The Gyroscopic Marble Maze

ECE 631 Final Project

Spring 2019

Mike Devoe & Erin Payne

Introduction

For our final project we decided to create a three dimensional gyroscopic maze controlled by the accelerometer sensor of the SMT32 IOT node. By placing the maze on a set of servers, the user can tilt the STM32 board to navigate a ball bearing along the x and y planes to get it from start to finish, solving the puzzle.

Layout

The physical maze itself consists of a multitude of 3D printed components and two servos to manipulate forward and backward. For the 3D printed parts we had to build a four sided, three dimensional maze, set atop two gimbal bases stacked in 90 degree opposition with slots to hold in place each servo to tilt the maze forward, backward, left and right. Figure 1 below displays the listed 3D printed parts before assembly.

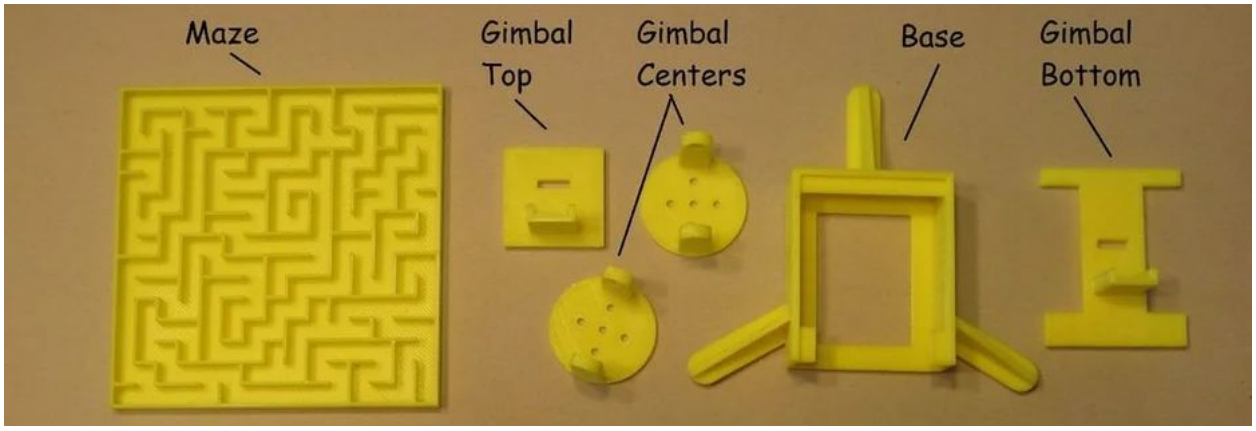

Figure 1.) Visual of listed 3D printed components making up the physical gyroscopic maze.

In terms of the hardware that was utilized for our project, we took advantage of each of the many devices we have been learning about through this course of this semester's class. To control the maze, we required use of one STM32 Discovery board, connected to one Raspberry Pi Zero W for wireless communication to one Raspberry Pi 3B+. This Raspberry Pi 3B+ was then connected to its personal MQTT server to broadcast communication to another STM32 Discovery board connected to another Raspberry Pi Zero W. Below, Figure 2 visually depicts all

of the devices used throughout this project. Through the use of wireless connections between the Raspberry Pi 3B+ and Raspberry Pi Zero Ws, we were able to accurately transfer the sensor data gathered by the motions of one STM32 board to be sent and translated into x and y movements within our two servos controlled by the other STM32 board.

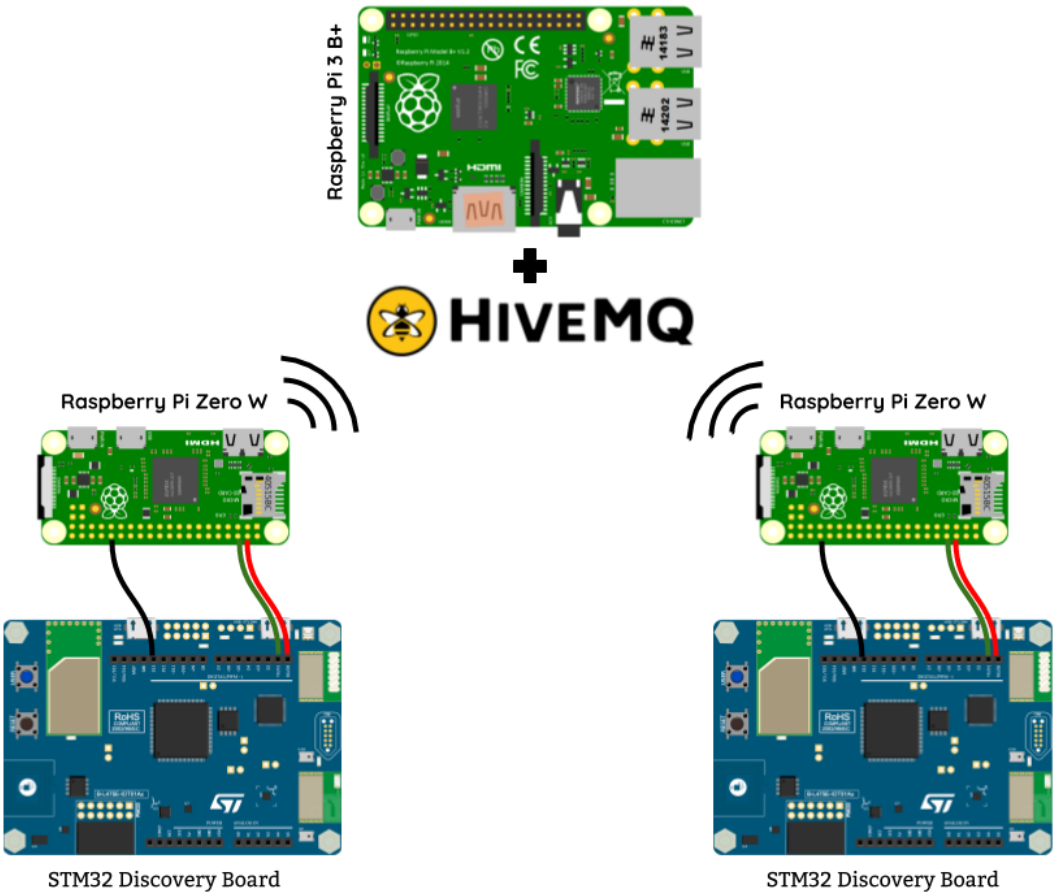

Figure 2.) Connection map of the various hardware utilized throughout this project.

In addition to the described hardware of this project, all communications were given direction to be sent, received, analyzed, and transmitted through code written in the OpenSTM32 System Workbench provided by ST Microcontrollers. Additionally we utilized the MQTT Websocket Client to be able to view the messages sent and received between the two STM Discovery boards via the two Raspberry Pi Zero Ws.

In order to manipulate pin assignment to allow for proper usage of the two servo motors via PWM output, we downloaded the STM32 CubeMX. As pictured below, this IDE allows for

the manipulation of pin assignments on the STM32 chip, as well as changing parameters in system architecture to fit the need of any project that may arise that the current, generic setup of the STM32 was not prepared for.

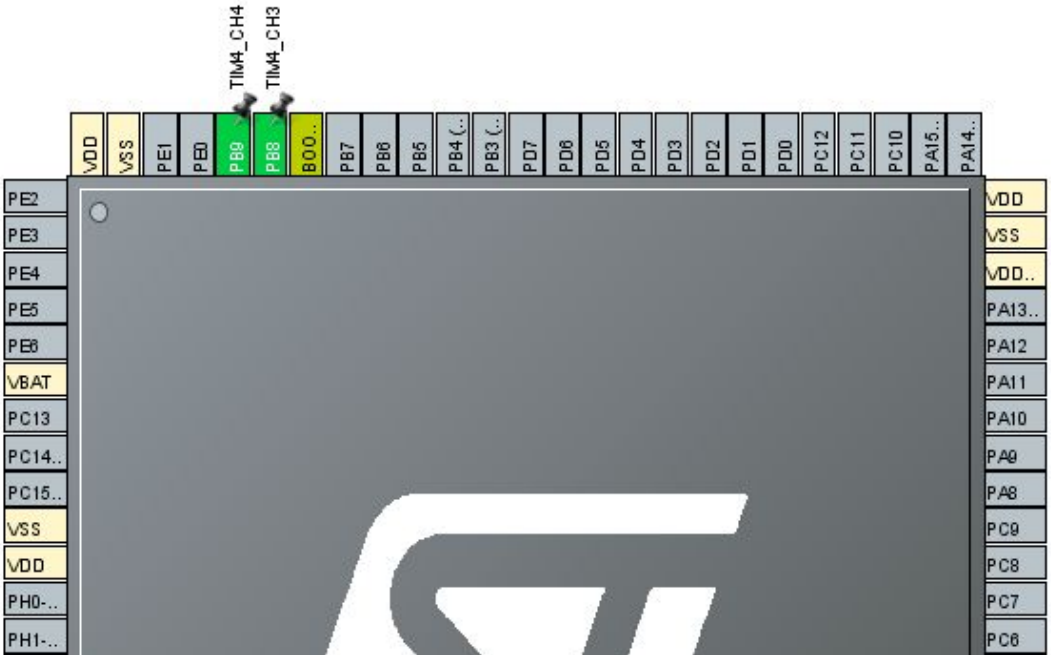

Figure 3.) STM32CubeMX Point & Click Chip Pinout Diagram with PB8 & PB9 assigned.

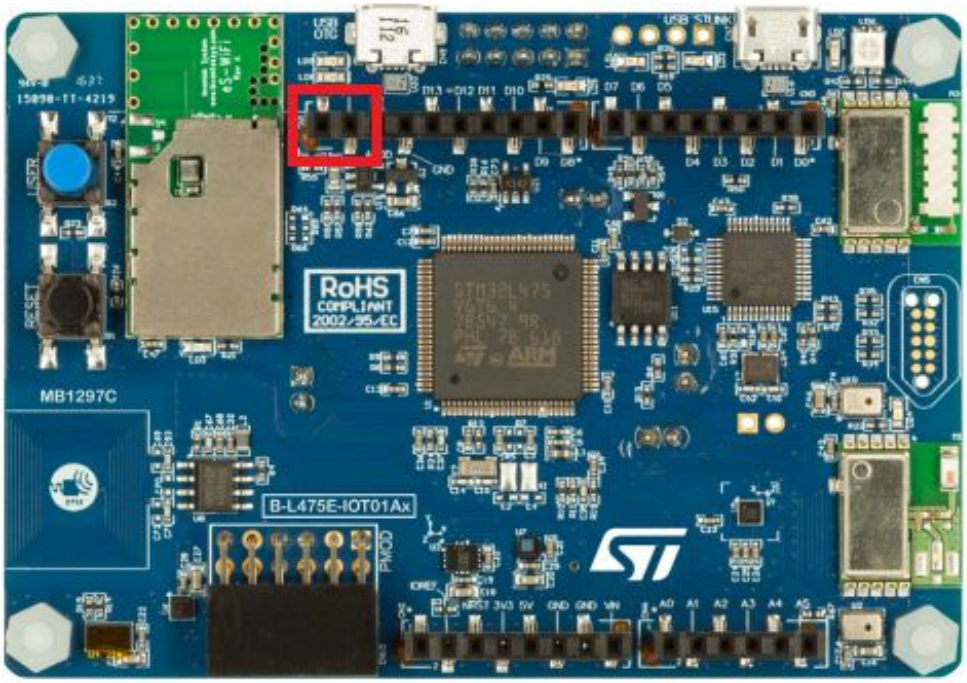

Figure 4.) STM32L475 with pins PB8 and PB9 highlighted..

In the case of this project, as pictured above, two pins needed to be dedicated to separate PWM outputs to control the two servos. Pins PB8 and PB9 were chosen, as they were not currently assigned to any process, needed or otherwise, as well as for their proximity to one another both on the chip and the board layout. These pins were given to channels 3 and 4 of Timer 4 of the main clock bus. As shown below, the hardware architecture had the 80MHz clock preset to a 4MHz configuration. In the setup of the pin assignments, a prescaler of 40 was chosen to bring the final clock speed down to 100KHz, and a divider of 2000 was applied on top of that to bring the period of both PWM pins to a 20ms increment. This exact value was required by both servos to function correctly.

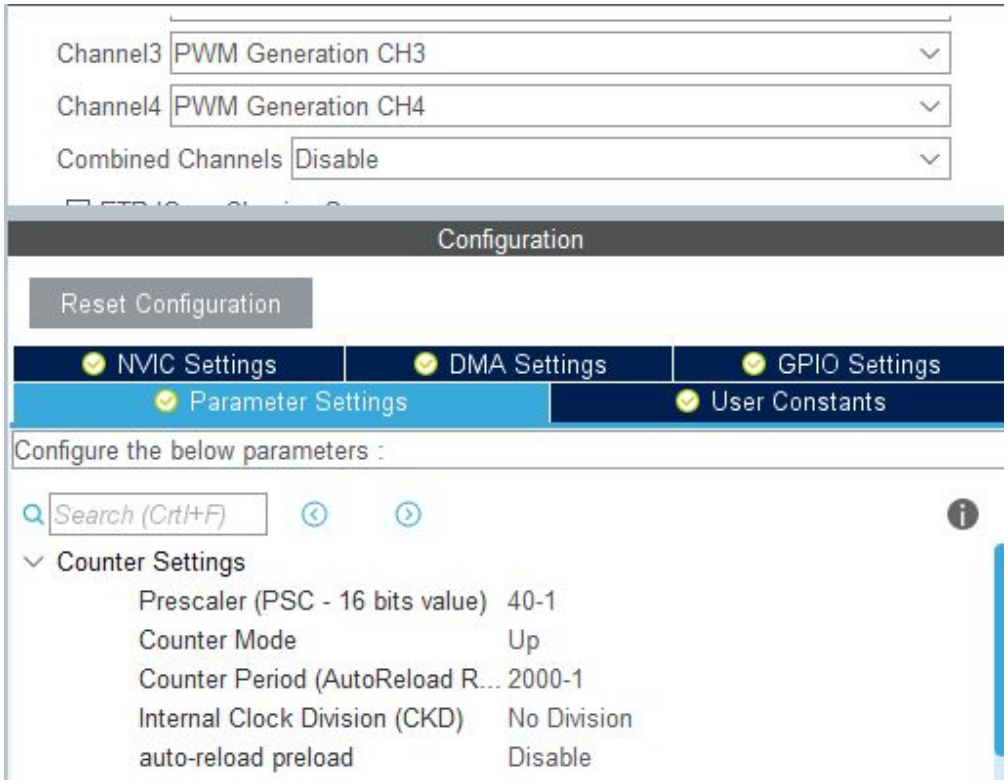

Figure 5.) Clock setup with user set prescaler and period.

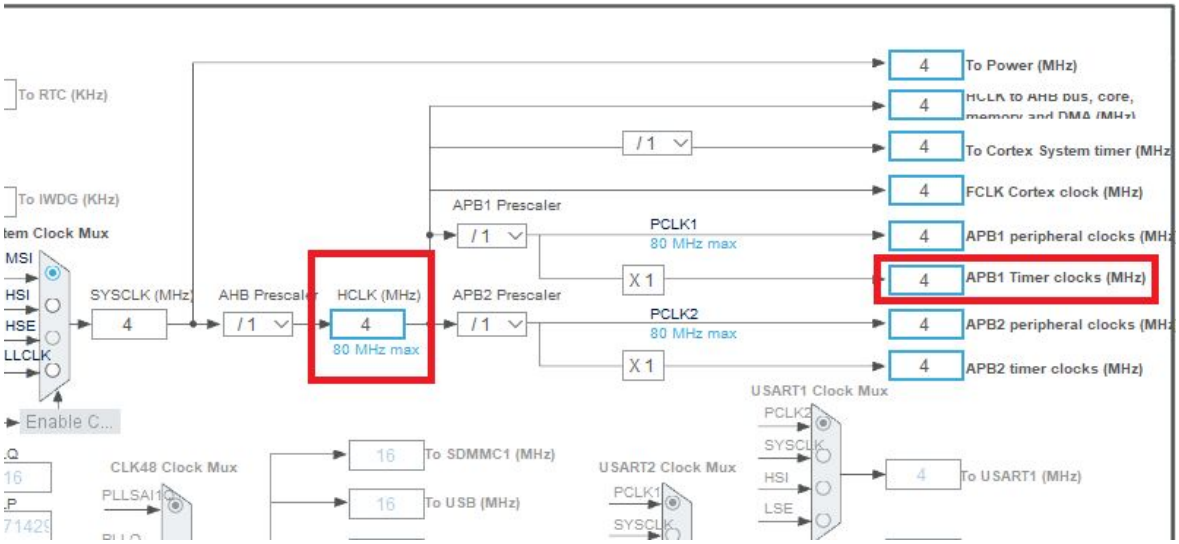

Figure 6.) The main clock set at 4MHz out of a possible 80MHz, and the bus Timer 4 is a part of, APB1.

Finally, after all of these specifications, code can be generated from this and placed into the main.c and main.h files of the project to set up the PWM pins for operation.

Design Partitioning

For this project we divided our code and hardware into the two most basic aspects of communication translation - sending and receiving. From the outset, both ends utilized the base code that had been created from Lab 7 just a few days prior, and an STM32 board and Raspberry Pi Zero W each. The intermediary was simply the Raspberry Pi 3B+ with the HiveMQ client running as the pipeline between the two.

The entirety of the communications process (barring the last step) were basically identical, following the setup of Lab 7. First, both ends would setup the WiFi connection, then the MQTT connection, then finally subscriptions. After the subscription step is where the separation occurs. The side utilizing the accelerometer and collecting the data then converted it to recognizable angle measures, and posted to the subscription “ece631/GyroMaze”. It did not look for a “SubscribedMessage” back or even a “Sent” receipt. The other end only looked for “SubscribedMessage”, as there was only one active device posting a single type of message to the only topic for this particular MQTT server.

With this modular design, each section can basically be its own device. The Raspberry Pi 3B+ would simply have to have the Wifi setup changed depending on which network the user wished to connect to, and the STM32 code for both ends would also need a similar small change when sending the Wifi connection JSON to the Raspberry Pi Zero W, as well as the IP for the current HiveMQ server. Outside of that, each could operate of its own accord, sending or receiving in any situation, provided the JSON messaging standard for all participating devices is formatted accordingly.

Design Issues\Limitations

One of the most surprising limitations to our project came from our use of sensors on the STM32. Initially we had intended on basing the majority of our code on the gyroscopic sensor of the STM Discovery board. Being that our project truly revolved around the rotational action of two servos navigating the x and y planes, we thought the gyroscope would provide the best return values on this type of movement. However, during the initial tests of the provided gyroscope code from the STM32 library, we quickly came to realize the feedback information the gyroscope sensor had to offer was a bit more than what was required of our simple two axis maze.

After reviewing the gyroscope code from the STM library, we found it useful to narrow down a list of requirements we determined were necessary to properly communicate the current corresponding angles of the STM board to translate proper motion shift to both of our servos. The biggest data point we found was necessary for movement shift of our servos was simply the directional angle measurements from point zero every time the STM32 board was shifted. Although linear values like angles were offered from the gyroscopic sensor code, we observed that they were integrated across the time the movement occured to offer additional feedback like the speed and acceleration of the given change. This kind of detailed analysis could have a variety of complex applications, but within our project the additional information just hindered our course of data extraction. We calculated that in order to separate angular measure from the final value returned from the STM code, our code would require a rather intricate system of derivatives to produce a basic linear value transferrable to servo x and servo y's movements.

In addition to this added complexity, we found that if the STM Discovery board was left suspended or resting in a tilted position, output values would begin to zero out as active momentum decreased. Since we intended to gather a sampling rate of between 0.5 seconds and 0.1 seconds during testing and final execution, we knew that these kinds of automated value changes would cause undesirable consequences to the accuracy of the motion detection of our code that could produce invalid directional changes of our servos that would result in unexpected movement of our maze.

At suggestion from our fellow peers, we decided to change focus to utilizing the STM board's built in accelerometer. Although the accelerometer focuses to reading changes in gravitational acceleration, we quickly found it to be a useful tool for measuring the directional tilt of the STM Discovery board. With the given value from the STM32's provided accelerometer code, we were able to do a few simple calculations to gather one linear angular measurement given in terms of degrees. Taking this manipulated value from the accelerometer, we were easily able to transfer our measurements to and from each axis' servo. In addition, because these values were static, we did not need to perform any additional checks to verify that each movement sent was a valid physical motion performed on the STM32 board itself.

Additionally, a majority of the time allotted by the class to work on the final project and report was actually spent on debugging the final lab which included setting up wifi, the MQTT connection for the Raspberry Pi Zero, collecting and then posting to the right subscriptions. As this lab had different setbacks for each team member, the completion at different times hindered the teams ability to focus efforts solely on the final project, greatly slowing the overall progress of the entire maze. After the completion of said lab, the work rate for the final project increased exponentially.

Testing

Testing (post Lab 7 completion) was done in 4 steps. Each done its own fully respective manner so as to eliminate it as a possible cause of error in the steps after it. These steps are as follows: first was collecting the accelerometer data, and converting it to recognizable values, second was packaging it a compatible JSON message, third was unpacking said JSON for use in the PWM effect, and fourth was setting up the PWM.

Collecting the accelerometer data was as simple as pulling from functions included in the Utilities section generated when a new project is created. The values of these initial inputs were tested, and were revealed to give values from approximately -1000 to 1000. After converting these values using the equation ((180/2000)*(data+1000)), and testing this new output, values from 0 to 180 were produced giving appropriate angle measures. Step 2 involved a little creative JSON thinking for JSON packing. Upon initial tests, it was discovered a JSON publishing message with two key value pairs in the second "Message" section was not an appropriate format. Through further run throughs, it was discovered the two initial X- and Y-axis angles needed to be packed in their own JSON string first. Then, the newline character had to be removed from that string, and then that could finally be put into the publishing JSON for transfer. The third step was simply unpacking that JSON into two separate values. This was confirmed by simply re-sending the same values back to another Topic from the receiving STM board. The final step was nearly completed, but fell short by a small margin in the setup phase, and wasn't able to receive adequate testing. If it were to have occured, this would have included the PWM and servo setup, which would have included intensive testing for correct PWM period and duty cycle, as well as fine tuning of servo angles and applied voltage from an outside source. The final section of this step would then be testing for the appropriate rate of X and Y angle data being sent to ensure a smooth responsiveness of the maze's movement without overloading the MQTT client and causing system failure.

Conclusion

After achieving solutions to the above setbacks faced, the team was able to prioritize and produce the majority of the data collection and communication aspects of the project to gather a functioning program for the maze. The only improvements we believed that could be met in this side of the project was through utilization of another MQTT client that could handle more frequent message transactions between client and node without timeout. With this improvement we believe that would could better increase the functionality of our code to minimize delay between movement of the STM32 board and its corresponding actions to the physical maze. Furthermore, from suggestion of Professor Herrmann, we believe we could further the usability

of our code be producing a multitude of different subscriptions for our data to allow variability to each STM board. In doing so, we could avoid data cross translations by offering select output to individual topics for the STM boards to selectively subscribe. This addition could furthermore add in future expansion of our project to allow in server or added node to easily access any array of data output we produce.

As for the physical aspects of our project, we believed that given added production time or decreased debugging time, we could have better designed the various printed components of our maze to produce a more effective assembly. With these improvements in mind, we believe the final project could be an even greater illustration of the skills we developed throughout the course of this semester.

Appendix

Link to code in course GitLab:

- https://gitlab.beocat.ksu.edu/s19.b.o.b

Resource references:

- https://www.instructables.com/id/3D-Printed-Maze-Controlled-by-Your-Android-Device/