The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Applied EEE Construction Project: Lab Report 2: Subsystem design and Main vehicle integration

Lab report from sessions 3 & 4 of the applied EEE construction Project

Michael Bagge-Hansen
eeymb8@Nottingham.ac.uk

Abstract:
This report outlines the plan, testing and build of the autonomous parking subsystem. The system utilises ultrasonic sensors mounted to servos, to map out the parking space. to improve accuracy the servos only move to 3 set positions. due to a deficit of pins the serial monitor was out of service, this made debugging the system very difficult, and as a result did not meet the brief within the give time.

November 2019
The University of Nottingham

Contents

Chapter 1

Subsystem design

1.1 circuit

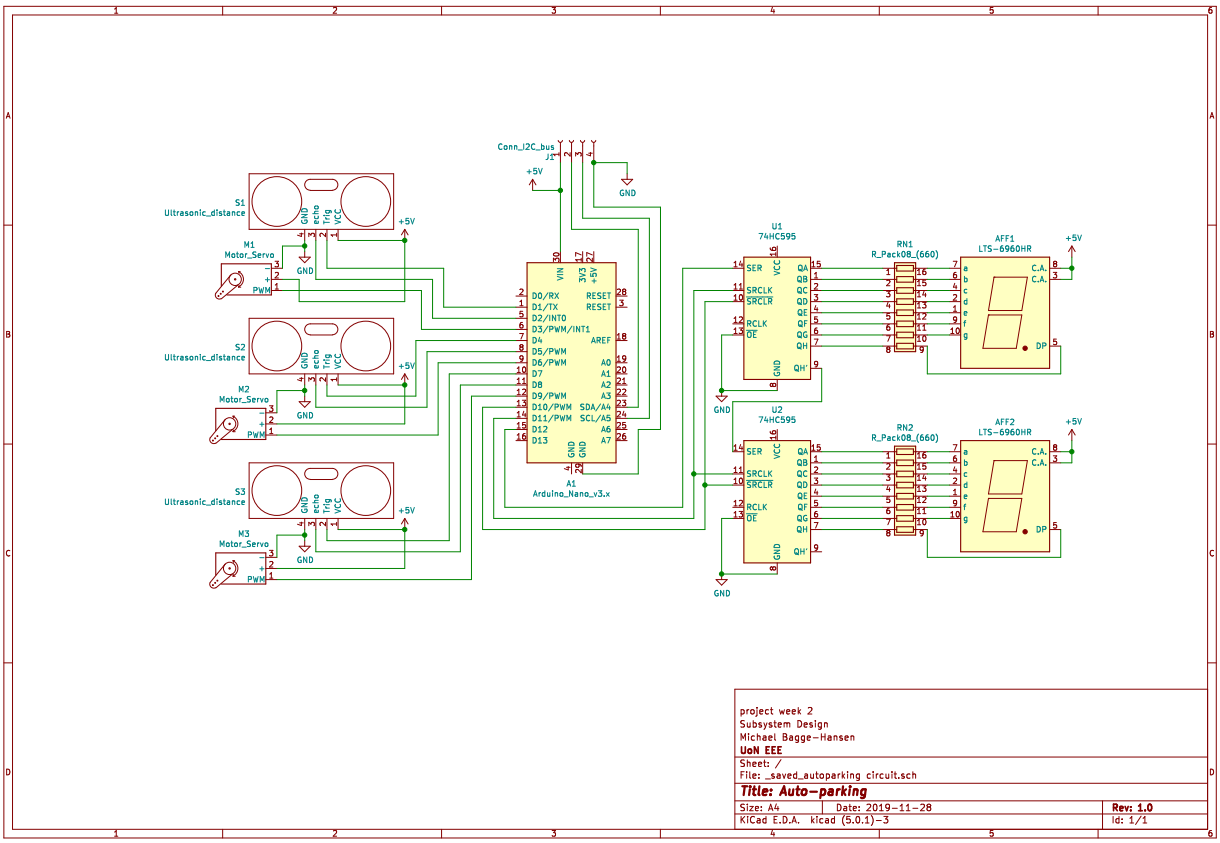

Figure 1.1: circuit schematic

figure 1.1 shows the plan for the circuit, as shown the trigger pin for one of the ultrasonic sensors is connected to digital 1 which is the serial write pin. upon testing it was found that despite research this meant that only the serial communication with the Arduino or ultrasonic sensor could work at a time. This became a problem as trouble shooting became difficult to do, as demonstrated in the remainder of the report.

1.2 placement and mounting

as decided in lab report 1, the servos where to placed it a triangular type layout with one on each back corner and one on at the front. it was found this layout would give the most coverage of the surroundings. as shown in figure 1.2.

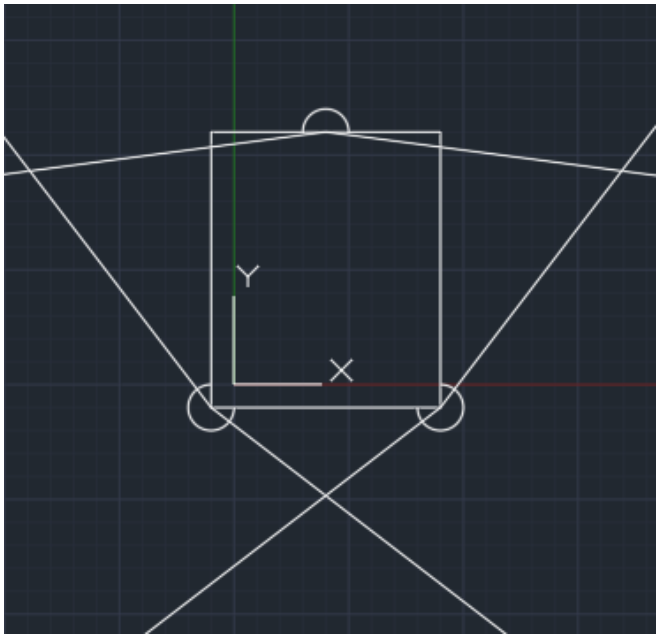

Figure 1.2: Sensor placement

1.2.1 Sensor to servo mount

Figure 1.3: Ultrasonic mount

to mount the ultrasonic sensor the the servos a custom part had to be designed, it took a few iterations and as shown in figure 1.4 still does not fit properly, this is due to inaccurate measuring of the sensor and shrinking that occurs with PLA plastics. the piezoelectric crystal does not fit fully, however because the ultrasonic sender and receiver fit properly it doesn't need a redesign.

the Servo mounts are left over from a previous project, to mount them to the PCB holes drilled into the 3d printed part, counter sunk, and then screwed onto already existing holes in the PCB. This is a quick solution to mounting the components as there was no waiting time to print the parts, however for later versions a custom part would be designed at it would fit better.

Figure 1.4: Ultrasonic mount with Sensor

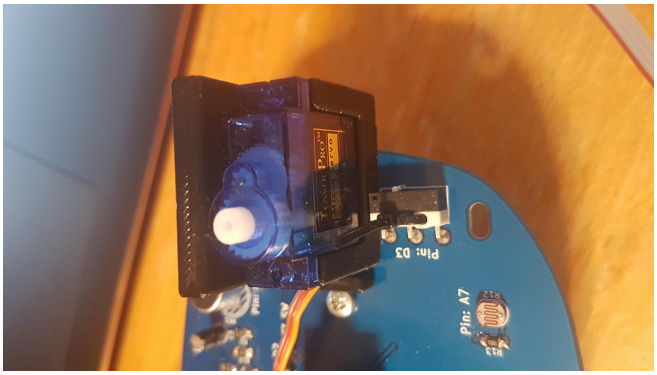

Figure 1.5: Assembly Back

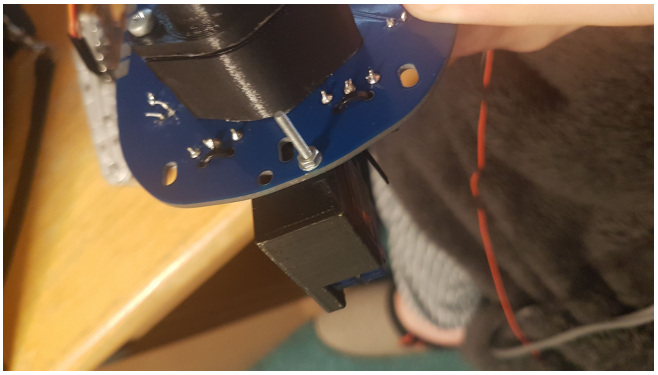

Figure 1.6: Assembly Front

1.3 communications

On both the main-bot and mini-bots there will be 2 Arduinos, the one for controlling the motors and one for controlling the parking system. This is done for multiple reasons:

- There are not enough pins on an Arduino nano to attach all the sensors for the subsystems and the motors
- As the Arduinos are single threaded processors only one thing can be done at a time.
- As multiple subsystems will be attached to the main bot it is easier to develop and test on separate Arduinos.
- Adding all the systems code to one Arduino nano, the Arduino might not have enough memory to run and store all the code.

To communicate between the 2 Ardunios on the mini-bot and the multiple Arduinos on the main-bot an I2c communication system was developed. This consisted of a 3 byte signal from the subsystem Arduino(slave) to the main board Arduino(slave). the byte order was as follows:

- a char for the command
- the first byte of a 16bit integer
- the second byte of a 16bit integer

to send this a function was developed:

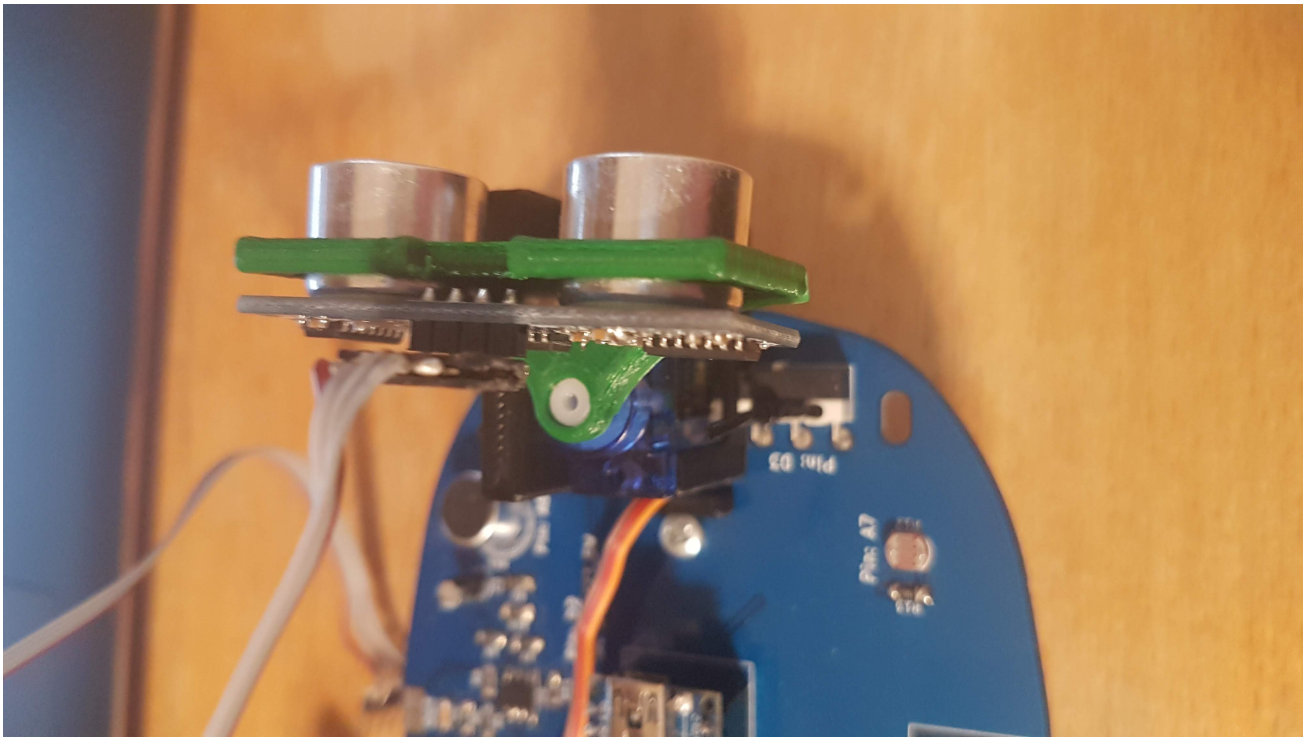

Figure 1.7: circuit schematic

```
#define I2CADDR 0x08

void i2cCommand( char command, int Mspeed) {
  int numa = (Mspeed << 8 ) & 0xff;
  int numb = Mspeed & 0xff;
  Wire.beginTransmission(I2CADDR); // transmit to device #8
  Wire.write(command);
  Wire.write(numa);
  Wire.write(numb);
  Wire.endTransmission();     // stop transmitting
}
```

Listing 1.1: i2c communication function

as the int passed as Mspeed is not a 16 bit integer split it needs to be split into 2 integers to be sent over i2c.

Chapter 2

Testing

2.1 7 segment display

to test the 7 segment displays they were initially mounted to a breadboard, this did not work as the size of the breadboard made it hard to see what connections where made, and using a multimeter was hard as lots of the wires where very close together.
the display was instead tested on the main board, as some of the connections between the shift register where not the same as the diagram, initially, the display was just displaying random lights, so a diagram was made to find the connections between pins on the display and pins on the shift register, this was then used to make an array corresponding the bits sent to the shift register and numbers displayed on the screen, shown in listing 2.1.

```
byte digitA[] = {
  /*0*/ B10000001, /*1*/ B11001111, /*2*/ B10010010, /*3*/ B10001010, /*4*/ B11001100,
  /*5*/ B10101000, /*6*/ B10100000, /*7*/ B10001111, /*8*/ B10000000, /*9*/ B10001000,
  };
byte digitB[] = {
  /*0*/ B00100100, /*1*/ B11110110, /*2*/ B01001100, /*3*/ B11000100, /*4*/ B10010110,
  /*5*/ B10000101, /*6*/ B00000101, /*7*/ B11110100, /*8*/ B00000100, /*9*/ B10000100
  };
```

Listing 2.1: 7 segment digit array

This could then be used in all other programs using the 7 segment displays.

2.2 Ultrasonic distance sensor

The Ultrasonic sensors utilised the new ping library. This allowed for a significantly more simple code, as all the code required to get the distance is to initialise the sensor and call the `ping cm` function, as shown in listing 2.3

```
#define TRIGGER_PIN  0  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     2  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.

```

```
data = sonar.ping_cm();
```

Listing 2.2: New ping usage

2.3 Servos

Using is farily simmilar to the Ultrasonic sensors, the library servo is used, which is included in the Arduino IDE install. All thats need to be done is to as with the ultrasonic Sensors it to initalise the servo and then call the function as shown in listing **??**

```
Servo myservo;  // create servo object to control a servo

//in setup:
myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  //Serial.begin(9600);

//in loop:
myservo.write(pos);
```

Listing 2.3: New ping usage

2.4 Mini-bot & Communication

Chapter 3

Build

3.1 final assembly

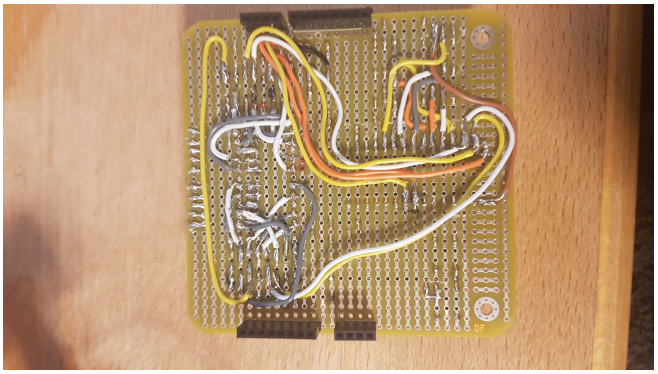

Figure 3.1: Assembly Back

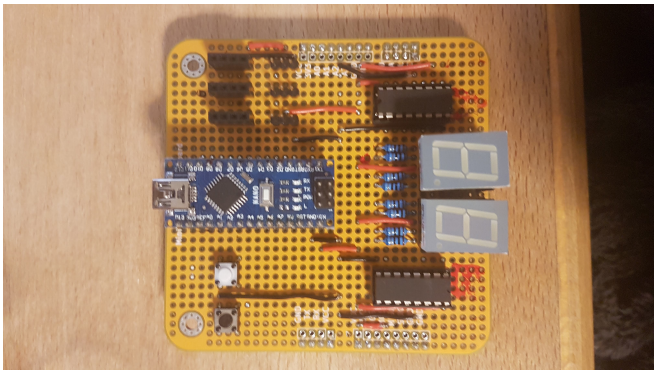

Figure 3.2: Assembly Front

All the components are positioned on the PCB to aid assembly and use, although not shown in figure 3.2 the ultrasonic sensors and servos have ports above the Arduino, they are aligned to have the sensor and servo form each corner to plug into the same plane on the PCB, this makes assembly and disassembly easier as it is easier to put the right cabled into the write places. To provide power to the various components a 5 volt and ground rail are used, this makes the soldering easier as only straight wires would be needed to power any component on the board. As shown in figure 3.1 most of the wires where not soldered through the holes in the PCB, this is for multiple reasons:

- some components where too close together
- easy to de-solder if a mistake is made
- hides messy wires as they would all be on the bottom

this did not effect durability as much as expected as none of the joints became loose or undone during testing.

3.2 Code

There where a few ideas for approaching the code, as the control surfaces for the mini-bot are different to those on the main bot, it was important that the function could park both vehicles. this was quite a challenge. as it was found that the ultrasonic sensors where not very accurate

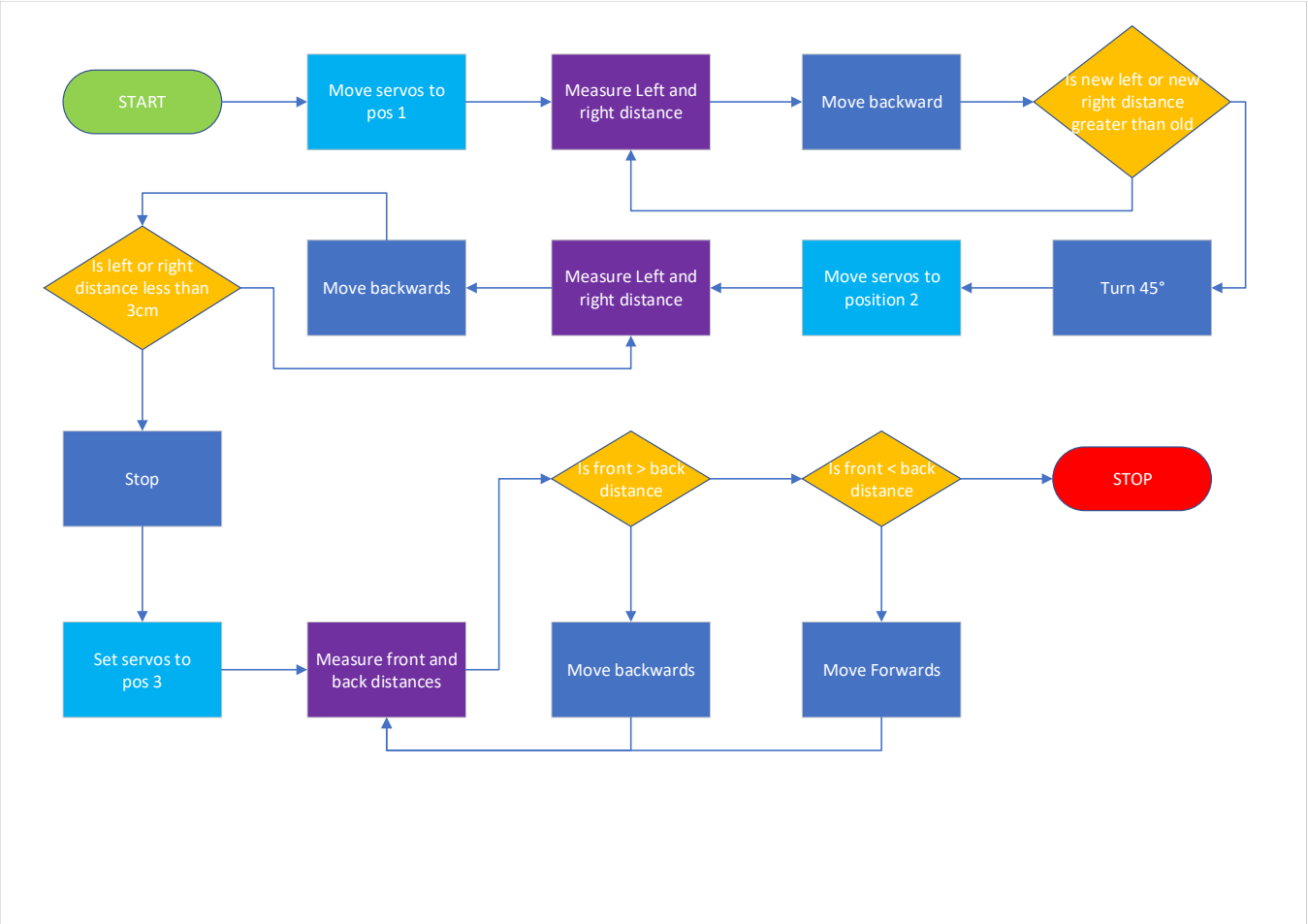

Figure 3.3: flowchart

or precise so any system with feedback loops would not be fiable. due to a time crunch the code was simplified to speed up its development. The code represented by 3.2 shows a simple function for parking the mini bot, as the driven wheels are at the back the mini-bot can turn around on of its back wheels this makes parking much easyer as once the outer wheel is in the final position the car can be manoeuvred around this point.

This fact makes parking a lot simpler. in the code shown, the approach it to find a parking spot by reversing until the wall on one side of the car stops, then turns 45 deg and continues to reverse until either back corner almost hits the end of the space, then it turns again and adjusts to make the front and back distances equal. to make this system more accurate the servos moved such that the sensors where always perpendicular to the wall. this helps with accuracy as it means the transmitter and receiver half of the sensors are an equal distance to the wall allowing an accurate measurement to be taken.

Chapter 4

Appendix

4.1 Testing Code

7 segment test A

```
/*  SevenSegmentLEDdisplay102a.ino
 *   2017-02-20
 *   Mel Lester Jr.
 *  Simple example of using Shift Register with a
 *  Single Digit Seven Segment LED Display
*/
// Globals
const int dataPin = 12;  // blue wire to 74HC595 pin 14
const int latchPin = 11; // green to 74HC595 pin 12
const int clockPin = 10; // yellow to 74HC595 pin 11

/* uncomment one of the following lines that describes your display
 *  and comment out the line that does not describe your display */
const char common = 'a';    // common anode
//const char common = 'c';    // common cathode

bool decPt = true;  // decimal point display flag

void setup() {
  // initialize I/O pins
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  decPt = !decPt; // display decimal point every other pass through loop

  // generate characters to display for hexidecimal numbers 0 to F
  for (int i = 0; i <= 15; i++) {
    byte bits = myfnNumToBits(i) ;
    if (decPt) {
      bits = bits | B00000001;  // add decimal point if needed
    }
    myfnUpdateDisplay(bits);    // display alphanumeric dig
    delay(500);                 // pause for 1/2 second
  }
}

```

```
void myfnUpdateDisplay(byte eightBits) {
  if (common == 'a') {                  // using a common anonde display?
    eightBits = eightBits ^ B11111111;  // then flip all bits using XOR
  }
  digitalWrite(latchPin, LOW);  // prepare shift register for data
  shiftOut(dataPin, clockPin, LSBFIRST, eightBits); // send data
  digitalWrite(latchPin, HIGH); // update display
}

byte myfnNumToBits(int someNumber) {
  switch (someNumber) {
    case 0:
      return B11111100;
      break;
    case 1:
      return B01100000;
      break;
    case 2:
      return B11011010;
      break;
    case 3:
      return B11110010;
      break;
    case 4:
      return B01100110;
      break;
    case 5:
      return B10110110;
      break;
    case 6:
      return B10111110;
      break;
    case 7:
      return B11100000;
      break;
    case 8:
      return B11111110;
      break;
    case 9:
      return B11110110;
      break;
    case 10:
      return B11101110; // Hexidecimal A
      break;
    case 11:
      return B00111110; // Hexidecimal B
      break;
    case 12:
      return B10011100; // Hexidecimal C or use for Centigrade
      break;
    case 13:
      return B01111010; // Hexidecimal D
      break;
    case 14:
      return B10011110; // Hexidecimal E
      break;
    case 15:
      return B10001110; // Hexidecimal F or use for Fahrenheit
      break;
    default:
      return B10010010; // Error condition, displays three vertical bars
```

```
      break;
  }
}
```

Listing 4.1: 7 segment testing code A

7 segment test B

```
#define dataPin 12
#define latchPin 13
#define clockPin 11

byte digitA[] = {
  /*0*/ B10000001,
  /*1*/ B11001111,
  /*2*/ B10010010,
  /*3*/ B10001010,
  /*4*/ B11001100,
  /*5*/ B10101000,
  /*6*/ B10100000,
  /*7*/ B10001111,
  /*8*/ B10000000,
  /*9*/ B10001000,
  };
byte digitB[] = {
  /*0*/ B00100100,
  /*1*/ B11110110,
  /*2*/ B01001100,
  /*3*/ B11000100,
  /*4*/ B10010110,
  /*5*/ B10000101,
  /*6*/ B00000101,
  /*7*/ B11110100,
  /*8*/ B00000100,
  /*9*/ B10000100
  };

void setup() {
  // initialize I/O pins
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  //count up routine
  for(int i=0; i < 100; i++){
    displayNum(i);
    delay(100);
  }
}

void digitdisplay(int A,int B){
  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, LOW);
  digitalWrite(clockPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, digitB[B]);
  shiftOut(dataPin, clockPin, LSBFIRST, digitA[A]);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
```

```
  digitalWrite(latchPin, HIGH);
  }
void displayNum(int num){
  int A = (num / 10)% 10;
  int B = num % 10;

  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, LOW);
  digitalWrite(clockPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, digitB[B]);
  shiftOut(dataPin, clockPin, LSBFIRST, digitA[A]);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin, HIGH);
  }
```

Listing 4.2: 7 segment testing code B

servo test A

```
#include <Servo.h>

const int LEFT = A2;
const int RIGHT = A3;

Servo myservo;  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

int leftVal = 100;
int rightVal = 100;

void setup() {
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  //Serial.begin(9600);
}

void loop() {
  leftVal = analogRead(LEFT);
  rightVal = analogRead(RIGHT);

  delay(10);

  if(leftVal < 5){
    if (pos < 180){
      pos ++;
      myservo.write(pos);
    }
    }
  else if(rightVal < 5){

    if (pos >= 0){
      pos --;
      myservo.write(pos);
    }
    }
}
```

Listing 4.3: Servo testing code A

servo test B

```
#include <Servo.h>

const int LEFT = A2;
const int RIGHT = A3;
int index = 0;

Servo myservo[3];  // create servo object to control a servo
// twelve servo objects can be created on most boards

int pos = 0;    // variable to store the servo position

int leftVal = 100;
int rightVal = 100;

void setup() {
  myservo[0].attach(3);  // attaches the servo on pin 9 to the servo object
  myservo[1].attach(6);
  myservo[2].attach(9);
  //Serial.begin(9600);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  leftVal = analogRead(LEFT);
  rightVal = analogRead(RIGHT);
    if (leftVal < 5) {
      if (index >= 2) {
        index = 0;
      } else {
        index++;
      }
  } else if (rightVal < 5) {
    if (index <= 0) {
      index = 2;
    } else {
      index--;
    }
  }
  sweep(index);
  flash(100,index+1);
  delay(1000);
}

void sweep(int i) {
  for (pos = 0; pos <= 180; pos += 1) { // goes from 0 degrees to 180 degrees
    // in steps of 1 degree
    myservo[i].write(pos);              // tell servo to go to position in
    variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the
    position
  }
  for (pos = 180; pos >= 0; pos -= 1) { // goes from 180 degrees to 0 degrees
    myservo[i].write(pos);              // tell servo to go to position in
    variable 'pos'
    delay(15);                       // waits 15ms for the servo to reach the
    position
  }
}

void flash(int frq, int period) {
  for (int i = 0; i < period; i++) {
```

```
    digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
    delay(frq);                       // wait for a second
    digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
    delay(frq);
  }
}
```

Listing 4.4: Servo testing code B

Ultrasonic sensor A

```
/*
  Ultrasonic Sensor HC-SR04 and Arduino Tutorial

  by Dejan Nedelkovski,
  www.HowToMechatronics.com

*/
#define dataPin 12
#define latchPin 10
#define clockPin 11

byte digitA[] = {
  /*0*/ B10000001, /*1*/ B11001111, /*2*/ B10010010, /*3*/ B10001010, /*4*/ B11001100,
  /*5*/ B10101000, /*6*/ B10100000, /*7*/ B10001111, /*8*/ B10000000, /*9*/ B10001000,
  };
byte digitB[] = {
  /*0*/ B00100100, /*1*/ B11110110, /*2*/ B01001100, /*3*/ B11000100, /*4*/ B10010110,
  /*5*/ B10000101, /*6*/ B00000101, /*7*/ B11110100, /*8*/ B00000100, /*9*/ B10000100
  };
// defines pins numbers
const int trigPin = 0;
const int echoPin = 2;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);

  pinMode(trigPin, OUTPUT); // Sets the trigPin as an Output
  pinMode(echoPin, INPUT); // Sets the echoPin as an Input
}
void loop() {
  // Clears the trigPin
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin on HIGH state for 10 micro seconds
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
```

```
  distance = duration * 0.034 / 2;
  // Prints the distance on the Serial Monitor
  displayNum(distance);
  delay(500);
}

void displayNum(int num){
  int A = (num / 10)% 10;
  int B = num % 10;

  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, LOW);
  digitalWrite(clockPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, digitB[B]);
  shiftOut(dataPin, clockPin, LSBFIRST, digitA[A]);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin, HIGH);
  }
```

Listing 4.5: Sensor testing code A

Ultrasonic sensor B

```
#include <NewPing.h>

#define dataPin 12
#define latchPin 10
#define clockPin 11

#define TRIGGER_PIN  0  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     2  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.


byte digitA[] = {
  /*0*/ B10000001, /*1*/ B11001111, /*2*/ B10010010, /*3*/ B10001010, /*4*/ B11001100,
  /*5*/ B10101000, /*6*/ B10100000, /*7*/ B10001111, /*8*/ B10000000, /*9*/ B10001000,
  };
byte digitB[] = {
  /*0*/ B00100100, /*1*/ B11110110, /*2*/ B01001100, /*3*/ B11000100, /*4*/ B10010110,
  /*5*/ B10000101, /*6*/ B00000101, /*7*/ B11110100, /*8*/ B00000100, /*9*/ B10000100
  };
// defines pins numbers
const int trigPin = 0;
const int echoPin = 2;
// defines variables
long duration;
int distance;
void setup() {
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
```

```
  pinMode(clockPin, OUTPUT);
}

int data = 0;

void loop() {
  data = sonar.ping_cm();
  displayNum(data);
  delay(500);
}

void displayNum(int num){
  int A = (num / 10)% 10;
  int B = num % 10;

  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, LOW);
  digitalWrite(clockPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, digitB[B]);
  shiftOut(dataPin, clockPin, LSBFIRST, digitA[A]);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin, HIGH);
  }
```

Listing 4.6: Sensor testing code B

Ultrasonic sensor C

```
#include <NewPing.h>

#define dataPin 12
#define latchPin 10
#define clockPin 11

#define TRIGGER_PIN  0  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     2  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 200 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.


byte digitA[] = {
  /*0*/ B10000001, /*1*/ B11001111, /*2*/ B10010010, /*3*/ B10001010, /*4*/ B11001100,
  /*5*/ B10101000, /*6*/ B10100000, /*7*/ B10001111, /*8*/ B10000000, /*9*/ B10001000,
  };
byte digitB[] = {
  /*0*/ B00100100, /*1*/ B11110110, /*2*/ B01001100, /*3*/ B11000100, /*4*/ B10010110,
  /*5*/ B10000101, /*6*/ B00000101, /*7*/ B11110100, /*8*/ B00000100, /*9*/ B10000100
  };
// defines pins numbers
const int trigPin = 0;
const int echoPin = 2;
// defines variables
```

```
long duration;
int distance;
void setup() {
  pinMode(dataPin, OUTPUT);
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
}

int data = 0;

void loop() {
  data = sonar.ping_cm();
  displayNum(data);
  delay(500);
}

void displayNum(int num){
  int A = (num / 10)% 10;
  int B = num % 10;

  //ground latchPin and hold low for as long as you are transmitting
  digitalWrite(latchPin, LOW);
  digitalWrite(clockPin, LOW);
  shiftOut(dataPin, clockPin, LSBFIRST, digitB[B]);
  shiftOut(dataPin, clockPin, LSBFIRST, digitA[A]);
  //return the latch pin high to signal chip that it
  //no longer needs to listen for information
  digitalWrite(latchPin, HIGH);
  }
```

Listing 4.7: Sensor testing code C

4.2 Main Code

```
#include <Wire.h>
#include <Servo.h>
#include <NewPing.h>

#define SONAR_NUM 3      // Number of sensors.
#define MAX_DISTANCE 100 // Maximum distance (in cm) to ping.

#define I2CADDR 0x08

//define 7seg pins
#define DATA_PIN 12
#define LATCH_PIN 10
#define CLOCK_PIN 11


const int LEFT = A2;
const int RIGHT = A3;

int leftVal = 100;
int rightVal = 100;

Servo myservo[3];  // create servo object to control a servo
// twelve servo objects can be created on most boards

NewPing sonar[SONAR_NUM] = {   // Sensor object array.
  NewPing(7, 8, MAX_DISTANCE), // Each sensor's trigger pin, echo pin, and max
    distance to ping.
```

```
  NewPing(4, 5, MAX_DISTANCE),
  NewPing(13, 2, MAX_DISTANCE)
};

int Mspeed = 140;

void setup() {
  // put your setup code here, to run once:
  Wire.begin(); // join i2c bus (address optional for master)

  // initialize I/O pins
  pinMode(DATA_PIN, OUTPUT);
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(CLOCK_PIN, OUTPUT);

  myservo[0].attach(3);  // attaches the servo on pin 9 to the servo object
  myservo[1].attach(6);
  myservo[2].attach(9);

  delay(300);
  myservo[0].write(0); //right
  delay(300);
  myservo[1].write(0); // left
  delay(300);
  myservo[2].write(0); // frount
  delay(3000);

  Serial.begin(9600);
}

int index = 0;
void loop() {
  // put your main code here, to run repeatedly:
  servoPos(0);

  parkpt1();
//  leftVal = analogRead(LEFT);
//  rightVal = analogRead(RIGHT);
//  if (leftVal < 5 || rightVal < 5) {
//    parkpt1();
//    if (index >= 2) {
//      index = 0;
//    } else {
//      index++;
//    }
//    switch (index) {
//      case 0:
//        displayNum(01);
//        parkpt1();
//        break;
//      case 1:
//        displayNum(02);
//        parkpt2();
//        break;
//      case 2:
//        displayNum(03);
//        parkpt3();
//        break;
//    }
//  }
//  delay(100);
```

```
}

int parkpt1() {
  int changeTol = 5;
  int wallTol = 3;
  //set servo
  servoPos(1);
  //backward
  i2cCommand('B', Mspeed);
  Serial.println("going backwards");
  //check for space

  int startDist = sonar[1].ping_cm();
  delay(50);
  int dist = sonar[1].ping_cm();
  int change = dist - startDist;
  while (change < changeTol) {
    flash(50, 2);
    dist = sonar[1].ping_cm();
    change = dist - startDist;
    displayNum(dist);
    delay(50);
  }

  //forward a bit
  //  i2cCommand('F', Mspeed);
  //  delay(200);
   i2cCommand('F', 0);
}

int parkpt2() {
  int changeTol = 5;
  int wallTol = 3;
  //set servo
  servoPos(2);

  //45 angle (right backwards)
  i2cCommand('r', Mspeed);
  delay(500);
  i2cCommand('r', 0);

  //backward till a sensor hits wall

  int Ldist = sonar[1].ping_cm();
  delay(50);
  int Rdist = sonar[0].ping_cm();

  i2cCommand('B', Mspeed);

  while (Ldist > wallTol && Rdist > wallTol) {
    flash(50, 2);
    Ldist = sonar[1].ping_cm();
    delay(50);
    Rdist = sonar[0].ping_cm();
    displayNum(Ldist);
    delay(50);
  }

  i2cCommand('F', 0);
}

```

```
149
150 int parkpt3() {
151   int changeTol = 5;
152   int wallTol = 3;
153   //45 angle (right forwards)
154   i2cCommand('r', -Mspeed);
155   delay(500);
156   i2cCommand('r', 0);
157
158   //set servo
159   servoPos(3);
160
161   //backwards till sensor even
162   int Ldist = sonar[1].ping_cm();
163   delay(50);
164   int Rdist = sonar[0].ping_cm();
165   delay(50);
166   int Frontdist = sonar[2].ping_cm();
167   delay(50);
168   int Backdist = (Rdist + Ldist) / 2;
169
170
171   while ((Backdist - Frontdist) > changeTol) {
172     flash(50, 2);
173     i2cCommand('B', Mspeed);
174     Ldist = sonar[1].ping_cm();
175     delay(50);
176     Rdist = sonar[0].ping_cm();
177     delay(50);
178     Frontdist = sonar[2].ping_cm();
179     delay(50);
180     Backdist = (Rdist + Ldist) / 2;
181   }
182   i2cCommand('F', 0);
183   //forwards till sensors even
184   while ((Frontdist - Backdist) > changeTol) {
185     flash(50, 2);
186     i2cCommand('F', Mspeed);
187     Ldist = sonar[1].ping_cm();
188     delay(50);
189     Rdist = sonar[0].ping_cm();
190     delay(50);
191     Frontdist = sonar[2].ping_cm();
192     delay(50);
193     int Backdist = (Rdist + Ldist) / 2;
194   }
195   i2cCommand('F', 0);
196
197   return 10;
198 }
199
200
201 void i2cCommand( char command, int Mspeed) {
202   int numa = (Mspeed << 8 ) & 0xff;
203   int numb = Mspeed & 0xff;
204   Wire.beginTransmission(I2CADDR); // transmit to device #8
205   Wire.write(command);
206   Wire.write(numa);
207   Wire.write(numb);
208   Wire.endTransmission();     // stop transmitting
209 }
```

```
210
211 void servoPos(int point) {
212   switch (point) {
213     case 0:
214       displayNum(00);
215       delay(300);
216       myservo[0].write(0); //right
217       delay(300);
218       myservo[1].write(0); // left
219       delay(300);
220       myservo[2].write(0); // frount
221       break;
222     case 1:
223       displayNum(11);
224       delay(300);
225       myservo[0].write(90); //right
226       delay(300);
227       myservo[1].write(0); // left
228       delay(300);
229       myservo[2].write(0); // frount
230       break;
231     case 2:
232       displayNum(22);
233       delay(300);
234       myservo[0].write(45); //right
235       delay(300);
236       myservo[1].write(45); // left
237       delay(300);
238       myservo[2].write(0); // frount
239       break;
240     case 3:
241       displayNum(33);
242       delay(300);
243       myservo[0].write(0); //right
244       delay(300);
245       myservo[1].write(90); // left
246       delay(300);
247       myservo[2].write(0); // frount
248       break;
249   }
250 }
251
252 //for 7seg display
253 byte digitA[] = {
254   /*0*/ B10000001, /*1*/ B11001111, /*2*/ B10010010, /*3*/ B10001010, /*4*/
      B11001100,
255   /*5*/ B10101000, /*6*/ B10100000, /*7*/ B10001111, /*8*/ B10000000, /*9*/
      B10001000,
256 };
257 byte digitB[] = {
258   /*0*/ B00100100, /*1*/ B11110110, /*2*/ B01001100, /*3*/ B11000100, /*4*/
      B10010110,
259   /*5*/ B10000101, /*6*/ B00000101, /*7*/ B11110100, /*8*/ B00000100, /*9*/
      B10000100
260 };
261
262 void displayNum(int num) {
263   int A = (num / 10) % 10;
264   int B = num % 10;
265
266   if (A >= 9 || A < 0 || B >= 9 || B < 0 ) {
```

```
267       flash(50, 4);
268       A = 0;
269       B = 0;
270     }
271
272     //ground latchPin and hold low for as long as you are transmitting
273     digitalWrite(LATCH_PIN, LOW);
274     digitalWrite(CLOCK_PIN, LOW);
275     shiftOut(DATA_PIN, CLOCK_PIN, LSBFIRST, digitB[B]);
276     shiftOut(DATA_PIN, CLOCK_PIN, LSBFIRST, digitA[A]);
277     //return the latch pin high to signal chip that it
278     //no longer needs to listen for information
279     digitalWrite(LATCH_PIN, HIGH);
280 }
281
282 void flash(int frq, int period) {
283     for (int i = 0; i < period; i++) {
284       digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage
        level)
285       delay(frq);                       // wait for a second
286       digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage
        LOW
287       delay(frq);
288     }
289 }
```

Listing 4.8: Main code