Lar.ino – Code for the main robot

```
#include <Servo.h>                                          // Servo library
#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 = 175;                      //129 to 255
int reverse_speed = 106;            //0 to 127
int rotational_speed = 60;          //between 0 to 127, higher indicating faster turn

int helperServoPin = 7;                     // Servo which allows the stored balls to move towards the shooting wheels
int frontSensorPin = 2;                     // Front Ultrasonic sensor for obstacle avoidance
int rearSensorPin = 3;              // Rear Ultrasonic sensor for obstacle avoidance
int frontMotorPin = 5;              // Front motor for collecting balls
int rearMotorsPin = 6;              // Rear motor for shooting the balls
int piezoPin = 11;                          // To play start sound
int directionModePin = 12;          // To select which side the robot is on - blue or yellow
int pullSwitchPin = 13;                     // To start the movement when pullswitch is pulled

// RGB Led to indicate when system is ready and to show the selected mode
int redLedPin = 10;
int greenLedPin = 9;
int blueLedPin = 8;


unsigned long time_start;           // to record the time at which the pullswitch is pulled
int pullSwitchState = 0;
int directionMode = 0;
int angleH = 1;                             // helper servo position in degrees

Servo helperServo;                          // Servo which allows the stored balls to move towards the shooting wheels

//Initialize and set everything up
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(frontMotorPin, OUTPUT);
  pinMode(rearMotorsPin, OUTPUT);
  pinMode(pullSwitchPin, INPUT);
  pinMode(directionModePin, INPUT);
  pinMode(frontSensorPin, INPUT);
  pinMode(rearSensorPin, INPUT);
  pinMode(redLedPin, OUTPUT);
  pinMode(greenLedPin, OUTPUT);
  pinMode(blueLedPin, OUTPUT);
  helperServo.attach(helperServoPin);

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

```
69        setHelperServo();
                              // To put helper servo in correct starting
position
70      }
71
72      //Loops quickly til pullswitched is pulled, then starts the
movement and action
73      void loop(){
74        pullSwitchState = digitalRead(pullSwitchPin);
        //1 means time to start movement, 0 means wait
75        directionMode = digitalRead(directionModePin);           //1
means yellow side, 0 means blue side
76
77        if(directionMode==0) {
78            setLedColor(0, 0, 255);
                // Make LED Blue
79        } else {
80            setLedColor(255, 255, 0);
                      // Make LED Yellow
81        }
82        if(pullSwitchState == 1) {
83          time_start = millis();
                      // Records time at which pullswitch pulled
84          setLedColor(0, 0, 0);
                      // Set LED to off to indicate everything is fine
85
86          //YELLOW SIDE MODE
87          if(directionMode==1) {
88                      performYellowSideActions();
89          }
90          //BLUE SIDE MODE
91          else {
92                      performBlueSideActions();
93          }
94        }
95        delay(200);
96      }
97
98      // Performs the primary actions for yellow side mode
99      void performYellowSideActions() {
100
101             delay(20000);  // Wait for the smaller robot to go out of
the way
102
103             // Move over the 5 titanium ores location
104             straight(800, 150);
105             delay(1000);
106             auto_correction(750, encoder1(), encoder2(), 0, -1, -1);
107             delay(1000);
108             leftRot(43, -1);
109             delay(1000);
110             auto_correction(43, encoder1(), encoder2(), 3, -1, -1);
111             delay(1000);
112             straight(250, -1);
113             delay(100);
114             auto_correction(250, encoder1(), encoder2(), 0, -1, -1);
115             delay(1000);
116             leftRot(81, -1);
117             delay(100);
118             auto_correction(79, encoder1(), encoder2(), 3, -1, -1);
119             delay(100);
120
121             // Start the front motor to push the titanium ores into
the storage
122             frontMotorStart();
123             delay(3000);
124             straight(700, 170);
125             delay(1000);
126             frontMotorStop();
127             delay(1000);
128
129             // Position towards the net
130             rightRot(45, -1);
131             delay(500);
132             rev(100, -1);
133             delay(500);
134
135             // Start shooting into the net
136             rearMotorsStart();
137             delay(4000);
138             openHelperServo();
139             delay(1000);
140             rearMotorsStop();
141             delay(100);
142             closeHelperServo();
143             delay(100);
144
145             //Trip 2 towards the large collection of titanium ores
146             straight(200, -1);
147             delay(100);
148
149             // Start collection of titanium ores
150             frontMotorStart();
151             delay(3000);
152             straight(200, -1);
153             delay(1000);
154             frontMotorStop();
155             delay(1000);
156
157             //Position towards net
158             rev(200, -1);
159             delay(100);
160             rightRot(2, -1);
161             delay(100);
162             rev(200, -1);
163             delay(100);
164
165             //Start shooting
166             rearMotorsStart();
167             delay(4000);
168             openHelperServo();
169             delay(1000);
```

```
            rearMotorsStop();
            delay(100);

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

    // Performs the primary actions for blue side mode
    void performBlueSideActions() {

            delay(20000);  // Wait for the smaller robot to go out of
the way

            // Move over the 5 titanium ores location
            straight(800, 150);
            delay(1000);
            auto_correction(750, encoder1(), encoder2(), 0, -1, -1);
            delay(1000);
            rightRot(50, -1);
            delay(1000);
            auto_correction(50, encoder1(), encoder2(), 2, -1, -1);
            delay(1000);
            straight(300, -1);
            delay(100);
            auto_correction(300, encoder1(), encoder2(), 0, -1, -1);
            delay(1000);
            rightRot(100, -1);
            delay(100);
            auto_correction(85, encoder1(), encoder2(), 2, -1, -1);
            delay(100);

            // Start the front motor to push the titanium ores into
the storage
            frontMotorStart();
            delay(3000);
            straight(650, 200);
            delay(1000);
            frontMotorStop();
            delay(1000);

            // Position towards the net
            leftRot(55, -1);
            delay(500);
            rev(100, -1);
            delay(500);

            // Start shooting into the net
            rearMotorsStart();
            delay(4000);
            openHelperServo();
            delay(1000);
            rearMotorsStop();
            delay(100);
            closeHelperServo();
            delay(100);

            //Trip 2 towards the large collection of titanium ores
            straight(900, -1);
            delay(100);

            // Start collection of titanium ores
            frontMotorStart();
            delay(3000);
            straight(200, 200);
            delay(1000);
            frontMotorStop();
            delay(1000);

            //Position towards net
            rev(200, 80);
            delay(100);
            rightRot(2, -1);
            delay(100);
            rev(1200, -1);
            delay(100);

            //Start shooting
            rearMotorsStart();
            delay(4000);
            openHelperServo();
            delay(1000);
            rearMotorsStop();
            delay(100);

            // 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(senseDistance1()<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(senseDistance2()<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();
        frontMotorStop();
        rearMotorsStop();
        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;
      }
    }

    // Wait indefinitely for time to be up once the routine is
performed
    void waitForTimeUp() {
      while(true) {
        isTimeUp();
        delay(1000);
      }
    }

    void frontMotorStart() {
      digitalWrite(frontMotorPin, HIGH);
    }

    void rearMotorsStart() {
      digitalWrite(rearMotorsPin, HIGH);
    }

    void rearMotorsStop() {
      digitalWrite(rearMotorsPin, LOW);
    }

    void frontMotorStop() {
      digitalWrite(frontMotorPin, LOW);
    }

    // Used for testing purposes only
    void rearMotorTest() {
           for(int i=0; i<100; i++) {
                  digitalWrite(rearMotorsPin, HIGH);
                  delayMicroseconds(100);
                  digitalWrite(rearMotorsPin, LOW);
                  delayMicroseconds(100);
           }
    }

    // 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);
        }
      }
    }


    //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();
    }

    //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);
}

// ULTRASONIC CODE

// FRONT ULTRASONIC SENSOR
double pulse1, inches1, cm1;

//Returns distance to obstacle infront in mm
double senseDistance1() {
  //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.
  pulse1 = pulseIn(frontSensorPin, HIGH);
  //147uS per inch
  inches1 = pulse1/147;
  cm1 = inches1 * 2.54;
  Serial.print("Front sensor distance: ");
  Serial.print(cm1);
  Serial.print(" cm");
  Serial.println();
  delay(50);
  return cm1*10;
}

// REAR ULTRASONIC SENSOR
double pulse2, inches2, cm2;
```

```
567     //Returns distance to obstacle behind in mm
568     double senseDistance2() {
569       //Used to read in the pulse that is being sent by the MaxSonar
device
570       //Pulse Width representation with a scale factor of 147 uS per
Inch.
571       pulse2 = pulseIn(rearSensorPin, HIGH);
572       //147uS per inch
573       inches2 = pulse2/147;
574       cm2 = inches2 * 2.54;
575       Serial.print("Rear sensor distance: ");
576       Serial.print(cm2);
577       Serial.print(" cm");
578       Serial.println();
579       delay(50);
580       return cm2*10;
581     }
582
583     // Sets LED color
584     void setLedColor(int red, int green, int blue)
585     {
586       analogWrite(redLedPin, red);
587       analogWrite(greenLedPin, green);
588       analogWrite(blueLedPin, blue);
589     }
590
591     // HELPER SERVO CODE
592
593     //Opens helper Servo
594     void openHelperServo()
595     {
596       // scan from 0 to 180 degrees
597       for(angleH = 0; angleH < 80; angleH++)
598       {
599         helperServo.write(angleH);
600         delay(15);
601       }
602
603       delay(500);
604     }
605
606     //Sets helper servo to the initial position
607     void setHelperServo()
608     {
609           helperServo.write(1);
610           delay(500);
611     }
612
613     // Closes helper servo
614     void closeHelperServo()
615     {
616       // now scan back from 180 to 0 degrees
617       for(angleH = 79; angleH > 1; angleH--)
618       {
619         helperServo.write(angleH);
620         delay(15);
621       }
622
623       delay(500);
624     }
625
626     /*
627       MELODY CODE
628       Copied from public domain
629
630      Plays a melody
631
632      created 21 Jan 2010
633      modified 30 Aug 2011
634      by Tom Igoe
635
636     This example code is in the public domain.
637
638      http://www.arduino.cc/en/Tutorial/Tone
639
640     */
641
642     // notes in the melody:
643     int melody1[] = {
644       NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3
645     };
646
647     // note durations: 4 = quarter note, 8 = eighth note, etc.:
648     int noteDurations1[] = {
649       4, 8, 8, 4
650     };
651
652     void playStartTone() {
653       // iterate over the notes of the melody:
654       for (int thisNote = 0; thisNote < 4; thisNote++) {
655
656         // to calculate the note duration, take one second
657         // divided by the note type.
658         //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
659         int noteDuration = 1000 / noteDurations1[thisNote];
660         tone(piezoPin, melody1[thisNote], noteDuration);
661
662         // to distinguish the notes, set a minimum time between them.
663         // the note's duration + 30% seems to work well:
664         int pauseBetweenNotes = noteDuration * 1.30;
665         delay(pauseBetweenNotes);
666         // stop the tone playing:
667         noTone(piezoPin);
668       }
669     }
670
671     // notes in the melody:
672     int melody2[] = {
673       NOTE_C4, NOTE_G3, NOTE_G3, NOTE_A3
674     };
675
676     // note durations: 4 = quarter note, 8 = eighth note, etc.:
```

```
677   int noteDurations2[] = {
678     4, 8, 8, 4
679   };
680
681   void playStopTone() {
682     // iterate over the notes of the melody:
683     for (int thisNote = 0; thisNote < 4; thisNote++) {
684
685       // to calculate the note duration, take one second
686       // divided by the note type.
687       //e.g. quarter note = 1000 / 4, eighth note = 1000/8, etc.
688       int noteDuration = 1000 / noteDurations2[thisNote];
689       tone(piezoPin, melody2[thisNote], noteDuration);
690
691       // to distinguish the notes, set a minimum time between them.
692       // the note's duration + 30% seems to work well:
693       int pauseBetweenNotes = noteDuration * 1.30;
694       delay(pauseBetweenNotes);
695       // stop the tone playing:
696       noTone(piezoPin);
697     }
698   }
```