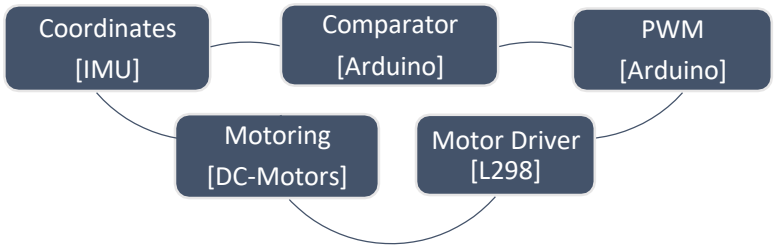


Figure 38. Block diagram of fundamental operation process

4 SOFTWARE IMPLEMENTATION

4.1 Programming

Programming is the most important part determining the outcome of this process. The main programming was divided into two parts including automatic self-balancing mode and controlling via Bluetooth mode with the support of related libraries.

4.1.1 Importing libraries

Before starting programming, the initial stage was to be selecting a relevant libraries into software IDE 1.8.5. Each device was demanded the specific documents which supported to its application. There were assorted libraries which was used in this project such as SoftwareSerial, PID_v1, LmotorController, I2Cdev and MPU-6050.

4.1.1.1. SoftwareSerial library

The Arduino hardware has built-in support for serial communication on pins 0 and 1. The native serial support happens via a piece of hardware called a UART. This hardware allows the Atmega chip to receive serial communication even while working on other tasks, as long as there room in the 64 byte serial buffer.

The SoftwareSerial library has been developed to allow serial communication on other digital pins of the Arduino, using software to replicate the functionality. It is possible to have multiple software serial ports with speeds up to 115200 bps. A parameter enables inverted signaling for devices which require that protocol.

The library has the following known limitations

- If using multiple software serial ports, only one can receive data at a time.
- Not all pins on the Mega and Mega 2560 support change interrupts, so only the following can be used for RX: 10, 11, 12, 13, 14, 15, 50, 51, 52, 53, A8 (62), A9 (63), A10 (64), A11 (65), A12 (66), A13 (67), A14 (68), A15 (69).
- Not all pins on the Leonardo and Micro support change interrupts, so only the following can be used for RX: 8, 9, 10, 11, 14 (MISO), 15 (SCK), 16 (MOSI).
- On Arduino or Genuino 101 the current maximum RX speed is 57600bps
- On Arduino or Genuino 101 RX doesn't work on Pin 13

4.1.1.2. PID_v1 library

Concerning a basic function and definition of PID, It stands for proportional–integral–derivative controller. It is also a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value e(t) as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms, denoted P, I, and D respectively, which give the controller its name.

The distinguishing feature of the PID controller is the ability to use the three control terms of proportional, integral and derivative influence on the controller output to apply accurate and optimal control. The block diagram on below shows the principles of how these terms are generated and applied.

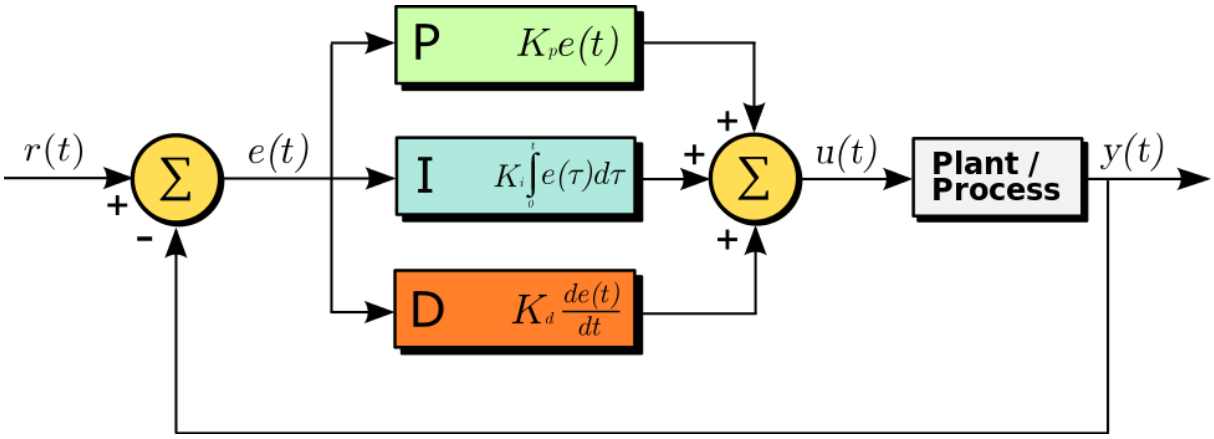


Figure 39. A block diagram of a PID controller in a feedback loop

It shows a PID controller, which continuously calculates an error value e(t) as the difference between a desired setpoint SP =r(t) and a measured process variable PV =y(t) and applies a correction based on proportional, integral, and derivative terms. The controller attempts to minimize the error over time by adjustment of a control variable u(t), such as the opening of a control valve, to a new value determined by a weighted sum of the control terms.

- Term P is proportional to the current value of the SP − PV error e(t). For example, if the error is large and positive, the control output will be proportionately large and positive, taking into account the gain factor "K". Using proportional control alone in a process with compensation such as temperature control, will result in an error between the setpoint and the actual process value, because it requires an error to generate the proportional response. If there is no error, there is no corrective response.
- Term I accounts for past values of the SP − PV error and integrates them over time to produce the I term. For example, if there is a residual SP − PV error after the application of proportional control, the integral term seeks to eliminate the residual error by adding a control effect due to the historic cumulative value of the error. When the error is eliminated, the integral term will cease to grow. This will result in the proportional effect diminishing as the error decreases, but this is compensated for by the growing integral effect.
- Term D is a best estimate of the future trend of the SP − PV error, based on its current rate of change. It is sometimes called "anticipatory control", as it is effectively seeking to reduce the effect of the SP − PV error by exerting a control influence generated

by the rate of error change. The more rapid the change, the greater the controlling or dampening effect.

In conclusion, in the PID library, all codes were programmed based upon formula and function block of PID control loop above with three main variables as Kp, Ki, Kd respectively.

4.1.1.3. LmotorController library

The motor control library contains function blocks that are optimized for the dsPIC33F and dsPIC33E families of Digital Signal Controllers (DSC). All functions in this motor control library have input(s) and output(s), but do not access any of the DSC peripherals. The library functions are designed to be used within an application framework for realizing an efficient and flexible way of implementing a motor control application. In this case, LmotorController was used to control the speed and directions of the DC-motors via H-Bridge L298N motor drive board. Below is the code for LmotorController library

4.1.1.4. I2Cdev library

The I2C Device Library is a collection of uniform and well-documented classes to provide simple and intuitive interfaces to an ever-growing collection of I2C devices. Each device is built to make use of the generic i2cdev code, which abstracts the I2C bit-level and byte-level communication away from each specific device class, making it easy to keep the device code clean while providing a simple way to modify just one class to port the I2C communication code onto different platforms such as Arduino, PIC, simple bit-banging, etc.

The I2Cdev code is built to be used statically, reducing the memory requirement if you have multiple I2C devices in projects. Only one instance of the I2Cdev class is required.

4.1.1.5. MPU-6050 library

The InvenSense MPU-6050 sensor contains a MEMS (Micro Electro Mechanical Systems) accelerometer and a MEMS gyro in a single chip. It is certainly accurate, as it contains 16-bits analog to digital conversion hardware for each channel. Therefore, it captures the x, y, and z channel at the same time. The sensor uses the I2C-bus to interface with the Arduino.

Figure 40. MPU-6000 Family Block Diagram

4.1.2 Main programming

After applying all necessary libraries, the first step was to import them to the Include Library section in IDE 1.8.5 software. There were five different libraries in use

```
#include <SoftwareSerial.h>
#include <PID_v1.h>
#include <LMotorController.h>
#include <I2Cdev.h>
#include <MPU-6050_6Axis_MotionApps20.h>
```

4.1.2.1. Self-balancing mode

For activating MPU control and its status, the following stages were made along with comments

```
MPU-6050 mpu;
bool dmpReady = false; // set true if DMP init was successful
uint8_t mpuIntStatus; // holds actual interrupt status byte from MPU
uint8_t devStatus; // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize; // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount; // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer
volatile bool mpuInterrupt = false; // indicates whether MPU interrupt pin has gone high
void dmpDataReady()
{
 mpuInterrupt = true;
}
```

In next step, motion vectors variables using for locating robot position were declared.

```
Quaternion q; // [w, x, y, z] quaternion container
VectorFloat gravity; // [x, y, z] gravity vector
float ypr[3]; // [yaw, pitch, roll] yaw/pitch/roll container and gravity vector
```

A PID controller is based on user-tunable gains (Kp, Ki & Kd) that, when properly adjusted, can minimize the overshoot of the transient response. Rather than manually specifying a fixed parameter set for the PID, this project tackled the self-calibration of the controller with the popular technique proposed by Ziegler and Nichols. This heuristic tuning method is performed by setting the three gains (Kp, Ki & Kd) to zero, and then increasing the proportional term Kp until it reaches the ultimate gain Ku at which the controller presents self-sustained oscillations of period Tu. Ziegler and Nichols then provide with the mapping between Ku & Tu and the three gains of the PID controller. The ti and td values were converted into the standard PID form using Ki = Kp/ti and Kd = Kp.td.

	K_p	t_I	t_D
Ziegler-Nichols (ZN)	$K_u/1.7$	$T_u/2$	$T_u/8$

Figure 33. Ziegler and Nichols method

Caculated Kp, Ki & Kd variables were estimated as following

```
double input, output;
double Kp = 42.8;// PID Constants
double Ki = 374;
double Kd = 2.5;
PID pid(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);
```

About the method to specify the value for originalSetpoint, whole robot's body was absolutely kept stable in horizontal line by two supported bases. Then the Serial Plotter was accessed to define the originalSetpoint. The value was approximated to match the actual model.

```
double originalSetpoint = 185.68;
double setpoint = originalSetpoint;
```

Through the testing duration, motor speed values were determined along with motor controller set up pins

```
double motorSpeedA= 0.45;
double motorSpeedB= 0.55;
int ENA = 10;
int IN1 = 9;
int IN2 = 8;
int IN3 = 7;
int IN4 = 6;
int ENB = 5;
LMotorController motorController(ENA, IN1, IN2, ENB, IN3, IN4, motorSpeedA, motorSpeedB);
```

At the beginning of void setup(), I2C bus setting was programed. Automatically, the I2Cdev library doesn't operate itself so there was algorithm to join I2C bus.

```
#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
Wire.begin();
TWBR = 24; // 400kHz I2C clock (200kHz if CPU is 8MHz)

#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
Fastwire::setup(400, true);
#endif
```

Additionally, there were parts for checking the I2Cdev. The following step was for initializing device

```
Serial.println(F("Initializing I2C devices..."));
 mpu.initialize();
```

Then the connection of MPU-6050 was arranged

```
Serial.println(F("Testing device connections..."));
Serial.println(mpu.testConnection() ? F("MPU-6050 connection successful") : F("MPU-6050 connection failed"));
```

After that was the loading and configuring the DMP (Digital Motion Processor) as well as PID setup

```
Serial.println(F("Initializing DMP..."));
devStatus = mpu.dmpInitialize();
 // make sure it worked (returns 0 if so)
 if (devStatus == 0)
 {
 // turn on the DMP, now that it's ready
 Serial.println(F("Enabling DMP..."));
 mpu.setDMPEnabled(true);

 // enable Arduino interrupt detection
 Serial.println(F("Enabling interrupt detection (Arduino external interrupt 0)..."));
 attachInterrupt(0, dmpDataReady, RISING);
 mpuIntStatus = mpu.getIntStatus();

 // set DMP Ready flag so the main loop() function knows it's okay to use it
 Serial.println(F("DMP ready! Waiting for first interrupt..."));
 dmpReady = true;

 // get expected DMP packet size for later comparison
```

```
 packetSize = mpu.dmpGetFIFOPacketSize();

 //setup PID
 pid.SetMode(AUTOMATIC);
 pid.SetSampleTime(10);
 pid.SetOutputLimits(-255, 255);
}
else
 {
 // ERROR!
 // 1 = initial memory load failed
 // 2 = DMP configuration updates failed
 // (if it's going to break, usually the code will be 1)
 Serial.print(F("DMP Initialization failed (code "));
 Serial.print(devStatus);
 Serial.println(F(")"));
 }
```

By using the calibration program MPU-6050_DMP6 written by Jeff Rowberg, accelerometer and gyroscope offset were defined

```
mpu.setXGyroOffset(17);
mpu.setYGyroOffset(78);
mpu.setZGyroOffset(-85);
mpu.setZAccelOffset(751);
mpu.setXAccelOffset(1369);
mpu.setYAccelOffset(-48);
```

In the void loop() was about to set up the MPU operation and DMP data.

```
// if programming failed, don't try to do anything
 if (!dmpReady) return;

 // wait for MPU interrupt or extra packet(s) available
 while (!mpuInterrupt && fifoCount < packetSize)
 {
 //no mpu data - performing PID calculations and output to motors
pid.Compute();
motorController.move(output, MIN_ABS_SPEED);
}

// reset interrupt flag and get INT_STATUS byte
 mpuInterrupt = false;
 mpuIntStatus = mpu.getIntStatus();

 // get current FIFO count
 fifoCount = mpu.getFIFOCount();
```

```
 // check for overflow (this should never happen unless our code is too inefficient)
 if ((mpuIntStatus & 0x10) || fifoCount == 1024)
 {
 // reset so we can continue cleanly
 mpu.resetFIFO();
 Serial.println(F("FIFO overflow!"));

 // otherwise, check for DMP data ready interrupt (this should happen frequently)
 }
 else if (mpuIntStatus & 0x02)
 {
 // wait for correct available data length, should be a very short wait
 while (fifoCount < packetSize) fifoCount = mpu.getFIFOCount();

 // read a packet from FIFO
 mpu.getFIFOBytes(fifoBuffer, packetSize);

 // track FIFO count here in case there is > 1 packet available
 // (this lets us immediately read more without waiting for an interrupt)
 fifoCount -= packetSize;

 mpu.dmpGetQuaternion(&q, fifoBuffer);
 mpu.dmpGetGravity(&gravity, &q);
 mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
 input = ypr[1] * 180/M_PI + 180;
```

4.1.2.2. Self-balancing with Bluetooth control mode

There was part of program for operating Bluetooth connection. Firstly, the Bluetooth module pins configuration was declared.

```
const int rxpin = 11;
const int txpin = 3;
SoftwareSerial blue(rxpin, txpin);
  char var = 0;
  int moveState = 0;
```

In the void setup(), the different baud rate of SoftwareSerial and Bluetooth were defined for not being overlapped.

```
Serial.begin(115200);
blue.begin(38400);
blue.setTimeout(10);
```

In the void loop(), the was the code for motion settings for forwad and backward control.

```
if(Serial.available()>0)
 {
  char var = Serial.read();
  delay(10);
   if(var =='F')
  {
  setpoint = originalSetpoint - movingAngleOffset;
  digitalWrite (IN1, HIGH);
  digitalWrite (IN2, LOW);
  digitalWrite (IN3, HIGH);
  digitalWrite (IN4, LOW);
  analogWrite (ENA, abs(MIN_ABS_SPEED));
  analogWrite (ENB, abs(MIN_ABS_SPEED));
  delay(1000);//forward
  }
   else if(var =='B')
  {
  setpoint = originalSetpoint + movingAngleOffset;
  digitalWrite (IN1, LOW);
  digitalWrite (IN2, HIGH);
  digitalWrite (IN3, LOW);
  digitalWrite (IN4, HIGH);
  analogWrite (ENA, abs(MIN_ABS_SPEED));
  analogWrite (ENB, abs(MIN_ABS_SPEED));
  delay(1000);//backward
  }
}
```

4.2 **MIT App Inventor 2 interface programming**

The remote control interface was created through App Inventor 2 with the support of specific components listed. It was built by 5 control buttons matched with forward, backward, left, right, stop and Bluetooth accessing button.

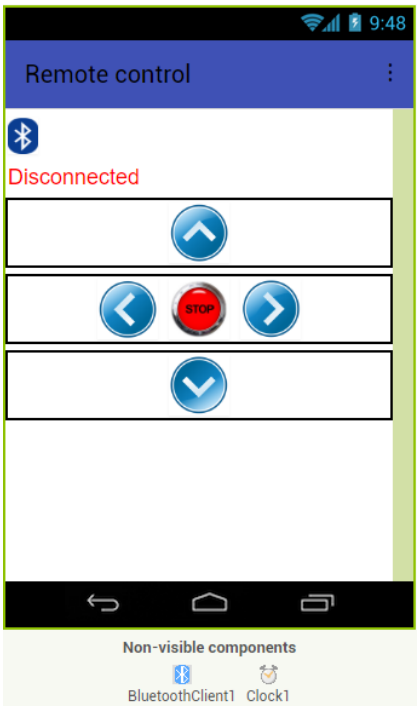


Figure 41. Bluetooth control interface

Function blocks were utilized to program the remote control interface. In the beginnning, Bluetooth connection setting was design to show an address list of Bluetooth devices. After selecting device, the status of Bluetooth on the interface will notify Connected or Disconnected to users. Regarding controlling display, button 1 was labeled as forward, button 3 as backward, button 5 as left, button 6 as right and button 4 as stop.

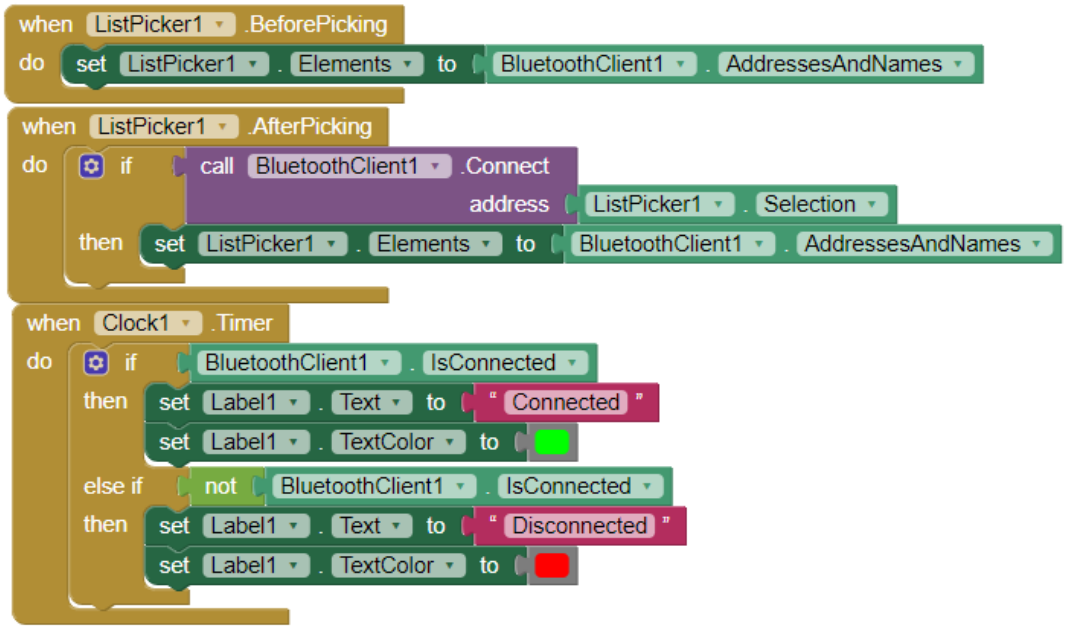


Figure 42. Funtion block of Bluetooth address list of devices for selection

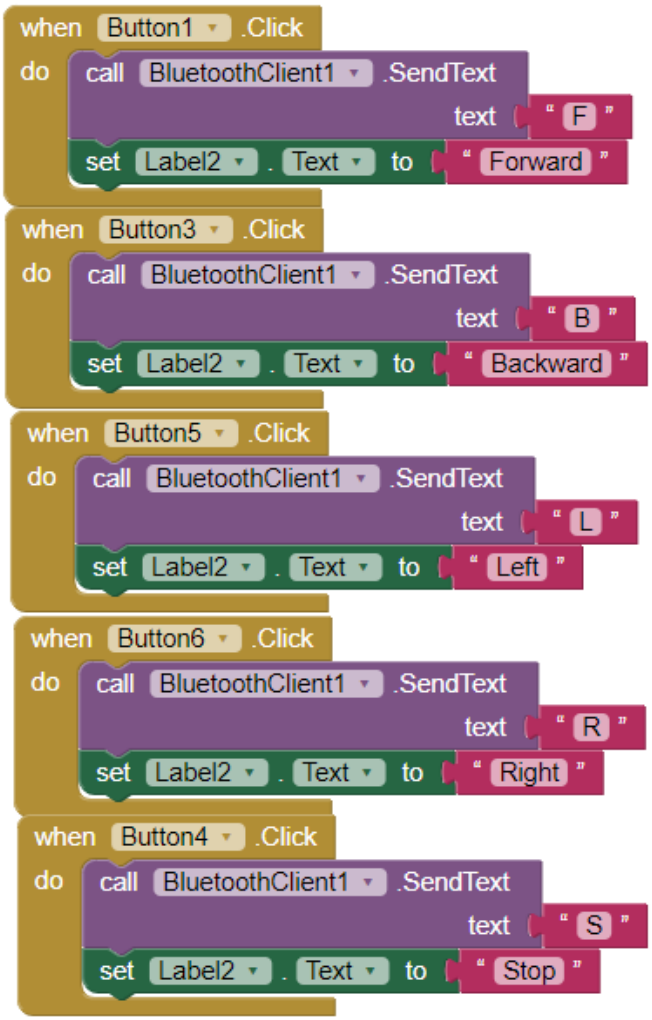


Figure 43. Function block of controlling interface

5 RESULT

5.1 Self-balancing Mode

Through testing and programming duration, the robot perfectly performed a great result although several technical errors remained. Fundamentally, it was able to stay balance itself for a long time or even forced forward and backward, it still rapidly returned to balance. Nevertheless, the current DC-motors which were used in the system were an obstacle in the stability or the mobility mode because of low torque, limited radius of wheel size and restricted revolutions per minute. These features mostly affected self-control ability of system as well as it caused a little bit of instability while standing tall. Fortunately, the disadvantage was minimized to the acceptable range. Hence, the robot was still accomplished and operated its function as planned.

5.2 Bluetooth Control Mode

In this mode sector, basically the remote function was created to control the motion of robot's system such as forward and backward moving via Android phone. Subsequently, the remote interface was designed by using MIT App Inventor 2 with an appropriate result. However, turning left and turning right functions are being researched for example the information about rottation PID variables set up and rotation angle caculation method. Beside that, there was a detriment while using Bluetooth signal that the range for controlling was still limited so it led to the lost signal problem whenever the robot was out of distance.

6 CONCLUSION

In this project, the self-balancing purpose was achieved along with remote motion control by way of Bluetooth signal. In additional, it has a capability to run on the rough or inclined surface.

For a long term plan, it could be developed from Bluetooth remote control into Wi-Fi connection control for further distance as well as streaming camera and coordination equipment could be also assembled when the robot is utilized for scouting or exploring terain.

REFERENCES

Ali, F., Arshad, S. & Rafiq, A. 2016. Self-Balancing Robot: Using IMU as a Feedback Element. Department of Electrical Engineering. University of Engineering and Technology Lahore.

Fabacademy. 2017. Rajan, R. Final Project: Self-Balancing Robot. Fablab Trivandrum, Technopark, Kerala, India. Read on 10.9.2018. http://archive.fabacademy.org/2017/fablabtrivandrum/students/280/final%20project.html

Garcia-Saura, C. 2015. Self-calibration of a differential wheeled robot using only a gyroscope and a distance sensor. Imperial College London. Department of Computing.

GitHub. 2018. Rowberg, J. Arduino library. GitHub, Inc. Read on 12.9.2018. https://github.com/jrowberg/i2cdevlib/tree/master/Arduino

How To Mechatronics. 2017. Arduino DC Motor Control Tutorial – L298N | PWM | H-Bridge. HowToMechatronics. Read on 12.9.2018. https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

How To Mechatronics. 2017. Arduino and HC-05 Bluetooth Module Tutorial. HowToMechatronics. Read on 12.9.2018. https://howtomechatronics.com/tutorials/arduino/arduino-and-hc-05-bluetooth-module-tutorial/

Instructables. 2017. Arduino Self-Balancing Robot. Read on 8.9.2018. https://www.instructables.com/id/Arduino-Self-Balancing-Robot-1/

InvenSense. 2013. MPU-6000 and MPU-6050 Product Specification Revision 3.4. InvenSense Inc. California, United States of America.

Pololu Robotics & Eletronics. 2018. Arduino Uno R3. Pololu Corporation. Read on 8.9.2018. https://www.pololu.com/product/2191

C. Sundin and F. Thorstensson, “Autonomous Balancing Robot”, 2012.

R. C. Ooi, “Balancing a Two-Wheeled Autonomous Robot,” The University of Western Australia, 2003.

B. Bonafilia, N. Gustafsson, P. Nyman and S. Nilson, “Self-Balancing twowheeled robot,” Chalmers University of Technology.

K. Gornicki, “Autonomous Self Stabilizing Robot,” The University of Manchester, 2015.

W. An and Y. Li, “Simulation and Control of a Two-wheeled Self-balancing Robot,” in IEEE Internation Conference on Rotics and Biometrics, Shenzen, 2013.

InvenSense, “MPU-6050,” [Online]. Read on 5.9.2018. https://www.invensense.com/products/motion-tracking/6-axis/mpu-6050/

North Carolina State University, “Using the MPU-6050,” [Online]. Read on 5.9.2018. www.cs.unca.edu/~bruce/Fall13/360/IMU_Wk8.pptx

Component101. 2018. Arduino Uno Datasheet. Read on 8.9.2018. https://components101.com/microcontrollers/arduino-uno

P. Koopman, "Perils of the PC Cache", Embedded Systems Programming, May 1993, 6(5) 26-34.

Elprocus. 2013-2018. Basics Of Embedded System and Applications. Read on 5.9.2018.

https://www.elprocus.com/basics-of-embedded-system-and-applications/