Duck Watch: A Smart System For Public Lap Pools

Daniel Griffith, Jill Thetford

***Abstract*—This paper presents Duck Watch, a proof of concept for a smart swimming pool. With our system, a swimmer will be able to log on to our website prior to leaving his or her house to help them decide if the conditions are favorable to go to a public lap swimming pool. Our system will inform the user of a number of environmental factors such as water temperature, exterior temperature, and humidity, as well as whether or not there are any open lanes currently.**

I. Introduction

THE initial goal of this project was to design a system that would allow people to determine if it is an optimal time to visit a local pool.

The originator of this project was our adviser, Dr. Martin Kaliski. Being an avid swimmer, Dr. Kaliski expressed great interest in the idea of being able to check the conditions of his local lap pool before ever leaving his house. His original suggestion was to create a network of sensors that could be placed around and possibly in a pool that could gather data and somehow relay that data to a website or mobile application that a person could check before heading to the pool.

Over the two quarters of which this project spanned, the scope was very much like a slinky - constantly expanding and shrinking back down. The desired final project went through a number of iterations. Towards the beginning, we looked into creating a fitness band that swimmers could wear in the pool. This band would have a number of environmental sensors in it that could collect data that would be then transmitted to a database. The benefit of a system like this was that you could very easily gain a count of the occupancy if every person in the pool was required to wear one of these bands. We would even be able to tell you who was at the pool based on an identification marker for each band. The problem with such a system, however, was figuring out how to accurately locate where in the pool a swimmer was. Lane isolation was an issue to figure out a power-efficient solution to. Ultimately this is what made this final product something that was not feasible for the time frame given for this project. We reached out to a few different activity band companies to see if they were open to research projects, but unfortunately they all returned with the response that they either were not interested or did not have the resources at the time.

Eventually, we came to the agreement that we would not be able to put sensors on the swimmers themselves, which meant that we would need to put sensors in the water. It was at this point that we decided that a proof of concept system using a mock pool would be the most feasible scope for this project. The reasons for this were mostly due the fact that it would be very hard to get permission from a public lap pool to put test equipment in their pool and block off a few lanes. Once a proof of concept was decided to be the final goal, the next question that arose was how to simulate the environment - both the pool goers and other environmental weather conditions that a swimmer would care about. Many of the weather conditions that we needed to simulate were able to be done by physically manipulating the sensors (e.g. manipulating temperature with body heat or tilting the wind sensor to simulate windy conditions).

Adviser: Martin Kaliski, Professor Emeritus California Polytechnic State University San Luis Obispo.

II. Design

Hardware

This section will focus on the different hardware components of the project. The hardware components can be split into two parts: sensors and micro-controllers. We determined the optimal method of gathering data at a pool would be to use a network of various sensors that could gather key information which would be sent to a central micro-controller.

Sensors: We determined that the key information potential pool goers cared about was the occupancy (or availability) of the pool and weather conditions at the pool. To get this information we needed several types of environmental sensors. Sensors for both the internal and external temperature of the pool were used in conjunction with a humidity sensor, a wind sensor, and a UV index sensor to determine the weather conditions at the pool. A system of motion sensors were used to determine the lane availability.

Micro-Controllers:

To manage our network of sensors we decided to use a central micro-controller that would pull and temporarily store data gathered from the sensor. Additionally, we needed some way for the data to be transmitted to our online database from our central micro-controller. To achieve this we chose to send the data from our main micro-controller to a secondary micro-controller that had WiFi capabilities.

Architecture:

Each of the components described above needs to communicate with a central micro-controller. This central micro-controller will interface with another micro-controller that has WiFi capabilities. The black box diagram shown in Figure 1 provides a visualization of the hardware architecture for this system.

Software

This section will discuss the various software components of this project. The software portions of this project can be broken down into three main categories: the micro-controller that processes all of the sensor data, the micro-controller that transmits the data to an online database, and the website

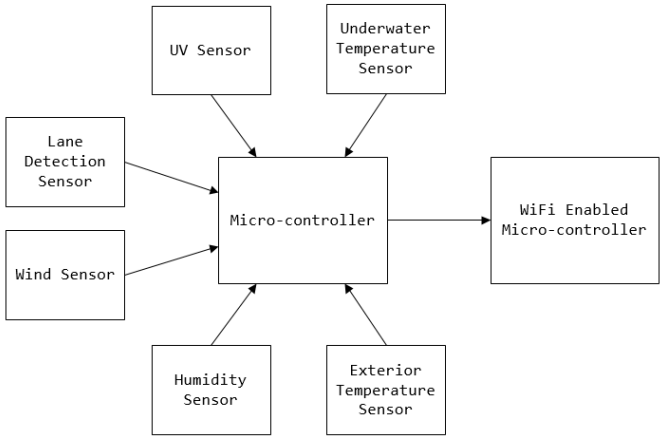

Fig. 1. Hardware Black Box Diagram

that displays the data. For this project, we needed a micro-controller that had enough ports and memory to handle all of the processing required for the various sensors discussed in the Hardware section above.

Sensor Processing:

Our central micro-controller constantly polls each of our sensors sequentially to gather data. Every 30 seconds, the micro-controller builds and sends a packet with the sensor data to the WiFi enabled micro-controller. The flow is depicted in Figure 2.

During each loop of the micro-controller, the raw sensor data is collected and processed via a series of communication protocols and algorithms designed to make sense of the data collected over time.

Data Transmission:

Once all of the sensor data is collected and processed on the first micro-controller, it must occasionally be pushed to a database online so that the user interface can access the information. The sensor information is transmitted from the first micro-controller to the second via a serial connection. The two micro-controllers do not have any way of signaling to each other that data is going to be transmitted or what the data being transmitted corresponds to, so we first need to construct a packet with a known structure for the receiving micro-controller to be able to detect and parse. This packet structure is show in Figure 3. The packet consists of a header "<[=" and a trailer "=]>" that are used to detect the start and end of a packet. The order of the data is set and both micro-controllers are expected to stick to this ordering. The values are comma separated so that the values can be easily parsed out once the packet has been validated.

This second micro-controller that is in charge of transmitting data to the online database has the following typical flow. When the device is powered on, it will first try establishing a WiFi connection. Once a connection is established, the micro-controller will enter its main loop where it checks for a packet, validates the data, parses it, and then transmits the data to the online database via HTTP GET commands. The flow for this micro-controller can be seen in Figure 4.

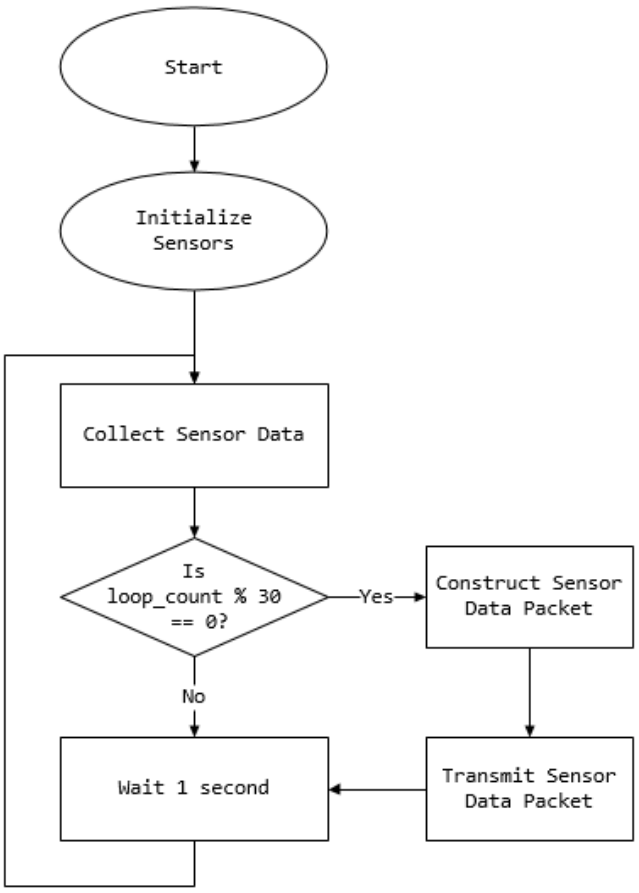

Fig. 2. Sensor Processing Software Flow Diagram

```
<[=surf_temp,sub_temp,ext_temp,
humidity,windy,uv_ndx,
ln_1_status,ln_2_status=]>
```

Fig. 3. Network Packet For Communication Between Micro-Controllers

User Interface: Once the data has been transmitted to the online database, it can be viewed by going to our website [13]. When a user navigates to the website, the page accesses the database to pull the data relevant to the particular pool a user wants to look at. The website was not designed to automatically refresh when it receives new data, so users have to manually refresh the page when the database receives new data.

Architecture: All three of these software components work together to enable the sensor data to get to the user in an easy to understand format. A software architecture diagram is shown in Figure 5 to demonstrate how all of the pieces fit together.

III. Implementation

In this section, we will discuss the actual hardware that was selected for the project, the software implementation, and the

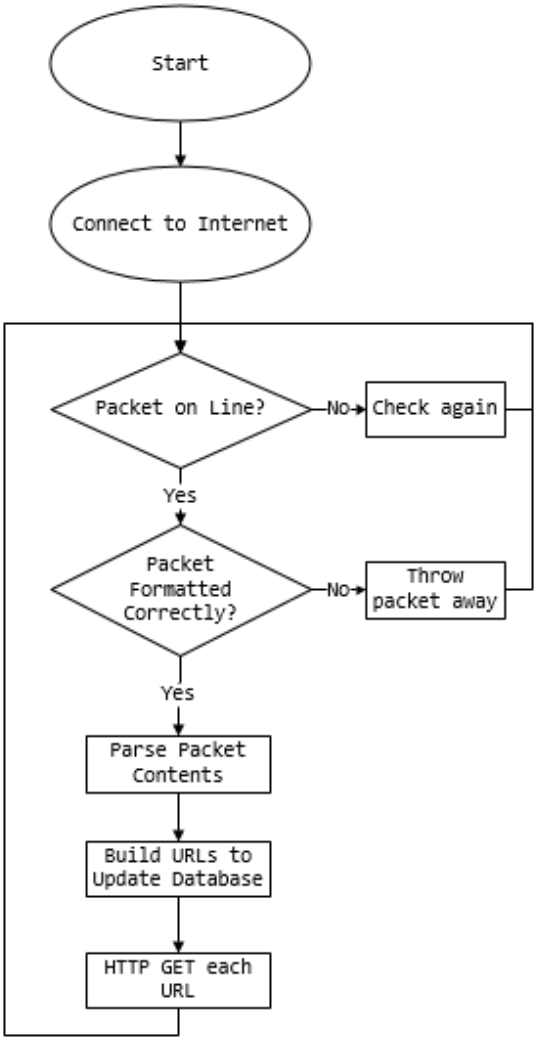

Fig. 4. Software Flow for WiFi Micro-Controller

mechanical system of the project.

Hardware

For our network of sensors we used the sensors described below. A full wiring diagram that shows how all of these components interface with the micro-controllers is shown in Figure 6

BME 280: A temperature and humidity sensor developed by Bosch Sensortec [7] that uses the I^2C protocol to communicate with our micro-controller.

ML8511: A UV index sensor developed by Lapis Semiconductor [16] that transmits an analog signal to communicate with our micro-controller.

DS18B20: A waterproof temperature sensor developed by Maxim Integrated [10] that uses the Dallas 1-Wire protocol to communicate with our micro-controller.

HC-SR501: A passive infrared sensor (PIR) [15] that uses a digital signal to communicate with our micro-controller.

Tilt Ball: A tilt ball sensor [18] that uses a digital signal to communicate with our micro-controller.

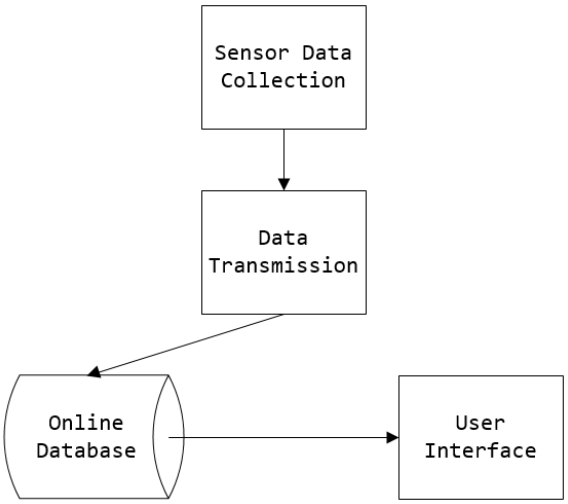

Fig. 5. Software Architecture

Arduino Uno: An 8-bit AVR RISC-based micro-controller board developed by Arduino [4] that uses the ATmega328P micro-controller developed by Atmel [5] and can handle all of the protocols the sensors above use to communicate. This micro-controller served as the central component of our system.

ESP8266: A low-cost micro-controller with WiFi capabilities developed by Espressif [14] that includes a full TCP/IP stack with DNS support used to transmit data to our database.

Software

As mentioned in the design section, the software components of this project can be broken down into three main sections: the micro-controller that process all of the sensor data, the micro-controller that transmits the data to an online database, and the website application which displays all of the sensor information in a user-friendly format.

For the Sensor Data Collection component, the Arduino Uno [4] was selected. This micro-controller has enough ports to handle all of our sensors and has enough processing power to handle the calculations. For the Data Transmission component, we selected the ESP8266 [14] as it has the capability of establishing and maintaining WiFi connections. Our Web Server is hosted and maintained by Thetford Web Development [17]. The database used for this project is hosted by Digital Ocean [9].

Arduino Uno: The Arduino Uno was programmed using C++. Each individual protocol had its own class for our sensors to be constructed with. The sensors that did not require any special protocol had their own unique class that contained all of the methods necessary for collecting the data and applying any history algorithms.

The BME280 sensor had a very helpful library [8] which included all of the calculations that needed to be performed on the raw data collected by the sensor. Parts of this library were used in our code for the purpose of performing the conversions on the data. The whole library could not be used in tandem

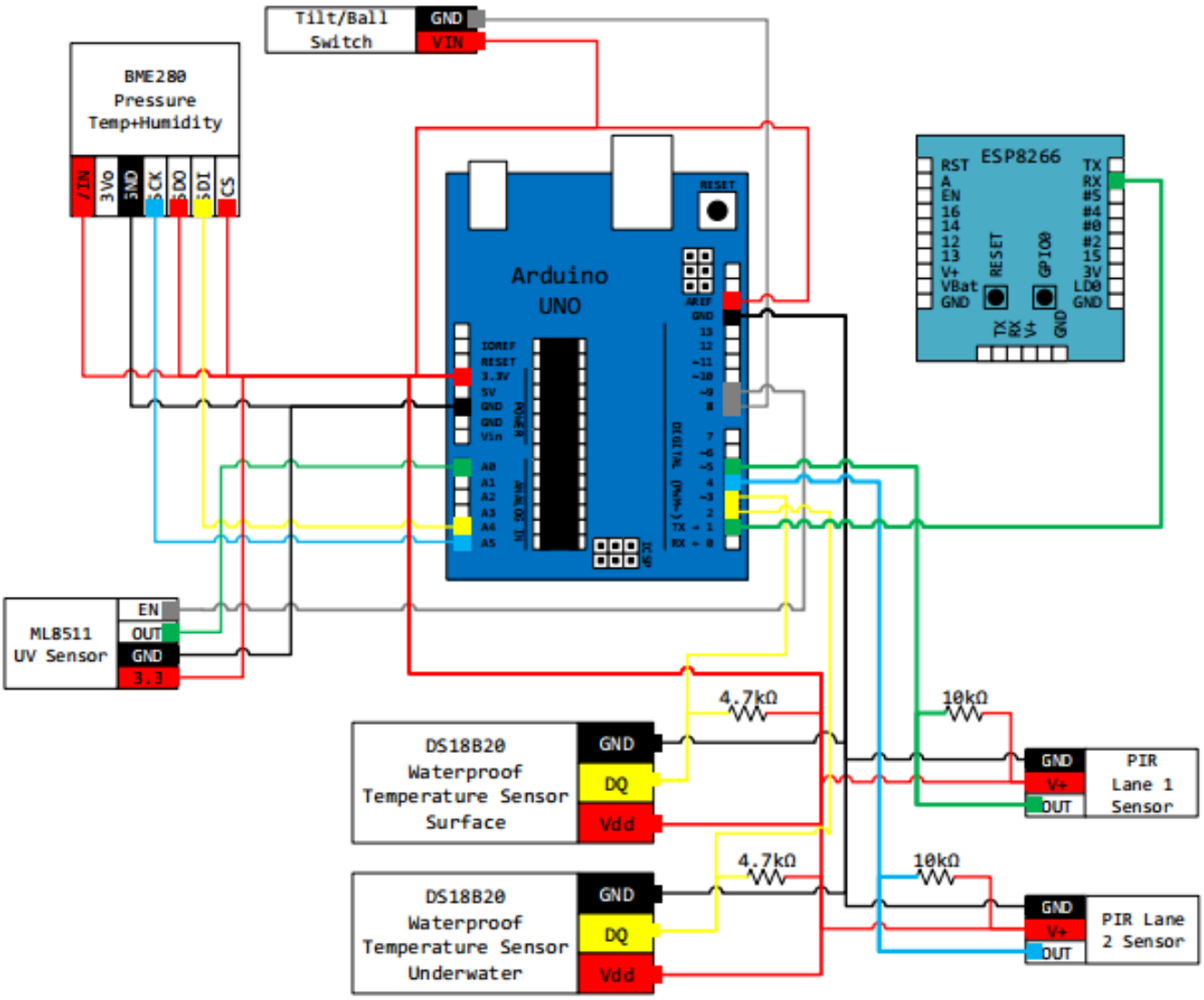

Fig. 6. Full Wiring Diagram

with our code as the library was written for the Arduino IDE [2].

The Uno was developed using Atmel Studio 7 [6]. All source code for the Arduino Uno can be found on git hub in a public repository [11].

ESP8266: The ESP8266 had to be developed using the Arduino IDE [2]. All libraries that the ESP8266 could be programmed with [1] used the functionality of the Arduino IDE. As a result, our options were limited for development. The source code for the ESP8266 can be found on git hub in a public repository [12].

Website Application: For the scope of this project, we designed a website that consisted of a single page that displayed the data gathered from the system set up for the model pool, as seen in Figure 7.

To store the data generated by the system, a MySQL database was used. The database consisted of two separate tables, one for each individual pool and one for each individual lane.

The entries in the pool table consisted of a unique identification number, the surface temperature of the pool, the underwater temperature of the pool, the exterior temperature at the pool, the humidity at the pool, the wind status at the pool (whether a significant amount of wind is present at the pool), and the UV index at the pool.

Each individual lane in the lane table is represented by the pool identification number, the lane number in reference to the pool, and a unique key. Additionally, the status of the lane (whether the lane was occupied or unoccupied), the depth of the lane, and the length of the lane are held in the table. All values in the lane table are held constant except for the lane statuses.

Mechanical

This project was designed to be a proof of concept. As such, we needed a way to control the number of swimmers in the pool at a given time as well as which lanes in a pool that they

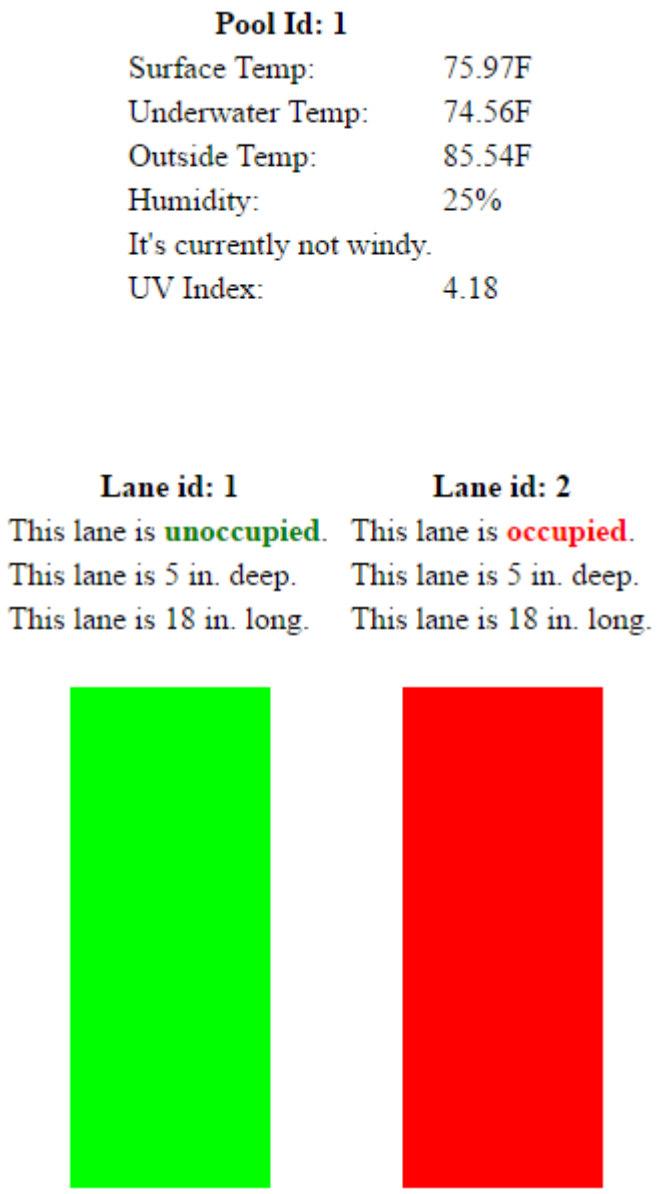

Fig. 7. Duck Watch Web Application

were in. Gaining access to a public pool would not have been feasible for the time constraints of this project. Instead, we opted for a controlled environment. We constructed a mock swimming pool from a large Tupperware container.

In order to simulate swimmers in the pool, we decided to use rubber ducks. The issue with rubber ducks, however, is that we still needed to be able to control them. To accomplish this, we built a halo-like harness that could sit on top of the ”swimming pool.” The harness supports were constructed primarily from wood. Four metal shafts were placed in a rectangular orientation with a gear on each shaft. These shafts would be used to hold a timing belt that would form the track for the rubber ducks to follow while swimming. On one of the shafts was a motor to drive the timing belt. This motor was powered by a power supply that was built in a required course at Cal Poly, IME 156. The IME 156 Basic Electronics Manufacturing class was one of the first classes that we took at Cal Poly during our freshman year. In this class we soldered all of the components needed for the power supply to a printed circuit board and constructed the housing and terminals for the power supply.

Once the halo structure was constructed, it was used to mount many of the sensors as well. The final product of the mock swimming pool with all of the sensors mounted is shown in Figure 8.

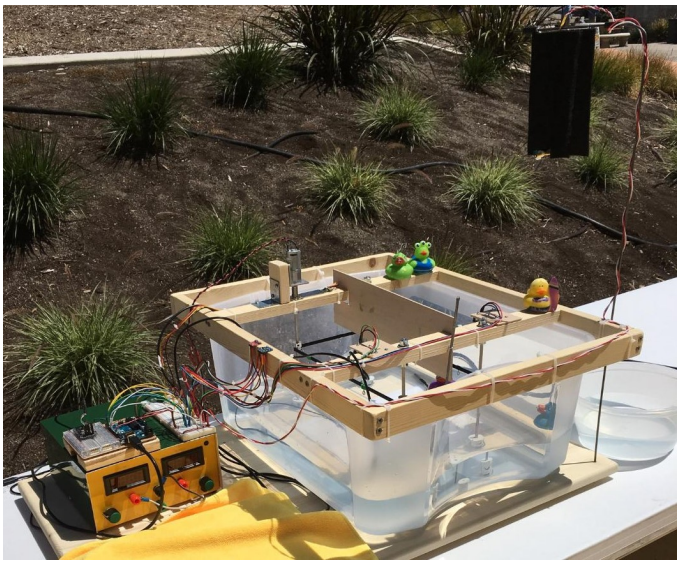

Fig. 8. Final Setup of Pool

As mentioned before, this harness allowed us to simulate swimmers in a pool. The passive infrared (PIR) sensors that were used to detect if a lane is occupied were mounted at the end of each of the lanes, which is just outside the shafts that were opposite the motor.

IV. Testing

This section will discuss the testing and necessary calibration of each sensor and the testing of the system as a whole.

The following sensors were tested in the following ways:

BME280:

We performed functional verification tests for this sensor by reading data from the sensor and verifying that the data was accurate.

ML8511:

We performed functional verification tests for this sensor by reading data from the sensor and verifying that the data was accurate.

DS18B20:

To test the water durability of these sensors we first fully submerged the probe into a cup of water and read data from the sensor. To further test the sensor’s durability we left it fully submerged in water for 24 hours and then read data from the sensor. We determined that the sensor was able to still gather accurate data after being fully submerged in both cases.

HC-SR501:

To test the PIR we triggered the sensor from various distances. To calibrate and ensure that the sensor was reading our motions correctly we used an oscilloscope to read the output of the sensor.

Tilt Ball:

To test the tilt ball sensor we connected the sensor to an LED that would turn on when the sensor was triggered.

ESP8266:

The ESP8266 was first tested by loading a pre-configured Arduino IDE sketch to blink the LED on the board. This allowed us to verify that we could successfully upload code to the board. Once validated, the next step in testing was confirming that we could connect to a WiFi access point. A cell phone with a hot spot enabled was used to verify that the device connected and received an IP address.

V. Future Work

As previously stated, the scope of this project fluctuated over the duration of the project. Given more time and more resources this design could be scaled up to be used with a full-scale pool, as we originally had planned. All of the sensors used could be used in a larger design with the exception of the wind sensor.

Due to monetary constraints we settled for a poor-man's wind sensor using a rudimentary flag and a tilt ball sensor. In a full-scale system we would have liked to use an anemometer to measure wind speed and have a more accurate wind reading. Furthermore, in a scaled up version of this system a more advanced micro-controller, such as the Arduino Mega [3], would be used to accommodate the increase in number of sensors.

VI. Conclusion

Given the project goals and the time allotted for its completion, we believe that this project was a success. Over the course of two very fast-paced quarters, we managed to iterate over a number of different designs and settle on a final design to implement.

The first quarter was spent largely on planning and determining which sensors we would acquire for the final product. The second quarter was spent implementing all of the software, testing it, and constructing the physical pool. As with any project, Murphy's Law is always quick to strike at the most inopportune times. Luckily, none of the issues we ran into stopped us dead in our tracks. In the end, we were able to successfully update a database with environmental data about a mock swimming pool.

References

[1] Arduino core for esp8266 wifi chip. https://github.com/esp8266/Arduino.

[2] Arduino ide 1.6.8. https://www.arduino.cc/en/Main/Software.

[3] Arduino mega by arduino. https://www.arduino.cc/en/Main/ArduinoBoardMega2560.

[4] Arduino uno by arduino. https://www.arduino.cc/en/Main/ArduinoBoardUno.

[5] Atmel 8-bit microcontroller with 32kbytes in-system programmable flash datasheet.

[6] Atmel studio 7. http://www.atmel.com/Microsite/atmel-studio/.

[7] Bme280 combined humidity and pressure sensor datasheet. https://cdn-shop.adafruit.com/datasheets/BST-BME280_DS001-10.pdf.

[8] Bosch sensortec bme280 driver. https://github.com/BoschSensortec/BME280_driver.

[9] Digital ocean. https://www.digitalocean.com/.

[10] Ds18b20 programmable resolution 1-wire digital thermometer datasheet. https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf.

[11] Duck watch ardunio uno git hub source. https://github.com/JillThet/DuckWatch/tree/master/UnoCode.

[12] Duck watch esp8266 git hub source. https://github.com/JillThet/DuckWatch/tree/master/WiFiCode.

[13] Duck watch web app. http://jillthetford.com/SeniorProject/today.php?pool=1.

[14] Esp8266ex datasheet. https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf.

[15] Hc-sr501 pir motion detector datasheet. https://www.mpja.com/download/31227sc.pdf.

[16] Ml8511 uv sensor datasheet. https://cdn.sparkfun.com/datasheets/Sensors/LightImaging/ML8511_3-8-13.pdf.

[17] Thetford web development. http://thetfordwd.com/.

[18] Tilt/ball switch sensor by oddwire. http://www.oddwires.com/tilt-ball-switch-sensor/.