Full Throttle STEM™ Rover:
Control a Rover from an Android Device

Challenge Problem and Resources

Developed by:

The teachers, students, and mentors in the
Gaming Research Integration for Learning Laboratory™ (GRILL™)
Summer 2014

TABLE OF CONTENTS

1. CHALLENGE PROBLEM: CONTROL A ROVER FROM AN ANDROID DEVICE

Control a rover from an Android device by connecting the rover and Android to programmable boards with radio transmitter/receiver shields and writing the app for the Android device and programming the boards.

1.1. THE TOOLS

The hardware required includes a rover, an Android device, and two Arduino boards with radio transmitter/receiver shields. Additional tools needed include Arduino Integrated Development Environment (IDE) and Android development software such as Eclipse with the ADT plugin, or Android Studio.

1.2. THE CHALLENGE

Program the app to send commands to the rover and have the rover respond to those commands.

2. TUTORIALS

Wright Scholars, in collaboration with educators and the GRILL™ team, created the tutorials described below as possible solutions to solve the challenge problem. At the time of creation these were working tutorials; however, with software updates and changes in technology, additional steps may be required. Teachers are encouraged to communicate any issues, problems, or suggested changes to these tutorials to ensure the dissemination of helpful materials to support challenge problem implementation.

2.1. ANDROID COMMUNICATING WITH THE ARDUINO

The Arduino board will need a program that will allow it to read and write data to and from the Android Tablet. To communicate properly with each other, both the tablet and the board will need to take turns sending data.

This tutorial includes the following topics:

- Creating the Arduino sketch (Section 2.1.1)
- Creating the Android code (Section 2.1.2)

2.1.1. CREATING THE ARDUINO SKETCH

This section outlines steps to create the Arduino sketch. The steps include example code for reference.

1. Start with a blank Sketch that only has the setup and loop methods.
2. Open a communication port, in the setup() method, on the Arduino end and begin at a baud rate of 9600.
3. Next declare a static char array called input_line with a size of 50 in the loop() method. Then declare a static unsigned int called input_pos and set it equal to 0. The input_line char array will store received bytes and the input_pos will decide where to put the received bytes.
4. After declaring the variables make an if-statement that checks the availability of the serial port. Essentially this checks if there is any incoming data to be received.
5. Create a char inside the if-statement called inByte that will store the byte read from the serial.
6. Now make a switch statement that will take in inByte and have two cases in addition to the default case. The first case should be that inByte equals '\n'. If the case is '\n' then change the size of input_line to input_pos and set the value of input_line equal to 0 (*input_line [input_pos] = 0;*). This will essentially terminate the byte. Next use a method that has not

been written yet called process_data that will take in our char array and allow us to use the collected data. Lastly reset the buffer input_pos by setting the value to 0. Make sure to break to end case one. Case two should check to see if inByte is equal to '\r'. If it is, then simply break. The default case should add the inByte to input_line at the position of input_pos as long as the array is not full. To end the switch, break the default.

```
void setup()
  {
    //begin the serial
    Serial.begin(9600);
  }
void loop()
  {
    //char array with a size of 50 that will store the bytes
    static char input_line [50];
    //tracks the position data will be stored at
    static unsigned int input_pos = 0;

    //if there is data being recieved
    if (Serial.available() > 0)
      {
        //read the serial and store the data to the char
        char inByte = Serial.read();
        switch(inByte)
          {
            //if inByte == '\n'
```

7. Next make the process_data method that takes in a char array. This method is more for organization of the code. The data could be processed within the switch statement, but wouldn't be good coding.
8. The Arduino will always be listening and should only send data if the Android said that it is okay for it to. The Arduino and Android will be sending the letter "T" back and forth to signify whose turn it is to send data. In the process_data method, create an if-statement to

check if the data is equal to a “T”. If the data is equal to “T” then run another custom method called sendData().

9. Before writing code for the sendData() method, make a global string called dataToSend and set it equal to “”. This will make the variable null unless we set it to some data that we want to send from the Arduino.
10. Now, inside the sendData() method, create an if-statement that will check to see if dataToSend equals “”. If dataToSend does not equal “”, then println dataToSend to the Serial.
11. Now that the available data has been sent, println the string “T” to the serial to inform the Android that it is now its turn to send data.
12. Lastly, set the dataToSend string to “” once more in order to not send that same data repeatedly.

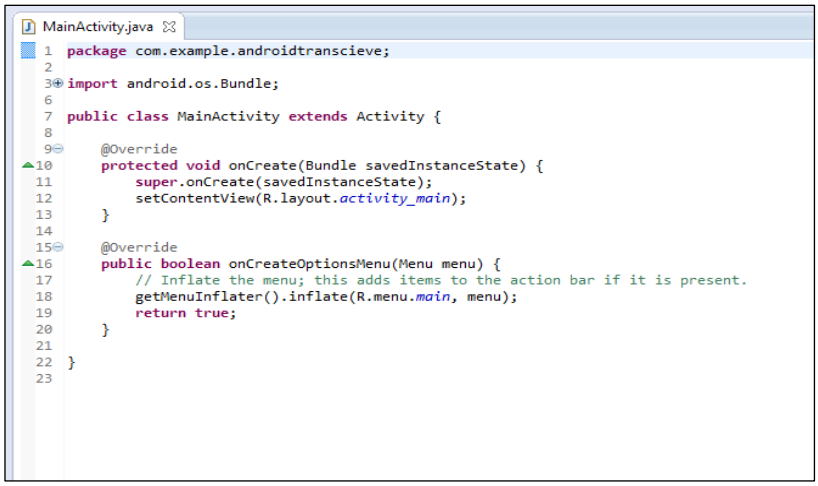
```
//terminator reached! process input_line here...

              process_data(input_line);

              //reset buffer for next time

              input_pos = 0;

              break;

            case '\r':

              break;

            default:

              // keep adding if not full...

              // allow for terminating null byte

              if (input_pos < (MAX_INPUT - 1))

                input_line [input_pos++] = inByte;

              break;

           }

       }

       //do other stuff here, like get sensor data

       //and set dataToSend equal to the readings

  }

void process_data(char * data)

  {

    //if the data equals T then run the sendData() method
```

```
    if(String(data).equals("T"))
      {
        sendData();
      }
  }
void sendData()
  {
    // if there is data in the dataToSend string,
    // send it through the serial
    if (!dataToSend.equals(""))
      {
        Serial.println(dataToSend);
      }
    //alway send T back to give the Android its turn
    Serial.println("T");
    //reset the dataToSend string
    dataToSend = "";
  }
case '\n':
 //terminating null byte
input_line [input_pos] = 0;
```

2.1.2. CREATING THE ANDROID CODE

The Android tablet will need a program that will allow it to read and write data to and from the Arduino board. To communicate properly with each other, both the tablet and the board will need to take turns sending data. The tablet will use a third party app installed on the tablet to send and receive data.

1. Create a new Android Application Project. After naming the project, leave the default settings.
2. Open the main java file that Eclipse created. The default file name is MainActivity.java and should be located in the src folder under com.example. The code is in this file.



3. Declare the following variables in the public class MainActivity:

```
private final static String DATA_RECEIVED_INTENT = "primavera.arduino.intent.action.DATA_RECEIVED";

private final static String SEND_DATA_INTENT = "primavera.arduino.intent.action.SEND_DATA";

private final static String DATA_EXTRA = "primavera.arduino.intent.extra.DATA";
```

4. In the onCreate method, create a new Intent Filter called filter. Then add an action to that filter and pass the DATA_RECEIVED_INTENT string.
5. Enter the code below to register a new BroadcastReceiver.

```
IntentFilter filter = new IntentFilter();

        filter.addAction(DATA_RECEIVED_INTENT);

        registerReceiver(new BroadcastReceiver() {

            @Override

            public void onReceive(Context context, Intent intent) {

                final String action = intent.getAction();

                if (DATA_RECEIVED_INTENT.equals(action)) {

                    final byte[] data = intent.getByteArrayExtra(DATA_EXTRA);

                    Toast.makeText(context, new String(data), Toast.LENGTH_SHORT).show();

                }

            }

        }, filter);
```

New data will display in a Toast, which is essentially a pop-up box. The code takes the data and converts it to a string. It is then passed into a method called checkData() that you will need to create.

6. Create the custom method called checkData, which will pass in a string. Inside the checkData() method, create an if-statement that checks if the string passed in equals T. If so, set the boolean canSend to true.
7. Go back to the onCreate method in the onRecieve method after the code checkData(dataString); and place an if-statement checking canSend. If canSend is true, then run a sendData() method.
8. Create a sendData() method that takes in no arguments, and inside create an if-statement that checks if dataToSend does not equal “”. If dataToSend equals something other than “”, then broadcast an intent with the data as an extra on the intent. Finally send a T to hand the turn over.

```
public void sendData(View v) {

        if (!dataToSend.equals(""))

        {

           dataToSend += "\n";

           byte[] DATA = dataToSend.getBytes();

           Intent intent = new Intent(SEND_DATA_INTENT);

           intent.putExtra(DATA_EXTRA, DATA);

           sendBroadcast(intent);

           dataToSend = ""

        }

        Intent intent = new Intent(SEND_DATA_INTENT);

        intent.putExtra(DATA_EXTRA, terminator);

        sendBroadcast(intent);

    }
```

9. Now drag a button into the layout and edit the Button child XML.

```
<Button

        android:id="@+id/button1"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:layout_alignParentTop="true"

        android:layout_centerHorizontal="true"

        android:layout_marginTop="190dp"
```

```
        android:text="Say Hi"

        android:onClick="sayHi" />
```

10. Create a public method called sayHi that takes in a View argument that runs when the button is pressed. Because the onClick of the button is set to the same name as this public method, the method will run when the button is clicked.
11. Inside the sayHi method, set dataToSend equal to “Hi\n”. Anytime you click the button, the program will wait until it is its turn to send data, and will then send “Hi” when it can.
12. To initiate the communication, the Arduino must be sent a “T” first. Create another button and set the onClick to initiate and name the button Start Communication or something similar.
13. Create the method for initiate, and make sure that it is public and takes in View as an argument.

```
public void initiate (View v) {

        if (i == 0) {

            Intent intent = new Intent(SEND_DATA_INTENT);

            intent.putExtra(DATA_EXTRA, terminator);

            sendBroadcast(intent);

            i++;

        }

    }
```

Note: Make sure to declare the int i outside of the method so it is accessible to all methods and set it to 0. This will ensure the user can only send a “T” the first time and the rest is automated to prevent any crashes.

2.2. CONFIGURING XBEES AND WIRING AN RC CAR TO AN ARDUINO

This tutorial explains how Arduino radio transmitters and receivers can control rover movement. Suggested tools for this challenge problem include two XBees (S2 Recommended – must be configured to communicate with each other), two XBee dongles, one Arduino Board, and sufficient multicolored wires.

This tutorial includes the following topics:

- XBEE Configuration (Section 2.2.1)
- Arduino programming (Section 2.2.2)

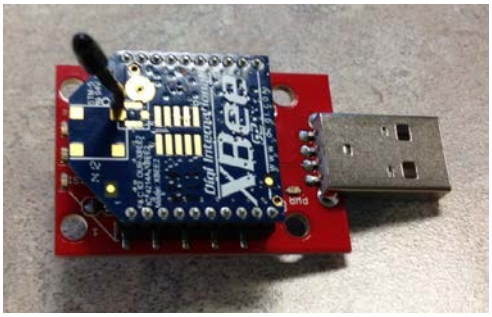
- Wiring the RC car, Arduino, and an XBee (Section 2.2.3)

2.2.1. XBEE CONFIGURATION

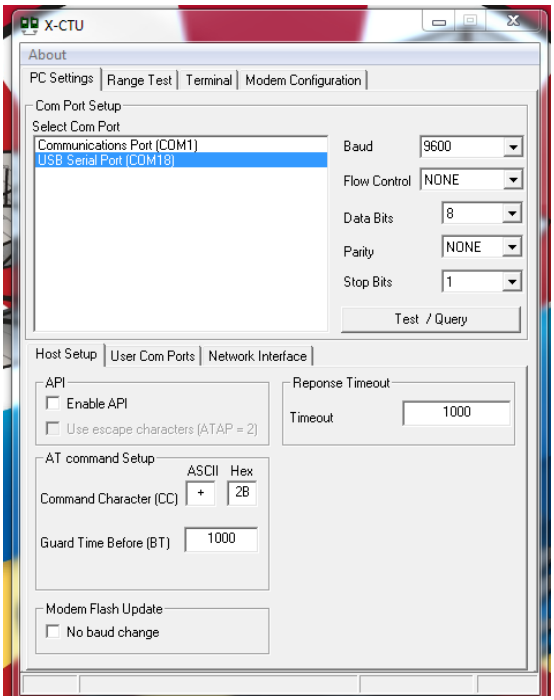
This section presents the steps required to configure the XBee.

To configure the XBee:

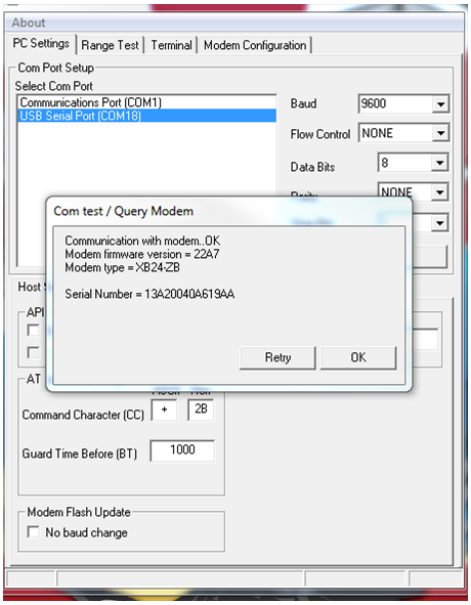
1. Download and install the latest version of X-CTU Software from the following link: http://www.digi.com/support/productdetail?pid=3352&osvid=57&type=utilities.
2. Choose an installer under the "General Diagnostics, Utilities, and MIBs" tab that will work best for your computer.
3. Insert one of the XBees into its dongle and connect it to the USB Port on your computer. This XBee will be the Router.



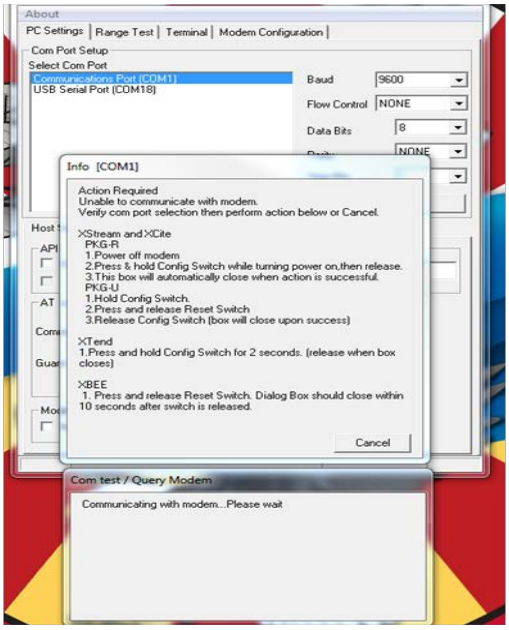
4. Open X-CTU Software.
5. Select a port from the Select Com Port list. It will not be Com 1.



6. Click the Test/Query button. If successful, the window will appear similar to the one shown below, left.



Successful!



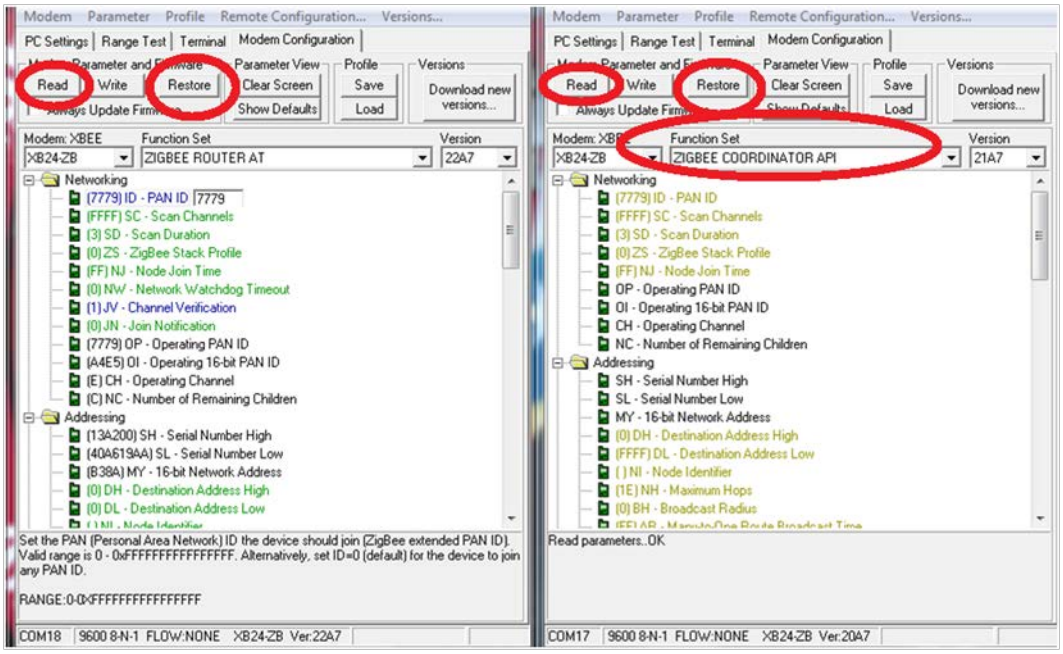
Unsuccessful!

7. If unsuccessful, the window will appear similar to the one shown above, right. Unplug the dongle and plug it into the computer again. Close and reopen the X-CTU window. If the problem persists, there is an issue with the connection between the XBee and dongle or the dongle and your computer. It is best not to have other devices plugged into the USB ports this will make it easier for you to identify the XBee's port name.
8. Click the Modem Configuration tab.
9. Click Download New Versions… to install updates for the X-CTU software and click Done.
10. Plug in the second XBee and dongle and open a new X-CTU window. This XBee will be the Coordinator. Make sure to leave the Router XBee and its X-CTU window alone.
11. Select a port for the Coordinator Xbee from the Select Com Port list. Be sure to select the new com port and not the router's com port.
12. Click the Test/Query button.
    a. In the Router X-CTU window:
        1. Click the Modem Configuration tab.
        2. Click the Read button.
        3. Click the Restore button.
        4. Click the Read button again.

5. Inside the Networking folder, set the PAN ID to an arbitrary number by clicking on it and typing in the number.
6. Change the Channel Verification to 1-ENABLED.
7. Inside the Addressing folder, set the Destination Address High value to 0.
8. Click the Write button.

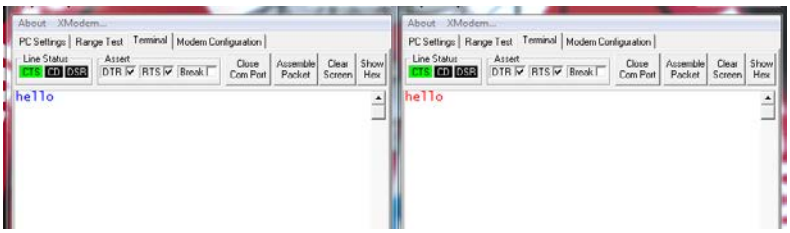
b. In the Coordinator X-CTU window:

1. Click the Modem Configuration tab.
2. Click the Read button.
3. Click the Restore button.
4. Click the Read button again.
5. Change the Function Set from Zigbee Coordinator API to Zigbee Coordinator AT.
6. Set this PAN ID to the same arbitrary number used in the Router window.
7. Click the Write button. The XBee configuration is now complete.



Router Coordinator

Click the Terminal tab in both X-CTU windows to verify that your XBees are communicating. Write in the Router window. The Coordinator window should display the same text.



Notes:

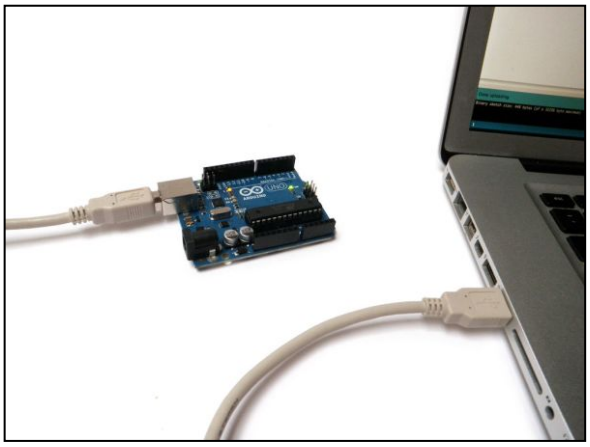
- If the XBees are not communicating properly, click the “PC settings” tab and set baud rate to the same number— usually 9600.
- Refer to the following video tutorials for additional assistance:
  - http://www.youtube.com/watch?v=mPx3TjzvE9U
  - http://www.loveelectronics.co.uk/Tutorials/7/xbee-tutorial-how-to-set-up-your-xbees

2.2.2. ARDUINO PROGRAMMING

This section details how to program the Arduino to control the RC car, this section also includes example code for reference.

To program the Arduino to control the RC car:

1. Download and install the latest version of Arduino software from the Arduino website: http://arduino.cc/en/Main/Software
2. Choose a download that will work best for your computer.
3. Attach the Arduino board to the computer by connecting the USB adapter for the Arduino.



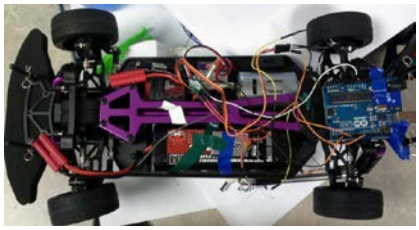
4. Launch the Arduino software, opening a window called a sketch. Copy the following code into a new Arduino window. You can also find this code in the file Arduino_Xbee.txt (see supplemental materials). This program will be for the Arduino attached to the RC car.

```
//Import libraries
#include <Servo.h>
String motorData="";
Servo servox;
Servo servoy;
//Set Initial Speed and turn to zero
int motorSpeed =90;
int motorTurn =90;
```

```
void setup(){
 Serial.begin(9600);
 servox.attach(9);
  servoy.attach(6);
}
void loop(){
 //If there is new data
 if(Serial.available()){
  int i = Serial.read();
  char c = i;
   motorData= motorData+c;
 }
 //keep motor moving at current speed and turn
 moveMotor(motorSpeed,motorTurn);
 //If the new data is correct
 if(motorData.substring(0,1)!="@"){
  motorData="";
 }
 //Once data is long enough, begin to change motor
 if(motorData.length()>7){
  //Serial.println("Step 1:"+motorData);
   checkEnding();
 }
}
//To make sure data was entered in correctly
void checkEnding(){
  if(motorData.substring(7)=":"){
    //Serial.println("Step 2:"+motorData);
    motorSpeed =StringToInt(motorData.substring(1,3));
    motorTurn =StringToInt(motorData.substring(5,8));
    //  Serial.print("Step 3:");
    //  Serial.print(motorSpeed);
    //  Serial.print(" ");
    //  Serial.println(motorTurn);
    moveMotor(motorSpeed,motorTurn);
    motorData="";
  }else{
   motorData="";
  }
}
//adjust speed and turn
void moveMotor(int Mspeed, int Mturn){
```

```
  servox.write(Mspeed);
  servoy.write(Mturn);
}
//convert and integer to a string.
int StringToInt(String s){
  int solution =0;
  int counter = 2;
  int multiplier=1;
  while(counter>=0){
    if(s.substring(counter,counter+1)=="1"){
      solution= solution+multiplier*1;
    }else if(s.substring(counter,counter+1)=="2"){
      solution= solution+multiplier*2;
    }else if(s.substring(counter,counter+1)=="3"){
      solution= solution+multiplier*3;
    }else if(s.substring(counter,counter+1)=="4"){
      solution= solution+multiplier*4;
    }else if(s.substring(counter,counter+1)=="5"){
      solution= solution+multiplier*5;
    }else if(s.substring(counter,counter+1)=="6"){
      solution= solution+multiplier*6;
    }else if(s.substring(counter,counter+1)=="7"){
      solution= solution+multiplier*7;
    }else if(s.substring(counter,counter+1)=="8"){
      solution= solution+multiplier*8;
    }else if(s.substring(counter,counter+1)=="9"){
      solution= solution+multiplier*9;
    }
    multiplier = multiplier*10;
    counter--;
  }
 return solution;
}
```

2.2.3. WIRING THE RC CAR, ARDUINO, AND AN XBEE



The RC car has two servo motors. Each servo motor has three wires coming out of it. One servo controls steering and the other controls speed. The servo attached to the steering mechanism (which has red, brown, and orange wires attached to it) is the top servo motor that attaches to pin 6. The servo attached to the speed mechanism (which has white, black, and red wires coming out

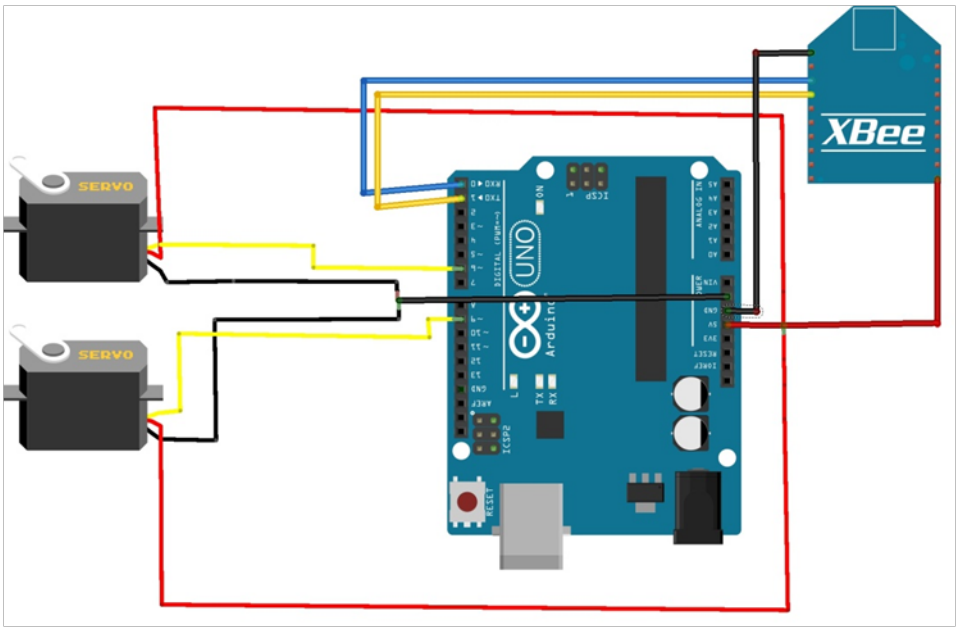
of it) is the bottom servo motor that attaches to pin 9. (Depending on the RC car that is being used, the colors of the wires may change.)The following steps detail how to wire the RC car, Arduino, and the XBEE to enable the RC car to run.

Notes:

- Follow the Fritzing © schematic (below) as closely as possible.
- The wire colors in the diagram may not exactly match your model.
- Many of the wires may need extension wires added to allow the wires to span the distance from their origin to the Arduino board.

Steering Servo

Speed Servo



- The wire colors in the diagram may not exactly match your model.
- Many of the wires may need extension wires added to allow the wires to span the distance from their origin to the Arduino board.
- The 3 red wires, one from each servo and one from the XBee, will need to be connected together with an additional wire; the additional wire will connect to the 5 volt pin.
- The 2 black wires, one from each servo, will need to be connected together with an additional wire; the additional wire will connect to the ground pin.

To connect the XBee to the Arduino board:

1. Connect the red wire to the 5-volt (5v) pin on the Arduino board.
2. Use the black wire to connect the XBee ground pin to the ground (GND) pin on the Arduino board.
3. Connect the XBee DOUT pin to the RX or 0 pin on the Arduino board using the blue wire.
4. Use the yellow wire to connect the XBee DIN pin to the TX or one pin on the Arduino board.

Note: You may need to solder the wires into the XBee dongle in order for them to stay in. Do not solder the XBee dongle with the XBee in it.

The RC car has two servo motors. Each servo motor has three wires coming out of it. One servo controls steering and the other controls speed. The servo attached to the steering mechanism (which has red, brown, and orange wires attached to it) is the top servo motor that attaches to pin 6. The servo attached to the speed mechanism (which has white, black, and red wires coming out of it) is the bottom servo motor that attaches to pin 9. (Depending on the RC car that is being used, the colors of the wires may change.)

To connect the top servo to the Arduino board:

1. Connect the red wire to the 5-volt (5v) pin on the Arduino board.
2. Connect the brown wire to the ground (GND) pin on the Arduino board. The brown wire on the servo motor may be black.
3. Connect the orange wire to the 6-digital pin on the Arduino board . The orange wire on the servo motor may be yellow or white.

To connect the bottom servo (speed control) motor to the Arduino board:

1. Connect the red wire to the 5-volt (5v) pin on the Arduino board.
2. Connect the black wire to the ground (GND) pin on the Arduino board.
3. Connect the white wire to the 9-digital pin on the Arduino board. The white wire on the servo motor may be yellow.