Cosc 4P78 Project: "Martini Bot"

Michael Alsbergas, Peter Wilson

April 19, 2016

1 Instructions

To activate the martini bot, simply turn both power switches to the on position. Toggling the labeled ON/OFF switch in the front of the Robot will give it power to the Motors, allowing the robot to move. Additionally, there is a battery pack on the back of the robot that required to be turned on as well. This battery pack will give power to the Romeo and well as all the other components connected to that microcontroller.

Once the robot has been powered on, it will remain in an "idle" state until it receives a signal from the target remote. The remote will send the signal when it is initially turned on by plugging in the battery. The signal can also be sent by pressing one of the reset buttons on the remote control. When not communicating with the robot, the remote will poll the XBee waiting for a command from the robot so it too will be idle until the process has begun.

See Appendix A (8) for a diagram on how to set up the robot and Appendix B (9) for the robot code

When the robot has receives the signal to begin from the remote control, it will begin its search for the target remote. After the robot has found and repositioned itself near its target, by being within out predefined threshold of 10 cm, it will return to its "idle" state and can be reactivated by pressing the reset button on the remote.

See Appendix C (10) for a diagram on how to set up the remote and Appendix D (11) for the remote code

2 Problem:

The problem task that this robot was designed to solve is when a person want a drink or food but does not want or is not able to retrieve it themselves. Potentially, a solution to this problem could be used to deliver all manners of payloads.

This problem, from the eyes (rangefinders) of a robotic solution, is locating and moving towards a specific and unknown point without having any prior knowledge of the surrounding area aside from some core assumptions.

3 Solution:

The hardware on the robot itself includes an ultrasonic rangefinder, an XBee radio transceiver, 2 DC motors and wheels equipped with encoders, a Romeo arduino chip to control it all, and of course the appropriate amount of batteries to power it all. It utilizes the built in Motor controls of the Romeo to power the DC motors and has an AT tiny with the provided code to handle the distances from the encoders. These are all mounted on the robotics kit shell that was provided in the class. Since all the components wouldn't fit on the Romeo itself, we needed to add a couple other boards to host the extra items.

Similarly, the target remote is made of the same kind of ultrasonic rangefinder, the same kind of XBee radio transceiver, an AT mega arduino chip to control it, and a 9V battery as the power supply all contained within a clear plastic case to keep it together. The transmitter for the XBee radio as well as the Ultrasonic sensor need to be outside the box so special affordances was made in the arranging of the box so they would protrude out. The provided container had an opening at the top which we felt would be perfect for the ultrasonic sensor. That way, the controller could be handled like a remote, where the user could point the remote in the direction of the robot in an easy, comfortable way. Additionally, there was room at the bottom of the case with a small square opening. We used this to stick the antenna of the XBee radio so it was not contained in the case and could easily send information back and forth to the robot.

Our initial solution using this hardware was to use the 2 ultrasonic's on the devices to communicate with each other so that the robot could determine the distance to the target. Combining that and the distance traveled using the encoders, we could use trigonometry to determine the direction of the target. Once both the distance and direction were found, it was a simple matter of going to the target. Yet, since we're using ultrasonic's, we had to build our robot with the assumption that there would be no obstacles between it and a straight line to the target since the ultrasonic's could not read each other if there was one.

The problem was then coordinating the ultrasonic's so that they triggered at the same time, at least roughly. That's where the XBees are used. The transceivers would be used to send a signal to notify the other of when to send/receive an ultrasonic pulse. The robot would send an activation signal to the remote and would then listen for a pulse from the remote. The XBees are also used to have the remote send the activation signal to the robot while it's in its idle state.

However, we quickly discovered that even when using the XBees for timing, the ultrasonic's would still not determine an accurate distance between each other. Not only were the received readings inaccurate, but they were also erratic and inconsistent. Yet, we noticed that all the readings obtained using the 2 together were always less than the actual distance. With this, we rethought our solution.

The robot would spin so that it and its ultrasonic were facing a direction. Then, it would use the ultrasonic to take a number of readings in front of it

to determine a baseline. Then, it sends a radio signal to the remote telling it to emit a series of non-stop pulses. The robot would then take another set of readings in the same direction and then send a stop radio signal to the remote. If the second set of readings had the same average of the first, the robot would rotate and try the next direction. But, if the robot was facing the target at the time of the second set of readings, the values and their average would be considerably smaller due to the interference given off by the target, which means that it's facing the target.

Since the robot now has a general direction in which the target might be in and a rough approximation of the distance, all it needs to do is move forward. Because of the assumption of the unobstructed straight line, the robot can move forward with confidence. The robot will move half of the approximated distance, and if that movement/distance is small enough, the robot will know that it reached its target and will become idle to wait for its next target. Since the robot only receives a rough approximation of the direction, it may lose track of the target. When this happens, it simply begins its process of turning around until it locates the target again.

Due to the Romeo's motor control ability to have a DC motor spin in reverse, it allows the robot to actually rotate on the spot rather than turn on a pivot. The encoders no longer become essential in our final solution, however they are still helpful in moving forward a calculated distance as well as controlling the turning angle of the robot. Neither of these parts are truly essential, but their use is recommended to solve other potential problems around the robot's movement (such as hitting a nearby wall while pivoting).

4 Roadblocks During Creation

An early design of the robot had the target remote constantly sending ultrasonic signals to the robot and the robot only listening for those signals. Once the robot got any sort of reading, the assumption would be made that the sensor is in the cone radius of the robot (i.e. he is facing the direction of the target) The robot will move in that direction, still checking the ultrasonic readings until the distance was small or the signal was lost, in which case it will rotate until the signal is found. In practice, this design didn't work for our needs. The signals from the target weren't registering on the ultrasonic sensor of the robot and all the signals timed out, even when we pointed the sensors right at each other.

We believed the issue to be a timing one. If the one sensor wasn't sending at the same time as the other started reading, the numbers would be completely off. We then began to utilize the XBee radios to attempt to sync up the timing of the pulses. How this would work is the robot would send a message to the target to request a reading from the target and then immediately start looking for a response. When the target got the command, it would send one pulse and stop. The robot would hypothetically get the reading and know exactly where the target is or if it would need to rotate to get another reading. Again, once we implemented this, we realize we were in the same boat as before. We were

getting a signal timeout even if they started at the same time. Additionally, we increased the timeout to a minute and still we were getting no readings.

After some trouble shooting, we monitored the reading of the ultrasonic sensors and we realize that if we had both sensors shooting at each other at the same time, the distance value was roughly half the distance as if only one was firing. We realized that we could use this information to our advantage. The design changed from only the trigger sending the ultrasonic signal and the robot reading it to both of them sending the signal and the robot noting if there was a difference. This, in turn, changed the problem quite a bit. Instead of just getting the distance, the problem changed to getting the difference in the distance based on them both firing vs only one firing. How we tackled this was we had the robot stop and get a distance reading. It would then signal the trigger to start sending signals. The robot would then get another reading. If the 2 readings were appreciably different, we could make the assumption that we are heading towards the target and that we should move forward. In our testing, we found the best results if we defined appreciably different to be 70% or less or the original reading. i.e. if the reading with the sensor on is less than or equal to 70% of the standard reading than it is safe to say that we are heading in the correct direction.

Although this was leading to much better results than before, we noticed we were getting a lot of false positives and false negatives. When testing in the lab, we noticed that the robot would often head in the wrong direction, often towards the metal stools or the other students in the lab. When we hooked the robot up to the serial monitor, we noticed the reading were not reliably consistent. With the robot not moving, pointing in the same direction, we saw the reading fluctuating from 40-60 cm. However, we noted that on average, the distance was closer to 50cm. To take advantage of this observation, instead of taking 1 reading for each (before and after), we decided to take 20 readings and get the average of those to compare. Now only if the averages were appreciably different would we move forward.

In testing, this change did a great job of cutting down on false positives and led to the robot more accurately homing toward the target remote control. Testing the robot on the hallway led to great results with false positives being fairly small, however the robot still tended to have a hard time actually making it to the target. Especially in the computer lab where there was metal chairs and other students walking around.

Again, we hooked the robot up to the sensor and this time pointed it around the room. We noticed the we were getting some odd readings. We'd consistently get readings between 51-53cm and then out of nowhere we would get a reading over 3000 cm. If that were to happen when the robot is operating, it would throw off all the averages and lead to a false positive for sure. Our solution to this problem was to have the robot throw away up to 5 unusually high readings when making the average. Since it happened very infrequently, we made the assumption that it is unlikely to happen more than 5 times in 20 readings if it indeed was a false reading. This allowed us to completely toss out misleading data and completely remove false positives. In practice, this led to almost no

false directions for the robot. The only time there were false positives was when the robot is facing a wall away from the target, in which we assume the problem is that the ultrasonic pulses are bouncing off the wall.

5 Remaining Problems

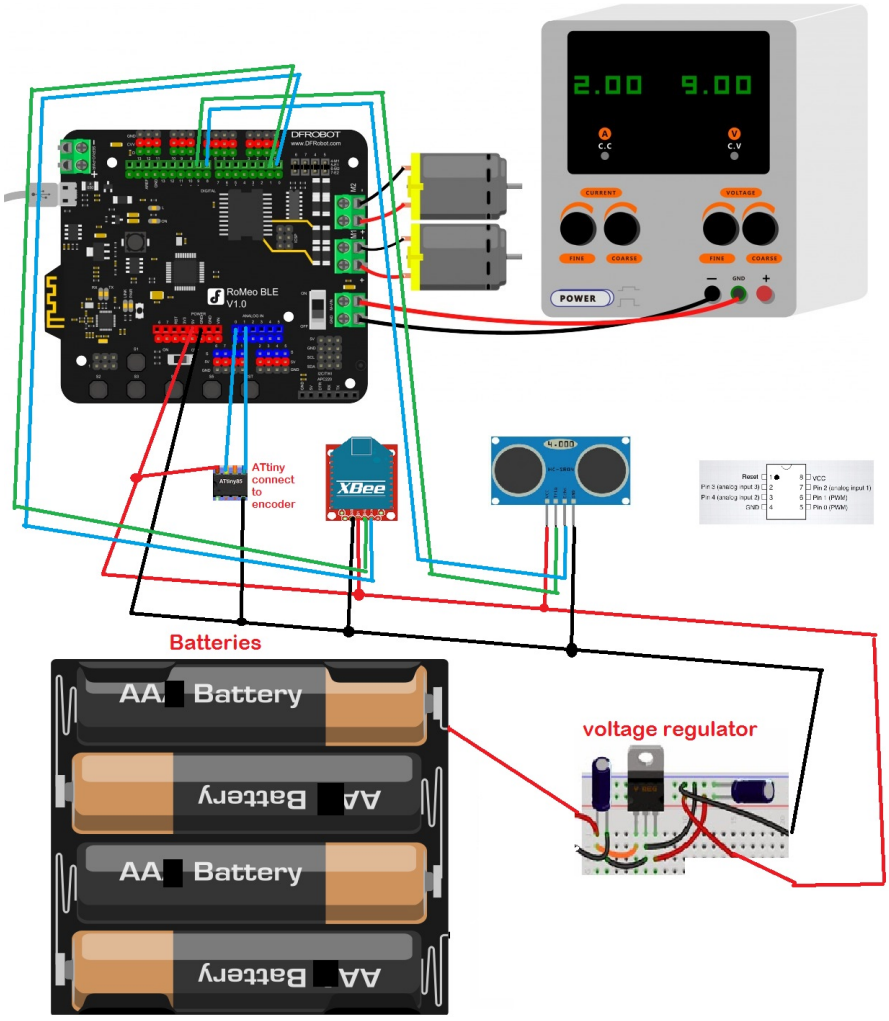
The only remaining problems are the possibility of false positives and a maximum range. When used in an environment that has other sources of ultrasonic interference, the robot may get confused and begin to move in the wrong direction. This has been noticed when the robot is facing baggy clothing which tend to absorb ultrasonic waves. Also in the robotics lab where there are a lot of metal stool legs, it appears that some of the ultrasonic interference from the target is reflected off the legs and is registered by the robot as the interference it is looking for. We have also been unable to coax it to go through a metal door frame. This may be because either the metal deflects the pulses too much or the robot just picks up interference pulses bouncing off walls and objects behind it. To help reduce the chances of getting obscurely high values during the robot's test readings, we programmed it to ignore all readings above 3000cm (100ft). So, if the target is beyond this range, the robot will ignore it. This is also assuming that the max range of the ultrasonic's and the XBees are greater than this distance.

6 Resources

The XBees print to the serial and do not need/have any dedicated libraries to use. The rangefinders need the ultrasonic library to use normally. We also modified the ultrasonic library for the sake of the target to emit a steady stream of pulses without needing to read in any echo data

7 Appendix

Appendix A: Martini-bot Diagram



Appendix B: Martini-bot Code

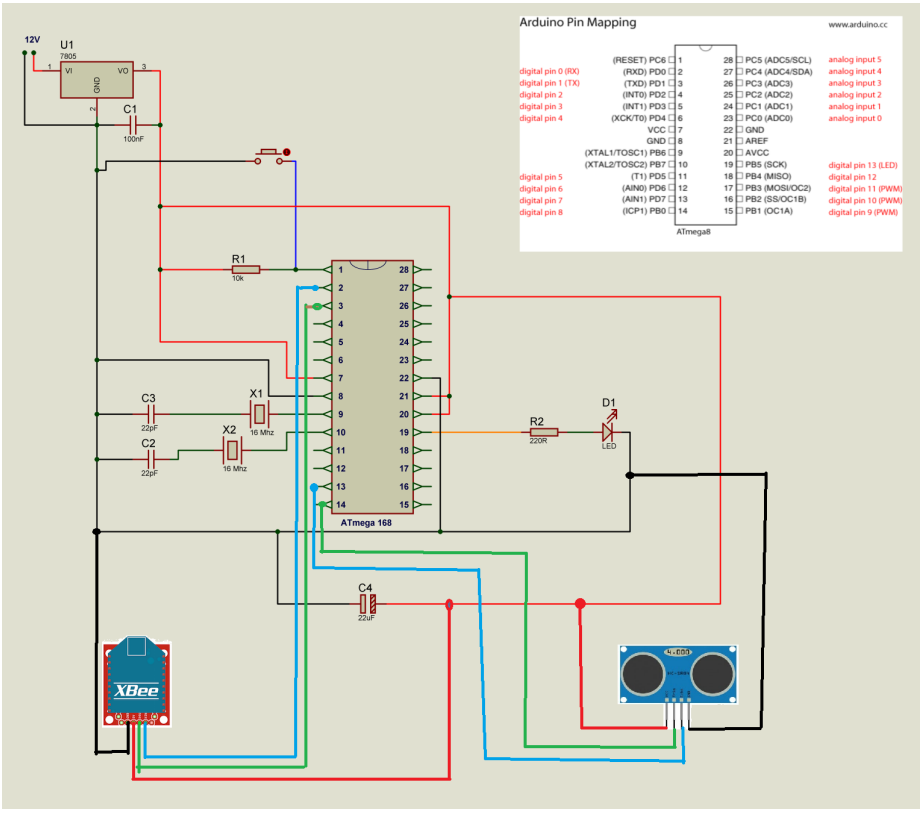
```
#include "Ultrasonic2.h"
#include <Wire.h>

long distance, check;
Ultrasonic ultrasonic(9,8);
int leftDistance;
int rightDistance;
int x;

//Standard PWM DC control
int E1 = 5;      //M1 Speed Control
int E2 = 6;      //M2 Speed Control
int M1 = 4;     //M1 Direction Control
int M2 = 7;     //M1 Direction Control

void setup() {
  // setup the xbee connection
  Wire.begin();
  Serial.begin(115200);
  int i;
  for(i=4;i<=7;i++)
    pinMode(i, OUTPUT);

  leftDistance = 0;
  rightDistance = 0;

  idle();
}

void loop() {
  //Test Direction
  distance = 0;
  check = 0;

  x = 20;
  delay(25);
  for (int i = 0; i < x; i++){
    int temp = 0;
    temp = ultrasonic.Ranging(CM);
    if (temp < 3000){
      distance = distance + temp;
    }
    else {
```

```
    x++;
    if(x==25)
    {
      break;
    }
  }
  delay(10);
}
distance = distance / 20;

Serial.write('B');
delay(5);
x = 20;

for (int i = 0; i < x; i++){
  int temp = 0;
  temp = ultrasonic.Ranging(CM);
  if (temp < 3000){
    check = check + temp;
  }
  else {
    x++;
    if(x==25)
    {
      break;
    }
  }
  delay(10);
}
check = check / 20;

Serial.write('S');
delay(5);

if (distance * 0.7 > check){
  //move Forward
  leftDistance = 0;
  rightDistance = 0;
  advance(100,100);
  while(leftDistance < check/2)
  {
    delay(50);
    UpdateDistance();
  }
  stop();
  if (check / 2 < 10){ //Terminate
```

```
      idle();
    }
  }
  else {
    leftDistance = 0;
    turn_a_bit();
  }
  delay(50);
}



void turn_a_bit(){
  turn_L(100,100);
  while(leftDistance < 3)
  {
      delay(20);
      UpdateDistance();
  }
  stop();
  leftDistance = 0;
}

void UpdateDistance()
{
  Wire.requestFrom(0x7, 1); // request 1 byte from slave device #7
   if (Wire.available()) { //make sure you got something
     byte c = Wire.read();
      leftDistance += c&0x0F;
      rightDistance += (c>>4)&0x0F;
     }
}

void stop(void)                          //Stop
{
  digitalWrite(E1,LOW);
  digitalWrite(E2,LOW);
}
void advance(char a,char b)           //Move forward
{
  analogWrite (E1,a);       //PWM Speed Control
  digitalWrite(M1,HIGH);
  analogWrite (E2,b);
  digitalWrite(M2,HIGH);
}
void back_off (char a,char b)           //Move backward
```

```
{
  analogWrite (E1,a);
  digitalWrite(M1,LOW);
  analogWrite (E2,b);
  digitalWrite(M2,LOW);
}
void turn_L (char a,char b)                    //Turn Left
{
  analogWrite (E1,a);
  digitalWrite(M1,LOW);
  analogWrite (E2,b);
  digitalWrite(M2,HIGH);
}
void turn_R (char a,char b)                    //Turn Right
{
  analogWrite (E1,a);
  digitalWrite(M1,HIGH);
  analogWrite (E2,b);
  digitalWrite(M2,LOW);
}

void idle(){
  char c;
  while (c != 'A'){
    if (Serial.available() > 0){
      c = Serial.read();
    }
    delay(50);
  }
}
```

Appendix C: Remote Diagram



Appendix B: Remote Code

```
#include "Ultrasonic2.h"

Ultrasonic ultrasonic(8,7);

void setup() {
 // setup the xbee connection
  Serial.begin(115200);
  Serial.write('A');
}

void loop() {

  //check for radio signal
  if(Serial.available() > 0)
  {
      //read command
      char input = Serial.read();



      //if send
      if(input == 'B')
      {
        //send until told otherwise
        while(Serial.available() == 0)
        {
          ultrasonic.Send();
        }
      }
  }

}
```