Radio Frequency Identification Localization and Mapping

Millennium Engineering: Department 502

Date: 05/08/12

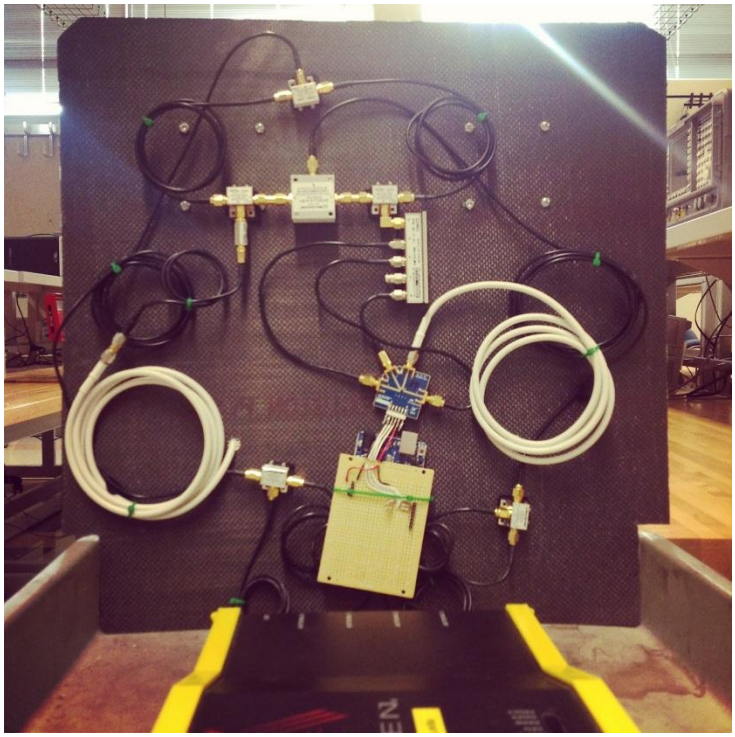
Team Lead: Danielle Fuller

Team Members: Matthew Hannon

Matthew Greenlee

Priyadarshini Mishra

Aaron Gillespie



Abstract:

The goal of this project was to design a system capable of detecting and localizing radio frequency identification (RFID) tags from up to 10 feet away and within 15 degrees of spatial accuracy. After assembly and testing, the RFID system was capable of locating tags at a maximum distance of 15 feet and was able to determine if object tags were more likely to the left, right, or directly in front of the system.

Introduction:

Radio frequency identification (RFID) systems are used to track inventory, identify lost animals, prevent shoplifting, and even help prevent counterfeit drugs from entering the market [1]. RFID systems use combination transmitter/receivers, usually called "readers", and antenna-like devices, usually called "tags". The reader emits electromagnetic energy which activates or "excites" the tag. The tag then broadcasts a unique identification number (or some other pre-programmed information) which the reader receives and interprets. In most applications, tags gather the energy required to perform function from the fields transmitted by the reader; in other words, without the need for an internal power supply. This makes them extremely low-cost and useful for high-volume applications, such as managing vast inventories. These systems are so useful that Walmart, the largest retailer in the world, has asked all their suppliers to add RFID tags to the pallets in their shipments [4]. Walmart and many other stores also use RFID to deter potential shoplifters. This is accomplished by affixing RFID tags to often-shoplifted items and placing banks of readers at each of the exits.

As a technology, RFID is still relatively young. New applications for it are being explored by many parties. One of these applications is the use of RFID systems for object localization and mapping. Current RFID localization systems are available, but usually fall into two different categories: the immobile and the expensive. The first category involves mounting several (usually four or more) antennas in known locations around a room. Each of these antennas is constantly searching for tags. When a tag is detected, the system is able to locate its approximate position using either relative received signal strengths or by other, more complex methods. These systems have been shown to be accurate to within one foot of the actual position of the object tag(s) [2]. There are portable implementations of similar systems; however such implementations often include very high levels of sophistication, such as intelligent automated robots which use RFID and other technologies to map an entire room [3]. With these high levels of sophistication comes increased unit cost.

This report details the design of a proof-of-concept RFID mapping and localization system. In accordance with the wishes of the customer, Honeywell FM&T in Kansas City, this system was designed to be relatively inexpensive and mobile. Ideally the system would be capable of locating tags up to 10 feet away from the detector, measuring the angle of incidence to within 15° of 15º of the actual value, and give an approximation for the radial distance to the detected tag. For the purpose of the project, ideal conditions were to be assumed; examples of such include no

multipath, no pre-existing systems to be designed around, and a clear line-of-sight between reader and tag.

Methodology:

Per the request of the customer, the team decided to make use of components off the shelf (COTS) rather than designing and building units in-house. This allowed more time for theoretical design while reducing the risk of manufacturing error and creating a more professional end-product.

Antennas

It was decided that a two-antenna array would be used for both transmission and reception. The HyperLink Wireless 900 MHz Flat Patch Antenna (HG908P) from L-Com was chosen due to its size, weight, linear polarization, and common applications with RFID.

Steering Network

To avoid manually moving the unit to pinpoint a tag, an electronic steering network was designed to direct the focus of the reader in three discrete directions: right, left, or center. The network was comprised of a SP3T RF switch (SKY13385-460LF), a 4-way, 0º power splitter (ZX10-4-11), and three custom-cut lengths of coaxial cable, as illustrated in Figure 1. By attaching this in-line with one of the antennas the transmitted/received signals could be phase-shifted by switching between wire lengths. This phase shift induces an interference pattern which "steers" the peak or null of the composite signal.

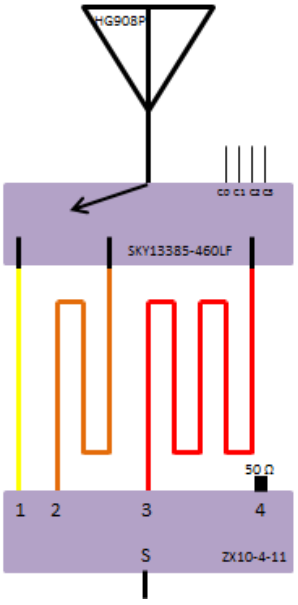


Figure 1: Steering network

Beam steering behaves in accordance with the following Equation 1:

$$\Delta\phi = 360° \times d \times \frac{\sin(\theta_s)}{\lambda} \quad \text{Eq. 1}$$

In Equation 1, Δϕ is defined as the phase shift between the antenna elements, *d* as the physical distance between the antennas, *Θs*as the steering angle (angle off perpendicular to the plane of the antennas), and λ as the wavelength on the coaxial cable. The antennas chosen for this project had a horizontal beam-width of 75º, so a steering angle of 18º was chosen to in order to most nearly bisect the left and right halves of the beam pattern. The antennas were 8.5" wide and touched at the edges, hence the physical distance between their centers was also 8.5". The system operated at frequencies around 900 MHz; which corresponds to an ideal wavelength of 13.12" in free space. The coaxial cable used, 174/U 50 *Ω* cable, is listed to have a propagation velocity of 0.66 times the speed of light. This reduced the effective wavelength to 8.66". Inserting these values in Equation 1, the phase difference required was determined to be 72.1º. The difference in physical coaxial line length was computed using the following Equation 2:

$$\ell = \lambda \times \frac{\Delta\phi}{360°} \quad \text{Eq. 2}$$

Inserting the effective wavelength and desired phase shift into Equation 2, the difference in line length was calculated as 1.735". This value was approximated as 1.75", which affected the steering angle by less than 0.2º. The medium length coaxial line, used to "steer" to the center, was the basis for determining the long and short lengths. The short wire, used to steer to the left, was ideally 1.75" shorter than the medium. The long wire, used to steer right, was ideally 1.75" longer.

The insertion loss for the switch and splitter were pulled off the data sheets and found to (typically) total 1.1 dB. A 1 dB attenuator was attached to the second antenna to mirror the loss of the switch and splitter on the first antenna. An equivalent coaxial cable connecting the second antenna to the system was cut to equal the medium line's electrical length using delay measurements taken with a network analyzer. The delay through the switching network was observed to be a difference of 2 ns between the middle length of the sum network, and the attenuator by itself. By relating the propagation velocity of the cable to the desired delay of 2 ns, a matching cable length was cut for the second antenna as well.

Sum/Delta Network

Since interference tracking was the goal of the antenna array, the scope was expanded by using a monopulse technique used in radar systems. Monopulse uses sum and delta antenna

configurations to create different interference patterns. This way, both the constructive and destructive interference could be used to track tags. Figure 2 shows this setup in more detail. The splitters for each antenna signal were implemented with a 2-way, 0º power splitter (ZX10-2-12), as was the summing network (the sum of the two signals will peak at the intersection point). For the delta, a 2-way, 180º power splitter was used, effectively subtracting one signal from the other and creating a null.

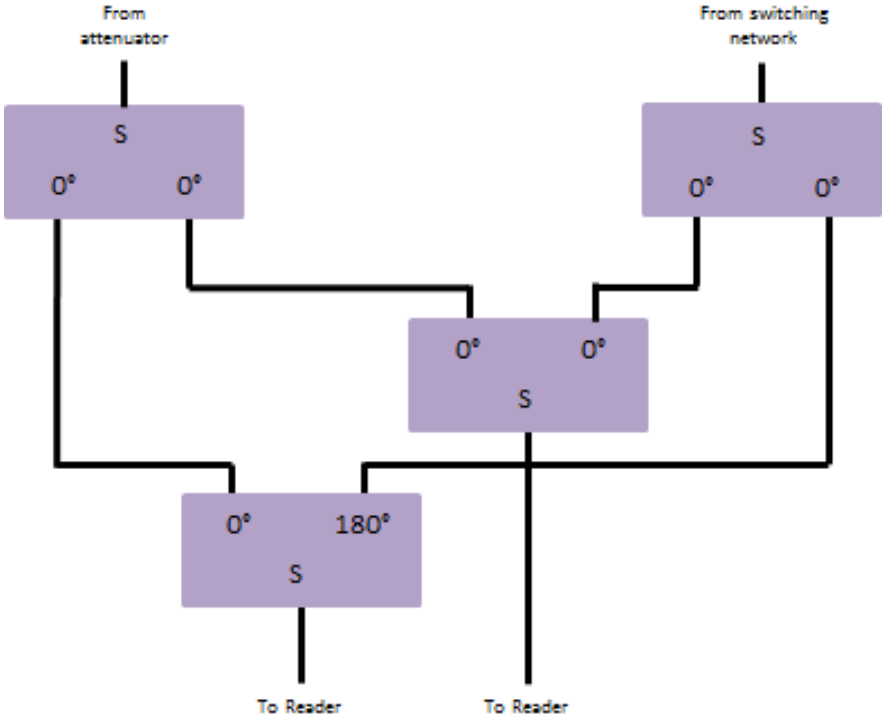


Figure 2: Sum/Delta Network

UHF Reader

To adjust for the absence of Honeywell support, a few changes had to be made. The biggest of these was the UHF reader, which ended up being an Alien 9800 loaned to the group by Dr. Daniel Deavours. The original design centered around the Impinj Speedway, chosen for its ability to read phase information and its monostatic ports--each only either transmitting or receiving. However, the Alien reader does not document phase and is multistatic with two pairs of ports: one pair transmitting and receiving while the other pair is inactive. Due to this change, directional couplers had to be added to the reader ports (ZX30-20-4), and RSSI became the primary data capture tool.

System Structure

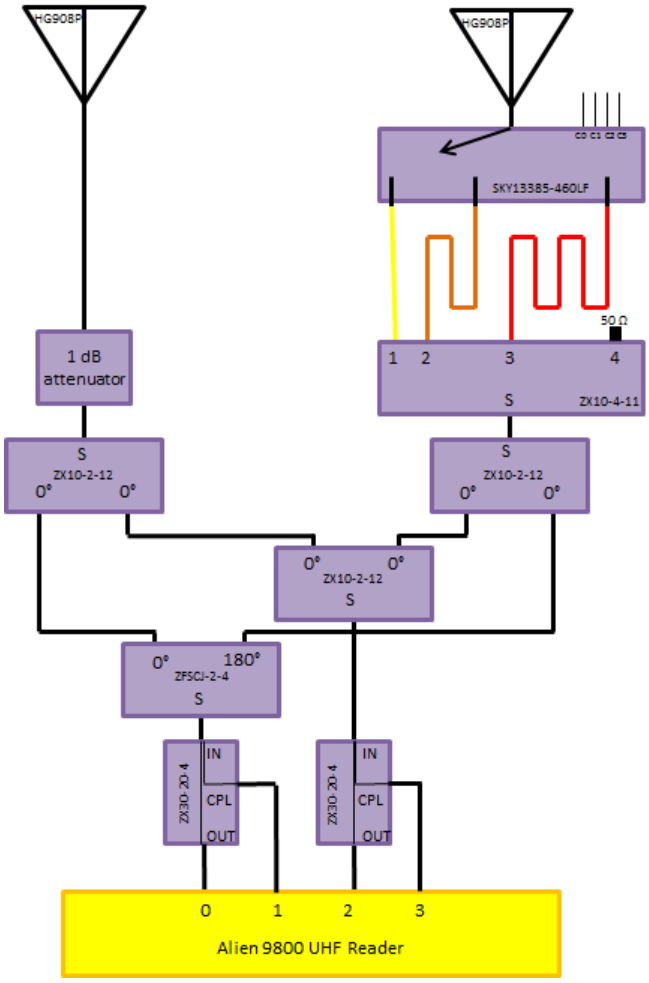


Figure 3: Complete RFID system

Software Design

The Alien 9800 offered a wide variety of development solutions and languages. The Alien RFID readers have publicly available software development kits (SDK) and firmware upgrades to optimize and expand reader functionality. The programming languages available were .Net, Visual Basic, and Java. Each language had source code available demonstrating basic setup and configuration to talk to the reader. The .Net source code offered the most expansive collection of sample applications illustrating the use of new features such as RSSI filter control, speed filter control, and access to per-antenna port RSSI information. For this reason as well as the simplicity in usage of .Net development, .Net was chosen as the language for this project.

Using different parts of the available sample applications, a foundation for the program quickly developed. A first step in this process was to figure out what routines the software needed to handle the demands of RFID localization. These features were determined as requirements: isolation of tags, RF switch control via the Arduino, TCP/IP (Ethernet) and serial input/output, tag RSSI automated sequencing, and distance approximation through RF power attenuation. A block diagram of the software design components is rendered in Figure 4. Please note that the required routines just discussed are inside of the Functionality block.

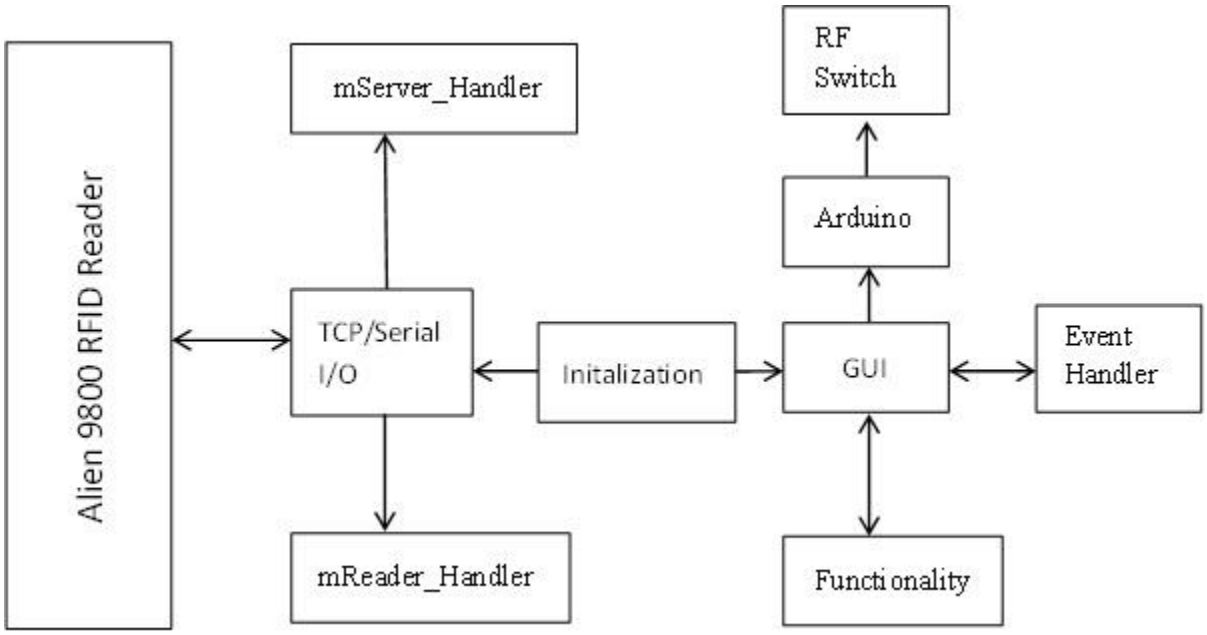


Figure 4: Block diagram of software system

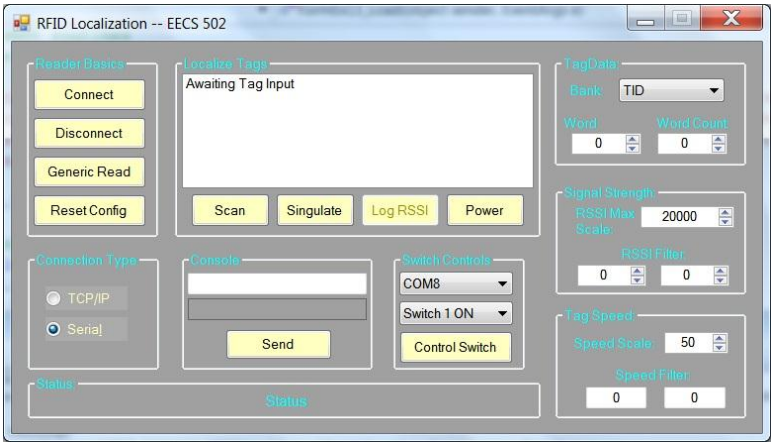


Figure 5: Custom-made GUI

Figure 4 shows how the program begins with the *Initialization* block, which is responsible for setting up connections with the operating system and touching up the user interface. The language of use, .Net, revolves around the user interface to trigger most events on which to act. These GUI and operating system events are handled in the Event Handler block. The connection to the Arduino was accomplished with openly available code responsible for sending and receiving information on the communications (COM) port. Once this capability was in place, it was then a matter of creating known code words that the Arduino code, as well as the local .Net program, understood. For example, when the Arduino sets the proper digital ports high and low to control the RF switch, it responds with a one character code word 's'. The program interprets this, and knows that it can now continue its tasks because the switch has been successfully changed. The program loops until a timeout period or until it receives the letter success code word. The mReader_Handler and mServer_Handler blocks which interface with the TCP/Serial block are responsible for processing all the Alien 9800 reader event messages. The reason there are two reader handler blocks is because of the complications with the Ethernet and serial connection. The final block in the software design is the Alien 9800 reader which communicates entirely through the TCP/Serial block which feeds this information back to the rest of the system.

Each routine mentioned above was developed to be effective over a wide range. For example, the RF switch control worked well largely due to a feedback mechanism which reported the state of the switch. An example of a less effective routine was the automation of RSSI and RF power attenuation. This routine never functioned correctly due to conflicts between the serial and TCP/IP I/O mechanism. The serial port connection was useful for monitoring the boot up sequence and pre-initialized reader configuration. The Ethernet connection was needed by some of the sample applications that were built around a multi-reader system. The aforementioned conflict occurred due to incompatibilities between the serial and Ethernet connections needed for the two different computers. This problem arose with the lab computers having available a serial connection but no Ethernet port. This restriction was due to required network security procedures, namely a computer with open administrative rights is a danger to a normally secured network when connected. This problem was solved by disabling the Ethernet port and enabling wireless communication. Stepping away from this issue with the automation routine, another more serious problem to the effectiveness of the automation was incoherent and inconsistent data. The data was gathered from a non-ideal environment which created a large amount of backscattering and other nonlinearities. There were also issues with the consistency of the Alien 9800's ability to read an isolated tag. The reader was observed to read a tag continuously for a period of one to two seconds; but would then refuse to read

the tag any further until either the tag was moved or the configuration was reset (sometimes to the extreme of rebooting the whole system). Possible sources to these problems could include incorrect reader configuration, lack of a computer with administrative access and both serial and Ethernet ports, and rushed software development.

Results:

Using the relative lengths calculated by Equation 2 the available coaxial cables were cut to length and tested. Due to inexperience cutting coaxial cables and human error, the actual lengths were not precisely equivalent to the ideal lengths. The medium length cable in the switching network was cut to 8.0". Based upon that, the short was cut to 6.375", 1.625" shorter than the medium length and 0.125" off the ideal length difference. This 0.125" changes the steering angle by only about 1.3º, and thus was deemed acceptable. The long cable was cut to 9.75", 1.75" longer than the medium length and equal to the ideal calculated length difference. As mentioned in the methodology of the switching network, to obtain a matched network on the second antenna side, a 1 dB attenuator and additional custom length of coaxial cable were required. Using the relation discussed above, an electrically matching length of 15.576" was found to be necessary. After cutting the cable, the actual value was found to be 15.75", 0.15" off the calculated value.

Each sub-system was tested individually to ensure all components of the full system were working correctly prior to connection. The sum, delta, and switching networks were all separately tested. Due to the nature of the sum and delta configurations, they were unable to be conclusively tested until the arrival of the RFID reader. Preliminary testing hooked up to the network analyzer yielded the anticipated results. The switching network was observed to have phase values as shown in Figure 6. The steering had a phase difference of 66.4º between short and medium, and 75.3º between medium and long. This was near enough to the ideal values of 72.1º of steering left and right, with the discrepancy being attributed to imperfections of crimped cable lengths.

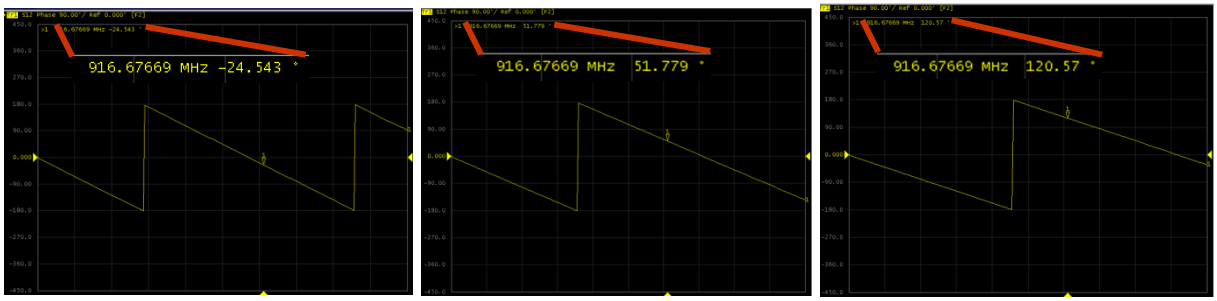


Figure 6: Long, medium and short length phase shifts

Under the more ideal conditions of Dr. Deavours' lab, the maximum range of the system was determined to be 15 feet. Tested in the Dunwoodie Laboratory in Eaton Hall, the maximum distance attained was 8 feet. Due to the setbacks incurred in this project, tags were not mapped using phase information obtained from the reader. Instead, a system was developed comparing relative RSSI at each switch position to determine position right, left, and center. Based on data observed during testing of the complete system, at each switch position a consistent range of RSSI values were observed—shown in Figure 7. Using these values, a system was developed to classify the comparison of each relative RSSI average to a tag location. This is described in Table 1.

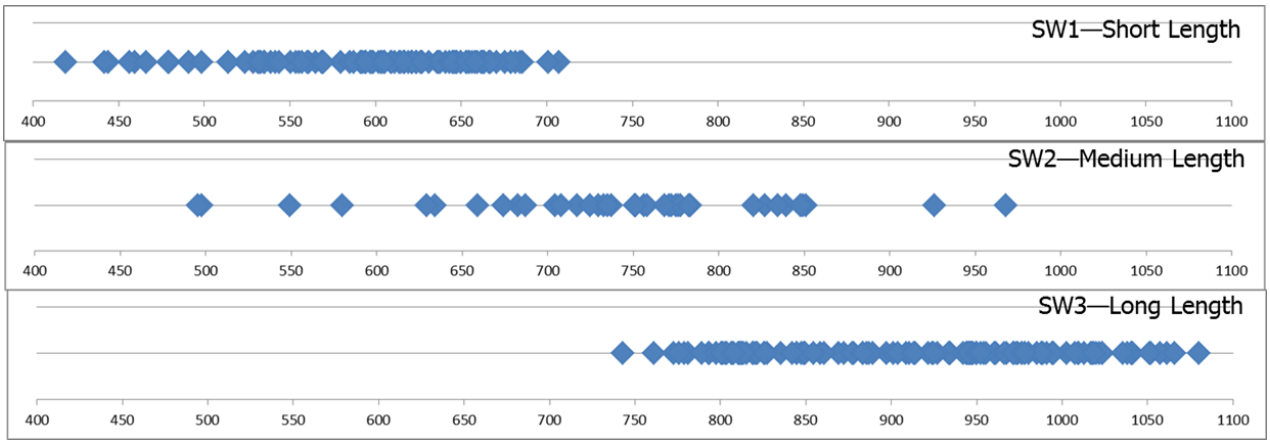


Figure 7: RSSI values with tag located left of center

Table 1: Relative RSSI to Find Location

Relative RSSI Values	Tag location
S1 < S2 < S3	Left
S3 < S2 < S1	Right
S1 < S3< S2	Center

Lessons Learned:

Throughout this project there were many lessons learned. The first and most important of these was to avoid putting critical aspects of any time-sensitive project into the hands of a third party, if at all possible. In this case, the customer agreed to provide an RF switch for the steering network and an RFID reader to build the system around. This shift of accountability resulted in weeks of development time spent waiting for a delivery that would never happen. This also stressed the importance of predicting risks and having a solid mitigation plan for any potential setbacks. The mitigation plan used when the reader was unable to be delivered relied on Dr. Deavours being able to lend an RFID reader to the project. Fortunately he was able to provide a reader one week before the project deadline; however, this reader functioned differently than what was initially chosen for the design. This forced last-minute design changes, additional costs, and sacrificed performance capabilities. The new reader required the use of $30 bidirectional couplers to perform the sum-and-delta networking, which induced more loss in the system. To bypass the missing switch, a $125 replacement was ordered--creating additional wait time for the delivery and less time to design and test the network.

Another important thing to remember when designing an array is to maintain equivalent electrical lengths following each antenna. In the case of this project, a very involved switching network was added to only one of the antennas, so without balancing the loss to the other side of the array, the non-switching signal would be stronger and more dominant than its counterpart. Since the sum and delta functions rely on signal strength comparisons, this uneven distribution of power would result in misleading readings.

The importance of subsection testing became obvious when the whole system was not functioning properly. Since both the switching network and sum/delta units had been individually tested and verified on the network analyzer, it was clear that communication problems between tags and the reader lay elsewhere in the system. This expedited the troubleshooting process, saving valuable time in an already brief testing period.

One last lesson learned was the importance of flexibility. By having a backup plan, along with strong morale among teammates, even the worst of scenarios can be overcome. When things do not go as planned, a flexible mindset allows adjustments to be made and new creative solutions to surface. This project is able to produce positive results and a proof-of-concept design due to the lack of resignation or stubbornness of team members, proving that incomplete functionality does not mean success was not achieved.

Recommendations:

The time constraint, limited budget, and lack of customer cooperation greatly limited the capacity of the project. Even one more week with all parts in hand would have allowed great strides to be made. The potential for expansion is quite large if more funding and time are allowed in the future. Below are a few of these potentials.

Shorter coaxial cables:

The cables used in this project were lent by Dr. Allen and only came in 2.5' or 5' lengths. Since the distance between any two elements was no more than one foot, these were excessive. To prevent the extra loss and mess that come with longer cables, it is recommended that professionally made, specific-length are used. The same goes for the three switching lengths, which were cut and crimped by hand.

Thicker coaxial cables:

The receive power path for RFID systems is normally not an issue, although the transmitter path can offer relatively high power which can result in large loss in the form of heat. This can cause damage to the RFID system if proper coaxial cable is not utilized. The ideal coaxial cable is a low loss line, causing less signal degradation. In future designs coaxial lines handling transmit power should be measured, and well within their maximum operating voltages.

Use of the Impinj:

If funding becomes available, the Impinj reader would be a valuable investment for the project. Phase comparison along with RSSI could provide more accurate location determination. The monostatic ports would eliminate the need for couplers, and the original design where both the sum and delta ports were simultaneously observed could be implemented.

Three-dimensional mapping:

Currently, only location in the x- and y- planes can be determined. If two more antennas were added that incorporated the same switching network, the location on the z-plane could be found. With a specific algorithm, this could provide a pinpoint location in three-dimensional space.

Portable implementation:

Use of the system is currently limited by the length of cables connecting the reader to the desktop. If a field-programmable gate array (FPGA) took the place of the computer, the unit

would be able to travel with ease. Another possibility would be adding a wireless communication network, allowing the received signals to be sent through air to the cable-connected reader. If modulated at a carefully chosen frequency far away from 900 MHz, there should be no interference between the patch and base station antennas. A further possibility for mobility would be to utilize the Alien 9800 network communications such as connecting to a pre-programmed website URL in the reader with an active domain name server (DNS) on a network for communicating data as well as receiving instruction. A networking switch and an IP address could also be used. A potential solution to controlling the RF switch via the network would be to use the onboard reader GPIO ports, properly conditioned, to talk to the Arduino. This setup would require only access to a power source and a network wall connection for the reader to operate.

References:

[1] "Medical Uses for RFID Products", VizinexRFID. Accessed on 5/1/2012, Available at [http://www.vizinexrfid.com/medical-uses-for-rfid-products/581/]

[2] "Walmart issues RFID Mandate" by P. Mah, TechRepublic, Accessed on 5/1/2012, Available at [http://www.techrepublic.com/blog/tech-news/wal-mart-issues-rfid-mandate/1946]

[3] "Object Localization Using RFID" by K. Chawla, G. Robins, and L. Zhang, Accessed on 5/1/2012, Available at [http://www.cs.virginia.edu/~robins/papers/rfidloc_iswpc2010.pdf]

[4] "Mapping and Localization with RFID Technology" by M. Philipose, K. Fishkin, D. Fox, D. Hannel, W. Burgard, Intel Research Seattle, University of Washington, and University of Freiberg, (12/2003) Accessed on 5/1/2012, Available at [http://edge.rit.edu/content/P12015/public/Referenca%20Materials/RFID%20READER%20SKYTEK%20M9%20Module/Mapping%20and%20Localization%20with%20RFIDs.pdf]

Appendix A: Source Code

1. RSSI Logging Automation
2. Alien Reader Event Handler
3. Arduino and Switch Control Function

1. RSSI Logging Automation

```
//Algorithm for outputting RSSI information to a file in an automated fashion.
//It works by setting the switch to 1, reading tags for a specified amount of time, switching to switch 2, and repeating for all switches.
//Next it turns on the alternative TX/RX pair to obtain RSSI information from the delta path.
//This process is then repeated.

private void Log_RSSI_Click(object sender, EventArgs e)
{
   try
   {
      //This is where basic RSSI info will be logged from sum and delta ports
      int WaitTime = 15000;
      Log_Type = 1;

      //Goals
      //Isolate tag
      //DONE.. enabled by mask button..

      lblStatus.Text = "RSSI Logging starting...";
      AlienLog.WriteLine(true, "-----RSSI Data Log Beginning-----");
      AlienLog.WriteLine(true, "TAGID" + " " + "RSSI" + " " + "Speed" + " " + "TX_Ant" + " " + "RX_Ant" + " " + "Switch");
      AlienLog.WriteLine(true, "--------------------------------------------------------------------------------------------");
      //Set switch 1
      IDC_SWITCH_DROP.SelectedIndex = 0;
      IDC_SWITCH_BUTTON.PerformClick();

      //Pause for micro to confirm switch has been changed
      while (!OP_Complete)
      {
         System.Threading.Thread.Sleep(500);
      }
      OP_Complete = false;

      //Probe it on tx1, and log rssi
      reader.AutoMode = "OFF";
      reader.ClearTagList();

      //Set antenna sequence to tx/rx pair on 0,1
      reader.AntennaSequence = "0,1";
      reader.AutoMode = "ON";

      //SLeep 2000 msecs
      System.Threading.Thread.Sleep(WaitTime);
      reader.AutoMode = "OFF";
      //set switch 2
      IDC_SWITCH_DROP.SelectedIndex = 1;
      IDC_SWITCH_BUTTON.PerformClick();

      //Pause
      while (!OP_Complete)
      {
         System.Threading.Thread.Sleep(500);
      }
      OP_Complete = false;
```

```
//Probe it on tx1, and log rssi
reader.ClearCurrentMessages();

reader.AutoMode = "OFF";
reader.ClearTagList();
reader.AntennaSequence = "0,1";

reader.AutoMode = "ON";

//SLeep 2000 msecs
System.Threading.Thread.Sleep(WaitTime);
reader.AutoMode = "OFF";


//set switch 3
IDC_SWITCH_DROP.SelectedIndex = 2;
IDC_SWITCH_BUTTON.PerformClick();

//Pause
while (!OP_Complete)
{
   System.Threading.Thread.Sleep(500);
}
OP_Complete = false;

//Probe it on tx1, and log rssi
reader.ClearCurrentMessages();
reader.ClearTagList();

reader.AutoMode = "ON";

System.Threading.Thread.Sleep(WaitTime);
reader.AutoMode = "OFF";


//repeat with only tag coming in on tx2
IDC_SWITCH_DROP.SelectedIndex = 0;
IDC_SWITCH_BUTTON.PerformClick();

//Pause
while (!OP_Complete)
{
   System.Threading.Thread.Sleep(500);
}
OP_Complete = false;

//Probe it on tx1, and log rssi
reader.AutoMode = "OFF";
reader.ClearCurrentMessages();
reader.ClearTagList();

//Change antenna sequence to look at delta network
reader.AntennaSequence = "2,3";

reader.ClearCurrentMessages();
reader.AutoMode = "ON";

//SLeep 2000 msecs
System.Threading.Thread.Sleep(WaitTime);
reader.AutoMode = "OFF";
//set switch 2
IDC_SWITCH_DROP.SelectedIndex = 1;
IDC_SWITCH_BUTTON.PerformClick();

//Pause
while (!OP_Complete)
{
   System.Threading.Thread.Sleep(500);
}
OP_Complete = false;
```

```
        //Probe it on tx1, and log rssi
        reader.ClearCurrentMessages();
        reader.ClearTagList();

        reader.AutoMode = "ON";

        //SLeep 2000 msecs
        System.Threading.Thread.Sleep(WaitTime);
        reader.AutoMode = "OFF";


        //set switch 3
        IDC_SWITCH_DROP.SelectedIndex = 2;
        IDC_SWITCH_BUTTON.PerformClick();

        //Pause
        while (!OP_Complete)
        {
            System.Threading.Thread.Sleep(500);
        }
        OP_Complete = false;
        reader.ClearCurrentMessages();
        reader.ClearTagList();

        //Probe it on tx1, and log rssi
        reader.AutoMode = "ON";

        //SLeep 2000 msecs
        System.Threading.Thread.Sleep(WaitTime);
        reader.AutoMode = "OFF";

        //Basic rssi TESTING COMPLETE
        AlienLog.WriteLine(true, "-----RSSI Data Log Ending-----");
        lblStatus.Text = "RSSI Testing Complete!";
        Log_Type = 0;
    }
    catch(Exception exc)
    {
        Log_Type = 0;
        lblStatus.Text = "Exception caught: " + exc.Message;
        AlienLog.WriteLine(true, "Exception caught in Log_RSSI_Click: " + exc.Message);
    }
}
```

2. Alien Reader Event Handler

```
//This is the event handler function that interfaces with the reader to process messages such as it found a tag.
//This function is where the automated RSSI logger function above actually gets back the RSSI information and outputs it to a file to be later analyzed in excel.

void mReader_ServerMessageReceived(string msg)
{
    //Check for if program is already closing and whether the parameter is invalid, then ignore
    // if (mbDisposing) return;
    if (msg == null) return;
    AlienLog.WriteLine(true, "INSIDE FUNCTION!!!");

    //Debug.WriteLine("mServer_ServerMessageReceived():\r\n" + msg.Replace("\0", "\\0"));
    lock (this)
    {
        Doit method = delegate
        {
            //this is so this routine and the clear routine do not coexist?
            lock (moTagLock)
```

```
{
   try
   {
      string key = null;       // tags id
      string tagOnly = null;    // tagstream message without MAC address

      try { tagOnly = msg.Substring(msg.IndexOf(" ") + 1); }
      catch { }
      if (tagOnly != null)
      {
         TagInfo tagInfo = AlienUtils.ParseCustomTag(CUSTOM_FORMAT, tagOnly);
         if (tagInfo == null)
         {
            return;
         }
         key = tagInfo.TagID;

         if (!CurrentTag_List.ContainsKey(key))
         {
            //FIXME: IGNORE this stuff...for now..
            Point p = new Point(CurrentTags.Count * alienTagContol0.Size.Width, alienTagContol0.Location.Y);
            Size size = alienTagContol0.Size;
            AlienRFIDTag myControl = new AlienRFIDTag();
            myControl.TagID = tagInfo.TagID;
            myControl.Location = p;
            myControl.Size = size;

            if ((tagInfo.TagDataArray.Length > 0) &&
               (!string.IsNullOrEmpty(tagInfo.TagDataArray[0])))
               myControl.TagData = tagInfo.TagDataArray[0];

            //this.Controls.Add(myControl);
            //myControl.Show();
            // CurrentTags.Add(key, myControl);

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

            //Start our stuff
            RFIDTag_Info newTagInfo = new RFIDTag_Info();
            CurrentTag_List.Add(key, newTagInfo);

            //Populate List
            //string temp = tagInfo.TagID;

            //TagListing.Items.Add(temp.Replace(" ", ""));
         }
         string temp = tagInfo.TagID;
         TagListing.Items.Add(temp.Replace(" ", "") + "-> RSSI: " + tagInfo.RSSI);

         //New stuff
         CurrentTag_List[key].Frequency = tagInfo.Frequency;
         CurrentTag_List[key].RSSI = tagInfo.RSSI;
         CurrentTag_List[key].RSSILimit = (double)nudRSSIScale.Value;
         CurrentTag_List[key].Speed = tagInfo.Speed;
         CurrentTag_List[key].SpeedScale = (double)nudVelocityScale.Value;
         CurrentTag_List[key].TagData = tagInfo.TagDataArray[0];
         CurrentTag_List[key].TagID = tagInfo.TagID;
         CurrentTag_List[key].DiscoveryTime = tagInfo.DiscoveryTime;
         CurrentTag_List[key].LastSeenTime = tagInfo.LastSeenTime;

         CurrentTag_List[key].TransmitAntenna = tagInfo.TxAntenna;
         CurrentTag_List[key].RecieveAntenna = tagInfo.RxAntenna;
         CurrentTag_List[key].ReadCount = tagInfo.ReadCount;
         CurrentTag_List[key].Direction = tagInfo.Direction;

         //TagListing.Items.Add("CRC(bits): " + tagInfo.TagDataArray[0] + "- TagID: " + tagInfo.TagID);
         switch (Log_Type)
         {
            case 0:
```

```
                        AlienLog.WriteLine(true, "Found Tag: 'ID': " + tagInfo.TagID + "TX: " + tagInfo.TxAntenna + "RX: " + tagInfo.RxAntenna
+ "RSSI: " + tagInfo.RSSI + "Speed: " + tagInfo.Speed + "Direction: " + tagInfo.Direction + "Last seen: " + tagInfo.DiscoveryTime);
                        break;

                    case 1:
                        AlienLog.WriteLine(true, key.Replace(" ", "") + " " + tagInfo.RSSI + " " + tagInfo.Speed + " " + tagInfo.TxAntenna + " " +
tagInfo.RxAntenna + " " + IDC_SWITCH_DROP.SelectedIndex + 1);
                        break;

                    case 2:
                        if (reader.RFAttenuation >= 160)
                            reader.RFAttenuation = 160;
                        else
                            reader.RFAttenuation += 10;

                        AlienLog.WriteLine(true, key.Replace(" ", "") + " " + reader.RFAttenuation + " " + tagInfo.RSSI + " " + tagInfo.Speed + " "
+ tagInfo.TxAntenna + " " + tagInfo.RxAntenna + " " + IDC_SWITCH_DROP.SelectedIndex + 1);
                        break;

                    default:
                        AlienLog.WriteLine(true, "Found Tag: 'ID': " + tagInfo.TagID + "TX: " + tagInfo.TxAntenna + "RX: " + tagInfo.RxAntenna
+ "RSSI: " + tagInfo.RSSI + "Speed: " + tagInfo.Speed + "Direction: " + tagInfo.Direction + "Last seen: " + tagInfo.DiscoveryTime);
                        break;
                }


            }
        }
        catch (Exception ex)
        {
            lblStatus.Text = ex.Message;
            AlienLog.WriteLine(true, "Exception in 'method': " + ex.Message);
        }
    }
  };

  this.BeginInvoke(method);
 }
}
```

3. Arduino and Switch Control Function

```
//This function is responsible for talking to the Arduino to control the RF switch. The codes a,b,c,d along with their corresponding ASCII
//representations are the  codewords used for communication between the two devices.

private void IDC_SWITCH_BUTTON_Click(object sender, EventArgs e)
{
 if(serialPort1 != null)
    if (serialPort1.IsOpen)
    serialPort1.Close();

 System.ComponentModel.IContainer components = new System.ComponentModel.Container();
 serialPort1 = new System.IO.Ports.SerialPort(components);

 //Find out com port to connect to Aldruino
 serialPort1.PortName = Switch_COM.Text.Trim();
 serialPort1.BaudRate = 9600; //Set default rate

 serialPort1.Open();
 if (!serialPort1.IsOpen)
 {
    lblStatus.Text = "Oops couldn't open com port for micro";
    return;
```

```
    }

    // this turns on !
    serialPort1.DtrEnable = true;

    // callback for text coming back from the arduino
    serialPort1.DataReceived += OnReceived;

    // give it 2 secs to start up the sketch
    System.Threading.Thread.Sleep(2000);

    //Codes
    //a -- 97
    //b -- 98
    //c -- 99
    //d -- 100
    // serialPort1.Write(new byte[] { 'a' }, 0, 1);
    lblStatus.Text = "Selecting Switch...";
    using (serialPort1)
    {
       int Switch_Selection = IDC_SWITCH_DROP.SelectedIndex;

       OP_Complete = false;

       switch (Switch_Selection)
       {
          case 0:
             //Switch 1
             lblStatus.Text = "Switch 1 Turning on...";
             serialPort1.Write(new byte[] { 97 }, 0, 1);

             break;

          case 1:
             //Switch 2
             lblStatus.Text = "Switch 2 Turning on...";
             serialPort1.Write(new byte[] { 98 }, 0, 1);

             break;

          case 2:
             //Switch 3
             lblStatus.Text = "Switch 3 Turning on...";
             serialPort1.Write(new byte[] { 99 }, 0, 1);

             break;

          case 3:
             //Switch Shutdown
             lblStatus.Text = "Switches Turning Off...";
             serialPort1.Write(new byte[] { 100 }, 0, 1);

             break;

       }

       System.Threading.Thread.Sleep(2000);
    }

}
```