Lab 2 Report

Kerollos Lowandy, Pablo Ruiz, Mohammed Khassim
Completion Date: 03/23/2021
Microprocessor II and Embedded System Design
Lab 2: Game Control with Joystick, Gyro and Accelerometer

1.INTRODUCTION:

In this lab, we were tasked with both a joystick and IMU controller system to control a Snake python game. Since the python game operated with the “W”, “A”, “S”, and “D” keys on the keyboard, our task was to make the output of the joystick and IMU write the desired letter onto the serial bus, so that the game would function as expected. We also needed to find a way of triggering a buzzer to go off whenever the game saw an apple as eaten. As part of the graduate section of this lab, we had the additional task of having the points per apple increase to 20 points if a “shaking” was detected before eating an apple. If this shaking was detected, the apple’s color would change to yellow instead of red.

2. DESIGN:

2.1. HARDWARE:

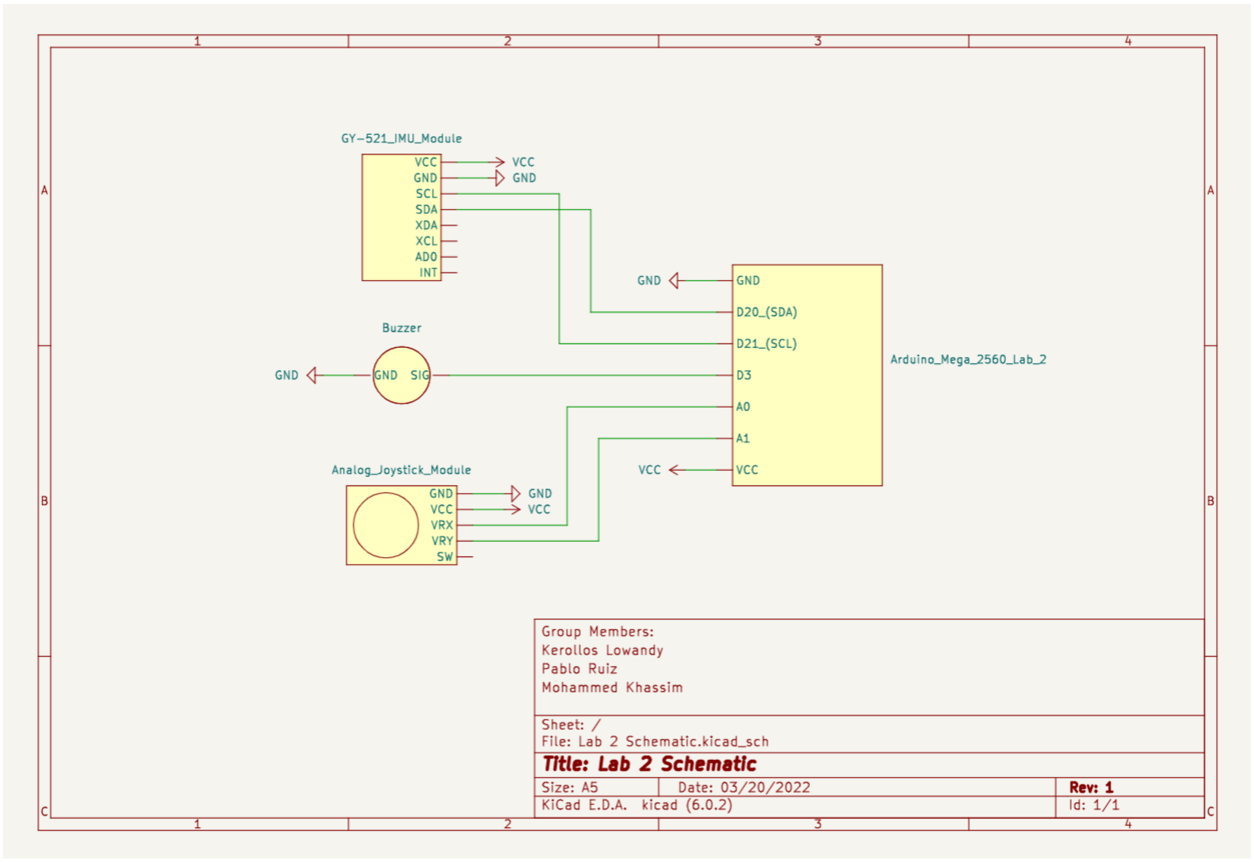

Figure 1. Lab 1 electrical schematic.

The hardware setup for this lab was relatively simple and required only five I/O ports on the arduino. The buzzer was connected to the D3 header which is capable of outputting PWM signals, all that is necessary to drive the buzzer. The joystick is a simple combination of two potentiometers and therefore only needs two connections to the arduino ADCs through the A0 and A1 headers apart from the VCC and GND connections. Since the switch function of the joystick was not used in this lab tha pin was not connected to the arduino. The IMU only needed an I2C bus connection to the designated arduino mega2560 I2C pins (D20 and D21) apart from the VCC and GND connections. These two wires were sufficient for communication since the MPU-6050 has an onboard processor and can thus communicate digitally with the arduino. The other pins present on the IMU were not necessary in our application since we only have one I2C slave device on the bus and we are not using interrupts.

2.2. SOFTWARE:

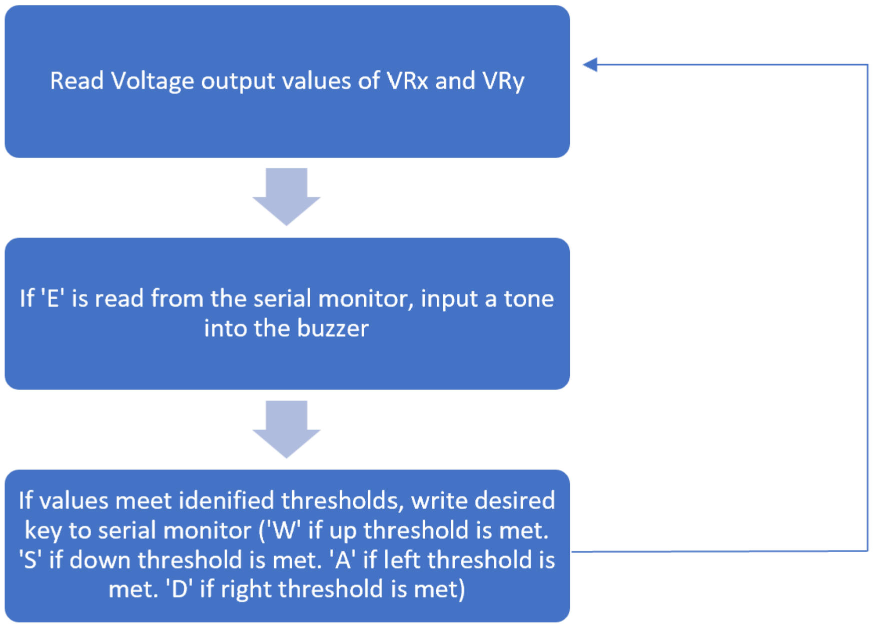
Read Voltage output values of VRx and VRy
If 'E' is read from the serial monitor, input a tone into the buzzer
If values meet idenified thresholds, write desired key to serial monitor ('W' if up threshold is met. 'S' if down threshold is met. 'A' if left threshold is met. 'D' if right threshold is met)

Figure 2. Joystick Program flow chart.

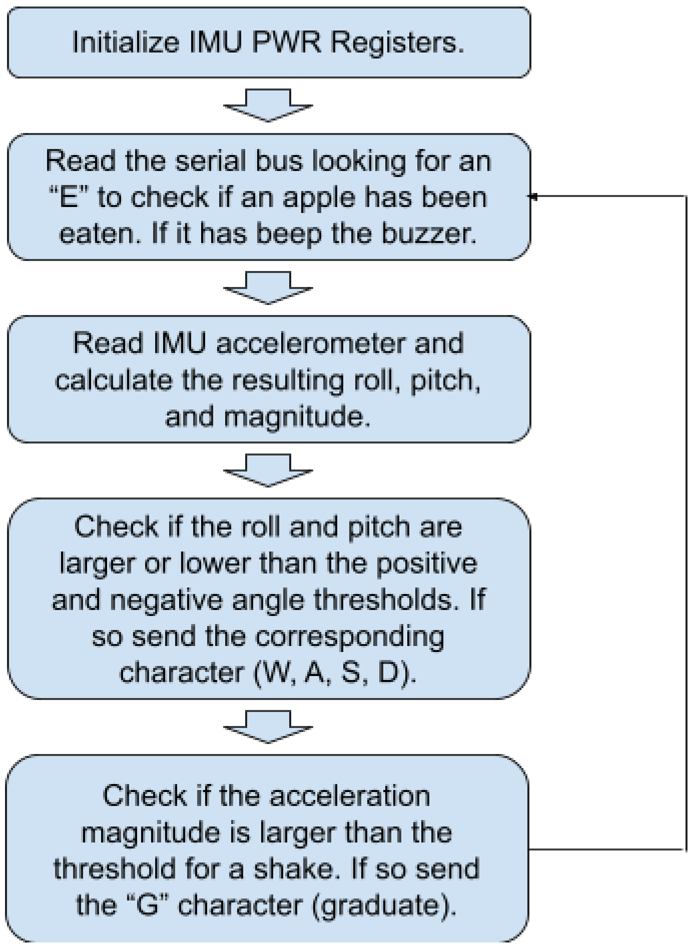
Initialize IMU PWR Registers.
Read the serial bus looking for an "E" to check if an apple has been eaten. If it has beep the buzzer.
Read IMU accelerometer and calculate the resulting roll, pitch, and magnitude.
Check if the roll and pitch are larger or lower than the positive and negative angle thresholds. If so send the corresponding character (W, A, S, D).
Check if the acceleration magnitude is larger than the threshold for a shake. If so send the "G" character (graduate).

Figure 3. IMU program flow chart.

The software for this lab was simple once the logic was understood. The Snake game operated through the keyboard buttons and when pressed, the keyboard buttons are pushed to the serial port of the computer. This is why we had to change the serial port defined in the code, which was for Mac users, to COM8, one of the serial ports usable for any Windows PC. With this, our code was able to communicate back and forth with the Snake game code.

The snake code had to be slightly modified so that there was a flag set whenever an apple was eaten. This was how we were able to trigger the buzzer with our arduino code. This was done by writing in the Snake game python code to print 'E' to the serial monitor anytime if an apple was eaten. At the same time, the arduino code was told to read the serial monitor for 'E'. If it was able to detect 'E' then it was to send a tone into the pin that the buzzer was connected to.

Writing the commands for the snake to move was very similar conceptually. With this however, the arduino was reading the output of the VRx and VRy pins of the joystick, or the output of the IMU through the I2C bus. The default sensitivity of the IMU was used, which is ±2g. After testing the output values, we were able to determine the threshold values necessary for each direction. With that, we set conditions for each "keypress" to be written to the serial monitor, based on the threshold values defined. Since the Snake code reads the serial monitor for normal keypresses, it is able to take in our encoded keypresses and move the snake. For the joystick, this was the entirety of our code.

For the IMU however, we recorded the raw values from the accelerometer in the X, Y, and Z axes and then used trigonometry to calculate both the roll, pitch and resulting magnitude of the acceleration vector. Thus the resulting units for the roll and pitch were degrees and the units for the magnitude was Gs. Consequently, the thresholds could be set in the same standard units which made classifying the current position of the IMU much simpler. To detect when the IMU is shaken a simple threshold was set for the magnitude of the acceleration vector. Since standard gravity at sea level is 1g we selected 1.5g as a threshold which according to our testing is only achieved when intentionally and violently shaking the IMU. Once a shake is detected the character "G" is sent through the serial bus to the computer. The Snake game reads this character from the serial bus and sets a flag which changes the color of the snake food to gold and doubles the points obtained by eating the food to 20. Once the food has been eaten it resets the flag and the color and value of the food return to its standard values.

Github Link:
https://github.com/tame-potato/Micro-2-Lab-2

3. RESULTS:

Joystick Demo Video Link:
https://studentuml-my.sharepoint.com/:v:/g/personal/pablo_ruiz_student_uml_edu/EY9TLskrZL9HpS9TCKlVyUcBK2HRgC4dZWigMDTyOgCLhQ?e=1IuqWC

IMU Demo Video Link:
https://studentuml-my.sharepoint.com/:v:/g/personal/pablo_ruiz_student_uml_edu/ESK2yu0IGE1JqcY86HtGRy8BEg597HruYoGfjRFIU70TDA?e=8qIjdp

4. DISCUSSION:

The main problem that arose in our design process came in setting up the IMU. The joystick code was very straightforward in its setup once we knew that we were writing to the serial bus. We did have to find out that our Windows computers had a different serial port than the python code defined and change it to the correct port, but that was not much of a problem. In order for the IMU to work however, we had to first find the I2C address of the MPU-6050 and then find the correct register to write to in order to initialize and reset the IMU as well as the correct registers to read in order to obtain the accelerometer data. After finding said register addresses in the MPU-6050 spec sheet, the rest was able to work smoothly. We knew how to read from the arduino’s internal ADCs for VRx and VRy in order to get the thresholds for going in any direction, or “pressing a key.” We knew to place a flag in the python code to trigger the bonus points and the buzzer, for which we used the same tone function as in lab 1. We learned of the capabilities of serial port communication between the arduino and an application running on a different system and could now apply this knowledge in future applications of software/hardware communications.