Smart Ping Pong Ball Machine

From: Name - Gurwarris Singh Sohi
Discipline: Computer Engineering Technology
Date: 14 January, 2020

Declaration of Joint Authorship

We, Gurwarris Singh Sohi, Nicolas Cristiano, and Shubham Sharma, confirm that this work submitted is the joint work of our group and is expressed our own words. Any uses made within it of the works of any other author, in any form (ideas, equations, figures, texts, tables, programs), are properly acknowledged at the point of use. A list of the references used is included. The work breakdown is as follows: Each of us provided functioning, documented hardware for a sensor or effector. Gurwarris provided the functioning for a TB6621FNG Dual Motor Driver. Nicolas Cristiano provided the functioning for a SG90 micro Servo Motor. Shubham sharma provided functioning for ROB11015 Solenoid . In the integration effort Nicolas Cristiano is the lead for further development of our mobile application, Gurwarris Sohi is the lead for the Hardware, and Shubham Sharma is the lead for connecting the two via the Database.

Proposal

We have created a mobile application, worked with databases, completed a software engineering course, and prototyped a small embedded system with a custom PCB as well as an enclosure (3D printed/laser cut). Our Internet of Things (IoT) capstone project uses a distributed computing model of a smart phone application, a database accessible via the internet, an enterprise wireless (capable of storing certificates) connected embedded system prototype with a custom PCB as well as an enclosure (3D printed/laser cut), and are documented via this technical report targeting OACETT certification guidelines.

Intended project key component descriptions and part numbers

Development platform: Raspberry Pi 4 B+

Servo Motor SG90: Used to adjust the horizontal rotation of the ping-pong machine

TB6621FNG Motor Driver: Used to adjust the launch speed of the machine in addition to its vertical launch properties

ROB 11015 5V Pull Solenoid: Used to control the time interval between balls launching

We will continue to develop skills to configure operating systems, networks, and embedded systems using these key components to cooperate with one another so they can be integrated into a fully functional ping-pong machine in addition to the ping-pong Smartphone application.

Our application will contain several play options to optimize the amount of success the player will have in being able to develop their skills, including adjustable difficulty, launch interval and horizontal launch. The application will also allow the user to save and access the user’s settings from the last 30 play sessions. These settings are stored into the application’s database after each session has ended.

During the play session, when the ball enters the machine, the solenoid will pull in the ball for however long the user has set the time interval for. Once the ball has launched, it will be able to get launched horizontally within a 180 degree radius, while the difficulty will determine the ball’s launch speed and elevation.

Our project description/specifications will be reviewed by Sebastien Dwornik, ideally an employer in a position to potentially hire once we graduate. They will also ideally attend the ICT Capstone Expo to see the outcome and be eligible to apply for NSERC funded extension projects. This typically means that they are from a Canadian company that has been revenue generating for a minimum of two years and have a minimum of two full time employees.

The small physical prototypes that we build are to be small and safe enough to be brought to class every week as well as be worked on at home. In alignment with the space below the tray in the Humber North Campus Electronics Parts kit the overall project maximum dimensions are 12 13/16" x 6" x 2 7/8" = 32.5cm x 15.25cm x 7.25cm.

Keeping safety and Z462 in mind, the highest AC voltage that will be used is 16Vrms from a wall adapter from which +/- 15V or as high as 45 VDC can be obtained. Maximum power consumption will not exceed 20 Watts. We are working with prototypes and that prototypes are not to be left powered unattended despite the connectivity that we develop.

Executive Summary

Our team has worked towards, creating an affordable and competitive Ping Pong Ball machine. This project was started with comparing the various Ping Pong Ball machines in the market currently. The common observation was that the machines present in the market right now are either too expensive or don’t give you much competitive play features. Our Smart Ping Pong ball machine will incorporate a feasible product that allows a user to play at various difficulties, with the new smart features like database and statistic support and be controlled by an app run on any android device. This machine will achieve complex play settings using minimal and cheaper hardware but, also using smart and intricate firmware. Using less and cheaper hardware allows us to keep the cost down, and compete with currently sold machines in the market. To make manufacturing easier, we can sell the product in the form an assembly kit to keep the manufacturing costs low. Even if, the cost is low, this product can compete with the most competitive and expensive products in the market.

Contents

List of Figures

1.0 Introduction

Our project involves creating a Smart Ping-Pong machine that includes a wide array of play options to allow for a wide range of playing styles. This will allow for players of different skillsets to improve on their ping-pong abilities in a fun and varied manner. In the playing application, the user can select their ideal play preferences to give them a comfortable play experience or to ease them into using the machine. These settings include: play difficulty, organized into easy, medium, and hard categories which affects both the launch speed and the types of shots the machine is able to perform; easy difficulty provides forehand and backhand shots at the standard elevation, whereas the hard difficulty provides the prior 2, in addition to the topspin and backspin shots at random elevations; launch angle, which changes the horizontal launch of the ping-pong ball within a 45-degree radius; and the launch interval which affects the amount of time balls launch between one another, which range from 0.3 to 5 seconds. Our playing application involves a series of databases that allow the player to save and record the fields for the play options of their past play sessions. After 30 sessions have already been input, any new sessions will gradually override the old ones to give the player a better idea of their progress using the machine. The range of elevations the balls launch depends on the difficulty set. It will be set at a permanently neutral elevation if set to easy but will fluctuate if set to a higher difficulty. In terms of scope, first we need to create a hardware prototype and then test for errors to evaluate if we need to change anything for the refined version. Afterwards, we need to create the Firebase databases, which will act as the bridge between the play application and the hardware. The default settings of the play application are set to a launch angle of 5-degrees, a time interval of

5 seconds and an easy difficulty setting. However, the database will be able to transfer the settings of the user’s last play session to their current one, in order to make for a more efficient play experience. Schedule-wise, we plan on building our firmware and application first. We will then develop our databases, which connect to the firmware and application. Lastly, we will develop the enclosure for the machine, which includes the structure for the device and the piping from where the ball launches.

1.1 Scope and Requirements

It is an Internet of Things (IoT) capstone project that uses a distributed computing model of a smart phone application, a database accessible via the internet, an enterprise wireless (capable of storing certificates) connected embedded system prototype with a custom PCB as well as an enclosure (3D printed/lasercut), and is documented via an OACETT certification acceptable technical report. This project will use a database hosted by Firebase which connects the mobile application to the machine. The machine hardware is developed around a Raspberry Pi 4 Model B which can connect to a wireless connections using enterprise wireless certificates. The Raspberry Pi also support good memory and processing power to be able to support multi-threading to control multiple components of the machine. The Pi outputs multiple PWMs and GPIOs to control a Servo Motor SG90 to aim the ball, TB6621FNG that control 2 9V DC motors to launch the ball and Pull type Solenoid to prevent multiple balls to be launched from being launched in a hectic way.

2.0 Background

We would like to thank our mentor Sebastian Dwornik from PDA for supporting this project. He has been instrumental in accomplishing our ideas and hopes for this project by providing support in terms of sorting through our ideas giving them objective approaches, helping with the design for the app and the hardware. Our idea was based after reviewing the designs and features of some of the available machines in the market. The current devices available are either too costly or provided minimal features (13 Best Ping Pong Robots That Will Improve Your Game Instantly, 2020). These machines are all controlled using some connected basic remote and in the age where people control everything in their households using smart switches and mobile applications (Pathan, 2019), why limit this machine to such old methods. Also, connecting the machine with the app and the database will help the user track their progress as they continue to use their machine from beginner to advanced player. Every professional player likes to keep track of their progress/status and this idea has been incorporated with training new players at a young age/beginners (Sumich, 2013). So this machine can be used to help such beginners and help save them some money to provide such training and features at a minimal cost. The more expensive machines available in the market support some different shot types but are unable to change them continuously during a single play though. This machine aims to change that and give highly random shot types to simulate the difficulty of an actual match. To complete this project we have learned how to create android applications in our software production class. We have also learned how to test and create a hardware prototypes for the components of the machine inclusive with a firmware and enclosure for the development platform and the components use in our hardware project class. We have also learned how to create and handle json based databases hosted by Firebase and to Read, write and update them using the applications we create in our software production class. We have already created and presented some hardware prototypes incorporating our individual components for the machine and now we will integrate them to create the refine prototype that can be controlled and operated through the a Firebase database.

3.0 Methodology

This project is called Smart Ping Pong Ball machine, which controls our prototype using our mobile application. The design for hardware uses simple hardware components with a firmware developed on a Raspberry Pi 4 Model B.

3.1 Required Resources

3.1.1 Parts, Components, Materials

When constructing the circuit for our individual projects, it was expected that we use of Breadboards first to troubleshoot for any potential errors. Afterwards, we had to implement our designs onto PCB's which would then connect to a circuit board, the Raspberry Pi 4. 40-pin outputs were used in all circuits to directly connect the PCB's onto the Pi's. An SD card and SD card adapter were used to transfer data and files from the computer directly onto the Pi to allow the Pi to operate properly. An Ethernet cable and a USB-to-Ethernet cable were required to connect the Pi's directly to the computers, which allowed software such as Raspian to be activated and us to remote access it for its GUI and CLI. 3 millimeter acrylic cases were used to encapsulate our hardware designs to ensure that everything in the designs stay in place. For the servo motor design, a 3-pin serial header was used to connect the wires of the servo motor directly onto the PCB. A 220 ohm resistor and a 2.2 kilo-ohm resistor were also used in the design to moderate the amount of current that flowed through the design to ensure the circuit didn't overheat. Additional packs of both types of resistors were required as all the remaining ones were damaged during the soldering process. It also included an NPN transistor to control the flow of the current to allow the Servo Motor to rotate both clockwise and counter clockwise. Finally, an LED was included as a visual way to confirm that the circuit was properly constructed. For the Solenoid design, another NPN transistor, a diode and a 2.2k Ohm resistor was used to amplify the current output to the Solenoid at a low voltage value. The resistor and transistor were used to form the current amplifier, and the diode was used so to cut-off the current form flowing into both directions. Female headers were used to mount the PCB onto the Pi and the input wires to the RB-1011 5V Push-Pull Solenoid onto the PCB. An external 9V power supply was used to power provide the adequate the voltage for the Solenoid and the TB6621FNG Dual Motor Diver. The motor driver drives the 2 6 to 12V bidirectional DC motors using the external power supply and the PWM and GPIO input from the Raspberry Pi 4.

3.1.2 Manufacturing

We all created our PCB's using Fritzing. Once we finished our designs, we sent the designs to the prototype lab for them to print out. The printing process usually took a day from when we sent them the PCB design in gerber file format. During the construction of the Servo Motor PCB, 4 copies had to be printed for various reasons. In the initial design, the copper wiring was placed on opposite layers for some components. As a result, the schematic had to revised to ensure greater design consistency. The next couple of PCB's had to be discarded as a result of soldering complications. As the proper soldering technique had not fully developed, some of the components were put in poor condition and parts of the copper wiring ended up getting singed out. This essentially rendered those PCB's impractical. The final PCB includes a 3-pin serial header, a 40-pin serial header, an NPN transistor, 2 resistors and an LED. The PCB case for the Servo Motor PCB was made using the Corell software. After the design was made, it was sent to the prototype lab for them to construct using the laser cutter, which took approximately 20 minutes. This design had to be printed out a few times to make minor spacing changes to the ports. For the Solenoid, the PCB had to be printed up to 5 times, because of the fault in the initial design led to some of the components becoming unusable for the secondary attempts. During the third attempt, the design was fixed but due to unstable to testing parameters like required minimum current and maximum voltage limit and lack of knowledge regarding these factors led to the small components like transistors and diodes being short-circuited. Also, during the initial design of the PCB using Fritzing, it lacked the components being used which had to be resolved using similar components for the design. But these components not being the same ones as the ones to be used had wrong specifications that resulted in impractical PCB design. The case was laser-cut on black opaque 4mm acrylic. The design was based on a case for a Raspberry Pi 3 but was edited to support ports and nodes on the Raspberry Pi 4. For the Dual Motor driver, the PCB was required 3 attempts. The first attempt was rendered useless, because complications while soldering the sockets on the wrong side of the PCB. The second attempt was the same design which led to the Raspberry Pi crashing and corrupting its EPROM. The fault was discovered and fixed during the third attempt and solved which led to the smooth operation of the DC motors using the TB6621FNG driver. Printing the case for the DC motors component required multiple attempts. Initial designs were based on a case for a Raspberry Pi 3 and edited using Inkscape, but the designs created, and cut were not on scale with design. It took a few tries to figure out how to fix this problem, as it was discovered that the in-software measurements of the design were scrambled while saving them as .cdr format files. When we used Corel Draw for the design, we were successful in cutting the case for the Raspberry Pi. Then we created a fixture to hold the 2 DC motors at a fixed distance so that a ping pong ball could be launched through them. But it took a few recalculations and re-cutting to get the holes just right to hold them in place.

3.1.3 Tools and Facilities

In order to obtain the PCBs, we had to go to the prototype lab to submit our PCB cases designed using Fritzing software in gerber file format and receive the actual PCB which typically took 24 hours. In order to properly solder the various components onto the PCB's, soldering stations from the prototype lab and room J 232 were used. The soldering station consisted of a soldering iron that melted the lead-free soldering wire in place when a component was placed on top. A de-soldering iron with suction pump was also available in case a component needed to be taken out or readjusted. After each solder, a sponge was present to moisten the soldering iron to ensure that no extra solder was left because it could dampen the heat from the soldering iron. During the soldering process, we had to use a vent as a safety precaution to ensure that we didn't inhale any of the fumes that emitted from the solder. In addition, we had to use safety glasses when soldering to ensure that the fumes or any stray piece of metal didn't get in our eyes. Holders were available to hold the PCBs in place while soldering. We also used microscopes with flashlights to solder accurately at small and complex joints. When soldering the PCB for the Servo Motor for the first few times, improper technique was used, resulting in too much solder getting placed onto the ports, melting parts of the copper wire, and resulting in the boards looking disorganized overall. It took about 4 weeks for this PCB to get soldered properly. When creating a case for the component prototypes, we used Inkscape and/or CorelDraw to design it and sent it to the prototype lab in .cdr format for getting laser-cut. We used the datasheets for the Raspberry Pi, solenoid, servo motor and the motor driver to design the circuits and test them. We had some tools available to test our circuits and troubleshoot them when any problems happened. For example, when We had doubts about my circuit and my testing code for the motor driver, we used the tools like oscilloscope and multimeter to first test the output from Pi for its GPIOs and PWMs. These same tools where used to troubleshoot when my Raspberry Pi was shorted by the PCB we had created. We used these devices to troubleshoot the circuit for the Solenoid circuit. We used VNC viewer and Putty to remote access our Raspberry Pi's GUI and CLI modes for development of our prototype code. We used NOOBS software to load the Raspian OS image onto the SD card and used *Thonny* as the IDE to program in python. We are using Firebase to host the databases for our project and using it was the medium between our mobile application and the hardware. We used Android Studio to develop our mobile application written in Java. We also used LinkedIn Learning as a source, to learn various ways to implement our ideas in the both mobile application and the hardware prototypes. We used Trello to keep track of our progress and in combination with Slack that we used to contact and converse with our collaborator Sebastian Dwornik, helped us share our progress with a more featured and described means.

3.1.4 Shipping, duty, taxes

For the Servo Motor project, everything was ordered off of Amazon Prime with everything arriving around 1 day after the time of shipment. There weren’t any duty fees applied to the ordered parts as everything was sourced from Canadian suppliers. There weren’t shipping fees since everything was ordered through an Amazon Prime membership. Otherwise, the shipping fees would have been $6.99/shipment. There weren’t any regular taxes applied to the components, aside from the GST on the microSD card and Adapter, $1.12 (or 13%) of the initial price, $9.32.

For the Solenoid prototype, everything was ordered from Amazon and everything arrived within two days of ordering them. As none of the components were shipped from outside Canada, there were no extra duty charges and all of them were shipped together, having reached the minimum amount, no shipping charges were added. So the final invoice was just the price of the components plus the 13% Tax. But after that some small components had to be reordered and, not having reached the free shipping threshold, $6.99/shipment for two separate shipments were added with including the 13% HST.

For the Motor Driver prototype, The DC motors and the socket headers were bought from Sayal Electronics, so no shipping charges. The Raspberry Pi Kit was ordered from Amazon and having reached the no shipping threshold, only 13% HST was paid. But the motor driver was ordered from Elmwood Electronics, and including with cost of the driver, $10.56 shipping charge and 13% HST was paid.

3.1.5 Time expenditure

For Gurwarris' project, he ordered his dual motor driver from Elmwood Electronics. When he ordered the component, they didn't give him an exact delivery date, which ended up being 3 weeks. However, the male socket headers were already soldered onto the motor driver upon delivery, saving Gurwarris extra work. The Raspberry Pi kit was ordered from Amazon and arrived the day after. The kit was a product from Canakit, which included an already set up SD card with a bootable OS image for a Raspian. Which saved him some time in the working time setting up and went directly to development for the firmware code which was tested on the breadboard prototype. It took me a few hours to design the circuit for both the breadboard and the PCB. Which took up to 24 hours to print. There were some errors in the PCB that had to be led to Raspberry Pi being corrupted. So, it had to be printed again and it took some time for the PCB prototype to function. It took up to four weeks to get the prototype functioning. Also for the acrylic case, it took me up to 4 attempts to get just my case for the Raspberry Pi up to the standards and each attempt took about a couple of hours to edit/fix and 15-30 minutes to cut. The fixture for the motors also took me a whole day of a continuous process of re-measuring and re-cutting to fit it.

For Nicolas' project, all of his components from Amazon Prime, each of which took a day to deliver. The first delivery, consisting of everything aside from the resistors, was in early October. The second delivery, consisting of the 220 and 2.2 kilo ohm resistor packs, was in late November. Complications arose from connecting the Pi to the desktop, which made it difficult to determine if the motor was working properly with the code it was given. It took roughly 5 weeks to determine if the code was working, with an additional 4 spent designing and soldering the PCB.

For the Solenoid prototype, the components arrived two days after they were ordered from Amazon. But because of clashing information and some of it being inaccurate for the circuit to be used. It took me 3 weeks to design the accurate circuit and test it out on the breadboard. This kind of wasted a lot of my time as the PCBs produced had to be soldered, but they ended up not only no working, but rendering my components useless. So we had to order the components again and keep testing the circuit, So the final PCB prototype was finished in 7 weeks .The case took only a couple of attempts to polish and fit for my raspberry Pi.

3.2 Development Platform

3.2.1 Mobile Application

Research has shown that tennis players, both new and experienced, are using tennis ball machines in order to gain experience and develop their abilities in an effective and efficient manner. Capitalizing on this trend, hardware has been developed for a ping-pong machine, which works largely in the same manner as a tennis machine, only on a smaller scale. Software has also been developed for an application that allows the player to adjust various play factors for how the ping-pong machine operates.

Login Activity

Upon opening the application, the player is required to input their individual data into a signup page. These fields include:

- First Name
- Last Name
- Email
- Password
- Password Confirmation

There are several parameters that need to be achieved for the user to be able to sign up. Most significantly, none of the fields are allowed to be empty. The first name and last name must not have any numbers or special characters (ie ?,!$). The email and password must not have any spaces in between characters. The password must be eight to sixteen characters, and have at least one lowercase letter, one uppercase letter, one special character and one number. The password confirmation must match up perfectly with the password. After the user signs up, they can login for all subsequent sessions, where they only need to input their email and password.

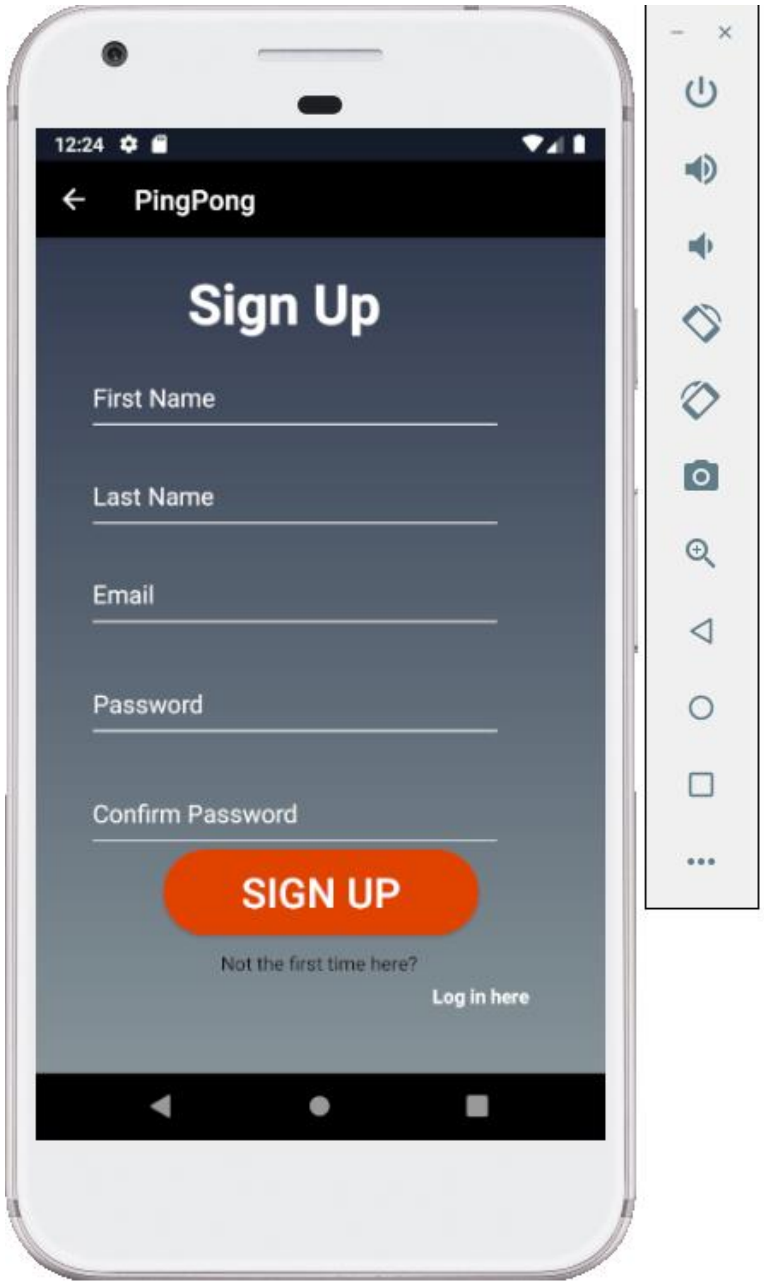


Figure 1: Sign Up Page

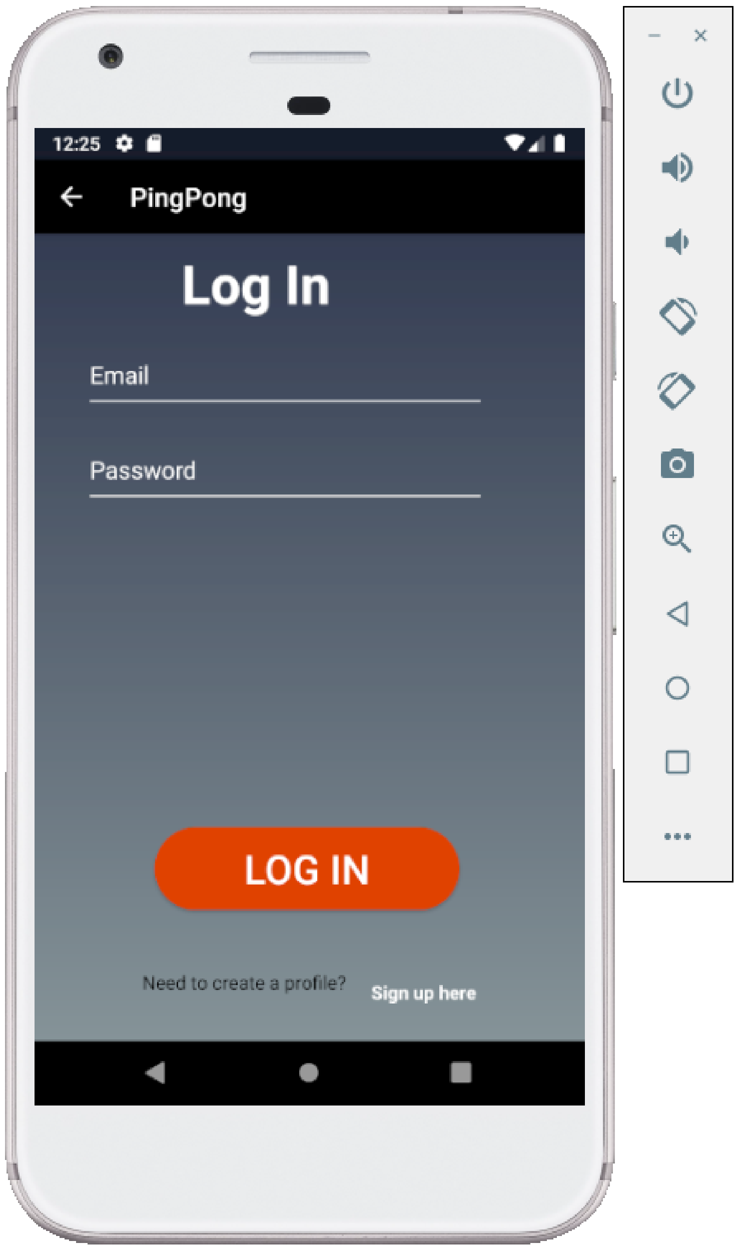


Figure 2: Login Page

Data Visualization Activity

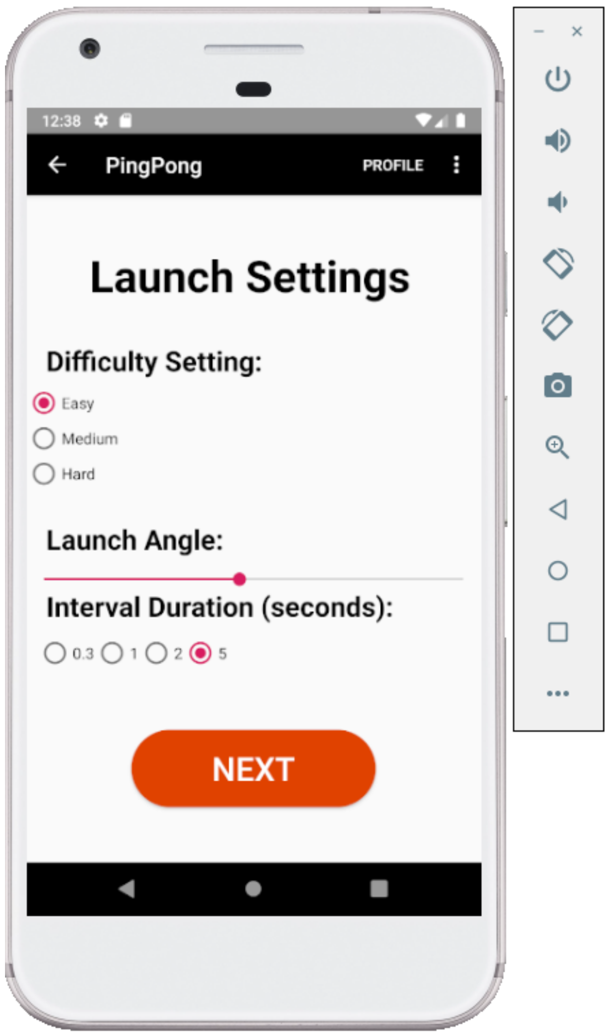
Once the user logs into the application, they can access a profile page, which displays their first name, last name, email address and a serial number that denotes what machine the given account is linked to. When the user first logs in, the application's database stores what serial number they have. The profile screen also lets the users access their play settings from the last 30 play sessions.

Action Control Activity

Before the user enters their play session, they can select from a range of play settings. These settings include:

- Difficulty: easy, medium and hard difficulties are selectable, which impact both the launch speed, the range of elevations and type or variation of shots the ping pong balls launch at the user
- Angle: the user can control the horizontal launch angle of the ball from a neutral center position to a 25 degree range, both left and right of the center.
- Launch Interval: the user can control the time interval the balls launch between one another. They can choose between 0.3, 1, 2 and 5 seconds.

If the user does not choose any of their individual settings, they are set to default, which consists of an easy difficulty, a 10-degree launch angle, and 5 second launch interval. Once the user chooses their settings, they can launch their play session in another screen. When launched, the application displays a timer that shows the duration of the play session. If the user wants to change their settings, they need to press the stop button so they can re-access the settings page. They are then able to restart the play session once everything is adjusted to their liking.



Launch Settings

Difficulty Setting:

[✓ Easy] Medium Hard

Launch Angle:

Interval Duration (seconds):

0.3 1 2 [✓ 5]

]

Figure 3: Launch Settings Page

3.2.2 Image/firmware

The firmware is currently being developed on a Raspberry Pi 4. The development platform is running on the Raspian OS and it is programmed in *python* language. The

IDE being used for this development is *ThonnyPython.* The image of Raspian being used for the development was the one provided by Humber College. It has been set up to allow remote access, using an Ethernet cable and Ethernet-USB adapter to connect to any PC or laptop. The Raspian OS image has been configured to allow remote access and can be accessed using CLI with Putty and using GUI with VNC viewer. The firmware is developed to control the Servo motor, the solenoid and Dual DC motor driver. The code for firmware is now an integrated version of the base code for the prototypes for the given devices. The new firmware was developed using the code for the dual motor driver as base. The present version is not setup to get the play difficulty values stored in the firebase by the mobile application. But in the present version of the code play settings are hard-coded in the code and for a later version, a defined function can be added to get the values from the firebase periodically and use them to run the devices to provide a play level according the settings set by the user. The firmware uses the *GPIO* and *time* module from the Raspberry Pi *python* modules. The *GPIO* module functions are used to use the In/Out pins of the Raspberry Pi in BOARD setting. During the initial setup, 5V outputs are set to turn on the VCCs for the motor driver and the Servo motor. Also, GPIOs used for the Motor Driver logic and the Solenoid are set to output and the 3 PWM outputs to control the Servo motor and the two DC motors. The PWMs are setup to output a 100Hz frequency to provide easy calculation for the value of Duty Cycle needed for favorable physical output from the DC motors and the Servo motor. After the setup is completed by making sure that the GPIOs that control the motors and the solenoid are set to the values of HIGH and LOW as needed to set the solenoid in position and the directions for the motors are set. There are different functions defined to simulate the different shots for the Ping-Pong ball, serve up the ball at the given intervals and turn the machine at a variable angle. The firmware runs in a loop to systematically and periodically, serve up the ball then launch the ball straight. Then the loop runs while turning the angle periodically and alternating between the straight shot, top spin and back spin shots. The angle variation is random, ranging between zero to fifty degrees. For the upcoming version, the defined function needs to be implemented that periodically updates the currently hard-coded values for angle, interval and difficulty.

3.2.3 Breadboard/Independent PCBs

The PCB's were designed using the Fritzing application. In order to start development on the PCB's, we first needed to gain familiarity with how to use the Fritzing application. In order to do so, we got assistance from the teacher, in addition to watching YouTube tutorials to gain an understanding of Fritzing's different uses. We first tested our circuit designs on our Breadboard's to check for errors and to ensure that everything was working as intended. Then created the circuit on a breadboard to test out the voltages and signal outputs to help get an idea on the parameters that were available to create an optimal design for creating the PCB. After our Breadboard designs worked, we started to create the PCB designs. For optimal design purposes, we needed to ensure that the copper wiring that connected to the ports at the top of the Pi were all arranged on the top layer of the PCB. Likewise, we needed to ensure that the wiring for all other components were arranged on the bottom layer. During the design to change the layer for a wiring connection, the bend points were changed to vias

When planning the PCB for the Servo Motor, the PCB was designed to contain an LED to confirm that all components were connected properly when the Raspian code was ran. We also needed to ensure that it contained a 220 Ohm resistor and 2.2 kilo-Ohm resistor connected in parallel to moderate the current of the circuit that would ensure the circuit wouldn't malfunction from too much current. We needed to provide the circuit with 5V of voltage to give it enough power to operate. Finally, we needed to ensure three separate ports for the 3 pins of the Servo Motor that would control the Voltage, Ground, and Pulse wires.

When developing the PCB for the Dual Motor Driver, the PCB was designed to support the all the Logic controls for the direction controls for the DC motors, using GPIOs and speed control using the PWM outputs form the Pi. The VCC logic for the driver used the 5V output from the Pi to enable the driver to control the DC motors. The PCB supports both the outputs from the motors and an external power supply to provide the VM (External Voltage) between 6-12V for the motors. All the components, inputs and outputs have their grounds (GND) connected to the common GND of the Raspberry Pi.

When developing the PCB for the Solenoid, the PCB was designed to support external power supply that provides at least 1 Amp current to the solenoid at 6-9V input. A single GPIO output is used to control the push-pull function by sending LOW-HIGH signals respectively. The circuit design uses a NPN transistor with 220 Ohm and 2.2kOhm resistors amplify the minimal current from the GPIO signal of the Pi. A diode is used to block the back current created from the emf created due to the magnetic effect of the Solenoid.

3.2.4 Printed Circuit Board

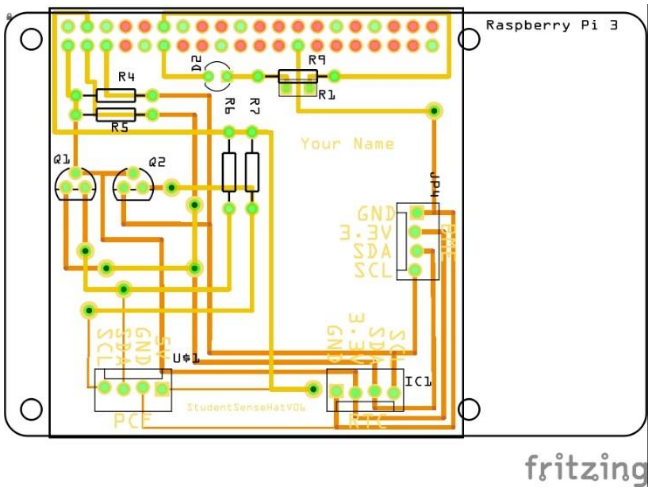


Figure 5. PCB design This work is a derivative of "http://fritzing.org/parts/" by Fritzing, used under CC:BY-SA 3.0.

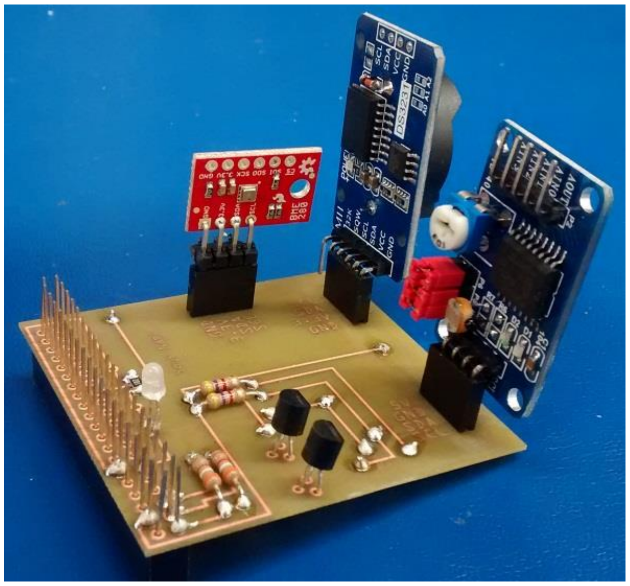


Figure 6. Humber Sense Hat Prototype PCB.

3.2.5 Enclosure

Demo

/1 Hardware present?

/1 Case encloses development platform and custom PCB.

/1 Appropriate parts securely attached.

/1 Appropriate parts accessible.

/1 Design file in repository, photo in report.

How did you build your Prototype: Case?

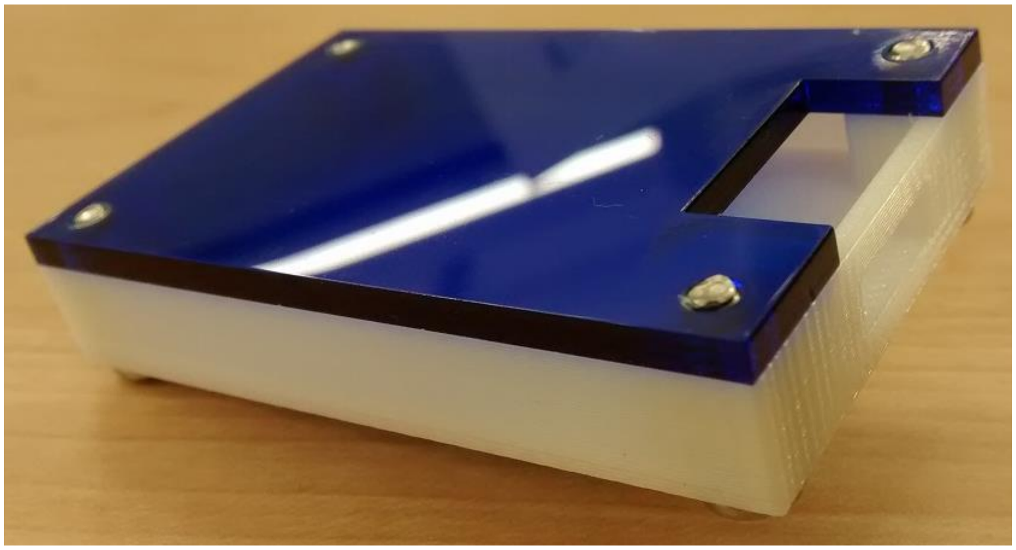


Figure 7. Example enclosure.

3.3 Integration

Demo

/1 Hardware present?

/1 Data sent by hardware

/1 Data retrieved by mobile application

/1 Action initiated by mobile application

/1 Action recieved by hardware

Report

/1 Enterprise wireless connectivity (250)

/1 Database configuration (250 words)

/1 Security considerations (500 words)

/1 Unit testing (900 words)

/1 Production testing (100 words)

3.3.1 Enterprise Wireless Connectivity

A certificate is created when Wi-Fi is activated that ensure that the device it wants to connect to is safe for use and will not get hacked. This is a similar process to connecting a device to a non-local Wi-Fi hotspot, such as at Humber. When the Wi-Fi system acknowledges that the device is safe for use, it allows the device WiFi access and full access to all features and data of the Ping-Pong application. The user's interaction with the application not only allows full interaction with the machine but also connectivity to the application's database without any acknowledgement of the backend aspects. The firmware on machine is to be set up as the only service running on the Pi which only sends and receives that from the Firebase account so it make the account which prevents any harmful data being sent across. So it makes the machine safe to be connected to the any Home or Private Wi-Fi.

3.3.2 Database Configuration

The database uses email and password verification in both the login and signup verification screens to give the system additional information about the unique credentials for each user. Once a new user creates a new account in the signup page, the database stores the information so that the user can conveniently access the information when they use the application any sequential times when logging in. The database is hosted by FireBase, a real time database that allows the application's data to get transferred back and forth instantaneously. The group decided on using FireBase as it provided an easy and convenient interface that allows the users to change their user passwords and email accounts whenever necessary. FireBase is synchronized with the Ping-Pong Machine in real time to allow for the machine to change it's launch characteristics, such as launch angle, launch interval and difficulty settings instantly. This allows for a convenient play experience for the user as they get to experience the changes made in the application immediately after they make them. The group decided on implementing FireStore into our databases as well to allow for storing data that has to be used over time to show the user their growth over time. With FireStore, we can create a requirement both through the application and the machine that the database would only be able to store the user's latest 30 play sessions. This allows the app receive the whole play data at a quicker rate as it we can just get the whole collection soring a user's data in one go. It also gives the user a more accurate impression of their progress in using the machine, since they only need to look through their most recent sessions instead of their full history. Cloud FireStore and Real Time Database allow for permanent access to the data instead of needing to create extensive backend code for checking and updating the database data, as they do not implement SQL. Values sent to the database through to Cloud Firestore include the launch angle, time interval, difficulty and the timestamp of when the play was used, consisting of both the time and date.

3.3.3 Security

Whenever a user creates a password when signing up to the application for the first time, their password gets hashed when it gets stored into the database, getting reformed into a random series of characters. This process is done by the Authentication module of Firebase and even the app creators cannot look up a user's passwords and the Authentication is marked with a system generated User ID that prevents user's from getting into other user's data. Also None of the user's can view any login credentials from FireBase aside from their own. All of these procedures are done in an effort to prevent user's from hacking into accounts they wouldn't otherwise be permitted to use. Security-wise, FireBase does all the authentication on its own instead of having to create backend code to enable security and user safety. Firebase Authentication allows user's to change their passwords and emails associated to the accounts, so if they feel that their information is compromised they can move their accounts to completely different emails but still keep their data about play settings. We can also monitor the amount data passing through any user's interaction with the Firabase which is really small as it only sends some Strings and integers across each time. So if huge chunk of data is being transmitted then it means, the security of he data has been compromised and we can close all traffic or traffic from certain accounts to prevent losing any sensitive data.

3.3.4 Testing

The production testing the machine involves many phases like testing the PCB the components connected to the PCB the Mechanical fitting of the all the components and enclosures. But one of the initial testing we can perform for the machine is the it's connectivity with the database as it takes instructions from the app. For that we don’t even need to connect all the hardware to the Pi. We can just, instead of running the firmware for controlling all the components using the data we get from the app. We can just use the same Firebase connectivity certificate for the machine to run a pseudo code that can read data from the Real Time Database and just print it as an output. To show that accurate data is being received by the machine.

4.0 Results and Discussions

In regards to the software, our project worked perfectly given that our code worked as it needed to. Both in terms of the backend connectivity with server and frontend interaction with the user. The user was able to interact with the database through the application and their play data was stored for future analysis. Our prototype regarding the hardware is incomplete as unforeseen circumstances prevented us from getting the required components from Humber College to fully build the machine. The group managed to learn quite a bit throughout the process of working on the group project. Firstly, the group was able to learn to combine all the pertinent hard skills, such as programming and circuit building, in a more concrete and visibly useful manner. The group learned how to collaborate and communicate effectively with one another, in terms of delegating and organizing tasks, in terms of when to get each done, and how long each task would take in accordance with what the professors outlined in the course schedule. The need to communicate effectively became even more useful after the campus closed as there was a necessity to keep one another informed on each other's progress verbally and visually through various programs, as we no longer had the privilege of working together in person. The group was able to organize and update all pertinent information through Trello, and Github, which served a similar purpose, but was useful in keeping the professor informed of our progress as well as keeping a centralized version control of all the documentation, PCB designs, mechanical designs and source code for both the app and the machine. We also had the privilege throughout the year of collaborating with an industry professional, Sebastien Dwornik. We managed to communicate with him regularly through a messaging program, Slack, in addition to regular video conferences to discuss our progress with him. This allowed him to make recommendations or suggest new ideas to help our project develop in a more efficient manner, providing us a more nuanced perspective in working in a professional work environment and improving our communication skills even further. In terms of developing the actual project, we were able to use various programs such as Fritzing and Android Studio that managed to expand upon various concepts that we learned about from previous semesters. Fritzing gave us a more convenient way to develop and rework our circuits based on schematic criteria using one or more essential sensors or motors. It also helped immensely with developing Printed Circuit Boards (PCB's), as we went from just connecting wires according to circuit design to designing, building and testing one ourselves. During the project, we had to work concurrently between creating the code for our sensors and motors and creating the code for our application. In order to develop the code for the sensors and motors, it was required that we had to learn basic Python, enforcing the need to keep our minds adaptable and flexible to be willing to learn new skills as the project went along. We were also able to expand our knowledge in working with databases as we had to implement databases with different programs such as Cloud Firestore, in addition to implementing the necessary code in our Android Studio application code. In order to develop our application, we used Android Studio, and the Firebase tool supported by it that easily helped us connect to our Firebase Databases. I felt that the project was an overall success, barring a few setbacks that occurred from unforeseen circumstances.

5.0 Conclusions

The project was started with the idea of mass producing the prototype in mind. It was designed this way because of the problem specified for the project to be solved was, making a cheaper and competitive machine and bring it to market. The prototype was built using the basic components, but was run using advanced and refined methods in respect of the machines available in the market. The advanced methods like using a mobile application to control the machine, storing user data in Firebase database to keep track of progress, and running the machine on a Broadcom platform like the Raspberry Pi that has higher capabilities in term of memory and processing speed that are levels higher than simple built PLC chips. These methods were implemented so that it can accomplish the mark of competitive gameplay, instead of just randomly throwing balls at the user.

Hence, all the materials used are easily available and can be produced like in a production line, creating thousands of the different parts separately and assembling them and packaging them to make a marketable product.

For the goal of keeping the product cheap, the product needed to be based on limited number of components, but we believe that every product needs to be upgraded after some time. So, for further progress of the machine, we could still keep improving the application like, analyzing player history and display the growth in forms of charts and tables. We could also keep improving the firmware, which can be updated remotely on already sold machines. The improvements to the machine could entail more complexity in the process of setting up and launching the ball by adding more spin and lob shots.

Therefore, like all the innovations in history we need to keep improving on this idea and we may end up creating a device that leads to inspiration for some engineer in the future generation.

6.0 References

13 Best Ping Pong Robots That Will Improve Your Game Instantly. (2020, January 11). Retrieved from bestpingpongtables: https://www.bestpingpongtables.review/robot/

Cheng, F. (2018). *Build Mobile Apps with Ionic 4 and Firebase: Hybrid Mobile App Development.*

OACETT. (2017, March). *I need to Complete a Technology Report*. Retrieved from The Ontario Association of Certified Engineering Technicians and Technologists: https://www.oacett.org/Membership/Technology-Report-and-Seminar

Robuck, M. (2018, 11). AWS goes deep and wide with machine learning services and capabilities. *Fierceinstaller*.

7.0 Appendix

7.1 Firmware code

This is the link to our **Updated code:**

https://github.com/Warris-Sohi/SmartTennisBallMachine/blob/master/Firmware/SmartTennisFirmware.py

```
##
 # Maker's Digest
 # DC Motor Control with tb6612fng dual h-bridge motor controller
##
from time import sleep      # Import sleep from time
from datetime import datetime
import RPi.GPIO as GPIO     # Import Standard GPIO Module

GPIO.setmode(GPIO.BOARD)      # Set GPIO mode to BCM
GPIO.setwarnings(False);

# PWM Frequency
pwmFreq = 100

# Setup Pins for motor controller
GPIO.setup(7, GPIO.OUT)                # Solenoidcntrl
GPIO.setup(12, GPIO.OUT)       # ServoPWM
GPIO.setup(36, GPIO.OUT)       # AIN2
GPIO.setup(32, GPIO.OUT)       # AIN1
GPIO.setup(37, GPIO.OUT)       # STBY
GPIO.setup(31, GPIO.OUT)       # BIN1
GPIO.setup(33, GPIO.OUT)       # BIN2
GPIO.setup(40, GPIO.OUT)       # PWMA
GPIO.setup(38, GPIO.OUT)       # PWMB

pwmstp = GPIO.PWM(12, pwmFreq)    # pin 18 to PWM
pwmb = GPIO.PWM(38, pwmFreq)    # pin 13 to PWM
pwma = GPIO.PWM(40, pwmFreq)
pwma.start(100)
pwmb.start(100)
pwmstp.start(200)


## Functions
##############################################################################
##
def turnAngle():

        pwmstp.ChangeDutyCycle(10)
        pwmstp.ChangeDutyCycle(20)
        pwmstp.ChangeDutyCycle(30)
```

```
def shoot(spd):
    runMotor(0, spd, 1)
    runMotor(1, spd, 0)

def topshot(spd1,spd2):
    runMotor(0, spd1, 1)
    runMotor(1, spd2, 0)

def backshot(spd1,spd2):
    runMotor(0, spd1, 1)
    runMotor(1, spd2, 0)

def runMotor(motor, spd, direction):
    GPIO.output(37, GPIO.HIGH);
    in1 = GPIO.HIGH
    in2 = GPIO.LOW

    if(direction == 1):
        in1 = GPIO.LOW
        in2 = GPIO.HIGH

    if(motor == 0):
        GPIO.output(32, in1)
        GPIO.output(36, in2)
        pwma.ChangeDutyCycle(spd)
    elif(motor == 1):
        GPIO.output(31, in1)
        GPIO.output(33, in2)
        pwmb.ChangeDutyCycle(spd)


def interval()
       GPIO.output(7,GPIO.HIGH)
       sleep(1)
       GPIO.output(7,GPIO.LOW)

def motorStop():
    GPIO.output(37, GPIO.LOW)

## Main
#############################################################################
#
def main(args=None):
    try:
        while True:
            interval()
            shoot(50)     # run motor forward
            sleep(1)        # ... for 2 seconds
            motorStop()     # ... stop motor
            turnAngle()
            interval()      # delay between motor runs

            topshot(50,30)    # turn Left
            sleep(1)        # ... for 2 seconds
            motorStop()     # ... stop motors
            turnAngle()
            interval()      # delay between motor runs

            backshot(30,50)   # turn Right
            sleep(1)        # ... for 2 seconds
            motorStop()     # ... stop motors
            turnAngle()
            sleep(2)        # delay between motor runs
    except KeyboardInterrupt:
        pass
        motorStop()
        GPIO.cleanup()
if __name__ == "__main__":
    main()
```

7.2 Application code

Here is the link to complete application in our repository: https://github.com/Warris-Sohi/SmartTennisBallMachine/tree/master/Mobile%20App/PiPo

Login.java:

```java
package com.sppm.pipo;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;

public class Login extends AppCompatActivity {
    EditText email,password;
    private FirebaseAuth mAuth;
    String Email,Password;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
        email=findViewById(R.id.usernameText);
        password=findViewById(R.id.passwordText2);
        mAuth = FirebaseAuth.getInstance();
        final Button button = findViewById(R.id.buttonLogin);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Email = email.getText().toString().trim();
                Password = password.getText().toString().trim();
                if (TextUtils.isEmpty(Email)){
                    Toast.makeText(Login.this, "Enter Email",
Toast.LENGTH_SHORT).show();
                    return;
                }else if (TextUtils.isEmpty(Password)){
                    Toast.makeText(Login.this, "Enter Password",
Toast.LENGTH_SHORT).show();
                    return;
                }else if (Password.length()<8) {
                    Toast.makeText(Login.this, "Password must be greater then
8 characters", Toast.LENGTH_SHORT).show();
                    return;
                }
                mAuth.signInWithEmailAndPassword(Email, Password)
                        .addOnCompleteListener(Login.this, new
OnCompleteListener<AuthResult>() {
                            @Override
                            public void onComplete(@NonNull Task<AuthResult>
task) {
                                if (task.isSuccessful()) {
                                    // Sign in success, update UI with the
signed-in user's information
                                    //Log.d(TAG, "signInWithEmail:success");
                                    FirebaseUser user =
mAuth.getCurrentUser();
                                    updateUI(user);
                                } else {
                                    // If sign in fails, display a message to
the user.
                                    //Log.w(TAG, "signInWithEmail:failure",
task.getException());
                                    Toast.makeText(Login.this,
"Authentication failed.",
                                            Toast.LENGTH_SHORT).show();
                                    updateUI(null);
                                }

                                // ...
                            }
                        });
            }
        });
    }

    private void updateUI(FirebaseUser fu) {
        if (fu != null) {
            Intent intent = new Intent(Login.this, Play.class);
            startActivity(intent);}
    }
    @Override
    public void onStart() {
        super.onStart();
        // Check if user is signed in (non-null) and update UI accordingly.
        FirebaseUser currentUser = mAuth.getCurrentUser();
        updateUI(currentUser);
    }

    public void onClick(View view) {
        Intent intent = new Intent(Login.this, SignUp.class);
        startActivity(intent);
    }
}
```

MainActivity.java:

```
package com.sppm.pipo;

import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button button1;
        button1 = (Button) findViewById(R.id.Log_In);
        button1.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                Intent intent = new Intent(MainActivity.this, Login.class);
                startActivity(intent);
            }
        });

    }
    public void onClick(View view) {
        Intent intent = new Intent(MainActivity.this, SignUp.class);
        startActivity(intent);
    }

    }
```

Play.java:

```
package com.sppm.pipo;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.RadioGroup;
import android.widget.SeekBar;
import android.widget.Toast;

import com.google.firebase.auth.FirebaseAuth;

public class Play extends AppCompatActivity {
    String difficulty="easy";
    int LaunchAngle;
    int TimeInterval;
    RadioButton diff;
    RadioGroup radioGroup;
    private SeekBar LAseekBar;
    private SeekBar IDseekBar;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_play);
        radioGroup = (RadioGroup)findViewById(R.id.RadioGroup);
        LAseekBar = (SeekBar) findViewById(R.id.LA);
        IDseekBar = (SeekBar) findViewById(R.id.ID);
        LAseekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            int progress = 0;

            @Override
            public void onProgressChanged(SeekBar seekBar, int progresValue,
boolean fromUser) {
                progress = progresValue;
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {

            }
        });
        IDseekBar.setOnSeekBarChangeListener(new
SeekBar.OnSeekBarChangeListener() {
            int progress = 0;

            @Override
            public void onProgressChanged(SeekBar seekBar, int progresValue,
boolean fromUser) {
```

```
                progress = progresValue;
            }

            @Override
            public void onStartTrackingTouch(SeekBar seekBar) {
            }

            @Override
            public void onStopTrackingTouch(SeekBar seekBar) {
            }
        });
        final Button button = findViewById(R.id.nextButton);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                int selectedId = radioGroup.getCheckedRadioButtonId();
                diff=(RadioButton) findViewById(selectedId);
                if(selectedId==-1){
                    Toast.makeText(Play.this,"Nothing selected for
difficulty", Toast.LENGTH_SHORT).show();
                }
                else{
                    difficulty=diff.getText().toString();
                }
                TimeInterval= IDseekBar.getProgress();
                LaunchAngle= LAseekBar.getProgress();

                Intent intent = new Intent(Play.this, Start.class);
                intent.putExtra("difficulty",difficulty);
                intent.putExtra("TI",TimeInterval);
                intent.putExtra("LA",LaunchAngle);

                startActivity(intent);
            }
        });
    }

    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu1, menu);
        return true;
    }
    public boolean onOptionsItemSelected(MenuItem item) {
//respond to menu item selection
        switch (item.getItemId()) {
            case R.id.Profile:
                startActivity(new Intent(this, Profile.class));
                return true;
            case R.id.Settings:
                startActivity(new Intent(this, Settings.class));
                return true;
            case R.id.SignOut:
                FirebaseAuth.getInstance().signOut();
                startActivity(new Intent(this, MainActivity.class));
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

Post.java:

```
package com.sppm.pipo;

public class Post {
    String difficulty;
    int LA,TI;
    public Post(){}
    public Post(String difficulty,int LA,int TI){
        this.difficulty=difficulty;
        this.LA=LA;
        this.TI=TI;
    }

    public String getDifficulty() {
        return difficulty;
    }

    public void setDifficulty(String difficulty) {
        this.difficulty = difficulty;
    }

    public int getLA() {
        return LA;
    }

    public void setLA(int LA) {
        this.LA = LA;
    }

    public int getTI() {
        return TI;
    }

    public void setTI(int TI) {
        this.TI = TI;
    }
}
```

Profile.java:

```
package com.sppm.pipo;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.firestore.FirebaseFirestore;
import com.google.firebase.firestore.QueryDocumentSnapshot;
import com.google.firebase.firestore.QuerySnapshot;

public class Profile extends AppCompatActivity {
    private static final String TAG = "CloudFireStore";
    TextView idLabel,emailLabel,docText;
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_profile);
        FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
        idLabel=(TextView)findViewById(R.id.profileid);
        emailLabel=(TextView)findViewById(R.id.profile_email);
        docText=(TextView)findViewById(R.id.docText);
        String email = user.getEmail();
        String uid = user.getUid();
        idLabel.setText(uid);
        emailLabel.setText(email);
        /*final Button button = findViewById(R.id.button4);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                db.collection("users")
                        .get()
                        .addOnCompleteListener(new
OnCompleteListener<QuerySnapshot>() {
                            @Override
                            public void onComplete(@NonNull
Task<QuerySnapshot> task) {
                                if (task.isSuccessful()) {
                                    for (QueryDocumentSnapshot document :
task.getResult()) {
                                        Log.d(TAG, document.getId() + " => "
+ document.getData());
                                        String
difficultydata=document.getString("Difficulty");
                                        String
LAdata=document.getString("Launch");
                                        String
TIdata=document.getString("Interval");
                                        String
timedata=document.getString("Timestamp");

docText.setText(difficultydata+";"+LAdata+";"+TIdata+";"+timedata);
                                    }
                                } else {
                                    Log.w(TAG, "Error getting documents.",
task.getException());
                                }
                            }
                        });
            }
        });*/
    }
}
```

SignUp.java:

```
package com.sppm.pipo;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseUser;
import com.google.firebase.auth.UserProfileChangeRequest;

public class SignUp extends AppCompatActivity {
    private static final String TAG = "EmailPassword";
    private FirebaseAuth mAuth;
    EditText fname,lname,email,password,cpassword;
    String Name,Email,Password;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sign_up);
        mAuth = FirebaseAuth.getInstance();
        fname=findViewById(R.id.fnameText);
        lname=findViewById(R.id.lnameText);
        email=findViewById(R.id.emailText);
        password=findViewById(R.id.passwordText);
        cpassword=findViewById(R.id.cpasswordText);

        Button button2;

        button2 = (Button)findViewById(R.id.SignUpbutton);
        button2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Name = fname+" "+lname;
                Email = email.getText().toString().trim();
                Password = password.getText().toString().trim();

                if (TextUtils.isEmpty(Email)){
                    Toast.makeText(SignUp.this, "Enter Email",
Toast.LENGTH_SHORT).show();
                    return;
                }else if (TextUtils.isEmpty(Password)){
                    Toast.makeText(SignUp.this, "Enter Password",
Toast.LENGTH_SHORT).show();
                    return;
                }else if (Password.length()<8) {
                    Toast.makeText(SignUp.this, "Password must be greater
then 8 characters", Toast.LENGTH_SHORT).show();
                    return;
                }
                createAccount(Email,Password);


    }
});
    }
    @Override
    public void onStart() {
        super.onStart();
        // Check if user is signed in (non-null) and update UI accordingly.
        FirebaseUser currentUser = mAuth.getCurrentUser();
        updateUI(currentUser);
    }

    private void updateUI(FirebaseUser currentUser) {
        if (currentUser != null) {
        Intent intent = new Intent(SignUp.this, Play.class);
        startActivity(intent);}
    }

    public void onClick(View view) {
        Intent intent = new Intent(SignUp.this, Login.class);
        startActivity(intent);
    }
    private void createAccount(String email, String password) {
        Log.d(TAG, "createAccount:" + email);

        // [START create_user_with_email]
        mAuth.createUserWithEmailAndPassword(email, password)
                .addOnCompleteListener(this, new
OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task) {
                        if (task.isSuccessful()) {
                            // Sign in success, update UI with the signed-in
user's information
                            Log.d(TAG, "createUserWithEmail:success");
                            FirebaseUser user = mAuth.getCurrentUser();
                            updateUI(user);
                        } else {
                            // If sign in fails, display a message to the
user.
                            Log.w(TAG, "createUserWithEmail:failure",
task.getException());
                            Toast.makeText(SignUp.this, "Authentication
failed.",
                                    Toast.LENGTH_SHORT).show();
                            updateUI(null);
                        }
                    }
                });
        // [END create_user_with_email]
    }

}
```

Start.java:

```
package com.sppm.pipo;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.gms.tasks.OnFailureListener;
import com.google.android.gms.tasks.OnSuccessListener;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.firestore.DocumentReference;
import com.google.firebase.firestore.FirebaseFirestore;

import java.sql.Date;
import java.sql.Timestamp;
import java.util.HashMap;
import java.util.Map;

public class Start extends AppCompatActivity {
    private static final String TAG = "CloudFireStore";
    FirebaseFirestore db = FirebaseFirestore.getInstance();
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_start);
        // Write a message to the database
        FirebaseDatabase database = FirebaseDatabase.getInstance();
        final DatabaseReference dRef = database.getReference("PiPo_DATA");
        TextView labeldiff=(TextView)findViewById(R.id.diffLabel);
        TextView labelLA=(TextView)findViewById(R.id.LALabel);
        TextView labelTI=(TextView)findViewById(R.id.TILabel);
        Bundle bundle = getIntent().getExtras();
        final String difficulty = bundle.getString("difficulty","easy");
        final int LA=bundle.getInt("LA",50);
        final int TI=bundle.getInt("TI",2);
        labeldiff.setText("Difficulty="+difficulty);
        labelLA.setText("Launch Angle="+LA);
        labelTI.setText("Time Interval="+TI);
        int time = (int) (System.currentTimeMillis());
        Timestamp tsTemp = new Timestamp(time);
        String ts =  tsTemp.toString();

        final Post post =new Post(difficulty,LA,TI);
        final Map<String, Object> playpost = new HashMap<>();
        playpost.put("Difficulty", difficulty);
        playpost.put("Launch", LA);
        playpost.put("Interval", TI);
        playpost.put("Timestamp",ts);
        Button button = findViewById(R.id.LaunchButton);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                Toast toast = Toast.makeText(getApplicationContext(),
                        difficulty+TI+LA,
                        Toast.LENGTH_SHORT);
                toast.show();
                dRef.push().setValue(post);
                // Add a new document with a generated ID
                db.collection("playSettings")
                        .add(playpost)
                        .addOnSuccessListener(new
OnSuccessListener<DocumentReference>() {
                            @Override
                            public void onSuccess(DocumentReference
documentReference) {
                                Log.d(TAG, "DocumentSnapshot added with ID: "
+ documentReference.getId());
                                Toast toast =
Toast.makeText(getApplicationContext(),
                                        difficulty+TI+LA,
                                        Toast.LENGTH_SHORT);
                            }
                        })
                        .addOnFailureListener(new OnFailureListener() {
                            @Override
                            public void onFailure(@NonNull Exception e) {
                                Log.w(TAG, "Error adding document", e);
                                Toast toast =
Toast.makeText(getApplicationContext(),
                                        "Cloud Firestore update failed",
                                        Toast.LENGTH_SHORT);
                            }
                        });
            }
        });

    }
}
```