Determining the Wind-Current Relationship Near Shorelines

Shriya Nadgauda, Daphne Poon, Evan Hassman, Jacob Donenfeld
Section 1: Team 4

***Abstract* — The primary goal of our experiment was to explore the correlation between wind and current speeds near shorelines. We did so by designing, building, and deploying an autonomous ocean robot. Using a weather vane and a servo motor, we were able to measure the direction of the wind and position a flow sensor to measure currents parallel and perpendicular to the wind direction. Collecting wind speed data using an anemometer allowed us to determine whether wind has an effect on surface currents near ocean shores. Using results we gathered at Dana Point, CA, we found that in an enclosed bay, that a large factor in noticeable currents is the wind.**

I. INTRODUCTION

Wind speed, as one of the factors affecting current speed, needs to be studied in order to better understand and predict ocean surface currents. In Weber's 1983 paper (1) on ocean currents, he shows that the ocean surface velocity is roughly 3% of wind velocity, but that result is generally only applicable in the open ocean. We measured wind speed and current velocity, in the same direction as the wind and orthogonal, to better understand their correlation, especially close to the shore. We hope understanding this relationship better can help predict tide conditions at beaches. This could be useful for deploying current turbines and wind turbines by knowing the direction of each and if one affects the other.

As we wanted to determine the correlation between wind speed and surface current, we designed our robot to be as buoyant as possible. Furthermore, we programmed our navigation to keep our robot at a single location. To collect data, we used 3 sensors: an anemometer to measure wind speed, a weather vane to determine wind direction and a flow sensor. We attached a servo motor to our flow sensor to get data in the direction of and perpendicular to the direction of wind.

II. SCIENTIFIC GOALS

Our goal for this project was to quantify the relationship between wind speed and surface current speeds, using wind speed, wind direction and current speed sensors. The study of this relationship is relevant in understanding how varying wind conditions can affect the speed of waves near the shore, which in turn is essential for keeping beaches safe. A better understanding of the wind-current relationship will also be useful for deploying current turbines and wind turbines by knowing the direction of each and if one affects the other.

III. SENSORS

To effectively compare the wind speed with the the surface current speed, we needed sensors that would measure the two speeds. We had originally planned to use an anemometer to measure both wind and current speed, but we got access to a flow sensor which was better at measuring current. In order to actually measure how wind speed affected current speed, we wanted to measure the current in the direction of the wind, and then in the perpendicular direction.

To do so, we needed a weather vane to detect the wind direction, and then a servo motor to turn the flow sensor to point in the direction of the wind or perpendicular to it.

A. Anemometer

The anemometer design we used was from the reference designs provided by the E80 teaching team (2). The anemometer used a AH9246 Hall Effect Sensor (3) and a magnet. The magnet was attached to a 3D printed part attached to the end of the bolt that spun with the anemometer cups. The hall effect sensor is a digital sensor that outputs a 1 normally and then a 0 when there is a magnet nearby.

We used the anemometer to record the wind speed which was necessary as our goal was the determine how the speed of the surface current related to the wind speed.

The sensor can operate with an input voltage between 2.2 V to 5.5V. We provided the sensor with 3.3V. The sensor outputs the input voltage, in this case 3.3V to output a 1 and 0V to output a 0.

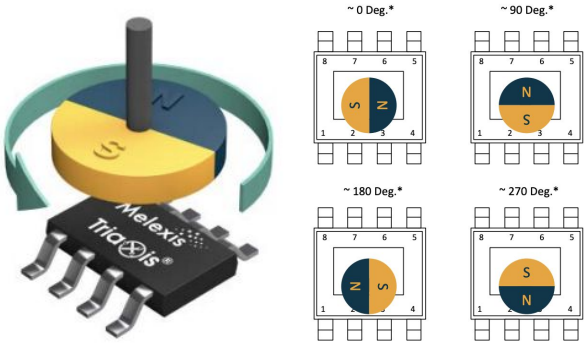
The CAD anemometer design is shown in Figure 1.

FIGURE 1. ANEMOMETER DESIGN

B. Weather vane

For the weather vane, we used an MLX90316 Analog Output Rotary Position IC Sensor (4). This sensor was used as part of a reference design provided by the E80 teaching team. The reference design provided the 3D printed parts necessary as well as the hardware too. This sensor's output voltage varied depending upon the orientation of a magnetic field, Thus, a magnet was placed directly above the sensor on the axle of the weather vane, seen in Figure 2. This functionality and setup eliminates any friction and resistance from a potentiometer or other similar devices, thus making for a better, more accurate design.

FIGURE 2. MLX90316 FUNCTIONALITY



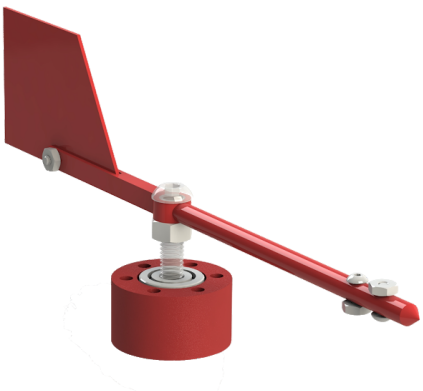
In the reference design the sensor was used in its SPI output mode. For our design we used the analog output setting as the Teensy SPI programmed pins were already being used to communicate with the SD card. The sensor that we originally worked with came in "standard mode". The datasheet provided no discussion of how to program the output mode of the sensor. Due to complications surrounding the output mode of this standard sensor, we began using the preprogrammed analog output version.

The analog output of this sensor was rail-to-rail and ratiometric output that linearly increased based on the orientation of a magnetic field. The output voltage ranged from 8% to 92% of V_{DD}, as specified in the datasheet. We powered the chip with 5V, the typical nominal supply voltage, which yielded an output range of 0.4 - 4.6V. Additionally, the output impedance of the sensor was unknown, as it was not listed on the data sheet. With these specifications, we designed an appropriate circuit to read the output voltage with the Teensy, discussed in Section IV B.

We are trying to observe the correlation between wind speed and top level currents. Thus, the flow sensor must be orientated in the same direction as the wind to observe a correlation.

The CAD design of the weather vane is shown in Figure 3.

FIGURE 3. WEATHER VANE DESIGN



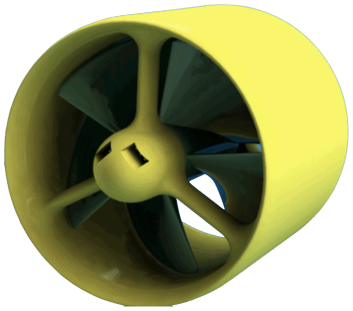
C. Flow Sensor

The flow sensor we used on our robot was borrowed from Professor Clark's HMC LAIR lab and was designed by Ben Chasnov, HMC '16. The sensor used a A1324-26 Linear Hall Effect Sensor (5), which would track the number of rotations of the propellor, green in Figure 4. A hall effect sensor on the side of the the wall of the propeller shell would detect a rotation magnet attached to the fins of a propellor. The propellor would rotate when fluid flowed through the sensor, and the frequency of the output sine wave would be a function of the velocity of the fluid.

This sensor was powered with 3.3V volts, and the Hall Effect sensor is digital, V_{DD} is either 3.3V or 0V.

The CAD design of the flow sensor is shown in Figure 4.

FIGURE 4. FLOW SENSOR DESIGN



1. Servo

A servo motor connected the flow sensor to the robot and controlled the direction that the flow sensor pointed such that the flow sensor would be pointed in the direction of the wind.

The servo was powered with 5V and grounded as well. There was a third connection, signal, that determined the orientation of the servo. We used a 180° servo and a standard servo library to write to it an angle between 0 and 180.

IV. CIRCUIT DESIGN

Our flow sensor and anemometer were very similar in their input power voltage, and output signal voltage. However, our weather vane varied from those two quite a bit. For this reason, we had to design different circuits to accommodate these sensors. Before soldering in our circuits to the protoboard, Figure 6, we had to first test and design them. The circuits diagram for all three sensors is in Figure 5.

A. Anemometer

The anemometer used a digital hall effect sensor, AH9246 as mentioned before, to measure the wind speed. Since its digital output for 1 is V_{DD} and we can only set a signal of a 3.3V to the Teensy pins, we powered the sensor with 3.3V using a regulator. We connected the hall effect sensor to an analog Teensy pin. We chose to use pin A2, 16. However, since the sensor is digital output, we set it up as a digital pin in the Arduino code. We also placed a bypass capacitor between power and GND to ensure a constant steady voltage to power. Additionally, we were worried about the sampling rate of the Teensy, ~10Hz, affecting our data measurement. During

calibration we noticed no issues, even at wind speeds that we didn't expect to see, 15+ mph.

B. Weather vane

As mentioned before, the weather vane used a MLX90316 sensor to measure the direction of the wind. For this sensor, we followed the 16.1 Analog Output Wiring in SOIC-8 Package as given in the datasheet (2). We then placed an unity gain buffer on the output wiring to eliminate any impedance interference. This was then followed by a voltage divider of ⅜ to lower the output voltage range from 0.4V - 4.6V to 0.24V - 2.76V, just in case 5V was outputted, only 3V would reach the Teensy. We connected the sensor output to an analog Teensy pin, and read the pin as an analog source in the Arduino code. We used pin A0, 16.

C. Flow Sensor

Like the the anemometer, this sensor used a hall effect sensor, A1324-26. This sensor was also digital and thus its power voltage was regulated to 3.3V, like the anemometer. It was also connected to an analog Teensy pin, but was read digitally in the Arduino code, we used pin A1, 15. This sensor too had a bypass capacitor between power and GND to dampen any noise in the power signal. We were also worried about the flow sensor not being sampled fast enough by the Teensy. During calibration we found that we would not have an issue since we didn't expect currents faster than 0.25 m/s.

FIGURE 5. CIRCUIT DIAGRAM

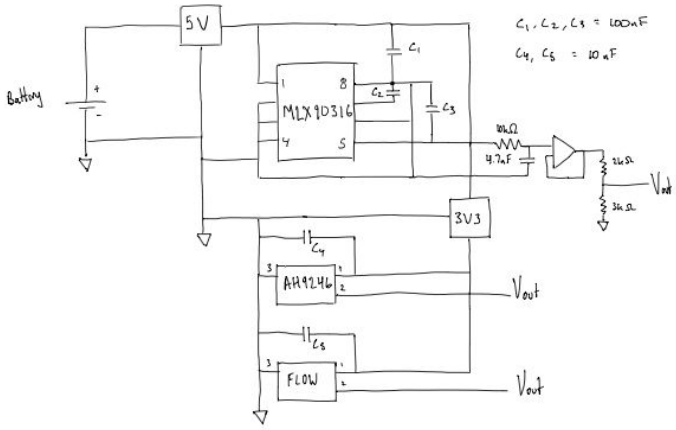
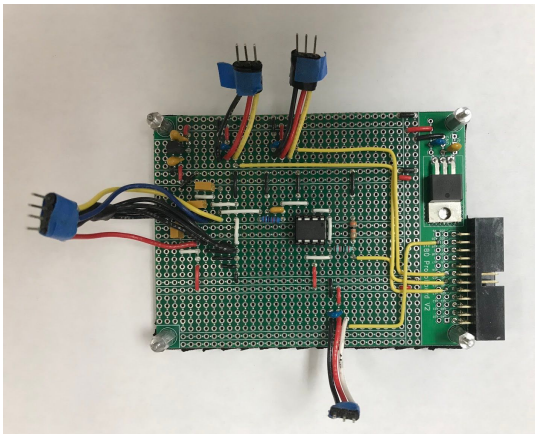


FIGURE 6. PROTOBOARD



IV. SENSOR CALIBRATION

Before deployment, we calibrated our sensors so that we could relate the outputs of each sensor to their physical meaning. We created calibration curves by testing in controlled environments (for example, a wind tunnel), which would allow us to understand the data we would collect during deployment.

A. Weather Vane and Servo

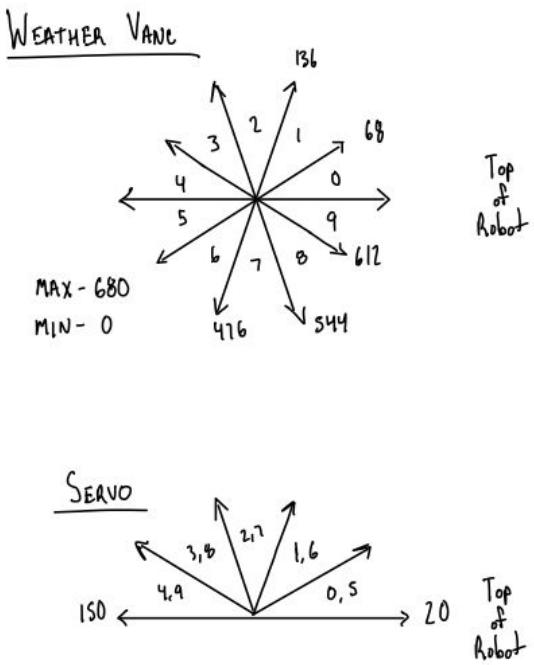
Since the weather vane controls the servo, we needed to calibrate it and determine the weather vane outputs at given directions and then map those outputs to appropriate servo inputs.

To stop the servo from constantly moving due to noise from weather vane, we divided the 360° range into ten ranges, seen in Figure 7, and a weather vane output in a range would correspond to one servo direction. We moved the weather vane around while recording the Teensy output to the Arduino Console so we could make a mapping of the direction and its output.

After calibrating the weather vane, we needed to map its direction to the servo. The servo could turn 180°. We found that the angle between writing a 0 and 180 to the servo produced a spread greater than 180°. Instead we found that the difference between 20 and 150 was much closer to the desired 180° range. Since our flow sensor was bidirectional, it was not an issue that it could not rotate 360°. After determining the orientation and output of the servo we could then map one to the other.

The mapping of the weather vane output to the Teensy and the Teensy input to the servo are shown below in Figure 7, along with the labeled ten sections we broke the direction into.

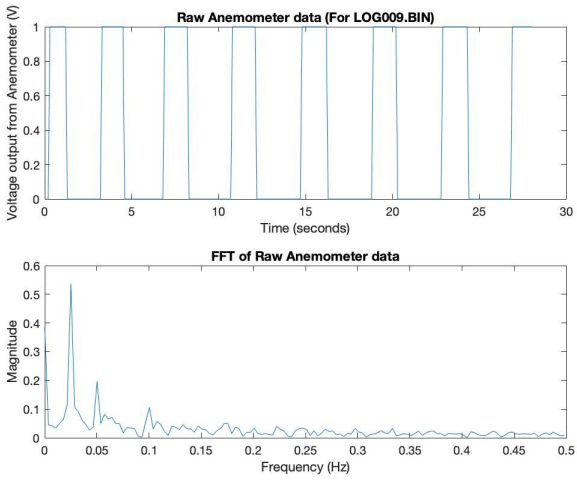
FIGURE 7. TOP-DOWN VIEW OF WEATHER VANE TO SERVO MAPPING



B. Anemometer and Flow Sensor

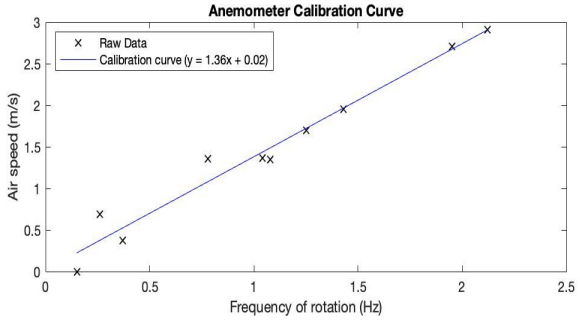
We calibrated the anemometer and flow sensors using the wind tunnel at Harvey Mudd College. Using our previous wind tunnel calibration of the fan RPM to wind, as apart of lab 6 (6), we measured the rotational frequency of the sensors by performing Fourier analysis on the output PWM signal at a given airspeed. We were then able to approximate the frequency of the peak of the fft. After doing so, we converted the signal into a readable frequency in Figure 8.

FIGURE 8. SAMPLE ANEMOMETER CALIBRATION



We were then able to calibrate the anemometer and generate the calibration curve shown in Figure 9.

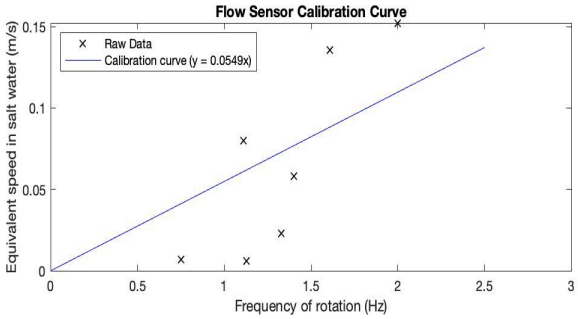
FIGURE 9. ANEMOMETER CALIBRATION CURVE



$R^2 = 0.6624$, ERROR IN SLOPE: ± 0.679, ERROR IN Y-INT: 0.854

Since Reynolds numbers are constant across different fluid kinematic viscosities, we were able to calibrate the flow sensor in the wind tunnel. By rearranging the equation for Reynolds number, or $Re = \frac{v_{air}}{\nu_{air}} = \frac{V_{water} l}{\nu_{water}}$, where l is the chord length of the robot, V is the velocity in a certain fluid, and ν is the kinematic viscosity of a fluid. Since the chord length is constant in any medium, we find that $V_{water} = V_{air} \frac{\nu_{water}}{\nu_{air}}$. Using this conversion, we were able to calibrate the flow sensor and arrive at the calibration curve in Figure 10.

FIGURE 10. FLOW SENSOR CALIBRATION CURVE



$R^2 = 0.6998$, ERROR IN SLOPE: ± 0.3803

V. CONTROL LOOP

A. Proportional Control

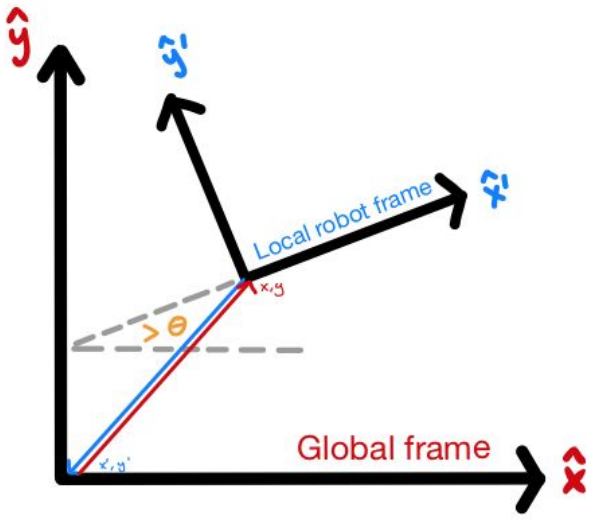
To be able to collect accurate current speed data in our experiment, we needed the robot to stay fixed in one location and to withstand external forces. To do so, we applied an opposing motor force to counteract the force due to surface currents. Since the servo and weather vane worked independent of the robot's orientation, we only used orientation in our control loop. Our motors were aligned orthogonal to each other in order to achieve holonomic motion on the surface of the water.

At the start of every run, after locking GPS signal and waiting two minutes, we recorded the initial GPS location and set that as (0, 0) in the global coordinate frame. The robot's location would then be reported with respect to this coordinate, by transforming from a spherical longitude-latitude coordinate system into a flat cartesian coordinate system. The equation below shows the exact conversion from one system to another.

$$\Delta X = R_{Earth} \cdot \Delta\text{Longitude} \cdot \cos(\text{Latitude}_{\text{origin}})$$
$$\Delta Y = R_{Earth} \cdot \Delta\text{Latitude}$$

For our coordinate conversion, simply knowing the robot has traveled some number of meters in the x direction away from the origin is not enough information to drive the motors. We need to know the orientation of the robot to transform global coordinates into the robot's local coordinates using a counter-clockwise rotation matrix. This would give us the location of the origin with respect to the robot's orientation and location. This is shown in Figure 11.

FIGURE 11. GLOBAL TO LOCAL FRAME CONVERSION



From this, we could drive the motors based on the local x' and y' coordinates.

As shown in Figure 11, a global coordinate (x, y) can be converted into the robot's local coordinate (x', y') based on the orientation of the robot to the global coordinate frame.

B. Weather Vane & Servo

The weather vane's output was reported in values between 0 through 680. This range was discretized into values of 0 through 9. For example, any weather vane value between 0 and 68 would be a 0, and so on. The flow sensor direction was only necessary to control in the range of 180 degrees, as a forwards or backward direction will not change the water flow we record. Thus, if the signal for the weather vane was greater than or equal to 5, we subtracted 5 from it, equivalent to subtracting 180 degrees from an angle above 180. We then used a simple function to map the values 0-4 onto the servo's range of 20-150.

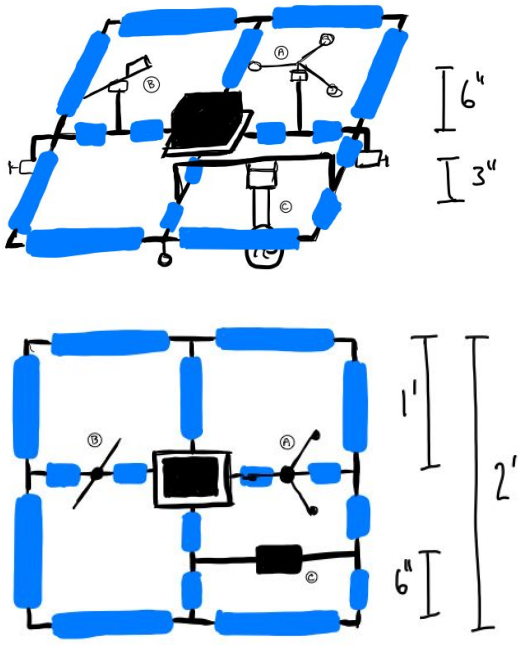
servoPos = (editedPosition / 26) + 33

In order to keep servo movement from being a blocking function of the code, in every main loop of our robot, we updated the servo's position by 1 step if the desired servo position was off from the current servo position.

VI. MECHANICAL ROBOT DESIGN

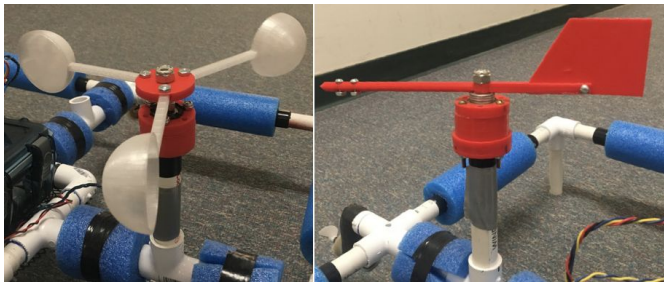
In order to achieve our scientific goals, we need to design a robot that could easily float on the surface and stay in a given position. To achieve these goals, we decided that a large rectangular robot with two sets of parallel motors positioned orthogonal to each other would provide the best buoyancy and overall design. The designs of the robot are seen in Figure 12.

FIGURE 12. PROPOSED ROBOT DESIGN



For the mechanical design of our anemometer and weather vane, we used the reference designs (2) provided by the E80 teaching team. We 3D printed the parts necessary using the provided STL files. The anemometer and weather vane 3D parts perfectly fit onto the end of a PVC pipe, making the connection to the robot extremely easy. The sensors would sit ~6" above the water as seen in Figure 13.

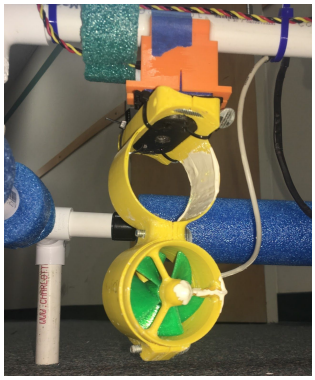
FIGURE 13. ANEMOMETER (LEFT) AND WEATHER VANE (RIGHT)



As a part of our experiment, the flow sensor need to be mounted on a servo that could change the direction in which the sensor was pointed, to align with the weather vane or be perpendicular to it. We mounted the servo and sensor on a

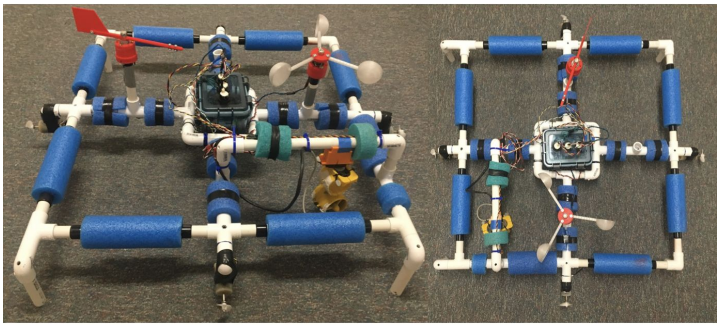
PVC bar across part of our robot using a custom 3D printed part and epoxy, as seen in Figure 13. The servo mount is in orange and the flow sensor is in yellow. We connected the servo to the PVC, and servo to the flow sensor using another custom 3D printed part and zipties, as seen in Figure 13.

FIGURE 13. SERVO MOUNT AND FLOW SENSOR



The final assembled robot can be seen below in Figure 14.

FIGURE 14. FINAL ROBOT CONFIGURATION

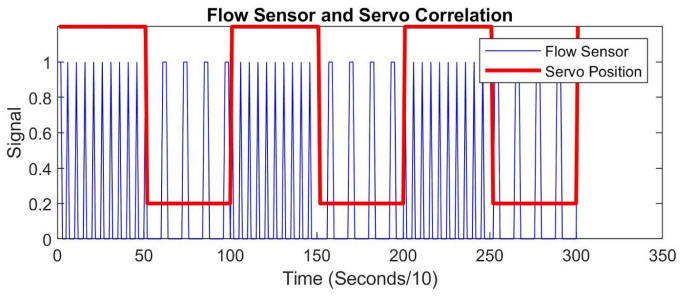


VII. MODELING

A. Sensor Response Modeling

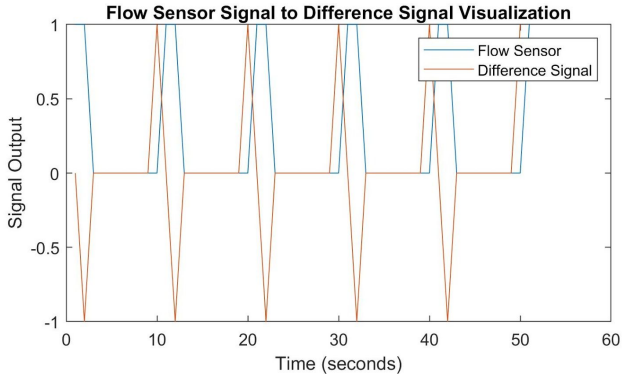
The flow sensor and anemometer were modeled using PWM signals. On our robot, these were both digital PWM signals. We expected our flow sensor to have two different periods for the PWM signal, alternating every 5 seconds, due to facing a different direction with different current speed. This is shown in Figure 15.

FIGURE 15. FLOW SENSOR AND SERVO CORRELATION



We would splice the flow sensor data into segments of 5 seconds, corresponding to the 5 seconds it faced each direction. To analyze this segment of data, the difference function would be called upon it. This converted each value of a given point into its value's difference from it's left and right neighbor. So, if a signal spiked from 0 to 1, it would be 1 for the spike, but then instantly go back to 0 for all remaining values of 1, as the neighboring values are also 1. This is shown in Figure 16.

FIGURE 16. FLOW SENSOR TO DIFFERENCE SIGNAL



Every time the signal spiked the new signal would be a 1. For no change, 0, and when it goes from 1 to 0, it is -1. The peaks of this new signal could be counted easily, which was then converted into a period by dividing this count by total time elapsed.

B. Physical Modeling

Since our robot stayed on the surface of the water, we were not very concerned with its hydrodynamics. Thus, we did not run any COMSOL Multiphysics simulations. However, we did make sure to pay attention to designing a robot that did not generate too much drag from wind. We made our robot keep a low profile and use round PVC piping as the structural beams due to aerodynamics in any wind direction. That being said, since we were position staying, the goal of our p-control system was to maintain positions to counteract any outside forces: wind or currents.

VIII. EXPERIMENTAL PROCEDURE

A. Experimental Protocol and Deployment

We deployed multiple times over the course of the day at Dana Point in order to gather more data. We had developed a launch checklist beforehand (Appendix A), but ran into a few issues at our actual deployment at Dana Point that did not allow us to follow that procedure exactly.

The movement of the motors caused the PVC frame to vibrate which caused the weather vane to move. Additionally, the motor control signal from the Teensy caused noise in the signal read from the Teensy. As a result, we couldn't gather

data while implementing p-control. Our solution was to implement bang bang p-control. We would position stay for a short period of time and then switch to data logging, and then start the process again. Because the water in the bay was fairly calm, the robot should have been able to stay relatively still even with the bang bang system.

Additionally, our p-control was not working (specific details are discussed in sub-section B), so after checking with the proctors and professors, we disabled that functionality. To keep the robot stationary, we simply held the robot in place since we were planning on deploying close to the shore in shallow water.

This meant that we were able to still follow the launch procedure for the at home base and recovery sections. Our final Teensy control code is in Appendix B.

B. Motor Failure

On the day of the deployment at Dana Point, our motors stopped responding to the controls properly; both sets of motors would not simultaneously spin. After troubleshooting using an oscilloscope, we discovered that the input signal into the two H-bridges that were connected to the two pairs of motors were the same, even though the signal coming out the Teensy pins were different. The Teensy pins were properly generating the motor signal we wanted, thus we did not have a coding issue, however, between those pins and the input to the H-bridge, the signal was distorted.

The signals going into the 3 H-bridges were the same whenever we attempted to control multiple motors with different signals. When only one motor was being controlled at a time, the signal coming out of the H-bridges was correct. However, if we tried to turn on 2 motors at the same time, only 1 motor would turn on, even though their signals to their H-bridges were the same.

With the help of multiple student advisors as well as a professor we were unable to determine the root cause of this problem.

C. Data Processing

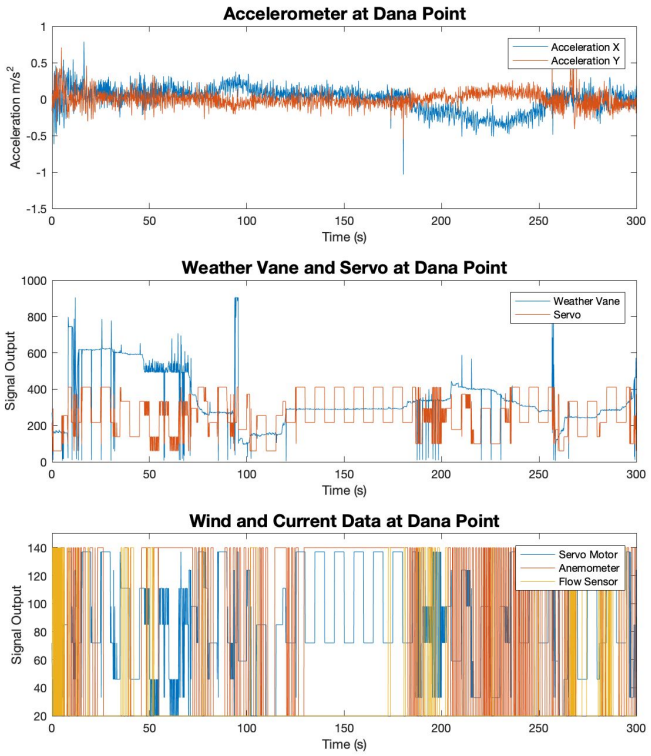
Our first step in processing the data was simply plotting all the measured signals against time. This allowed us to isolate the desired section of our data, where the robot was stationary in the water, not moving to or from launch. We observed the accelerometer in order to determine this time frame.

From there, we compared the data from the weather vane to the servo. When the weather vane was stationary, we expected the servo to look like a steady square wave signal as it switches between being parallel and perpendicular to the direction of the wind. As the weather vane moved, the amplitude of the servo's signal should change to reflect its new position.

Finally, we analyzed the anemometer and flow sensor data. For the anemometer, we determined the frequency of the output PWM signal and from there the wind speed. For the flow sensor, we determined the rate of rotation of the sensor in two parts: the direction of the wind and perpendicular to it. We could do this using the fact that the servo switched between being parallel and perpendicular to the direction of the wind every 5 seconds.

Figure 17 shows the data we collected from a single deployment. In order to view all the signals on the same graph well, we had a to scale our digital signals (from the anemometer and flow sensor) since they ranged between 0 and 1. The analog signals ranged from 0 to 680, for the weather vane and 20 to 150 for the servo. Furthermore, we calibrated the accelerometer data such that it was centered at 0.

FIGURE 17. DANA POINT DATA (TRIAL 3)



We had three successful trials at Dana Point and the data from them can be seen summarized in Table 1. We determined the wind speed by determining the average frequency with which the anemometer spun and then using our calibration curve to turn the frequency into wind speed. We analyzed the flow sensor data in two parts, when the servo was parallel to and then perpendicular to the direction of the wind. For each section of the data, we determined the average frequency with which the sensor spun and then used our calibration curve to get a current speed. The percentage difference is calculated using the following equation.

$$\frac{parallel - perpendicular}{parallel}$$

TABLE 1. EXPERIMENTAL DATA RESULTS

Trial	Wind Speed (m/s)	Current parallel to wind (m/s)	Current perpendicular to wind (m/s)	% Difference
1	0	0.060389 ± 0.023	0.06588 ± 0.025	9.1%
2	3.57 ± 2.41	0.0161 ± 0.0061	0.0102 ± 0.0039	36.6%
3	2.19 ± 2.06	0.0154 ± 0.005846	0.00878 ± 0.003341	43.0%

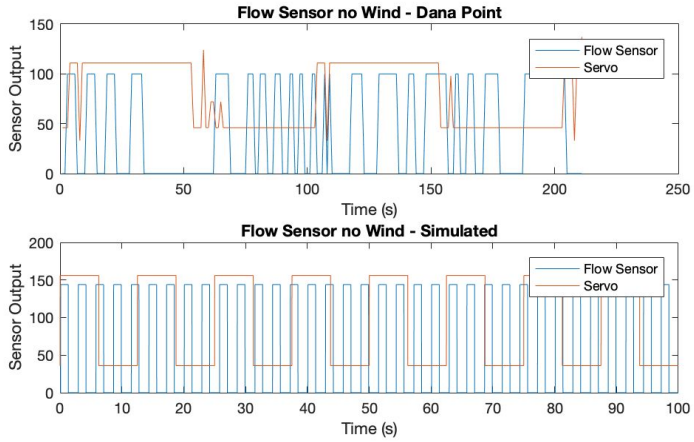
One key aspect of the flow sensor data to note is that the flow sensor does not immediately stop or start spinning when the servo turns. As a result, the signal observed right after the flow sensor turns is a result of the current in both directions. This could be improved upon in future testing by allowing the flow sensor to reach a steady state in each direction before turning the servo. However, this strategy does require the wind direction to be relatively stable.

Additionally, there is some delay in the movement of the flow sensor. It can not move to the perpendicular position immediately, or immediately back to the parallel position.

IX. COMPARISON OF DATA TO MODEL

There are two scenarios in our models. The first is that there is little to no wind. In this case, we expected the current measured to be similar in both directions. This can be seen in Figure 18.

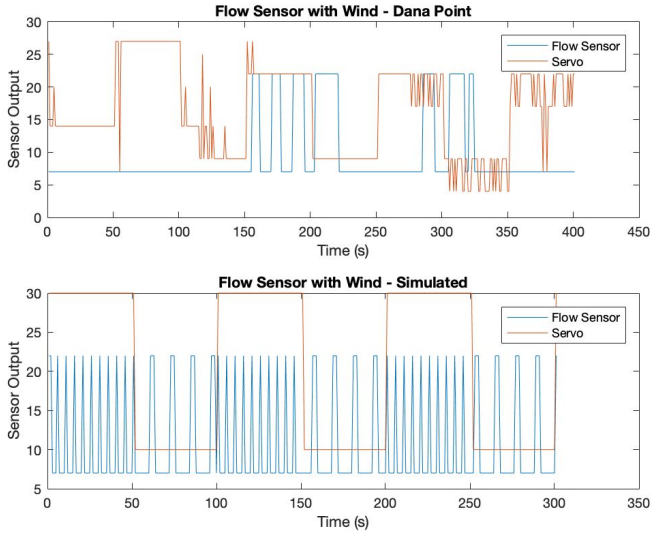
FIGURE 18. NO WIND EXPERIMENTAL (TOP) VS SIMULATED (BOTTOM) FLOW SENSOR AND SERVO DATA



In our experimental data, the signal coming out of the flow sensor changed when the sensor moved from being parallel the direction of wind to perpendicular the direction of wind. However, the difference in the signals was relatively small at 9%. One potential reason for this is that surface current is caused by more factors than simply the wind.

The second scenario we modeled was when there was wind. In this case, we expected to see the flow sensor spin faster in the direction of wind and slower or not at all in the perpendicular direction. Our results along with our simulated data can be seen in Figure 19.

FIGURE 19. WIND EXPERIMENTAL FLOW SENSOR AND SERVO DATA



In our data collected at Dana point, there is lot more noise in the servo position than our simulated data. This is due to the direction of the wind changing which in turn moves the weather vane which causes our servo to move. Furthermore, the signal coming out of the flow sensor is not the same each time the sensor was oriented parallel to the direction of wind. This could be a result of the wind speed changing over time which would then change the surface current. However, we can see that the flow sensor spins more in the direction of the wind (when the servo is high) than when the servo is perpendicular to the direction of the wind. The average difference between the parallel and perpendicular direction was 39.8%.

We did not get that all of the surface current was in the direction of the wind, as we had modeled. This could be a result of two factors. First, surface current may be caused by other factors beyond the wind so there may be current in other directions. Furthermore, as the flow sensor does not immediately stop when it is turned, the some of the readings for when the flow sensor is perpendicular to the direction of the wind may be a result of the flow sensor slowing down rather than current.

X. CONCLUSION

A. Lessons Learned

We predicted that proportion of the surface current at the beach would be due to the wind, and the rest of the surface current would be a result of the difference in temperature between the water at the surface and the water below and

differing water densities. We can see this in our results in two places. First, when there is no wind there was still surface current, showing that current exists without wind. Second, while there was a difference between the speed when the flow sensor was parallel and perpendicular to the direction of the wind, there was still current when the flow sensor was perpendicular.

An unexpected issue we ran into was our inability to collect data while implementing p-control as a result of the motors influencing the weather vane data. This shows that while pieces may work individually, testing everything together is also crucial.

B. Future Work

Given more time, we could have included more variables in our model for the wind-current relationship. One example is to use the current depth of the water and our robot's distance from the shore as variables in our wind-current velocity model. We believe that the depth of the water may dampen the energy from the wind to the water surface, and the distance from the shore may also have a similar effect.

Furthermore, ideally the robot would be able to autonomously navigate and collect data simultaneously. To do this, we would have to determine how to remove the motor noise from the weather vane readings and how to prevent the vibrations of the motors from moving the weather vane.

We were also only able to conduct 3 successful trials. While our data does appear to show a correlation between wind speed and surface current, we would need to conduct more trials in order to make a concrete statement about the correlation or lack thereof.

XI. ACKNOWLEDGEMENT

We would like to thank Professors Clark for letting us use his flow sensor and Ben Chasnov '16 for designing the sensor and providing us with the reference designs.

REFERENCES

[1] J. E. Weber, "Steady Wind- and Wave-Induced Currents in the Open Ocean," *Journal of Physical Oceanography*, vol. 13, no. 3, pp. 524–530, March 3 1983. [Online].

[2] Reference Designs, Harvey Mudd College, Claremont, CA, 2018. https://sites.google.com/g.hmc.edu/e80/project/reference-designs?authuser=0 [Online]. [Accessed: 15-March-2019].

[3] Diodes Incorporated, AH9426 Ultra High Sensitivity Micropower Omnipolar Hall-Effect Switch Datasheet, Revision 2-2, September 2015. [Online].

[4] Melexis, MLX90316 Rotary Position Sensor IC Datasheet, Revision 011, August 17 2017. [Online].

[5] Allegro Microsystems LLC, A1324, A1325, and A1326 Low Noise, Linear Hall Effect Sensor ICs with Analog Output Datasheet, Revision 5, February 14 2019. https://www.digikey.com/en/datasheets/allegromicrosystemsllc/allegro-microsystems-llca132456datasheetashx [Online].

[6] Lab 6: Fluid Dynamics, Harvey Mudd College, Claremont, CA, 2018. Accessed On: March. 3, 2019.

[7] "Air - Dynamic and Kinematic Viscosity," Engineeringtoolbox.com, 2019. [Online]. [Accessed: 03-May-2019].

[8] "Physical properties of seawater 2.7.9," Npl.co.uk, 2017. [Online]. [Accessed: 03-May-2019].

[9] Deziel, C. (2018). What Are Surface Currents Caused By?. [online] Sciencing. Available at: https://sciencing.com/what-surface-currents-caused-5003471.html [Accessed 7 May 2019].

APPENDIX

A. Launch CheckList

At home base

- ❏ Wipe SD Card
- ❏ Check battery voltage and current with multimeter or charger
  - ❏ If necessary, plug in and charge battery
- ❏ Connect Teensy to a laptop, run the uploaded code, and make sure GPS / IMU / ADC are showing up on serial monitor
- ❏ Make sure weathervane, anemometer and flow sensor are showing up on the serial monitor as well
- ❏ Check servo, and make sure it turns to face the angle it reads
- ❏ Seal lid of waterproof box
  - ❏ Make sure o-ring is in contact with the box
  - ❏ Make sure you the locking mechanism is completely snapped shut
- ❏ Test waterproofing in freshwater bucket to make sure there are no leaks
- ❏ Ensure weathervane, anemometer and flow sensor are all plugged in, and attached to the robot

At launch site

- ❏ Plug SD card into Teensy
- ❏ Place battery in box
- ❏ Place motherboard and Teensy in box, on top of the battery
- ❏ Plug battery into the motherboard
- ❏ Plug weathervane wires into Teensy
  - ❏ Make sure the wires match (blue for +5V, green for signal, black for GND)
- ❏ Plug servo wires into Teensy
  - ❏ Make sure the wires match (red for +5V, yellow for signal, black for GND)
- ❏ Plug motor wires into Teensy
  - ❏ Make sure the wires match (for motor pair A: blue for GND, yellow for power; for motor pair B: green for GND, white for power)
- ❏ Plug anemometer into Teensy
  - ❏ Make sure the wires match (red for +5V, yellow for signal, black for GND)
- ❏ Plug flow sensor into teensy
  - ❏ Make sure the wires match (red for +5V, white for signal, black for GND)
- ❏ Ensure anemometer and weathervane are correctly oriented on top of the robot
- ❏ Ensure flow sensor servo can turn without obstruction
- ❏ Ensure weathervane can turn without obstruction
- ❏ Power on the Teensy by connecting the motherboard to the battery
- ❏ Wait for GPS signal to fix
- ❏ Close waterproof box
- ❏ Row/swim out to location
- ❏ (Data will automatically start logging and the GPS positions will be recorded 2 minutes after the GPS signal is locked)
- ❏ Place the robot in water, making sure that the motors and flow sensor are submerged
- ❏ Watch to make sure the robot doesn't move around too much (the P-control should move the robot back to its original position if it *does* move, but be around in case it doesn't work properly)

Recovery

- ❏ Swim/row out to robot
- ❏ Remove the robot from water
- ❏ Dry the outside of the robot with a towel
- ❏ Turn off Teensy by unplugging the battery
- ❏ Remove SD card from motherboard
- ❏ Check the SD card to ensure that data was logged

B. Arduino Code

```
// general
#include <Arduino.h>
#include <Wire.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include <Pinouts.h>
#include <TimingOffsets.h>
#include <SensorGPS.h>
#include <SensorIMU.h>
#include <StateEstimator.h>
#include <ADCSampler.h>
#include <ErrorFlagSampler.h>
#include <BRButton.h> // A template of a data
source library
#include <MotorDriver.h>
#include <Logger.h>
#include <Printer.h>
#include <PControl.h>
#define UartSerial Serial1
#include <GPSLockLED.h>
#include <Servo.h>

#include <Flowsensor.h>
#include <Anemometer.h>
#include <WindVane.h>
#include <ServoDriver.h>

//////////////////////* Global Variables
*//////////////////////

MotorDriver motor_driver;
StateEstimator state_estimator;
PControl pcontrol;
SensorGPS gps;
Adafruit_GPS GPS(&UartSerial);
ADCSampler adc;
ErrorFlagSampler ef;
SensorIMU imu;
Logger logger;
Printer printer;

Flowsensor flowsensor;
Anemometer anemometer;
WindVane windvane;
Servo myservo;
ServoDriver sDriver;

// loop start recorder
bool perpendicular = false;
bool moveServo = false;
int loopStartTime;
int currentTime;
int servo_position;
int current_way_point = 0;

volatile bool EF_States[NUM_FLAGS] = {1,1,1};

// 120 seconds
int startDelay = 12000;

// variable to store the servo yaw
// yaw values are given in degrees
int flow_yaw = 0;

// tolerance on the difference in angle between
the flow sensor and wind direction
int tolerance = 3;

float initial_lat = 0;
float initial_long = 0;
float lastSwapTime = 0;
float lastServoTime = 0;

void setup() {
  startDelay = 0;
  logger.include(&imu);
  logger.include(&gps);
  logger.include(&state_estimator);
  logger.include(&pcontrol);
  logger.include(&motor_driver);
  logger.include(&adc);
  logger.include(&ef);
  logger.include(&flowsensor);
  logger.include(&anemometer);
  logger.include(&windvane);
  logger.include(&sDriver);
  logger.init();

  printer.init();
  ef.init();
  imu.init();
  UartSerial.begin(9600);
  gps.init(&GPS);
  motor_driver.init();


  flowsensor.init();
  anemometer.init();
  windvane.init();
  sDriver.init();

  const int number_of_waypoints = 2;
  const int waypoint_dimensions = 2;       //
waypoints are set to have two pieces of
information, x then y.
  double waypoints [] = { 0, 10, 0, 0 };   //
listed as x0,y0,x1,y1, ... etc.
  pcontrol.init(number_of_waypoints,
waypoint_dimensions, waypoints);
  // **************WHAT IS THIS ABOVE ME
  state_estimator.init();

  printer.printMessage("Starting main loop",10);
  loopStartTime = millis();
  printer.lastExecutionTime         =
loopStartTime - LOOP_PERIOD +
PRINTER_LOOP_OFFSET ;
  imu.lastExecutionTime             =
loopStartTime - LOOP_PERIOD + IMU_LOOP_OFFSET;
```

```
  gps.lastExecutionTime             =
loopStartTime - LOOP_PERIOD + GPS_LOOP_OFFSET;
  adc.lastExecutionTime             =
loopStartTime - LOOP_PERIOD + ADC_LOOP_OFFSET;
  ef.lastExecutionTime              =
loopStartTime - LOOP_PERIOD +
ERROR_FLAG_LOOP_OFFSET;
  state_estimator.lastExecutionTime =
loopStartTime - LOOP_PERIOD +
STATE_ESTIMATOR_LOOP_OFFSET;
  pcontrol.lastExecutionTime        =
loopStartTime - LOOP_PERIOD +
P_CONTROL_LOOP_OFFSET;
  flowsensor.lastExecutionTime      =
loopStartTime - LOOP_PERIOD + FLOWSENSOR_OFFSET;
  anemometer.lastExecutionTime      =
loopStartTime - LOOP_PERIOD + ANEMOMETER_OFFSET;
  windvane.lastExecutionTime        =
loopStartTime - LOOP_PERIOD + WIND_VANE_OFFSET;
  logger.lastExecutionTime          =
loopStartTime - LOOP_PERIOD +
LOGGER_LOOP_OFFSET;

  // attaches the servo on pin 32 to the servo
object
  // DO NOT use pin 9 - will cause GPS errors
and false short circuit flags
  myservo.attach(31);

  delay(startDelay);
  gps.read(&GPS);

  // unsure if this is necessary now - ask Jacob
  pcontrol.longStart = gps.state.lon;
  pcontrol.latStart = gps.state.lat;

  //Blocking code to wait for satalites and set
origin
  // DO NOT add a delay - will cause the GPS to
stop working
//  Serial.print("Looking for satalites.. ");
//  gps.read(&GPS); // blocking UART calls
//  while (gps.state.num_sat < 6){
//    Serial.println(gps.printState());
//    gps.read(&GPS);
//  }
Serial.println("Aquired Satalites and set
origin");
  initial_lat =  gps.state.lat;
  initial_long = gps.state.lon;


  //Set servo to position 0.
  for (int pos = 180; pos >= 20; --pos) { //
goes from 180 degrees to 0 degrees
  myservo.write(pos);              // tell servo
to go to position in variable 'pos'
    delay(15);                       // waits
15ms for the servo to reach the position
  }
 Serial.println("Set servo to position 0");

 // 20 to 150 is 180 degree range
}


void loop() {
  // initalize a wind direction variable
  int wind_yaw;

  currentTime = millis();

  // Read from IMU
  if ( currentTime-imu.lastExecutionTime >
LOOP_PERIOD ) {
    imu.lastExecutionTime = currentTime;
    imu.read();     // blocking I2C calls
  }

  // Read GPS
  if (true){//(gps.loopTime(loopStartTime)) {
    gps.lastExecutionTime = currentTime;
    gps.read(&GPS); // blocking UART calls
  }

  // Read Anemometer
  if (currentTime-anemometer.lastExecutionTime >
LOOP_PERIOD) {
    anemometer.lastExecutionTime = currentTime;
    anemometer.read(); // blocking UART calls
  }

  // Read WindVane
  if (currentTime-windvane.lastExecutionTime >
LOOP_PERIOD) {
    windvane.lastExecutionTime = currentTime;
    windvane.read(); // blocking UART calls
  }

  // Read Flowsensor
  if (currentTime-flowsensor.lastExecutionTime >
LOOP_PERIOD) {
    flowsensor.lastExecutionTime = currentTime;
    flowsensor.read(); // blocking UART calls
  }

    // get the windvane position in a way the
servo can read it
     wind_yaw = windvane.servoWrite();

     // code to move servo as necessary
    int delta_yaw;
    if (currentTime - lastSwapTime > 5000){
      perpendicular = !perpendicular;
      lastSwapTime = currentTime;
    }


   // delta_yaw = wind_yaw - servo_position;
       if (perpendicular){
      if (wind_yaw > 75){
        wind_yaw = wind_yaw - 65;
      }
      else {
        wind_yaw = wind_yaw + 65;
      }
    }
```

```
    if (wind_yaw > servo_position){
      for (int i = servo_position; i < wind_yaw;
i++){
        myservo.write(i);
        servo_position++;
      }
    }
    else {
      for (int i = servo_position; i > wind_yaw;
i--){
        myservo.write(i);
        servo_position--;
      }
    }
      sDriver.setPosition(servo_position);
    // ****** END SERVO CODE ********* //

    // Not doing P-Control as Motors Don't Work
//    // P-Control Call
//    if (
currentTime-pcontrol.lastExecutionTime >
LOOP_PERIOD ) {
//      pcontrol.lastExecutionTime =
currentTime;
//
//      state_estimator.state.origin_lat =
initial_lat;
//      state_estimator.state.origin_lon =
initial_long;
//
//
pcontrol.calculateControl(&state_estimator.state
, &gps.state);
//      Serial.println("calledControl");
//      // this is where the motors are actually
instructed to drive
//
motor_driver.drive(pcontrol.uL,pcontrol.uR,0);
//    }
```

```
  // Update State Esitmator
  if (
currentTime-state_estimator.lastExecutionTime >
LOOP_PERIOD ) {
    state_estimator.lastExecutionTime =
currentTime;
    state_estimator.updateState(&imu.state,
&gps.state, initial_lat, initial_long);
  }
```

```
  if (currentTime - adc.lastExecutionTime >
LOOP_PERIOD) {
    adc.lastExecutionTime = currentTime;
    adc.updateSample(servo_position);
  }

  // Log Data onto the SD card
  // Should be the last thing to run
  if (currentTime- logger.lastExecutionTime >
LOOP_PERIOD && logger.keepLogging) {
    logger.lastExecutionTime = currentTime;
    logger.log();
  }

    if ( currentTime-printer.lastExecutionTime >
LOOP_PERIOD ) {
    printer.lastExecutionTime = currentTime;
    printer.printValue(0,logger.printState());
    printer.printValue(1,gps.printState());

printer.printValue(2,state_estimator.printState(
));

printer.printValue(3,pcontrol.printWaypointUpdat
e());

printer.printValue(4,pcontrol.printString());

printer.printValue(2,motor_driver.printState());

printer.printValue(3,imu.printRollPitchHeading()
);
    printer.printValue(4,imu.printAccels());
    printer.printValue(5,adc.printSample());

printer.printValue(6,anemometer.printState());

printer.printValue(7,flowsensor.printState());
    printer.printValue(8,windvane.printState());
    printer.printValue(9,sDriver.printState());
    printer.printToSerial();
    state_estimator.printState();
    Serial.print("yaw is ");
    Serial.println(pcontrol.yaw);
    Serial.print("perp:  ");
    Serial.println(perpendicular);
  }

}
```