CENG2400 Group 13 Final Project Report

Chan Eugene 1155193465, Chan Yik Kuen Eric 1155192801, Lei Hei Tung Hazel 1155194969

Table of Contents

Division of Labor

- Master Code: Eugene, Eric
- Slave Code: Eugene, Hazel
- Report: Eugene, Eric, Hazel

Introduction

The CENG2400 final project required us to implement a gyroscope-controlled laser turret, where orienting the gyroscope in different ways results in the laser aiming in corresponding directions. Our group aimed to implement the laser turret and make it as responsive as possible, hoping to hit as many targets as possible on the demo day.

Generally speaking, as long as we utilize the functions provided in the demo code, the laser turret will function to an acceptable level. The workflow of the program is as follows:

1) Master end: we take raw data like `fGyro[]` and `fAccel[]` from the MPU6050, convert them into usable data, and calculate the necessary angle to send to the slave program through the UART interface.
2) Slave end: we take the calculated angle from UART and send it to the servo with PWM.

How we implement the workflow can vary. In the following sections, the report will detail the possible approaches, how those approaches work, and how we decided on the approach to be used in the final product. It will also review some of the issues we faced when implementing the selected approaches.

Master Code

The master's role is to calculate the necessary angle to output to the slave. The MPU6050 outputs two useful sets of data, those being `fGyro[3]` (the angular velocity of the gyroscope) and `fAccel[3]` (the linear acceleration). Provided with this data, we devised a few ways to calculate the required angular displacement.

Methods:

1) Only using `fGyro[3]: angle[i] = prevAngle[i] + fGyro[i] *` **`dt`**
2) Complementary filtering:
   Complementary filtering is slightly more complex than just using fGyro[3]. In theory, it provides a more accurate calculation for the required angle in the long term. It works by using `fAccel[3]` to calculate a small portion of the required angle. How small the portion is is decided by the size of the **complementary coefficient**. Here is the example equation:

   ```
   angle[i] = coeff * (prevAngle[i] + fGyro[i] * dt) + (1 - coeff) *
   angularAccel[i]
   ```

   *For both approaches, **dt** is used in theory to do *pseudo-integration*, however, in practice, **dt** can be used to control the sensitivity of the servo (i.e. degree of hand movement corresponding to servo movement). In the final code, **dt** was tailored such that the shooter could aim the turret without being 100% precise.

   `angularAccel` has to be calculated separately, as the raw data provided by `fAccel[i]` is **linear**, and cannot be used for our calculations. Below is the equation needed to calculate angular acceleration:

   ```
   angularAccely = atan2(-1*fAccel[0], sqrt(pow(fAccel[1], 2) +
   pow(fAccel[2], 2))) * 180 / M_PI
   angularAccelx = atan2(fAccel[1], sqrt(pow(fAccel[0], 2) +
   pow(fAccel[2], 2))) * 180 / M_PI
   ```

   *It is important to note that angular acceleration for the z-axis (yaw) cannot be calculated accurately due to how the accelerometer for the MPU6050 works. In short, the accelerometer works by sampling the strength of the earth's gravity. Therefore, there is no meaningful information that can be calculated for the z-axis

In practice, the complementary filter is not perfect. Our group encountered an issue with pitch when using the complementary filter. As mentioned, the complementary coefficient determines how much acceleration affects the final calculation. **There exist instances where angular acceleration is 0 and the angle is constantly multiplied by the complementary coefficient which is < 1**. As a result, whenever the gyroscope is pointed upwards and held in place, the laser turret resets to its original position within a few seconds. This makes it impossible to aim at targets that are in elevated positions. We may have made mistakes in our code that caused this, however, in order to have a turret that can maintain reasonable accuracy for the demo day, our group decided to take the simpler approach of using just `fGyro[]` to make our calculations.

Using just `fGyro[]` did not come without its own set of problems, however, as it is indeed less accurate than using a fully functioning complementary filter. During initial testing, our group did not find significant issues with the accuracy of the laser, but during the demo day, we realized the noise from using `fGyro[]` alone caused our calculated angle to decrease every few seconds. This created a form of "reverse recoil" that made it hard to aim at small targets for extended periods. A video of this phenomenon can be seen here.

From the video, we can see that both pitch and yaw decrease despite the gyroscope being stationary. Unfortunately, we could not find a solution to this issue, and the only way to lessen the effects is to increase the delay of the program.

Summarising the part for calculating the required angle, our group decided to use just `fGyro[3]` for calculations, resulting in an imperfect but certainly acceptable product.

After calculating the angle, the next step is to send the results to the servo side. Initially, we believed that we could simply send the angles in sequence, separated only by a "," and ended in "\n" (i.e. "100,60\n"), however, this caused a few problems on the servo end. In some cases, there are faults that result in **sequences merging** (e.g. "100,60100,60") this will cause the laser to shake uncontrollably. Details of the problem will be discussed in the servo section, as most of the issues stem from there. For now, we can say that adding a starting character "#" helped with resolving this issue.

The algorithm our group devised for the UART interface is somewhat stupid. Our algorithm works on the preface that UART only accepts ASCII characters. Seeing as the angles calculated are integers, our first thought is to simply convert each digit into its ASCII equivalent, and send it to the slave in a packet with the following format:

```
[#,digit1,digit2,digit3,’,’,digit1,digit2,digit3,\n]
```

*Each digit undergoes: digit + ‘0’ to make it a character
*Additional information regarding this format can be found in the servo section
*This is also the packet that made it into the final code

At first glance, this seems to be a *reasonable* way to send packets. However, to split the integer into 3 digits, we have to run a for loop to calculate each digit. This creates a lot of overhead for the MPU and is incredibly inefficient (even though the difference in practice is negligible).

```
int i;
for (i = 0; i < 3; i++) {
      YawToSend[i] = RoundedYaw % 10 + '0';
      PitchToSend[i] = RoundedPitch % 10 + '0';
      RoundedYaw /= 10;
      RoundedPitch /= 10;
}
```

After the demo day, our group has learned of a different way to send packets from another group, and that is to send packets in this format:

```
[#‘integer1’‘integer2’]
```

Since ASCII ranges from 0 - 256, it fully satisfies the angle requirements for this application without needing a complex conversion algorithm. Seeing as copying their approach is plagiarism, we decided, in the end, to continue with our naive approach. This section is simply to note that we know of another possible way to improve the program.

Slave Code

The task of the slave end is simple. It takes the calculated angle from the master end through the UART interface and uses it to turn the servos, thus aiming the turret.

As mentioned, we took the naive approach when it comes to UART communication, and we built an algorithm that converted the individual characters back into usable integers.

```
yawBT = 180 - ((input[0] - '0') * 100 + (input[1] - '0') * 10 + (input[2] - '0'));
pitchBT = (input[4] - '0') * 100 + (input[5] - '0') * 10 + (input[6] - '0');
```

Two approaches come to mind when it comes to controlling the servo itself:

1) incrementing/decrementing the angle by 1 until it reaches the desired angle
2) Directly plugging the desired angle into the PWM function

The first approach is one we learned in a previous course CENG2030. This approach heavily limits the rate at which the servos can rotate in exchange for extraordinary stability. The servo can only rotate as quickly as the system delay allows it, and causes extremely high latency between the gyroscope moving, and the laser actually reaching the desired position. Example code:

```
if(ui8Adjust_vertical < z_val) {
      ui8Adjust_vertical++;
      if(ui8Adjust_vertical > 141) {
          ui8Adjust_vertical = 141;
      }
       PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, ui8Adjust_vertical *
ui32Load/1000);
}
```

The second approach is simply using the demo code provided, inserting the desired angle directly into the function. With this approach the servo is allowed to rotate as quickly as it can, creating a smooth and responsive experience for the shooter. The only problem with this approach is that it is susceptible to a lot of human error, since any small movement will be reflected instantly. Example code:

```
yaw_duty_cycle = angleToPWMDutyCycle(yawBT);
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0, PWMGenPeriodGet(PWM1_BASE,
PWM_GEN_0) * yaw_duty_cycle);
pitch_duty_cycle = angleToPWMDutyCycle(pitchBT);
PWMPulseWidthSet(PWM1_BASE, PWM_OUT_1, PWMGenPeriodGet(PWM1_BASE,
PWM_GEN_0) * pitch_duty_cycle);
```

Our final decision depended on the circumstances of the demo. Since the demo is timed and there are many targets, we decided responsiveness was more important than stability. We opted to use the second approach.

As mentioned in the gyroscope section, our implementation of UART communication has several faults that cause the servo to shake uncontrollably. Part of the problem was the format of the packet we sent, but the major issue lies in the way we get characters from UART. Here is an excerpt of the code used to get characters from UART.

```
while(UARTCharsAvail(UART5_BASE)) {
    char ch = UARTCharGet(UART5_BASE);
    if (ch == '\n') {
        UARTCharPut(UART0_BASE, ch);
        break;
    }
    input[i++] = ch;
    UARTCharPut(UART0_BASE, ch);
}
input[i] = '\0';
```

The problem with the above code lies in `UARTCharsAvail`. Using this as the condition for the while loop creates an edge case where all characters are not in UART before we attempt to extract them. For example, the master end can send a packet “100,60”, as the slave end continuously pops the characters, it is possible that available characters are clear before the final “\n” is sent. This causes the **sequence merging** issue mentioned earlier. In the final version of the code, we used `while(1)` as the main loop, and allowed the loop to break only when “\n” is met. Additionally, the starting character “#” is also a condition to begin reading. These changes ensured no information was lost in transition and completely fixed the servo shaking problem. Revised code:

```
while (1) {
        char ch = UARTCharGet(UART5_BASE);
        // using # as starting character is necessary to make sure each
received packet is in order with no corrupt data
        if(ch == '#'){
            receiving = true;
            UARTCharPut(UART0_BASE, ch);
        }
        else if(receiving){
            if (ch == '\n') {
                UARTCharPut(UART0_BASE, ch);
                receiving = false;
                break;
            }
            input[i++] = ch;
            UARTCharPut(UART0_BASE, ch);
        }
    }
```

Conclusion

For this project, we believe that we have succeeded in creating a responsive laser turret. Even though there are imperfections in our implementation, we could still hit all of the targets within allotted time.

References

1. *Debra. (2013, March 31). Gyroscopes and accelerometers on a chip – Geek Mom projects.* [*https://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/*](https://www.geekmomprojects.com/gyroscopes-and-accelerometers-on-a-chip/)
2. *Shane Colton. (2007, June 25). The Balance Filter A Simple Solution for Integrating Accelerometer and Gyroscope Measurements for a Balancing Platform* [*https://www.geekmomprojects.com/wp-content/uploads/2022/03/filter.pdf*](https://www.geekmomprojects.com/wp-content/uploads/2022/03/filter.pdf)