

## Jas.ino – code for the secondary robot

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

3
4  #define MD25ADDRESS      0x58                    //
Address of the MD25
5  #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
6  #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
7  #define ENCODERONE       0x02                    //
Byte to read motor encoder 1
8  #define ENCODERTWO       0x06                    //
Byte to read motor encoder 2
9  #define ACCELERATION     0xE                     //
Byte to define motor acceleration
10 #define CMD               0x10                   //
Byte to reset encoder values
11 #define MODE_SELECTOR     0xF                    //
Byte to change between control MODES
12
13 int MD25_Mode = 2;
14 //Explanation for different modes
15 // 0: Each wheel speed controlled separately, values between 0 to
255
16 // 1: Each wheel controlled separately with values between -128
to 127
17 // 2: Speed1 controls both motors speed, speed2 becomes the turn
value, values between 0 to 255
18 // 3: Speed1 controls both motors speed, speed2 becomes the turn
value, values between -128 to 127
19
20 int front_speed = 180;                            //129 to 255
21 int reverse_speed = 100;                          //0 to 127
22 int rotational_speed = 60;                        //between 0 to 127, higher
indicating faster turn
23
24 int frontSensorPin = 5;                           // Front Ultrasonic
sensor for obstacle avoidance
25 int rearSensorPin = 4;                           // Rear Ultrasonic sensor
for obstacle avoidance
26 int directionModePin = A1;                        // To select which side the
robot is on - blue or yellow
27 int pullSwitchPin = 8;                           // To start the movement
when pullswitch is pulled
28 int piezoPin = A0;                                // To play start
sound
29
```

```
30 //RGB Led to indicate when system is ready and to show the
selected mode
31 int redLedPin = 1;
32 int greenLedPin = A2;
33 int blueLedPin = A3;
34
35 unsigned long time_start;                         // to record the time at
which the pullswitch is pulled
36 int pullSwitchState = 0;                          // Initialize
37 int directionMode = 0;                            // Initialize
38
39 void setup(){
40     Serial.begin(9600);                            // Begin
serial communication
41     Wire.begin();                                   // Begin
I2C bus
42     delay(100);                                     // Wait
for everything to power up
43     setMD25Mode(MD25_Mode);                        // Set MD25 mode
44     encodeReset();                                  // Resets the encoders
45
46     //Set all the inpoint and output pins for different sensors and
actuators
47     pinMode(pullSwitchPin, INPUT);
48     pinMode(directionModePin, INPUT);
49     pinMode(frontSensorPin, INPUT);
50     pinMode(rearSensorPin, INPUT);
51     pinMode(redLedPin, OUTPUT);
52     pinMode(greenLedPin, OUTPUT);
53     pinMode(blueLedPin, OUTPUT);
54
55     playStartTone();                                // Play start tone
56     setLedColor(255, 0, 0);                        // Red color to indicate still setting up
57 }
58
59 //Loops quickly til pullswitched is pulled, then starts the
movement and action
60 void loop(){
61     pullSwitchState = digitalRead(pullSwitchPin);
62     //1 means time to start movement, 0 means wait
directionMode = digitalRead(directionModePin);      //1
means yellow side, 0 means blue side
63
64     if(directionMode==0) {
65         setLedColor(0, 0, 255);
66         // Make LED Blue
67     } else {
68         setLedColor(255, 255, 0);
69         // Make LED Yellow
70     }
71
72     if(pullSwitchState == 1) {
```

```

71     time_start = millis();
72     // Records time at which pullswitch pulled
73     setLedColor(0, 0, 0);
74     // Set LED to off to indicate everything is fine
75     //YELLOW SIDE MODE
76     if(directionMode==1) {
77         performYellowSideActions();
78     }
79     //BLUE SIDE MODE
80     else {
81         performBlueSideActions();
82     }
83     delay(200);
84 }
85
86 // Performs the primary actions for yellow side mode
87 void performYellowSideActions() {
88
89     // Move towards module 1
90     rev(360, -1);
91     delay(1000);
92     auto_correction(360, encoder1(), encoder2(), 1, -1, -1);
93     delay(1000);
94     CloseBoth(); //Collect module 1
95     delay(1000);
96
97     // Move towards module 2
98     rev(90, -1);
99     delay(1000);
100    leftRot(49, -1);
101    delay(1000);
102    auto_correction(49, encoder1(), encoder2(), 3, -1, -1);
103    delay(1000);
104    rev(1200, -1);
105    delay(1000);
106    auto_correction(1200, encoder1(), encoder2(), 2, -1, -1);
107    delay(1000);
108    leftRot(120, -1);
109    delay(1000);
110    auto_correction(120, encoder1(), encoder2(), 3, -1, -1);
111    delay(1000);
112    straight(135, -1);
113    delay(1000);
114    correct_at_milestone(); // Correct at milestone
115    close to the scoring placce
116    delay(1000);
117
118    //Score Module 2
119    // Increase acceleration for scoring the module to make
120    sure it falls
121    encodeReset();
122    setMD25Mode(2);
123    do{
124        changeAccelerationRegister(5);
125        writeSpeed1(220);

```

```

124    }while(abs(encoder1()) < 270);
125    halt();
126
127    //Reorient robot for scoring module 1
128    delay(1000);
129    straight(100, -1);
130    delay(1000);
131    rev(188, -1);
132    delay(1000);
133    leftRot(268, -1);
134    delay(1000);
135    OpenServo1(); //Drop
136
137    Module 1
138    OpenServo2();
139    delay(500);
140    straight(100, -1);
141    delay(1000);
142    rightRot(264, -1);
143    delay(1000);
144    straight(42, -1);
145    delay(1000);
146    correct_at_milestone();
147    delay(1000);
148
149    //Score Module 1
150    // Increase acceleration for scoring the module to make
151    sure it falls
152    encodeReset();
153    setMD25Mode(2);
154    do{
155        changeAccelerationRegister(5);
156        writeSpeed1(225);
157        }while(abs(encoder1()) < 300);
158        halt();
159
160    // Make the speeds faster to finish the tasks within the
161    90 second time limit
162    front_speed = 200;
163    reverse_speed = 50;
164    rotational_speed = 80;
165
166    // Move towards module 3
167    delay(1000);
168    rightRot(200, -1);
169    delay(1000);
170    rev(700, -1);
171    delay(1000);
172    // Collect module 3
173    CloseBoth();
174    delay(1000);
175    // Move towards the starting area with 3rd module
176    leftRot(125, -1);
177    delay(1000);
178    rev(700, -1);
179    delay(1000);
180    rightRot(60, -1);
181    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
198
199     module 1
200     delay(1000);
201
202     // Move towards module 2
203     rev(90, -1);
204     delay(1000);
205     rightRot(49, -1);
206     delay(500);
207     auto_correction(49, encoder1(), encoder2(), 2, -1, -1);
208     delay(500);
209     rev(1200, -1);
210     delay(1000);
211     auto_correction(1200, encoder1(), encoder2(), 3, -1, -1);
212     rightRot(127, -1);
213     delay(1000);
214     auto_correction(127, encoder1(), encoder2(), 2, -1, -1);
215     delay(1000);
216     straight(125, -1);
217     delay(1000);
218     correct_at_milestone();
219     //Correct at milestone
220     delay(1000);
221
222     //Score Module 2
223     // Increase acceleration for scoring the module to make
224     sure it falls
225     encodeReset();
226     setMD25Mode(2);
227     do{
228         changeAccelerationRegister(5);
229         writeSpeed1(220);
230     }while(abs(encoder1()) < 270);
231     halt();
232     delay(1000);
233
234     //Reorient robot for scoring module 1
235     straight(100, -1);

```

```

232     delay(1000);
233     rev(188, -1);
234     delay(1000);
235     rightRot(268, -1);
236     delay(1000);
237     OpenServo1(); //Drop
238
239     Module 1
240     OpenServo2();
241     delay(500);
242     straight(100, -1);
243     delay(1000);
244     leftRot(264, -1);
245     delay(1000);
246     straight(42, -1);
247     delay(1000);
248     correct_at_milestone();
249     delay(1000);
250
251     //Score Module 1
252     // Increase acceleration for scoring the module to make
253     sure it falls
254     encodeReset();
255     setMD25Mode(2);
256     do{
257         changeAccelerationRegister(5);
258         writeSpeed1(225);
259     }while(abs(encoder1()) < 300);
260     halt();
261
262     // Make the speeds faster to finish the tasks within the
263     90 second time limit
264     front_speed = 200;
265     reverse_speed = 50;
266     rotational_speed = 80;
267
268     // Move towards module 3
269     delay(1000);
270     leftRot(200, -1);
271     delay(1000);
272     rev(850, -1);
273     delay(1000);
274
275     // Collect module 3
276     CloseBoth();
277     delay(1000);
278
279     // Move towards the starting area with 3rd module
280     rightRot(115, -1);
281     delay(1000);
282     rev(750, -1);
283     delay(1000);
284     leftRot(58, -1);
285     delay(1000);
286     rev(1000, -1);
287     delay(1000);
288
289     //Drop module 3 in starting area

```

```

286     OpenServo1();
287     OpenServo2();
288     delay(1000);
289
290     // Done with routine, wait for time up
291     waitForTimeUp();
292     delay(10000);
293 }
294
295 //Uses sensor readings to avoid collisions
296 void avoidCollision(bool useFrontSensorReading) {
297     //Check if need to use front sensor or rear sensor
298     if(useFrontSensorReading){
299         if(xsenseDistance1()<170) {          // if obstacle closer than
17 cm, stop
300             halt();                          // Stop movement
301             delay(1000);
302             avoidCollision(true);             // keep checking continuously
to start moving once obstacle clears
303         }
304         // Using rear sensor
305     } else {
306         if(xsenseDistance2()<170) {          // if obstacle closer than
17 cm, stop
307             halt();                          // Stop movement
308             delay(1000);
309             avoidCollision(false); // keep checking continuously to
start moving once obstacle clears
310         }
311     }
312 }
313
314 // Returns boolean indicating if time is up
315 bool isTimeUp() {
316     if(millis()-time_start >= 90000) {
317         // 90 seconds TIME UP
318         Serial.print("Time up");
319         halt();                          // Stop
motion
320         OpenServo3();                     // FIRES THE FUNNY
ACTION
321         setLedColor(200, 0, 0);           // Set LED to red to
show that time has finished
322         playStopTone();                   // plays stop tone
323         delay(100000000);                 // Waits til someone turns
off the robot
324         return true;
325     } else {
326         // TIME NOT UP
327         return false;
328     }
329 }
330
331 // Corrects at the milestone using the front ultrasonics sensor
readings
332 // Milestone is when the robot is facing the lunar module scoring
area and about 8 to 10 cm away from it

```

```

333 // It orients the robot in correct direction depending on the
ultrasonic readings
334 void correct_at_milestone() {
335     delay(200);
336     // Reads the sensor readings from the 2 front sensors
337     int dis1 = senseDistance1();
338     int dis2 = senseDistance2();
339
340     //Checks different cases comparing sensor reading with
expected reading of 250 mm
341     if((dis1>250 and dis2>250) or (dis1<250 and dis2<250)) {
342         //ALL GOOD, no need to correct
343         Serial.println("all good continue...");
344     } else if(dis1<250) {
345         // Facing too much left, need to rotate right
346         rightRot(0.4 , 160);
347     } else if(dis2<250){
348         // Facing too much right, need to rotate left
349         leftRot(0.4, 100);
350     }
351 }
352
353 //MOVE STRAIGHT
354 void straight(float distance, int optional_speed){
355     encodeReset();
356     setMD25Mode(2);
357     //set default speed if optional_speed is specified to be -1
358     if (optional_speed<0) {
359         optional_speed=front_speed;
360     }
361     //Continously check if time is not up and there is still
distance to travel
362     while(abs(encoder1()) < distance and !isTimeUp()) {
363         avoidCollision(true);             // Uses front
sensor reading to avoid collision
364         changeAccelerationRegister(1);    // To set
acceleration value
365         writeSpeed1(optional_speed);      // To write speed
value - performs the movement
366     }
367     halt();
368 }
369
370 //MOVE BACK
371 void rev(float distance, int optional_speed){
372     setMD25Mode(2);
373     encodeReset();
374     //set default speed if optional_speed is specified to be -1
375     if (optional_speed<0) {
376         optional_speed=reverse_speed;
377     }
378     //Continously check if time is not up and there is still
distance to travel
379     while(abs(encoder1()) < distance and !isTimeUp()){
380         avoidCollision(false);           // Uses rear
sensor reading to avoid collision

```

```

381     changeAccelerationRegister(1);          // To set
acceleration value
382     writeSpeed1(optional_speed);           // To write speed
value - performs the movement
383     }
384     halt();
385 }
386
387 //RIGHT ROTATE
388 void rightRot( float distance, int optional_speed) {
389     encodeReset();
390     setMD25Mode(2);
391     //set default speed if optional_speed is specified to be -1
392     if (optional_speed<0) {
393         optional_speed=128+rotational_speed;
394     }
395     //Continously check if time is not up and there is still
distance to travel
396     while(abs(encoder1())<distance and !isTimeUp()){
397         changeAccelerationRegister(1);          // To set
acceleration value
398         writeSpeed2(optional_speed);           // To write speed
value - performs the movement
399     }
400     halt();
401 }
402
403 //LEFT ROTATE
404 void leftRot(float distance, int optional_speed) {
405     encodeReset();
406     setMD25Mode(2);
407     //set default speed if optional_speed is specified to be -1
408     if (optional_speed<0) {
409         optional_speed=128-rotational_speed;
410     }
411     //Continously check if time is not up and there is still
distance to travel
412     while(abs(encoder1())<distance and !isTimeUp()){
413         changeAccelerationRegister(1);          // To set
acceleration value
414         writeSpeed2(optional_speed);           // To write speed
value - performs the movement
415     }
416     halt();
417 }
418
419 // AUTO CORRECTION CODE
420 // original_distance is the actual distance the wheels were meant
to travel
421 // en1 and en2 are the encoder readings after the original motion
is complete
422 // m_case represents the type of original motion ...
423 // m_case: 0, 1, 2 and 3 represent straight, reverse, right and
left respectively
424 // correctionSpeed to specify the speed for the correction, set -
1 to use default speeds

```

```

425 // correctionPercentage to specify percentage to be corrected,
set -1 to use default
426 void auto_correction(float original_distance, float en1, float
en2, int m_case, int correctionSpeed, float correctionPercentage) {
427
428     //Sets default correction percentage for turn cases
429     if(m_case==2||m_case==3) {
430         if(correctionPercentage<0) {
431             correctionPercentage=0.005;
432         }
433     //Sets default correction percentage for straight and reverse
cases
434     } else {
435         if(correctionPercentage<0) {
436             correctionPercentage=0.01;
437         }
438     }
439     //STRAIGHT CASE
440     if(m_case==0) {
441         if(abs(en1)>original_distance && abs(en2)>original_distance)
{
442             // OVERSHOOT case
443             rev(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
444         } else if (abs(en1)<original_distance &&
abs(en2)<original_distance){
445             // UNDERSHOOT case
446             straight(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
447         }
448     }
449     //REVERSE CASE
450     else if(m_case==1) {
451         if(abs(en1)>original_distance && abs(en2)>original_distance)
{
452             // OVERSHOOT case
453             straight(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
454         } else if (abs(en1)<original_distance &&
abs(en2)<original_distance){
455             // UNDERSHOOT case
456             rev(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
457         }
458     }
459     //RIGHT TURN CASE
460     else if(m_case==2) {
461         if(abs(en1)>original_distance && abs(en2)>original_distance)
{
462             // OVERSHOOT case
463             leftRot(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);
464         } else if (abs(en1)<original_distance &&
abs(en2)<original_distance) {
465             // UNDERSHOOT case
466             rightRot(correctionPercentage*(abs(en1)-original_distance),
correctionSpeed);

```

```

467     }
468 }
469 //LEFT TURN CASE
470 else if(m_case==3) {
471     if(abs(en1)>original_distance && abs(en2)>original_distance)
472     {
473         // OVERSHOOT case
474         rightRot(correctionPercentage*(abs(en1)-original_distance),
475 correctionSpeed);
476     } else if (abs(en1)<original_distance &&
477 abs(en2)<original_distance) {
478         // UNDERSHOOT case
479         leftRot(correctionPercentage*(abs(en1)-original_distance),
480 correctionSpeed);
481     }
482 }
483 //Function to stop motors by sending a 128 to the speeds.
484 void halt() {
485     writeSpeed1(128);
486     writeSpeed2(128);
487     delay(500);
488 }
489 // Set MD25 operation MODE
490 void setMD25Mode(int mode) {
491     Wire.beginTransmission(MD25ADDRESS);
492     Wire.write(MODE_SELECTOR);
493     Wire.write(mode);
494     Wire.endTransmission();
495 }
496 // Sets the acceleration to register
497 void changeAccelerationRegister(int acc_register) {
498     Wire.beginTransmission(MD25ADDRESS);
499     Wire.write(ACCELERATION);
500     Wire.write(acc_register);
501     Wire.endTransmission();
502 }
503 // Sets a combined motor speed value
504 void writeSpeed1(int m_speed) {
505     Wire.beginTransmission(MD25ADDRESS);
506     Wire.write(SPEED1);
507     Wire.write(m_speed);
508     Wire.endTransmission();
509 }
510 // Sets a combined motor speed value
511 void writeSpeed2(int m_speed) {
512     Wire.beginTransmission(MD25ADDRESS);
513     Wire.write(SPEED2);
514     Wire.write(m_speed);
515     Wire.endTransmission();
516 }
517 }
518 }
519

```

```

520 //Resets the encoder values to 0
521 void encodeReset() {
522     Wire.beginTransmission(MD25ADDRESS);
523     Wire.write(CMD);
524     Wire.write(0x20); // Putting the value 0x20 to reset encoders
525     Wire.endTransmission();
526 }
527 //Read and display value of encoder 1 as a long
528 long encoder1() {
529     Wire.beginTransmission(MD25ADDRESS); // Send byte to get a
530     reading from encoder 1
531     Wire.write(ENCODERONE);
532     Wire.endTransmission();
533
534     Wire.requestFrom(MD25ADDRESS, 4); // Request 4 bytes from MD25
535     while(Wire.available() < 4); // Wait for 4 bytes to arrive
536     long poss1 = Wire.read(); // First byte for encoder 1, HH.
537     poss1 <<= 8;
538     poss1 += Wire.read(); // Second byte for encoder 1, HL
539     poss1 <<= 8;
540     poss1 += Wire.read(); // Third byte for encoder 1, LH
541     poss1 <<= 8;
542     poss1 += Wire.read(); // Fourth byte for encoder 1, LL
543     delay(50); // Wait for everything to make sure everything is
544     sent
545     return(poss1);
546 }
547 //Read and display value of encoder 2 as a long
548 long encoder2() {
549     Wire.beginTransmission(MD25ADDRESS);
550     Wire.write(ENCODERTWO);
551     Wire.endTransmission();
552
553     Wire.requestFrom(MD25ADDRESS, 4); // Request 4 bytes from MD25
554     while(Wire.available() < 4); // Wait for 4 bytes to become
555     available
556     long poss2 = Wire.read();
557     poss2 <<= 8;
558     poss2 += Wire.read();
559     poss2 <<= 8;
560     poss2 += Wire.read();
561     poss2 <<= 8;
562     poss2 += Wire.read();
563
564     return(poss2);
565 }
566 // SERVO CODE
567 const int servoPin1 = 12;
568 const int servoPin2 = 13;
569
570 const int loopTime = 17; // Determines how
571 much the servos turn

```

```

572     const int servoSpeed = 100;           // Determines the
speed at which servos turn
573     const int referenceSpeed = 1508;      // Reference value,
characteristic of the servo being used
574
575     //We use PWM signals to control the movement of the servos
576
577     //Closes first servo
578     void CloseServo1() {
579         pinMode(servoPin1, OUTPUT);
580         for(int i=0; i<loopTime; i++) {
581             digitalWrite(servoPin1, HIGH);
582             delayMicroseconds(referenceSpeed+servoSpeed);
583             digitalWrite(servoPin1, LOW);
584             delay(20);
585         }
586     }
587
588     //Closes second servo
589     void CloseServo2() {
590         pinMode(servoPin2, OUTPUT);
591         for(int i=0; i<loopTime+10; i++) {
592             digitalWrite(servoPin2, HIGH);
593             delayMicroseconds(referenceSpeed-servoSpeed);
594             digitalWrite(servoPin2, LOW);
595             delay(20);
596         }
597     }
598
599     //Opens second servo
600     void OpenServo2() {
601         pinMode(servoPin2, OUTPUT);
602         for(int i=0; i<loopTime+20; i++) {
603             digitalWrite(servoPin2, HIGH);
604             delayMicroseconds(referenceSpeed+servoSpeed);
605             digitalWrite(servoPin2, LOW);
606             delay(20);
607         }
608     }
609
610     //Opens first servo
611     void OpenServo1() {
612         pinMode(servoPin1, OUTPUT);
613         for(int i=0; i<loopTime+20; i++) {
614             digitalWrite(servoPin1, HIGH);
615             delayMicroseconds(referenceSpeed-servoSpeed);
616             digitalWrite(servoPin1, LOW);
617             delay(20);
618         }
619     }
620
621     //Opens servo controlling the rocket launch
622     void OpenServo3() {
623         pinMode(servoPin3, OUTPUT);
624         for(int i=0; i<250; i++) {
625             digitalWrite(servoPin3, HIGH);
626             delayMicroseconds(2500);

```

```

627             digitalWrite(servoPin3, LOW);
628             delay(20);
629         }
630     }
631
632     //Closes servo 1 and servo 2 simantenously
633     void CloseBoth() {
634         pinMode(servoPin1, OUTPUT);
635         for(int i=0; i<25; i++) {
636             digitalWrite(servoPin1, HIGH);
637             digitalWrite(servoPin2, HIGH);
638             delayMicroseconds(referenceSpeed-servoSpeed);
639             digitalWrite(servoPin2, LOW);
640             delay(2*servoSpeed);
641             digitalWrite(servoPin1, LOW);
642             delay(20);
643         }
644     }
645
646     ////////////////////////////////////ULTRASONIC SENSORS
CODE////////////////////////////////////
647     ////////////////////////////////////The sensors used for
MILESTONES////////////////////////////////////
648
649     // RIGHT FRONT ULTRASONIC SENSOR
650     const int trigPin1 = 6;
651     const int echoPin1 = 7;
652     //LEFT FRONT ULTRASONIC SENSOR
653     const int trigPin2 = 9;
654     const int echoPin2 = 10;
655
656     // defines variables
657     long duration1, duration2;
658     int distance1, distance2;
659
660     //Returns the distance to obstacle in mm detected by right front
ultrasonic sensor
661     int senseDistance1() {
662         pinMode(trigPin1, OUTPUT); // Sets the trigPin as an Output
663         pinMode(echoPin1, INPUT); // Sets the echoPin as an Input
664         Serial.begin(9600); // Starts the serial communication
665
666         delay(50);
667
668         // Clears the trigPin
669         digitalWrite(trigPin1, LOW);
670         delayMicroseconds(5);
671
672         // Sets the trigPin on HIGH state for 10 micro seconds
673         digitalWrite(trigPin1, HIGH);
674         delayMicroseconds(10);
675         digitalWrite(trigPin1, LOW);
676
677         // Reads the echoPin, returns the sound wave travel time in
microseconds
678         duration1 = pulseIn(echoPin1, HIGH);
679

```

```

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 in front 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[] = {

```



```

787     4, 8, 8, 4
788 };
789
790 void playStartTone() {
791     // iterate over the notes of the melody:
792     for (int thisNote = 0; thisNote < 4; thisNote++) {
793
794         // to calculate the note duration, take one second
795         // divided by the note type.
796         //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
797         int noteDuration = 1000 / noteDurations1[thisNote];
798         tone(piezoPin, melody1[thisNote], noteDuration);
799
800         // to distinguish the notes, set a minimum time between them.
801         // the note's duration + 30% seems to work well:
802         int pauseBetweenNotes = noteDuration * 1.30;
803         delay(pauseBetweenNotes);
804         // stop the tone playing:
805         noTone(piezoPin);
806     }
807 }
808
809 // notes in the melody:
810 int melody2[] = {
811     NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3
812 };
813
814 // note durations: 4 = quarter note, 8 = eighth note, etc.:
815 int noteDurations2[] = {
816     4, 8, 8, 4
817 };
818
819 void playStopTone() {
820     // iterate over the notes of the melody:
821     for (int thisNote = 0; thisNote < 4; thisNote++) {
822
823         // to calculate the note duration, take one second
824         // divided by the note type.
825         //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
826         int noteDuration = 1000 / noteDurations2[thisNote];
827         tone(piezoPin, melody2[thisNote], noteDuration);
828
829         // to distinguish the notes, set a minimum time between them.
830         // the note's duration + 30% seems to work well:
831         int pauseBetweenNotes = noteDuration * 1.30;
832         delay(pauseBetweenNotes);
833         // stop the tone playing:
834         noTone(piezoPin);
835     }
836 }

```