Sumo Robot

Report

This is the project report for the robot “Interpreter” made for the class MAE 6194

Kalpesh Patil
kalpeshpatil33@gwu.edu

Kanishke Gamagedara
kanishkegb@gwu.edu

Spring 2017

The George Washington University

Contents

1 Summary 3

2 Physical Structure 4

3 Sensors and Components 5

3.1 Line Detection 5

3.2 Sonar Sensors 6

3.3 Motors 7

4 Attacking Mechanism 9

5 Wireless LCD Serial Monitor 10

6 Algorithm 12

7 Future Developments 13

8 List of Components 14

9 Appendix 1: Sumo Robot Code 15

10 Appendix 2: Wireless LCD Serial Monitor Code 21

Summary

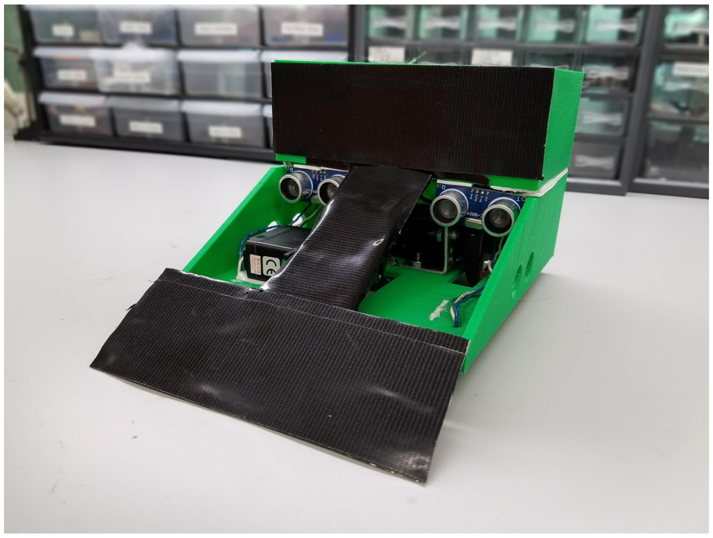


Figure 1: Interpreter

We made the “Interpreter” for the Sumo Robot Competition, which was held as a part of the course MAE 6194. This robot has an LCD serial monitor, which is operated through XBee, which displays (or interprets) everything it does. This lead to the name Interpreter. The basic functions and features of the robot are listed below.

- Finds the opponent using sonar sensors
- Detects the line using QTI sensors
- If and opponent is detected, accelerates towards it
- If the opponent is close enough, raise the flipping shield
- Send every task the robot is performing through XBee
- Another Arduino board reads the transmitted data via an XBee, and prints to an LCD display

This report is organized as follows. First, we will discuss the details on the body structure and the sensors. Then we will be explaining how the attacking mechanism works. Next section is on details of the LCD serial monitor system which is used to continuously monitor and debug the robot. Then we will provide an overview of the overall algorithm which is used to control the robot. Finally the issues we faced and the proposed future developments. The codes are attached in the appendix.

Physical Structure

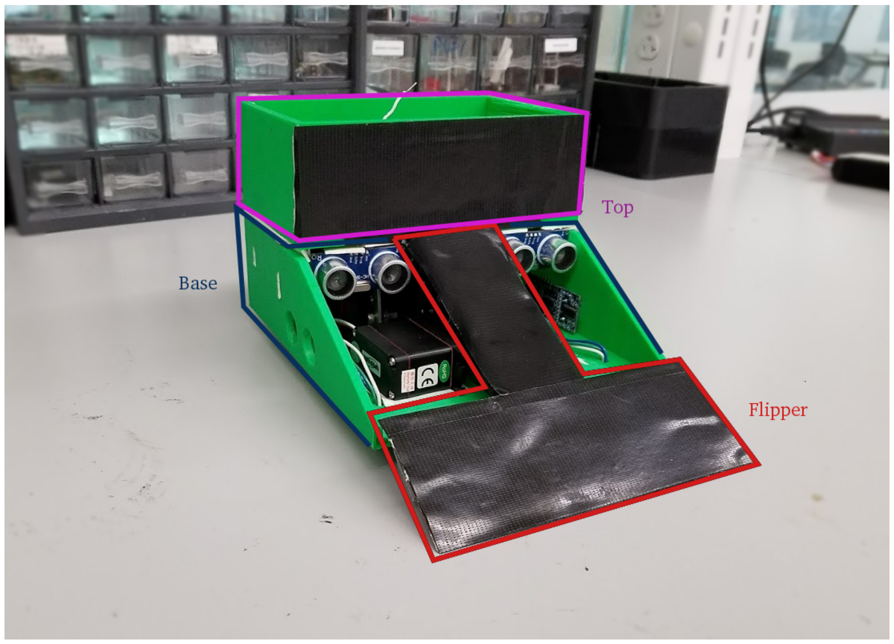


Figure 2: Structure

In this section, we discuss about the body structure of the robot and the sensors. The robots structure mainly has three parts.

1. Base
2. Top
3. Flipper

Base holds all the motors, back and side sonar sensors, and the QTI sensors at the bottom. On top of the Base, we have placed the Top which securely holds the Arduino board, batteries, Xbee unit, and the two front sonar sensors. Flipper is hinged to the front bottom edge of the Top and a servo motor with an arm is placed on the base, directly under the flipper, to activate the attacking mechanism. Top and the Base was 3D-printed and the flipper was cut from a thin wood board and a sheet metal piece at the bottom edge.

Sensors and Components

A physical structure of a robot is nothing without sensing or actuation devices. In this section, we describe the sensors and components we used on our robot.

Line Detection

The most important objective of a Sumo Robot is to detect the boundary line of the ring. For this, we used QTI sensors by Parallax[1], where QTI stands for Q - charge, T - transfer and, I - infrared. We used three QTI sensors, one in the center and two in the front left and front right.

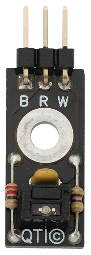


Figure 3: QTI Sensor

Referring fig. 3, pin W is supplied with 5 V and pin B is connected to the ground. Pin R is the sensor pin. As name implies, the sensor need to be charged. This is done by setting the sensor pin to HIGH during the initialization. In the case of middle QTI sensor, we need to shutdown the robot as soon as it detects the white line. For this, we used interrupts. Example code can be found below.

```
void setup() {
    pinMode(MIDDLE_pinQTIsensor, INPUT);  // set sensor pin as an input
    digitalWrite(MIDDLE_pinQTIsensor, HIGH);  // charging phase of the QTI

    attachInterrupt(digitalPinToInterrupt(MIDDLE_pinQTIsensor), MIDDLE_whiteLineISR, LOW);
    // attach the interrupt
 }

void MIDDLE_whiteLineISR() {
    goStop();  // stop the robot as soon as middle QTI detects a line
}
```

[1]https://www.parallax.com/product/555-27401

Sonar Sensors

The next objective of the robot is to identify the opponent robot. We used five HC-SR04 sonar sensors[2]. Two sensors were used in the front, and other three were used on left, right and back. The sensor sends a sonar signal and receives it back. The time difference between sending and receiving is used to calculate the distance of and obstacle. The sensor has a range of 2 - 500 cm for obstacle detection.

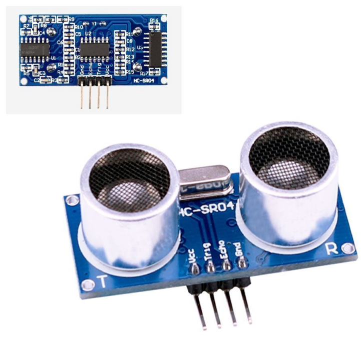


Figure 4: HC-SR04

This sensor has four pins: power, ground, trigger, and echo. Power in is supplied with 5 V. We need to write the pulse to the trigger pin and then read the echo. For this we use built-in Arduino function "pulseIn"[3]. This function returns the time the length of the pulse in microseconds. Since we know that the speed of sound is $340\ m/s = 0.034\ cm/\mu s$, we can calculate the distance to the obstacle.

```
int FRONT_distance = 0  // initialize the distance to an obstacle facing front
                        // sensor as a global variable

void FRONT_HC() {
    // writes the pulse
    digitalWrite(FRONT_trigPin, LOW);
    digitalWrite(FRONT_trigPin, HIGH);
    delay(100);
    digitalWrite(FRONT_trigPin, LOW);

    FRONT_duration = pulseIn(FRONT_echoPin, HIGH);  // read the echo
    FRONT_distance = (FRONT_duration / 2) * 0.034;  // calculate the distance
}

void loop {
    FRONT_HC();  // call the front sonar function
}
```

[2] https://goo.gl/ZwPIIL
[3] https://www.arduino.cc/en/Reference/pulseIn

Motors

Motors are the actuators of our robot. We used three motors: two for the motion of the robot and one for the attacking mechanism. For the wheel motors, we use Parallax continuous rotation servo motors[4]. Each of the motors has a power pin (5 V), ground pin, and a servo pin. In this section we mainly focus on wheel motors. The motor we used for the attacking mechanism is described in section 4.

Figure 5: Parallax continuous rotation servo motor

First thing you have to do with the servo is to calibration. For example, servo motor is supposed to stop when the servo pin is written 90. But, there might be cases where the motor might not be 100% stopped when 90 is written to the servo pin. To calibrate, we have to write 90 to servo and rotate the small screw until the motor is completely stopped. We have two servo motors facing opposite sides. The list of servo pin values that need to written to each motor is given in table 1

Table 1: Servo pin values to be written for each operation

Operation	Left motor	Right motor
Forward	0	190
Backward	190	0
Stop	90	90
Left rotate	0	0
Right rotate	190	190

Further, depending on the time period you write these values, we can determine the angle of rotation. If you left rotate for 200 ms and 400 ms, robot will turn 90 and 180 degrees respectively.

Below is snippet gives a basic idea on this.

[4] https://www.parallax.com/product/900-00008

```
// define servo pins as global variables
#define LEFT_servo = 9
#define RIGHT_servo = 10

void setup() {
    // attach the servos to servo object
    leftServo.attach(LEFT_servo);
    rightServo.attach(RIGHT_servo);
}

void goRight() {
    leftServo.write(190);
    rightServo.write(190);
}

void goStop() {
    leftServo.write(90);
    rightServo.write(90);
}

void loop() {
    // rotate 90 degrees right
    goRight();
    delay(200);

    // stop
    goStop();

    // rotate 180 degrees right
    goRight();
    delay(200);
}
```

Attacking Mechanism

In this section, we explain details about the attacking mechanism installed in our robot. Our mechanism is a simpler flipper in front of the robot. The flipper as described in fig. 1 is in rest position when it is in non-attacking mode. The front edge is touching the ground so that the opponent might ride up the top surface of the flipper (refer fig. 6).

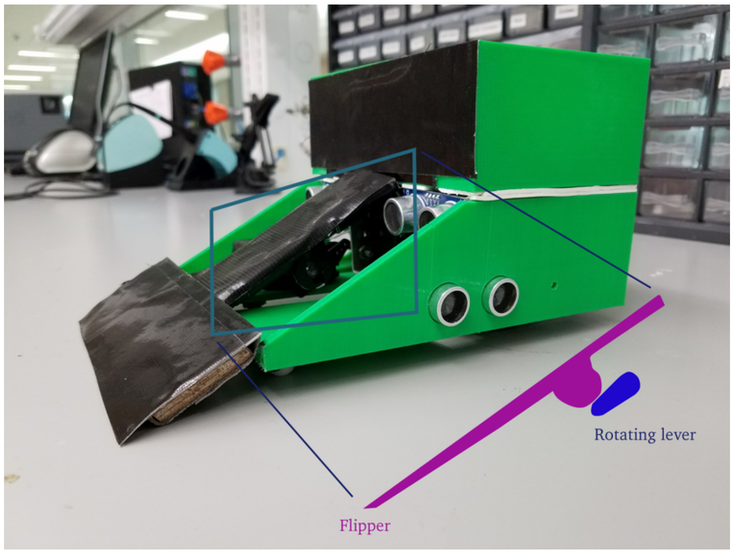


Figure 6: Flipper in resting position

As soon as the front sonar sensors detect an object closer than 8 cm, the attack servo motor starts rotating rising the flipper (refer fig. 7).

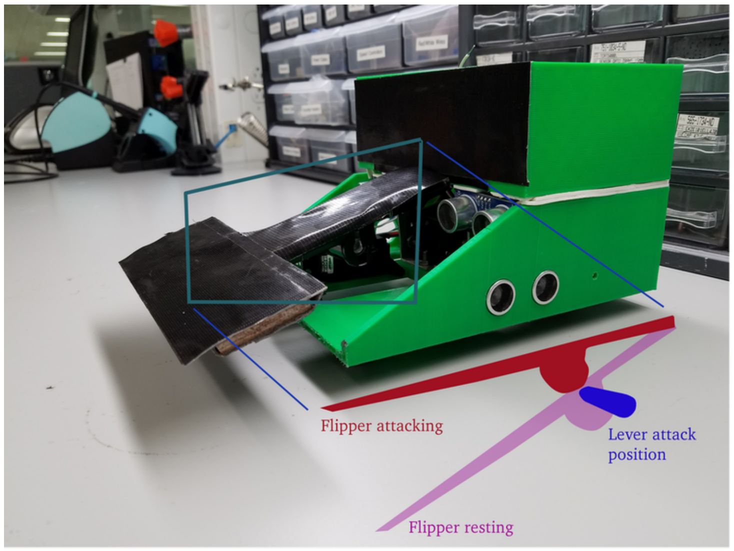


Figure 7: Flipper in action

Wireless LCD Serial Monitor

Display system is our innovation part and this subsystem gives our robot its name “Interpreter”. Arduino board needs to be programmed the way we need the robot to work. But we might not be able to write a code such that the robot does exactly what we need it to do, in the first time. We had to do change parameters, add or delete lines, or change the whole sections of the code multiple times until the robot does exactly what we want it to do.

While we were debugging the code, we faced instances where we wanted see which function the robot is executing when it performs an unnecessary action. First option is to use the serial monitor in Arduino IDE. But, it requires a cable connection between Arduino board and a laptop. This is not convenient since you have to run with the robot carrying your laptop. Instead, we decided to develop a wireless data transmission system using XBees and an LCD display system to act as a serial monitor.

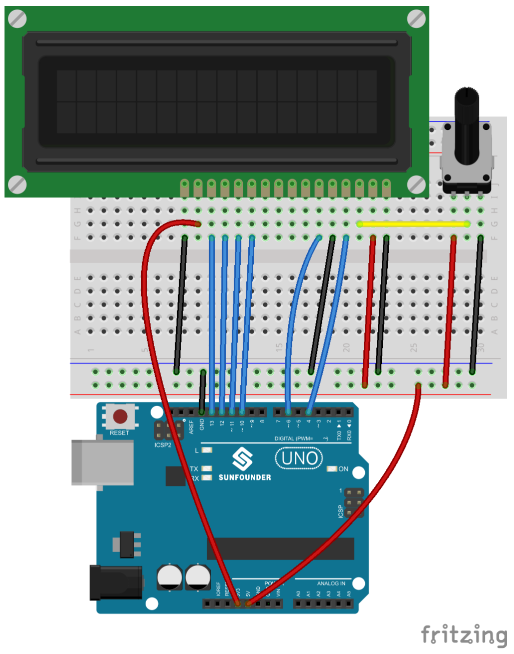


Figure 8: LCD wiring diagram

Our code inside the robot is a collection of function and we call each function in the main loop. Each function is assigned with an integer. We connected one Xbee for a Serial port in the robot Arduino. When a function is called, the integer assigned with the function is sent to XBee. We have another XBee, where the paired other XBee is attached. An LCD display also is attached to this Arduino board. The program inside this Arduino reads from XBee and prints to the LCD screen.

Figure 8 shows the wiring diagram for the LCD. Below snippet shows how our system is run on the robots Arduino board. This keeps writing integer 6 to the XBee.

```
int_lcd = 0  \\ integer for LCD system

void setup() {
    Serial.begin(115200);  \\ initialize XBee serial port
}

void goAttack() {
    Serial.println("Attack");
    attackServo.write(0);
    int_lcd = 6;
}

void loop() {
    goAttack()
}
```

Second Arduino board's code will read this value and print respective string to the LCD display (refer fig. 9). This part is hard to show as a snippet. So please refer Appendix 2.

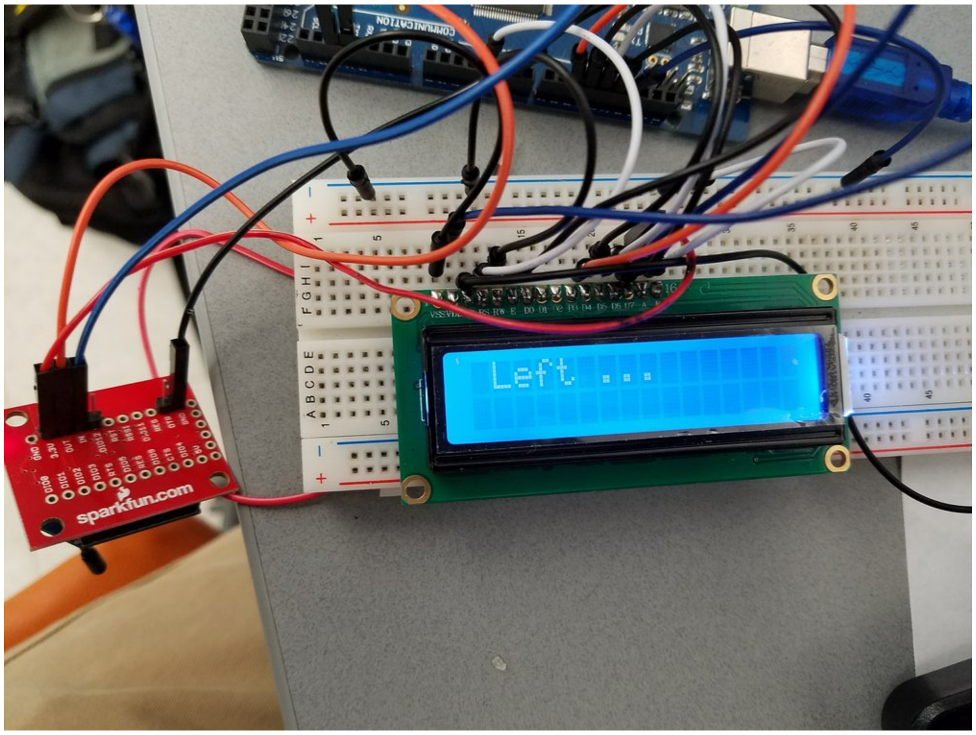
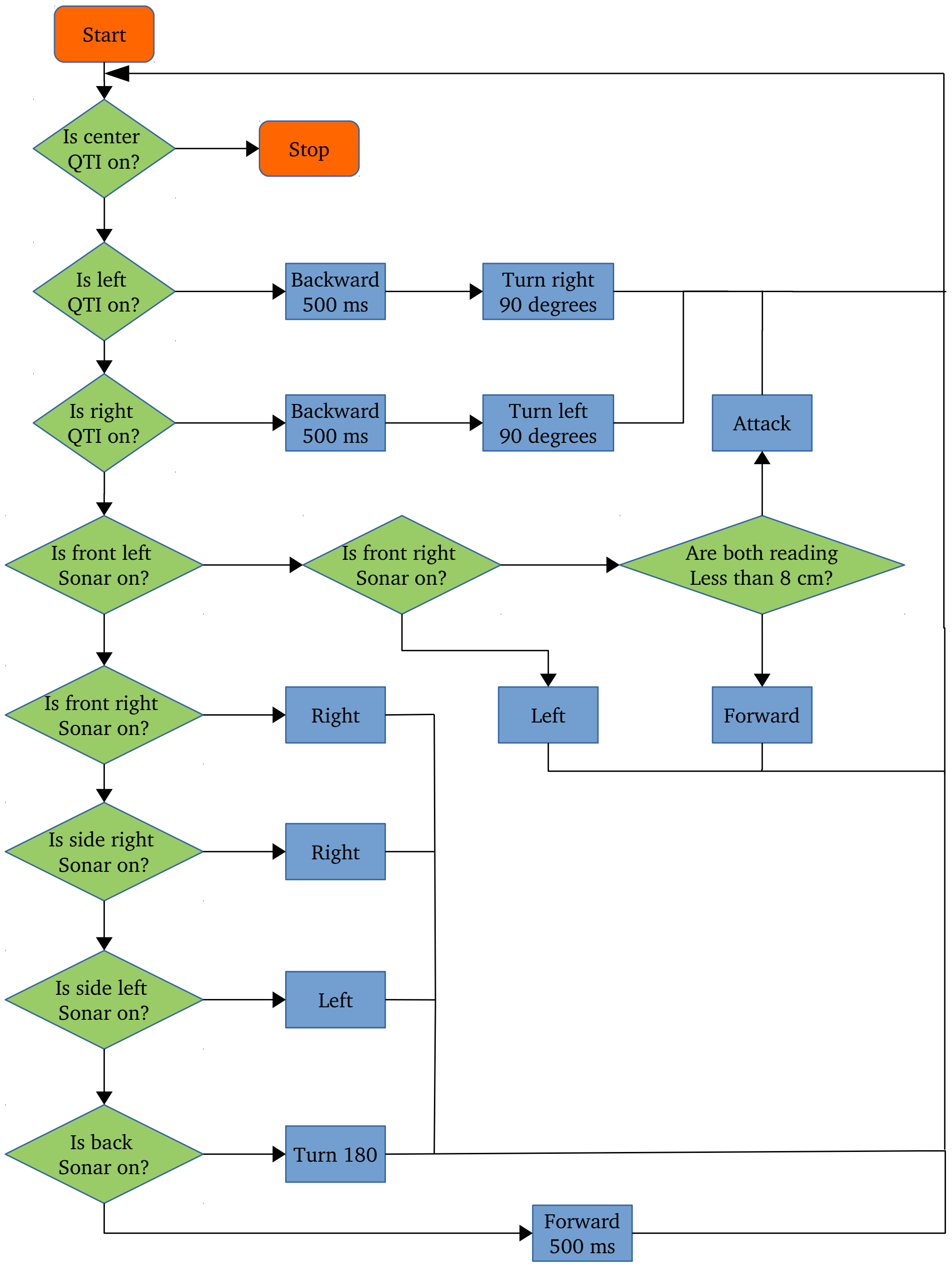


Figure 9: LCD serial monitor

Algorithm


Start
Is center QTI on?
Stop
Is left QTI on?
Backward 500 ms
Turn right 90 degrees
Is right QTI on?
Backward 500 ms
Turn left 90 degrees
Attack
Is front left Sonar on?
Is front right Sonar on?
Are both reading Less than 8 cm?
Is front right Sonar on?
Right
Left
Forward
Is side right Sonar on?
Right
Is side left Sonar on?
Left
Is back Sonar on?
Turn 180
Forward 500 ms

Future Developments

We faced few problems while working on this. One was the wheel we used. We used a wheel with a thin rubber tire. During the competition, the rubber part of the wheel worn out due to friction. This lost the grip on our robot while pushing the other robots. One major future development will be getting thicker tires.

Further, we suggest using two more QTI sensors on the back of the chassis. It will detect if the robot is being pushed backwards and we can program robot to move sideways to avoid being pushed all the way out.

Another major improvement will be adding optical sensors/ IR sensors as a secondary opponent recognition sensor. Some opponents were using sonar absorbing/ non-reflecting materials to cover their robots. Having this secondary system will avoid such situations.

Unfortunately, our attacking mechanism stopped working during the competition due to loose connection. We could not figure it out during the competition. Our last future improvement is to have a proper printed PCB instead of using jumper cables.

List of Components

Component	No. of units	Unit price
Wheels	2	$3.50
Attack servo	1	$8.99
QTI	1	$9.99
Sonar sensors	1 set of 5	$15.99

Appendix 1: Sumo Robot Code

```
int int_lcd = 0;
int int_lcd_prev = 0;
int t_delay = 200;

#define LEDPIN 22  // LED

// Wheel Variables
#include <Servo.h>
Servo leftServo;
Servo rightServo;
Servo attackServo;
#define LEFT_servo 4
#define RIGHT_servo 5
#define ATTACK_servo 7

// Sonar distances
#define HCdistance 50
#define AttackHCdistance 8

// QTI MIDDLE Variable
volatile int MIDDLE_whiteLine = 1;
int MIDDLE_sensorVal;
#define MIDDLE_pinQTIsensor 2

// QTI LEFT Variable
volatile int LEFT_whiteLine;
int LEFT_sensorVal;
#define LEFT_pinQTIsensor 6

// QTI RIGHT Variable
volatile int RIGHT_whiteLine;
int RIGHT_sensorVal;
#define RIGHT_pinQTIsensor 3

// HC-SR04 FRONT LEFT Variable
#define FRONT_LEFT_trigPin 27
#define FRONT_LEFT_echoPin 28
long FRONT_LEFT_duration, FRONT_LEFT_distance;

// HC-SR04 FRONT RIGHT Variable
#define FRONT_RIGHT_trigPin 29
#define FRONT_RIGHT_echoPin 30
long FRONT_RIGHT_duration, FRONT_RIGHT_distance;

// HC-SR04 BACK Variable
#define BACK_trigPin 9
```

```
#define BACK_echoPin 8
long BACK_duration, BACK_distance;

// HC-SR04 LEFT Variable
#define LEFT_trigPin 11
#define LEFT_echoPin 10
long LEFT_duration, LEFT_distance;

// HC-SR04 RIGHT Variable
#define RIGHT_trigPin 13
#define RIGHT_echoPin 12
long RIGHT_duration, RIGHT_distance;


void goForward() {
  // moves robot forward
  leftServo.write(0);
  rightServo.write(190);
  int_lcd = 1;
}

void goBackward() {
  // moves robot backward
  leftServo.write(190);
  rightServo.write(0);
  int_lcd = 2;
}

void goLeft() {
  // moves robot left
  leftServo.write(0);
  rightServo.write(0);
  int_lcd = 3;
}

void goRight() {
  // moves robot right
  leftServo.write(190);
  rightServo.write(190);
  int_lcd = 4;
}

void goStop() {
  // stops robot
  digitalWrite(22, HIGH);
  leftServo.write(90);
  rightServo.write(90);
  attackServo.write(90);
  int_lcd = 5;
}

void goAttack() {
  // starts flipping
  attackServo.write(0);
  int_lcd = 6;
}

void goU() {
  // turn 180
  leftServo.write(0);
  rightServo.write(0);
  int_lcd = 7;
}

void FRONT_LEFT_HC() {
  digitalWrite(FRONT_LEFT_trigPin, LOW);
  digitalWrite(FRONT_LEFT_trigPin, HIGH);
  delay(100);
  digitalWrite(FRONT_LEFT_trigPin, LOW);
  FRONT_LEFT_duration = pulseIn(FRONT_LEFT_echoPin, HIGH);
  FRONT_LEFT_distance = (FRONT_LEFT_duration / 2) * 0.034;
}

void FRONT_RIGHT_HC() {
  digitalWrite(FRONT_RIGHT_trigPin, LOW);
  digitalWrite(FRONT_RIGHT_trigPin, HIGH);
  delay(100);
  digitalWrite(FRONT_RIGHT_trigPin, LOW);
  FRONT_RIGHT_duration = pulseIn(FRONT_RIGHT_echoPin, HIGH);
  FRONT_RIGHT_distance = (FRONT_RIGHT_duration / 2) * 0.034;
}

void BACK_HC() {
  digitalWrite(BACK_trigPin, LOW);
  digitalWrite(BACK_trigPin, HIGH);
  digitalWrite(BACK_trigPin, LOW);
  BACK_duration = pulseIn(BACK_echoPin, HIGH);
  BACK_distance = (BACK_duration / 2) * 0.034;
}

void LEFT_HC() {
  digitalWrite(LEFT_trigPin, LOW);
  digitalWrite(LEFT_trigPin, HIGH);
  digitalWrite(LEFT_trigPin, LOW);
  LEFT_duration = pulseIn(LEFT_echoPin, HIGH);
  LEFT_distance = (LEFT_duration / 2) * 0.034;
```

```
}

void RIGHT_HC() {
  digitalWrite(RIGHT_trigPin, LOW);
  digitalWrite(RIGHT_trigPin, HIGH);
  digitalWrite(RIGHT_trigPin, LOW);
  RIGHT_duration = pulseIn(RIGHT_echoPin, HIGH);
  RIGHT_distance = (RIGHT_duration / 2) * 0.034;
}

void setup() {

  Serial.begin(115200);  // XBee serial port
  pinMode(LEDPIN, OUTPUT);  // LED is off at the beginning

  // attach sevos
  leftServo.attach(LEFT_servo);
  rightServo.attach(RIGHT_servo);
  attackServo.attach(ATTACK_servo);

  // initialize QTIs
  pinMode(LEFT_pinQTIsensor, INPUT);
  pinMode(MIDDLE_pinQTIsensor, INPUT);
  pinMode(RIGHT_pinQTIsensor, INPUT);

  digitalWrite(LEFT_pinQTIsensor, HIGH);
  digitalWrite(MIDDLE_pinQTIsensor, HIGH);
  digitalWrite(RIGHT_pinQTIsensor, HIGH);

  // attach interrupt
  attachInterrupt(digitalPinToInterrupt(MIDDLE_pinQTIsensor), MIDDLE_whiteLineISR, LOW);
  interrupts(); // enable interrupts

  // configure sonar sensor pins
  pinMode(FRONT_LEFT_trigPin, OUTPUT);
  pinMode(FRONT_LEFT_echoPin, INPUT);
  pinMode(FRONT_RIGHT_trigPin, OUTPUT);
  pinMode(FRONT_RIGHT_echoPin, INPUT);
  pinMode(BACK_trigPin, OUTPUT);
  pinMode(BACK_echoPin, INPUT);
  pinMode(LEFT_trigPin, OUTPUT);
  pinMode(LEFT_echoPin, INPUT);
  pinMode(RIGHT_trigPin, OUTPUT);
  pinMode(RIGHT_echoPin, INPUT);

  // do not move at the beginning
  goStop();
  delay(500);
```

```
}

// interrupt for the middle QTI
void MIDDLE_whiteLineISR() {
  goStop();
  MIDDLE_whiteLine = 0;
  int_lcd = 10;
}

void loop() {
  // read from sonar sensors
  FRONT_LEFT_HC();
  FRONT_RIGHT_HC();
  LEFT_HC();
  RIGHT_HC();
  BACK_HC();

  // read QTI sensors
  LEFT_sensorVal = digitalRead(LEFT_pinQTIsensor);
  RIGHT_sensorVal = digitalRead(RIGHT_pinQTIsensor);

  // if front right QTI is on
  if (  RIGHT_sensorVal == 0 ) {
    goBackward();
    delay(500);
    goLeft();
    delay(t_delay);
  }

  // if front left QTI is on
  if (  LEFT_sensorVal == 0 ) {
    goBackward();
    delay(500);
    goRight();
    delay(t_delay);
  }

  // do if middle QTI is not detecting line
  if (MIDDLE_whiteLine == 1) {

    // if opponent is closer, attack
    if (FRONT_LEFT_distance < AttackHCdistance && FRONT_RIGHT_distance < AttackHCdistance ) {
      goAttack();
      delay(t_delay);
    }

    // if opponent is forward, go push
    if (FRONT_LEFT_distance < HCdistance && FRONT_RIGHT_distance < HCdistance) {
      goForward();
      delay(t_delay);
    }

    // if opponent is forward left, go left
    if (FRONT_LEFT_distance < HCdistance && FRONT_RIGHT_distance > HCdistance) {
      goForward();
      delay(t_delay);
    }

    // if opponent is forward right, go right
    if (FRONT_LEFT_distance > HCdistance && FRONT_RIGHT_distance < HCdistance) {
      goForward();
      delay(t_delay);
    }

    // if opponent is on left, go left
    if (LEFT_distance < HCdistance ) {
      goLeft();
      delay(t_delay);
    }

    // if opponent is on right, go right
    if (RIGHT_distance < HCdistance ) {
      goRight();
      delay(t_delay);
    }

    // if opponent is on back, rotate 180
    if (BACK_distance < HCdistance) {
      goU();
      delay(2 * t_delay);
    }
  }
  else {
    goStop();
    digitalWrite(LEDPIN, HIGH);
    while (1);
  }

  // update XBee only if LCD int is changed
  if (int_lcd_prev != int_lcd) {
    Serial.println(int_lcd_prev);
  }
  int_lcd_prev = int_lcd;
}
```

Appendix 2: Wireless LCD Serial Monitor Code

```
#include <LiquidCrystal.h>                  // include the library code

char array1[] = " Bot Info               "; // welcome string

int delay_t = 200; //the value of delay delay_tef
int positionCounter1 = 0; // for left moving animation

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(4, 6, 10, 11, 12, 13);

void setup() {
  Serial1.begin(115200);  // Xbee
  lcd.begin(16, 2);       // set up the LCD's number of columns and rows:
  lcd.clear();            // clear the LCD
  lcd.setCursor(15, 0);   // set the cursor to column 15, line 0

  // flush the buffer
  while (Serial1.available() > 0)
    Serial1.read();

  // print welcome string
  positionCounter1 = 0;
  while (positionCounter1 < 15) {
    lcd.scrollDisplayLeft();
    lcd.print(array1[positionCounter1]);
    delay(delay_t);
    positionCounter1++;
  }
}



void loop() {
  String in;
  if (Serial1.available() > 0) {
    // read the string from XBee
    in = Serial1.readStringUntil('\n');
  }

  // convert the String to Int
  int in_int = in.toInt();

  lcd.setCursor(0, 1);
  lcd.clear();

  // print the respective string to LCD
  switch (in_int) {
    case 1:
      Serial.println(in_int);
      lcd.print(" Robot in front      ");
      break;
    case 2:
      lcd.print(" Go back ..      ");
      break;
    case 3:
      lcd.print(" Left ...      ");
      break;
    case 4:
      lcd.print(" Right ...      ");
      break;
    case 5:
      lcd.print(" Stop !      ");
      break;
    case 6:
      lcd.print("Attack      ");
      break;
    case 7:
      lcd.print(" Rotate 180   ");
      break;
    case 10:
      lcd.print("  Middle QTI int.  ");
      break;
    default:
      break;
  }

  //  lcd.print(in_int);
  delay(1000);

  // flush the buffer
  while (Serial1.available() > 0)
    Serial1.read();
}
```