

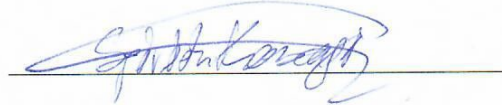
ESTIMATION OF A PLUME WITH AN UNMANNED TERRESTRIAL VEHICLE

A Major Qualifying Project
Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in Partial Fulfillment of the Requirements for the
Degree of Bachelor of Science
in Aerospace Engineering

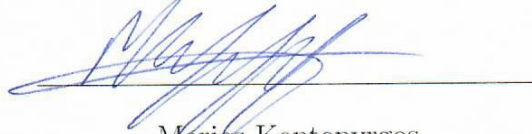
By



Sylvester Halama



Spiridon Kasapis



Mariós Kontopyrgos



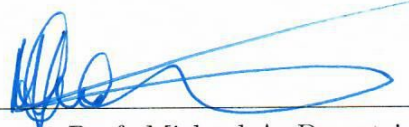
Owen McGrath



Benjamin Preston

March 2, 2018

Approved by: _____



Prof. Michael A. Demetriou
Aerospace Engineering Program,
Mechanical Engineering Department, WPI

Abstract

This work involves the design and implementation of a gas-sensing mobile robot as an experimental tool to localize a carbon dioxide source. The autonomous robot achieves navigation through an embedded microcontroller using a strap-down accelerometer and a fusion of four CO₂ sensors. A mass flow controller and diffuser are used to dependably generate a plume that simulates a point source. A base station receives sensor data and calculates the robot's position using the accelerometer data filtered using a low pass filter followed an Extended Kalman Filter. This method has applicability for unmanned vehicles tracking emissions of contaminants and their effects in the environment.

Acknowledgements

We would like to thank certain individuals for their continued help throughout this project, which would not have been possible without them. We would first like to thank our advisor Professor Demetriou for his guidance throughout this project. We would also like to thank Professor Gatsonis for his help on the plume generation system and plume simulations. Lastly, we would like to thank the Mechanical and Aerospace Engineering Department staff and more specifically Erica Stults, Barbara Furnham and Randy Robinson for their logistical assistance.

Table of Authorship

Section	Primary Author	Project Work
Introduction		
Project Motivation	SH, BP	ALL
Overview of Earlier Relevant MQPs	SH, BP	ALL
Robot		
Previous Robot	BP	ALL
New Robot	BP, OM	ALL
Sensor Platform Design	SK	SK
Structural Stability Simulation	SK	SK, SH, BP
Base Station	OM	OM
Camera Incorporation	MK, OM	OM, MK
Wireless Communication	OM	OM
CO ₂ Sensors	OM, BP	OM, SH
CO ₂ Sensor Calibration	OM	OM
IMU and Vibration Damping	BP, SK	BP, SK
Dynamics and Controls		
Kinematic Equations	SK	ALL
Simulink Control Simulation	SK	ALL
Controller	SH	SH
Noise in the Simulation	SH	SH, MK
Dynamic Model	OM, SK	ALL
Extended Kalman Filter	SK, OM	OM, MK
Complete Control Diagram	SH, MK	SH, MK
Low Pass Filter	SH	SH
Plume Generation System		
Flow Meter	MK	MK, SH
Solenoid Control	SH	MK, SH
LabVIEW Program	MK	MK
Point Source Simulation	SK	SK, OM, MK, BP
Plume Simulation	SH	MK, SH
Future Work and Suggestions		
Interpolation	BP, SK	BP, SK
Budgeting	SK	SK
Hardware	OM	OM
Results and Conclusion		
Filtering Results	SH, MK	OM, MK, SH
Spatial Gradient Based Guidance	MK	ALL
Project Conclusion	SH, BP	ALL
Appendices		
Appendix A	OM, MK, SH	OM, MK, SH
Appendix B	OM, BP	ALL
Appendix C	OM	OM
Appendix D	SK	SK
Paper Editing	SK, BP	SK, BP

Contents

1	Introduction	1
1.1	Project Motivation	1
1.2	Overview of Earlier Relevant MQPs	1
2	Robot	3
2.1	Previous Robots	3
2.2	New Robot	5
2.3	Sensor Platform Design	7
2.4	Structural Stability Simulations	9
2.5	Base Station	13
2.6	Wireless Communication	14
2.7	CO ₂ Sensors	15
2.8	CO ₂ Sensor Calibration	17
2.9	IMU and Vibration Damping	17
3	Dynamics and Controls	19
3.1	Kinematic Equations	19
3.2	Simulink Control Simulation	19
3.3	Controller	21
3.4	Noise in the simulation	22
3.5	Dynamic Model	23
3.6	Extended Kalman Filter	23
3.7	Complete Control Diagram	26
3.8	Low Pass Filter	27
4	Plume Generation System	30
4.1	Flow meter	30
4.2	Solenoid control	30
4.3	LabVIEW Program	31
4.4	Point Source Simulation	32
4.5	Plume Simulations	32
5	Future Work and Suggestions	34
5.1	Interpolation	34
5.2	Budgeting	35
5.3	Hardware	36
6	Results & Conclusions	37
6.1	Filtering Results	37
6.2	Spatial Gradient Based Guidance	38
6.3	Project Conclusion	39
A	Code Appendix	41
A.1	Robot Controller Code	41
A.2	Kalman Filter Algorithm	42
A.3	Robot Arduino Mega Code	43
A.4	Base Station Python Code	55
B	Robot Wiring Diagram	59
C	Connecting Xbee Modules in XCTU Software [11]	60
D	Cost Breakdown	62
E	LabView Operation	63

List of Figures

1	Khepera IV & iRobot Create 2 Comparison. [14] [16]	4
2	Dagu Wild Thumper Robot. [21]	5
3	Dagu Wild Thumper Robot Underside. [21]	5
4	Wheel Motor with Encoder and Wired Connections. [21]	6
5	Wheel Motor Casing. [21]	6
6	Arduino Mega. [3]	6
7	Pololu Simple Motor Controller. [23]	6
8	Xbee Series 1. [11]	7
9	Adafruit BNO055 IMU. [1]	7
10	Three views of the Sensors' Stand Design on top of Robot.	8
11	Three views of the Sensors' Stand Design on top of Rudolf.	9
12	The CoM for the Entire Robot (left) and the CoM of the Sensor Stand Only (right).	9
13	Simulation of Loads on the Edges of the Cross Section of the Design.	10
14	Simulation of acceleration forces on the bottom part of the design. The Left is for 4G and the right is for 0.5G accelerations.	10
15	Solidworks Simulation for an Acceleration of 0.5G.	11
16	Assembled Robot	12
17	April Tag Detection Monitor	14
18	Wireless Communication Between the Robot and Base Station	15
19	COZIR Ambient 10k ppm. [7]	15
20	SprintIR WR 20%. [9]	15
21	SprintIR WR 20% CO_2 Sensor.	16
22	Simulink Block Diagram Representation of the Kinematic Equations (1).	20
23	Simulink Block Diagram Representation of the Kinematic Equations (2).	21
24	Simulink Controller Block Diagram.	22
25	Noise Introduced in the Simulink Model.	22
26	Feedback Loop of the Robot.	23
27	Complete Robot Control Diagram.	26
28	Reponse of a Butterworth Filter. [19]	27
29	FFT of X Acceleration.	28
30	FFT of Y Acceleration.	28
31	FFT of θ Acceleration.	28
32	Schematic Used to Control the Solenoid.	31
33	LabVIEW Block Diagram.	31
34	Old Diffuser.	32
35	New Diffuser.	32
36	Concentration Along X-Axis.	33
37	Concentration Along Y-Axis.	33
38	Concentration Along Z-Axis for a Specified x and y.	33
39	The Robot Trajectory (Actual and Filtered).	37
40	Plume Source Localization Using the Gradient Method.	38
41	XBee Add Device Icon	60
42	XBee Console Icon and Link Icon	61
43	LabVIEW Front Panel Diagram.	63

List of Tables

2.1 Sensor Stands Comparison.	8
D.2 Project Cost Breakdown	62

1 Introduction

1.1 Project Motivation

Many gases exist that, when released, can be extremely harmful to humans as well as difficult to detect. While detection of the presence of these harmful gases is paramount in preventing danger to humans, it is also vitally important to be able to map these plumes of these gas. This mapping allows for their location, size, and movement to be tracked and predicted.

Previous groups of students at Worcester Polytechnic Institute (WPI) have worked to make this concept a reality by attempting to test the effectiveness of an algorithm created to map gas plumes. These groups have moved towards creating an assembly that can generate plumes of gas in a controlled environment as well as a system to prove the concept of the plume mapping theory. This proof of concept has been performed using a wireless autonomous land-based robot, communicating with a base station to sense, locate, and map the gas plumes.

This group aimed to improve and build upon the efforts of the previous students so that we may prove the concept and be able to effectively generate, locate, and map a plume of gas in a controlled environment. The plume generation will be accomplished using a refined version of the system used in previous projects and the locating and mapping will be done by a new robot and updated base station.

1.2 Overview of Earlier Relevant MQPs

The plume MQP has seen many iterations beginning with different teams working on different parts of the overall project. This project began by looking at past MQPs to determine the steps previous teams took in order to further advance this project. We used their reports to gain a firm understanding of the project and to understand what needs to be improved.

MAD1501

In this project, the team began by deciding on the robot that would best suite the needs of tracking a gas plume. The team decided on using a Kephra IV because of its small size (13cm in diameter and 7cm in height) along with multitude of on-board proximity sensors including an accelerometer and a gyroscope. For the gas sensor, the team chose the COZIR Ambient 10K CO₂ Sensor. This sensor was chosen for its low power consumption, low response time, high accuracy, measurement range, and measurement frequency. Next, the team researched wind sensors, but were unable to find a practical sensor for the project, that being an accurate 360 degree sensor for a reasonable price given the project budget. The team then designed and cut a structure to hold the CO₂ sensors. Focusing on the need for a light-weight structure

that was easy to manufacture, the team decided to use acrylic as the main build material. [2]

NAG1501

This team was tasked with finding an appropriate gas and optimal flow rates. The team chose carbon dioxide as the gas to be used in the project because of its low risk factor, low cost, and ease of acquisition and refill. Using simulations of the diffusivity equation, the team found that flow rates between 100 and 500 milligrams per second were the best options for the experiment. [6]

MAD1601

This team created the kinematic and dynamic equations that were to be used for the guidance and navigation of the robot. Sensor noise represents a significant source of error and needs to be accounted for. As such, the team also worked to linearize the kinematic and dynamic models in order to apply an Extended Kalman Filter to the guidance system. The team designed and fabricated a new mount for the four CO₂ sensors along with a new electronic shelf to house components. These additions were mounted on top of the Kephra IV robot. [5]

NAG1602

This team created new simulations to confirm the results of past teams. They found that mass flow rates between 500 and 700 milligrams per second would be most effective, as opposed to the earlier results of 100 to 500 milligrams per second. The team bought a Sierra SmartTrak C100L mass flow controller and a Modern Devices Wind Sensor to improve the gas plume generation and prepare for mobile plume testing. They created a LabView program to control the plume generation system, read CO₂ sensor data, and read wind speed and temperature data. However, the wind sensor was of a lower quality than would be necessary for successful and accurate mobile plume mapping. [4]

MAD1702

This team bought a new robot, the iRobot Create 2, to replace the Kephra IV robot. This decision was made because of the greater dynamic stability of the iRobot Create 2 due to its wider base. The iRobot Create 2 had an inertial measurement unit, a Raspberry Pi, and a Teensy 3.5 board installed to measure acceleration, perform computations, and read sensor data respectively. Additionally, the team formulated a new Extended Kalman Filter for the new robot. The team also put together the plume generation system. Using a pressure regulator, a solenoid, a mass flow controller, and diffuser, they were able to accurately control the flow of CO₂. [12]

2 Robot

The robot makes up the mobile portion of the experimental set up. This unit holds gas sensors to determine plume strength and find the highest concentration of gas in an area. As such, the robot needs to be a mobile, yet stable, platform from which gas sensors can be mounted so that they can move and indicate a gas plume's location. The robot base must hold not only the gas sensors, but also a power source, location sensors, and a method to communicate with the base station.

2.1 Previous Robots

As noted in the Past MQP Overviews section, the earliest groups working on this problem used the Khepera IV robot as the base of their mobile unit. This robot was chosen for its high quality and inclusion of a wide array of features such as ultrasonic distance sensors, omni-directional infrared (IR) distance sensors, wheel encoders, on-board inertial measurement unit (IMU), and embedded Linux controller. However, not all of these sensors could be used due to the size of the tower created to mount the gas sensors. For instance, the tower's arms stretched further from the Khepera IV than the short-range IR sensors could reach. This fact renders them useless as the arms would crash into any obstacle or wall before the sensors would know there was an obstruction there. Additionally, other sensors such as the ultrasonic long-range distance sensors were described by previous groups as unreliable and thus unusable. Finally, the Khepera IV is only 14 centimeters in diameter and only weighs 540 grams. [16] As the earlier groups found out, this extremely small base contributed to a very unstable ground unit. The center of gravity of the assembly was located far above the robot itself thus allowing for a significant swaying action of the tower when the robot moved.

For these reasons, last year's group decided to change the base robot and move away from the Khepera IV, in favor of the iRobot Create 2. Shown in Figure 1, this robot has a much wider and heavier base than the Khepera IV. This base lowered the center of gravity and contributed to a more stable robot assembly. However, while the iRobot Create 2 has wheel encoders, it does not have any of the other sensors the previous robot had or an embedded Linux controller. The last group needed to make up for this fact by mounting a 9 Degree of Freedom IMU on the robot and relying on the wheel encoders more for location measurements.

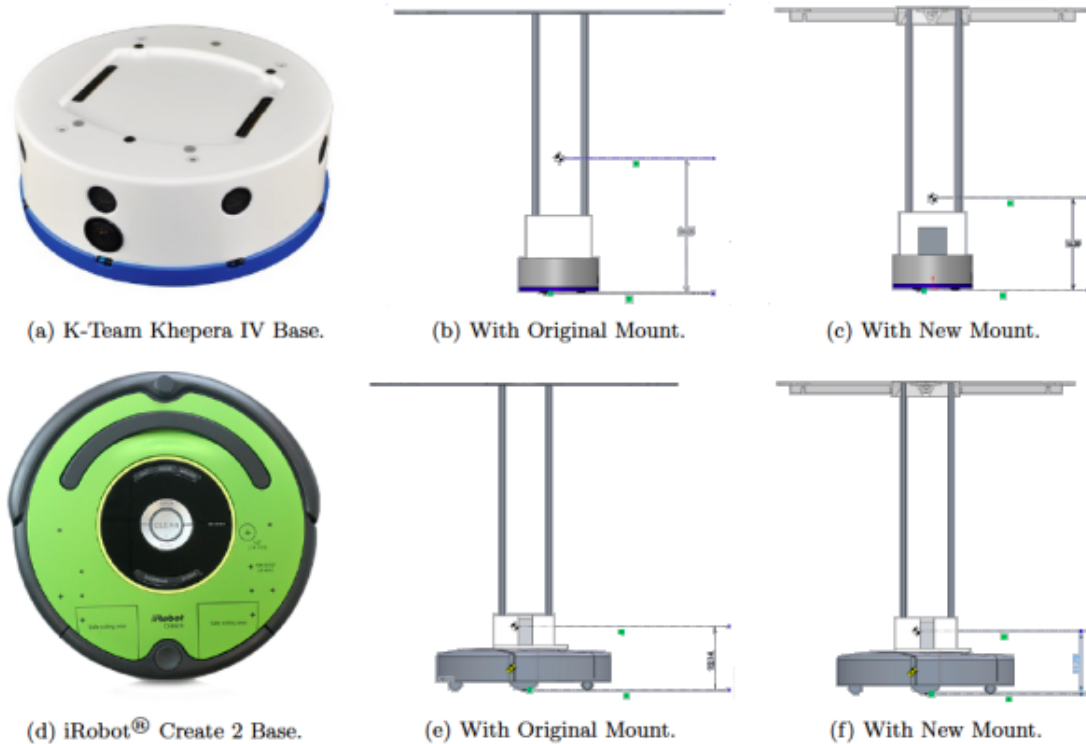


Figure 1: Khepera IV & iRobot Create 2 Comparison. [14] [16]

Additionally, while this robot provided for a more stable base than the Khepera IV, it still suffered from swaying of the tower during movement due to the heavy weight of the tower and the natural motion of the iRobot Create 2. This swaying represents a significant problem as it adds noise to the gas sensor measurements as well as the risk that the tower or assembly falls apart due to the inertial forces exerted on them. The iRobot Create 2 was also in poor condition due to the modifications that the previous group had needed to make to it. For instance, the tray section of the robot that held the IMU and other components was supported with duct tape and supports of the gas sensor tower needed to be removed as they were too heavy.

While both of the previously used robots have strengths, each has a number of weakness as well due simply to their base design. As a result of these short-comings, our group decided that a new robot should be purchased for the project. We wanted to find a robot with strong aspects such as a stable wheel configuration, weight of at least 1.5 kilograms, a center of gravity located low on the robot's z-axis, existing sensors or IMUs, etc. However, rather than find a robot with most of these attributes and then try to work around its weaknesses, our group decided to find a sturdy and stable base that could be easily customized and added to. This search lead us to the Dagu Wild Thumper 4WD All-Terrain Chasis.

2.2 New Robot

The Dagu Wild Thumper 4WD All-Terrain Chassis (Wild Thumper) was the base robot selected for the current project as it fit the description of what was necessary (shown in Figure 2, and Figure 3 below). This robot without any additions weighs 1.9 kilograms and is 28 centimeters by 30 centimeters when the wheels are attached. It consists of 2 millimeter thick anodized aluminum plate with 4 millimeter holes every 10 millimeters. This design provides for strength and stability while also allowing for ease in modifying and attaching accessories or other devices to the robot. The Wild Thumper also has four separate motors, one motor with a 75:1 gearbox for each wheel. This motor gearbox combination can reach a top speed of about 2 miles per hour. As will be shown in the next section, the weight and low-laying design of the robot is ideal for our gas sensor tower design in that it contributes a low center of gravity for the entire assembly and a solid base to mount the stand from. [21]



Figure 2: Dagu Wild Thumper Robot. [21]

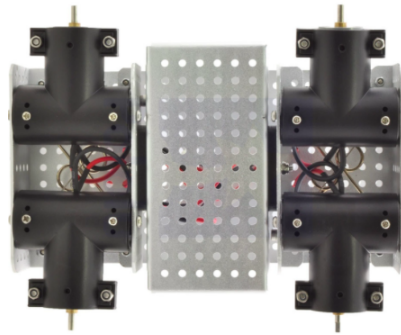


Figure 3: Dagu Wild Thumper Robot Underside. [21]

In terms of additional accessories and modifications for the robot, the design allowed for easy alterations to meet the needs of the group. For instance, the wheel motors that come standard on the Wild Thumper do not have encoders, which are vital for the robot to understand its location. As such, the group purchased wheel motors with a 75:1 gearbox that also had encoders. The modification for switching these motors with the existing ones was fairly simple in that all the group did was sever the connections from the existing motors to the robot, remove them from the motor housing and then replace them with the new motors and connect them to the robot (seen below in Figures 4 and 5).

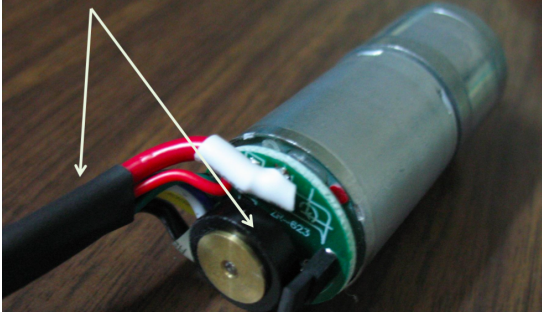


Figure 4: Wheel Motor with Encoder and Wired Connections. [21]



Figure 5: Wheel Motor Casing. [21]

In addition to replacing the wheel motors, the group also needed to add computing and controller elements to the robot so that it was able to function autonomously and communicate with the base station. The robot is using an Arduino Mega (Arduino), seen in Figure 6, as it has enough pin locations to handle the inputs of the IMU, 4 motor encoders, 2 motor controllers, Xbee, battery, and 4 gas sensors. The motor controllers being used are Pololu Simple Motor Controllers 18v7, seen in Figure 7. Each of these is paired with two of the wheel motors on the Wild Thumper. This setup was arranged so that one board was controlling a single side of the robot and the other board the other side. For instance, one controls the left front and left rear wheel motors while the other controls the right front and right rear motors. This setup will contribute to the differential method of steering used by the robot, which will be discussed in a later section.



Figure 6: Arduino Mega. [3]

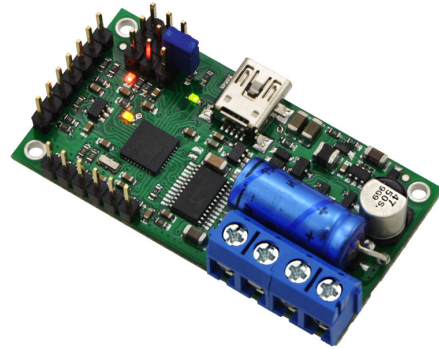


Figure 7: Pololu Simple Motor Controller. [23]

In order to communicate with the base station, the robot uses the Xbee Series 1 (Xbee), seen in Figure 8 as the group chose to not use the Raspberry Pi the previous

groups had. The Xbee was chosen as it has the ability to send and receive data over a wireless connection. As the Extended Kalman Filter’s computations are carried out on the base station, values from the IMU (shown in Figure 9) are sent over Wifi. This will be discussed further in Section 2.6. The complex and numerous wired connections between all of these devices can be found in Appendix B.



Figure 8: Xbee Series 1. [11]



Figure 9: Adafruit BNO055 IMU. [1]

This new robot, coupled with the modifications and additions made to it during this project, allowed for the minimization of the problems that previous groups have experienced as a result of their base robot. The final physical piece of the robot assembly is that of the sensor stand and the addition of gas sensors.

2.3 Sensor Platform Design

In order for the robot to carry the CO₂ sensors, the previous years’ teams designed cross-like sensor stands which were attached to the robot. The sensor stand was designed in this manner so as to keep the sensors at a distance from the ground to avoid the floor bias or plume recirculation close to the ground. The length of the cross design was chosen to keep the sensors apart from each other so that their readings are independent. The old designs included an array of four CO₂ COZIRTM that were arranged in a cross. The design depicted in this figure is a machined metal plate made out of Aluminum. Due to the high material density of 2800kg/m³ the design was relatively heavy, especially the aluminum cross, which elevates the center of mass of the structure and makes the robot oscillate in movement.

To counteract this effect, the current team designed a new structure made completely out of acrylonitrile butadiene styrene (ABS plastic), which has a density of 1500kg/m³. The stand that is depicted in Figure 10 was designed so that it has a reduced weight while being strong enough to support the sensors. It also contributes to a very low center of mass, lowering the oscillation of the robot.

In order for the stand to be able to be 3D printed, the design was split into 3

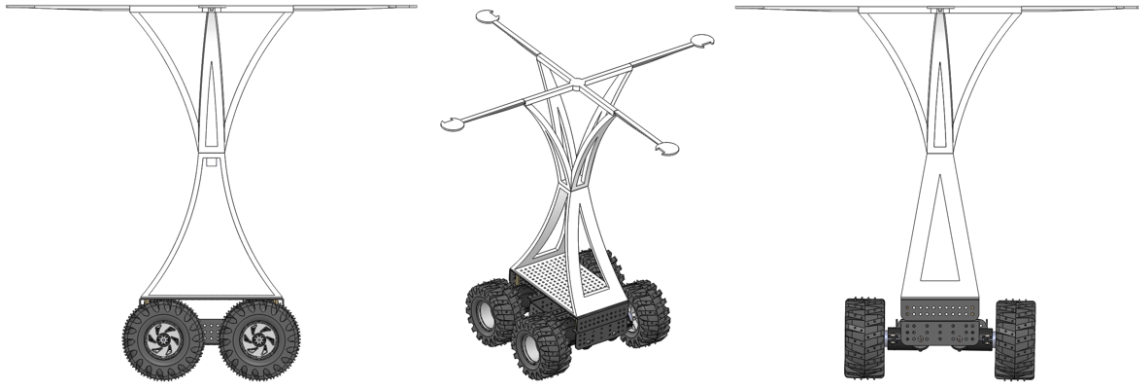


Figure 10: Three views of the Sensors' Stand Design on top of Robot.

sections, the bottom half, the top half and the upper extensions where the sensors will be placed. The specifications of the new sensor stand are included in the table below along with a comparison with the old MQP's design.

	MAD 1701	New Design	% Increase/Decrease
Total Weight	1 kg	0.642 kg	-36%
Total Height	0.5 m	0.45 m	-10%
CoM Height	0.36 m	0.22 m	-39%
Sensors' Distance	0.5 m	0.55 m	+10%

Table 2.1: Sensor Stands Comparison.

As shown in the table, there is a significant decrease (-36%) in total weight of the structure and the center of mass is shifted downwards (-39%). This design makes the robot much more stable and greatly reduces oscillation resulting from the previous top-heavy stand. The sensors' distances were increased slightly to make sure that the readings are independent from one another. The only drawback of the design is a slight (-10%) decrease in the height of the structure, which will not affect the experiment as our new robot is taller so it will make up for this small height loss.

Below are detailed sketches of the robot including the sensor stand. Figure 11 shows the dimensions of the whole structure including the robot and Figure 12 shows the centers of mass (CoM) of the sensor stand and the entire robot. We additionally have to take into account that the total CoM will be shifted slightly due to the instillation of the circuit boards and sensors. The change however, is insignificant dynamic analysis continued to be accurate.

Before proceeding to 3D Printing the stand, the group ran some SolidWorks simulations to verify that the design meet the necessary requirements.

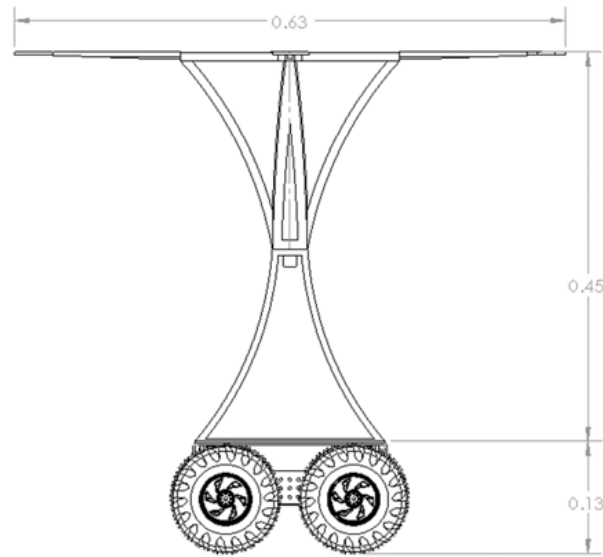


Figure 11: Three views of the Sensors' Stand Design on top of Rudolf.

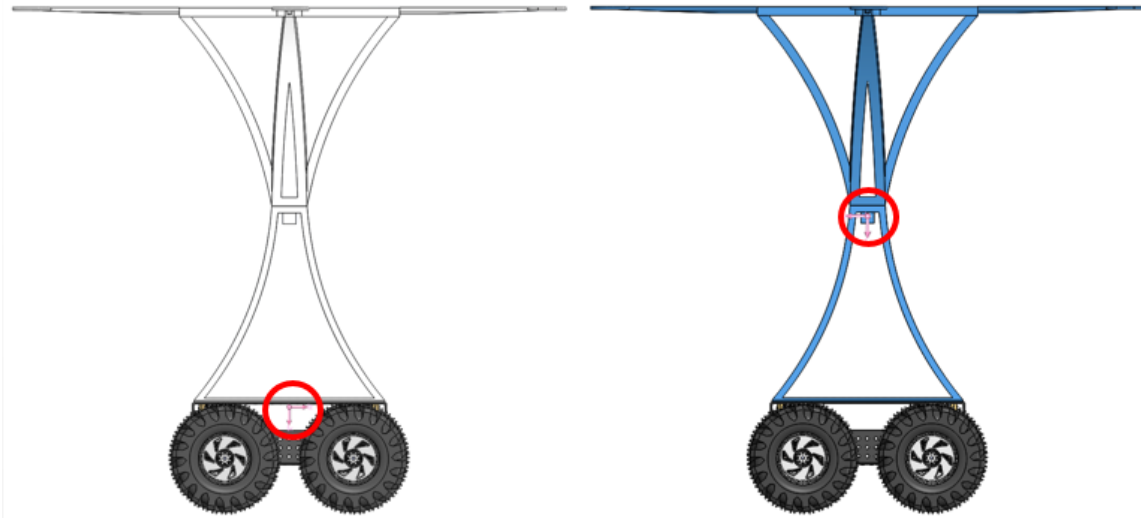


Figure 12: The CoM for the Entire Robot (left) and the CoM of the Sensor Stand Only (right).

2.4 Structural Stability Simulations

The structural stability simulations were run on SolidWorks' Structural Analysis Software. By defining the material that is going to be used to 3D Print the sensor stand parts the group was able to first simulate the stresses on the edges of cross that makes up the upper portion of the stand. Loads of 70 grams were placed in the simulator where the sensors are attached as shown in Figure 13.

The simulation showed that the stresses are insignificant (nothing close to fracture

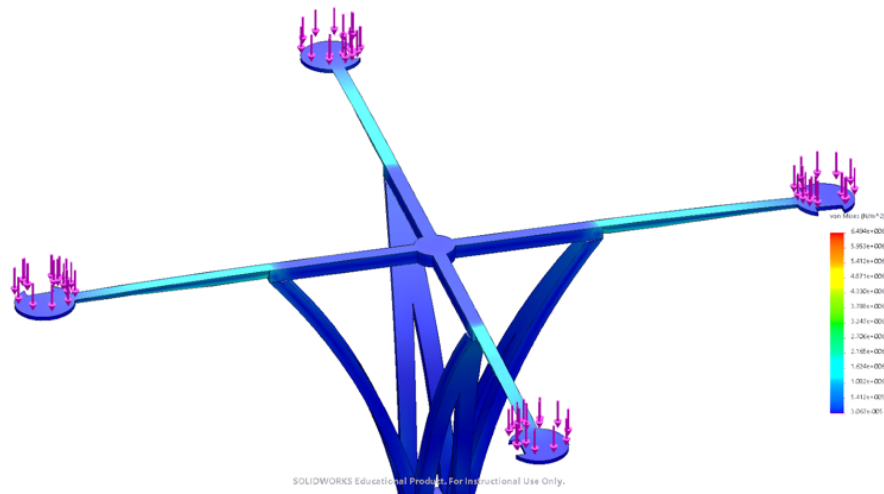


Figure 13: Simulation of Loads on the Edges of the Cross Section of the Design.

toughness) and the deflection at the edge of the design is at maximum 3mm, which is insignificant. Here it is important to note that a single sensor's weight is approximately 20 grams and not 70 grams as was input in the simulation, thus there is certainty that the design is strong and will not deflect in reality.

The second test conducted was an acceleration test. The previous teams reported that when the robot accelerated (started moving or was slowing down) the stand was very likely to oscillate. Using Solidworks, a force was applied equivalent to an acceleration of 4G's to the very bottom part of the structure as shown on the left part of Figure 14.

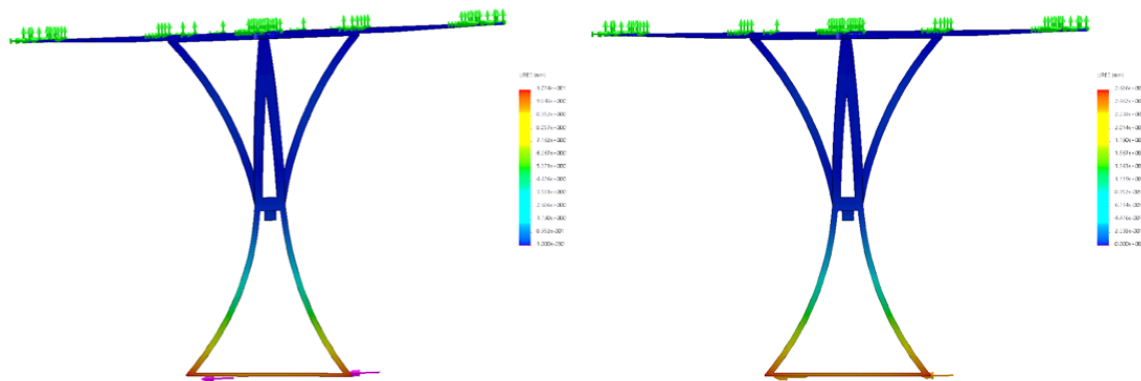


Figure 14: Simulation of acceleration forces on the bottom part of the design. The Left is for 4G and the right is for 0.5G accelerations.

The deflection of the top section was approximately 1mm, which is insignificant but still present as shown in the left picture (top part slightly tilted on the left). Yet again, the structure will never accelerate at a value near 4G, so the same test was

run for 0.5G (still a large value for the project's application). For the second test the deflection was insignificant as observable on the right section of Figure 14.

Lastly, the group conducted one more simulation for the acceleration. For this test the Accerelation Simulation Tool that Solidworks has was used. As shown in Figure 15 the stresses are insignificant for most of the structure (some barely observable on the middle part) and the deflection is again insignificant (not observable) for an acceleration of 0.5G.

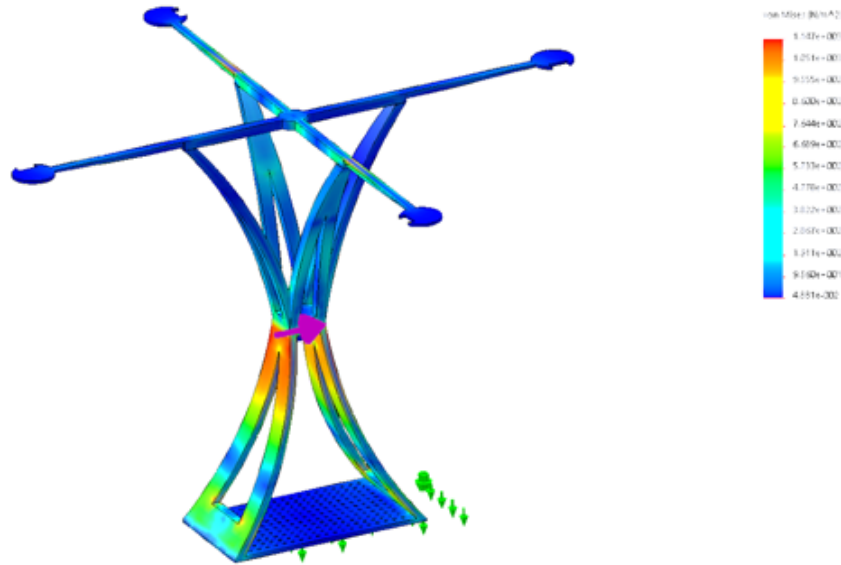


Figure 15: Solidworks Simulation for an Acceleration of 0.5G.

Here it is important to note that these simulations were run only for the stand itself and not the robot. These simulations show that the new stand is a much more well thought out design than the ones used by the previous MQP teams for the reasons discussed in the previous section.



Figure 16: Assembled Robot

The team proceeded to 3D Print this design after performing the successful simulations described above and gaining approval from Dr. Demetriou. The printer used was a Dimension SST 1200es at Higgins labs, WPI, a high quality printer with a precision of 0.006 in. The printing was accomplished over three days (one day for each of the three parts) and another day was used to dissolve the supporting material. A small change was made between the original design and the final print. Instead of printing a solid base for the top section, the piece was split into two smaller parts vertically which omitted some of the plastic used on the base. This alteration was

made to ensure a safer printing process that would not fail in addition to reducing the plastic used. This alteration additionally lowered the weight of the stand. The structural integrity of the printed design was proven through the same simulations as the previous design. The outcome was satisfactory and as rigid as expected. In Figure ?? we can see the entire robot assembly including the base robot, the 3-D printed sensor stand (white), the CO_2 sensors attached on top and the wiring that goes down and into the body of the robot where the arduino is located. Here it is important to note that the device attached on the very center of the base platform is the IMU, which will be discussed further in Section 2.9.

2.5 Base Station

In order to command both the robot and the plume generation system a "Base Station" computer was devised. Previous MQP teams used a Ubuntu Linux desktop as it is the most advantageous operating system for programming. Unfortunately this posed many issues as WPI could not provide licensing for *LabViewTM* on Linux. These issues prompted a reconfiguration of the base station.

The new base station runs the *Windows 7TM* operating system as it is the most compatible with any software. It has *LabViewTM* installed for the plume generation, *MatlabTM* installed for plume modeling, and a *UbuntuTM* Virtual Machine to run the Camera Trajectory Sensing. A wireless dongle is connected to the computer to connect to the Robot's Router onboard. This allows for data transfer between the robot and the base station.

To verify that the Extended Kalman filter is outputting accurate results, a camera was mounted to the ceiling facing downwards to record the pose and position of the robot at every frame. This was done by the previous MQP teams because the previous year's Kalman Filters would not work, therefore the robot had to somehow be located. In the case of the MAD 1702 MQP, the camera was mostly used to calibrate the already working Kalman Filter and compare its precision. [12] The plot of the trajectory output from the camera can be compared with the plot of the trajectory output of the Extended Kalman Filter to check its accuracy. Trajectory is plotted using MIT's AprilTag Library. Tag 36h11 (similar to a QR code) was printed on a paper and mounted on top of the robot directly in the center of the CO_2 Sensor Stand. In order to calculate the area of the experimentation field the lens equation was used,

$$A = 2 \tan^{-1} \left(\frac{W_{lens}}{2L_{focal}} \right), \quad (1)$$

where W_{Lens} is the lens width and L_{Focal} is the focal length. Lens width is defined as the diameter of the camera's lens actual length and the focal length the distance from the lens to its focus. Using the above equation, as well as empirical data, the

dimensions of the camera's field of view were determined. These calculations ensured that an experiment was never done where the camera would not be able to collect data.

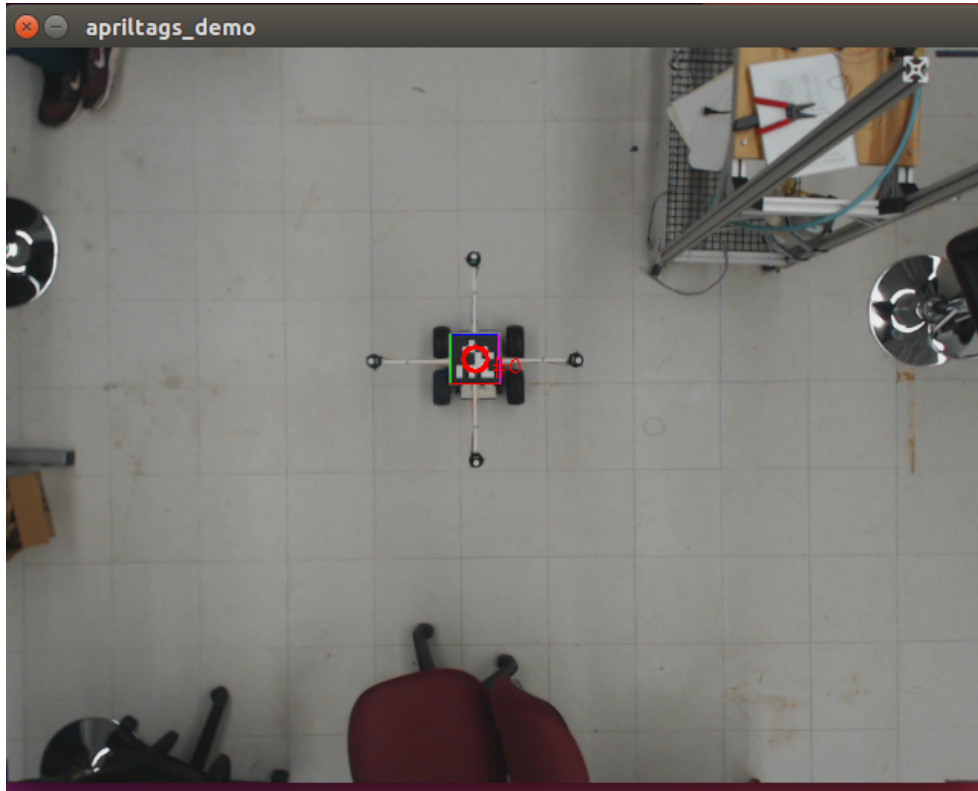


Figure 17: April Tag Detection Monitor

2.6 Wireless Communication

As a *Raspberry PiTM* is no longer being used by the robot, an alternative wireless communication method from the base station to the robot and back is necessary. A convenient wireless communication hardware suitable for use with the *Arduino Mega* is the Xbee Series 1 Wireless Module. [11] Two wireless Xbees can be configured with a one time installation of firmware (See Appendix C) and become treated as a wireless serial connection. This setup is especially advantageous for this application because it allow for the use of the PySerial library to easily parse information sent from the Arduino Mega to the Base Station. Once the information is parsed, the data is fed through the filters and a trajectory is outputted.

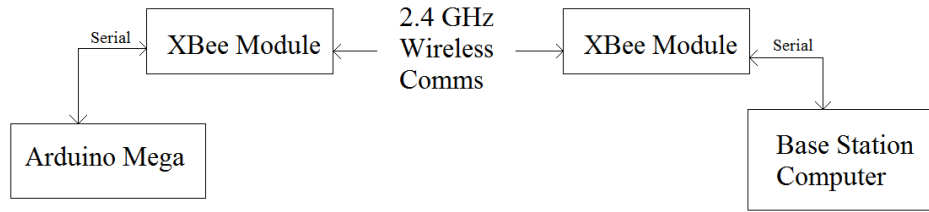


Figure 18: Wireless Communication Between the Robot and Base Station

2.7 CO₂ Sensors

One of the most important tasks of the robot will be to record CO_2 concentration data from the sensors so that it can both manipulate that data to determine its next point to navigate to, as well as provide data for plume modeling on the base station. Previous projects used the *COZIRTM* Ambient 10,000 ppm CO_2 sensor, seen below in Figure 19. However, in early experiments it became clear that these sensors were inadequate for the needs of the project. The settling time, or time the sensor needs to return to an ambient CO_2 reading after a rise or fall in adjacent CO_2 levels, was far too long. For the *COZIRTM*, the settling time is listed as between 30 seconds and 3 minutes. The group found the 3 minute bound to be the more accurate estimation. This length of settling time prevented the running of a smooth experiment as the robot would turn or change direction, only to continue turning the same way instead of fixating on the CO_2 source. The turning error resulted from the saturation of the sensors and their inability to return to an ambient reading between movements. As such, explored other possible sensors were explored, finally settling on the *SprintIRTM* WR 20% CO_2 Sensor in Figure 20.



Figure 19: COZIR Ambient 10k ppm. [7]



Figure 20: SprintIR WR 20%. [9]

Not only does the *SprintIRTM* have a settling time of only four seconds, which allowed for experiments to be performed smoothly, but also a higher capability of CO_2 sensing. As shown in Figure 21 where the two sensors responses are

compared, the Sprintir reads the CO_2 concentration very fast and peaks at the concentration, whereas the COZIR takes time to respond, and by the time it responds the concentration has already been reduced significantly. The *SprintIRTM* can sense up to 200,000 ppm while the *COZIRTM* could only sense up to 10,000 ppm. Simulations have shown that near the plume generation system, CO_2 levels can reach 85,000 ppm. Thus, these new sensors allow the experiment to be performed extremely close to the plume generation. Additionally, the sensor has the same pin inputs and draws the same voltage from the system as the *COZIRTM* so no modification needed to be made to the wiring, coding or power input. Last but not least, although the *SprintIRTM* is more expensive as it costs \$155, the cost difference is not great and the company that provides it is the same. As such, to substitute the *COZIRTM* with the *SprintIRTM* sensors was a simple process and significantly improved our experiment quality.

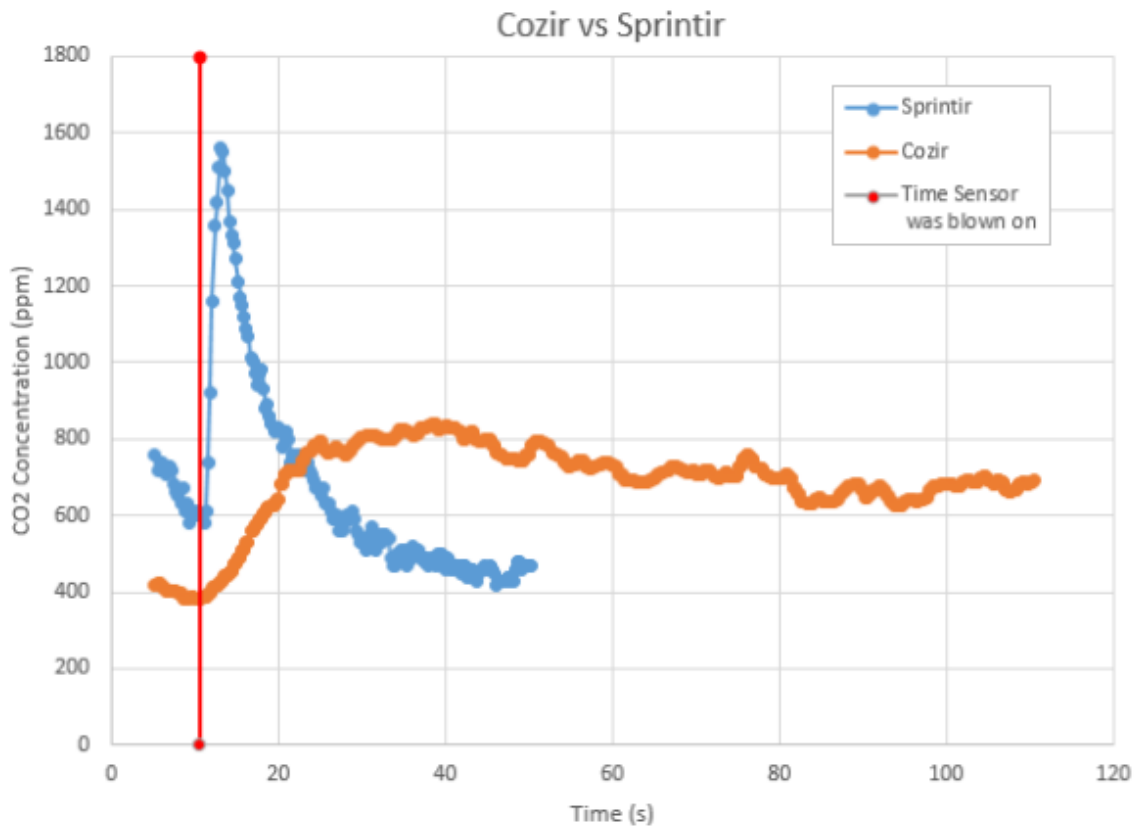


Figure 21: *SprintIR WR 20% CO₂ Sensor.*

2.8 CO₂ Sensor Calibration

Similar to most measurement equipment, the CO₂ sensors need to be regularly calibrated in order to obtain accurate measurements of CO₂ concentrations. Without regular calibration before each experiment, it is impossible to obtain accurate results. In order to calibrate, Github user Roder's COZIR CO₂ library is used. The sensors are calibrated in fresh air for experiments in unknown concentrations. Fresh air is estimated to have a CO₂ concentration of 450ppm. This can be accomplished by sending the serial command "Grn". The CO₂ sensors also have an option for calibration in a known CO₂ concentration. This can be accomplished by sending the serial command "X#rn" where "#" is the value in ppm of the known CO₂ concentration. [8]

2.9 IMU and Vibration Damping

In previous sections it was noted that in order to locate our robot and find its position accurately, a camera with an April-tag can be used. In real world applications though, where the robot operates outside the lab or even outside the range of the camera, a different method of tracking the exact position of the robot is required.

As is standard in many engineering controls applications, an Inertial Measurement Unit (IMU) was used. An IMU is an electronic device that measures and reports a body's specific force and angular rate, using a combination of accelerometers and gyroscopes. For our robot we are using an *Adafruit BNO055* IMU similar to Figure 2.9, which we attached at the measured center of our robot's body, seen in Figure ???. The IMU needed to be attached at the center of the robot in order for the readings to be as accurate as possible. It was also assumed that the center of mass is located at the center of the robot as it is set up in such a manner that it is symmetric on both x and y axis and the weight is distributed evenly between these axis.

While running our simulations, it came to our attention that the IMU readings were very noisy. While the robot is moving the motors, as expected, cause vibrations. These vibrations are carried throughout the entirety of the robot reaching the IMU and causing it to vibrate as well. To obtain readings that are accurate, these vibrations needed to either be eliminated or the noise from the measurements filtered out using software which will be discussed on Section 3.4. The group decided to use both methods to mitigate as much noise as possible.

In order to eliminate the vibrations, the group used Poron foam. This material is made out of Polyurethane, which is a polymer known for its vibration absorbing characteristics. A 12x12 inch, extra soft, wear-resistant and quick-recovery Polyurethane foam sheet, which is half an inch thick, was purchased. A 3x2 inch piece was then cut to fit it on top of the robot base platform. Although soft, it was possible to screw the IMU on top of the Poron and then glue the foam on top of the piece of wood initially screwed to on top of the robot (as seen in Figure ??). It was

important to make sure the screws did not go through the foam and touch the piece of wood, as this would transmit vibrations to the IMU, making our damper useless. In the future the excess foam that was purchased could be placed between the motors and the robot body so that the system's vibrations can further damped.

3 Dynamics and Controls

3.1 Kinematic Equations

To begin deriving the equations of motion that were used for the simulations, the group had to identify what type of steering the robot would be using. The two methods of steering are the Ackerman steering where the wheels of the vehicle are able to turn (cars use this method) and differential steering where the vehicle is being turned due to a difference in wheel speeds on one side of the vehicle compared to the other. The Dagu Wild Thumper chosen as the base robot uses differential steering, thus the group used equations which involve different wheel velocities for the left pair of wheels (V_L) and the right pair of wheels (V_R).

x can be defined as the x-axis position of the CoM of the Robot and y as the y-axis position on the inertial frame so that \dot{x} is the velocity of the CoM in the x-direction and \dot{y} the velocity of the CoM in the y-direction. Let ψ be the heading of the vehicle (angle between x-axis and its total velocity vector V_T) such that:

$$\dot{x} = \frac{V_L + V_R}{2} \cos \psi, \quad (2)$$

$$\dot{y} = \frac{V_L + V_R}{2} \sin \psi, \quad (3)$$

$$\dot{\psi} = -\frac{V_L - V_R}{W}, \quad (4)$$

where W is the width of the robot (the distance between the left and right wheels). Note here that the cosine and sine on the equations above are transformation parameters from body-centered coordinates to inertial coordinates.

3.2 Simulink Control Simulation

To simulate the control of the robot we used Simulink. The first step in doing so was to create Simulink block diagrams of the equations derived above. For Equations 2 and 3 we created the following block diagram:

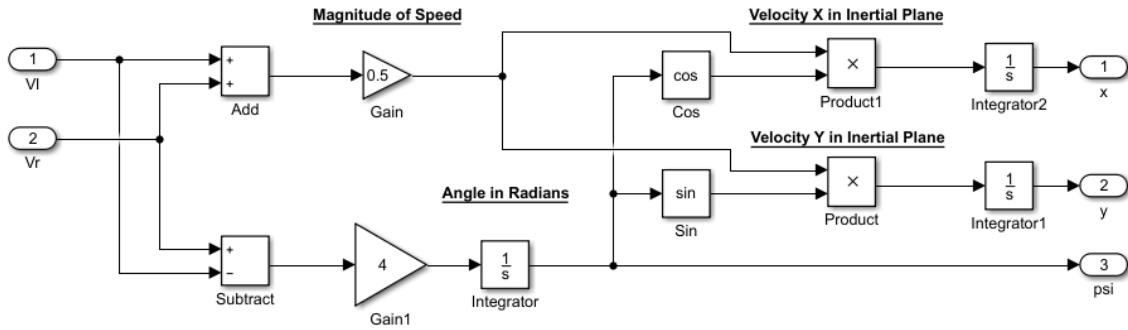


Figure 22: Simulink Block Diagram Representation of the Kinematic Equations (1).

The point of making the block diagram is to simulate the way the robot was going function. The inputs that are going to be given in the real life experiment are V_L and V_R , thus on the block diagrams these are the inputs. By adding them together, then dividing them by two or the vehicle width and then multiplying them with cosine and sine ψ respectively, it is possible to reconstruct the equations derived in Section 3.1. Next, these are integrated (integration is the $1/s$ block) so that x , y and ψ are yielded as outputs.

The block diagram in Figure 22 will be referred to as Plant 1, which is what the Robot will be doing. It will get the wheel velocities as inputs and then translate them to x , y and ψ outputs. These outputs will be read by the wheel encoders and the IMU, devices which are subject to noise. To simulate the noise, the block diagram depicted in Figure 23 was created.

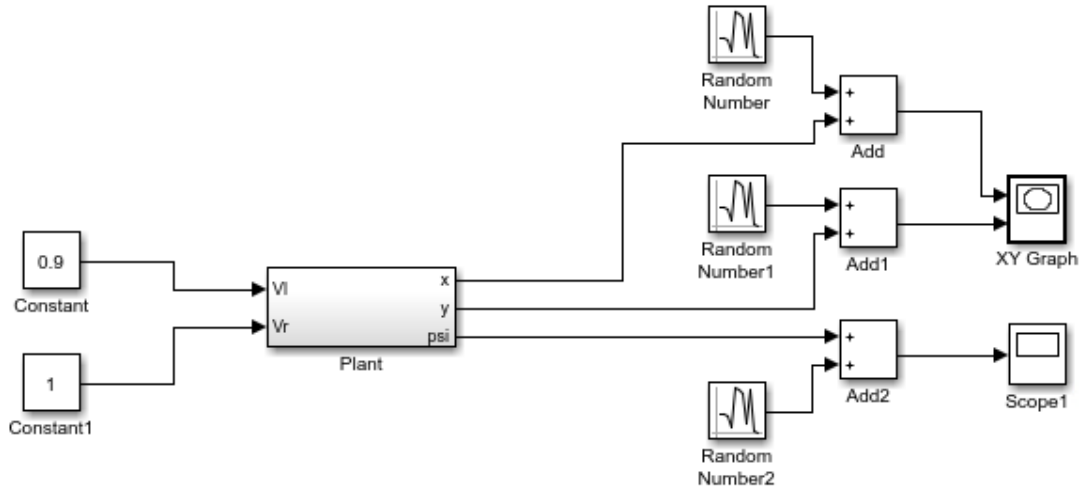


Figure 23: Simulink Block Diagram Representation of the Kinematic Equations (2).

Shown in the figure above, the entire block diagram in Figure 22 is replaced with the larger block named Plant. Because the outputs are subject to noise, on this diagram, all three output readings are taken and noise is added in the form of random numbers (note the Random Number Blocks).

3.3 Controller

A controller was created in Simulink to simulate what the controller would do in real world settings. The controller works as follows: the controller outputs a constant differential V_L and V_R to rotate the robot until the heading angle of the robot is within 5 degrees of the absolute heading between the robot and the point that the robot is trying to reach. Once the controller detects that it is within 5 degrees, the controller then sets V_L and V_R equal to a certain value and gives it to the plant. The controller is given a requested x and y position, along with the estimated state from the Kalman filter. The outputs are V_L and V_R values that are given to the plant. A model of the controller is shown below and the code for the controller can be seen in Appendix A.4. This model was initially based on the Pure Pursuit tracking algorithm, but was then changed to the current stop and turn model due to simplicity. [10]

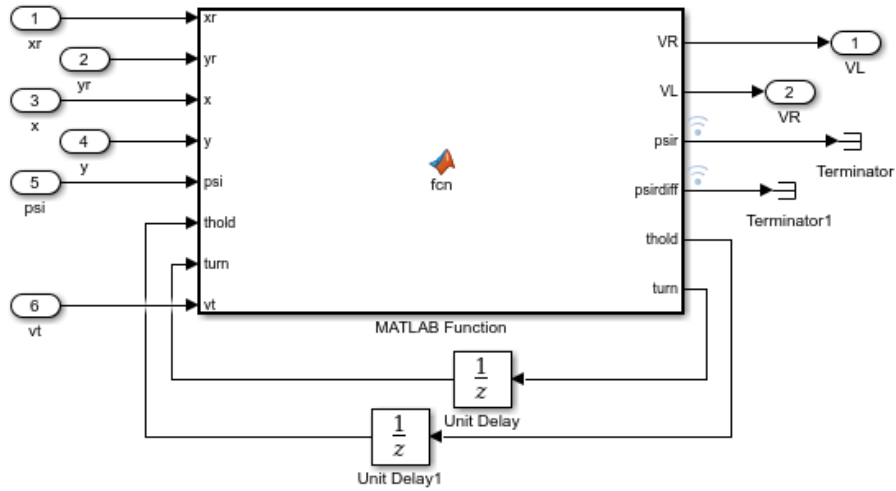


Figure 24: Simulink Controller Block Diagram.

3.4 Noise in the simulation

To simulate receiving data from accelerometers placed on the robot, the next step in the Simulink block is to take the second derivative of the x and y produced by the plant, and the first derivative of phi given by the plant. Random gaussian noise is then added to each value. These values are then given to the Extended Kalman Filter as measurements.

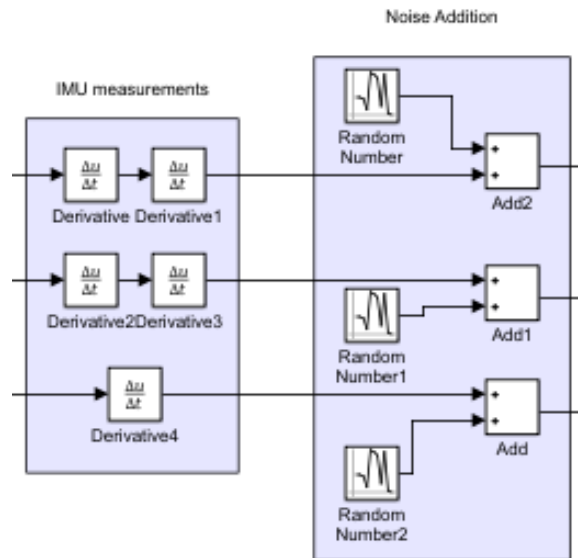


Figure 25: Noise Introduced in the Simulink Model.

3.5 Dynamic Model

Following completion of the kinematic equations and Simulink, a dynamic model was created for the robot. The model begins with the Robot's controller. This software controller takes measurements from the robot (x , y , ψ , and concentration sensor values) and calculates V_L and V_R from the differential steering model. V_L and V_R are input into the kinematic equations 2, 3, and 4 to determine x , y , and ψ . Since these measurements will be noisy, an Extended Kalman Filter will need to be implemented to linearize an estimate of the measurements. The measurements will then be input back into the controller, establishing the Feedback Control Loop seen in Figure 26

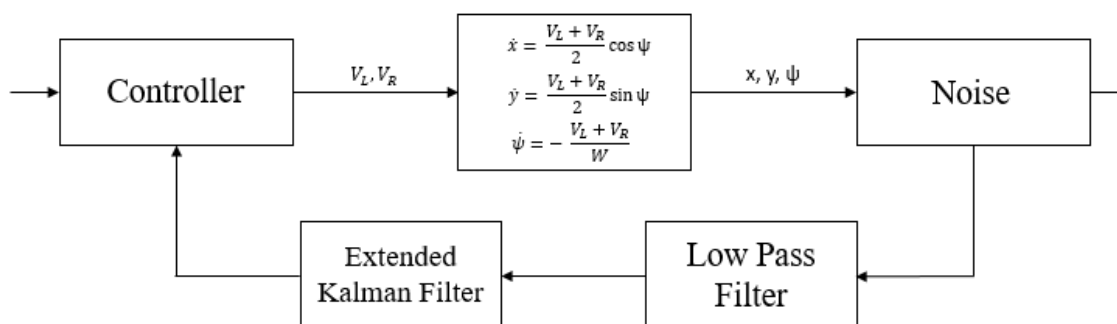


Figure 26: Feedback Loop of the Robot.

3.6 Extended Kalman Filter

The Kalman Filter (KF) is an algorithm that uses a series of measurements observed over time, which contain statistical noise and other inaccuracies, and produces estimates for the states of the system, "cleaning" out the noise of the measurements. The filter is named after Rudolf E. Kálmán, one of the primary developers of the filter and the theory behind it. The nonlinear version of the KF is referred to as the Extended Kalman Filter (EKF). [13] In the case of the current robot, the group used the EKF to clear out the noise from our IMU and encoder measurements. In order to begin the analysis, the group began with continuous time state equations:

$$\dot{x} = \begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} v(t) \cos(\theta(t)) \\ v(t) \sin(\theta(t)) \\ \theta(t) \end{bmatrix} \quad (5)$$

The goal is to discretize this state equation in order to run the algorithm in discrete time. To perform the time discretization the difference quotient is used

$$\dot{x}(t) \approx \frac{x(t + \Delta T) - x(t)}{\Delta T}, \quad (6)$$

Which can be written as

$$x(t + \Delta T) = x(t) + \Delta T \dot{x}(t). \quad (7)$$

Using the kinematic equations (2), (3) and (4) discussed in the section above, ΔD and $\Delta\theta$ can be defined as

$$\Delta D = \frac{V_{L,k+1} + V_{R,k+1}}{2} \Delta T, \quad (8)$$

$$\Delta\theta = \frac{V_{L,k+1} - V_{R,k+1}}{W} \Delta T. \quad (9)$$

Therefore the discrete time state equation becomes [20]

$$x_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \Delta D \cos(\theta_k + \Delta\theta) \\ y_k + \Delta D \sin(\theta_k + \Delta\theta) \\ \theta_k + \Delta\theta. \end{bmatrix} \quad (10)$$

As described above, the state equations are non-linear because they involve sinusoids. In this case the Extended Kalman Filter needs to be used and the state equations linearized in discrete time. Once the equations are linearized, a filter gain can be determined based off of the process error covariance and the conversion of measurement to state. The process error and the states are then updated with the measurements, innovation matrix, and Kalman gain. To begin, define:

$$F_k = x_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}, \quad (11)$$

and

$$u_{k+1} = \begin{bmatrix} \Delta D \\ \Delta\theta \end{bmatrix}, \quad (12)$$

To time linearize the model it is necessary to obtain two matrices, A_k and B_k . These matrices are the Jacobian of the state matrix in respect with the states and the control input respectively such that: [20]

$$A_k = \nabla F_k|_{x_k} = \begin{bmatrix} 1 & 0 & -\Delta D \sin(\theta_k + \Delta\theta) \\ 0 & 1 & \Delta D \cos(\theta_k + \Delta\theta) \\ 0 & 0 & 1 \end{bmatrix}, \quad (13)$$

and

$$B_k = \nabla F_k|_{u_k} = \begin{bmatrix} \cos(\theta_k + \Delta\theta) & -\Delta D \sin(\theta_k + \Delta\theta) \\ \sin(\theta_k + \Delta\theta) & \Delta D \cos(\theta_k + \Delta\theta) \\ 1 & 0 \end{bmatrix}. \quad (14)$$

Therefore the equation for the future state at time $k+1$ becomes:

$$x_{k+1} = A_k x_k + B_k u_k + n_k, \quad (15)$$

Where n_k is the process noise. The last step before putting together the EKF algorithm is to define the measurement model. As discussed, the EKF inputs are the IMU measurements. The IMU reads acceleration on the x and y direction and angular velocity with respect to the center of the IMU (and the center of mass). Therefore, a Measurement Model is needed to convert the IMU accelerations and angular velocities to states (x, y, θ) . To do that the group used a pseudo-integration method. Since $\Delta T < 1 \text{ sec}$ the observation matrix C can approximate the integrals to $1/\Delta T$ such that our measurements are equal to

$$\begin{bmatrix} \ddot{x}_{measured} \\ \ddot{y}_{measured} \\ \dot{\theta}_{measured} \end{bmatrix} = \begin{bmatrix} \frac{1}{\Delta T^2} & 0 & 0 \\ 0 & \frac{1}{\Delta T^2} & 0 \\ 0 & 0 & \frac{1}{\Delta T} \end{bmatrix} \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} + \begin{bmatrix} noise_{\ddot{x}} \\ noise_{\ddot{y}} \\ noise_{\dot{\theta}} \end{bmatrix}, \quad (16)$$

So that the measurement model becomes:

$$Z_{k+1} = C x_{k+1} + noise \quad (17)$$

Now that all of the required matrices have been defined, the EKF iterative algorithm can be put together, which is going to be split in two parts, the Predict Stage ($k+1|k$) and the Update Stage ($k+1|k+1$). Every iteration of the algorithm includes:

the **Predict Stage**

$$P_{k+1|k} = B P_{k|k} B^T + Q \quad (18)$$

$$\hat{x}_{k+1|k} = \hat{x}_{k|k} + \Delta x_{k|k} \quad (19)$$

and the **Update Stage**

$$S_{k+1|k+1} = CP_{k+1|k}C^T \quad (20)$$

$$K_{k+1|k+1} = P_{k+1|k}C^T S_{k+1|k+1}^{-1} \quad (21)$$

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1|k+1}(Z_{k+1|k+1} - Cx_{k+1|k}) \quad (22)$$

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1|k+1}S_{k+1|k+1}K_{k+1|k+1}^T \quad (23)$$

In the algorithm above the Innovation Matrix is denoted S, P is the Predicted Covariance Estimate, K is the Kalman Gain and Q the Process Noise Covariance Matrix. A Matlab example of this algorithm is presented in Appendix A.2

3.7 Complete Control Diagram

By putting together everything mentioned above, a complete Simulink block diagram is formulated, presented below. The block diagram includes the controller, the plant, the IMU measurements, the noise additions, the Extended Kalman Filter and the required scopes in order to plot the outcomes.

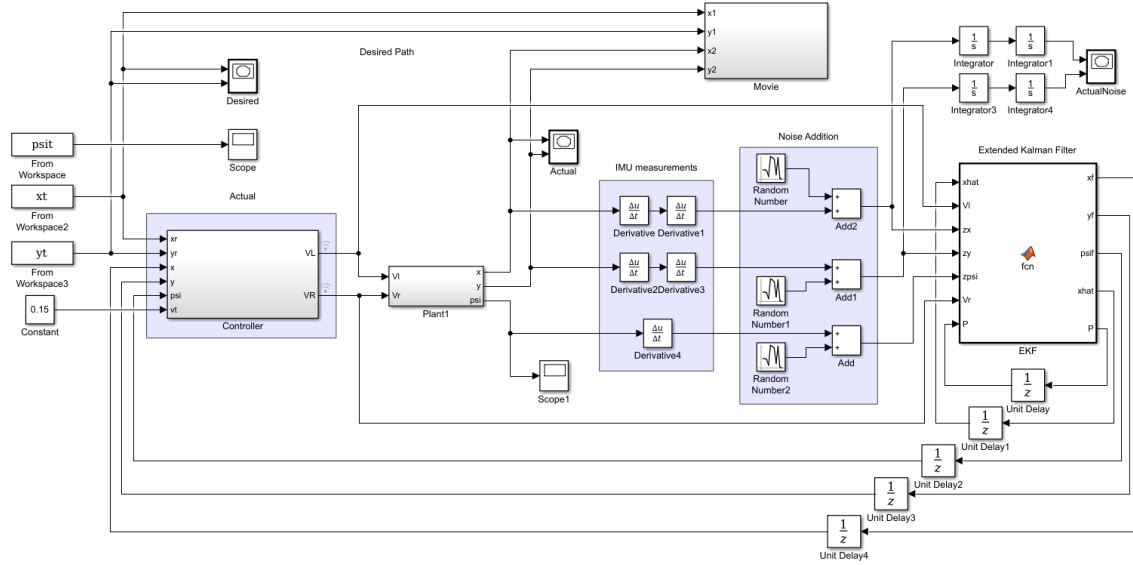


Figure 27: Complete Robot Control Diagram.

3.8 Low Pass Filter

The data received from the Inertial Measurement Unit always has a certain amount of noise associated with it. As the Kalman Filter was not successfully removing all of the noise, the group decided to use a Low Pass Filter to remove some of the higher frequency noise prior to sending the data through the EKF thus improving measurements.

A low pass filter is a filter that passes signals with a frequency lower than the cutoff frequency and attenuates signals higher than the cutoff frequency. The purpose of our low pass filter was to remove the higher frequency noise received from the IMU and also let the low frequency information pass through to the Kalman Filter. In continuous time, low pass filters may be described in terms of the Laplace transform and Bode plots. Equation (24) is the Laplace notation for a first order low pass filter, otherwise known as a Butterworth filter

$$H(s) = \frac{1}{1 + \frac{s}{\omega_c}}, \quad (24)$$

where ω_c is the cutoff frequency, and s is the Laplace transform variable. [15] In the discrete-time domain, a difference equation is used

$$y[n] = ay[n - 1] + bu[n], \quad (25)$$

where $y[n]$ is the output in discrete-time, $y[n-1]$ is the previous output, $u[n]$ is the input, and a and b are filter coefficients that are found when designing the filter. [26] The magnitude response of a Butterworth filter is shown below.

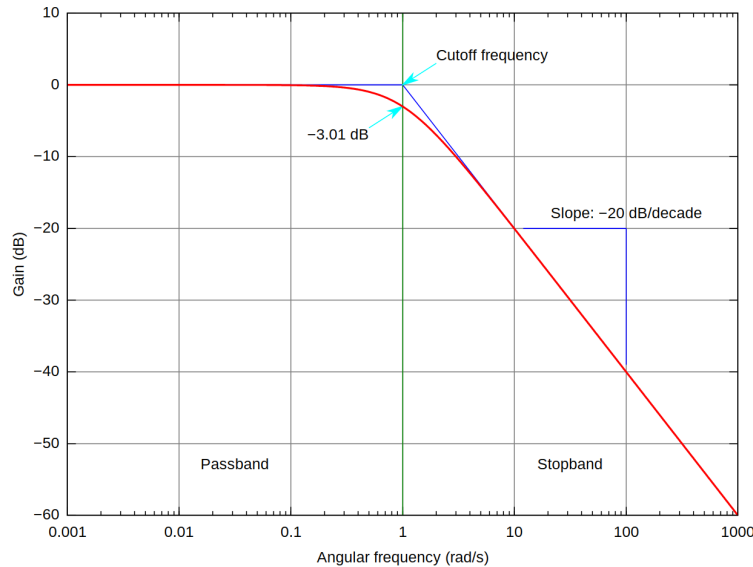


Figure 28: Reponse of a Butterworth Filter. [19]

To begin, the group ran a sample path using the robot and recorded the IMU values. The specific path was not important as long as the robot started moving, stopped moving, and rotated during the path. The values were then put through a Fast Fourier Transform function in MATLAB. The Fast Fourier Transform is an algorithm that will map a time-based signal into a frequency-based signal. You can then analyze which frequency ranges are present within the time-based signal. An example of the FFT of the x-acceleration, y-acceleration, and theta-acceleration are shown below.

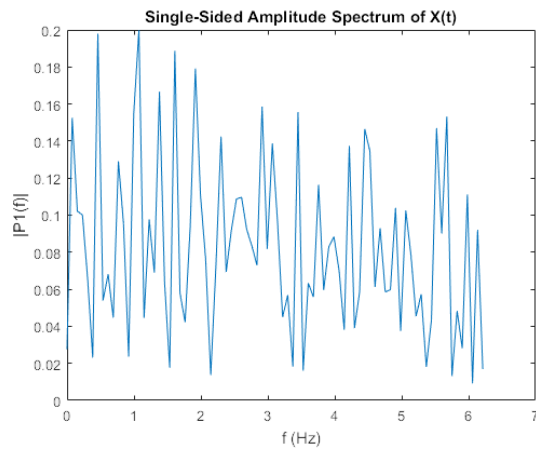


Figure 29: FFT of X Acceleration.

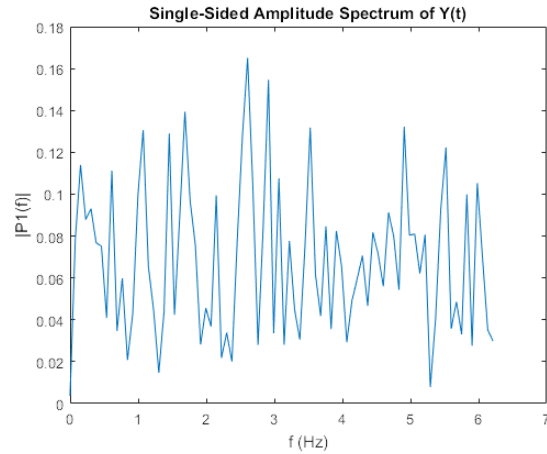


Figure 30: FFT of Y Acceleration.

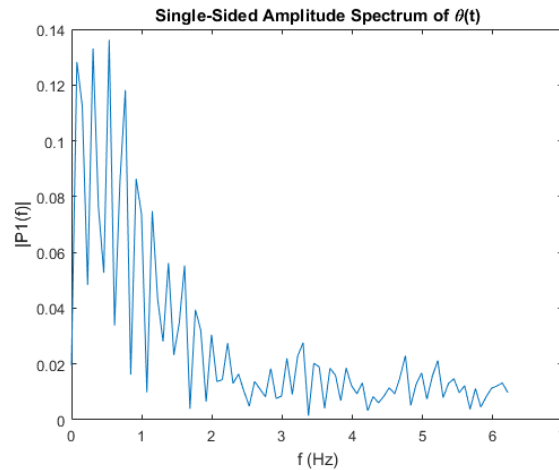


Figure 31: FFT of θ Acceleration.

As shown in the figures above, the x and y accelerations have essentially an equal amount of power in the lower and higher frequency sections, while the theta

acceleration has a better result, showing higher power in the lower frequencies and decreasing power in the higher frequencies. Optimally, because the accelerations are not cyclic, having only low frequency measurements is desired.

4 Plume Generation System

The plume stand was built by a previous team in order to provide a sturdy place to hold either CO₂ sensors or the diffuser. The stand is made from steel and has a height of about 1 meter. For the plume generation the current group continued to use the hardware and setup used by the previous project, though to control the plume generation the current group chose to use LabVIEW[®]. To control the mass flow rate of the plume we used the Sierra SmartTrak[®] C100L. Additionally, the group has used the Omega[®] SV3308 Solenoid Valve to control the on/off operation of the plume. This operation was done manually in the past using a switch but this was integrated into the LabVIEW[®] program. The CO₂ tank is connected to a Harris[®] Model 9296 Regulator to decrease the pressure from 700psi.

4.1 Flow meter

Unfortunately, due to an incorrect calculation by the previous team that purchased this product, current experiments were limited in terms of flow rates. This is due to the fact that the flow meter only allows 15SLPM of CO₂ through it, which works out to 459 mg/s. The test runs of the apparatus validated this result as flow rate values higher than 467 mg/s were not achieved even though the selected flow rates were higher. This made plume simulation results from previous projects obsolete and therefore new simulations with different flow rate values were needed.

4.2 Solenoid control

The previous MQP team used a mechanical switch connected to an 18V power source in order to control the solenoid. The group decided that using LabVIEW[®] to control the solenoid would prove to be more convenient and safer if the system needed to be disabled quickly. The following figure was the schematic used to implement LabVIEW[®] control of the solenoid.

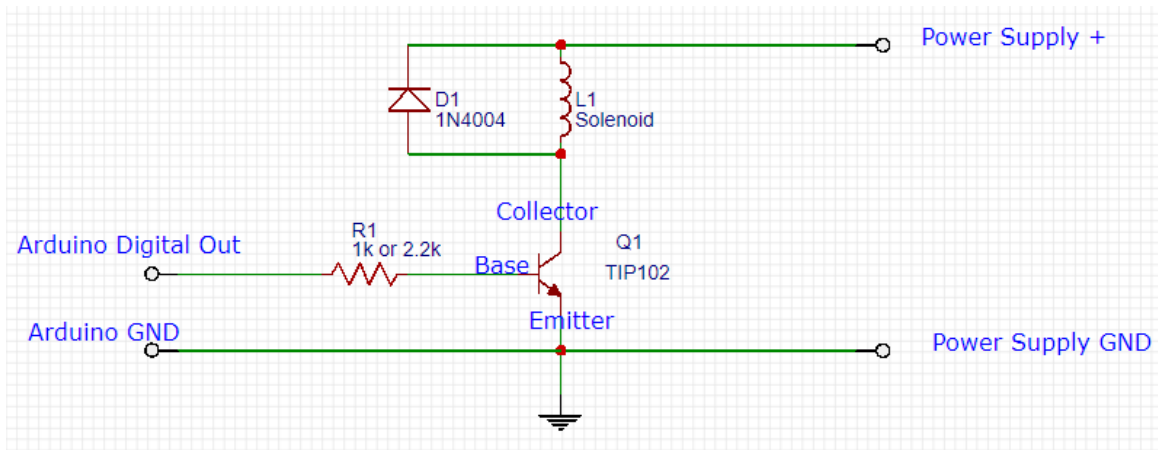


Figure 32: Schematic Used to Control the Solenoid.

It is important to note key aspects of the diagram above. First, a heat sink will most likely be needed on the transistor. The diagram in Figure 27 is for DC solenoids rated up to about 24We Lo. 12VB2A, 6V04A, 24V01A etc. Additionally, the protection diode should preferably be a schottky type, which has better response times. Something similar to a MUA340 is good for loads up to 3A.

4.3 LabVIEW Program

The LabVIEW[®] program designed can switch the solenoid valve, set a mass flow rate setpoint as well as monitor the actual mass flow rate. This is done by sending and receiving voltages of 0 to 5 volts through the Arduino. To connect to the Arduino, the virtual instrument package Linc was used. Additionally, to make it easier for the user, the input for the mass flow rate is in milligrams and then it is converted to voltage. To give a value other than the 0 or 5 volts we used a Duty Cycle function and an Arduino port that can support this function.

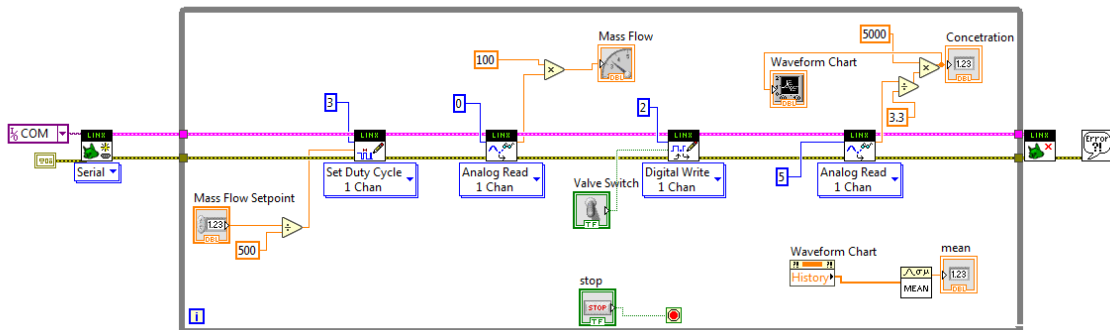


Figure 33: LabVIEW Block Diagram.

This virtual instrument package can also collect data from a carbon dioxide sensor. As such, this system can be used to test the plume simulations. To accurately collect data from the sensor and to avoid spikes in the data, an average of the last 10 values is calculated and projected on the front panel.

4.4 Point Source Simulation

The experiment this project was trying to verify requires simulating a point source of gas which the robot is going to be detecting. To simulate the point source, the previous years used a ping-pong ball in which they poked holes in order for the CO_2 that exits the tank with a certain speed to slow down and defuse. This technique seemed to work for the previous MQP teams, but because of the improvement of the level of accuracy and precision in the current experiments this year, it was determined that it would be appropriate to build a better diffuser for the experiment which is going to simulate a point source more accurately. In the picture below one can see the old and new diffusers. The main problem with the ping-pong ball was that because of the gas' velocity, it would not diffuse uniformly while more gas would come out from some holes than from others. The new designed diffuser has a large chamber where it allows the gas mean velocity to drop close to 0 and therefore act as a point source.



Figure 34: Old Diffuser.



Figure 35: New Diffuser.

4.5 Plume Simulations

In the experiments our group used data produced by the plume simulation program. To do this the group placed 4 CO_2 sensors at known positions around the source. Then the group had the plume generation run for the same time as the simulation, which is 120 seconds, and then compare our concentration data with the ones from the simulation. The simulation parameters were selected to be appropriate with our experiment limitations. The dimensions of the simulation are 5x5x5 meters and the time interval is of 1 second.

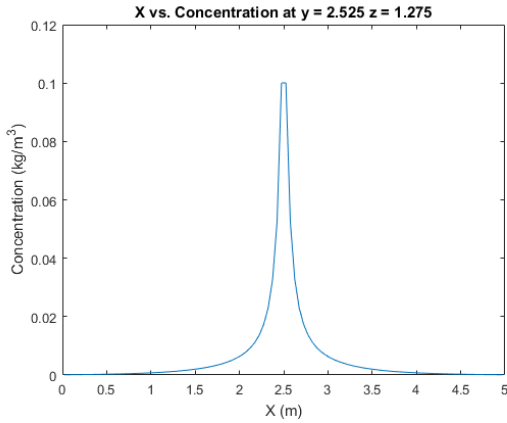


Figure 36: Concentration Along X-Axis.

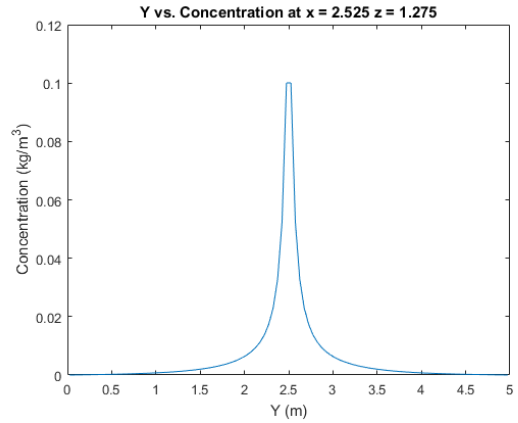


Figure 37: Concentration Along Y-Axis.

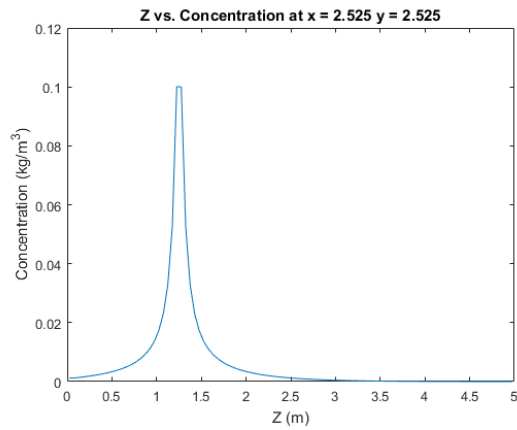


Figure 38: Concentration Along Z-Axis for a Specified x and y .

The figures above show the concentration over one of the axes with the other 2 directional components being held constant. The constants selected for each case are the same with the source location. This means that the maximum plane is obtained for each graph. Furthermore, all of these graphs are at 120 seconds, which is the last time stamp as well as at a flow rate of 400 mg/s which is the maximum rate achievable with the flow meter.

5 Future Work and Suggestions

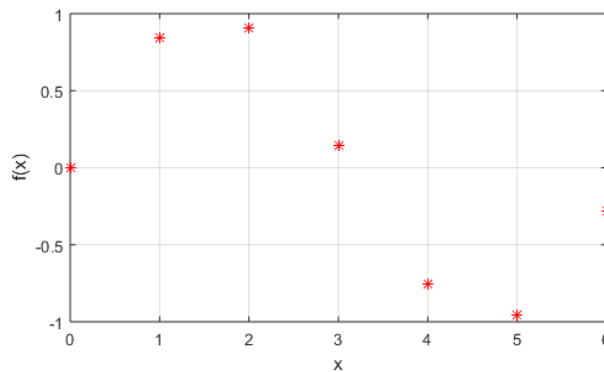
5.1 Interpolation

The main goal of this project was to verify the results of a gas plume simulation and reconstruct the plume based to our sensor readings. However, it is not possible to get CO_2 concentration data for every single point in space, therefore a mathematical model needs to be used in order to estimate the gas concentration at points for which the group did not have data. A suggestion for future work is the use of *Interpolation* in future projects. In this section what interpolation is will be briefly described as well as how it can be used in the context of these projects.

Interpolation is a method used to construct new points of a function within the range of a discrete set of known data points. The following example will help to gain a better understanding of what interpolation is and how it can be used. Assuming that one has the following set of x and $f(x)$ values that are part of an unknown continuous function:

$f(x)$	0.0000	0.8415	0.9093	0.1411	-0.7568	-0.9589	-0.2794
x	0	1	2	3	4	5	6

The following points plotted would look as follows:



The question that is being addressed is whether there is a function that passes through all these points, and if there is, what is the equation that describes it. On a first look one can imagine a polynomial function to be the solution to this problem, but the solution is never obvious. To find solutions that are mathematically accurate, one needs to use the interpolation equations. There are a number of interpolation methods including, but not limited to: Piecewise Constant, Linear, Polynomial and Spline Interpolation. These are all one-dimensional interpolation methods, though most high dimension interpolation simply involves applying these methods in one dimension first and then in the next dimension and so on. For instance, Figure ?? shows the general

equations used for a linear interpolation. While this example seems fairly simple, the equations for linear interpolation in two dimensions, or bilinear interpolation, are shown in Figure ??.

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0} = \frac{y_0(x_1 - x) + y_1(x - x_0)}{x_1 - x_0}, \quad (26)$$

By linearly interpolating on a 2D space we obtain four new points, $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$ and $Q_{22} = (x_2, y_2)$ so that

$$f(x, y_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \quad (27)$$

$$f(x, y_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}), \quad (28)$$

so that

$$f(x, y) \approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2), \quad (29)$$

which in matrix form can be written as follows

$$f(x, y) = \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}. \quad (30)$$

At first the problem may seem elementary, even in two dimensions, but in the case of the project one needs to use interpolation in three dimensions. In this case the equations get more involved and the interpolation errors can grow in scale if interpolation is performed multiple times. However, there are built in MATLAB functions for interpolation, therefore one suggestion for future efforts is working with these in order to interpolate the plume concentration at specific points within the area created by the four gas sensors.

5.2 Budgeting

An important part of every engineering project is the available budget which is used to buy the required materials. Although the project involved a lot of software and theory development, the group chose to build a new robot from the ground up, therefore a lot of hardware needed to be purchased. The project's initial budget was \$1250 for the five person team (\$250 per student), and the purchase orders and expenses are

listed in the table located in Appendix D.

As we can see on the budget table we went over budget for almost \$ 600 This is because we decided that the robots and hardware used the previous years did not fit our needs so we build the entire robot from the beginning. Although the group attempted to purchase the cheapest materials with the best possible quality, a budget of \$ 1,250 was insufficient. Note that the gross total does not take into account mistakes that the team made while building the robot, such as burning some motor controllers and Arduinos due to a lack of knowledge regarding circuitry. The most important thing that the team would like to warn students that will work on this project in the future is to be careful with project hardware and battery connections. By not burning controllers and electronics due to wiring connection errors and short-circuiting, one could save a significant amount of money and time, which is always crucial for an engineering project. Before starting work on any robot circuitry the team recommends brushing up on Electrical and Computer Engineering knowledge from any relevant courses (i.e. ECE 2010 etc.).

5.3 Hardware

Any error on our results can be attributed to the quality of hardware used. First of all, our IMU's raw data was very noisy and the data obtained was only particularly valid after significant filtering efforts. An upgrade to the IMU would allow the results to have increased accuracy due to less measurement noise and process error. Another potential hardware upgrade would be to the Arduino Mega. Since the microcontroller did not have enough processing power, the use of a base station computer was necessary to computer filtering outputs. With a much better microcontroller, Filtering could be calculated onboard the robot and results could be obtained both in real time and quicker. This would be optimal for future work in remodeling the plume instead of just localization.

An issue we ran into a lot with the spatial gradient based guidance was the robot leaving the boundaries due to it not being close enough to the plume for it to obtain accurate differences in CO_2 measurements. This led to the robot not being able to compute accurate rotations to navigate toward the plume, so it would sometimes leave the boundaries of the experiment. A recommendation for resolving this issue is to use black-white color sensors and use black tape to lay out a boundary. That way if the robot sensed it was crossing the tape, it could turn back to be within the boundary. The current microcontroller has enough empty IO ports to support the addition of these sensors.

6 Results & Conclusions

6.1 Filtering Results

A sample path of the robot was run using both the camera and the EKF simultaneously. In terms of the low pass filters, because the signal was more noisy from the x and y accelerometers, separate Butterworth filters were designed for the x and y, and theta accelerations. For x and y measurements, the Butterworth filters were set to 4th order, with a cutoff frequency of 0.4 Hz, and a sampling frequency of 40 Hz. As the theta acceleration had less noisy measurements, the filter was set to filter out less, with an order of 3, a cutoff frequency of 1 Hz, and a sampling frequency of 20 Hz.

In Figure 39 one can note the effectiveness of the Low Pass and Extended Kalman Filters as the actual position, taken with the camera, is nearly mirrored by the filtered outputs. Note that the red line is that of the filtered outputs while the blue is the camera trajectory.

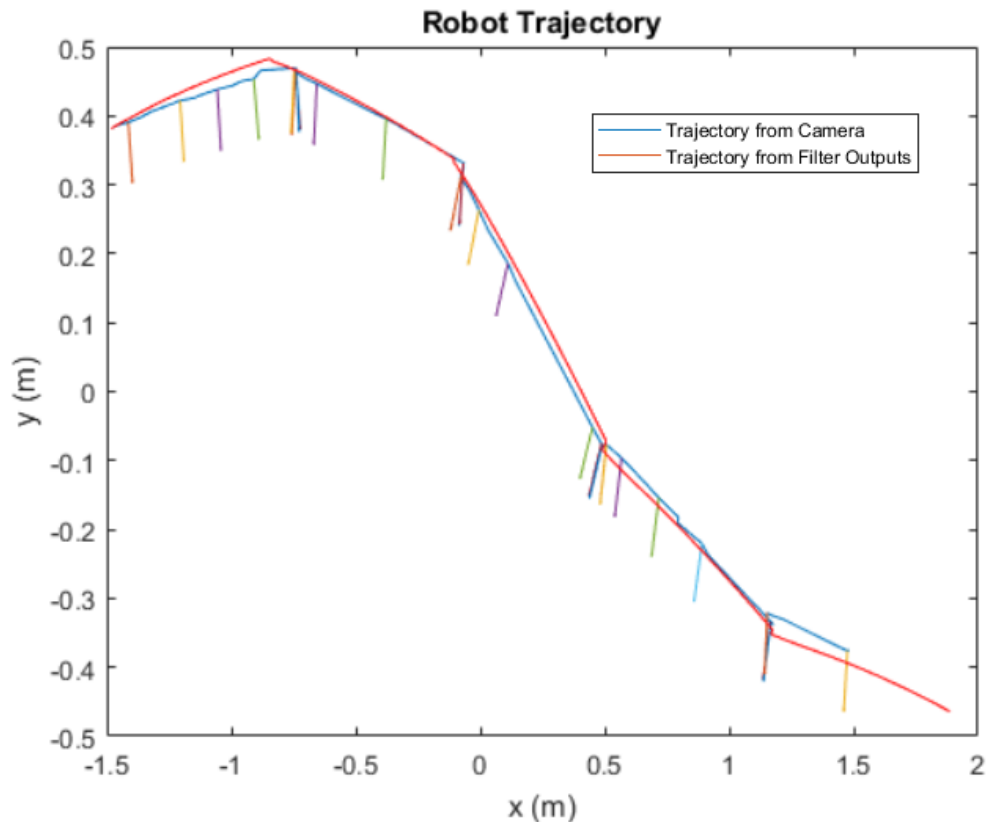


Figure 39: The Robot Trajectory (Actual and Filtered).

6.2 Spatial Gradient Based Guidance

The goal of this project was to localize a carbon dioxide plume source using the gradient search method. Using the data from the IMU, filtered through the Low Pass Filter and the EKF, to know the robot's position and navigating itself using the measurements from the SprintIR sensors, the team ran various tests to determine if the plume source could be localized. The robot undergoes two maneuvers for each iteration until it reaches the source of the plume. First it calculates an angle to rotate based on the differences in concentration between the front and back sensors as well as the left and right sensors using trigonometric equations. The robot then rotates that angle at a set constant rate of rotation that will be used for each turn. The robot then stops and moves forward for a very short distance with a set velocity that is also used for each iteration. The robot then repeats the sequence until it reaches a point where one of the sensors is saturated. A sensor is considered saturated when it reads a concentration above 4000 ppm. This number was chosen because at the height of the sensors, the concentration would only reach 4000 ppm only if the robot was within 10 centimeters of the plume source. This distance was decided to be sufficient for the experiment's purposes. The robot thus traveled towards the source albeit not in a direct path as the levels of carbon dioxide around its sensors were not high enough at the beginning of a test to register a significant difference.

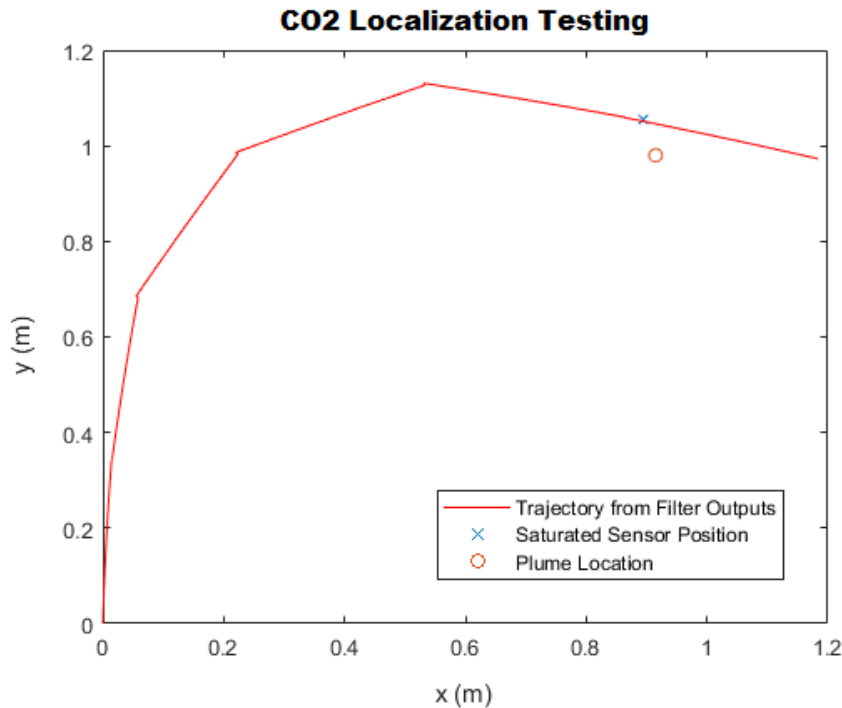


Figure 40: Plume Source Localization Using the Gradient Method.

For instance, in the test shown above in Picture 40, the robot stopped in the vicinity of the plume source with its rear sensor almost directly below it. The data from the IMU was processed through the EKF and the graph below was produced showing the trajectory of the robot. The rear sensor read 5500 ppm which was a reasonable result since it was the closest to the source. To then calculate the position of the plume source, the position of the rear sensor was calculated. The plume source is located within a 10 centimeter radius from the rear sensor. On the graph below this sensor is marked with an X. The trajectory of the robot is the line starting from the origin. The actual plume source location was measured before the experiment and was found to be at (0.98,0.91)m. This is marked on the graph below with an O. The absolute distance between the center of the estimated position and the actual position is 7.5cm which is within the allowed radius.

An important fact to note is that there has to be a significant amount of time between each experiment. This is so that the room adequately disperses any residual carbon dioxide gas that might affect the robot's measurements but also to allow for time to recalibrate the sensors to ambient levels.

6.3 Project Conclusion

Localization of a plume has a number of different applications that can have useful purposes in the future. This project explored the viability of a robot that uses four CO_2 sensors for CO_2 sensing, an Inertial Measurement Unit for localization, and a plume generation system for controlled release of CO_2 . An Extended Kalman Filter, along with a Low Pass Filter, was used to process noisy sensor data and give us more accurate locations. A simulation was designed in Simulink to determine best values for velocities and control methods. A robot was assembled and a sensor stand was 3D printed to provide a stable platform for the microcontrollers and CO_2 sensors. Tests were run that showed the robot was able to locate the source of the plume using a spatial gradient-based search algorithm that used the differences in concentrations of CO_2 gas. Data from these tests was post-processed to determine the location of the robot during the test and thus the plume.

Overall, the project presented a number of difficulties. These ranged from inconsistent results of previous groups to burning robot components due to small wiring mistakes. For instance, one major obstacle was that of creating an Extended Kalman Filter for the project. The EKF was completely designed for this project and the majority of the difficulties surrounding the filter were due to this fact. The major challenge being tuning the filter to function correctly. However, this complication, along with others that occurred during the course of the project forced the need to learn additional concepts, skills, and theories that added to the overall value and success of the work performed.

Over the span of the project, a number of objectives were accomplished. These include the complete redesign of the robot, base station, EKF, and communication

system, the outcomes of successful spatial gradient based guidance, a working EKF and Low Pass Filter, as well as a functional plume generation system. Given the short time span of this work, the number of desirable outcomes represents an effective project, though of course there are still a number of ways to forward the work done by this group and build on the work outlined in this report to accomplish more.

A Code Appendix

A.1 Robot Controller Code

```
1 function [VR, VL, psir, psirdiff, thold, turn] =...
2 fcn(xr, yr, x, y, psi, thold, turn, vt)
3 psir = atan2(yr-y,xr-x);
4 %atan2 is used because it returns the 4-quadrant inverse tangent,
5 %while atan is limited from -pi/2 to pi/2
6 psirdiff = psi-psir;
7 if (abs(psi - psir) > 0.1)
8     if psi < psir
9         %vl is greater
10        VL = vt;
11        VR = -vt;
12        turn = 1;
13    else
14        VL = -vt;
15        VR = vt;
16        turn = 1;
17    end
18 else
19    %move
20    if(turn == 1)
21        if(thold == 5)
22            thold = 0;
23            VR = 2*vt;
24            VL = 2*vt;
25        else
26            thold = thold + 1;
27            VR = 0;
28            VL = 0;
29        end
30    else
31        VR = 2*vt;
32        VL = 2*vt;
33    end
34 end
```

A.2 Kalman Filter Algorithm

```

1 %Inputs
2 ts = 0.1; %Sampling Time
3 W = 0.25; %Robot Width (m)
4 nsteps = 200;
5 load('fig8_200_01.mat');
6 load('noisedata.mat');
7
8 %Initial Conditions
9 Th(1) = pi;
10 x(1) = 1;
11 y(1) = 0.15;
12 %initial state, updates every discrete time increment
13 xhat = [x(1) ; y(1) ; Th(1) ];
14 sigmaD = eps;
15 sigmaTheta = eps;
16 sigmaX = 1.878e-4;
17 sigmaY = 2.921e-4;
18 sigmaOmega = 3.6774e-5;
19 R = [sigmaX^2,0,0 ; 0,sigmaY^2,0 ; 0,0,sigmaOmega^2]; %Variances Matrix
20 C = [2/(ts^2),0,0;0,2/(ts^2),0;0,0,1/ts]; %Observation Matrix
21 P(:, :, 1) = eye(3); %Process Errors Covariance Matrix
22
23 for i = 2:nsteps
24     %Change in states at this timestep
25     dD = 0.5*ts*(Vr(i-1)+Vl(i-1));
26     dTh = ts*(Vr(i-1)-Vl(i-1))/W;
27     %State Predict (same as F)
28     xhat(:, i) = [ xhat(1, i-1) + dD*cos(xhat(3, i-1)+dTh) ; ...
29                 xhat(2, i-1) + dD*sin(xhat(3, i-1)+dTh) ; ...
30                 xhat(3, i-1) + dTh] ;
31     %Jacobian of F in k-1 with respect to the states
32     dFdXhat = [1,0,-dD*sin(xhat(3, i-1)+dTh); ...
33               1,0,dD*cos(xhat(3, i-1)+dTh); ...
34               0,0,1];
35     %Process Error Covariance Predict
36     P(:, :, i) = dFdXhat*P(:, :, i-1)*dFdXhat';
37     %Innovation Matrix
38     S = (C*P(:, :, i)*C') + R;
39     %Kalman Gain
40     K = P(:, :, i)*C'*inv(S);
41     %State Update
42     xhat(:, i) = xhat(:, i) + K*(Z(:, i) - C*(xhat(:, i) - xhat(:, i-1)));
43     %Process Error Update
44     P(:, :, i) = (eye(3) - K*C)*P(:, :, i);
45 end

```

A.3 Robot Arduino Mega Code

```
1  /*----- -
2  |    -- \ | |   |----- \ | |   |----- \ | |
3  | |  --) | | |   |-----) | | |   |-----) | | |
4  | |  ---// | | |   |-----// | | |   |-----// | | |
5  | |  ---// | | |   |-----// | | |   |-----// | | |
6  | |  ---// | | |   |-----// | | |   |-----// | | |
7
8  A 4-wheeled differential steering robot that measures CO2 concentration
9
10 Authors: Owen McGrath, Ben Preston, Spyros Kasapis, Marios Kontopyrgos,
11 Sly Halama
12
13 =====
14
15 EXTERNAL LIBRARIES:
16
17 cozir-master from Github user Rodir at https://github.com/roder/cozir
18
19 adafruit_bno055 from Adafruit at
20 https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor
21
22 adafruit_sensor from Adafruit at
23 https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor
24
25 =====
26
27 FUNCTIONS:
28
29 getImuMeasurements() - Pulls new accelerations and angular velocity
30 from the IMU
31
32 getCO2Measurements() - Pulls new CO2 Concentrations from the Sensors
33
34 getVel(int pin) - Calculates frequency of the square wave outputted
35 from the motors in either A or B, converts to a velocity
36
37 getVelAverage(int side) - Calculates velocity average for Vl and Vr
38
39 exitSafeStart(SoftwareSerial mc) - Resets the motor Controllers from
40 "Error Mode" so that the motors can move again after there is an
41 error(red light turns off)
42
43 void setMotion(int type) - Sets motion mode for the robot
44
45 */
46 #include <cozir.h>
47 #include <Adafruit_Sensor.h>
48 #include <Adafruit_BNO055.h>
```

```

49 #include <utility/imumaths.h>
50 #include <Wire.h>
51 #include <SoftwareSerial.h>
52
53 //MATHEMATICAL CONSTANTS AND DIMENSIONS
54
55 #define pi 3.141592653589
56 #define w .25 //robot width
57 #define MAX_SPEED 3200
58 String command;
59 int count = 0;
60 long motionStartTime = 0;
61 long currentTime = 0;
62 int motionStep = 0;
63 int motionMode = 5;
64 int timeAmount = 30000;
65 double omega = 1.338667;
66 double angle = 0;
67 double forwardTime = 0;
68 int mode = 0;
69 int isMoving = 0;
70 double deltaT = 0;
71 int prevTime = 0;
72 int waitingForCommand = 1;
73 int msmtCounter = 0;
74 char separator = '+';
75
76
77
78
79 //ARDUINO INITIALIZATIONS
80
81 //arduino pins
82
83 //CO2 Sensors
84 #define co2_f_rx 10
85 #define co2_f_tx 3
86 #define co2_b_rx 11
87 #define co2_b_tx 5
88 #define co2_l_rx 12
89 #define co2_l_tx 7
90 #define co2_r_rx 13
91 #define co2_r_tx 9
92
93 //motor controllers
94 #define l_controller_rxPin 15
95 #define l_controller_txPin 14
96 #define r_controller_rxPin 17
97 #define r_controller_txPin 16
98
99 //encoders

```

```

100 #define rb_encoder_A 35
101 #define rb_encoder_B 37
102 #define rf_encoder_A 39
103 #define rf_encoder_B 41
104 #define lb_encoder_A 43
105 #define lb_encoder_B 47
106 #define lf_encoder_A 49
107 #define lf_encoder_B 46
108
109 //sample rate for determining encoder frequency
110 #define SAMPLES 16
111
112 //serial connections
113 SoftwareSerial rmc=SoftwareSerial(r_controller_rxPin , r_controller_txPin);
114 SoftwareSerial lmc=SoftwareSerial(l_controller_rxPin , l_controller_txPin);
115 SoftwareSerial co2_f = SoftwareSerial(co2_f_rx , co2_f_tx);
116 SoftwareSerial co2_b = SoftwareSerial(co2_b_rx , co2_b_tx);
117 SoftwareSerial co2_r = SoftwareSerial(co2_r_rx , co2_r_tx);
118 SoftwareSerial co2_l = SoftwareSerial(co2_l_rx , co2_l_tx);
119
120 //IMU INITIALIZATIONS
121 Adafruit_BNO055 bno = Adafruit_BNO055(55);
122 float z[3] = {0};
123
124 //COZIR INITIALIZATIONS
125 int cf = 0;
126 int cb = 0;
127 int cr = 0;
128 int cl = 0;
129 int cmax_idx_lr;
130 int cmax_idx_fb;
131 int c[4] = {0};
132
133 //MOTOR CONTROLLER INITIALIZATIONS
134 #define accelerationID 5
135 #define decelerationID 9
136
137
138
139
140 /* FUNCTION: getImuMeasurements()
141 * DESCRIPTION:
142 * Pulls new accelerations and angular velocity from the IMU
143 */
144 void getImuMeasurements(){
145 sensors_event_t event;
146 bno.getEvent(&event);
147 imu::Vector<3> gyro=bno.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
148 imu::Vector<3> accel=bno.getVector(Adafruit_BNO055::VECTOR_LINEARACCEL);
149 z[0] = accel.x();
150 z[1] = accel.y();

```

```

151 z[2] = gyro.z();
152 }
153
154 /* FUNCTION: getCO2Measurements()
155 * DESCRIPTION:
156 * Pulls new CO2 Concentrations from the Sensors
157 */
158 void getCO2Measurements() {
159     COZIR czr_f(co2_f);
160     co2_f.listen();
161     cf = czr_f.CO2();
162
163     COZIR czr_r(co2_r);
164     co2_r.listen();
165     cr = czr_r.CO2();
166
167     COZIR czr_b(co2_b);
168     co2_b.listen();
169     cb = czr_b.CO2();
170
171     COZIR czr_l(co2_l);
172     co2_l.listen();
173     cl = czr_l.CO2();
174 }
175
176 /* FUNCTION: getVel(int pin)
177 * DESCRIPTION:
178 * Calculates frequency of the square wave outputted from the motors in
179 * either A or B, converts to a velocity
180 * PARAMETERS:
181 * int pin //Pin number for encoder frequency measurement
182 * RETURNS:
183 * Wheel Velocity (m/s)
184 */
185 float getVel(int pin) {
186     long freq = 0;
187     for(unsigned int j=0; j<SAMPLES; j++){
188         freq+= pulseIn(pin, HIGH, 250000);
189     }
190     return .39*freq/SAMPLES/898;
191 }
192
193 /* FUNCTION: getAppVel(int side)
194 * DESCRIPTION:
195 * Returns predetermined velocities for a motion mode (see setMotion)
196 * PARAMETERS:
197 * int side //0 for Vl, 1 for Vr
198 * RETURNS:
199 * Wheel Velocity (m/s)
200 */
201 float getAppVel(int side) {

```

```

202 switch(motionMode){
203     case 1:
204         if(side == 0){return .2539;} else {return .2467;}
205     case 2:
206         return -0.261;
207     case 3:
208         if(side == 0){return .186562;} else {return -.182777;}
209     case 4:
210         if(side == 0){return -.195741;} else {return .176269;}
211     case 5:
212         return 0;
213 }
214 }
215
216 /* FUNCTION: getVelAverage(int side)
217 * DESCRIPTION:
218 * Calculates velocity average for Vl and Vr
219 * PARAMETERS:
220 * int side //Left velocity -> 0 or Right Velocity -> 1
221 * RETURNS:
222 * Velocity Average for each side both A and B pins. (m/s)
223 */
224 float getVelAverage(int side){
225     //Need this if statement because getVel doesn't work if the motors are
226     //not moving, it pauses for a while
227     if(isMoving == 0){
228         return 0;
229     } else {
230         switch(side){
231             case 0:
232                 return .25*(getVel(lb_encoder_A)+getVel(lb_encoder_B)+
233                     getVel(lf_encoder_A)+getVel(lf_encoder_B));
234             case 1:
235                 return .25*(getVel(rb_encoder_A)+getVel(rb_encoder_B)+
236                     getVel(rf_encoder_A)+getVel(rf_encoder_B));
237         }
238     }
239 }
240
241 /* FUNCTION: exitSafeStart(Software Serial mc)
242 * DESCRIPTION:
243 * Resets the motor Controllers from "Error Mode" so that the motors
244 * can move again after there is an error(red light turns off)
245 * PARAMETERS:
246 * SoftwareSerial mc //Serial Connection to the motor Controller
247 */
248 void exitSafeStart(SoftwareSerial mc){
249     mc.write(0x83);
250 }
251
252

```

```

253
254 /* FUNCTION: setMotorSpeed(int speed, SoftwareSerial mc)
255 * DESCRIPTION:
256 * Writes Speed command to the Motor controllers over serial
257 * PARAMETERS:
258 * int speed //Needs to be in the range -3200 to 3200
259 * SoftwareSerial mc //Serial Connection to the Motor Controller
260 */
261 void setMotorSpeed(int speed, SoftwareSerial mc){
262     if (speed < 0){
263         mc.write(0x86); // motor reverse command
264         speed = -speed; // make speed positive
265     }else{
266         mc.write(0x85); // motor forward command
267     }
268     mc.write(speed & 0x1F);
269     mc.write(speed >> 5);
270 }
271
272
273 /* FUNCTION:
274 * setMotorLimit(unsigned char limitID, unsigned int limitValue,
275 * SoftwareSerial mc)
276 * DESCRIPTION:
277 * Sets Acceleration and Deceleration values for ramping the motors.
278 * PARAMETERS:
279 * unsigned char limitID // Type of limit to be changed (reference polo
280 * SoftwareSerial mc //Serial Connection to the Motor Controller
281 */
282 unsigned char setMotorLimit(unsigned char limitID,
283 unsigned int limitValue, SoftwareSerial mc){
284     mc.write(0xA2);
285     mc.write(limitID);
286     mc.write(limitValue & 0x7F);
287     mc.write(limitValue >> 7);
288 }
289
290
291 /* FUNCTION: setMotion(int type, int time)
292 * DESCRIPTION:
293 * Sets motion mode for the robot
294 * PARAMETERS:
295 * int type //trajectory type for the robot
296 *     MOVE FORWARD ->1
297 *     MOVE IN REVERSE ->2
298 *     CW ROTATION ->3
299 *     CCW ROTATION ->4
300 *     STOP MOTION ->5
301 */
302 void setMotion(int type){
303     switch (type){

```



```

304     case 1:
305         motionMode = 1;
306         isMoving = 1;
307         setMotorSpeed(.252*MAX_SPEED, lmc);
308         setMotorSpeed(-.2538*MAX_SPEED, rmc);
309         break;
310     case 2:
311         motionMode = 2;
312         isMoving = 1;
313         setMotorSpeed(-.245*MAX_SPEED, lmc);
314         setMotorSpeed(.259*MAX_SPEED, rmc);
315         break;
316     case 3:
317         motionMode = 3;
318         isMoving = 1;
319         setMotorSpeed(-.245*MAX_SPEED, lmc);
320         setMotorSpeed(-.2538*MAX_SPEED, rmc);
321         break;
322     case 4:
323         motionMode = 4;
324         isMoving = 1;
325         setMotorSpeed(.255*MAX_SPEED, lmc);
326         setMotorSpeed(.262*MAX_SPEED, rmc);
327         break;
328     case 5:
329         motionMode = 5;
330         isMoving = 0;
331         setMotorSpeed(0, lmc);
332         setMotorSpeed(0, rmc);
333         break;
334     }
335 }
336
337
338 void setup(){
339     Serial.begin(9600);
340     Serial.println("Setup beginning");
341
342     //initializations for the IMU
343     bno.begin();
344     bno.setExtCrystalUse(true);
345
346
347     // initialize software serial object with baud rate of 19.2 kbps
348     rmc.begin(19200);
349     lmc.begin(19200);
350     // the Simple Motor Controller must be running for at least 1 ms
351     // before we try to send serial data, so we delay here for 5 ms
352     delay(5);
353     // if the Simple Motor Controller has automatic baud detection
354     // enabled, we first need to send it the byte 0xAA (170 in decimal)

```

```

355 // so that it can learn the baud rate
356 rmc.write(0xAA); // send baud-indicator byte
357 lmc.write(0xAA); // send baud-indicator byte
358
359 //SET MOTOR ACCEL AND DECEL VALUES
360 setMotorLimit(accelerationID , 1, rmc);
361 setMotorLimit(accelerationID , 1, lmc);
362 setMotorLimit(decelerationID , 1, rmc);
363 setMotorLimit(decelerationID , 1, lmc);
364
365 // next we need to send the Exit Safe Start command, which
366 // clears the safe-start violation and lets the Motor run
367 exitSafeStart(rmc);
368 exitSafeStart(lmc);
369
370 //START AFTER BEING POWERED ON FOR 15 SEC
371 motionStartTime = millis();
372 Serial.println("Setup Complete");
373 }
374
375 void loop(){
376 // getCO2Measurements();
377 getImuMeasurements();
378 currentTime = millis();
379 deltaT = currentTime - prevTime;
380 prevTime = currentTime;
381 //// PRINT VELOCITIES FROM ENCODERS
382 // Serial.print(", ");
383 // Serial.print(getVelAverage(0), 8);
384 // Serial.print(", ");
385 // Serial.println(getVelAverage(1), 8);
386
387 //PRINT PREDETERMINED VELOCITIES, IMU DATA, AND CO2 CONCENTRATIONS
388 if(isMoving){
389 // Serial.print(deltaT, 8);
390 // Serial.print(" ");
391 Serial.print(getAppVel(0),8);
392 Serial.print(" ");
393 Serial.print(getAppVel(1),8);
394 Serial.print(" ");
395 Serial.print(z[0], 8);
396 Serial.print(" ");
397 Serial.print(z[1], 8);
398 Serial.print(" ");
399 Serial.println(z[2], 8);
400 msmtCounter++;
401 // Serial.println(" ");
402 // Serial.print(cf, 8);
403 // Serial.println(" ");
404 // Serial.print(cb, 8);
405 // Serial.println(" ");

```

```

406 // Serial.print(cl, 8);
407 // Serial.println(" ");
408 // Serial.print(cr, 8);
409 // Serial.println(" ");
410 }
411
412 ///GRADIENT MODE
413 // start new motion sequence after time "timeAmount" (in ms)
414 if(currentTime-motionStartTime > timeAmount){
415     if(mode == 0){
416         if(count){
417             deltaT = timeAmount/msmtCounter;
418             msmtCounter = 0;
419             Serial.print(">");
420             Serial.println(deltaT);
421         }
422         count++;
423         //stop for a delay, get the CO2 measurements then display them
424         setMotion(5);
425         delay(2000);
426         getCO2Measurements();
427         c[0] = cf*10;
428         c[1] = cr*10;
429         c[2] = cb*10;
430         c[3] = cl*10;
431         Serial.print("F: "); Serial.print(cf); Serial.print(", ");
432         Serial.print("R: "); Serial.print(cr); Serial.print(", ");
433         Serial.print("B: "); Serial.print(cb); Serial.print(", ");
434         Serial.print("L: "); Serial.println(cl);
435         //Determine the greatest concentration in front/back then L/R.
436         if(c[1] > c[3]){
437             cmax_idx_lr = 1; //right
438         } else if(c[3] > c[1]){
439             cmax_idx_lr = 3; //left
440         } else {
441             return;
442         }
443
444         if(c[0] > c[2]){
445             cmax_idx_fb = 0; //front
446         } else if(c[2] > c[0]){
447             cmax_idx_fb = 2; //back
448         } else {
449             return;
450         }
451         //calculate the gradient between front/back,
452         //left/right then find the angle of rotation
453         if(cmax_idx_fb == 0){
454             if(cmax_idx_lr == 1){
455                 angle = atan2(c[1]-c[3], c[0]-c[2]);
456             }

```

```

457     else {
458         angle = atan2(c[3]-c[1], c[0]-c[2]);
459     }
460
461 } else {
462     if(cmax_idx_lr == 1){
463         angle = pi - atan2(c[1]-c[3], c[2]-c[0]);
464     }
465     else {
466         angle = pi - atan2(c[3]-c[1], c[2]-c[0]);
467     }
468 }
469 //Calculate the time it needs to rotate at angular velocity omega
470 timeAmount = (angle/omega)*1000;
471 Serial.print("Rotating at angle: "); Serial.print(angle);
472 Serial.print(", for time: "); Serial.println(timeAmount);
473
474 if(cmax_idx_lr == 1){
475     Serial.print("Moving Clockwise, concentration is: ");
476     Serial.println(c[cmax_idx_lr]);
477     setMotion(3);
478 } else {
479     Serial.print("Moving Counter Clockwise, concentration is: ");
480     Serial.println(c[cmax_idx_lr]);
481     setMotion(4);
482 }
483 motionStartTime = millis();
484 mode = 1;
485 } else if(mode == 1){
486     if(cmax_idx_lr == 1){
487         deltaT = .031*timeAmount;
488     } else {
489         deltaT = .032*timeAmount;
490     }
491     msmtCounter = 0;
492     Serial.print(">");
493     Serial.println(deltaT);
494     Serial.println("Moving Forward");
495     setMotion(5);
496     delay(1000);
497     setMotion(1);
498     timeAmount = 2000;
499     motionStartTime = millis();
500     mode = 0;
501 }
502 }
503 if(c[0] > 4000 || c[1] > 4000 || c[2] > 4000 || c[3] > 4000){
504     Serial.println("END");
505     Serial.print("F: "); Serial.print(c[0]); Serial.print(", ");
506     Serial.print("R: "); Serial.print(c[1]); Serial.print(", ");
507     Serial.print("B: "); Serial.print(c[2]); Serial.print(", ");

```

```

508 Serial.print("L: "); Serial.println(c[3]);
509 setMotion(5);
510 while(1){}
511
512 }
513 ///TEST TRAJECTORY MODE
514 // if(currentTime-motionStartTime > timeAmount){
515 //     switch(motionStep){
516 //         //forward
517 //         case 0:
518 //             if(count == 2 || count == 4){
519 //                 deltaT = .031*timeAmount;
520 //             } else if(count == 6 || count == 8){
521 //                 deltaT = .032*timeAmount;
522 //             }
523 //             if(count){
524 //                 msmtCounter = 0;
525 //                 Serial.print(">");
526 //                 Serial.println(deltaT);
527 //             }
528 //             setMotion(5);
529 //             delay(500);
530 //             setMotion(1);
531 //             motionStartTime = millis();
532 //             prevTime = millis();
533 //             count++;
534 //             if(count == 1 || count == 3){
535 //                 motionStep = 1;
536 //             } else if(count == 5 || count == 7) {
537 //                 motionStep = 2;
538 //             } else {
539 //                 motionStep = 3;
540 //             }
541 //             timeAmount = 2000;
542 //             break;
543 //
544 //         //CW
545 //         case 1:
546 //             deltaT = timeAmount/msmtCounter;
547 //             msmtCounter = 0;
548 //             Serial.print(">");
549 //             Serial.println(deltaT);
550 //             setMotion(5);
551 //             delay(1000);
552 //             setMotion(3);
553 //             motionStartTime = millis();
554 //             prevTime = millis();
555 //             count++;
556 //             motionStep = 0;
557 //             timeAmount = 1200;
558 //             break;

```

```

559 //
560 // //CCW
561 // case 2:
562 //     deltaT = timeAmount/msmtCounter;
563 //     msmtCounter = 0;
564 //     Serial.print(">");
565 //     Serial.println(deltaT);
566 //     setMotion(5);
567 //     delay(1000);
568 //     setMotion(4);
569 //     motionStartTime = millis();
570 //     prevTime = millis();
571 //     count++;
572 //     motionStep = 0;
573 //     timeAmount = 1200;
574 //     break;
575 // //stop
576 // case 3:
577 //     deltaT = timeAmount/msmtCounter;
578 //     msmtCounter = 0;
579 //     Serial.print(">");
580 //     Serial.println(deltaT);
581 //     Serial.println("END\n");
582 //     setMotion(5);
583 //     while(1){}
584 //
585 // }
586 // }
587 }

```

A.4 Base Station Python Code

```
1 import serial
2 import sys
3 import time
4 from numpy import matrix
5 from numpy import linalg
6
7 ser = serial.Serial('COM20')      #INSERT SERIAL PORT HERE
8 ser.open
9
10 if ser.is_open :
11     print("\n")
12     print("===== \n")
13     print("Serial Communication Established with the Plume Robot, ")
14     print("beginning operation\n")
15     print("===== \n\n")
16 else :
17     print("===== \n\n")
18     print("Serial Communication Failed to Establish, ")
19     print("please check hardware connections\n")
20     print("Ending Script in 5 (s)\n")
21     time.sleep(5)
22     sys.exit()
23
24 is_recording = True;
25 f = open("robotdata.txt", "w");
26 while is_recording:
27     dataline = ser.readline().decode()
28     if dataline == "END\n":
29         f.close()
30         break
31     f.write(dataline)
32
33 print("\n")
34 print("===== \n")
35 print("Data successfully recorded, beginning parsing\n")
36 print("===== \n\n")
37
38 f = open("robotdata.txt", "r");
39
40 trajectories = []
41 trajectory = []
42 deltaT = []
43
44 for data in f.readlines():
45     if data == '\n':
46         continue
47     elif data[0] == '>':
48         deltaT.append(float(data[1:-2]))
```

```

49     trajectories.append(trajectory)
50     trajectory = []
51     else:
52         datapoint = data.split(" ", 5)
53         print(datapoint)
54         lastpoint = datapoint[4]
55         datapoint[4] = lastpoint[:-2]
56         trajectory.append(list(map(float, datapoint)))
57
58 # ROBOT INITIAL CONDITIONS
59 vl_i = 0
60 vr_i = 0
61 x_i = 0
62 y_i = 0
63 theta_i = 0
64 width = .25 #robot width in meters
65
66 #STATES + VELOCITIES-
67 vl = vl_i
68 vr = vr_i
69 x = x_i
70 y = y_i
71 theta = theta_i
72
73 #EKF INITIAL CONDITIONS
74
75 #these are R^2 values
76 sigmaD = 0
77 sigmaTheta = 0
78 sigmaX = float(.0001878)
79 sigmaY = float(.0002921)
80 sigmaOmega = float(.0000036774)
81
82 R = matrix([[sigmaX,0,0],[0,sigmaY,0],[0,0,sigmaOmega]])
83 P = R;
84
85 firstMeas = True
86
87 print("Initial Conditions are VL = ", vl_i, ", VR = ", vr_i, ", x = ",
88 x_i, ", y = ", y_i, ", theta = ", theta_i, "\n")
89 print("\n")
90
91 #SEND VALUES TO FILTERS
92 count = 0
93 for x in trajectories:
94     for point in x:
95         vl, vr, x_accel, y_accel, omega = point.split();
96         ts = float(deltaT(count))
97         vl = float(vl)
98         vr = float(vr)
99         Z = matrix([[float(x_accel)], [float(y_accel)], [float(omega)]])

```



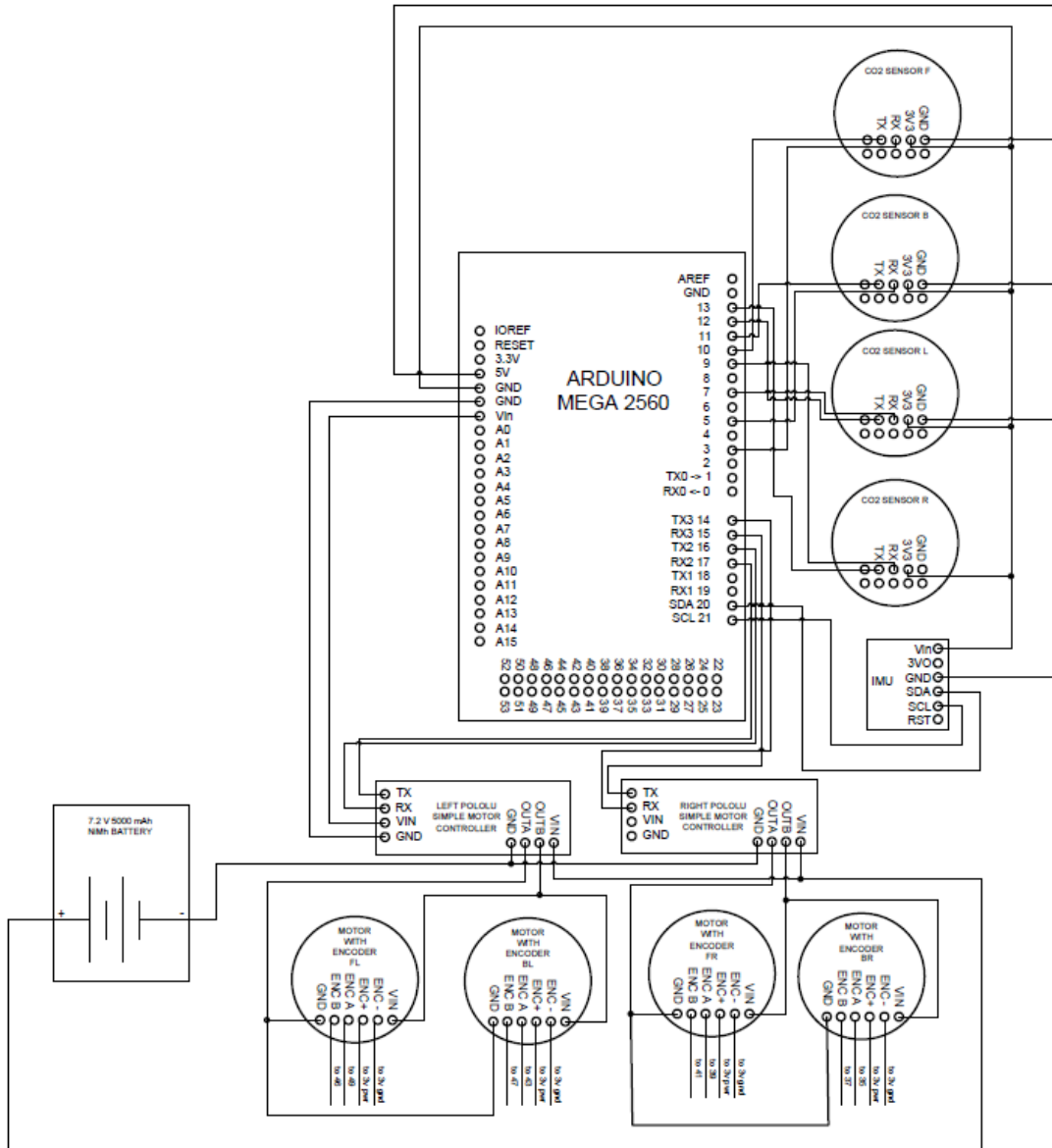
```

100     if firstMeas == true:
101         Z_filt = Z
102         firstMeas = false
103
104         #PUSH MEASUREMENTS THROUGH LOW PASS FILTER
105         Z_filt = lowpassfilter(Z, Z_filt)
106         Z = Z_filt;
107
108         #PUSH FILTERED MEASUREMENTS THROUGH EKF
109         EKF(vl, vr, Z, ts)
110         count = count+1
111
112     print("Final position is: X: ", xhat(1), " Y: ", xhat(2), " Theta: "
113           , xhat(3))
114
115     def ekf(vl, vr, Z, ts):
116         global xhat
117         global P
118
119         C = matrix([1/(.001*ts^2), 0, 0], [0, 1/(.001*ts^2), 0], [0, 0, 1/(.001*ts^2)])
120
121         #linearized change in pose for this discrete time step
122         dD = .5*.001*ts*vl*vr
123         dTh = .001*ts*(vr-vl)/width
124
125         #state prediction
126         xhat_p = matrix([[xhat.item(0)+dD*cos(xhat.item(2))],
127                        [xhat.item(1)+dD*sin(xhat.item(2))], [xhat.item(2)+dTh]])
128
129         #jacobian of f with respect to states
130         dFdXhat = matrix([[1, 0, -1*dD*sin(xhat.item(2))+dTh],
131                          [0, 1, dD*cos(xhat.item(2))+dTh], [0, 0, 1]])
132
133         #jacobian of f with respect to control input
134         dFdU = matrix([[ -1*dD*sin(xhat.item(2))+dTh],
135                        [0, 1, dD*cos(xhat.item(2))+dTh], [0, 0, 1]])
136
137         #Process noise covariance
138         Q = dFdU.dot(matrix([[sigmaTheta, 0], [0, sigmaD]]).dot(dFdU.transpose()))
139
140         #Process Error Covariance Prediction
141         P = dFdXhat.dot(P.dot(dFdXhat.transpose()))+Q
142
143         #Innovation Matrix
144         S = C.dot(P.dot(C.transpose()))+R
145
146         #Kalman Gain
147         K = P.dot((C.transpose()).dot(inv(S)))
148
149         #State Update
150         xhat = xhat_p+K.dot(Z-(C.dot(xhat_p - xhat)))

```

```
151
152 #Process Error Update
153 P = (numpy.identity(3)-(K.dot(C)))*P
154
155
156 def lowpassfilter(Z, Z_filt):
157     Z_filt = (.3758*Z)+(.6242*Z_filt)
158     return Z_filt
```

B Robot Wiring Diagram



C Connecting Xbee Modules in XCTU Software [11]

1. Install the XCTU software (<https://www.digi.com/products/xbee-rf-solutions/xctu-software/xctu>)
2. Connect your XBee S1 modules to their breakout boards and connect both to your computer with mini USB cables.
3. Click on the "Add Device Icon" and select the proper COM port of your Xbee module. The default settings for 9600-8-N-1 are recommended. Repeat for the other module

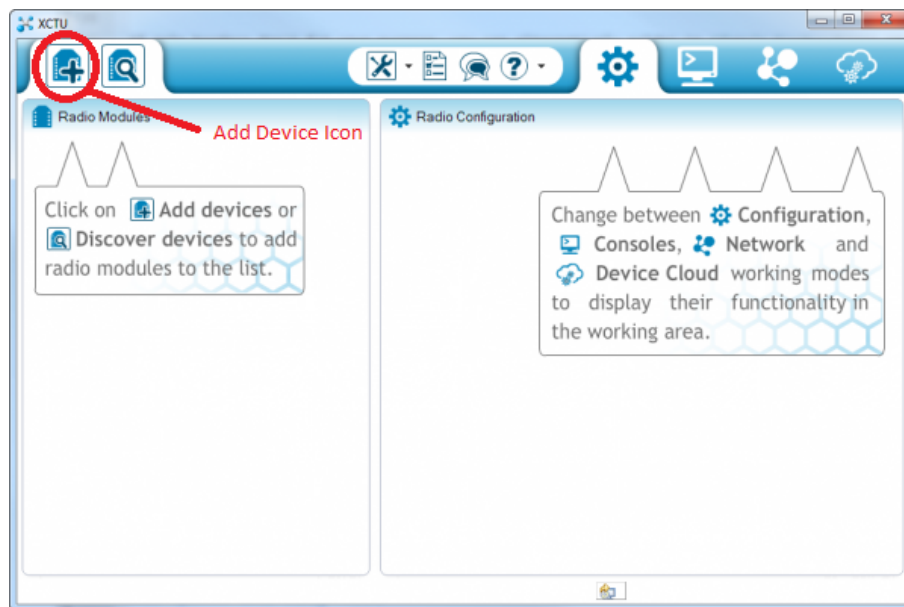


Figure 41: XBee Add Device Icon

4. To connect your XBee S1 modules wirelessly, click on the Console Icon. Then Click on the Link Icon, the border should turn green. You can type text into the console to send from one module to the other. Confirm the connection by sending a message in the console then selecting the other module to see if it received the message.

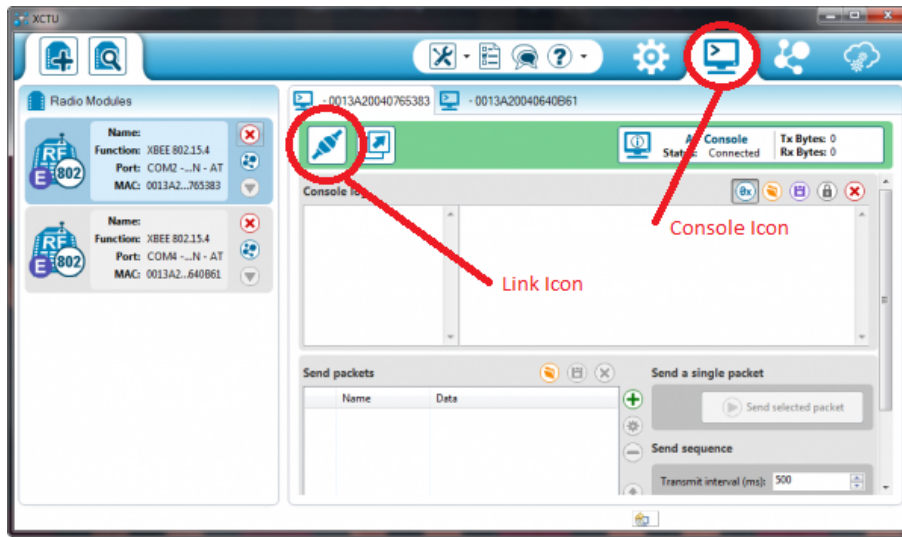


Figure 42: XBee Console Icon and Link Icon

5. Now that your XBee's are linked, you can disconnect one Xbee and Breakout Board from your computer and connect it to your microcontroller using the 5V Power and Ground. Make sure to connect TX to TX and RX to RX, unlike Arduino SoftwareSerial connections. For Arduino's these pins are usually pins 0 and 1.
6. With the other XBee module connected to the Computer, Open up a Serial Monitor connecting with the port of the XBee connected (Arduino IDE is recommended for this). You should be able to see what your microcontroller is printing. You have now established a wireless serial connection to your microcontroller.

D Cost Breakdown

Part	Quantity	Total Cost (\$)	Provider	Source
CO_2 Tanks	3	215.7	Praxair	MAD 1802 MQP
Robot Body Kit	1	174.95	Pololu	MAD 1802 MQP
Motor Driver Shields	2	99.90	Pololu	MAD 1802 MQP
Motors with Encoders	4	147.80	Pololu	MAD 1802 MQP
Wheels	8	29.90	Pololu	MAD 1802 MQP
Battery	1	35.26	Amazon	MAD 1802 MQP
Battery Charger	1	25.99	Amazon	MAD 1802 MQP
Motor Controllers	4	135.80	Pololu	MAD 1802 MQP
SprintIR CO_2 Sensor	1	141.55	CO_2 Meter	MAD 1802 MQP
Total Shipping Costs	N\A	41.00	N\A	MAD 1802 MQP
Total: \$ 1073.85				MAD 1802 MQP
Sensor Stand 3D Print	1	150.00	Higgins Labs	N\A
Diffuser 3D Print	1	40.00	Higgins Labs	N\A
Total: \$ 190.00				N\A
SprintIR CO_2 Sensor	3	424.65	CO_2 Meter	MAD 1801 MQP
Total Shipping Cost	N\A	14.60	N\A	MAD 1801 MQP
Total: \$ 439.25				MAD 1801 MQP
Poron Foam	1	15.49	McMaster-Carr	Spiridon Kasapis
Total Shipping Cost	N\A	7.22	N\A	Spiridon Kasapis
Total: \$ 22.71				Spiridon Kasapis
Xbee 24 S1	2	26.95	Amazon	Owen McGrath
Xbee USB to Serial	2	6.99	Amazon	Owen McGrath
Arduino Mega	1	11.99	Amazon	Owen McGrath
ESP 8266	1	6.99	Amazon	Owen McGrath
Crimp Tool	1	13.97	Amazon	Owen McGrath
Dupont Connector Kit	1	9.99	Amazon	Owen McGrath
Total: \$ 76.88				Owen McGrath
Gross Total: \$ 1802.69				

Table D.2: Project Cost Breakdown

E LabView Operation

1. Open LabView version 2017 or later and load the 'plume.vi'. Connect to the Arduino on the plume generation stand and then run the VI.
2. Setup a time interval for which you desire a gas flow. Turn the valve switch on. This will start a timer and the flow. Select a mass flow rate. This has been limited to 410mg/s and can be readjusted if you wish.
3. After the timer reaches the desired flow time, the valve turns off and the flow stops. You can then turn off the valve switch which will zero the timer and turn it back on or increase the experimentation timer.

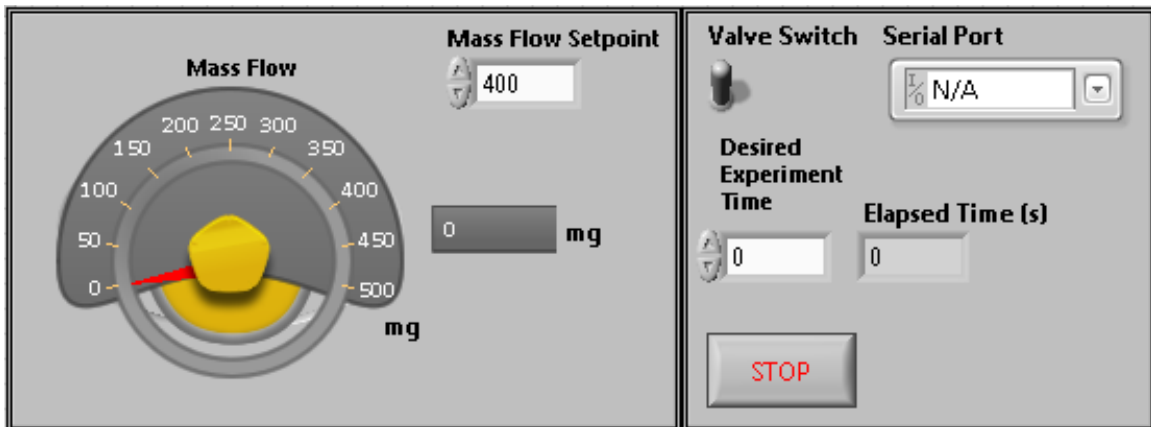


Figure 43: LabVIEW Front Panel Diagram.

References

- [1] Adafruit *Adafruit BNO055 Absolute Orientation Sensor* Retrieved From: <https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/overview>
- [2] Anglin, Mica; Hunt, Mitchell; Myles, Matthew *MAD 1501 Gas Source Localization with a Mobile Sensing Ground Vehicle* Worcester Polytechnic Institute, Worcester, MA, 2015.
- [3] Arduino *Arduino Mega* Retrieved From: <https://store.arduino.cc/usa/arduino-mega-2560-rev3>
- [4] Barney, M., Rivard, S. *NAG 1602 Design and Integration of an Indoor Plume Experimental Setup* Worcester Polytechnic Institute, Worcester, MA, 2015.
- [5] Christie, Nicholas; Colfer, John *MAD 1601 Plume Estimation Using a Gas-Sensing Mobile Robot* Worcester Polytechnic Institute, Worcester, MA, 2016.
- [6] Clark, Christopher; Greene, Mitchell; Seigle, Madeline *NAG 1501 Design of Plume Generation and Detection Systems* Worcester Polytechnic Institute, Worcester, MA, 2015.
- [7] CO2Meter.com *COZIR 2000 ppm CO2 Sensor* Retrieved From: <https://www.co2meter.com/products/cozir-2000-ppm-co2-sensor>
- [8] CO2Meter.com *GSS Sensor User Manual* August 2015
- [9] CO2Meter.com *SprintIR WR 100% CO2 Sensor* <https://www.co2meter.com/products/sprintir-100-percent-co2-sensor>
- [10] Coulter, R. Craig. *Implementation of the Pure Pursuit Path Tracking Algorithm*. Carnegie Mellon University, Pittsburgh, Pennsylvania, 1992.
- [11] Digi *Xbee Series 1 Wireless Serial Module*
Retrieved From: <https://www.sparkfun.com/products/8665>
- [12] Fast, E. D.; Harnais; S. M.; Wiesenberg, R. M. *MAD-1702 Plume Analysis and Detection*. Worcester Polytechnic Institute, Worcester, Massachusetts, 2017.
- [13] Hassanzadeh, Iraj; Abedinour Fallah, Mehdi *Design of Augmented Extended Kalman Filter for Real Time Simulation of Mobile Robots Using Simulink* 6th International Symposium on Mechatronics and its Applications Sharjah UAE (2009)
- [14] iRobot *iRobot Create2* Retrieved From: <http://www.irobot.com/About-iRobot/STEM/Create-2.aspx>

- [15] Kamenetsky, Max. *Filtered Audio Demo*. Retrieved from https://web.stanford.edu/~boyd/ee102/conv_demo.pdf
- [16] KTeam *Khepera IV Mobile Sensing Robot* Retrieved From: <https://www.k-team.com/mobile-robotics-products/khepera-iv>
- [17] Luong Van, Daniel; Katupitiya, Jayantha *An Adaptive Kalman Filter With Quadrature Encoder Quantisation Compensation* IFAC Mechatronic Systems, Sydney Australia 2004
- [18] NumPy *NumPy Scientific Computing Library for Python* Retrieved from: <http://www.numpy.org/>
- [19] Omegatron. (2008, July 4). *Butterworth Filter Response*. Retrieved from https://commons.wikimedia.org/wiki/File:Butterworth_response.svg
- [20] Otahal, Thomas J.; Tanner, Herbert G. *Extended Kalman Filter Implementation for the Khepera II Mobile Robot* (2009) University of New Mexico Mechanical Engineering Faculty Publications
- [21] Pololu *Dagu Wild Thumper 4WD All Terrain Chassis* Retrieved From: <https://www.pololu.com/product/1565>
- [22] Pololu *Pololu Simple Motor Controller User's Guide* 2017
- [23] Pololu *Pololu Simple Motor Controller 18v7* Retrieved From: <https://www.pololu.com/product/1372>
- [24] Pololu *75:1 Metal Gearmotor 25Dx54L mm HP 6V with 48 CPR Encoder* Retrieved From: <https://www.pololu.com/product/2275>
- [25] Roder *Cozir CO2 Sensor Arduino Library* Retrieved from: <https://github.com/roder/cozir>
- [26] Smith J.O. *Introduction to Digital Filters with Audio Applications*, <http://ccrma.stanford.edu/~jos/filters/>, online book, 2007 edition, accessed February 25, 2018.
- [27] Welch, Greg; Bishop, Gary *An Introduction to the Kalman Filter* UNC Chapel Hill Department of Computer Science (2006)
- [28] Wu, Xiaodong; Xu, Min; Wang, Lei *Differential Speed Steering Control for Four-Wheel Independent Driving Electric Vehicle* International Journal of Materials, Mechanics and Manufacturing Vol. 1, No4. November 2013