Appendix A – Block Diagrams

Figure A.1 is a block diagram of the base station for our design. Figure A.2 is a block diagram of the robot sprinkler. These parts communicate with one another through the wireless communication blocks.

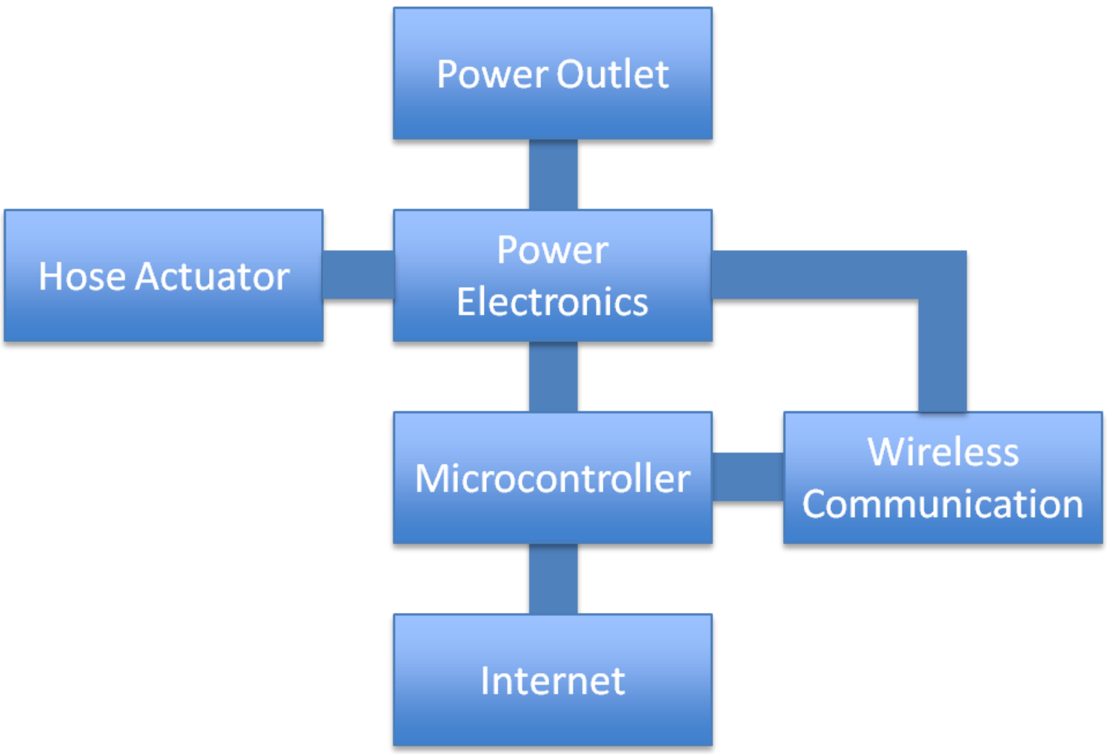


Figure A.1 Base Station Block Diagram

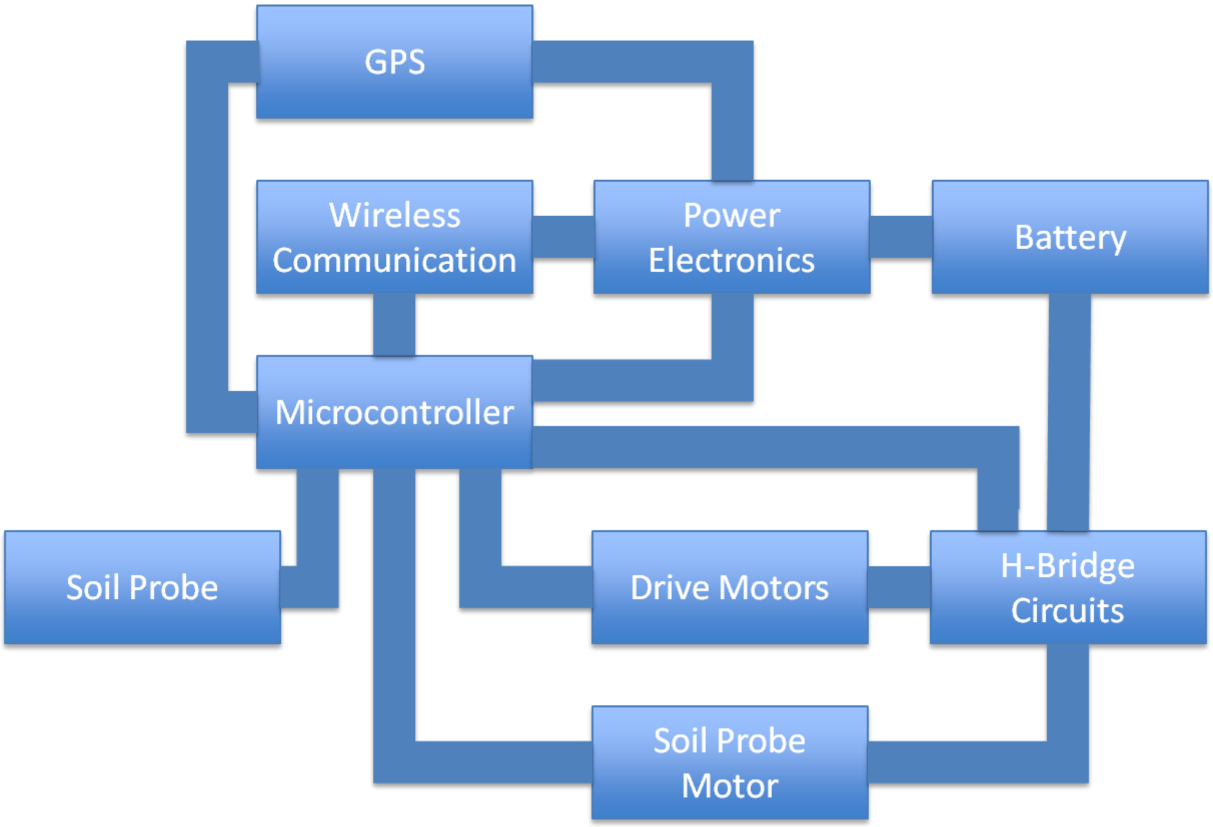


Figure A.2 Robot Sprinkler Block Diagram

Appendix B – Requirements and Verification

Power Outlet Requirements	Verification
1) Supplies 120V ± 5% at 60 Hz ± 0.5%	• Using a Volt-meter, insert the leads into the flat holes of the power outlet. Make sure not to touch the leads while testing. Record the AC voltage of the power outlet to make sure it is within the desired range. • Using an oscilloscope, insert the leads into the flat holes of the power outlet. Make sure not to touch the leads while testing. Record the frequency of the power outlet to make sure it is within the desired range.

Hose Actuator Requirements	Verification
1) Turns the flow of water on and off to the sprinkler a) When given an input signal of 24 VAC ± 1% at 60 Hz ± 0.5% it turns on b) For an input signal less than 5 VAC ± 1% 60 Hz ± 0.5% it remains off	a) • Using a signal generator, supply the hose actuator with 24 VAC ± 1% at 60 Hz ± 0.5%. • Check to see if the solenoid in it moves to the open position. b) • Using a signal generator, supply the hose actuator with a sweep of voltages up to 5 VAC ± 1% 60 Hz ± 0.5%. • Make sure that the solenoid does not move to the open position

Power Electronics Requirements	Verification
1) Low-Drop Voltage Regulator (LDO) is outputting 5 VDC ± 0.1% a) V_{IN} > 5.5VDC ± 0.5% b) I_{OUT} < 5 A ± 0.5%	a) • Probe the input with a Volt-meter. Display signal on oscilloscope. b)

	• Probe the output with an Amp-meter. Ensure that the current draw is less than 5A. Display waveform on an oscilloscope.
2) Wall outlet transforming to correct voltage a) V_{IN} = 120VAC± 5% b) V_{OUT} = 24VAC ± 5%	a) • Using a Volt-meter, measure the input voltage. Display the output on an oscilloscope. b) • Using a Volt-meter, measure the output voltage. Display the output on an oscilloscope.
3) USB connector is receiving 5VDC ± 1% from the computer	• Using a Volt-meter, measure the USB connector voltage. Display the output on an oscilloscope.

Microcontroller Requirements	Verification
1) The Arduino is receiving the correct amount of voltage a) The USB Connector is making a good connection b) V_{IN} at Arduino is 5VDC ± 5%	a) • Use a multimeter at the USB connector to ensure a connection is made. • Check for proper grounding by measuring voltage using a Volt-meter at output from LDO. b) • Probe the USB Arduino V_{CC} port with a Volt-meter and display signal on the output.
2) There must be data transfer between the microcontroller and the wireless communication. a) The Arduino must be able to send information to the transmitter in order to transmit data to the sprinkler robot. b) The Arduino must be able to read the information that the receiver	a) • Wire the transmitter and receiver to the Arduino. • Connect the Arduino to a PC and open serial communication • Send a binary signal that is at least 20 bits long • Have the Arduino print out the signal values that it is sending to the

<table>
<tr>
<td>is getting from the sprinkler robot.</td>
<td>transmitter.
Have the Arduino at the sprinkler robot print out the signal that it is receiving.
Compare the transmitted signal at the base station Arduino to the signal received at the sprinkler robot Arduino.
Do this test 10 times and make sure that the same signal is sent and received all 10 times. It must pass all 10 tests.
b)
Wire the transmitter and receiver to the Arduino.
Connect the Arduino to a PC and open serial communication
Send a binary signal that is at least 20 bits long
Have the Arduino print out the signal that it gets at the receiver.
Compare the printed received data to that of the sprinkler robot Arduino.
Do this test 10 times and make sure that the same signal is sent and received all 10 times. It must pass all 10 tests.</td>
</tr>
<tr>
<td>3) Gathers information about the weather to predict when to activate the sprinkler robot
a) Arduino gathers data from NOAA website
b) Determines when to activate the sprinkler robot</td>
<td>a)
Make sure that the internet requirements are satisfied.
Connect the Arduino to a PC and open serial communication.
Check what variable values are being stored in the Arduino. Makes sure that the variables being stored are the same as those from online
Do this test 5 times and have the weather data online be different for each test. It must pass all 5 tests.</td>
</tr>
</table>

	b) • Connect the Arduino to a PC and open serial communication. • Run through the program to see if the anticipated variables are the same as the actual variables. • Check if the predictions about robot activation match what is intended. • Do this test 5 times with different input variables each time. It must pass the test all 5 times.

Wireless Communication Requirements	Verification
1) The Arduino is sending the correct data type (byte) to the encoder. a) Correct data from the Arduino at each bit of the D0-D7 values. b) SEND pin is set to high c) DATA_OUT is outputting serial data stream	a) • Compare values set in Arduino code to the values measured with 8-LEDs at the D0-D7 pin. • Do this test 1000 times with different input variables each time. Record in table and look for errors. b) • Using a Volt-meter, measure the voltage at the TX_CNTL pin to ensure that the transmitter has been activated. Voltage should be set as high. c) • Using a Volt-meter, measure the voltage at the DATA_OUT pin and ensure bits are being transferred. Display output on oscilloscope. • Do this test 1000 times with different input variables each time. Record in table and look for errors.
2) Transmitter is sending data on the correct channel	• Probe the antenna voltage to ensure a signal is being transmitted by using a Volt-meter and oscilloscope. • Using a Spectrum Analyzer, find the

	signal from the range 906Mhz – 921Mhz. Ensure the dB is high enough for clean reception to be made.
3) Receiver chip is getting data from the transmitter chip a) Ensure channels are set to correct frequency for both receiver and transmitter b) DATA pin on receiver chip is outputting voltage	a) • Check dip-switches to make sure they are set to the correct values b) • Using a Volt-meter, measure the DATA pin on receiver chip. Ensure that there is a voltage being outputted. • Display signal on oscilloscope. • Do this test 1000 times with different input variables each time. Record in table and look for errors.
4) The decoder chip is receiving the data and sending it to the Arduino a) DATA_IN pin sees a rising edge b) Code Word matches the one in memory c) D0-D7 contain correct data	• • Using a Volt-meter, measure the Probe DATA_IN pin on decoder chip. Ensure that there is a voltage being inputted. • Display signal on oscilloscope. • Do this test 1000 times with different input variables each time. Record in table and look for errors. • LEARN mode accepts correct code word from encoder ensuring a transmission is possible • Do this test 1000 times with different input variables each time. Record in table and look for errors. • Display D0-D7 on an LED to ensure that they are the correct values being sent from Arduino. • Do this test 1000 times with different input variables each time. Record in table and look for errors. • Verify BAUD rate is correct by tracking bits per second on Arudino.

	Compare rate to that of Encoder and Decoder rate.

Internet Requirements	Verification
1) Connects to the NOAA website for weather information	• Plug an Ethernet cable into the internet outlet and then into a PC • Check that you can access the NOAA website at www.noaa.gov

GPS Requirements	Verification
1) Must update coordinates from global positioning satellites utilizing WAAP capabilities. a) In a stationary position with an accuracy of less than one meter. b) During movement with an accuracy of 2.5m ± 0.2m.	a) • Set up the Venus GPS device in the laboratory for stationary testing. • Connect the output of the Venus GPS device to the Arduino. • Enable WAAP on the Venus GPS device. • Connect one of the two GPS antennas on the roof of Everitt Laboratory to the GPS device. • Cross verify results with the actual coordinates of the antennas by printing out the results on the Arduino. • Cross verify coordinate data from the Venus GPS with the actual location of the antenna. • Fix the coding with the Arduino if the coordinate reading does not match the actual location within the acceptable error. • Repeat previous steps for the other GPS antenna on the roof of Everitt Lab. • Repeat steps using the GPS antenna that will be used on the Sprinkler Robot. • Once an accuracy of less than one

	meter has been achieved for each antenna, and is consistent within the acceptable error for multiple trials, the stationary position testing will be complete. b) • Install the Venus GPS device with its antenna on the Sprinkler Robot. • Move the robot around Everitt Lab's perimeter while sending the coordinate updates to the Base Station. • Print out the data at the base station to display the coordinate updates. • Cross verify the coordinate results with the actual coordinates. • Continue with cross verifying coordinates for movement around Everitt Lab until the proper readings show up. • Once an accuracy of 2.5m ± 0.2m has been achieved, and is consistent within the acceptable error for multiple trials, the movement testing will be complete.

Wireless Communication Requirements	Verification
1) The Arduino is sending the correct data type (byte) to the encoder. a) Correct data from the Arduino at each bit of the D0-D7 values. b) SEND pin is set to high c) DATA_OUT is outputting serial data stream	a) • Compare values set in Arduino code to the values measured with 8-LEDs at the D0-D7 pin. • Do this test 1000 times with different input variables each time. Record in table and look for errors. b) • Using a Volt-meter, measure the TX_CNTL pin to ensure that the transmitter has been activated.

	Voltage should be set as high. c) • Using a Volt-meter, measure the DATA_OUT pin and ensure bits are being transferred. • Display output on oscilloscope. • Do this test 1000 times with different input variables each time. Record in table and look for errors.
2) Transmitter is sending data on its channel a) Sending on the correct channel.	a) • Using a Volt-meter, measure the antenna voltage to ensure a signal is being transmitted • Display output on oscilloscpe • Using a Spectrum Analyzer, find the signal from the range 906Mhz – 921Mhz. Ensure the dB is high enough for clean reception to be made.
3) Receiver chip is getting data from the transmitter chip a) Ensure channels are set to correct frequency for both receiver and transmitter b) DATA pin on receiver chip is outputting voltage	a) • Check dip-switches to make sure they are set to the correct values b) • Using a Volt-meter, measure the DATA pin on receiver chip. Ensure that there is a voltage being outputted. • Display signal on oscilloscope. • Do this test 1000 times with different input variables each time. Record in table and look for errors.
4) The decoder chip is receiving the data and sending it to the Arduino a) DATA_IN pin sees a rising edge b) Code Word matches the one in memory c) D0-D7 contain correct data	a) • Using a Volt-meter, measure the Probe DATA_IN pin on decoder chip. Ensure that there is a voltage being inputted. • Display signal on oscilloscope. • Do this test 1000 times with different input variables each time. Record in

	table and look for errors. b) • LEARN mode accepts correct code word from encoder ensuring a transmission is possible • Do this test 1000 times with different input variables each time. Record in table and look for errors. c) • Display D0-D7 on an LED to ensure that they are the correct values being sent from Arduino. • Do this test 1000 times with different input variables each time. Record in table and look for errors. • Verify BAUD rate is correct by tracking bits per second on Arudino. Compare rate to that of Encoder and Decoder rate.

Power Electronics Requirements	Verification
1) First low-drop voltage regulator (LDO) is outputting 5 VDC and the second LDO is outputting 3.3 VDC. a) $V_{IN} > 5.5VDC \pm 0.1\%$ b) $I_{OUT} < 5$ A $\pm$ 0.5%	a) • Probe the output with a Volt-meter and read the voltage. Display signal on oscilloscope. • Probe the input with a Volt-meter. Display signal on oscilloscope. b) • Probe the output with an Amp-meter. Ensure that the current draw is less than 5A. Display waveform on oscilloscope.

Power Electronics Requirements	Verification
1) Both the 12 VDC, 7 AH and 12 VDC 35 Ah batteries are fully charged	• Using a Volt-meter, measure the output voltage of both batteries • If the voltage is less than 12 VDC± 0.5%, use a battery recharger

Microcontroller Requirements	Verification
1) The Arduino is receiving the correct amount of voltage a) The USB Connector is making a good connection b) V_{IN} at Arduino is 5VDC ± 5%	a) • Measure using a multimeter at the USB connector to ensure it is not a floating value. • Check for proper grounding by measuring the voltage using a Volt-meter at output from LDO. b) • Probe the USB Arduino V_{CC} port with a Volt-meter and display signal on an oscilliscope.
1) There must be data transfer between the microcontroller and the wireless communication. a) The Arduino must be able to send information to the transmitter in order to transmit data to the base station. b) The Arduino must be able to read the information that the receiver is getting from the base station.	a) • Wire the transmitter and receiver to the Arduino. • Connect the Arduino to a PC and open serial communication. • Send a binary signal that is at least 20 bits long • Have the Arduino print out the signal values that it is sending to the transmitter. • Have the Arduino at the base station print out the signal that it is receiving. • Compare the transmitted signal at the sprinkler robot Arduino to the signal received at the base station Arduino. • Do this test 10 times and make sure that the same signal is sent and received all 10 times. It must pass all 10 tests. b) • Connect the Arduino to a PC and open serial communication. • Send a binary signal that is at least 20 bits long

	• Have the sprinkler robot Arduino print out the signal that it gets at the receiver. • Compare the printed received data to that of the base station Arduino. • Do this test 10 times and make sure that the same signal is sent and received all 10 times. It must pass all 10 tests.
2) The microcontroller must receive the sprinkler robot's coordinates from the GPS. a) The Arduino must be able to read the information that the GPS is providing.	a) • Wire the Venus GPS device to the Arduino. • Connect the Arduino to a PC and open serial communication. • Make the Arduino print out the coordinate values that it receives from the Venus GPS device. • Compare the data being printed out with the actual data that the GPS device is sending to the Arduino. • Do this test 5 times and make sure that it passes all 5 times.
3) Sends the measured soil resistance value to the wireless transmitter a) Measures the soil resistance b) Sends the soil resistance to the wireless communication	a) • Verify that all soil probe requirements are satisfied b) • Connect the Arduino to a PC and open serial communication • Record the values of the variables that are being sent to the wireless communication • Using an oscilloscope, see if the variable values that are being sent match up with the measured output • Do this test 5 times and make sure that it passes all 5 times. Each time use different resistance values

Soil Probe Requirements	Verification
1) Measure soil resistance with up to 5% error a) Resistance between probes must be greater than 100MΩ b) Circuit reference resister must be the same as that used in the program c) Resistance value measured by Arduino is accurate within 5%	a) • Make sure that the soil probes are not contacting any other material • Using an Ohm-meter, measure the resistance across the two soil probes. Verify the resistance is greater than 100MΩ b) • Using an Ohm-meter, measure the resistance of the reference resister • Open up the program for the Arduino • Check that the variable value in the Arduino program is the same as the reference resister value. c) • Connect the Arduino to a PC and open serial communication • Put across the soil probes a test resistance value • Compare the stored variable value for measured resistance to that of the test resistance value • Do this test 5 times and make sure that it passes all 5 times. Use a different reference resistance value each time

Drive Motors Requirements	Verification
1) 144:1 turns ratio at 2.1A ± 1% and 12 VDC ± 1%	• Use the potentiometer to track how many turns the motor makes every minute. • Record the data and check if the motor is running properly.
2) Adjustable Speed	• Have the Arduino vary PWM signals and see if the speed can be adjusted.

	• Record values in a table for different PWM signals and compare the speed.

H-Bridge Circuit Requirements	Verification
1) Gate Driver being powered correctly a) SV+ and PV+ getting correct voltages b) UVout floating	a) • Probe the voltages at SV+ and PV+ to ensure they are 12VDC ± 1%. Display voltage signal on oscilliscope. b) • Probe the UVout pin with a multimeter and ensure that it is floating

2) PWM signals correct a) T_{gate} and B_{gate} receive the correct signals at the correct frequencies b) Power MOSFETs turning on	a) • Compare signals coming from the Arduino to those that are going into the gate driver to make sure they match • Ensure voltage is strong enough to power signal by using a Volt-meter and measuring the INtop pin and INbottom pin. • Compare the frequencies at INtop to T_{gate} using the oscilloscope and see if they match. • Compare the frequencies at INbot to B_{gate} using the oscilloscope and see if they match. b) • Probe the power MOSFET with a current source and ensure that current is flowing through it. Display signal on oscilloscope. • Probe the V_{GS} voltage and ensure it is high enough to turn the gate on based off of the data sheets.

Soil Probe Motor Requirements	Verification
1) Is able to insert and remove the soil probe into the ground with ½" accuracy a) Power electronics supplies ±12 VDC ± 0.5% b) Measures position within ½" accuracy	a) • Using a Volt-meter, check that voltage input into the motor • For forward slide movement, the voltage should be +12 VDC ± 0.5% • For reverse slide movement, the voltage should be -12 VDC ± 0.5% b) • Connect the Arduino to a PC and open serial communications

	• Move the slide to a location and check that the Arduino variables are storing the anticipated values for that location with ½” accuracy

APPENDIX C – SCHEMATICS

Figure C.1 is the schematic for the Venus GPS. Figure C.2 is the schematic of the Arduino Mega 2560. Figure C.3 is the schematic of the Ethernet Arduino. Figure C.4 is the schematic of the buck converter. Figure C.5 is the schematic of the power distribution for the sprinkler robot. Figure C.6 is the schematic of the power distribution for the base station. Figure C.7 is the schematic of the initial receiver circuit with a decoder. Figure C.8 is the schematic of the initial transmitter circuit with an encoder.

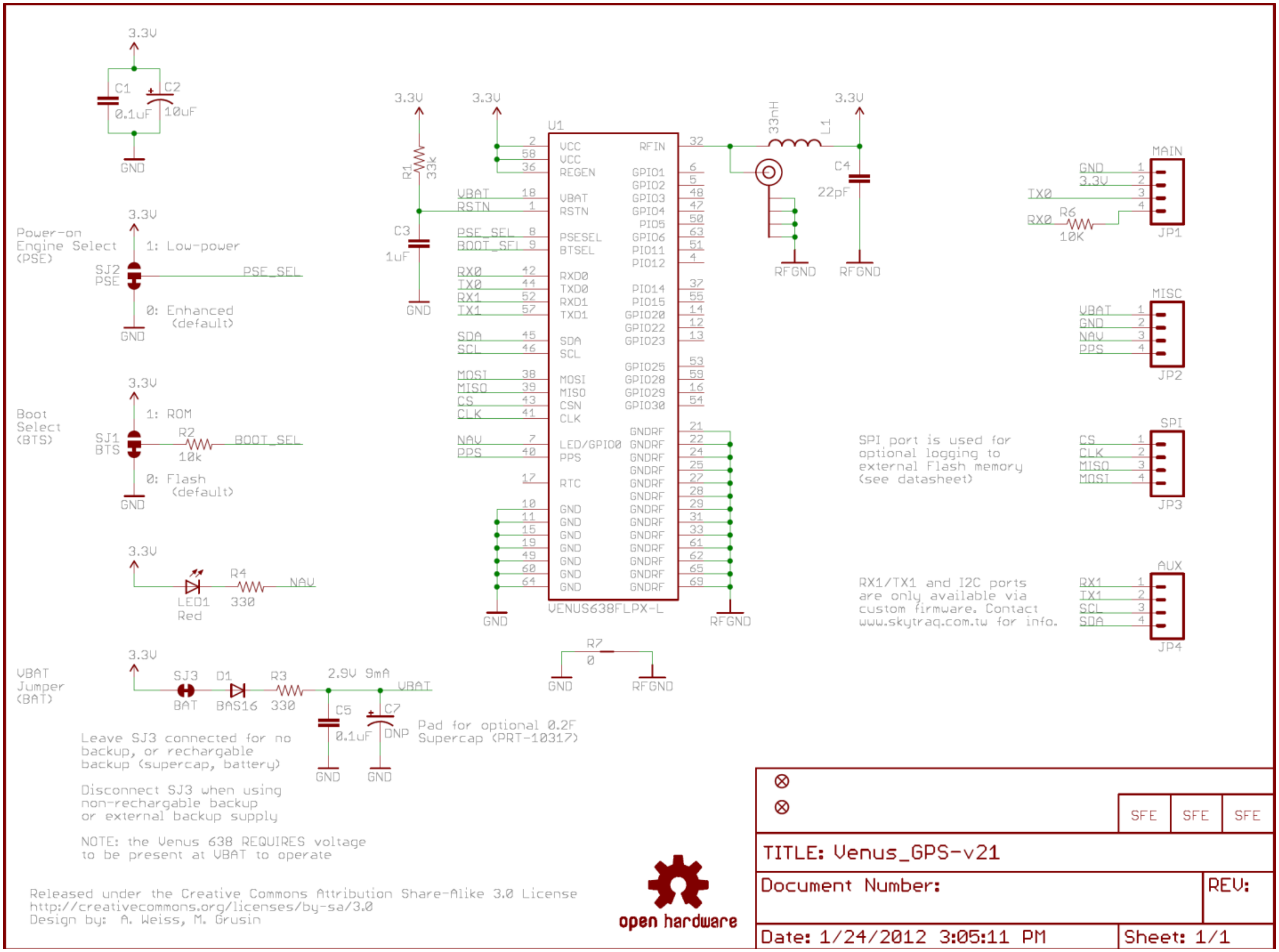


Figure C.1 Venus GPS v21

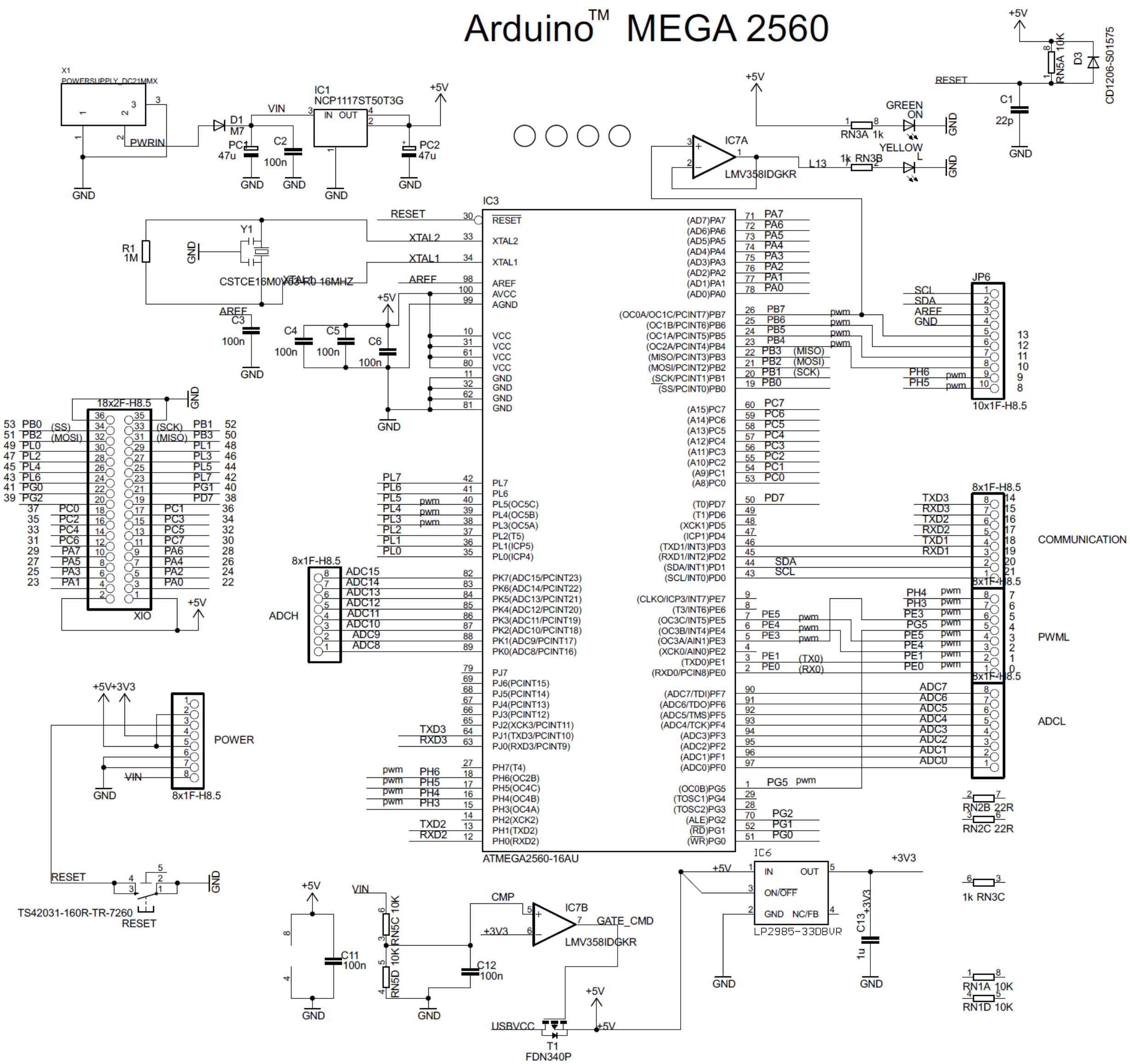


Figure C.2 Arduino MEGA 2560 Diagram

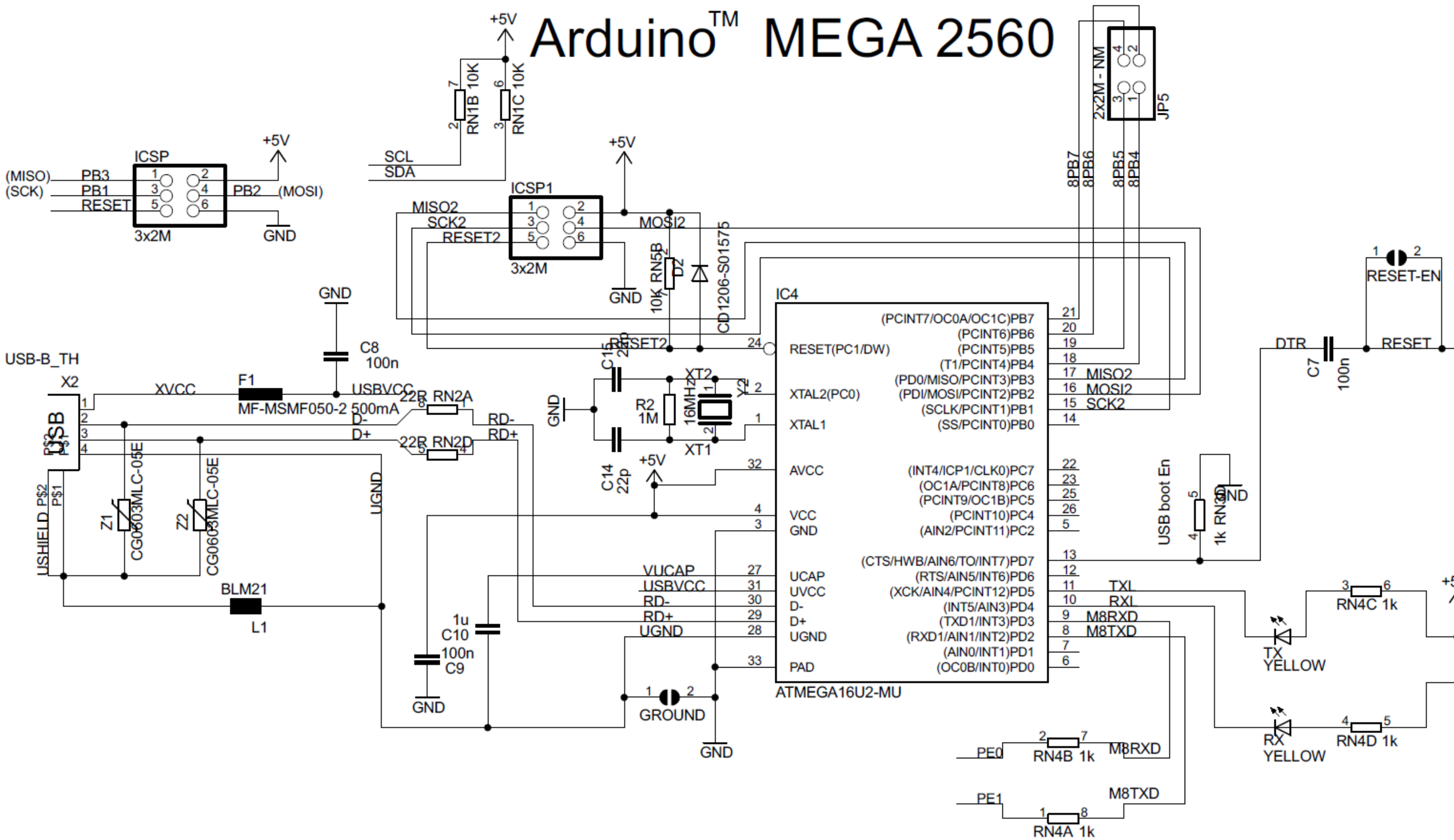


Figure C.2 Arduino MEGA 2560 Diagram (continued)

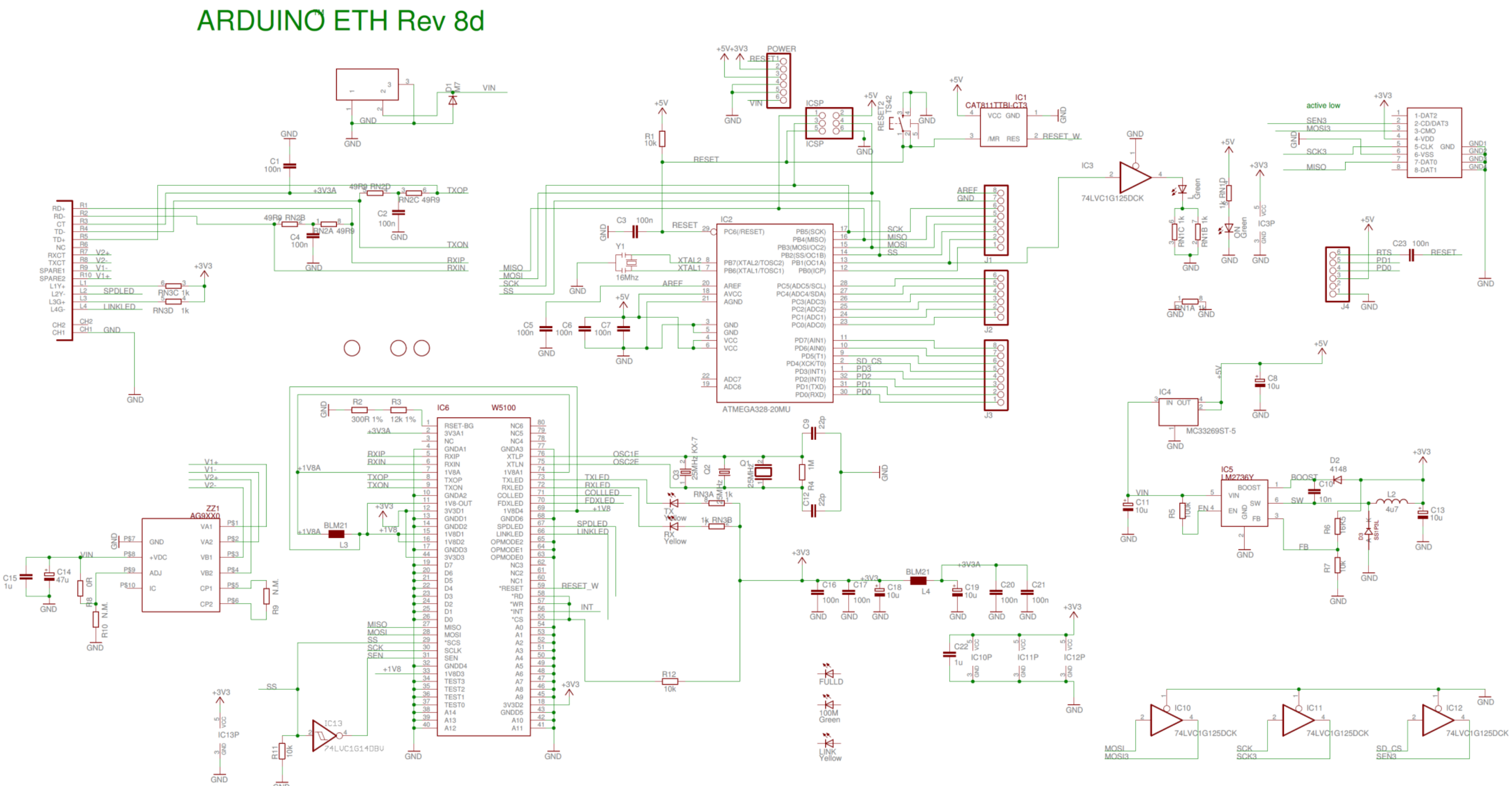


Figure C.3 Arduino Ethernet

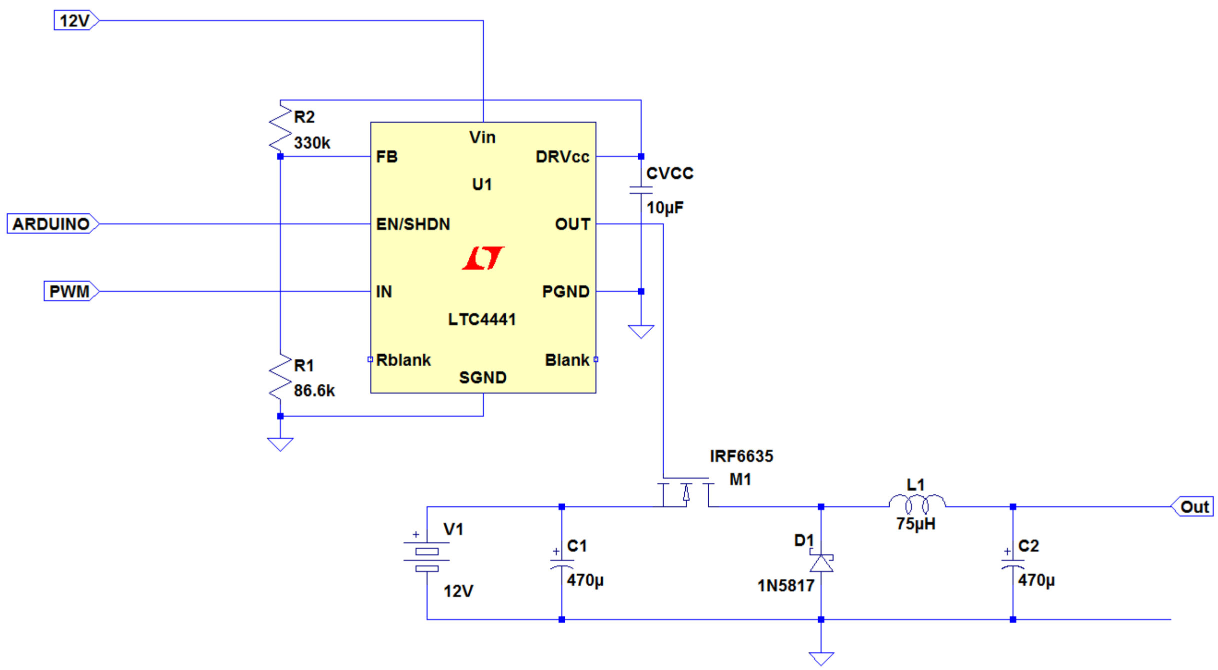


Figure C.4 Buck Converter 12V to 5V Design

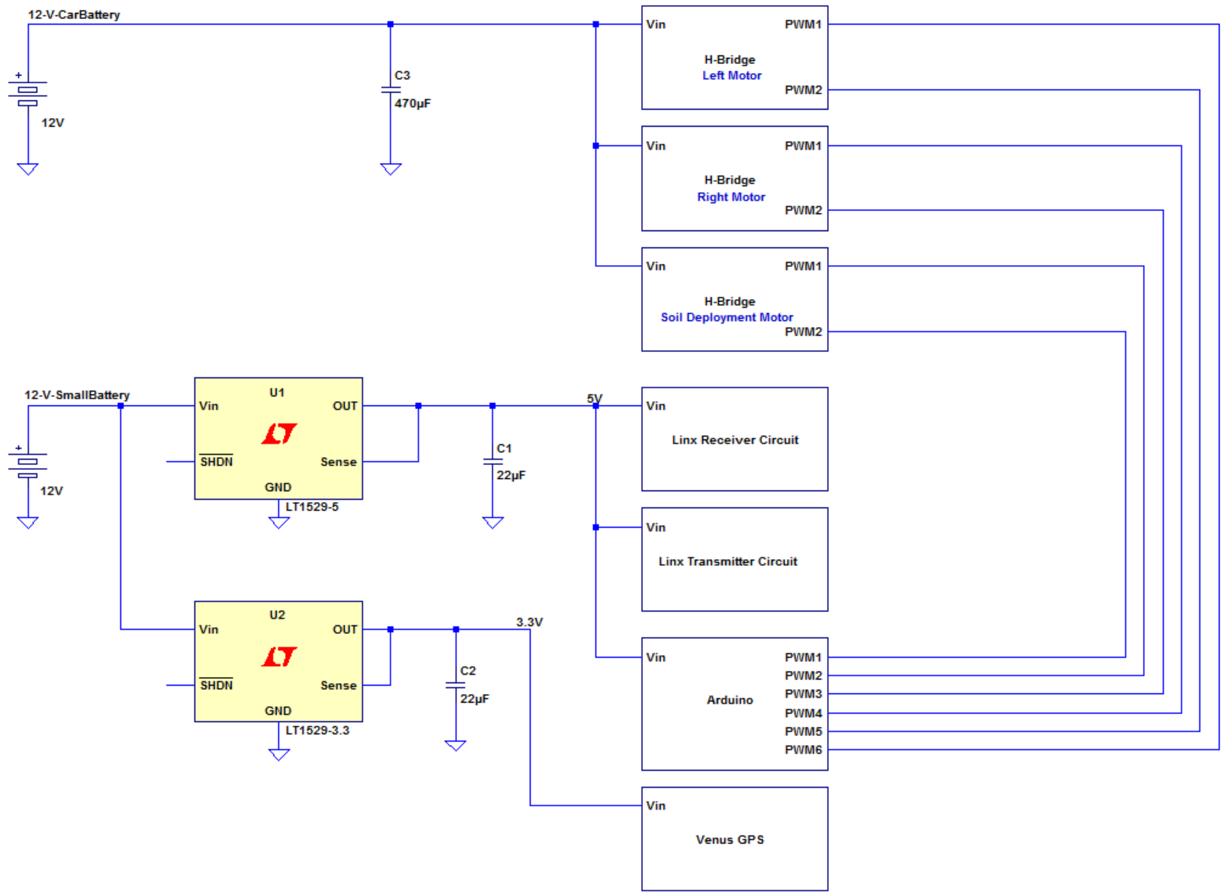


Figure C.5 Final Power Distribution of the Robot

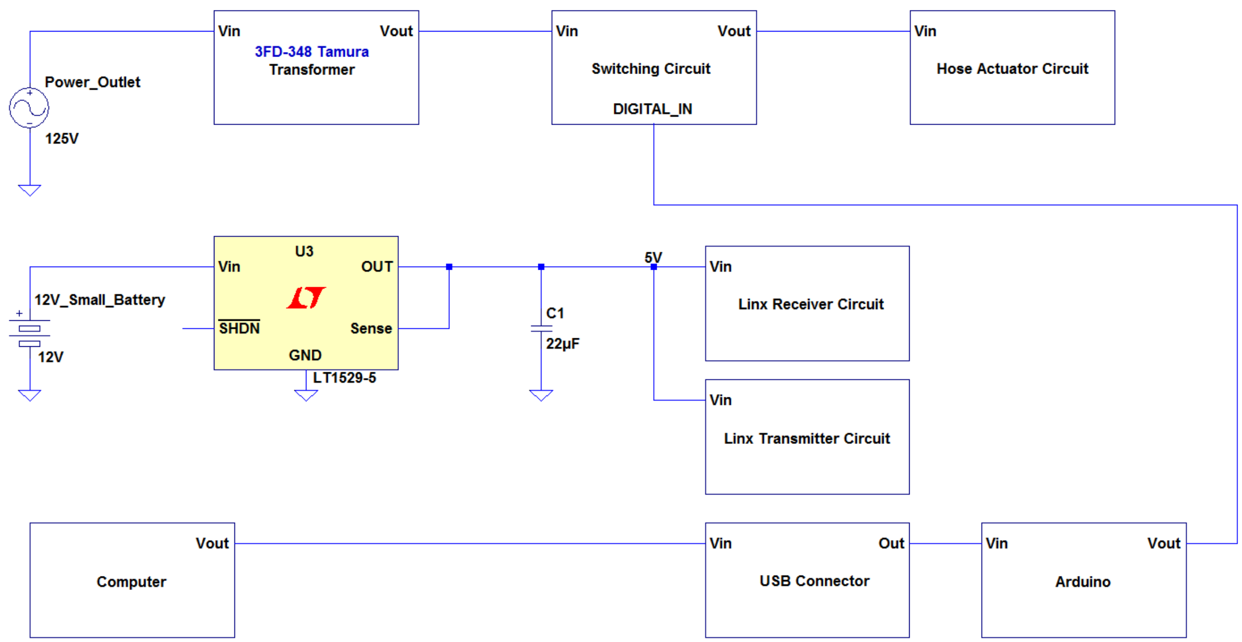
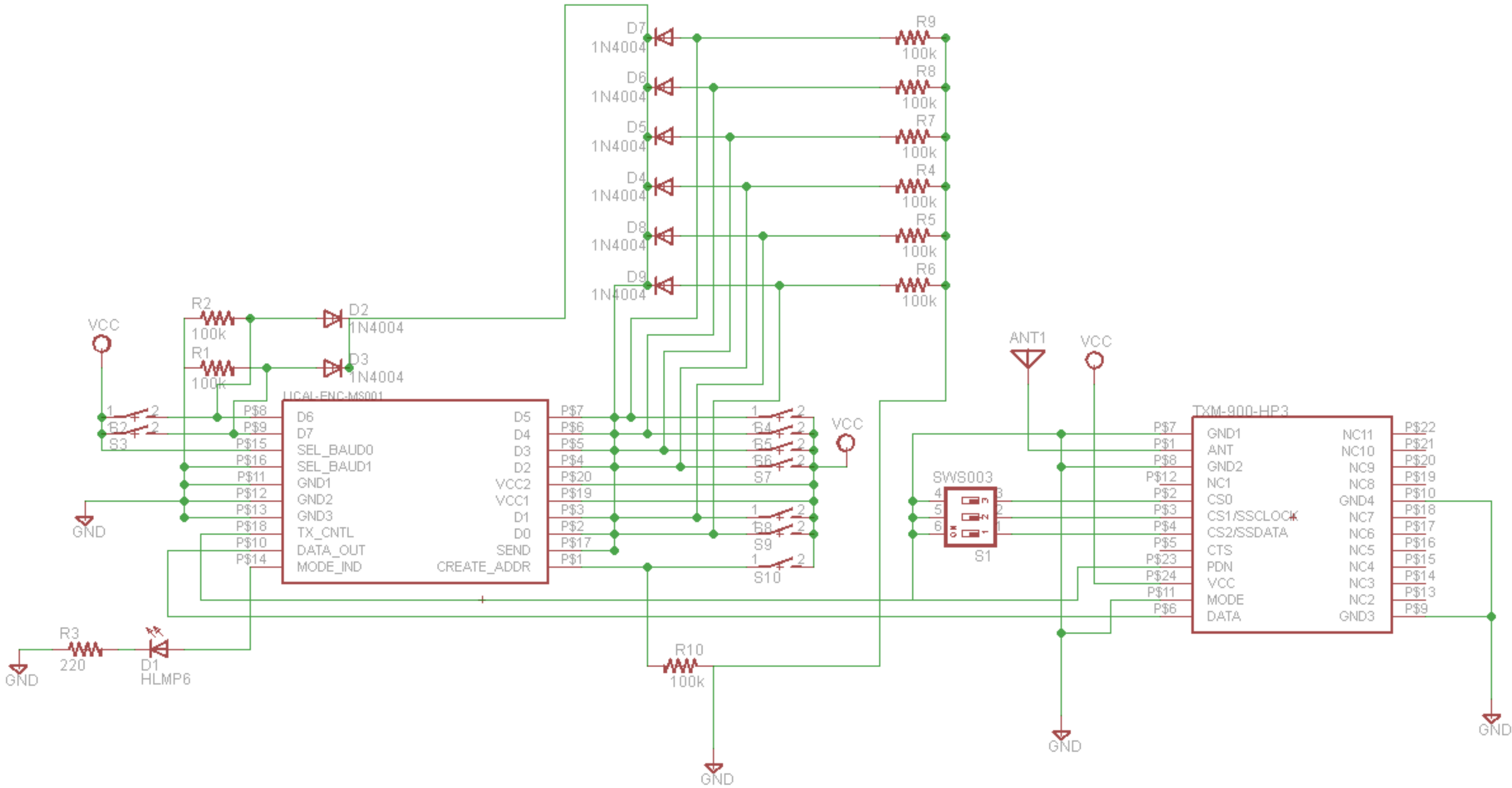


Figure C.6 Final Power Distribution of the Base Station



All switches are controlled externally by the Arduino which will create an address when ready and pick the byte it wants to transmitt

Figure C.7 Initial Receiver Circuit with Decoder

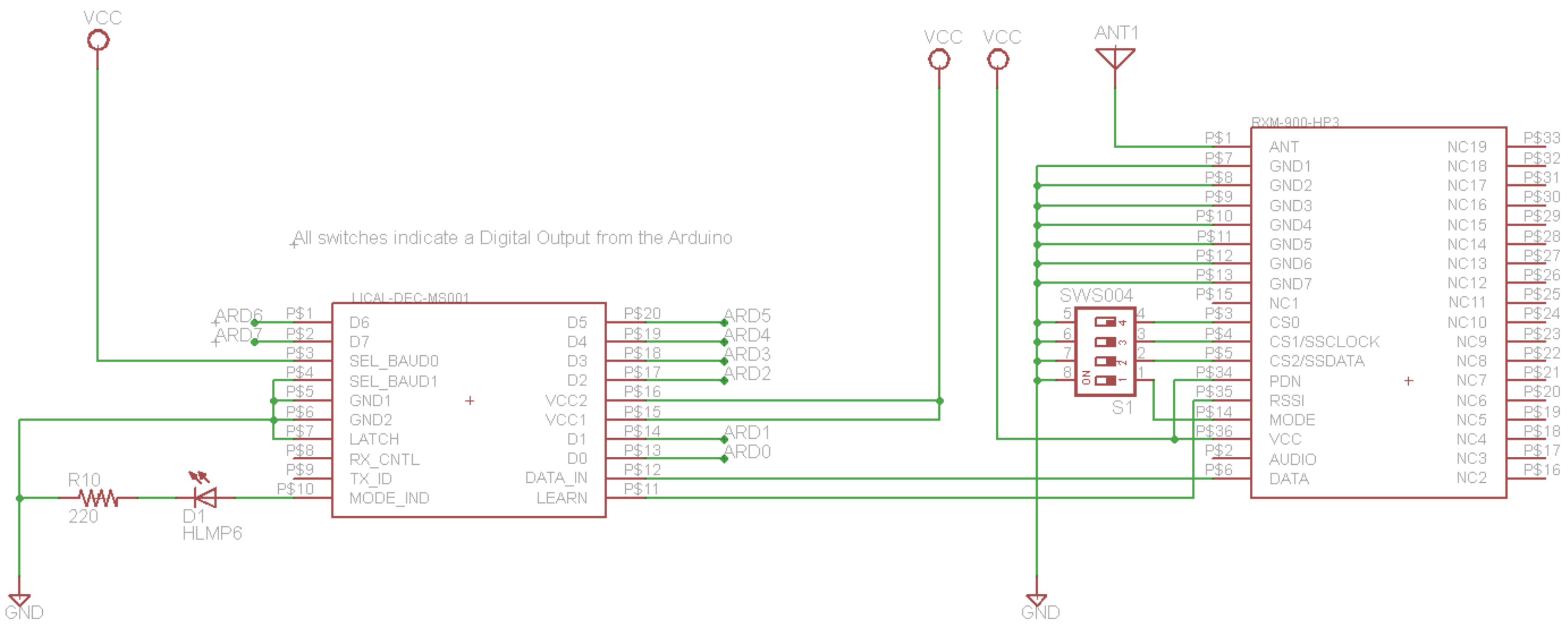


Figure C.8 Initial Transmitter Circuit with Encoder

APPENDIX D – Arduino Programs

Figure D.1 is the code for the base station Arduino Mega 2560. Figure D.2 is the code for the robot sprinkler Arduino Mega 2560. Figure D.3 is the code for the weather forecast for the Ethernet Arduino. Figure D.4 is a part of the code used for communication between the Arduino Megas.

```
//--------------------------------------------------------------------------//

//      Smart Sprinkler Robot System - Base Station
//

//      ECE 445 Senior Design                       //

//      University of Illinois at Urbana/Champaign        //

//      Denis Kurtovic, Kevin Johnson, and Jose L Orozco Jr.
//

//--------------------------------------------------------------------------//


  int integer = 678;


char startByte[4] = "#$%";

//char middleByte = '&';

char endByte[4] = "(!)";


char programMode = '1';      // start in mode 1


char nextProgramMode = 's';    // holds the wirelessly commanded
mode to be switched to

const unsigned long TIME_BETWEEN_COM_SEND = 5000; //
how often in ms a signal is sent

unsigned long lastComSendTime = 0;    // time of last sent wireless
communication


byte wirelessByteRecieved;

int stringPosition = 0;


const int ACTUATOR_PIN = 50;


//char c1[20];

//char c2[20];


void setup()

{

  Serial.begin(9600);

  Serial1.begin(9600);  // transmit

  Serial2.begin(9600);  // receive


  pinMode(ACTUATOR_PIN, OUTPUT);

}


void loop()

{

//Main Program----------------------------------------------------------------
-//

  // Communication from computer or sprinkler robot------------------
---------//

  if (Serial.available() > 0) {  // only when somthing is typed

    programMode = Serial.read(); // used to override the current
mode

  }


  // the string to be recieved is "#$%[single number for mode]&[soil
resistance](!)" , ex: #$%3&1234567(!)

  if (Serial2.available() > 0) { // when something is recieved
```

```
wirelessByteRecieved = Serial2.read();


if(wirelessByteRecieved == '#' && stringPosition == 0){
  stringPosition = 1;
}
else if (wirelessByteRecieved == '$' && stringPosition == 1){
  stringPosition = 2;
}
else if (wirelessByteRecieved == '%' && stringPosition == 2){
  stringPosition = 3;
}
else if (stringPosition == 3) { // read in the mode being sent
  nextProgramMode = wirelessByteRecieved;
  stringPosition = 4;
}
else if (wirelessByteRecieved == '&' && stringPosition == 4) {
  stringPosition = 5;

  if (programMode == '2') { // when waiting for soil probe results
    Serial.println("Soil probe results: ");
  }

}
else if (wirelessByteRecieved == '(' && stringPosition == 5) {
  stringPosition = 6;

  if (programMode == '2') { // when waiting for soil probe results
    Serial.println(" kOhms");
  }

}
else if (stringPosition == 5) {

    if (programMode == '2') { // when waiting for soil probe
results
      char c = wirelessByteRecieved;
      Serial.print(c);       // just prints out resistance results
    }

}
else if (wirelessByteRecieved == '!' && stringPosition == 6) {
  stringPosition = 7;
}
else if (wirelessByteRecieved == ')' && stringPosition == 7) {
  programMode = nextProgramMode;
  stringPosition = 0;

  Serial.println("Wireless data recieved");
  Serial.print("Advancing to mode ");
  Serial.println(programMode);
}
else {
    Serial.print("Not matching byte recieved: ");
    Serial.println(wirelessByteRecieved);
    stringPosition = 0;
  }
 }
 //----------------------------------------------------------------------------//

 //All of the program modes-------------------------------------------------
--//
 else { // serial2 not available, run the program
   if (programMode == '1') { // waits until the user starts the
program
                    // this is done by inputing '2' from the computer
   }

   // tells the robot to go to mode 2 where it will go to the GPS
location
   else if (programMode == '2') {
    //TELL ROBOT TO GO TO MODE 2
    if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

     // to the base station
     Serial1.write(startByte);
     Serial1.write('2');     // tell sprinkler robot to advance to next
mode
     Serial1.write(endByte);

     // verifies what was sent
```

```
      Serial.print("Sent to base station: ");
      Serial.print(startByte);
      Serial.print('2');
      Serial.println(endByte);

      lastComSendTime = millis();
    }
  }

  // the robot is measuring the soil
  else if (programMode == '3') {
    // this mode is not reached by base station
  }

  // the robot is sending soil resistance measurement to the robot
  else if (programMode == '4') {
    //  this mode is not reached by base station
  }

  // user decides if they want to water or not, then tells robot to
return home
  else if (programMode == '5') {
    Serial.println("Do you want to water the lawn? y/n");
    while (!Serial.available()) {
    }
  }

  else if (programMode == 'y') { // waters the lawn
    Serial.println("Watering the lawn for 60 seconds");
    //LAWN WATERING THING
    digitalWrite(ACTUATOR_PIN, HIGH);
    delay(60000);
    Serial.println("Turning water off");
    digitalWrite(ACTUATOR_PIN, LOW);

    programMode = 'x'; // Tells robot to return
  }

  else if (programMode =='n') { // does not water the lawn
    Serial.println("Not watering the lawn");
    programMode = 'x'; // Tells robot to return
  }

  else if (programMode == 'x') { // Tells robot to return
    //TELL ROBOT TO GOT TO MODE 6
    if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

      // to the base station
      Serial1.write(startByte);
      Serial1.write('6');      // tell sprinkler robot to advance to next
mode
      Serial1.write(endByte);

      // verifies what was sent
      Serial.print("Sent to base station: ");
      Serial.print(startByte);
      Serial.print('6');
      Serial.println(endByte);

      lastComSendTime = millis();
    }
  }

  // robot moves back to base station
  else if (programMode == '6') {
    //  this mode is not reached by base station
  }

  // robot informs the base station that it is done with the program
  else if (programMode == '7') {
    //  this mode is not reached by base station
  }

  // this mode is reached after it base station hears the robot is done
  else if (programMode == '8') {
    Serial.println("Sprinkler robot is done with program!");
    Serial.println("Send \"1\" to restart the program");
    while (!Serial.available()) {
    }
```

```
    }

    //--------------------------------------------------------------------------//

    // Extra tests for debugging purposes-----------------------------------
----//

    else if (programMode == 'f') {

      //TELL ROBOT TO GO TO MODE f, forward

      if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

        // to the base station

        Serial1.write(startByte);

        Serial1.write('f');     // tell sprinkler robot to advance to next
mode

        Serial1.write(endByte);

        // verifies what was sent

        Serial.print("Sent to base station: ");

        Serial.print(startByte);

        Serial.print('f');

        Serial.println(endByte);

        lastComSendTime = millis();

      }

    }

    else if (programMode == 'v') {

      //TELL ROBOT TO GO TO MODE v, reverse

      if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

        // to the base station

        Serial1.write(startByte);

        Serial1.write('v');     // tell sprinkler robot to advance to next
mode

        Serial1.write(endByte);

        // verifies what was sent

        Serial.print("Sent to base station: ");

        Serial.print(startByte);

        Serial.print('v');

        Serial.println(endByte);

        lastComSendTime = millis();

      }

    }

    else if (programMode == 'r') {

      //TELL ROBOT TO GO TO MODE r, right

      if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

        // to the base station

        Serial1.write(startByte);

        Serial1.write('r');     // tell sprinkler robot to advance to next
mode

        Serial1.write(endByte);

        // verifies what was sent

        Serial.print("Sent to base station: ");

        Serial.print(startByte);

        Serial.print('r');

        Serial.println(endByte);

        lastComSendTime = millis();

      }

    }

    else if (programMode == 'l') {

      //TELL ROBOT TO GO TO MODE l, left

      if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

        // to the base station

        Serial1.write(startByte);

        Serial1.write('l');     // tell sprinkler robot to advance to next
mode

        Serial1.write(endByte);

        // verifies what was sent

        Serial.print("Sent to base station: ");

        Serial.print(startByte);

        Serial.print('l');
```

```
      Serial.println(endByte);

      lastComSendTime = millis();
    }
  }

  else if (programMode == 's') {
    //TELL ROBOT TO GO TO MODE s, stop
    if (lastComSendTime + TIME_BETWEEN_COM_SEND < millis()) { // check if it is time to send results

      // to the base station
      Serial1.write(startByte);
      Serial1.write('s');      // tell sprinkler robot to advance to next mode
      Serial1.write(endByte);

      // verifies what was sent
      Serial.print("Sent to base station: ");
      Serial.print(startByte);
      Serial.print('s');
      Serial.println(endByte);

      lastComSendTime = millis();
    }
  }

  else if (programMode == 'p') {
    //TELL ROBOT TO GO TO MODE 3, probing the ground
    if (lastComSendTime + TIME_BETWEEN_COM_SEND < millis()) { // check if it is time to send results

      // to the base station
      Serial1.write(startByte);
      Serial1.write('3');      // tell sprinkler robot to advance to next mode
      Serial1.write(endByte);

      // verifies what was sent
      Serial.print("Sent to base station: ");
      Serial.print(startByte);
      Serial.print('3');
      Serial.println(endByte);

      lastComSendTime = millis();
    }
  }
 }
}
```

Figure D.1 Code for Base Station Arduino Mega 2560

```
//-----------------------------------------------------------------------------//

//          Smart Sprinkler Robot System - Sprinkler Robot
//

//          ECE 445 Senior Design                                //

//          University of Illinois at Urbana/Champaign                  //

//          Denis Kurtovic, Kevin Johnson, and Jose L Orozco Jr.
//

//-----------------------------------------------------------------------------//



//Initialization for the soil probe motor movement----------------------
------//

const int SOIL_MOTOR_DOWN_PIN = 52; // when HIGH, move
soil motor down

const int SOIL_MOTOR_UP_PIN = 48; // when LOW, move soil
motor up

const int MOVE_SOIL_MOTOR_TIME = 3000; // time to move
the soil probe motor in ms

const int BETWEEN_JUMP_CHECK_TIME = 500; // time
between checking that potentiometer is moving in ms



//-----------------------------------------------------------------------------//



//Initialization for the soil probe ohmeter---------------------------------
--//

const int SOIL_PIN = 0;        // selects the input pin for the soil
probe measurement

const int V_IN = 5;            // voltage across the circuit

const float SOIL_REF = 1000000; // 1M Ohm reference resistor



int soilRaw = 0;               // the raw recorded value, between 0-1023
for 0-5V

float soilVoltage = 0;         // stores the converted voltage value

float resistanceMeas = 0;      // stores the measured resistance

//-----------------------------------------------------------------------------//



//Initialization for the soil probe potentiometer--------------------------
---//

const int POTENTIOMETER_INPUT_PIN = 2; // selects the input
pin for the potentiometer

const int DISTANCE_PER_ROTATION = 1;   // converts the
roation to position in inches

                                       // each jump gives one distance per rotation

const int POTENTIOMETER_HIGH_PIN = 48; // pin for the HIGH
across the poteniometer



int potentiometerRaw = 0;             // stores the value coming from
the sensor



int potentiometerOld = -1000;         // value of last measured
potentiomter, starts at an impossible number



int potentiometerJumps = 0;           // number of rising edge
measurement jumps

//-----------------------------------------------------------------------------//



//Initialization for GPS------------------------------------------------------
//

#include <SoftwareSerial.h>



SoftwareSerial gpsSerial(10 , 11);   // connect the TX pin of the
GPS to PWM pin 10 of the Arduino



const int GPS_MEASUREMENT_TIME = 20000; // measures GPS
for 20 seconds before recording it

const int FORWARD_ORIENTATION_TIME = 10000; // drive
forward for 10 seconds before measuring again



const int SENTENCE_SIZE = 80;

char sentence[SENTENCE_SIZE];



char field[20];

char latitude[20];

char longitude[20];

char northsouth[20];

char eastwest[20];



int newFractionalGPSLatitude;  // these store the last four bits of the
GPS location

int newFractionalGPSLongitude;

int oldFractionalGPSLatitude;

int oldFractionalGPSLongitude;



const int BASE_STATION_FRACTIONAL_GPS_LATITUDE =
7003;  // last four digits of the base station return location

const int BASE_STATION_FRACTIONAL_GPS_LONGITUDE =
6479;

const int MEASUREMENT_FRACTIONAL_GPS_LATITUDE =
7138; // last four digits of the final destination

const int MEASUREMENT_FRACTIONAL_GPS_LONGITUDE
= 6259;
```

```
double deltaLatitude;  // these are used in calculating the GPS navigation

double deltaLongitude;

double angleInDegrees1;

double angleInDegrees2;

double turnAngle;

double distanceToDestination;



const double MS_PER_M_FORWARD_SPEED = 2000; // ms/meter speed of the motors

//---------------------------------------------------------------------------//



//Initialization for the drive motors-----------------------------------------//

const int RIGHT_DRIVE_FORWARD_PIN = 51;

const int RIGHT_DRIVE_REVERSE_PIN = 53;

const int LEFT_DRIVE_FORWARD_PIN = 47;

const int LEFT_DRIVE_REVERSE_PIN = 49;

//---------------------------------------------------------------------------//




char programMode = '1';               // start in mode 1



char nextProgramMode = 's';    // holds the wirelessly commanded mode to be switched to



//Initialization for wireless communication----------------------------------//

const unsigned long TIME_BETWEEN_COM_SEND = 5000; // how often in ms a signal is sent



byte wirelessByteRecieved;  // stores the byte recieved

int stringPosition = 0; // which part of the recieved string we are reading



char startByte[4] = "#$%";

char middleByte = '&';

char endByte[4] = "(!)";



unsigned long lastComSendTime = 0;     // time of last sent wireless communication

//---------------------------------------------------------------------------//



void setup()

{

  Serial.begin(9600);    // starts serial data

  Serial1.begin(9600);   // for the transmitter

  Serial2.begin(9600);   // for the reciever



  gpsSerial.begin(9600); // starts GPS communication



  pinMode(SOIL_MOTOR_DOWN_PIN, OUTPUT); // when HIGH moves soil probe motor down

  pinMode(SOIL_MOTOR_UP_PIN, OUTPUT); // when HIGH moves soil probe motor up

  pinMode(POTENTIOMETER_HIGH_PIN, OUTPUT);    // pin used for high voltage across potentionmeter

  digitalWrite(POTENTIOMETER_HIGH_PIN, HIGH); // pin is always set to HIGH



  pinMode(RIGHT_DRIVE_FORWARD_PIN, OUTPUT); // drive motor pins

  pinMode(RIGHT_DRIVE_REVERSE_PIN, OUTPUT);

  pinMode(LEFT_DRIVE_FORWARD_PIN, OUTPUT);

  pinMode(LEFT_DRIVE_REVERSE_PIN, OUTPUT);

}



void loop()

{

//Main Program---------------------------------------------------------------//

  // Communication from computer or base station------------------------------//

  if (Serial.available() > 0) {  // only when somthing is typed

    programMode = Serial.read(); // used to override the current mode

  }



  // the string to be recieved is "#$%[single number for mode](!)" , ex: #$%3(!)

  if (Serial2.available() > 0) {            // only when something is recieved



      wirelessByteRecieved = Serial2.read(); // read in the byte
```

```
if (wirelessByteRecieved == '#' && stringPosition == 0){
  stringPosition = 1;
}
else if (wirelessByteRecieved == '$' && stringPosition == 1) {
  stringPosition = 2;
}
else if (wirelessByteRecieved == '%' && stringPosition == 2) {
  stringPosition = 3;
}
else if (stringPosition == 3) {
  nextProgramMode = wirelessByteRecieved;
  stringPosition = 4;
}
else if (wirelessByteRecieved == '(' && stringPosition == 4) {
  stringPosition = 5;
}
else if (wirelessByteRecieved == '!' && stringPosition == 5) {
  stringPosition = 6;
}
else if (wirelessByteRecieved == ')' && stringPosition == 6) {
  programMode = nextProgramMode;
  stringPosition = 0;

  Serial.println("Wireless data recieved");
  Serial.print("Advancing to mode ");
  Serial.println(programMode);
      }
      else {
        Serial.print("Not matching byte recieved: ");
        Serial.println(wirelessByteRecieved);
        stringPosition = 0;
      }
    }

  //--------------------------------------------------------------------------//

  // Robot waits until activated by base station-----------------------------//
  if (programMode == '1') { // waits until activated by the base station////
    gpsMeasurementFunction(); // measure GPS while waiting
//    gpsCompleteFunction();
    // serial2 is reading in and waiting
  }

  //--------------------------------------------------------------------------//

  // GPS navigation to measurement location----------------------------------//
  else if (programMode == '2') { // go to the location specified
    Serial.println("GPS Navitation to Measurement Location");
    Serial.println("--------------------------------------------------");

    gpsCompleteFunction(); // finds the GPS location and prints it out

    oldFractionalGPSLatitude = atoi(&latitude[5]);  // uses only the last four digits of the GPS reading
    oldFractionalGPSLongitude = atoi(&longitude[6]);

//          // only use this for testing without GPS
//          oldFractionalGPSLatitude = 7003;  // first location
//          oldFractionalGPSLongitude = 6479; //
//          newFractionalGPSLatitude = 7003;  // second location
//          newFractionalGPSLongitude = 6429; //
//          ///-------------------------------------//

    // delta to the measurement site from current location
    deltaLatitude = MEASUREMENT_FRACTIONAL_GPS_LATITUDE - oldFractionalGPSLatitude;
    deltaLongitude = MEASUREMENT_FRACTIONAL_GPS_LONGITUDE - oldFractionalGPSLongitude;

    // from start position to measurement site
    distanceToDestination = sqrt(deltaLatitude * deltaLatitude + deltaLongitude * deltaLongitude); // distance from the current point to the final point

    Serial.print("  ");
    Serial.print(distanceToDestination * .07871);
    Serial.println(" meters to the measurement location");
```

```
    if (distanceToDestination > 32) {   // atleast 2.5 meters from final destination

      Serial.println("  Not within range of measurement location");

      Serial.println("--------------------------------------------------");

      driveForwardFunction(FORWARD_ORIENTATION_TIME); // drive forward for 10 seconds

      gpsCompleteFunction(); // finds the GPS location and prints it out

      newFractionalGPSLatitude = atoi(&latitude[5]);  // uses only the last four digits of the GPS reading

      newFractionalGPSLongitude = atoi(&longitude[6]);

//          // only use this for testing without GPS

//          oldFractionalGPSLatitude = 7003;  // first location

//          oldFractionalGPSLongitude = 6479; //

//          newFractionalGPSLatitude = 7003;  // second location

//          newFractionalGPSLongitude = 6429; //

//          ///-------------------------------------//

      deltaLatitude = newFractionalGPSLatitude - oldFractionalGPSLatitude;

      deltaLongitude = newFractionalGPSLongitude - oldFractionalGPSLongitude;

      Serial.println(oldFractionalGPSLatitude);

      Serial.println(oldFractionalGPSLongitude);

      Serial.println(newFractionalGPSLatitude);

      Serial.println(newFractionalGPSLongitude);

      Serial.println(deltaLatitude);

      Serial.println(deltaLongitude);

gpsTurnAngleFunction(MEASUREMENT_FRACTIONAL_GPS_LATITUDE, MEASUREMENT_FRACTIONAL_GPS_LONGITUDE);

      distanceToDestination = sqrt(deltaLatitude * deltaLatitude + deltaLongitude * deltaLongitude); // distance from the current point to the final point

      Serial.print("Distance to measurement location is ");

      Serial.print(distanceToDestination * .07871); // converts it to meters

      Serial.println(" meters");

      Serial.println("--------------------------------------------------");

      driveForwardFunction(distanceToDestination * .07871 * MS_PER_M_FORWARD_SPEED); // meters/sec speed of the robot

//              driveForwardFunction(1000); // for faster debugging

    }

    else {

      Serial.println("  Within range of measurement location");

      Serial.println("--------------------------------------------------");

      Serial.println("End of mode 2");

      Serial.println("--------------------------------------------------");

      programMode = '3';

      delay (2000);

    }

  }

  //--------------------------------------------------------------------------//

  // Soil Probe Measurement-----------------------------------------------------//

  else if (programMode == '3') { // measure the soil

//    soilMotorFunction(SOIL_MOTOR_UP_PIN, MOVE_SOIL_MOTOR_TIME); // moves probe to home

    soilMotorFunction(SOIL_MOTOR_DOWN_PIN, MOVE_SOIL_MOTOR_TIME); // insert probe

    Serial.println("5 sec delay before measurement");

    Serial.println("--------------------------------------------------");

    delay(5000);

    soilMeasureFunction(); // measure probe

    printSoilProbeFunction();

    soilMotorFunction(SOIL_MOTOR_UP_PIN, MOVE_SOIL_MOTOR_TIME); // remove probe

    programMode = '4'; // begins next mode
```

```
    Serial.println("End of mode 3");

    Serial.println("--------------------------------------------------");

  }

  //--------------------------------------------------------------------------//



  // Send soil probe measurement to base station-------------------------
-----//

  else if (programMode == '4') { // communicate results to the base
station




    if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results


      char rString[8];


      dtostrf((int)resistanceMeas, 8, 0, rString);


      Serial.println(resistanceMeas); //

      Serial.println(rString); //


      // to the base station

      Serial1.write(startByte);

      Serial1.write('5');      // tell base station to advance to next mode

      Serial1.write(middleByte);

      Serial1.write(rString); // send the soil resistance to the base
station

      Serial1.write(endByte);


      // verifies what was sent

      Serial.print("Sent to base station: ");

      Serial.print(startByte);

      Serial.print('5');

      Serial.print(middleByte);

      Serial.print(rString);

      Serial.println(endByte);


      lastComSendTime = millis();

    }

  }

  //--------------------------------------------------------------------------//



  // Wait until told to move to next stage----------------------------------
--//

  else if (programMode == '5') { // listen for next command, will be
to return home


    // this state is not reached because Mode 4 sends until it recieves
from the base

    // station a command to move to Mode 6


    // gpsMeasurementFunction(); // measure GPS while waiting


    // serial2 is reading in and waiting

  }

  //--------------------------------------------------------------------------//



  // GPS navigation to base station-----------------------------------------
--//

  else if (programMode == '6') { // go to the location specified,
return to base

    Serial.println("GPS Navitation to Base Station");

    Serial.println("--------------------------------------------------");


    gpsCompleteFunction(); // finds the GPS location and prints it out


    oldFractionalGPSLatitude = atoi(&latitude[5]);  // uses only the
last four digits of the GPS reading

    oldFractionalGPSLongitude = atoi(&longitude[6]);


//          // only use this for testing without GPS

//          oldFractionalGPSLatitude = 7003;  // first location

//          oldFractionalGPSLongitude = 6479; //

//          newFractionalGPSLatitude = 7053;  // second location

//          newFractionalGPSLongitude = 6479; //

//          ///-------------------------------------//


    // delta to the measurement site from current location

    deltaLatitude =
BASE_STATION_FRACTIONAL_GPS_LATITUDE -
oldFractionalGPSLatitude;

    deltaLongitude =
BASE_STATION_FRACTIONAL_GPS_LONGITUDE -
oldFractionalGPSLongitude;
```

```
  // from start position to measurement site

  distanceToDestination = sqrt(deltaLatitude * deltaLatitude +
deltaLongitude * deltaLongitude); // distance from the current point
to the final point

  Serial.print("  ");

  Serial.print(distanceToDestination * .07871);

  Serial.println(" meters to the measurement location");

  if (distanceToDestination > 32) {   // atleast 2.5 meters from final
destination

    Serial.println("  Not within range of base station");

    Serial.println("--------------------------------------------------");

    driveForwardFunction(FORWARD_ORIENTATION_TIME);
// drive forward for 10 seconds

    gpsCompleteFunction(); // finds the GPS location and prints it
out

    newFractionalGPSLatitude = atoi(&latitude[5]);  // uses only the
last four digits of the GPS reading

    newFractionalGPSLongitude = atoi(&longitude[6]);

//          // only use this for testing without GPS

//          oldFractionalGPSLatitude = 7003;  // first location

//          oldFractionalGPSLongitude = 6479; //

//          newFractionalGPSLatitude = 7053;  // second location

//          newFractionalGPSLongitude = 6479; //

//          ///-------------------------------------//

    deltaLatitude = newFractionalGPSLatitude -
oldFractionalGPSLatitude;

    deltaLongitude = newFractionalGPSLongitude -
oldFractionalGPSLongitude;

gpsTurnAngleFunction(BASE_STATION_FRACTIONAL_GPS_L
ATITUDE,
BASE_STATION_FRACTIONAL_GPS_LONGITUDE);

    distanceToDestination = sqrt(deltaLatitude * deltaLatitude +
deltaLongitude * deltaLongitude); // distance from the current point
to the final point

    Serial.print("Distance to final location is ");

    Serial.print(distanceToDestination * .07871); // converts it to
meters

    Serial.println(" meters");

    Serial.println("--------------------------------------------------");

    driveForwardFunction(distanceToDestination * .07871 *
MS_PER_M_FORWARD_SPEED); // meters/sec speed of the
robot

  }

  else {

    Serial.println("  Within range of base station");

    Serial.println("--------------------------------------------------");

    Serial.println("End of mode 6");

    Serial.println("--------------------------------------------------");

    programMode = '7';

    delay (2000);

  }

 }

 //--------------------------------------------------------------------------//

 //--------------------------------------------------------------------------//

 else if (programMode == '7') { // communicate the program is
finished

  if (lastComSendTime + TIME_BETWEEN_COM_SEND <
millis()) { // check if it is time to send results

    // to the base station

    Serial1.write(startByte);

    Serial1.write('8');      // tell base station to advance to next mode

    Serial1.write(middleByte);

    Serial1.write("000");    // do not send the soil resistance to the
base station

    Serial1.write(endByte);

    // verifies what was sent

    Serial.print("Sent to base station: ");

    Serial.print(startByte);

    Serial.print('8');

    Serial.print(middleByte);

    Serial.print("000");

    Serial.println(endByte);
```

```
    lastComSendTime = millis();

  }

}

//----------------------------------------------------------------------------//


//These modes are for user testing----------------------------------//

else if (programMode == 'f') { // for manual driving of the robot

  driveForwardFunction(2000);  // forward for 1 second

  programMode = 's';           // stop mode

}


else if (programMode == 'v') { // for manual driving of the robot

  driveReverseFunction(2000);  // reverse for 1 second

  programMode = 's';           // stop mode

}


else if (programMode == 'l') { // for manual driving of the robot

  driveLeftFunction(2000);     // left for 1 second

  programMode = 's';           // stop mode

}


else if (programMode == 'r') { // for manual driving of the robot

  driveRightFunction(2000);    // right for 1 second

  programMode = 's';           // stop mode

}

  else if (programMode == 's') {  // stop mode does nothing

  }



}


//Functions-------------------------------------------------------------------//


  //Soil probe ohmeter-------------------------------------------------------
//

  void soilMeasureFunction() {

    Serial.println("soilMeasureFunction");


    soilRaw = analogRead(SOIL_PIN);         // Reads the Input PIN

    soilVoltage = (5.0 / 1023.0) * soilRaw; // Calculates the Voltage
on the Input PIN

    int f = (V_IN / soilVoltage) - 1;

    resistanceMeas = SOIL_REF / f;          // Recorded soil resistance


    Serial.println("--------------------------------------------------");

  }

  //--------------------------------------------------------------------------//


  //Soil probe motor movement---------------------------------------------
----//

  void soilMotorFunction(int motorPin, int motorMoveTime) {

    Serial.println("soilMotorFunction");

    boolean motorHalted = false;

    digitalWrite(motorPin, HIGH);  // turn motor on

    Serial.print("  Motor Pin: ");

    Serial.print(motorPin);

    Serial.println(" HIGH");


    float startTime = millis();    // runing clock time


    float lastJumpCheckTime = millis(); // check number of jumps


    int previousJumps = -1; // initialize number of jumps at
unreachable number to start while loop


    while (startTime + motorMoveTime > millis() && !motorHalted)
{

      // moving the motor for an amount of time and jumps are
occuring

      soilPotentiometerFunction(); // check if potentiometer jumps are
occuring


      if (millis() > lastJumpCheckTime +
BETWEEN_JUMP_CHECK_TIME) {

      // check every 500ms

        lastJumpCheckTime = millis(); // update last time jumps were
checked


/*        Serial.print(previousJumps);  // uncomment to check jumps
being read

          Serial.print("   ");

          Serial.println(potentiometerJumps); */
```

```
    if (previousJumps == potentiometerJumps) {

      motorHalted = true; // motor has not moved since last
checked

                        // shut off motor early

      Serial.println("  Motor not moving, turning off motor");

    }

    previousJumps = potentiometerJumps; // update number of
jumps so far



   }

  }

  digitalWrite(motorPin, LOW);  // turn motor off

  Serial.print("  Motor Pin: ");

  Serial.print(motorPin);

  Serial.println(" LOW");



  Serial.println("--------------------------------------------------");



  delay(1000); // to protect the battery from shorting

 }

 //Soil probe motor potentiometer-----------------------------------------
---//

 void soilPotentiometerFunction() {

  potentiometerRaw =
analogRead(POTENTIOMETER_INPUT_PIN);          // read the
value from the sensor



  delay(1); // analogRead takes 100us to read a value;

    if (potentiometerOld == 0 && potentiometerRaw > 0) { // check
if potentiometer is rotating

      potentiometerJumps += 1;                              // increase
jumps by one

    }

//    Serial.print(potentiometerOld);

//    Serial.print("   ");

//    Serial.println(potentiometerRaw);



    potentiometerOld = potentiometerRaw;                    // sets the
last potentiometer position



  }

  //--------------------------------------------------------------------------//



  //Puts together all of the other GPS components-----------------------
------//

  void gpsCompleteFunction () {

    float gpsTime = millis();



    Serial.print("  Measuring GPS for ");

    Serial.print(GPS_MEASUREMENT_TIME);

    Serial.println(" ms");



    while (gpsTime + GPS_MEASUREMENT_TIME > millis() ) { //
reads GPS for the measurement time

      gpsMeasurementFunction();

    }

    printGPSFunction();

  }

  //--------------------------------------------------------------------------//



  //GPS position measurement---------------------------------------------
----//

  void gpsMeasurementFunction ( ) {

    static int i = 0;

    if (gpsSerial.available()) {

      char ch = gpsSerial.read( );

//      Serial.print(ch); // use to read everything the GPS is sending

      if (ch != '\n' && i < SENTENCE_SIZE) {

        sentence[i] = ch;

        i++;

      }

      else {

       sentence[i] = '\0';

       i = 0;

       getGPSStringFuction();

      }

    }

  }

  //--------------------------------------------------------------------------//



  //Finds the GPS data string we are using-------------------------------
----//

  void getGPSStringFuction( ) {           // changed this program from
$GPRMC
```

```
    getGPSFieldFunction(field , 0);        // calls the function below
this

    if (strcmp(field, "$GPGGA") == 0) {

      getGPSFieldFunction(latitude, 2);  // number

      getGPSFieldFunction(northsouth , 3);   // N/S

      getGPSFieldFunction(longitude, 4);    // number

      getGPSFieldFunction(eastwest, 5);  // E/W

    }

  }

  //---------------------------------------------------------------------------//



  //From the GPS data string, reads the individual fields----------------
-----//

  void getGPSFieldFunction(char* buffer, int index) {

    int sentencePos = 0;

    int fieldPos = 0;

    int commaCount = 0;

    while (sentencePos < SENTENCE_SIZE) {

      if (sentence[sentencePos] == ',') // if between fields

      {

        commaCount ++;

        sentencePos ++;

      }

      if (commaCount == index) { // if we found the wanted field

        buffer[fieldPos] = sentence[sentencePos];

        fieldPos ++;

      }

      sentencePos ++;

    }

    buffer[fieldPos] = '\0';

  }

  //---------------------------------------------------------------------------//



  //Finds the angle the robot needs to turn to face the final
destination-----//

  void gpsTurnAngleFunction(double lat, double lon) {

 //   deltaLatitude = newFractionalGPSLatitude -
oldFractionalGPSLatitude;

//    deltaLongitude = newFractionalGPSLongitude -
oldFractionalGPSLongitude;



    angleInDegrees1 = atan2(deltaLongitude, deltaLatitude) * 180 /
PI;



    deltaLatitude = lat - newFractionalGPSLatitude;

    deltaLongitude = lon - newFractionalGPSLongitude;



    angleInDegrees2 = atan2(deltaLongitude, deltaLatitude) * 180 /
PI;



    turnAngle = - angleInDegrees1 + angleInDegrees2;



    // this is to keep the robot from rotating more than 180 degrees

    if (turnAngle > 180) {

      turnAngle -= 360;

    }

    else if (turnAngle < -180) {

      turnAngle += 360;

    }



    if (turnAngle < 0) {

      Serial.print("  Turn right ");

      Serial.print(abs(turnAngle));

      Serial.println(" degrees");

      Serial.println("--------------------------------------------------");

      driveRightFunction(abs(turnAngle) * 30);

    }

    else {

      Serial.print("Turn left ");

      Serial.print(abs(turnAngle));

      Serial.println(" degrees");

      Serial.println("--------------------------------------------------");

      driveLeftFunction(abs(turnAngle) * 30);

    }

  }

  //---------------------------------------------------------------------------//



  //Drive the robot forward for the input time---------------------------
----//

  void driveForwardFunction(double time) {

    Serial.print("Driving forward for ");

    Serial.print(time);

    Serial.println(" ms");
```

```
   digitalWrite(RIGHT_DRIVE_FORWARD_PIN, HIGH); // turn both motors ON to drive forward

   digitalWrite(LEFT_DRIVE_FORWARD_PIN, HIGH);

   delay(time);

   digitalWrite(RIGHT_DRIVE_FORWARD_PIN, LOW); // turn both motors OFF to stop moving

   digitalWrite(LEFT_DRIVE_FORWARD_PIN, LOW);

   Serial.println("  Done driving");

   Serial.println("--------------------------------------------------");

   delay(1000); // to protect the battery from shorting

  }

  //--------------------------------------------------------------------------//



  //Drive the robot reverse for the input time---------------------------------//

  void driveReverseFunction(double time) {

   Serial.print("Driving reverse for ");

   Serial.print(time);

   Serial.println(" ms");

   digitalWrite(RIGHT_DRIVE_REVERSE_PIN, HIGH); // turn both motors ON to drive reverse

   digitalWrite(LEFT_DRIVE_REVERSE_PIN, HIGH);

   delay(time);

   digitalWrite(RIGHT_DRIVE_REVERSE_PIN, LOW); // turn both motors OFF to stop moving

   digitalWrite(LEFT_DRIVE_REVERSE_PIN, LOW);

   Serial.println("  Done driving");

   Serial.println("--------------------------------------------------");

   delay(1000); // to protect the battery from shorting

  }

  //--------------------------------------------------------------------------//



  //Drive the robot left for the input time----------------------------------//

  void driveLeftFunction(double time) {

   Serial.print("Turning left for ");

   Serial.print(time);

   Serial.println(" ms");

   digitalWrite(RIGHT_DRIVE_REVERSE_PIN, HIGH); // turn both motors ON to turn left

   digitalWrite(LEFT_DRIVE_FORWARD_PIN, HIGH);

   delay(time);

   digitalWrite(RIGHT_DRIVE_REVERSE_PIN, LOW); // turn both motors OFF to stop moving

   digitalWrite(LEFT_DRIVE_FORWARD_PIN, LOW);

   Serial.println("  Done turning");

   Serial.println("--------------------------------------------------");

   delay(1000); // to protect the battery from shorting

  }

  //--------------------------------------------------------------------------//



  //Drive the robot right for the input time--------------------------------//

  void driveRightFunction(double time) {

   Serial.print("Turning right for ");

   Serial.print(time);

   Serial.println(" ms");

   digitalWrite(RIGHT_DRIVE_FORWARD_PIN, HIGH); // turn both motors ON to turn right

   digitalWrite(LEFT_DRIVE_REVERSE_PIN, HIGH);

   delay(time);

   digitalWrite(RIGHT_DRIVE_FORWARD_PIN, LOW); // turn both motors OFF to stop moving

   digitalWrite(LEFT_DRIVE_REVERSE_PIN, LOW);

   Serial.println("  Done turning");

   Serial.println("--------------------------------------------------");

   delay(1000); // to protect the battery from shorting

  }

  //--------------------------------------------------------------------------//



  //Displays the measurements for the soil probe------------------------------//

  void printSoilProbeFunction() {

   Serial.println("printSoilProbeFunction");

   if (resistanceMeas > 1000000) {               // greater than 1 M Ohm, print in M Ohms

    Serial.print("  Voltage: ");

    Serial.println(soilVoltage);             // outputs voltage measured

    Serial.print("  R_Meas: ");

    Serial.print(resistanceMeas/1000000);       // outputs resistance in M Ohms

    Serial.println(" M Ohm");

    Serial.println("--------------------------------------------------");

   }

   else {                          // less than 1 M Ohm, print in k Ohms

    Serial.print("  Voltage: ");
```

```
    Serial.print(soilVoltage);            // outputs voltage measured

    Serial.println(" V");

    Serial.print("  Resistance Measured: ");

    Serial.print(resistanceMeas/1000);          // outputs resistance in
k Ohms

    Serial.println(" k Ohm");

    Serial.println("--------------------------------------------------");

   }

  }

  //--------------------------------------------------------------------------//


  //Displays the measurements for the potentiometer--------------------
-------//

  void printPotentiometerFunction() {

    Serial.println("Soil Probe Motor Potentiometer");

    Serial.print("  Number of Jumps: ");

    Serial.println(potentiometerJumps);

    Serial.print("  Current Position: ");

    Serial.print("not known yet");                        // outputs location
of slide

    Serial.println(" inches");

    Serial.println("--------------------------------------------------");

  }

  //--------------------------------------------------------------------------//

  //Displays the GPS measurements----------------------------------------
-----//

  void printGPSFunction() {

    Serial.print("  Latitude: ");

    Serial.print(latitude);

    Serial.println(northsouth);



    Serial.print("  Longitude: ");

    Serial.print(longitude);

    Serial.println(eastwest);

    Serial.println("--------------------------------------------------");

  }

  //--------------------------------------------------------------------------//
```

Figure D.2 Code for Robot Sprinkler Arduino Mega 2560

```
//---------------------------------------------------------------------------//

//          Smart Sprinkler Robot System - Sprinkler Robot
//

//          ECE 445 Senior Design                              //

//          University of Illinois at Urbana/Champaign                  //

//          Denis Kurtovic, Kevin Johnson, and Jose L Orozco Jr.
//

//---------------------------------------------------------------------------//



//Useful funciton for string parsing from here

//http://arduino.cc/forum/index.php/topic,39023.0.html

//Using the weather information from this site

//http://forecast.weather.gov/MapClick.php?w3u=1&w11u=1&w12
u=1&pqpfhr=24&w14=pqpf0&w15=pqpf1&w16=pqpf2&w17=pqp
f3&psnwhr=6&AheadHour=0&Submit=Submit&FcstType=digital
&textField1=40.11300&textField2=-
88.26490&site=all&unit=0&dd=0&bw=0&FcstType=digitalDWM
L



#include <SPI.h>

#include <string.h>

#include <Ethernet.h>



// Define Constants

#define MAX_STRING_LEN  100



// Setup vars

char tagStr[MAX_STRING_LEN] = "";

char dataStr[MAX_STRING_LEN] = "";

char tmpStr[MAX_STRING_LEN] = "";

char endTag[3] = {'<', '/', '\0'};

int len;



// Flags to differentiate XML tags from document elements (ie.
data)

boolean tagFlag = false;

boolean dataFlag = false;



// This records if the weather data has been collected yet

boolean dataCollected = false;



// These variable hold the final weather data in them

const char *siteLocation; // Stores the name of the forecasted site

int valueNumber = 0;      // Used to keep track of values recorded

int prob010;      // stores the chance of 0.10" in 24 hours

int prob025;      // stores the chance of 0.25" in 24 hours

int prob050;      // stores the chance of 0.50" in 24 hours

int prob100;      // stores the chance of 1.00" in 24 hours



byte mac[] = {0x90, 0xA2, 0xDA, 0x0D, 0x76, 0xB5}; // MAC
address of our arduino ethernet



//IPAddress ip(192,168,1,1);

                                          // each arduino has its own MAC
address

EthernetClient client;                    // initialize the library

byte server[] = {204, 227, 127, 207};       // IP address for
forecast.weather.gov



unsigned long lastConnectionTime = 0;         // last time you
connected to the server, in milliseconds

boolean lastConnected = false;               // state of the connection
last time through the main loop

const unsigned long POSTING_INTERVAL = 1000;   // delay
between updates, in milliseconds



// starts up ethernet communication

void setup() {

  Serial.begin(9600);               // enable serial communication

  delay(1000);                      // delay for the ethernet modual to start

  Ethernet.begin(mac);              // starts ethernet communication

                                    // IP address is assigned automatically

  Serial.println("");

  Serial.print("IP address: ");

  Serial.println(Ethernet.localIP()); // prints out the IP address

}



void loop() {



  if(!client.connected()) {

    httpRequest(); // this starts the communication with
forecast.weather.gov

  }
```

```
  // this will print out information from the net connection when it is available

  while (client.available()) {

    serialEvent();              // this prints out the desired information

//      char c = client.read();    // uncomment this for debugging purposes

//      Serial.print(c);           // it will print out all recieved data from

                                   // the net connection

  }



  if (!client.connected() && lastConnected) { // check for ending of connection

    Serial.println("disconnecting.");

    Serial.println();

    client.stop();



    // this prints out the results

    Serial.println("---------------------------------------------");

    Serial.print(prob010);

    Serial.println(" % chance of 0.10\" of rain in 24 hours");

    Serial.print(prob025);

    Serial.println(" % chance of 0.25\" of rain in 24 hours");

    Serial.print(prob050);

    Serial.println(" % chance of 0.50\" of rain in 24 hours");

    Serial.print(prob100);

    Serial.println(" % chance of 1.00\" of rain in 24 hours");

    Serial.println("---------------------------------------------");

    if (prob025 <= 20) {

      Serial.println("Insufficient rain predicted");

      Serial.println("Initiate sprinkler robot to check soil moisture");

    }

    else {

      Serial.println("Sufficient rain predicted");

      Serial.println("Sprinkler robot to remain inactive");

    }

    Serial.println("---------------------------------------------");

    dataCollected = true;

  }



  lastConnected = client.connected(); // state of the connection



  if (dataCollected) {

    Serial.println("send \"r\" to repeat the forecast");

    boolean userWait = true;

    while (userWait == true) {

      char c = 0;

      if (Serial.available() > 0) {     // only when somthing is typed

        c = Serial.read();  // used to set the display mode

      }

      if (c == 'r') {

        userWait = false;       // break the while loop

        dataCollected = false;  // want to collect data again

      }

    }

    Serial.println();

    Serial.println("restarting forecast...");

  }

}

//-----------------------------------------------------------------------------//



// this connects to the weather server and stores the data into client

void httpRequest() {

  if (client.connect(server, 80)) { // if there's a successful connection to forecast.weather.gov

    Serial.println("connecting...");

    // this puts the weather forecast data into client which is later read with client.read

    client.println("GET /MapClick.php?w3u=1&w11u=1&w12u=1&pqpfhr=24&w14=pqpf0&w15=pqpf1&w16=pqpf2&w17=pqpf3&psnwhr=6&AheadHour=0&Submit=Submit&FcstType=digital&textField1=40.11300&textField2=-88.26490&site=all&unit=0&dd=0&bw=0&FcstType=digitalDWML HTTP/1.0");

    client.println();

    Serial.println("data collected");



    lastConnectionTime = millis();  // records when the last connection was made

  }

  else {

    // if you couldn't make a connection:

    Serial.println("connection failed");

    Serial.println("disconnecting.");
```

```
    client.stop();

  }

}

//-----------------------------------------------------------------------------//


// this parses the weather forecast information to record and display what is important

void serialEvent() {

  // Read a char

  char inChar = client.read();


  if (inChar == '<') {            // start of tag

    addChar(inChar, tmpStr);

    tagFlag = true;

    dataFlag = false;

  }

  else if (inChar == '>') {        // end of tag

    addChar(inChar, tmpStr);


    if (tagFlag) {

      strncpy(tagStr, tmpStr, strlen(tmpStr)+1);

    }


    clearStr(tmpStr); // Clears temporary string value


    tagFlag = false;

    dataFlag = true;


//      Serial.print(tagStr);

//      Serial.print("   ");

//      Serial.println(dataStr);


 // Find specific tags and print data

      if (matchTag("<data>")) {       // searches for the start of data

        valueNumber = 0;            // this varaible keeps track of the number of <values> encountered

      }


      if (matchTag("<value>")) {      // searches for: <value>

        valueNumber++;              // records which value we are looking at

        if(valueNumber == 2) {       // value for 0.25" in 24 hours

          int i = atoi(dataStr);

          prob025 = i;

//          Serial.println(prob025);

        }

        else if(valueNumber == 16) { // value for 0.50" in 24 hours

          int i = atoi(dataStr);

          prob050 = i;

//          Serial.println(prob050);

        }

        else if(valueNumber == 30) { // value for 1.00" in 24 hours

          int i = atoi(dataStr);

          prob100 = i;

//          Serial.println(prob100);

        }

        else if(valueNumber == 44) { // value for 0.10" in 24 hours

          int i = atoi(dataStr);

          prob010 = i;

//          Serial.println(prob010);

        }

        clearStr(dataStr); // clears the data between previous <value> </value>

      }

   }

   else if (inChar != 10) { // not at the end of the line

      if (tagFlag) {

        addChar(inChar, tmpStr); // add tag char to string

        if ( tagFlag && strcmp(tmpStr, endTag) == 0 ) { // check for </XML> end tag, ignore it

          clearStr(tmpStr);

          tagFlag = false;

          dataFlag = false;

        }

      }

      if (dataFlag) {

        addChar(inChar, dataStr); // add data char to string

      }

   }


   // If we are at the end of the line
```

```
  if (inChar == 10 ) {

    if (matchTag("<city state=\"IL\">")) { // searces for: <city
state="IL">

      siteLocation = dataStr;

      Serial.print("creating forecast for: ");

      Serial.println(siteLocation);

    }

    // Clear all strings

    clearStr(tmpStr);

    clearStr(tagStr);

    clearStr(dataStr);



    // Clear Flags

    tagFlag = false;

    dataFlag = false;

  }

}

//----------------------------------------------------------------------------//



////////////////////

// Other Functions //

////////////////////



// Function to clear a string

void clearStr (char* str) {

  int len = strlen(str);

  for (int c = 0; c < len; c++) {

    str[c] = 0;

  }

}





//Function to add a char to a string and check its length

void addChar (char ch, char* str) {

  char *tagMsg  = "<TRUNCATED_TAG>";

  char *dataMsg = "-TRUNCATED_DATA-";



  // Check the max size of the string to make sure it doesn't grow
too

  // big.  If string is beyond MAX_STRING_LEN assume it is
unimportant

  // and replace it with a warning message.

  if (strlen(str) > MAX_STRING_LEN - 2) {

    if (tagFlag) {

      clearStr(tagStr);

      strcpy(tagStr,tagMsg);

    }

    if (dataFlag) {

      clearStr(dataStr);

      strcpy(dataStr,dataMsg);

    }



    // Clear the temp buffer and flags to stop current processing

    clearStr(tmpStr);

    tagFlag = false;

    dataFlag = false;



  } else {

    // Add char to string

    str[strlen(str)] = ch;

  }

}



// Function to check the current tag for a specific string

boolean matchTag (char* searchTag) {

  if ( strcmp(tagStr, searchTag) == 0 ) {

    return true;

  } else {

    return false;

  }

}
```

Figure D.3 Code for Weather Forecast with Arduino Ethernet

```
if (lastComSendTime + TIME_BETWEEN_COM_SEND < millis()) { // check if it is time to send results

  char rString[8];

  dtostrf((int)resistanceMeas, 8, 0, rString);

  Serial.println(resistanceMeas); //
  Serial.println(rString); //

  // to the base station
  Serial1.write(startByte);
  Serial1.write('5');       // tell base station to advance to next mode
  Serial1.write(middleByte);
  Serial1.write(rString); // send the soil resistance to the base station
  Serial1.write(endByte);

  // verifies what was sent
  Serial.print("Sent to base station: ");
  Serial.print(startByte);
  Serial.print('5');
  Serial.print(middleByte);
  Serial.print(rString);
  Serial.println(endByte);

  lastComSendTime = millis();
}
```

Figure D.4 Portion of Code Used for Wireless Communication

APPENDIX E – Pictures

Figure E.1 is a screenshot of the 24-hour weather forecast that is generated by the Ethernet Arduino. Figure E.2 is a screenshot of the GPS navigation of the sprinkler robot. Figure E.3 is a screenshot of the base station communicating wirelessly with the sprinkler robot. Figure E.4 is a screenshot of the Arduino Mega reading in the GPS location. Figure E.5 is a screenshot of the sprinkler robot measuring the soil resistance.

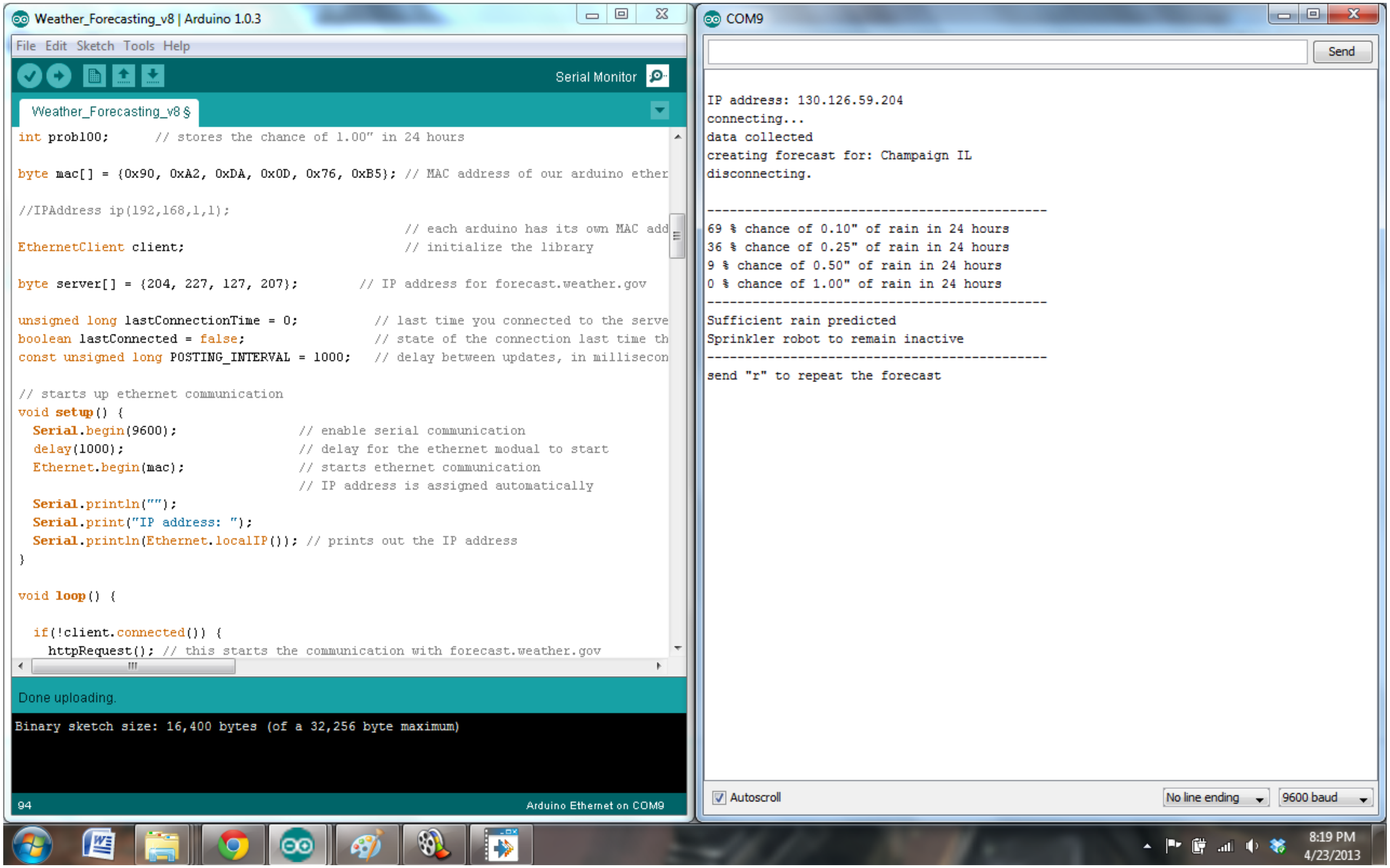


Figure E.1 Weather forecast generated by Ethernet Arduino

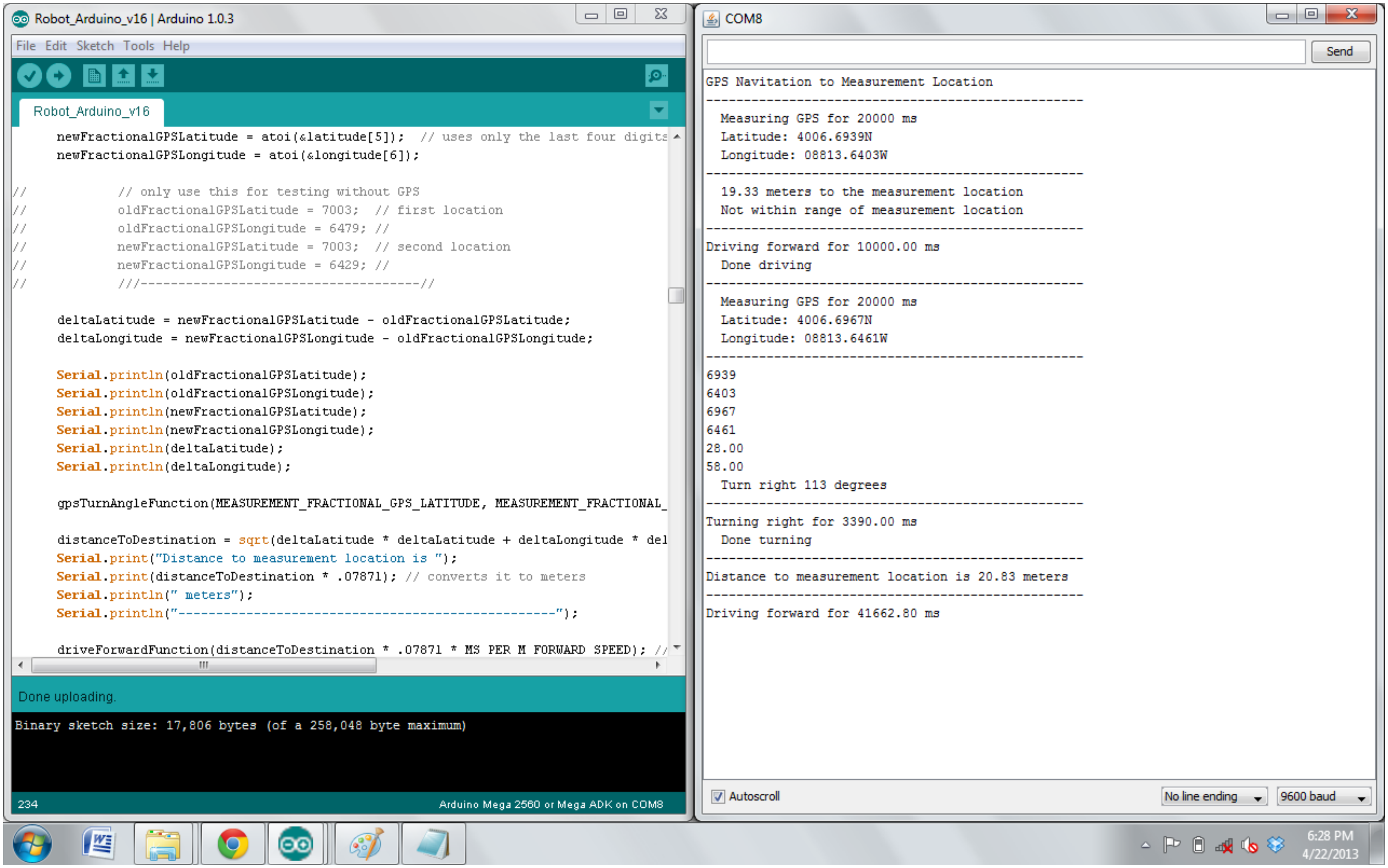


Figure E.2 GPS navigation of the sprinkler robot

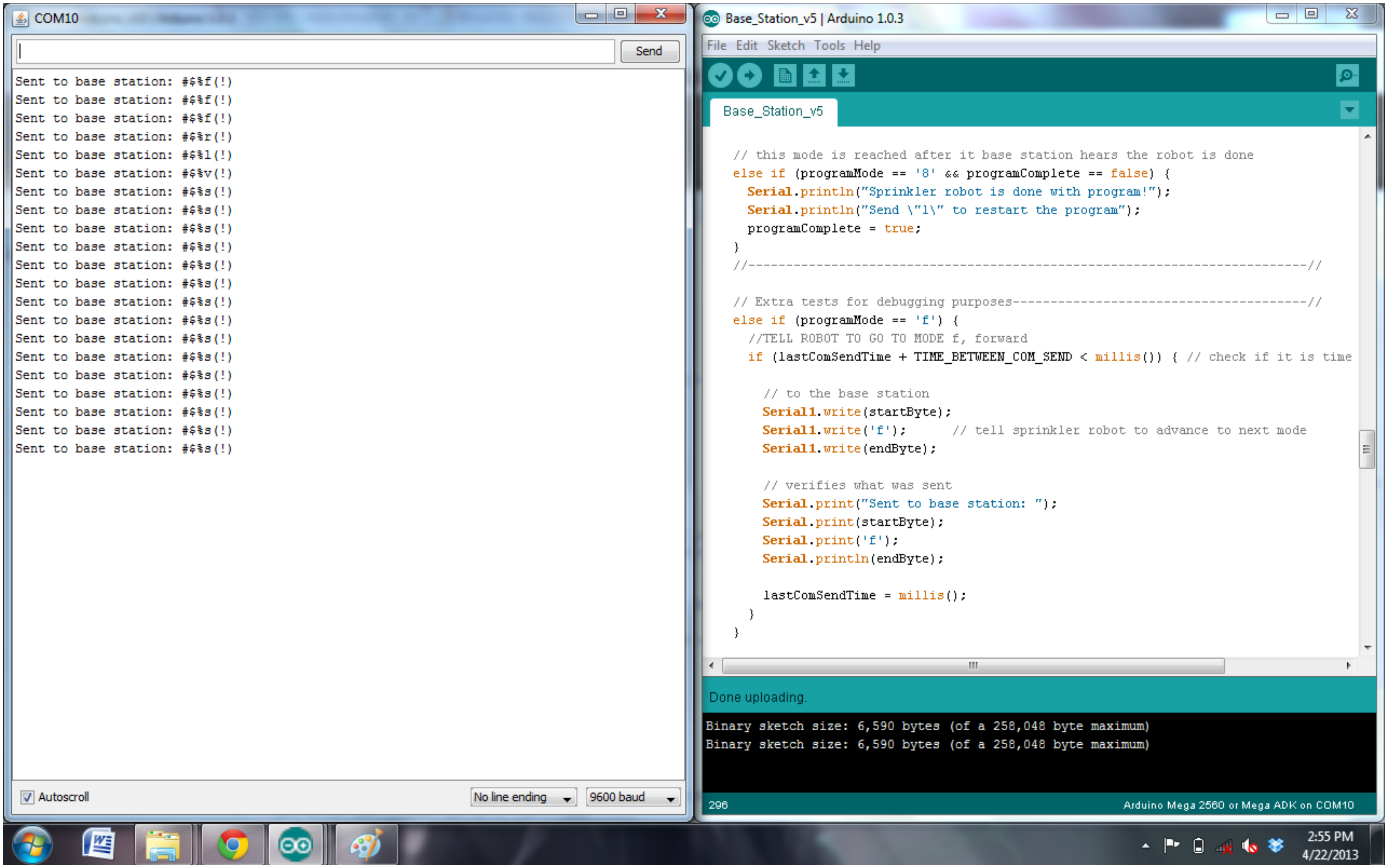


Figure E.3 Wireless communication to the sprinkler robot

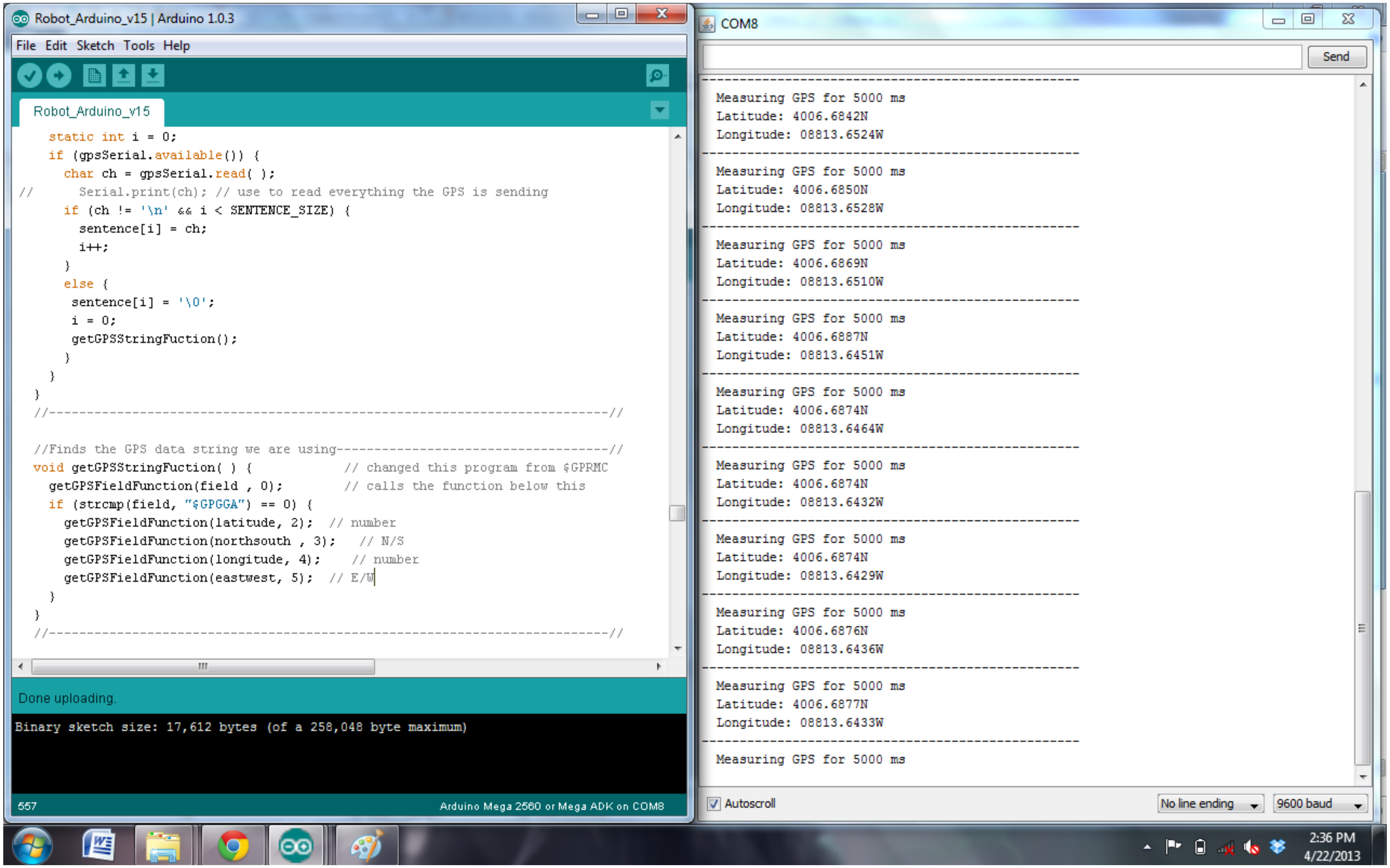


Figure E.4 Arduino Mega 2560 reading the GPS location

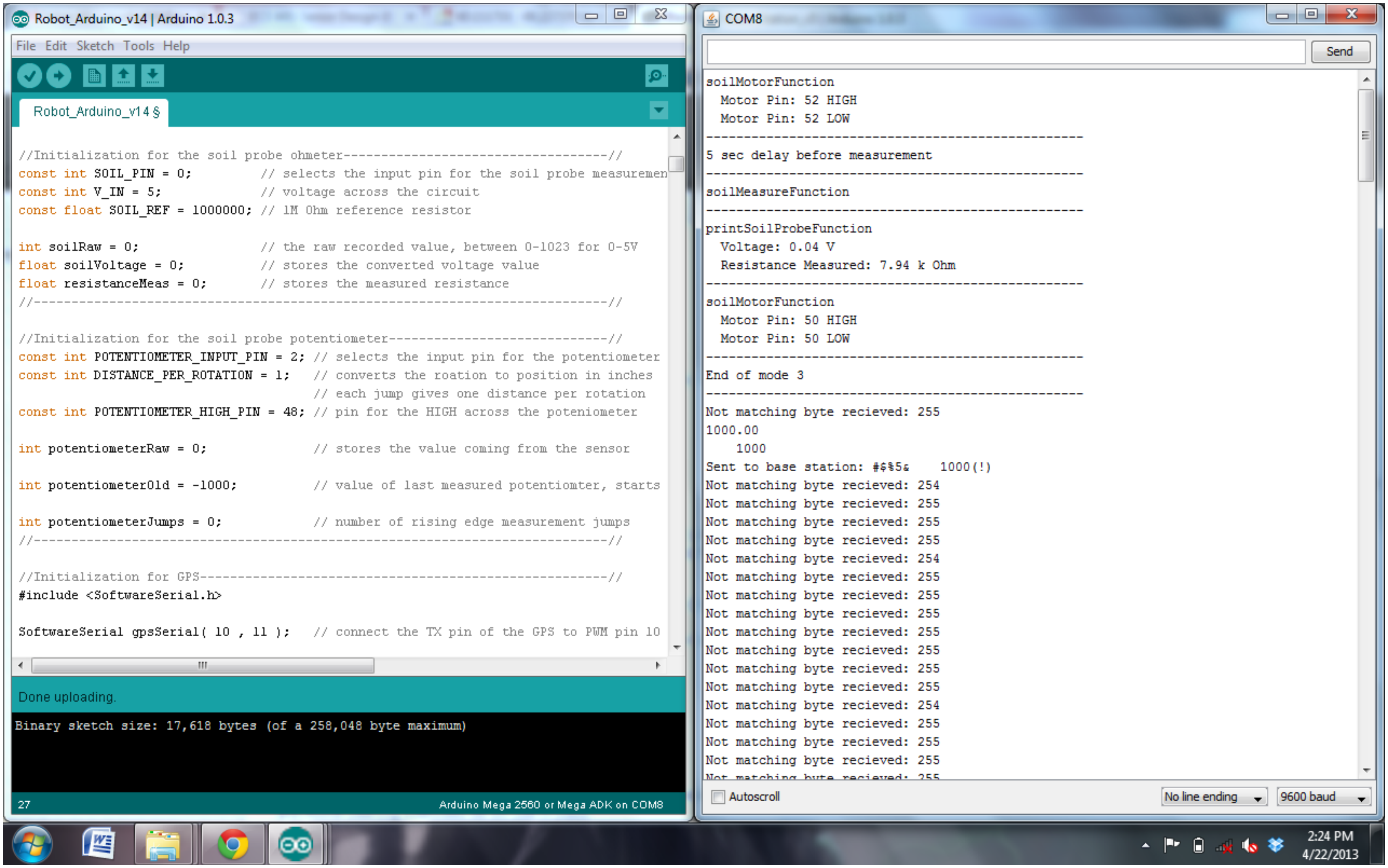


Figure E.5 Soil probe measurement

APPENDIX F – GPSNMEA REFERENCE

Start of Sequence	Payload Length (PL)	Payload		Checksum (CS)	End of Sequence
		Message ID	Message Body		
0xA0, 0xA1	Two Bytes	Message ID: 1 bytes; Payload up to 65535 bytes		1 byte	0x0D, 0x0A

Syntax of the NMEA message:

<0xA0, 0xA1><PL><Message ID><Message Body><CS><0x0D, 0x0A>

- Start of Sequence: Indicates the beginning of a message.
- Payload Length (PL): Is a 16 bit value that will tell you the length of the Payload.
- Payload: contains the Message ID and the Message Body. The Message ID can take the hexadecimal values between 0x01 to 0xFF, and the value will be cross referenced with the Message ID table so the meaning of the Message ID can be determined. For example: an output of 0xB3 is the Message ID that will give the WAAS status of the GPS receiver. The Message body will contain the message.
- Checksum (CS): Is the last message before the end of the sequence. It is an 8 bit exclusive OR to go along with only the payload bytes.
- End of Sequence: Indicates the end of the message.

APPENDIX G – MEASUREMENTS

Figure G.1 shows the percent error for known resistance values when measured by the soil probe. Figure G.2 shows soil measurements taken with wet soil. Figure G.3 shows soil resistance as a function of probe distances.

Resistor Measured	Measured Resistance	Error	Measured/ Reference
3.3kΩ	0kΩ	100.00%	0.0033
9.1kΩ	5.90kΩ	35.16%	0.0091
33kΩ	30.21kΩ	8.45%	0.0330
100kΩ	94.12kΩ	5.88%	0.1000
560kΩ	559.45kΩ	0.10%	0.5600
1MΩ	994.15kΩ	0.59%	1.0000
2.2MΩ	2.24MΩ	1.82%	2.2000
8.2MΩ	8.30MΩ	1.22%	8.2000

Figure G.1 Robot soil probe measurements taken using the Arduino Mega 2560

Probe Distance	Resistance (kΩ)
5cm	17.46
5cm	19.58
5cm	25.69
5cm	36.68
5cm	15.36

Figure G.2 Soil measurements taken with wet soil

Probe Distance (Inches)	Normalized Measurement	Resistance (MΩ)
0.5	41.0	8.20
0.5	48.6	9.72
0.5	44.1	8.82
0.5	49.0	9.80
0.5	43.5	8.70
1	43.4	8.68
1	51.4	10.28
1	38.9	7.78
1	53.7	10.74
1	57.7	11.54
1.5	69.4	13.88
1.5	75.5	15.10
1.5	74.6	14.92
1.5	69.1	13.82
1.5	73.0	14.60
2	88.6	17.72
2	91.1	18.22
2	77.2	15.44
2	74.0	14.80
2	80.0	16.00
2.5	72.7	14.54
2.5	79.0	15.80
2.5	78.5	15.70
2.5	77.6	15.52
2.5	73.0	14.60
3	83.8	16.76
3	80.3	16.06
3	81.8	16.36
3	79.2	15.84
3	78.2	15.64

Figure G.3 Soil measurements with varying probe distances with average soil moisture