WSN Final Project

Josh Hooks
CSEE4240
5/6/2014

Problem

The Sahara desert is the one of the largest, hottest deserts in the world (third in size, first in temperature). It is located in northern Africa and has a climate that is extremely dry and includes harsh winds and little to no precipitation. Most parts of the Sahara receive between 2-10cm of precipitation a year, depending on the area, and it is not uncommon to not see rain for an entire year. The terrain is very dry with 25% of the desert being sand dunes and several mountain ranges throughout. Vegetation is sparse throughout the desert and almost nonexistent near the center, which is why it is basically uninhabitable. Temperatures often reach up to 120°F (50°C) with the highest temperature ever recorded being 136°F (58°C). The southern and northern areas of the Sahara are slightly more habitable and do have some vegetation. Most the water sources come either from intermittent streams or from aquifers. Aquifers are rock layers underground that support water flow that is enough to be usable to the human population. Every once in a while these aquifers rise to the surface, but it's hard to know when it will happen. [1].

For the people who wish to study the Sahara desert and for the nomads that travel throughout its 3,500,000 square mile stretch of land, having a more reliable way of knowing where water is located can be a big improvement. The biggest problem is being able to report these aquifers that pop up randomly so that other people can see this information and be able to use it. With the use of mobile robotics, a big step in solving this problem can happen. The solution isn't perfect but it creates a much more up to date, close range, and inexpensive way than there is currently out there (i.e. satellite pictures).

Challenges and Constraints

One of the most immediate challenges posed by this problem is time. It takes time for these mobile robots to traverse the driest areas of the Sahara and report their information back. There won't be a connection to the server everywhere throughout the desert, so these robots will have to make it back to a central location at a specific time that allows them to transmit the collected data across the network. Managing the coordination of getting around the desert in a timely manner and reporting back to that location will be a major task.

The next major challenge in the problem is managing lost sensors. The chance of each sensor and robot traversing through the Sahara and all coming back in one piece, or at all, is slim to none. Making sure that the network of sensors is still operating as it should is very important. If each of these sensors is going off in a particular path, that direction and path is defined in a way that one of the robots needs to be going that way for any sensor to be collecting data around that area. If one of the sensors dies, it needs to be replaced or a working sensor needs to take over in that direction.

The third challenge is going to be managing power sources to keep these things running for a long time. The idea is to constantly have sensors out in the Sahara, so each of these sensors needs to be able to run for a long time. Coin cell batteries can keep small sensors running for a very long time, but they need to sleep an adequate amount to keep them going. Managing the sleep cycles with the transmitting cycles to figure out how large of a battery is needed will be a large task.

System Design

The overall basis for the design that needs to be implemented is to have an Arduino Uno with a humidity sensor, GPS, and XBee radio attached to numerous mobile robots that traverse the Sahara desert taking readings. These readings will then be brought back to a location either on the north, south, west, or east side of the desert. The idea is for the robots to make a circle in the directions (i.e. East to North, North to West) with each having two sets of robots heading out to collect data. For example, if it takes 2 days to get from the East side to the North side, 1 day after the first set of robots depart a second set of them should leave that way there is always sensor data coming in once a day. Not all robots will have the same path, so some will arrive when they can't yet send data and have to wait for that moment to arrive. Once it does and the data is transmitted, all robots from each point will then be sent out at the same time. The schematic in Figure 1 has a visual representation of this system.

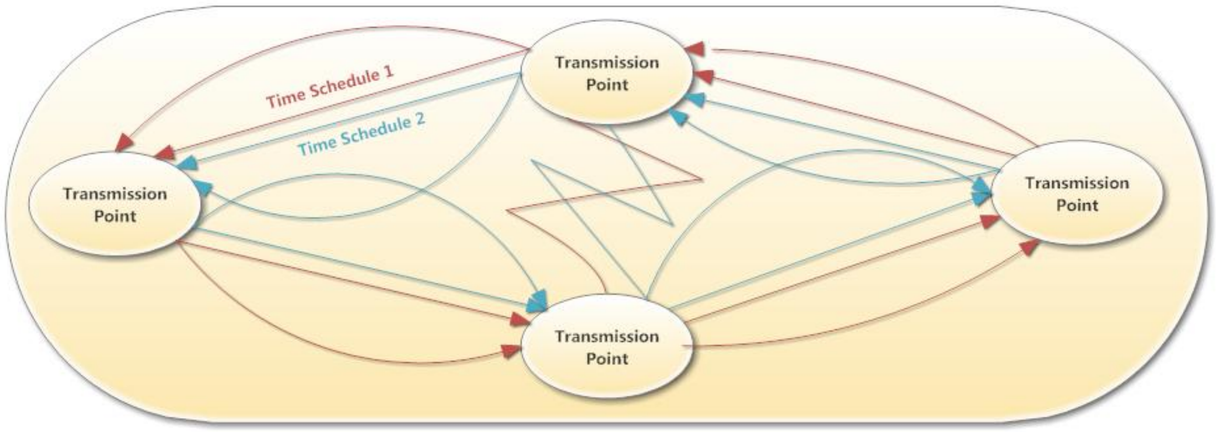

Figure 1: Robot Travel Schedule

The Sahara desert is ~5200 miles long and ~1000 miles tall. The path going from each transmission point is expected to be ~1600 miles long, so the mobile robots need to travel at a speed of 35 m/h in order to make it to the destination in 48 hours.

Since it would be impractical to have both the GPS and humidity sensor running at all times and just giving a large pile of data to the server, the Arduino's microcontroller will be used to implement some logic. The humidity sensor will always take in data at predetermined time intervals, but the GPS should only turn on when these readings are at least somewhat significant. That is, when there is a spike in the humidity. To incorporate this, the Arduino will parse the last 10 readings from the humidity sensor every time a new reading is taken. If the average of these readings is 10% less than the newest reading coming in, then the GPS will turn on and take a reading. That way, when the data is given to the database, you'll be able to see where on a map these spikes of humidity occur.

When these sensors reach their destination point, the Xbee radio will send its data to the coordinator radio. A computer powered gateway program will then poll for information from the coordinator and send that information to the database where it can be stored and kept for later use. Each robot will have 1 Arduino Uno, 1 XBee Series 2 radio, 1GPS, and 1 humidity sensor. The schematic in Figure 2 gives a visual representation as to what this looks like.

The humidity sensor being used to check data is the HIH-4030 sensor. It measures relative humidity in the air and reports it as a percent value. Absolute humidity is the amount of water vapor in the air divided by the mass of dry air at a given temperature. Relative humidity is the current absolute humidity divided by the maximum absolute humidity at the given temperature. A recording of 100% relative humidity indicates a high probability of rain. This sensor operates from -40°C to 85°C and more importantly, operates without any precision differences or time constraints from 0°C to 60°C. This more than covers the spread needed for operating at all time in the Sahara desert. Along with this, it operates on anywhere from 4V to 5.8V, so the Arduino's 5V pin will be an adequate power source. Typically this sensor will be pulling ~200μA of power when reading data with an absolute maximum of 500μA.

The GPS sensor being used to track longitude and latitude data is the LS20031. It can operate accurately anywhere from -30°C to 85°C. Its input voltage can range anywhere from 3V to 4.2V, which can be met by the Arduino's 3.3V pin. When transmitting data, the sensor outputs a current on 2mA, which is much higher than the humidity sensor (by a factor of 10).

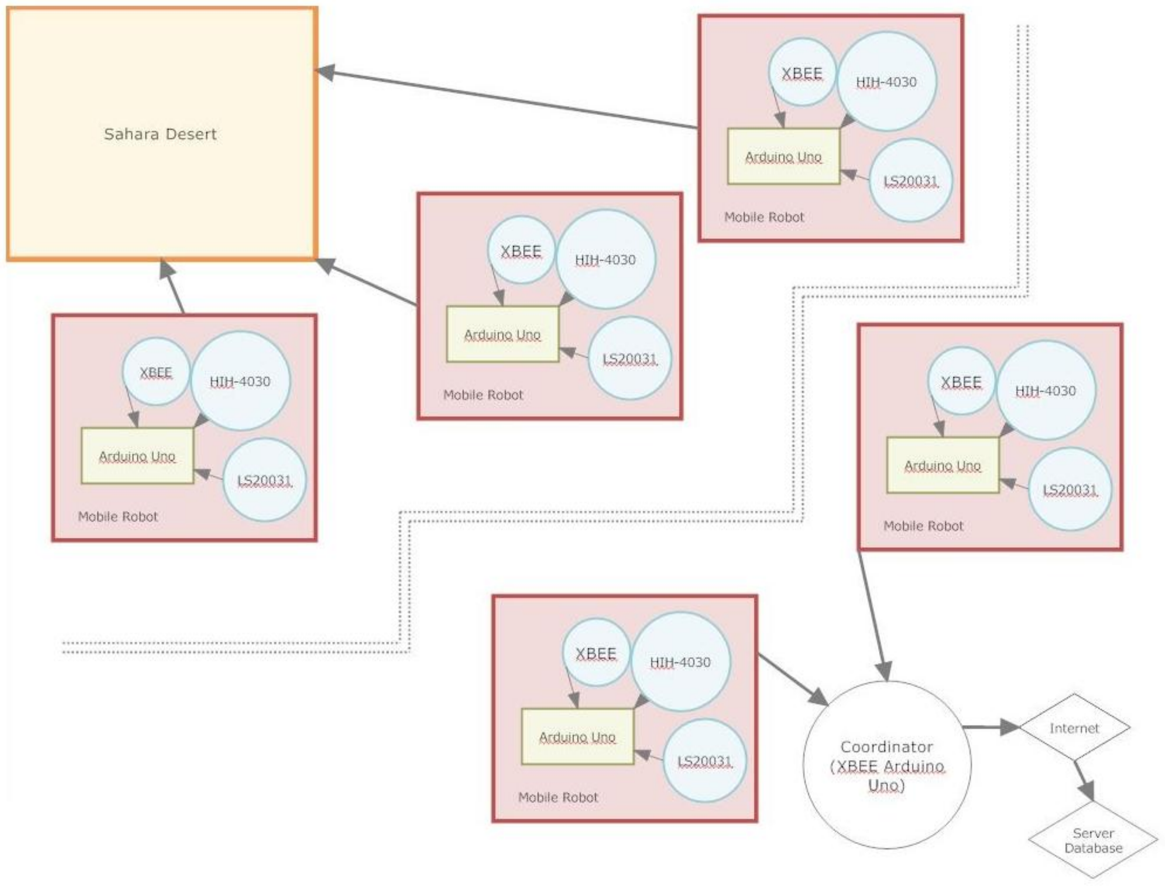

Figure 2: System Design Schematic

Performance Specifications

In order to facilitate a long lasting system design, an appropriate battery must be used to make sure this system lasts a long time. To do this, the sleep cycle must be determined so that an adequate amount of readings are taken to be accurate enough as not to miss any of the water that might be where the sensor isn't actually reading data. The table below shows some of the different scenarios for the humidity sensor alone.

		Years				
		1	2	3	4	5
Sleep(s)	Avg Current (mA)	Battery Required (mAh)				
1	1.0967	9613.76	19227.52	28841.28	38455.03	48068.79
2	0.5991	5251.72	10503.44	15755.17	21006.89	26258.61
4	0.3497	3065.80	6131.60	9197.40	12263.20	15329.00
6	0.2665	2336.43	4672.86	7009.30	9345.73	11682.16
8	0.2249	1971.61	3943.22	5914.83	7886.44	9858.05
10	0.1999	1752.67	3505.35	5258.02	7010.70	8763.37

Since it's almost impossible to tell how often the GPS will actually transmit data, a rough guess, on the high side, on how often this will happen had to be made and was added in on the next table. A reading every 2 hours was used.

		Years				
		1	2	3	4	5
Sleep(s)	Avg Current (mA)	Battery Required (mAh)				
1	1.0967	9857.26	19714.51	29571.77	39429.02	49286.28
2	0.5991	5495.22	10990.44	16485.66	21980.88	27476.10
4	0.3497	3309.30	6618.59	9927.89	13237.19	16546.49
6	0.2665	2579.93	5159.86	7739.79	10319.71	12899.64
8	0.2249	2215.11	4430.21	6645.32	8860.43	11075.54
10	0.1999	1996.17	3992.34	5988.51	7984.68	9980.85

With an 8000mAh coin cell battery priced at ~$10, this sensor system could last up to 3 years with the humidity sensor transmitting data every 6 seconds. Alternatively, it could last between 2 and 3 years transmitting data every 4 seconds. With how important sending data often is in this situation, the 4 second transmission interval seems like a much more realistic option.

Since the information isn't incredibly vital to make secure, there won't be a huge emphases on security. However, having secure transmissions is still important. XBee's built in encryption system will be used to encrypt and then decrypt the data before going to the gateway. Along with that, any access of the database will require a password.

Bandwidth specifications shouldn't result shouldn't result in any interference with other devices as this sensor network is operating in a place where there won't be other connections. If there are, they will be very infrequent and unimportant overall. Considering this, it allows the sensor network to operate efficiently on a 2.4 GHz band.

Database Structure

The database will be set up in a way that it incorporates an ID for each transmission, along with a mote id that corresponds to a particular radio address. This mote id will be the main source of information retrieval. Within that mote id table there will be a radio address, humidity sensor readings, GPS latitude and longitude readings, and the ID that goes with each. The user interface is then designed so that the information is displayed where the humidity points correspond to an actual GPS reading. This information can then be put into a mapping system (such as Google Maps) to find the location of where this reading was taken. The user interface isn't designed to show every reading that is in here, because the database will contain every humidity reading taken.

Sources

[1] http://geography.about.com/od/locateplacesworldwide/a/saharadesert.htm