Lab Assignment 5

Jack Fenton
fenton.j@husky.neu.edu
Nicholas Abadir
abadir.n@husky.neu.edu

Submit date: 3/12/2020
Due Date: 3/12/2020

Abstract

In this lab we used object oriented programming and makefiles to help organize our code and be able to modify code while still having it work with the original header file, with relative ease. We also used shell scripts to set up the addresses of the robotic arms in the root file so that we would be able to access their information using the sudo command when running. Furthermore, this is our most complex embedded design and our first actually controlling physical motion of machinery.

Introduction

The goal of this lab was to become more comfortable with makefiles in both the creation of the separate files and the actual makefile itself. As well as makefiles, we introduced the more physical aspect of embedded design and moved away from simply controlling LEDs and reading values. These makefiles help us a good deal when it comes to the modularity of the program even more so than simply using classes and structs. The reason why the makefiles are so useful, is if I want to change how the program behaves, most of the time I will simply have to edit one of the files. As well as that, everything becomes easier to find as the class definitions will be held in the Class.h and the functions of the class will be held in the Class.cpp, allowing me to use classes without all the clutter in the `main.cpp` file.

Lab Discussion

In the prelab, we were asked to define a PWM signal, which is a process that transforms the width of a pulse so as to control the power it outputs. A duty cycle is then the ratio between the pulse duration and the signal period, which allows us to give a signal at a speed which the robotic arm can comprehend. We then created a function to calculate the PWM signal to control the servo motor (degreeToOnDelay()).

```
float degreeToOnDelay(float degree)

{

   if(degree<10)degree=10;

   if(degree>170)degree=170;

   return (degree*10)+600;

}
```

This returns the PWM by increasing the degree size by 10 and adding by 600.

We also learned some of the basics for shell scripts and learned the uses of the redirection operators `" < " " > ".` The < operator allows us to read characters, lines, and scripts in that order, whereas the > allows us to modify the file itself by inserting a value or number.

The materials used in this lab were a robotic arm with 5 servos at the base, bicep, elbow, wrist and gripper. A ZEDboard, and a computer in which to code onto the ZEDboard on that would then communicate to the robot arm through the JE Pmod port.

Results and Analysis

Lab 5.5 Automating the port setup

In this assignment we were given code to set up the ports for the robotic arm on the ZEDboard so as to automate the process.

```
FILEPATH='/sys/class/gpio'
echo 13 > $FILEPATH/unexport 2>/dev/null
echo 10 > $FILEPATH/unexport 2>/dev/null
echo 11 > $FILEPATH/unexport 2>/dev/null
echo 12 > $FILEPATH/unexport 2>/dev/null
echo 0 > $FILEPATH/unexport 2>/dev/null
echo 13 > $FILEPATH/export
echo 10 > $FILEPATH/export
echo 11 > $FILEPATH/export
echo 12 > $FILEPATH/export
echo 0 > $FILEPATH/export
echo out > $FILEPATH/gpio13/direction
echo out > $FILEPATH/gpio10/direction
echo out > $FILEPATH/gpio11/direction
echo out > $FILEPATH/gpio12/direction
echo out > $FILEPATH/gpio0/direction
```

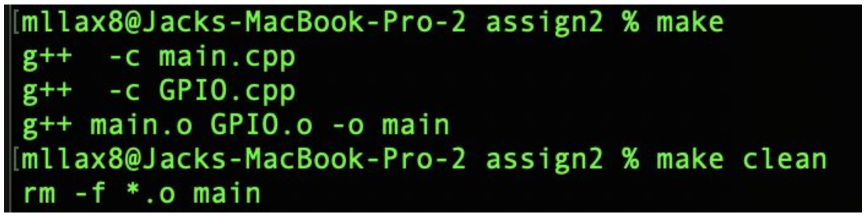
The first part where it unexports, redirects any errors in the inputs and has it thrown out into a null directory which essentially gets rid of the standard error. The second part where it exports, sets up a directory to take an input from the computer in case we want to move the robot and has it sends it to the path as 13, 10, 11, 12, and 0 and then the echo out sends out to the file path specified.

Lab 5.6 Controlling the Servos

Code from Makefile for `assign2`

```
CLFAGS = -g -Wall
CC = g++

PROJECT = main
REF1 = GPIO

$(PROJECT): $(PROJECT).o $(REF1).o
   $(CC) $(PROJECT).o $(REF1).o -o $(PROJECT)

$(PROJECT).o: $(PROJECT).cpp $(REF1).h
   $(CC) $(CFLAGS) -c $(PROJECT).cpp

$(REF1).o: $(REF1).cpp $(REF1).h
   $(CC) $(CFLAGS) -c $(REF1).cpp

clean:
   rm -f *.o $(PROJECT)
```

The Makefile is written to be easily changed between the different assignments. The variables `CFLAGS` and `CC` are used as the placeholders for declaring the language being built in and the way it should deal with exceptions. The end level file for the project is called `PROJECT`, which is used to make a `.o` file with the same name using the `.cpp` file with that name. The first, and in this case only, file referenced by the end file is called `REF1`, which is made into a `.o` file using its respective `.h` and `.cpp` files. The name of the files can be changed by just changing the variables `PROJECT` or `REF1` without having to change anything else in the file.



```
[mllax8@Jacks-MacBook-Pro-2 assign2 % make
g++   -c main.cpp
g++   -c GPIO.cpp
g++ main.o GPIO.o -o main
[mllax8@Jacks-MacBook-Pro-2 assign2 % make clean
rm -f *.o main
```

Lab 5.7 Controlling the robotic arm position

A GPIO object is created for each motor, each named after their respective place in the arm. The user written function `isInt()`, Appendix 1, is used to filter out bad inputs from the user for all the questions that need to be asked. The angle, in degrees, is then converted to an on delay by the function `degreeToOnDelay()`. The switch case is then used to write to the specified motor.

```
   int c_case=0;
   int angle;
   int delay;

   GPIO Base(13);
   GPIO Bicep(10);
   GPIO Elbow(11);
   GPIO Wrist(12);
   GPIO Gripper(0);
   while(c_case!=6)
   {
       c_case=isInt(1,6, "Select a Servo\n 1:Base\n 2:Bicep\n 3:Elbow\n 4:Wrist\n
5:Gripper\n 6:Quit Program\n");
       if (c_case==6) break;
       angle=isInt(0,180, "Pick an angle 0-180: ");
       delay=degreeToOnDelay(angle);

       switch(c_case)
       {
           case 1:
               //Change angle of Base
               Base.GeneratePWM(20000,delay,200);
               break;
           case 2:
               //Change angle of Bicep
               Bicep.GeneratePWM(20000,delay,200);
               break;
           case 3:
               //Change angle of Elbow
               Elbow.GeneratePWM(20000,delay,200);
               break;
           case 4:
               //Change angle of Wrist
               Wrist.GeneratePWM(20000,delay,200);
               break;
           case 5:
               //Change angle of Gripper
               Gripper.GeneratePWM(20000,delay,200);
               break;
       }
   }
```

For the function `GeneratePWM(int,int,int)`, the first argument will always be 20000, period of the servo motors being used, the second argument determines the length of the on pulse which is specifies the angle, and the third argument is the number of periods the motor will be controlled for. 50 periods make up 1 second. A pulse of 600 will be 0º and a pulse of 2400 will be 180º, so the difference of 100 in pulse will be 10º.

Lab 5.8 Controlling the robotic arm speed

Most of the structure of the code remained the same as in assign3, but now the user is prompted to give a starting and ending angle as well as a speed.

```
int c_case=0;
int stAngle, enAngle, chAngle;
int stDelay, enDelay;
int speed, time, periods;

GPIO Base(13);
GPIO Bicep(10);
GPIO Elbow(11);
GPIO Wrist(12);
GPIO Gripper(0);
while(c_case!=6)
{
   c_case=isInt(1,6, "Select a Servo
\n1:Base\n2:Bicep\n3:Elbow\n4:Wrist\n5:Gripper\n6:Quit Program\n");
   if (c_case==6) break;
   stAngle=isInt(0,180, "Pick a starting angle 0-180: ");
   enAngle=isInt(0,180, "Pick an ending angle 0-180: ");

   speed=isInt(0,180, "Pick a speed deg/s: ");

   chAngle=abs(enAngle-stAngle);

   time=chAngle/speed;
   periods=50*time;

   stDelay=degreeToOnDelay(stAngle);
   enDelay=degreeToOnDelay(enAngle);

   switch(c_case)
   {
       case 1:
           //Change angle of Base
           Base.GenerateVariablePWM(20000,stDelay,enDelay,periods);
           break;
       case 2:
           //Change angle of Bicep
           Bicep.GenerateVariablePWM(20000,stDelay,enDelay,periods);
           break;
       case 3:
           //Change angle of Elbow
           Elbow.GenerateVariablePWM(20000,stDelay,enDelay,periods);
           break;
       case 4:
           //Change angle of Wrist
           Wrist.GenerateVariablePWM(20000,stDelay,enDelay,periods);
           break;
       case 5:
           //Change angle of Gripper
           Gripper.GenerateVariablePWM(20000,stDelay,enDelay,periods);
           break;
   }
}
```

`chAngle` is the magnitude of the change between the starting and ending angle, which is then used to find the amount of time required to rotate at the given speed, `time`. The number of 20ms periods that pass is `50*time`, stored in the variable `periods`. The actions for the specified motor are sent to the method `GenerateVariablePWM(int,int,int,int)`:

```
void GPIO::GenerateVariablePWM(int period, int first_pulse, int last_pulse, int
num_periods)
{
   int change_pulse = last_pulse-first_pulse;
   float delt_pulse = float(change_pulse)/float(num_periods);
   float  c_pulse=first_pulse;

   for (int i = 0; i < 50; i++)
       {
               write(fd, "1", 1);
               usleep(first_pulse);

       write(fd, "0", 1);
               usleep(period - first_pulse);
       }

   for (int i = 0; i < num_periods; i++)
       {
               write(fd, "1", 1);
               usleep(int(c_pulse));

       write(fd, "0", 1);
               usleep(int(period - c_pulse));

       c_pulse+=delt_pulse;

       }

   for (int i = 0; i < 50; i++)
       {
               write(fd, "1", 1);
               usleep(last_pulse);
```

```
        write(fd, "0", 1);
                usleep(period - last_pulse);
        }
}
```

The change between the first and last pulse is type casted as a `float` so that the pulse can increase by less than 1 for each period. The middle for loop runs for the number of periods calculated in `ServoSpeed`, with the pulse, `c_pulse`, sent to the servo increasing by `delt_pulse` each time, starting from `first_pulse`. This method slowly increases the goal angle as seen by the servo, because the servo is only ever capable of being set to angle, with no regard for its path.

Conclusion

This lab allowed us to understand makefiles to a further extent working off of lab 4. The uses of these in real world applications is endless as the modularity of the program is simply too useful to pass up on. While also improving our embedded design skills, the work on the robotic arm is a sneak peek into how automated machines can work and function even though we've barely skimmed the surface of this complex field. With these two combined, it makes it easy to imagine the possibilities of how in-depthly you could control the arm with more time.

Appendix

Appendix 1: Function isInt(int,int,string)

Will accept a users input if it is an integer in the range specified, will reprompt if not accepted

```
int isInt(int min, int max, string prompt)
{
   int input=-1;
   while((input<min) || (input>max))
   {
       cout << prompt;
       cin >> input;
       if (cin.fail())
       {
           cin.clear();
           cin.ignore();
       }
       else break;
   }
   return input;
}
```