Team Redbull: A wing-based seek and flag vehicular system for mine detection.

Tyler Helmus, Ryan Clark, Jiajun Cao, Ansh Verma
School of Mechanical Engineering, Purdue University.

ABSTRACT
This report describes a vehicular system used for detecting and flagging ferrous blocks. The system consists of a capacitance based sensing mechanism laid over a deployable wing and a servo motor controlled cantilever arm to flag the block using electromagnets. Ultrasonic sensors were used to create a feedback loop for complementing the steering module. This system was used to seek and flag dummy butterfly mines (120mm X 20mm X 61mm) laid in a 7ft X 7ft work zone.

INTRODUCTION
The robot had to perform specific tasks stated by our course instructor which influenced our design rationales. It had to be autonomous and be less than $400. They must fit into a 12in X 12in X 12in cube for shipping and should autonomously deploy into the work zone. The robot should mark 4 dummy mines all within 90 seconds, in a manner that does not affect the color, texture, moisture content or friction coefficient of the test area. The following decisions were made by the team based on the above mentioned constraints:

Design Rationale
- *Sensing*: Since the dummy mines were made of ferrous material, we leveraged capacitive sensing to detect the location of the mine. This provided us with high fidelity sensing making the detection robust.
- *Deployable Wings*: In order to complete the task in 90 seconds, we increased our sensing area by spreading the capacitive electrodes over a wing like structure that would sweep the area underneath for mines. Also to cater for the size constraint of the robot, we made the wings deployable.
- *Flagging*: Since the mines were ferrous, we leveraged electromagnets to act as flags that can easily stick to mines in case of error releases. Our feeder mechanism for the flagging system is designed and printed to provide quick allocation of the magnets.
- *Dual Core*: The robot runs on two Arduinos (Mega and UNO). One Arduino controls the steering mechanism and the capacitive sensing, whereas the other consisted of feeder and flagging mechanism. This division of task was done to provide multitasking ability to the robot for quicker speed.

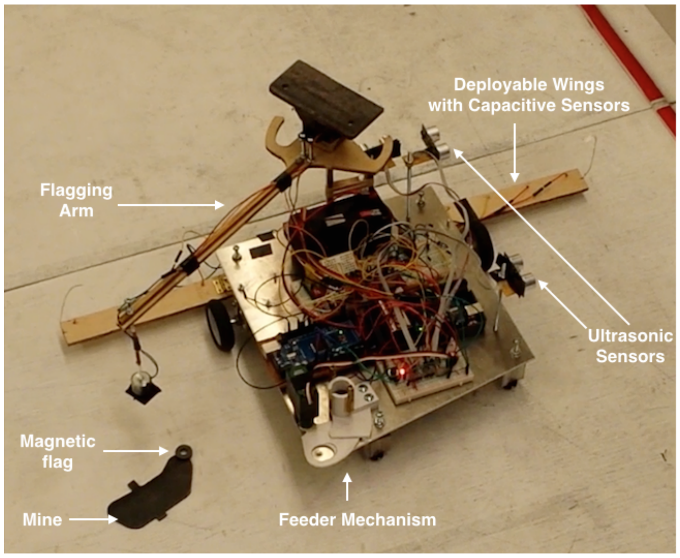


Figure 1. The robot sensing and dropping the mine

THE ROBOT
The chasis of the robot consisted of a Aluminum plate, 2 caster wheels, 2 motorized wheels, a wooden deployable wing and a wooden cantilever arm. The aluminum plate housed majority of the electronics and modules in order to provide mechanical stability to the system. Here we discuss the steering, mine sensing and flagging mechanism.

Steering and Guidance
The path of robot is designed as shown in Figure 1. The robot starts at a corner oriented approximately 30 with the edge of the wall, then goes forward a little and makes a right turn followed by the deployment of the wings, which is realized by rotation of the cantilever. After that the robot sweeps the arena for mine around the perimeter and comes to a stop once it reaches the center.

Three HC-SR04 ultrasonic sensors are used for feedback control of the steering. One is mounted in the front of the robot to measure the distance to the front wall, while the other two are mounted on the right side of the robot to measure distance to the side wall. The robot responds to the measurement from the front sensor to make a turn when hitting the wall and calculates the differences between two side sensors to make adjustment of the relative speed of the two wheels to make it parallel to the wall.

Two DC motors and a L298 H-bridge are used as actuators for the motion of robot. Two front wheels are driven by the motors whose speed can be adjusted independently. While mov-

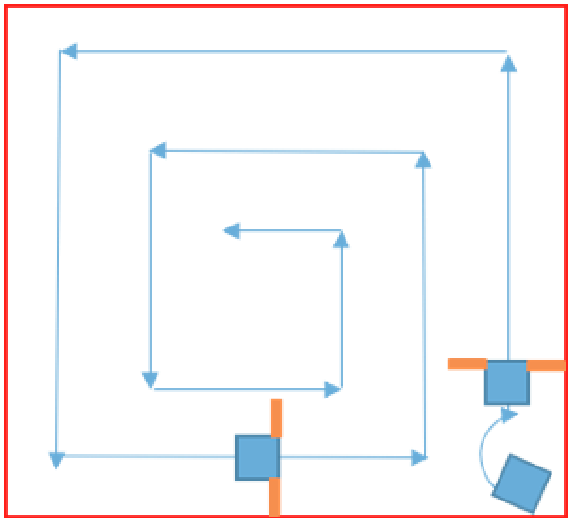


Figure 2. Path of the robot

ing forward, the speed of two wheels are tuned in accordance to the overall RPM required and the feedback from the side ultrasonic sensors to make sure the robot is moving straight. As for turning, the right wheels rotate forward and the left wheels rotate backwards in half speed to make a smooth turn in place. This process is open loop and the timing is adjusted by experiment. Therefore, the robot doesn't respond to mine detection during turn. However, since the turn is made in place, all the area will be covered during forward moving and no mine will be ignored in the arena.

Due to design of marking mechanism of the robot, it has to move forward when a mine is detected by one of the capacitive sensing pin on the wings. The front ultrasonic sensor is used for measuring the distance change from the front of the robot to the wall to make sure the robot is accurately positioned. The connection of three ultrasonic sensors, H-bridge and motors are detailed in wiring diagram in appendices. The principle of ultrasonic sensor is that it sends out ultrasonic wave pulses once triggered and return the time elapsed when the wave pulses are received. The distance to object can then be calculated as:

where (v)sound the speed of sound in air 340 m/s, t is the time elapsed between triggering the sensor the receiving the echo signals and d is the distance to the object detected. Due to the time delay and interference in receiving an echo signal, the readings from the ultrasonic sensors have many noises and slow response to change of environment. Hence, a filter is designed to make the measurements from the sensor reliable.

The default time to wait for an echo signal received is set to be 1 sec, which is reduced to 0.08 sec to get rid of any measurements that exceeds the maximum dimensions of the arena. Any ultrasonic sensor that doesn't give a reasonable reading within 0.08 sec will be assigned a fake measurement of the length/width of the arena. Besides that, due to the path design for the robot, the maximum range required for side sensors is half the length/width of the arena. Therefore, any readings that exceeds this range are considered noises and corrected the last reasonable reading of the ultrasonic sensor. Lastly, in order to make sure the robot wont make a turn right after another due to reading delays of ultrasonic sensor, the algorithm of turning is suspended for 2 seconds after a successful turn.

With the measurements of sensors corrected, the steering of the robot is made stable through a proportional controller that adjust the relative speed of wheels through feedback signal of differences of the two side ultrasonic readings. In order to make the self-correction response fast enough, the wheels are allowed to rotate in opposite direction if there is big discrepancy between the readings of the two sensors, i.e. the robot has a large deviation from a straight orientation.

While the robot is moving forward, it will act to external interrupt triggered by the capacitive touch sensors for mine detection. Once triggered, the robot will move forward accordingly based on the position of the pins triggered on wings and come to a full stop once it is accurately positioned. There will be a time delay of 5 seconds to wait for cantilever with the electromagnets to pick up a maker and place it on the mine.

Mine Sensing

The system comprises of MPR 121 Capacitive Touch Sensor [1]. The communication between the MPR 121 and the micro-controller is based on the I^{2}C serial protocol that performs measurement and signal analysis. The breakout board is a multiplexer that provides 12 capacitive electrodes that can detect a digital signal when a conductive body comes in contact with it. Wires from these 12 electrodes are laid on a deployable wing at equal spaces over a length of 24in. Also the wing is close enough to the ground so that the system distinctly detects an active event when passing over a mine.

Flagging

The purpose of this mechanism was to distribute flags which took the form of small disk shaped magnets. These were chosen as they would stick to the target when dropped from a height of several inches as well as providing us with some room for error due to the magnetic attraction between the flags and mines. The means of distribution for these flags needed to be able to span the breadth of the deployed wings and as such an additional mechanism was required to lift the flags to the appropriate positions. This task was accomplished through the use of a wooden arm that was manipulated via a servo. At the end of the arm was an electromagnet that picked up the flags and released them when the arm was in place. These magnets were dispensed by a servo operated magnet feeder whose purpose was to store and then distribute one magnet at a time for the electromagnet to pick up.

The flag distributing arm was controlled by a Futaba S3003 [2] servo which was capable of 180 of rotation while outputting 44 ounce-inches of torque at 5 volts. This servo was powered by a LM7805 voltage regulator [3] that stepped down a 9V battery to a constant 5V output. For signal smoothing, a 100F capacitor was introduced in parallel to this supply. The

[1]https://www.sparkfun.com/products/9695
[2]http://www.es.co.th/schemetic/pdf/et-servo-s3003.pdf
[3]https://www.sparkfun.com/datasheets/Components/LM7805.pdf

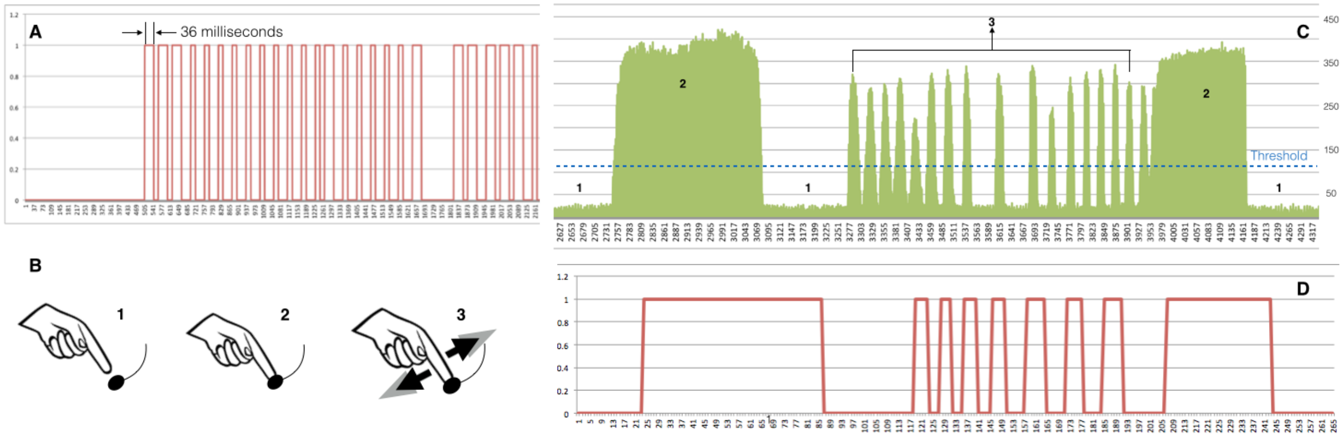


Figure 3. The Mine Senor testing (A) shows the frequency of switching state between the digital values when an event is initiated (36 milliseconds) (B) The simulations of events for the experiments, (C) Capacitance values of the conductive ink in picofarad at off state (1), prolonged touched (2) and then a series of touch and no touch states (3), (D) Representative digital output of the sensor on events from (C).

servo position was controlled by PWM port 9 on the flagging Arduino. This port utilized the servo.h library to move the servo to the appropriate angle based on the responses from the capacitive touch sensor. The range of calculated angles for given capacitive touch sensor responses can be seen in Table 1. The Arduino interpreted the capacitive touch sensors as a string of letters ranging from a to k which represented the available wires. This was then passed through a switch case statement that represented each case as a different angle the servo arm needed to rotate to. When no mine was being detected, the arm would default to the home position that was located directly above the magnet feeder.

The electromagnet was a W M3 ZYE1-P20/15 which requires a 12V supply in order to induce 2.5 kg of lift. This power was supplied directly from the drive trains 12V battery and regulated via a N-Channel MOSFET. This MOSFET was regulated by digital pin 6 on the Arduino with its source terminal connected to ground and the drain terminal connected to the negative wire of the electromagnet. Whenever a HIGH signal was output from the Arduino, the electromagnet's circuit would complete and thus it would charge to pick up the magnet. This HIGH signal would persist until the servo reached its final destination at which time it would go LOW and release the magnet. It is also worth noting that additional delays were added such that the servo would complete the pickup and dropping process before beginning the arms movement. An additional consideration when designing this system was the electromagnet's polarity. Depending on how the connections were made, the magnets would either be repelled or attracted to the electromagnet. This simply means the magnets could only be loaded in one way which was thus tested for correctness before each run. Additionally, magnets would tend to remain attached to the electromagnet after being picked up as the electromagnet had an iron core. To fix this issue, a spacer of electric tape was applied such that the inductive electromagnetic force was enough to pick up the magnet while the magnets weight was able to overcome the residual magnetic attraction after the electromagnet was switched off.

The final component of the flagging mechanism was the magnet feeder. This was composed of a 3D printed assembly and a Vex 3-Wire Servo [4] which operated off of the same 5V terminal as the arm servo. The feeder servo was set to rotate 90 out and back every time after the arm returned home from dropping a magnet. The 3D printed assembly itself consists of 3 components: the Magnet Tower, Magnet Track, and Servo Attachment. The Magnet Tower served to hold a stack of magnets and was designed with a slot such that a user can see how many magnets are left. This slot additionally enables a user to easily remove remaining magnets from the tower which would otherwise require the robot to be flipped over. The Magnet Tracks purpose on the other hand was to retain a feeding magnet such that it did not get stripped during the feeding process due to the attractive force of the stack of magnets in the Tower. Through testing, the distance this attraction would affect the fed magnets was 2 inches and as such that was the radius the servo would act on. The final component was the Servo Attachment which was designed to press fit onto the key of the Vex servo. This device would then strip magnets from the bottom of the stack one at a time as the servo was actuated. This piece was modified later during testing with a Dremel in order to finalize necessary clearances.

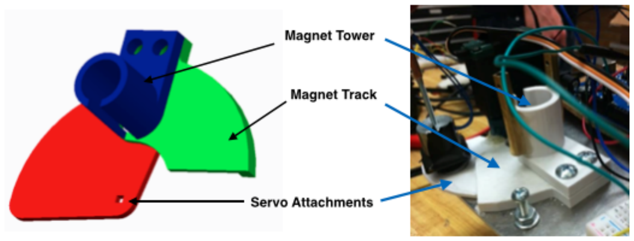


Figure 4. The Feeder Mechanism

RESULT

The capacitive electrodes were subjected a response test that reported a time of 36 milliseconds. On examining the data sheet of the device, we observed the threshold value of the

[4]http://www.vexrobotics.com/276-2162.html

capacitance to make the digital state high was 110 picofarads for the MPR121 board.

The robot was also tested on the field which reported the following :

- *Run 1 - 4 mines detected and flagged in 70 secs.*
- *Run 2 - 0 mines detected, amended the code in order to speed it up.*
- *Run 3 - 1 mine detected and flagged 120 secs.*
- *Championship - 3 mines detected and flagged.*

DISCUSSION

The capacitive sensors provided a high fidelity discreet sensing towards ferrous block of materials. As their threshold values were 110pf, it was appropriate enough the discreetly detect the touch state of the mine ignore any other false inputs. Also since they were simply detecting the touch events, this helped us reduce the computational expense of the robot to understand the location of the mine.

Limitation: This sensor configuration provided an overall robust system but we realized some limitations in the current system. The flagging mechanism is not algorithmically aware of the mines it has already flagged. The wheels and the caster provided some surface resistance which made overall velocity of the robot slower. Thus by switching to bigger wheels and smother casters we can work on increasing the speed. This also affected the cantilever feeder as lack of smooth transition led to vibrations in the arm.

Another aspect of this project was to understand the applicability of this system in actual scenario. To cater to that we made the sensing mechanism modular. So the user has to attach the appropriate probes (capacitive electrodes) suiting the environment. They can use smaller probes for regular terrain or use big probes that penetrate into the sand for terrains like desert. We can even employ capacitive proximity sensor, that can detect mines from a distance for deeper excavations.

CONCLUSION

We presented a system for sensing and flagging ferrous mines using capacitive digital multiplexers. We feed electromagnets to the flagging cantilever arm that would rotate to the location of the mine and drop the magnet on it. Ultrasonic sensors were attached to the periphery of the system to correct and navigate the robot in the desired path. In the spirit of making the system easily reproducible, we have made the project open-source by publishing our codes, circuit diagrams and CAD file at **`https://github.com/anshverma/MineBot`**. This report also consists of an Appendix which attaches the codes, circuit diagram, the calculation and the budget report for this robot.

ACKNOWLEDGMENT

We would like to thank Prof. Jeff Shelton and his teaching assistant Justin Jacob Padinjaremury for providing the necessary guidance in completing this project.

APPENDIX - A (Codes)

A.1) Code for Feeder

```
//
// This code was authored by Ansh Verma, Ryan Clark, Tyler Helmus and Jiajun
// Cao for the feeder mechanism, MPR 121 mine detection and Flagging Arm
// Servo of the Mine-robot for the course completion of ME 588 at Purdue
// University.
//

#include "mpr121.h"          //Import capacitive touch sensor library
#include <Wire.h>
#include <Servo.h>           // Import servo library

#define ARM_SPEED 10         // arm speed
#define ARM_HOME 71          // arm electromagnet-magnet pickup position
#define PICKUP_POS 80        // flag dispensor servo position for
electromagnet-magnet pickup
#define RETREIVE_POS 0       // flag dispensor servo position of the magnet
reservoir
#define FLAG_SPEED 1         // flag dispensor servo speed
#define MAG_LOAD 1700        // delay for magnet to drop into slot
#define MAG_LIFT_DELAY 1500 // ms delay which ensures magnet is lifted prior
to arm movin
#define WARN_LIGHT 7         // define pins and states with arbitrary value
#define MAGNET 6             //Define pin for magnet
#define IRQ_PIN 4            // Define pin for IRQ on capacitive touch
#define ARM_SERVO 9          //Define pin for PWM signal to arm servo
#define FLAG_SERVO 10        //Define pin for PWM signal to flagging servo
#define ARM_MOVE 500         //Define delay for arm movement to complete
#define BUTTON 8             //Define pin for start up sensing
#define MINE_DETECT 1000     //Define delay for robot to be in position prior
to lifting magnet
#define COMM A0              //Define pin for turning communication from drive
arduino: stops flagging when turning

Servo armServo, flagServo; //Name servo

int armPos = ARM_HOME;
boolean touchStates[12]; //to keep track of the previous touch states

void setup() {
  pinMode(IRQ_PIN, INPUT); //enable IRQ pin
  digitalWrite(IRQ_PIN, HIGH); //enable pullup resistor

  //Ryan Insert Begin---- Declare Pins
  armServo.attach (ARM_SERVO);//Main Servo
  flagServo.attach (FLAG_SERVO);//Mag Feed Servo
  pinMode(MAGNET, OUTPUT);//Enable magnet pin
  pinMode(BUTTON, INPUT);//enable start button
  pinMode(COMM, INPUT);//Define pin for turning communication
  digitalWrite(MAGNET, LOW);//Define starting LOW state for the magnet

  Serial.begin(9600);
  Wire.begin();
  armServo.write(ARM_HOME);//Start with servo in the home position
  getNextFlag();//Function to actuate magnet feeder servo to get the next
flag
  mpr121_setup();//setput the capacitive touch sensor
  while(!digitalRead(BUTTON)){//Look for when the start button is pushed
    delay(5);
  }
  delay(2500);
  deployWings();//deploy the wings in the start sequence

}

void loop()
{
  if(digitalRead(COMM)){ //If the robot is not turning, continue on with the
switch case and subsequent flagging procedure
  if(digitalRead(BUTTON)){ //Read startup button
    while(true){
      delay(10);
    }
  }

  Wire.requestFrom(0x5A, 2);//printing touch state from the capacitive touch
sensor
  byte LSB = Wire.read();
  byte MSB = Wire.read();
  uint16_t touched = ((MSB << 8) | LSB);
  Serial.println(touched);


  //BEGIN SWITCH CASE------------------------------
  switch (touched) {
  case 1:
    Serial.print("a");//check touch state to see which wire was touched
    delay(MINE_DETECT);//Delay for wheels to stop rotating
    digitalWrite(MAGNET, HIGH);//Turn electromagnet on
    delay(MAG_LIFT_DELAY);//Delay to ensure the magnet has been lifted up to
the electromagnet
    armPosition(168);//Move the arm servo to the necessary location
    delay(ARM_MOVE);//Wait for the servo to reach this location
    digitalWrite(MAGNET, LOW);//Turn off the electromagnet to release the
flag
    delay(MAG_LIFT_DELAY);//Ensure the magnet has been dropped
    armReturn();//Return the arm to the home position
    getNextFlag();//Actuate the magnet feeder servo
    break;
  case 2:
    Serial.print("b");
    delay(MINE_DETECT);
    digitalWrite(MAGNET, HIGH);
    delay(MAG_LIFT_DELAY);
    armPosition(145);
        delay(ARM_MOVE);
    digitalWrite(MAGNET, LOW);
    delay(MAG_LIFT_DELAY);
    armReturn();
    getNextFlag();
    break;
```

```
case 4:
  Serial.print("c");
  delay(MINE_DETECT);
  digitalWrite(MAGNET, HIGH);
  delay(MAG_LIFT_DELAY);
  armPosition(134);
      delay(ARM_MOVE);
  digitalWrite(MAGNET, LOW);
  delay(MAG_LIFT_DELAY);
  armReturn();
  getNextFlag();
  break;
case 8:
  Serial.print("d");
  delay(MINE_DETECT);
  digitalWrite(MAGNET, HIGH);
  delay(MAG_LIFT_DELAY);
  armPosition(119);
  delay(ARM_MOVE);
  digitalWrite(MAGNET, LOW);
  delay(MAG_LIFT_DELAY);
  armReturn();
  getNextFlag();
  break;
case 16:
  Serial.print("e");
  delay(MINE_DETECT);
  digitalWrite(MAGNET, HIGH);
  delay(MAG_LIFT_DELAY);
  armPosition(109);
  delay(ARM_MOVE);
  digitalWrite(MAGNET, LOW);
  delay(MAG_LIFT_DELAY);
  armReturn();
  getNextFlag();
  break;
case 32:
  Serial.print("f");
  delay(MINE_DETECT);
  digitalWrite(MAGNET, HIGH);
  delay(MAG_LIFT_DELAY);
  armPosition(99);
  delay(ARM_MOVE);
  digitalWrite(MAGNET, LOW);
  delay(MAG_LIFT_DELAY);
  armReturn();
  getNextFlag();
  break;
case 64:
  Serial.print("g");
  delay(MINE_DETECT);
  digitalWrite(MAGNET, HIGH);
  delay(MAG_LIFT_DELAY);
  armPosition(90);
  delay(ARM_MOVE);
  digitalWrite(MAGNET, LOW);
  delay(MAG_LIFT_DELAY);
  armReturn();
```

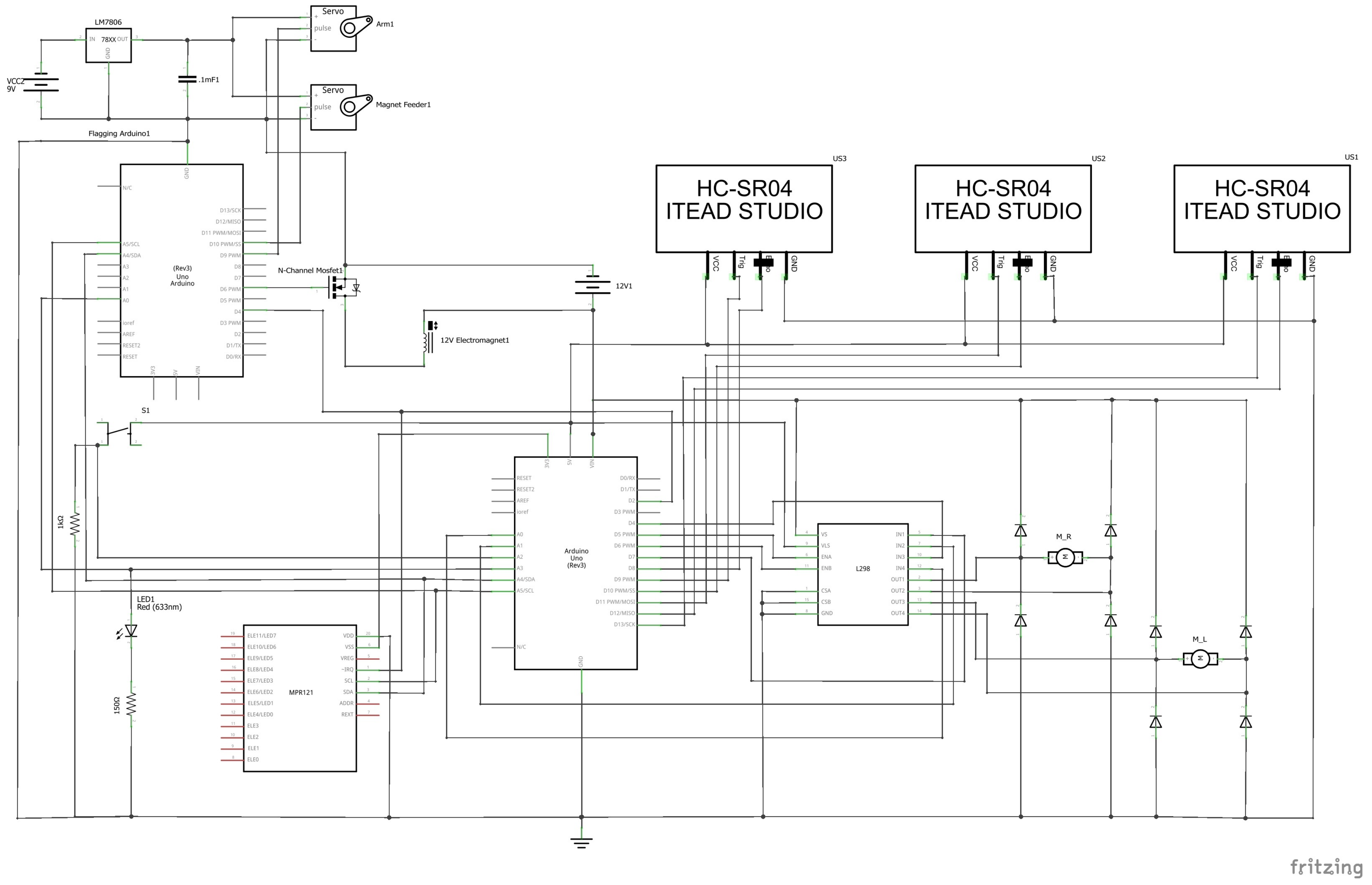
```
    getNextFlag();
    break;
  case 128:
    Serial.print("h");
    delay(MINE_DETECT);
    digitalWrite(MAGNET, HIGH);
    delay(MAG_LIFT_DELAY);
    armPosition(85);
    delay(ARM_MOVE);
    digitalWrite(MAGNET, LOW);
    delay(MAG_LIFT_DELAY);
    armReturn();
    getNextFlag();
    break;
  case 256:
    Serial.print("i");
    delay(MINE_DETECT);
    digitalWrite(MAGNET, HIGH);
    delay(MAG_LIFT_DELAY);
    armPosition(65);
    delay(ARM_MOVE);
    digitalWrite(MAGNET, LOW);
    delay(MAG_LIFT_DELAY);
    armReturn();
    getNextFlag();
    break;
  case 512:
    Serial.print("j");
    delay(MINE_DETECT);
    digitalWrite(MAGNET, HIGH);
    delay(MAG_LIFT_DELAY);
    armPosition(60);
    delay(ARM_MOVE);
    digitalWrite(MAGNET, LOW);
    delay(MAG_LIFT_DELAY);
    armReturn();
    getNextFlag();
    break;
  case 1024:
    Serial.print("k");
    delay(MINE_DETECT);
    digitalWrite(MAGNET, HIGH);
    delay(MAG_LIFT_DELAY);
    armPosition(48);
    delay(ARM_MOVE);
    digitalWrite(MAGNET, LOW);
    delay(MAG_LIFT_DELAY);
    armReturn();
    getNextFlag();
    break;
  case 2048:
    Serial.print("l");
    delay(MINE_DETECT);
    digitalWrite(MAGNET, HIGH);
    delay(MAG_LIFT_DELAY);
    armPosition(34);
    delay(ARM_MOVE);
    digitalWrite(MAGNET, LOW);
    delay(MAG_LIFT_DELAY);
    armReturn();
    getNextFlag();
    break;
  case 0:
    Serial.print("Default");
    digitalWrite(MAGNET, LOW);//Default position with magnet off
    break;
  }

  }
}


void mpr121_setup(void) {

  set_register(0x5A, ELE_CFG, 0x00);

  // Section A - Controls filtering when data is > baseline.
  set_register(0x5A, MHD_R, 0x01);
  set_register(0x5A, NHD_R, 0x01);
  set_register(0x5A, NCL_R, 0x00);
  set_register(0x5A, FDL_R, 0x00);

  // Section B - Controls filtering when data is < baseline.
  set_register(0x5A, MHD_F, 0x01);
  set_register(0x5A, NHD_F, 0x01);
  set_register(0x5A, NCL_F, 0xFF);
  set_register(0x5A, FDL_F, 0x02);

  // Section C - Sets touch and release thresholds for each electrode
  set_register(0x5A, ELE0_T, TOU_THRESH);
  set_register(0x5A, ELE0_R, REL_THRESH);

  set_register(0x5A, ELE1_T, TOU_THRESH);
  set_register(0x5A, ELE1_R, REL_THRESH);

  set_register(0x5A, ELE2_T, TOU_THRESH);
  set_register(0x5A, ELE2_R, REL_THRESH);

  set_register(0x5A, ELE3_T, TOU_THRESH);
  set_register(0x5A, ELE3_R, REL_THRESH);

  set_register(0x5A, ELE4_T, TOU_THRESH);
  set_register(0x5A, ELE4_R, REL_THRESH);

  set_register(0x5A, ELE5_T, TOU_THRESH);
  set_register(0x5A, ELE5_R, REL_THRESH);

  set_register(0x5A, ELE6_T, TOU_THRESH);
  set_register(0x5A, ELE6_R, REL_THRESH);

  set_register(0x5A, ELE7_T, TOU_THRESH);
  set_register(0x5A, ELE7_R, REL_THRESH);

  set_register(0x5A, ELE8_T, TOU_THRESH);
  set_register(0x5A, ELE8_R, REL_THRESH);
  set_register(0x5A, ELE9_T, TOU_THRESH);
  set_register(0x5A, ELE9_R, REL_THRESH);

  set_register(0x5A, ELE10_T, TOU_THRESH);
  set_register(0x5A, ELE10_R, REL_THRESH);

  set_register(0x5A, ELE11_T, TOU_THRESH);
  set_register(0x5A, ELE11_R, REL_THRESH);

  // Section D
  // Set the Filter Configuration
  // Set ESI2
  set_register(0x5A, FIL_CFG, 0x04);

  // Section E
  // Electrode Configuration
  // Set ELE_CFG to 0x00 to return to standby mode
  set_register(0x5A, ELE_CFG, 0x0C);  // Enables all 12 Electrodes


  // Section F
  // Enable Auto Config and auto Reconfig
  /*set_register(0x5A, ATO_CFG0, 0x0B);
   set_register(0x5A, ATO_CFGU, 0xC9);  // USL = (Vdd-0.7)/vdd*256 = 0xC9
@3.3V   set_register(0x5A, ATO_CFGL, 0x82);  // LSL = 0.65*USL = 0x82 @3.3V
   set_register(0x5A, ATO_CFGT, 0xB5);*/  // Target = 0.9*USL = 0xB5 @3.3V

  set_register(0x5A, ELE_CFG, 0x0C);

}


//boolean checkInterrupt(void) {
//  return digitalRead(IRQ_PIN);
//}


void set_register(int address, unsigned char r, unsigned char v) {
  Wire.beginTransmission(address);
  Wire.write(r);
  Wire.write(v);
  Wire.endTransmission();
}

void armPosition(int target) {//Code to move the arm after a mine is detected
  int pos;//position pulled from switch case
  armPos = target;//current position of arm so we can see which direction arm
must turn
  if (armPos > ARM_HOME) {//Turn counterclockwise
    for (pos = ARM_HOME; pos <= armPos; pos++) {//velocity control for servo
to move to desired position
      armServo.write (pos);
      delay(ARM_SPEED);//Delay which dictates the arm speed
    }
  }
  if (armPos < ARM_HOME) {//Same as above except for clockwise movement
    for (pos = ARM_HOME; pos >= armPos; pos--) {
      armServo.write (pos);
      delay(ARM_SPEED);
    }
  }
}

void armReturn() {//Speed controlled return function after flag is dropped
  int pos;
  if (armPos > ARM_HOME) {
    for (pos = armPos; pos >= ARM_HOME; pos--) {
      armServo.write(pos);
      delay(ARM_SPEED);
    }
  }
  if (armPos < ARM_HOME) {
    for (pos = armPos; pos <= ARM_HOME; pos++) {
      armServo.write(pos);
      delay(ARM_SPEED);
    }
  }
}

void getNextFlag() {//Function to actuate the magnet feeder

  flagServo.write(RETREIVE_POS);//Move to get a magnet
  delay(MAG_LOAD);//Delay to get in position
  flagServo.write(PICKUP_POS);//Return to the home position with a new flag
}

void deployWings(){//startup function to delploy the wings
  armPosition(34);//drop right wing
  delay(200);
  armReturn();
  armPosition(110);//drop left wing
  delay(200);
  armReturn();//return arm to home position
}
```

A.1) Code for Steering

```
#include "utility.h"
#include "Wire.h"

//define states
#define FORWARD     0    //S0(FW): robot goes forward
#define TURNING     1    //S1(TN): robot turns clockwise
#define ALIGNMENT   2    //S2(AL): robot aligns itself to go straight
#define MARKING     3    //S3(MK): robot deploys marking procedure (call
                        //subroutine mark()<---several inputs subject to
                        //change)
#define STOP        4    //S4(ST): robot comes to a stop

//define inputs
int ST = 0;       //straght (1)/not straight (0)
int NW = 0;       //near wall (1)/not near wall (0)
int CS = 0;       //mine detected (1)/mine not detected (0)
int EOT = 0;       //turning completed (1)/turning not completed (0)
int CT = 0;       //count of mines = 3 (1)/count of mines < 3 (0)
int ER = 0;      //error occurred (1)/no error (0)


//initialize the states
int state = FORWARD;  //start from S0(FW)

//define on/off button state
int button_state = LOW; //button default OFF


//define pins
#define start_button_pin A2
#define indicator_LED_pin A3

#define US1_trigPin 13
#define US1_echoPin 12
#define US2_trigPin 11
#define US2_echoPin 10
#define US3_trigPin 9
#define US3_echoPin 8


//define instances for ultrasonic sensors
ultrasonic US1;      //front ultrasonic sensors
ultrasonic US2;      //side-front ultrasonic sensor
ultrasonic US3;      //side-rear ultrasonic sensor

//define other parameters
float t = 0.5;     //tolerance for discrepancy of readings of two ultrasonic
                   //sensors (US2&US3) (in cm)
float offset = 0; //offset distance of ultrasonice sensors 2 &3
int dtw=22;       //distance to wall where the robot turns, increase after
                  //every turn
int bwd=18;       //backward distance to ensure a safe turn
int RPM=200;      //'RPM' of motor represented in PWM
int RPM_change;   //'change of 'RPM' for alignment purpose
int RPM_change_max = 3*RPM; //maximum allowable adjustment of RPM
float k_align=13;   //alignment RPM adjustment coefficent:
                    //RPM_change=k_align*(US2.dist()-US3.dist())
int turn_count=0;  //count of turn per cycle around perimeter
int pass_count=0; //count of passes around the perimeter


float us1_dist_last=60;
float us2_dist_last=60;
float us3_dist_last=60;

int *cap_pin_reading;  //capacitive sensor readings (array of 0 and 1s)
int num_of_mine;  //number of mine detected instantaneously
int num_of_mine_last;
float *x_mine; //array of mine positions
float l=29.2; //length of the beam
float *fwd_dis; //forward distance for marking
float current_dis; //initial distance to wall at the start of marking

long previousMillis_turn = 0;  //time that last successful turn completed
long previousMillis_mark = 0;  //time that last successfu marking completed

int start_flag = 1;  //flag for wing deployment procedure



void setup(){
  pinMode(start_button_pin,INPUT);
  pinMode(indicator_LED_pin,OUTPUT);

  //Serial.begin(9600);

  motor_init();                            //initialize the motors (H-bridge)

  US1._init(US1_trigPin,US1_echoPin);      //read raw measurements from the
ultrasoninc sensors
  US2._init(US2_trigPin,US2_echoPin);
  US3._init(US3_trigPin,US3_echoPin);

  mpr121_init();                          //initialize the capacitive touch
sensor mpr121

  digitalWrite(indicator_LED_pin,LOW);


}


void loop(){
  if (digitalRead(start_button_pin)){                             //start/
shut button
    button_state = !button_state;
    while (digitalRead(start_button_pin)) delayMicroseconds(1);
  }

  if (button_state){
    digitalWrite(indicator_LED_pin,HIGH);                         //the
indicator is on when the robot is moving and not in a turn, the same TTL
signal is sent to the other Arduino for making mechanism

    if (start_flag ==1){                                          //
deployemnt procedure
      forward(RPM);
      delay(800);
      turnRight(RPM);
      delay(1000);
      Stop();
      delay(3500);                                                //delay for
wing deployment
      start_flag =0;
    }


    float us1_dist = US1.dist();
    float us2_dist = US2.dist()-offset;
    float us3_dist = US3.dist();

    //if (us1_dist>2000) us1_dist = us1_dist_last;              //digital
filter to eliminate any unreasonable noises
    if (us2_dist>1000) us2_dist = us2_dist_last;
    if (us3_dist>1000) us3_dist = us3_dist_last;

    us1_dist_last = us1_dist;
    us2_dist_last = us2_dist;
    us2_dist_last = us2_dist;

    //Serial.print("Front distance = ");
    //Serial.print(us1_dist);
    //Serial.print(" cm.   ");
    //Serial.print("Side front distance = ");
    //Serial.print(us2_dist);
    //Serial.print(" cm.   ");
    //Serial.print("Side back distance = ");
    //Serial.print(us3_dist);
    //Serial.println(" cm.");

    //forward(RPM);
    //Serial.println(pass_count);

    cap_pin_reading = readTouchInputs();                  //read the logical
inputs arrays from mpr121
    num_of_mine_last = num_of_mine;                       //store the number
of mine detected last time (only for purpose of storing data during state
transition)
    num_of_mine = num_of_mine_detected(cap_pin_reading);  //calculate for
effective mines detected
//    for (int i=0;i<12;i++){
//      Serial.print("pin");
//      Serial.print(i);
//      Serial.print("=");
//      Serial.print(*(cap_pin_reading+i));
//      Serial.print("   ");
//    }
```

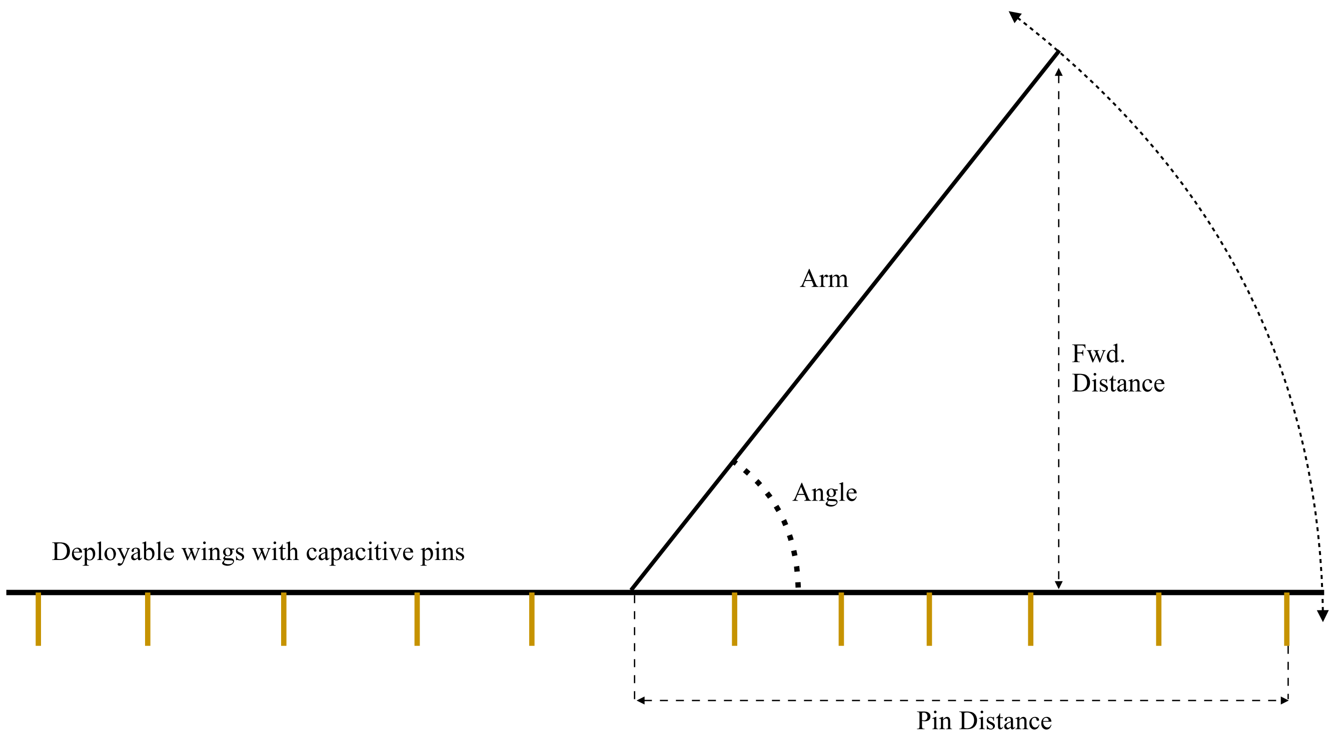
```
    //read inputs
    if (num_of_mine==1 &&  (millis()-previousMillis_mark>2000)){      //
calcualte the position of mine and send signal to marking if more than 1 mine
is detected and detected after a sufficient time after the previous detection
      CS=1;                                                           //
signals of mine detected
      x_mine = mine_position(cap_pin_reading, num_of_mine);           //
position of the mines
      fwd_dis=fwd_distance(cap_pin_reading, num_of_mine);             //
forward distance the robot has to travel
//      Serial.print("Mine position:  ");
//      for (int i=0;i<num_of_mine;i++){
//        Serial.print(*(x_mine+i));
//        Serial.println("  ");
//        Serial.print("forward distance:  ");
//        Serial.print(*(fwd_dis+i));
//        Serial.println("  ");
//      }
    }
    else{
      CS = 0;
      //Serial.println("no mine detected");
      //Serial.println();
    }



    if (abs(us2_dist-us3_dist)<t) ST=1; //judge if the robot is straight
    else ST=0;

    if (us1_dist<=dtw && (millis()-previousMillis_turn>2000)) NW=1;
//judge if a turn is required (can't be activated right after a successful
//turn)
    else NW=0;

    //in-state algorithm
    switch(state){                        //state of moving forward
      case FORWARD:
        forward(RPM);
        if (!NW && ST && !CS){
          state=FORWARD;
        }
        else if (!NW && !ST && !CS){
          state=ALIGNMENT;
        }
        else if (NW && !CS ){
          state=TURNING;
        }
        else{
          state=MARKING;
        }
        break;

      case TURNING:        //state of tuning (turn right in this case)
         digitalWrite(indicator_LED_pin,LOW); //turn off the LED and send a
                                    //signal to another Arduino not to mark
         reverse(RPM);
         while(US1.dist()<dtw+bwd && button_state){
             if (digitalRead(start_button_pin)){ //start/shut button
               button_state = !button_state;
               while (digitalRead(start_button_pin)) delayMicroseconds(1);
             }
             //Serial.print("U1_distance = ");
             //Serial.print(US1.dist());
             //Serial.println(" cm.   ");
         };
         if (button_state){
           turnLeft(RPM);
           delay(900);
           turn_count = turn_count +1;
           if ((turn_count == 3)  && (pass_count<3)){ //count the turn and
            //pass, and adjust the distance threshold for the robot to turn
            if (pass_count == 0){
              dtw =dtw +50;
            }
            else{
              dtw = dtw +40;
            }
            bwd = 0;
            turn_count = -1;
            pass_count = pass_count +1;
           }
         }
         previousMillis_turn = millis();
         if (pass_count >=2){
           state = STOP;
         }
         else state=FORWARD;
         digitalWrite(indicator_LED_pin,HIGH);

        break;



      case ALIGNMENT:
//state of the alignment (make the robot go straight)
         RPM_change = k_align*(us2_dist-us3_dist);
        if (RPM_change >RPM_change_max) RPM_change = RPM_change_max;
//set up a saturation for adjustment
        if (RPM_change <-RPM_change_max) RPM_change = -RPM_change_max;
        //Serial.print(RPM_change);
        //Serial.print("  ");
        //Serial.print(RPM+RPM_change);
        //Serial.print("  ");
        //Serial.println(RPM-RPM_change);
        alignment(RPM+RPM_change,RPM-RPM_change);
        if (!NW && ST && !CS){
          state=FORWARD;
        }
        else if (!NW && !ST && !CS){
          state=ALIGNMENT;
        }
        else if(NW && !CS ){
          state=TURNING;
        }
        else{
          state = MARKING;
        }

        break;



      case MARKING:        //state of marking
//      Serial.println("detected");
//      Serial.println(num_of_mine_last);
        current_dis = us1_dist;
        float fwd_dis_seq[num_of_mine_last];  //calculate the fwd distance
        float temp;
        for (int i=0;i<num_of_mine_last;i++){
          fwd_dis_seq[i]=*(fwd_dis+i);
        }
        for (int i=0;i<num_of_mine_last;i++){
          for (int j=i;j<num_of_mine_last;j++){
            if (*(fwd_dis_seq+i)>*(fwd_dis_seq+j)){
              temp = *(fwd_dis_seq+i);
              fwd_dis_seq[i]=*(fwd_dis_seq+j);
              fwd_dis_seq[j]=temp;
            }
          }
        }

        for (int i=0;i<num_of_mine_last;i++)//travel forward the distance
                                            //required
        {
          while (US1.dist() > (current_dis - *(fwd_dis_seq+i))+5){
            forward(0.5*RPM);
          }
          //Serial.println("marking");
          Stop();
          delay(5000);          //wait for the other Arduino to mark
        }
        state = FORWARD;
        previousMillis_mark = millis();
        break;

      case STOP:              //state of stop
        Stop();
        state = STOP;
        break;

    }
  } //end of operational program


  else{
    digitalWrite(indicator_LED_pin,LOW);        //shut off the indicator
    Stop();                                     //shut the motor
  }
}
```

APPENDIX - B (Circuit Diagram)



APPENDIX - C (Calculations)

Label	Angles (Deg)	Pin Distance (Cm)	Fwd Dist. (Cm)
a	168	24.2	16.34
b	145	19.4	21.82
c	134	14.6	25.29
d	119	9.8	27.51
e	109	4.8	28.2
f	99	0	29.2
g	90	-4.7	28.82
h	85	-9.5	27.61
i	65	-14.3	25.46
j	60	-20.1	21.18
k	48	-23.9	16.78
l	34	-28.6	5.89



APPENDIX - D (Budget Sheet)

Mounting Bracket	Pololu item #1084	1	$ 7.95	$ 7.95
Motors with Encoders	Pololu item #1445	2	$ 39.95	$ 79.90
Ultrasonic Module Distance Sensor for Arduino	HC-SR04	3	$4.99	$ 14.97
Ball Caster with 1/2" Metal Ball	953	2	$3.99	$ 7.98
Tamiya 70145 Narrow Tire Set (2 tires)	Pololu item # 63	1	$ 7.25	$ 7.25
Wheel hubs	Pololu item #1083	1	$ 7.95	$ 7.95
MPR121 Capacitive Touch Sensor		1	$ 9.95	$ 9.95
Electromagnet		1	$ 7.99	$ 7.99
Disk Magnets	Pack of 6	2	$ 1.47	$ 2.94
Hardware and Tube		1	$ 14.76	$ 14.76
Hardware and Epoxy		1	$ 8.30	$ 8.30
Electromagnet	W M3 ZYE1-P20/15	1	$ 4.45	$ 4.45
Arm Servo	Futaba S3003	1	$ 10.99	$ 10.99
Magnet Feeder Servo	Vex 3-Wire Servo Module	1	$ 19.99	$ 19.99
5V Voltage Regulator	LM7805	1	$ 1.00	$ 1.00
MOSFET	P45NF06	1	$ 1.00	$ 1.00
2 Switch DIP Switches		2	$ 0.50	$ 1.00
12V Connectors/Wires		2	$ 1.00	$ 2.00
12V Battery	Power Patrol SLA1005	1	$ 7.00	$ 7.00
9V Battery		1	$ 5.00	$ 5.00
Diode	N4007	8	$ 0.12	$ 0.96
Pushbutton	12mm Pushbutton Switch	1	$ 0.50	$ 0.50
Red LED		1	$ 0.50	$ 0.50
.1 mF Capacitor		1	$ 0.25	$ 0.25
Ardunino Uno		2	$ 24.99	$ 49.98
Resitors		2	$ 0.25	$ 0.50
		Total	.	$ 275.06