Jas.ino – code for the secondary robot

```
#include <Wire.h>                                                    // Calls for I2C bus library
#include "pitches.h"                                                 // For playing start and stop tones

#define MD25ADDRESS         0x58                                     // Address of the MD25
#define SPEED1              0x00                                     // Byte to send speed to both motors for forward and backwards motion if operated in MODE 2 or 3 and Motor 1 Speed if in MODE 0 or 1
#define SPEED2              0x01                                     // Byte to send speed for turn speed if operated in MODE 2 or 3 and Motor 2 Speed if in MODE 0 or 1
#define ENCODERONE          0x02                                     // Byte to read motor encoder 1
#define ENCODERTWO          0x06                                     // Byte to read motor encoder 2
#define ACCELERATION        0xE                                      // Byte to define motor acceleration
#define CMD                 0x10                                     // Byte to reset encoder values
#define MODE_SELECTOR       0xF                                      // Byte to change between control MODES

int MD25_Mode = 2;
//Explanation for different modes
// 0: Each wheel speed controlled separately, values between 0 to 255
// 1: Each wheel controlled separately with values between -128 to 127
// 2: Speed1 controls both motors speed, speed2 becomes the turn value, values between 0 to 255
// 3: Speed1 controls both motors speed, speed2 becomes the turn value, values between -128 to 127

int front_speed = 180;                      //129 to 255
int reverse_speed = 100;            //0 to 127
int rotational_speed = 60;          //between 0 to 127, higher indicating faster turn

int frontSensorPin = 5;                     // Front Ultrasonic sensor for obstacle avoidance
int rearSensorPin = 4;              // Rear Ultrasonic sensor for obstacle avoidance
int directionModePin = A1;          // To select which side the robot is on - blue or yellow
int pullSwitchPin = 8;              // To start the movement when pullswitch is pulled
int piezoPin = A0;                          // To play start sound

//RGB Led to indicate when system is ready and to show the selected mode
int redLedPin = 1;
int greenLedPin = A2;
int blueLedPin = A3;

unsigned long time_start;           // to record the time at which the pullswitch is pulled
int pullSwitchState = 0;            // Initialize
int directionMode = 0;              // Initialize

void setup(){
  Serial.begin(9600);
                    // Begin serial communication
  Wire.begin();                                                      // Begin I2C bus
  delay(100);                                                        // Wait for everything to power up
  setMD25Mode(MD25_Mode);
                    // Set MD25 mode
  encodeReset();
                            // Resets the encoders

  //Set all the inpout and output pins for different sensors and actuators
  pinMode(pullSwitchPin, INPUT);
  pinMode(directionModePin, INPUT);
  pinMode(frontSensorPin, INPUT);
  pinMode(rearSensorPin, INPUT);
  pinMode(redLedPin, OUTPUT);
  pinMode(greenLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);

  playStartTone();
                            // Play start tone
  setLedColor(255, 0, 0);
                    // Red color to indicate still setting up
}

//Loops quickly til pullswitched is pulled, then starts the movement and action
void loop(){
  pullSwitchState = digitalRead(pullSwitchPin);
 //1 means time to start movement, 0 means wait
  directionMode = digitalRead(directionModePin);           //1 means yellow side, 0 means blue side

  if(directionMode==0) {
      setLedColor(0, 0, 255);
       // Make LED Blue
  } else {
      setLedColor(255, 255, 0);
                    // Make LED Yellow
  }

  if(pullSwitchState == 1) {
```

```
        time_start = millis();
                  // Records time at which pullswitch pulled
        setLedColor(0, 0, 0);
                  // Set LED to off to indicate everything is fine

        //YELLOW SIDE MODE
        if(directionMode==1) {
                  performYellowSideActions();
        }
        //BLUE SIDE MODE
        else {
                  performBlueSideActions();
        }
      }
      delay(200);
    }

    // Performs the primary actions for yellow side mode
    void performYellowSideActions() {

            // Move towards module 1
            rev(360, -1);
            delay(1000);
            auto_correction(360, encoder1(), encoder2(), 1, -1, -1);
            delay(1000);
            CloseBoth();                          //Collect module 1
            delay(1000);

            // Move towards module 2
            rev(90, -1);
            delay(1000);
            leftRot(49, -1);
            delay(1000);
            auto_correction(49, encoder1(), encoder2(), 3, -1, -1);
            delay(1000);
            rev(1200, -1);
            delay(1000);
            auto_correction(1200, encoder1(), encoder2(), 2, -1, -1);
            delay(1000);
            leftRot(120, -1);
            delay(1000);
            auto_correction(120, encoder1(), encoder2(), 3, -1, -1);
            delay(1000);
            straight(135, -1);
            delay(1000);
            correct_at_milestone();       // Correct at milestone
close to the scoring placce
            delay(1000);

            //Score Module 2
            // Increase acceleration for scoring the module to make
sure it falls
            encodeReset();
            setMD25Mode(2);
            do{
            changeAccelerationRegister(5);
            writeSpeed1(220);
            }while(abs(encoder1()) < 270);
            halt();

            //Reorient robot for scoring module 1
            delay(1000);
            straight(100, -1);
            delay(1000);
            rev(188, -1);
            delay(1000);
            leftRot(268, -1);
            delay(1000);
            OpenServo1();                                          //Drop
Module 1
            OpenServo2();
            delay(500);
            straight(100, -1);
            delay(1000);
            rightRot(264, -1);
            delay(1000);
            straight(42, -1);
            delay(1000);
            correct_at_milestone();
            delay(1000);

            //Score Module 1
            // Increase acceleration for scoring the module to make
sure it falls
            encodeReset();
            setMD25Mode(2);
            do{
            changeAccelerationRegister(5);
            writeSpeed1(225);
            }while(abs(encoder1()) < 300);
            halt();

            // Make the speeds faster to finish the tasks within the
90 second time limit
            front_speed = 200;
            reverse_speed = 50;
            rotational_speed = 80;

            // Move towards module 3
            delay(1000);
            rightRot(200, -1);
            delay(1000);
            rev(700, -1);
            delay(1000);
            // Collect module 3
            CloseBoth();
            delay(1000);
            // Move towards the starting area with 3rd module
            leftRot(125, -1);
            delay(1000);
            rev(700, -1);
            delay(1000);
            rightRot(60, -1);
            delay(1000);
```

```
178             rev(1000, -1);
179             delay(1000);
180             //Drop module 3 in starting area
181             OpenServo1();
182             OpenServo2();
183             delay(1000);
184 
185             // Done with routine, wait for time up
186             waitForTimeUp();
187             delay(10000);
188     }
189 
190     // Performs the primary actions for blue side mode
191     void performBlueSideActions() {
192 
193             // Move towards module 1
194             rev(360, -1);
195             delay(1000);
196             auto_correction(360, encoder1(), encoder2(), 1, -1, -1);
197             CloseBoth();                                        //Collect
module 1
198             delay(1000);
199 
200             // Move towards module 2
201             rev(90, -1);
202             delay(1000);
203             rightRot(49, -1);
204             delay(500);
205             auto_correction(49, encoder1(), encoder2(), 2, -1, -1);
206             delay(500);
207             rev(1200, -1);
208             delay(1000);
209             auto_correction(1200, encoder1(), encoder2(), 3, -1, -1);
210             rightRot(127, -1);
211             delay(1000);
212             auto_correction(127, encoder1(), encoder2(), 2, -1, -1);
213             delay(1000);
214             straight(125, -1);
215             delay(1000);
216             correct_at_milestone();
                //Correct at milestone
217             delay(1000);
218 
219             //Score Module 2
220             // Increase acceleration for scoring the module to make
sure it falls
221             encodeReset();
222             setMD25Mode(2);
223             do{
224               changeAccelerationRegister(5);
225               writeSpeed1(220);
226             }while(abs(encoder1()) < 270);
227             halt();
228             delay(1000);
229 
230             //Reorient robot for scoring module 1
231             straight(100, -1);
232             delay(1000);
233             rev(188, -1);
234             delay(1000);
235             rightRot(268, -1);
236             delay(1000);
237             OpenServo1();                                       //Drop
Module 1
238             OpenServo2();
239             delay(500);
240             straight(100, -1);
241             delay(1000);
242             leftRot(264, -1);
243             delay(1000);
244             straight(42, -1);
245             delay(1000);
246             correct_at_milestone();
247             delay(1000);
248 
249             //Score Module 1
250             // Increase acceleration for scoring the module to make
sure it falls
251             encodeReset();
252             setMD25Mode(2);
253             do{
254             changeAccelerationRegister(5);
255             writeSpeed1(225);
256             }while(abs(encoder1()) < 300);
257             halt();
258 
259             // Make the speeds faster to finish the tasks within the
90 second time limit
260             front_speed = 200;
261             reverse_speed = 50;
262             rotational_speed = 80;
263 
264             // Move towards module 3
265             delay(1000);
266             leftrot(200, -1);
267             delay(1000);
268             rev(850, -1);
269             delay(1000);
270 
271             // Collect module 3
272             CloseBoth();
273             delay(1000);
274 
275             // Move towards the starting area with 3rd module
276             rightRot(115, -1);
277             delay(1000);
278             rev(750, -1);
279             delay(1000);
280             leftRot(58, -1);
281             delay(1000);
282             rev(1000, -1);
283             delay(1000);
284 
285             //Drop module 3 in starting area
```

```
            OpenServo1();
            OpenServo2();
            delay(1000);

            // Done with routine, wait for time up
            waitForTimeUp();
            delay(10000);
    }

    //Uses sensor readings to avoid collisions
    void avoidCollision(bool useFrontSensorReading) {
      //Check if need to use front sensor or rear sensor
      if(useFrontSensorReading){
        if(xsenseDistance1()<170) {      // if obstacle closer than
17 cm, stop
          halt();                                // Stop movement
          delay(1000);
          avoidCollision(true);          // keep checking continously
to start moving once obstacle clears
        }
      // Using rear sensor
      } else {
        if(xsenseDistance2()<170) {      // if obstacle closer than
17 cm, stop
          halt();                                // Stop movement
          delay(1000);
          avoidCollision(false); // keep checking continously to
start moving once obstacle clears
        }
      }
    }

    // Returns boolean indicating if time is up
    bool isTimeUp() {
      if(millis()-time_start >= 90000) {
          // 90 seconds TIME UP
        Serial.print("Time up");
        halt();                                          // Stop
motion
        OpenServo3();                          // FIRES THE FUNNY
ACTION
           setLedColor(200, 0, 0);             // Set LED to red to
show that time has finished
        playStopTone();                        // plays stop tone
        delay(100000000);              // Waits til someone turns
off the robot
        return true;
      } else {
          // TIME NOT UP
        return false;
      }
    }

    // Corrects at the milestone using the front ultrasonics sensor
readings
    // Milestone is when the robot is facing the lunar module scoring
area and about 8 to 10 cm away from it
    // It orients the robot in correct direction depending on the
ultrasonic readings
    void correct_at_milestone() {
          delay(200);
          // Reads the sensor readings from the 2 front sensors
          int dis1 = senseDistance1();
          int dis2 = senseDistance2();

          //Checks different cases comparing sensor reading with
expected reading of 250 mm
          if((dis1>250 and dis2>250) or (dis1<250 and dis2<250)) {
            //ALL GOOD, no need to correct
            Serial.println("all good continue...");
          } else if(dis1<250) {
            // Facing too much left, need to rotate right
            rightRot(0.4 , 160);
          } else if(dis2<250){
            // Facing too much right, need to rotate left
            leftRot(0.4, 100);
          }
    }

    //MOVE STRAIGHT
    void straight(float distance, int optional_speed){
      encodeReset();
      setMD25Mode(2);
      //set defaut speed if optional_speed is specified to be -1
      if (optional_speed<0) {
        optional_speed=front_speed;
      }
      //Continously check if time is not up and there is still
distance to travel
      while(abs(encoder1()) < distance and !isTimeUp()) {
        avoidCollision(true);                          // Uses front
sensor reading to avoid collision
        changeAccelerationRegister(1);          // To set
acceleration value
        writeSpeed1(optional_speed);            // To write speed
value - performs the movement
      }
      halt();
    }

    //MOVE BACK
    void rev(float distance, int optional_speed){
      setMD25Mode(2);
      encodeReset();
      //set defaut speed if optional_speed is specified to be -1
      if (optional_speed<0) {
        optional_speed=reverse_speed;
      }
      //Continously check if time is not up and there is still
distance to travel
      while(abs(encoder1()) < distance and !isTimeUp()){
        avoidCollision(false);                         // Uses rear
sensor reading to avoid collision
```

```
        changeAccelerationRegister(1);          // To set
acceleration value
        writeSpeed1(optional_speed);            // To write speed
value - performs the movement
      }
      halt();
    }

    //RIGHT ROTATE
    void rightRot( float distance, int optional_speed) {
      encodeReset();
      setMD25Mode(2);
      //set defaut speed if optional_speed is specified to be -1
      if (optional_speed<0) {
        optional_speed=128+rotational_speed;
      }
      //Continously check if time is not up and there is still
distance to travel
      while(abs(encoder1())<distance and !isTimeUp()){
        changeAccelerationRegister(1);          // To set
acceleration value
        writeSpeed2(optional_speed);            // To write speed
value - performs the movement
      }
      halt();
    }

    //LEFT ROTATE
    void leftRot(float distance, int optional_speed) {
      encodeReset();
      setMD25Mode(2);
      //set defaut speed if optional_speed is specified to be -1
      if (optional_speed<0) {
        optional_speed=128-rotational_speed;
      }
      //Continously check if time is not up and there is still
distance to travel
      while(abs(encoder1())<distance and !isTimeUp()){
        changeAccelerationRegister(1);          // To set
acceleration value
        writeSpeed2(optional_speed);            // To write speed
value - performs the movement
      }
      halt();
    }

    // AUTO CORRECTION CODE
    // original_distance is the actual distance the wheels were meant
to travel
    // en1 and en2 are the encoder readings after the original motion
is complete
    // m_case represents the type of original motion ...
    // m_case: 0, 1, 2 and 3 represent staraight, reverse, right and
left respectively
    // correctionSpeed to specify the speed for the correction, set -
1 to use default speeds
    // correctionPercentage to specify percentage to be corrected,
set -1 to use default
    void auto_correction(float original_distance, float en1, float
en2, int m_case, int correctionSpeed, float correctionPercentage) {

      //Sets default correction percentage for turn cases
      if(m_case==2||m_case==3) {
        if(correctionPercentage<0) {
          correctionPercentage=0.005;
        }
      //Sets default correction percentage for straight and reverse
cases
      } else {
        if(correctionPercentage<0) {
          correctionPercentage=0.01;
        }
      }
      //STRAIGHT CASE
      if(m_case==0) {
        if(abs(en1)>original_distance && abs(en2)>original_distance)
{
          // OVERSHOOT case
              rev(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        } else if (abs(en1)<original_distance &&
abs(en2)<original_distance){
              // UNDERSHOOT case
          straight(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        }
      }
      //REVERSE CASE
      else if(m_case==1) {
        if(abs(en1)>original_distance && abs(en2)>original_distance)
{
              // OVERSHOOT case
          straight(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        } else if (abs(en1)<original_distance &&
abs(en2)<original_distance){
              // UNDERSHOOT case
          rev(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        }
      }
      //RIGHT TURN CASE
      else if(m_case==2) {
        if(abs(en1)>original_distance && abs(en2)>original_distance)
{
              // OVERSHOOT case
          leftRot(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        } else if (abs(en1)<original_distance &&
abs(en2)<original_distance) {
              // UNDERSHOOT case
          rightRot(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
```

```
        }
      }
      //LEFT TURN CASE
      else if(m_case==3) {
        if(abs(en1)>original_distance && abs(en2)>original_distance)
{
              // OVERSHOOT case
          rightRot(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        } else if (abs(en1)<original_distance &&
abs(en2)<original_distance) {
              // UNDERSHOOT case
          leftRot(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
        }
      }
    }

    //Function to stop motors by sending a 128 to the speeds.
    void halt(){
      writeSpeed1(128);
      writeSpeed2(128);
      delay(500);
    }

    // Set MD25 operation MODE
    void setMD25Mode(int mode) {
      Wire.beginTransmission(MD25ADDRESS);
      Wire.write(MODE_SELECTOR);
      Wire.write(mode);
      Wire.endTransmission();
    }

    // Sets the acceleration to register
    void changeAccelerationRegister(int acc_register) {
        Wire.beginTransmission(MD25ADDRESS);
        Wire.write(ACCELERATION);
        Wire.write(acc_register);
        Wire.endTransmission();
    }

    // Sets a combined motor speed value
    void writeSpeed1(int m_speed) {
        Wire.beginTransmission(MD25ADDRESS);
        Wire.write(SPEED1);
        Wire.write(m_speed);
        Wire.endTransmission();
    }

    // Sets a combined motor speed value
    void writeSpeed2(int m_speed) {
        Wire.beginTransmission(MD25ADDRESS);
        Wire.write(SPEED2);
        Wire.write(m_speed);
        Wire.endTransmission();
    }

    //Resets the encoder values to 0
    void encodeReset(){
     Wire.beginTransmission(MD25ADDRESS);
     Wire.write(CMD);
     Wire.write(0x20); // Putting the value 0x20 to reset encoders
     Wire.endTransmission();
    }

    //Read and display value of encoder 1 as a long
    long encoder1(){
      Wire.beginTransmission(MD25ADDRESS); // Send byte to get a
reading from encoder 1
      Wire.write(ENCODERONE);
      Wire.endTransmission();

      Wire.requestFrom(MD25ADDRESS, 4); // Request 4 bytes from MD25
      while(Wire.available() < 4); // Wait for 4 bytes to arrive
      long poss1 = Wire.read(); // First byte for encoder 1, HH.
      poss1 <<= 8;
      poss1 += Wire.read(); // Second byte for encoder 1, HL
      poss1 <<= 8;
      poss1 += Wire.read(); // Third byte for encoder 1, LH
      poss1 <<= 8;
      poss1 +=Wire.read(); // Fourth byte for encoder 1, LL
      delay(50); // Wait for everything to make sure everything is
sent

      return(poss1);
    }

    //Read and display value of encoder 2 as a long
    long encoder2(){
      Wire.beginTransmission(MD25ADDRESS);
      Wire.write(ENCODERTWO);
      Wire.endTransmission();

      Wire.requestFrom(MD25ADDRESS, 4); // Request 4 bytes from MD25
      while(Wire.available() < 4); // Wait for 4 bytes to become
available
      long poss2 = Wire.read();
      poss2 <<= 8;
      poss2 += Wire.read();
      poss2 <<= 8;
      poss2 += Wire.read();
      poss2 <<= 8;
      poss2 +=Wire.read();

     return(poss2);
    }

    // SERVO CODE
    const int servoPin1 = 12;
    const int servoPin2 = 13;

    const int loopTime = 17;                    // Determines how
much the servos turn
```

```
    const int servoSpeed = 100;                 // Determines the
speed at which servos turn
    const int referenceSpeed = 1508;     // Reference value,
characteristic of the servo being used

    //We use PWM signals to control the movement of the servos

    //Closes first servo
    void CloseServo1() {
      pinMode(servoPin1, OUTPUT);
      for(int i=0; i<loopTime; i++) {
        digitalWrite(servoPin1, HIGH);
        delayMicroseconds(referenceSpeed+servoSpeed);
        digitalWrite(servoPin1, LOW);
        delay(20);
      }
    }

    //Closes second servo
    void CloseServo2() {
      pinMode(servoPin2, OUTPUT);
      for(int i=0; i<loopTime+10; i++) {
        digitalWrite(servoPin2, HIGH);
        delayMicroseconds(referenceSpeed-servoSpeed);
        digitalWrite(servoPin2, LOW);
        delay(20);
      }
    }

    //Opens second servo
    void OpenServo2() {
      pinMode(servoPin2, OUTPUT);
      for(int i=0; i<loopTime+20; i++) {
        digitalWrite(servoPin2, HIGH);
        delayMicroseconds(referenceSpeed+servoSpeed);
        digitalWrite(servoPin2, LOW);
        delay(20);
      }
    }

    //Opens first servo
    void OpenServo1() {
      pinMode(servoPin1, OUTPUT);
      for(int i=0; i<loopTime+20; i++) {
        digitalWrite(servoPin1, HIGH);
        delayMicroseconds(referenceSpeed-servoSpeed);
        digitalWrite(servoPin1, LOW);
        delay(20);
      }
    }

    //Opens servo controlling the rocket launch
    void OpenServo3() {
      pinMode(servoPin3, OUTPUT);
      for(int i=0; i<250; i++) {
        digitalWrite(servoPin3, HIGH);
        delayMicroseconds(2500);
        digitalWrite(servoPin3, LOW);
        delay(20);
      }
    }

    //Closes servo 1 and servo 2 simantenously
    void CloseBoth() {
      pinMode(servoPin1, OUTPUT);
      for(int i=0; i<25; i++) {
        digitalWrite(servoPin1, HIGH);
        digitalWrite(servoPin2, HIGH);
        delayMicroseconds(referenceSpeed-servoSpeed);
        digitalWrite(servoPin2, LOW);
        delay(2*servoSpeed);
        digitalWrite(servoPin1, LOW);
        delay(20);
      }
    }

    ////////////////////////////////////////ULTRASONIC SENSORS
CODE//////////////////////////////////////////////////
    ////////////////////////////////////The sensors used for
MILESTONES/////////////////////////////////////////////

    // RIGHT FRONT ULTRASONIC SENSOR
    const int trigPin1 = 6;
    const int echoPin1 = 7;
    //LEFT FRONT ULTRASONIC SENSOR
    const int trigPin2 = 9;
    const int echoPin2 = 10;

    // defines variables
    long duration1, duration2;
    int distance1, distance2;

    //Returns the distance to obstacle in mm detected by right front
ultrasonic sensor
    int senseDistance1() {
      pinMode(trigPin1, OUTPUT); // Sets the trigPin as an Output
      pinMode(echoPin1, INPUT); // Sets the echoPin as an Input
      Serial.begin(9600); // Starts the serial communication

      delay(50);

      // Clears the trigPin
      digitalWrite(trigPin1, LOW);
      delayMicroseconds(5);

      // Sets the trigPin on HIGH state for 10 micro seconds
      digitalWrite(trigPin1, HIGH);
      delayMicroseconds(10);
      digitalWrite(trigPin1, LOW);

      // Reads the echoPin, returns the sound wave travel time in
microseconds
      duration1 = pulseIn(echoPin1, HIGH);
```

```
680       // Calculating the distance
681       distance1= duration1*0.034/2;
682
683       // Prints the distance on the Serial Monitor
684       Serial.println("First Distance in cm: ");
685       Serial.println(distance1);
686
687       return distance1*10;
688     }
689
690     //Returns the distance to obstacle in mm detected by left front
ultrasonic sensor
691     int senseDistance2() {
692       pinMode(trigPin2, OUTPUT); // Sets the trigPin as an Output
693       pinMode(echoPin2, INPUT); // Sets the echoPin as an Input
694       Serial.begin(9600); // Starts the serial communication
695
696       delay(200);
697
698       // Clears the trigPin
699       digitalWrite(trigPin2, LOW);
700       delayMicroseconds(5);
701
702       // Sets the trigPin on HIGH state for 10 micro seconds
703       digitalWrite(trigPin2, HIGH);
704       delayMicroseconds(10);
705       digitalWrite(trigPin2, LOW);
706
707       // Reads the echoPin, returns the sound wave travel time in
microseconds
708       duration2 = pulseIn(echoPin2, HIGH);
709
710       // Calculating the distance
711       distance2= duration2*0.034/2;
712
713       // Prints the distance on the Serial Monitor
714       Serial.println("Second Distance in cm: ");
715       Serial.println(distance2);
716
717       return distance2*10;
718     }
719
720
721     ////////////////////////////////The sensors used for COLLISION
AVOIDANCE//////////////////////////////////////
722
723     //defines variables
724     double pulse1, inches1, cm1, pulse2, inches2, cm2;
725
726     //Returns distance to obstacle infront in mm
727     double xsenseDistance1() {
728       //Used to read in the pulse that is being sent by the MaxSonar
device. //Pulse Width representation with a scale factor of 147 uS per
Inch.
729       pulse1 = pulseIn(frontSensorPin, HIGH);
730       //147uS per inch
731       inches1 = pulse1/147;
732       cm1 = inches1 * 2.54;
733       Serial.print("Front sensor distance: ");
734       Serial.print(cm1);
735       Serial.print(" cm");
736       Serial.println();
737       delay(50);
738       return cm1*10;
739     }
740
741     //Returns distance to obstacle behind in mm
742     double xsenseDistance2() {
743       //Used to read in the pulse that is being sent by the MaxSonar
device. //Pulse Width representation with a scale factor of 147 uS per
Inch.
744       pulse2 = pulseIn(rearSensorPin, HIGH);
745       //147uS per inch
746       inches2 = pulse2/147;
747       cm2 = inches2 * 2.54;
748       Serial.print("Rear sensor distance: ");
749       Serial.print(cm2);
750       Serial.print(" cm");
751       Serial.println();
752       delay(50);
753       return cm2*10;
754     }
755
756     // Sets LED color
757     void setLedColor(int red, int green, int blue)
758     {
759       analogWrite(redLedPin, red);
760       analogWrite(greenLedPin, green);
761       analogWrite(blueLedPin, blue);
762     }
763
764     /*
765       MELODY CODE
766       Copied from public domain
767
768      Plays a melody
769
770      created 21 Jan 2010
771      modified 30 Aug 2011
772      by Tom Igoe
773
774     This example code is in the public domain.
775
776      http://www.arduino.cc/en/Tutorial/Tone
777
778     */
779
780     // notes in the melody:
781     int melody1[] = {
782       NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3
783     };
784
785     // note durations: 4 = quarter note, 8 = eighth note, etc.:
786     int noteDurations1[] = {
```

```
  4, 8, 8, 4
};

void playStartTone() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 4; thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations1[thisNote];
    tone(piezoPin, melody1[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(piezoPin);
  }
}

// notes in the melody:
int melody2[] = {
  NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3
};

// note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations2[] = {
  4, 8, 8, 4
};

void playStopTone() {
  // iterate over the notes of the melody:
  for (int thisNote = 0; thisNote < 4; thisNote++) {

    // to calculate the note duration, take one second
    // divided by the note type.
    //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
    int noteDuration = 1000 / noteDurations2[thisNote];
    tone(piezoPin, melody2[thisNote], noteDuration);

    // to distinguish the notes, set a minimum time between them.
    // the note's duration + 30% seems to work well:
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    // stop the tone playing:
    noTone(piezoPin);
  }
}
```