Criterion C: Development

Overview

In Criterion B, I intended to use 3 main functions; core algorithm, calibrate, and recording. The core algorithm is to determine the detection of the pendulum. The calibrate function is there to set up the device for data recording, which is the next function. Then there were a few other minor functions that tested the calibrate value and updated the LCD. Here I will go into depth on each part and present code snippets showing how I went about each. After that I will talk about the hardware aspect of the project.

Variables

Before I get into code, I want to go over variables. Here is my variable table:

//insert variable table (include libraries and pin declarations as well)

Core Algorithm

This section includes the 'setup' and 'loop' functions. First off I start a Serial port with the computer so that if I wanted to in the future print something to my computer I could do so quickly. After that I set my pin modes for my LED and buttons. The buttons are inputs and the LED an output. This all can be seen in Figure 1.

```
void setup() {
  Serial.begin(9600);
  lcd.begin(16, 2);
  pinMode(CALIBRATE_PIN, INPUT);
  pinMode(RECORDING_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
}
```

Figure 1

Next is the 'loop'. This is a longer function and will be broken down into a couple sections. The first part (shown entirely in Figure 1) is about the ultrasonic sensor and getting inputs from the sensor. This will only run if the device is calibrated and recording.

```
void loop() {
  if (Calibrated && Recording) {
    if ((CurrentDistance <= (CalibrateValue*0.75)) and (CurrentDistance != 0)) {    //if the pendulum is seen
      digitalWrite(LED_PIN, HIGH);
      if (Period > 0) {
        PeriodList[Passes] = Period;      //store the period
        Passes = Passes + 1;
      }
      Period = 0;
      StartTime = millis();
      lcd.setCursor(10, 1); lcd.print(Period);
    } else {                                                                    //if pendulum is not seen
      digitalWrite(LED_PIN, LOW);
      Period = millis() - StartTime;
      lcd.setCursor(10, 1); lcd.print(Period);
    }
  }
```

Figure 2

This logic says that if the current distance seen by the sensor is less than 75% of the calibrated value, and not equal to 0, then consider that a detection of the pendulum. I have an LED that turns on when this happens so that it is clear to the user that the pendulum was seen. The next part stores the period if the period is not 0 and increase the pass count. After that, set the period to 0 and reset the start time. Then the period is printed for the user so they can see it increase.

If the pendulum was not seen, i.e. the distance was more than 75% of the calibrated value or it was 0; turn the LED off, increase the period.

Figure 3 shows the 2nd part of the loop.

```
  //CALIBRATION
  CalState = digitalRead(CALIBRATE_PIN);
  if (CalState == LOW) {
    calibrate();
  }

  //RECORDING
  RecState = digitalRead(RECORDING_PIN);
  //Serial.println(digitalRead(RECORDING_PIN));
  if (RecState == LOW) {
    if (Recording) {
      recordingOff();
    } else {
      recordingOn();
    }
  }

  updateLCD();  //done every frame
}
```

Figure 3

The first section here checks for the calibration button press. If this is pressed, it runs the calibration function. The recording is slightly different. It checks to see if the recording is on or off, and it flips it. Lastly, the update function is called.

Calibrate

Figure 4 shows the calibrate function.

```
void calibrate() {
  if (!Recording) {
    lcd.clear();
    CalibrateValueLast = CalibrateValue;
    CalibrateValue = sr04.Distance();
    AcceptCalibrateValue = TestAcceptC(CalibrateValue, 0, 30);
    if (AcceptCalibrateValue) {
      Calibrated = true;
    } else {
      lcd.clear();
      if (CalibrateValue == 0) {
        lcd.setCursor(0, 0); lcd.print("CAL Can't = 0");
      } else {
        lcd.setCursor(0, 0); lcd.print("CAL Can't be >30");
      }
      lcd.setCursor(0, 1); lcd.print("Try Again");
      delay(2000);
      lcd.clear();
      CalibrateValue = CalibrateValueLast;
    }
  } else {
    lcd.clear();
    lcd.setCursor(0, 0); lcd.print("Cannot Calibrate");
    lcd.setCursor(0, 1); lcd.print("Recording");
    delay(1000);
    lcd.clear();
  }
}
```

Figure 4

Included here is a check to see if the device is recording. If it is, it skips to display to the LCD that you cannot calibrate while recording. If the device is not recording, I proceed to calibrate the device. To start, I save the previous calibration value. Then I get the current calibration value. Then I make sure that that new value is acceptable. If it is accepted then I make calibrated true. If not, then I revert the current calibrated value back to the old one, then display why the calibrate failed.

Recording

Figure 5 shows the ‘recording on’ function.

```
void recordingOn() {
  Period = 0;
  if (Calibrated) {
    AcceptCalibrateValue = TestAcceptC(CalibrateValue, 0, 30);
    if (AcceptCalibrateValue) {
      lcd.clear();
      lcd.setCursor(0, 0); lcd.print("Recording: ON");
      delay(1000);
      lcd.clear();
      Recording = true;
    }
  } else {
      lcd.clear();
      lcd.setCursor(0, 0); lcd.print("Not Calibrated");
      delay(2000);
      lcd.clear();
    }
}
```

Figure 5

When you start recording, the period is set to 0. After that I check to make sure that the device has a calibrated value. If so, then make sure that the calibrated value is acceptable. If so, then let the user know that you have successfully started recording and set recording to true. If the device is not calibrated, don't do anything but let the user know that the device is not calibrated.

Figure 6 shows the 'recording off' function.

```
void recordingOff() {
  //display all the recorded values
  //wait for a responce
  //clear all the periods
  lcd.clear();
  lcd.setCursor(0, 0); lcd.print("Recording: OFF");
  delay(1000);
  lcd.clear();
  Recording = false;
}
```

Figure 6

This function is very straight forward. First display all the recorded period values and wait for a response from the user to continue. Once the user moves on, clear all the periods. Lastly let the user know that the recording has turned off and set recording to false.

Other Functions

The two other functions are 'updateLCD' and 'TestAcceptC'. Each are shown below in figures 7 and 8.

```
void updateLCD() {
  lcd.setCursor(0, 0);
  if (CurrentDistance != CurrentDistanceLast) {
    lcd.clear();
    lcd.print(CurrentDistance);
    CurrentDistanceLast = CurrentDistance;
  }

  lcd.setCursor(0, 1); lcd.print("CAL:");

  lcd.setCursor(4, 1); lcd.print(CalibrateValue);

  lcd.setCursor(14, 0); lcd.print("cm");
}
```

Figure 7

```
bool TestAcceptC(float CalibrateValue, float MinC, float MaxC) {
  return CalibrateValue > MinC && CalibrateValue < MaxC;
}
```

Figure 8

In figure 7, 'updateLCD' is shown. This function displays the current distance if it has changed. Then goes and prints the units and calibration value. 'TestAcceptC' makes sure that the calibration value is between a certain min and max.

Hardware

//hardware