COURSEWORK 3

2016 December ITR

TEAM

Jack Galvin 1507192

Adam Green 1503487

Arthur Zargaryan 1507104

Artur Zargaryan

Report

Introduction

This project was spilt over two major parts, the first of which required us to get acquainted with the motor controller, Arduino and linear actuators and develop a predefined beat using simple programming. The second part of the course we linked a human input device with the drum, in this situation a sonar sensor was employed. The drum therefore beat at a frequency proportional to the distance of the hand from the sensor.

Part 1

Observations

The initial part of the project was simple, yet allowed us to understand the system we were working with. We notice a few interesting points highlighted below:

- The braking ability of the motor controller did not yield as good results as did reversing the direction of the actuator.
- We tested out several delays, observing that the lowest working delay was around 150ms. We also noticed that despite doubling the delay the drum did not reduce its frequency by half. We attribute this to the reaction delay of the motors.
- We experimented with the speed of the actuators, and finally concluded that the highest setting (255) was the one that yielded the best results.
- We further observed that the linear actuators had sever trouble working without an external power source. Despite working with up to 4 amps, and 12 volts we found that the best setting for the motor controller and motors were at about 12 volts and 1 amp.

We implemented 3 functions to simplify the creation of the beat:

- beatA(delay, power),
- bearB(delay, power),
- beatAandB(delay, power);

This made the construction of beats much easier. The limitation was however that the drum sticks could not beat completely independently. Non-the less we managed to create interesting beats.

Part 2

Heart Rate Sensor

Our group initially wished to implement a heart rate sensor and make the drum beat to the rhythm of the heart. We attempted to make this system work, however we came across a range of issues.

- The readings from the heart rate monitor, much like with other biometric sensors are extremely difficult to interpret. Additionally the sensor we employed was difficult to place well upon the subjects whose heart rate was to be recorded.
- The error and noise recorded was extremely high. The reading were also very inconsistent and rectifying these would have been very resource intensive. For this reason we decided to attempt a different input system and deviated into testing sonar sensors.

Sonar

Outline

The sonar would act as our input device, mapping the distance from the sonar to the frequency the drum would beat at. The closer to the sonar the hand would get the faster the drum would beat. The goal was to create as reliable and responsive a system as possible.

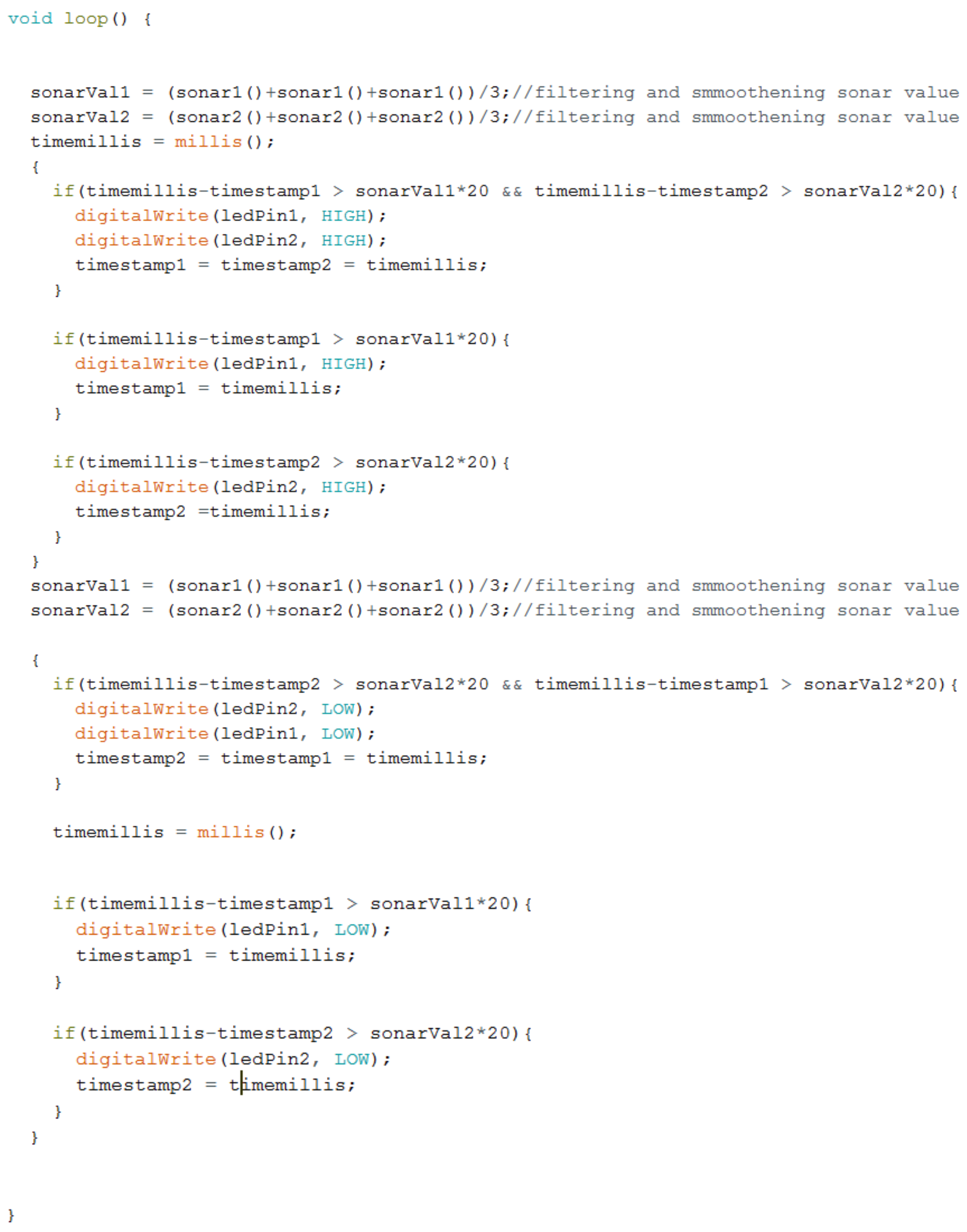
Dual Sonar

Sonars in themselves presented many problems, these will be discussed below, however we came across particular issues when working with 2 sonars in parallel. Our objective was to beat each drum stick at a frequency proportional to the distance our hands would be from their respective sensor, which presented many challenges as we required two simultaneous timers to run. If we were to simply actuate the drum sticks when they detected an arm this would have been simple and possible to execute without multithreading in the system.

We attempted to run multithreading on the Arduino, however the model we have employed is not capable of it. The alternative was to implement virtual threads, a special library was developed by the C community called ProtoThreads. However this had dismal performance, causing lag and reactivity issues. As the reaction time between sensing the arm and the drumming of the beat was critical for a pleasant experience we abandoned this path.

The next attempt was to implement a dual timer in the Arduino using time-stamps and the inbuilt Arduino clock (called upon using the millis() function). This implementation worked the best, however we still experienced issues in reactivity and consistency. Below I have included a screen shot of the program. The general premise of this code was to consistently run through the loop, read the sensor value, determine the time required to wait between each beating of the drum, calculate the elapsed time using the in-build Arduino clock and two separate timestamps. If the time was elapsed the drum would beat. This proved problems as there still was a linearity to the code and the beating of the drum (simulated below with the blinking of LEDs) was not independent.

Certain technique could be used in the future, however these require a lot of time to test and much more sophisticated code.



```
void loop() {


  sonarVal1 = (sonar1()+sonar1()+sonar1())/3;//filtering and smmoothening sonar value
  sonarVal2 = (sonar2()+sonar2()+sonar2())/3;//filtering and smmoothening sonar value
  timemillis = millis();
  {
    if(timemillis-timestamp1 > sonarVal1*20 && timemillis-timestamp2 > sonarVal2*20){
      digitalWrite(ledPin1, HIGH);
      digitalWrite(ledPin2, HIGH);
      timestamp1 = timestamp2 = timemillis;
    }

    if(timemillis-timestamp1 > sonarVal1*20){
      digitalWrite(ledPin1, HIGH);
      timestamp1 = timemillis;
    }

    if(timemillis-timestamp2 > sonarVal2*20){
      digitalWrite(ledPin2, HIGH);
      timestamp2 =timemillis;
    }
  }
  sonarVal1 = (sonar1()+sonar1()+sonar1())/3;//filtering and smmoothening sonar value
  sonarVal2 = (sonar2()+sonar2()+sonar2())/3;//filtering and smmoothening sonar value

  {
    if(timemillis-timestamp2 > sonarVal2*20 && timemillis-timestamp1 > sonarVal2*20){
      digitalWrite(ledPin2, LOW);
      digitalWrite(ledPin1, LOW);
      timestamp2 = timestamp1 = timemillis;
    }

    timemillis = millis();


    if(timemillis-timestamp1 > sonarVal1*20){
      digitalWrite(ledPin1, LOW);
      timestamp1 = timemillis;
    }

    if(timemillis-timestamp2 > sonarVal2*20){
      digitalWrite(ledPin2, LOW);
      timestamp2 = timemillis;
    }
  }


}
```

Figure 1 Code for 2 sonar that was not used in the end

Single Sonar

Implementing the single sonar was relatively straight forward, however the complexities of the program lay in optimizing the code, to receive the desired reaction time.

The four major issues we had to deal with were as follows:

- Increasing sensor reading consistency

- We employed a very simple filtering function that simply took the average value from three readings. This slowed down the code, however it reduced random erratic behavior. The problem by implementing such methods is that reading if the sensor value jumped too much then the filtered value would also changed significantly. This filtering function could mitigate errors that deviate from the real value by approximately 20%. Effectively damping the error by a factor of 3, taking it from 20% to about 7%.

```
sonarVal = (sonar()+sonar()+sonar())/3;//filtering and smoothening sonar value
```

- Negate the effect of the time the sensor takes to record values on the drumming delay
  - A timestamp is taken from the moment the sensor begins to read. Then when the drum is made to beat, the timestamp is subtracted to give the real value relative to the time when the sensor recording began. Effectively we are compensating here for the computational time of getting the distance using the sonar.

```
timestamp = millis();
sonarVal = (sonar()+sonar()+sonar())/3;

if (50*sonarVal -(millis()-timestamp)> 0 &&  sonarVal <= 50 ){
  Serial.println(20*sonarVal - (millis()-timestamp));
  delay(20*sonarVal - (millis()-timestamp));
}
```

- Increase the reactivity of the drum upon moving the arm
  - The program employs two delays, a delay on the onbeat (when stick strikes the drum) and a delay on the offbeat (when the stick lifts up). Instead of using a constant delay between these two, we are consistently reading the sensor values. Hence if the hand above the sonar suddenly moves down, the drum does not need to wait for a full cycle (on & off beat) to finish before changing the drumming rate.
- Stop drum beating if no hand is detected
  - In our system we considered that anything above 50 cm above the sensor is considered out of reach. Hence we implemented a if statement that effectively paused the code until an arm is detected.

```
while(sonarVal>50 ){
  sonarVal = (sonar()+sonar()+sonar())/3;
  Serial.println("wait");
}
```

Apendix

Sonar and Drum

```
//------Sonars--------
#define trigPin 8
#define echoPin 9
float distance;
long duration;
float sonarVal; // reading from sonar that is printed
static unsigned long timestamp = 0;

//------Motors-------
#define leftMotor  12
#define rightMotor  13
#define leftSpeed  3
#define rightSpeed  11


//------LED----------
//leds present for troubleshooting
#define ledPin1 2
#define ledPin2 4


void setup() {
  Serial.begin(9600);
  Serial.println("Start");

  //-----Setting up sonar---------
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  //-----Setting up motors---------
  //Direction A & B
  pinMode(leftMotor, OUTPUT);
  pinMode(rightMotor, OUTPUT);
  // Speed A & B
  pinMode(leftSpeed, OUTPUT);
  pinMode(rightSpeed, OUTPUT);

  analogWrite(leftSpeed, 255);
  analogWrite(rightSpeed, 255);
  //----Setting up led------------
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
}

void loop() {

  sonarVal = (sonar()+sonar()+sonar())/3;//filtering and smoothening sonar value

  timestamp = millis();//millis and timestamps are used throughout the program to ensure that the
sonar read time is compensated for
  while(sonarVal>50 ){
    sonarVal = (sonar()+sonar()+sonar())/3;
    Serial.println("wait");
  }
  Serial.println(sonarVal);
  if(50*sonarVal-(millis()-timestamp)>0 &&  sonarVal <= 50 ){// statement insuring that the delay
read is within the desired range
    delay(20*sonarVal-(millis()-timestamp));
  }

  delay(150);
  digitalWrite(rightMotor, HIGH);
  digitalWrite(leftMotor, LOW);
  digitalWrite(ledPin1, HIGH);
  digitalWrite(ledPin2, LOW);

  timestamp = millis();
  sonarVal = (sonar()+sonar()+sonar())/3;

  if (50*sonarVal -(millis()-timestamp)> 0 &&  sonarVal <= 50 ){
    Serial.println(20*sonarVal - (millis()-timestamp));
    delay(20*sonarVal - (millis()-timestamp));
  }
  delay(150);
  digitalWrite(rightMotor, LOW);
  digitalWrite(leftMotor, HIGH);
  digitalWrite(ledPin1, LOW);
  digitalWrite(ledPin2, HIGH);
}
```

```
float sonar(){//sonar data distance function
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance= duration*0.034/2;
  return distance;
}

void beatA(int dly, int power) {
  analogWrite(3, power);
  digitalWrite(12, HIGH);
  delay(dly);
  digitalWrite(12, LOW);
  delay(dly);
}

void beatB(int dly, int power) {
  analogWrite(11, power);
  digitalWrite(13, HIGH);
  delay(dly);
  digitalWrite(13, LOW);
  delay(dly);
}

void beatAandB(int dly, int power) {
  analogWrite(3, power);
  analogWrite(11, power);
  digitalWrite(12, HIGH);
  digitalWrite(13, HIGH);
  delay(dly);
  digitalWrite(12, LOW);
  digitalWrite(13, LOW);
  delay(dly);
}
```

Beat

```
//------Sonars--------
#define trigPin 8
#define echoPin 9
float distance;
long duration;
float sonarVal; // reading from sonar that is printed
static unsigned long timestamp = 0;

//------Motors-------
#define leftMotor  12
#define rightMotor  13
#define leftSpeed  3
#define rightSpeed  11



//------LED----------
//leds present for troubleshooting
#define ledPin1 2
#define ledPin2 4



void setup() {
  Serial.begin(9600);
  Serial.println("Start");

  //-----Setting up sonar---------
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  //-----Setting up motors---------
  //Direction A & B
  pinMode(leftMotor, OUTPUT);
  pinMode(rightMotor, OUTPUT);
  // Speed A & B
  pinMode(leftSpeed, OUTPUT);
  pinMode(rightSpeed, OUTPUT);

  analogWrite(leftSpeed, 255);
  analogWrite(rightSpeed, 255);

  //----Setting up led------------
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);



}

void loop() {
  beatAandB(150, 255);
  beatA(300,255);
  beatB(300,255);
  beatA(250,255);

  for(int i = 300; i> 150; i = i -30){
     beatA(i,255);
     beatB(i,255);
  }

  beatAandB(150, 255);
  beatAandB(150, 255);
  beatAandB(150, 255);

  beatA(150,255);
  beatB(150,255);
  beatA(150,255);
  beatB(150,255);
  beatA(150,255);
  beatB(150,255);

  for(int i = 150; i< 300; i = i + 30){
     beatA(i,255);
     beatB(i,255);
  }


}



float sonar(){//sonar data distance function
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distance= duration*0.034/2;
  return distance;
}


void beatA(int dly, int power) {
  analogWrite(3, power);
  digitalWrite(12, HIGH);
  digitalWrite(ledPin1, HIGH);
  delay(dly);
  digitalWrite(12, LOW);
  digitalWrite(ledPin1, LOW);
  delay(dly);
}

void beatB(int dly, int power) {
  analogWrite(11, power);
  digitalWrite(13, HIGH);
  digitalWrite(ledPin2, HIGH);
  delay(dly);
  digitalWrite(13, LOW);
  digitalWrite(ledPin2, LOW);
  delay(dly);
}

void beatAandB(int dly, int power) {
  analogWrite(3, power);
  analogWrite(11, power);
  digitalWrite(12, HIGH);
  digitalWrite(ledPin1, HIGH);
  digitalWrite(ledPin2, HIGH);
  digitalWrite(13, HIGH);
  delay(dly);
  digitalWrite(12, LOW);
  digitalWrite(ledPin1, LOW);
  digitalWrite(ledPin2, LOW);
  digitalWrite(13, LOW);
  delay(dly);
}
```