Worcester Polytechnic Institute

Robotics Engineering Program

RBE 2002 Firefighting Challenge

SUBMITTED BY
Connor Craigie
Nathan Stallings
Zeynep Seker

Date Submitted	**: 04/30/2019**
Date Completed	**: 04/29/2019**
Course Instructor	**: Prof. Putnam**
Lab Section	**: D02**

Abstract

This project was centered around designing a robot that is able to autonomously navigate an unpredictable cityscape and detect a fire in an unknown location. It then needs to report the location of the flame and blow it out. Rather than searching each block, our team designed a robot that could extinguish the flame from intersections. Our unique nozzle designed allowed us to do this with ease along with blowing out a flame on either the first or second floor. This, combined with an intelligently designed searching algorithm, allowed our robot to traverse the city and act as a formidable fire engine.

Introduction

For the “Firefighting Challenge” we were asked to design and build a robot that is able to detect an arbitrarily placed fire and extinguish it. The fire may be located either on the top floor or the bottom floor. The fire, in our case a zippo flame, will not be visible from our initial position. The robot must, autonomously, detect the fire. The robot should also find its way through the streets without getting in contact with any of the assigned building spaces, and modify its path according to the road block’s location. In order the reduce the number of actuators on our robot, we decided to incorporate a split end nozzle and blow out the flame from the same position whether its located on the top or bottom level.

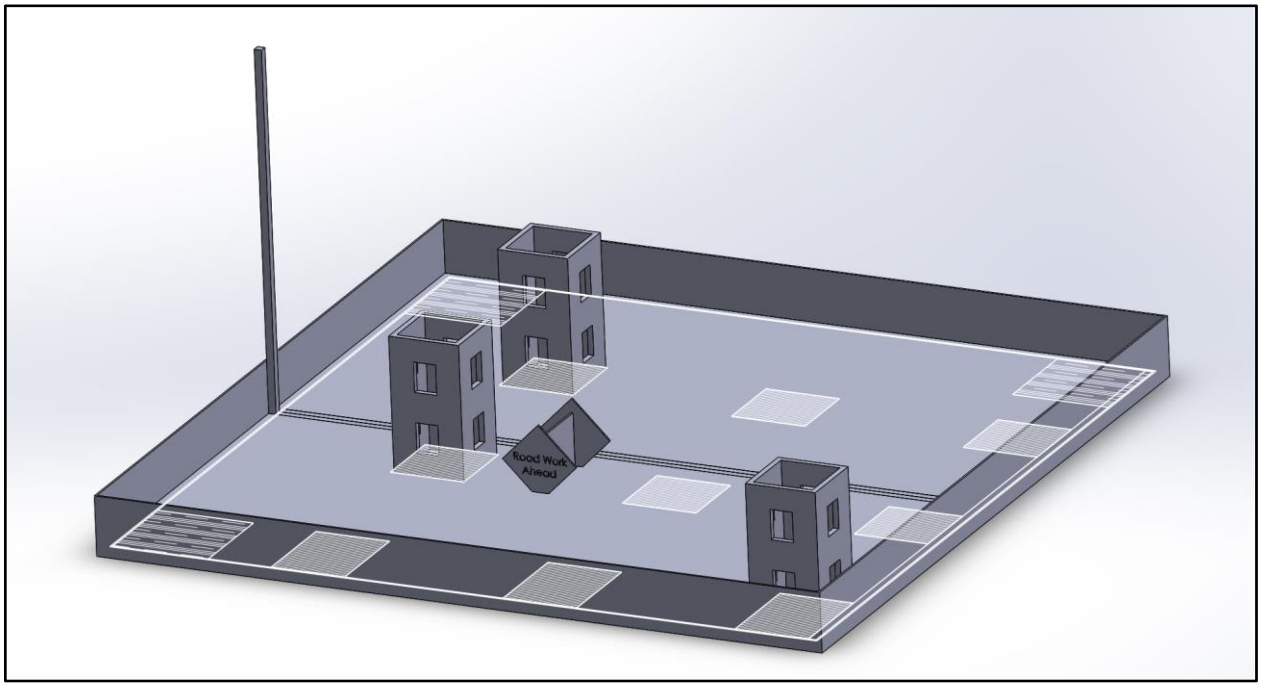


Figure 1: Render of the Game Field

Methodology

Drive Train

The driving chassis was provided to us in our base-bot configurations. We decided to use the provided configuration with a few modifications to fit our needs. One modification we needed to make was to rotate the drive motors. This moved the location of the wheels closer to the front of our robot, which shifted our turning center to directly under the turret. Not only did this provide a central point of rotation, but it also provided us room to place our lead-acid battery. The other change we made was to move the caster wheel to the back of the robot. We placed the lead-acid battery on the back of the robot, which shifted our center of gravity in between the wheels and ball caster. This prevented the robot from tipping. We made some slight modifications to the drive module as well. The gears continued to slip down the motor shaft causing the wheel module to fail. This caused problems with our driving functions. In order to fix this, we took Kevin Harrington's STL file for the small bevel gear on the motor shaft and derived it as an SLDPRT file using sketches in SolidWorks. This allowed us to model a new gear with an extended collar to prevent slipping. Besides these changes, the setup was the same for the base-bot. The Pololu Motors worked well for what we needed and their online data sheets allowed us to perform calculations to ensure that the 3 to 1 gear ratio would satisfy our needs. We calculated the stall torque to ensure that our wheels would not stall upon collision. We did this by using information about the motors from their data sheet. In order to calculate the torque that the motors would stall at, we took the provided stall torque and multiplied it by the drivetrain gear ratio.

$$T_{in} := 170\text{ozf}\cdot\text{in}$$

$$T_{Stall} := T_{in}\cdot\left(\frac{Gear_{Driven}}{Gear_{Driving}}\right) = 31.875\cdot\text{lbf}\cdot\text{in}$$

Figure 2: Calculating Stall Torque

The wheels will slip if the stall torque can overcome the frictional force. The frictional force is found with the equation. equation $F_f=\mu*W$, where W, the weight of the robot, is 8.682 pounds, and μ is the coefficient of friction. We can set up a sum of moments around the wheel to determine what will happen upon collision. The equation $\Sigma M = 0 = T - (r * Ff)$, where T is the stall torque, and r is the radius of the wheel, which in our case is .95", will tell us if the stall torque will be strong enough to overcome the force of friction between the wheel and the surface. Plugging in our known values and solving for the coefficient of friction gives us a coefficient of 3.864. Since the surface of the field is relatively smooth, it is safe to assume that the coefficient of friction is not 3.864, so our motors will slip rather than stall. This was not a large concern for us however, as our robot had enough torque to push the buildings or roadblock upon collision occurring during testing.

Split End Nozzle

The nozzle *(Figure 5)* inputs the air flow created by the EDF fan with a brushless motor. Two outputs cause air to flow through both nozzles that are identical in diameter. The upper nozzle is 28.87 degrees inclined and the lower nozzle is 8.81 degrees declined *(Figure 2).* The reason behind this nozzle shape is merely to put out the “fire” on each level of the building without adding an additional actuator for fan aim. This design assures that no additional current will be drawn from our battery. Also, programming will be relatively easier since we have one less motor/servo to program.

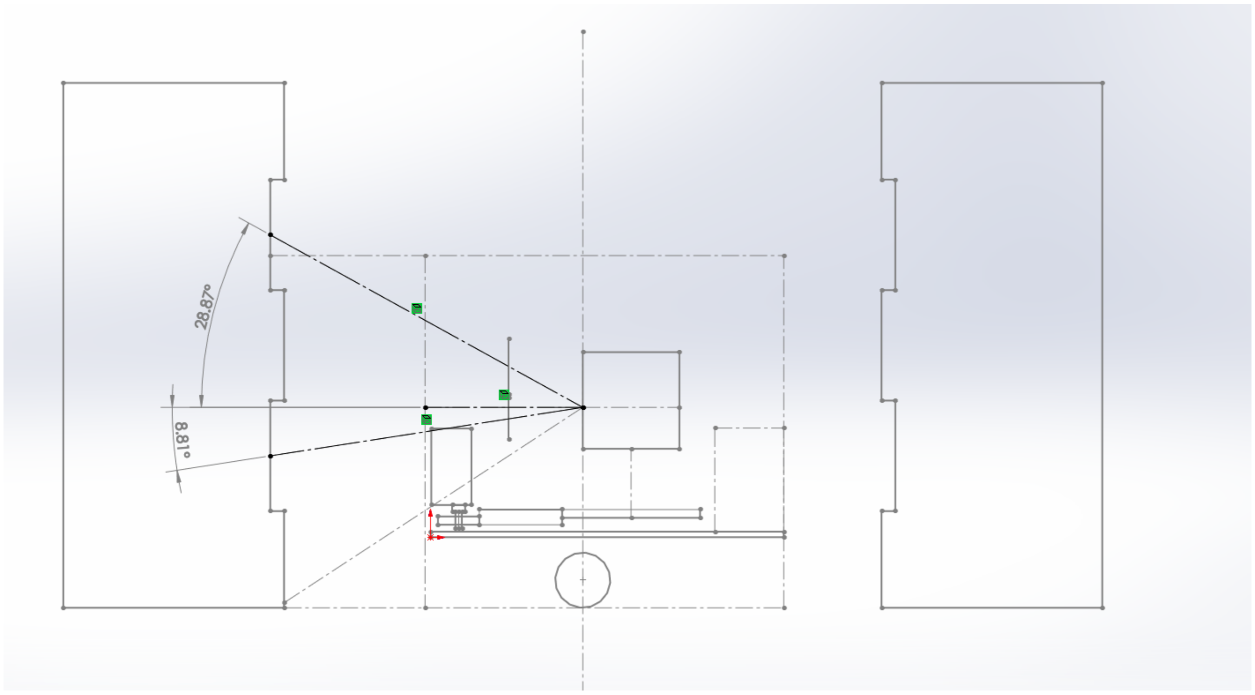


Figure 3: Vertical Nozzle Angles

We would not be able to control the airflow within the nozzle manually. Therefore we designed it in such a way that the air flow would be divided perfectly in half and travel through each individual outlet at the same rate. In order to achieve this, we have a dual extrusion to separate the airflow. We used this method to get as close to the threaded diameter as possible to avoid turbulence within the nozzle. Initially, we were planning on designing rectangular outlets. However, because sharp corners would cause minor losses we decided to continue with circular outlets. In order to confirm this design, we ran a flow simulation in SolidWorks *(Figure 4).* Also, we designed the nozzle to have threads. This way, the system is modular and can be replaced if need be *(Figure 5).*

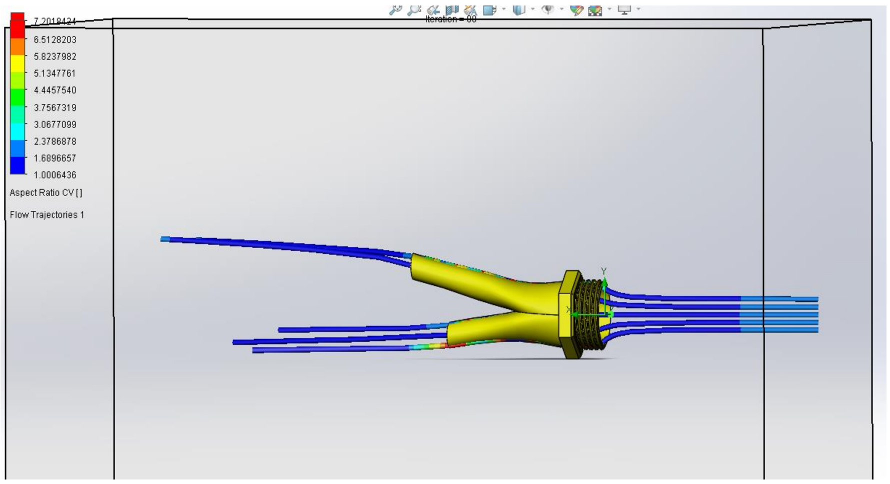


Figure 4: Flow Simulation

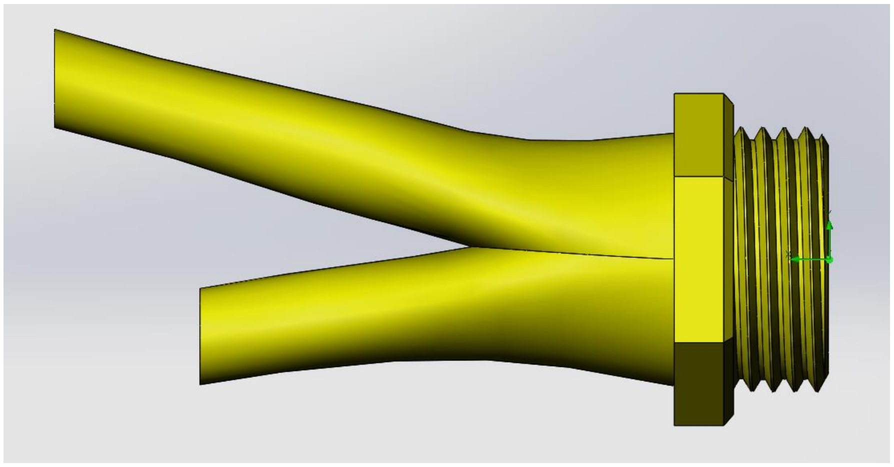


Figure 5: Threaded Nozzle Head

For the nozzle output diameters, we calculated the necessary air velocity to put out a Mini Zippo Lighter with windproof protection. In order to calculate this velocity, we spun the model building with the fire on the bottom level and timed the moment the fire went out *(Figure 6.a)*. By knowing the time and position during the rotation the fire went out, we were able to calculate its rpm value, and then convert that number into a tangential velocity. After our experimentations, we noticed that we need at least 3.5325 m/s to blow out one of these lighters (See *Figure 6.b* for the full air velocity experiment calculations and see *Figure 7* for the full air flow calculations).

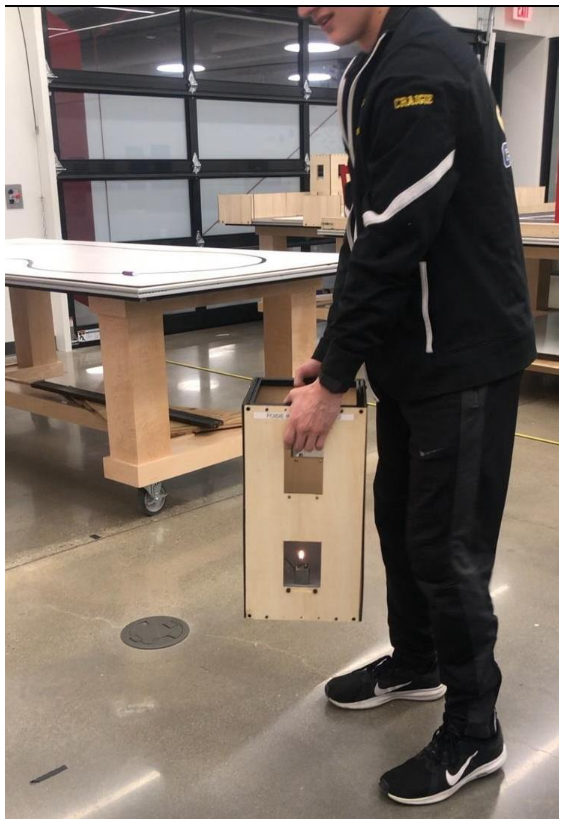


Figure 6.a: Setup for Velocity Experiment

The calculations for the diameters of the respective nozzles are based on Bernoulli's equation and the steady state flow principle. According to the steady-state principle, the cross-sectional area times the velocity of fluid will stay throughout the system as long as the mass flow rate is constant throughout the system. Since we only have one input, the EDF fan, we may assume the system is in steady state for the simplicity of our calculations. In this equation, we know our inlet diameter since it is a constant value obtained by the motor's diameter, we also know the minimum output velocity.

We have one inlet and two outlets on this nozzle. In order to sustain the steady flow principle, we must make sure $A_{in}V_{in} = A_{out,1} V_{out,1} + A_{out,2} V_{out,2}$. In our case $V_{out,1}$ and $V_{out,2}$ must be equal since we need a minimum of 3.5 m/s to blow out the Zippo lighter. With this knowledge, we can assume that the outlet diameter will also be equal to one another. So, the only value we are missing is the input velocity which we can determine by trying different power percentages for the brushless motor. After trying multiple percentages we finally decided on 60% of power. 60% power is more than enough to blow out the flame. We need at least 3.5 m/s. However, we preferred to have a faster airflow to make sure that we compensate for any errors that might have occurred during our velocity experiment.

After we completed our design and built the robot we turned on the fan to measure the output velocity through each outlet using an anemometer. We used a PYLE PMA85 model anemometer which has a limit of 30 m/s flow speed *(Table 1)*[2]. Both the upper an the lower outlets output a velocity that is out of the anemometer range (*Figure 8.a* and *Figure 8.b*). We obtain more than enough velocity to blow out the flame.

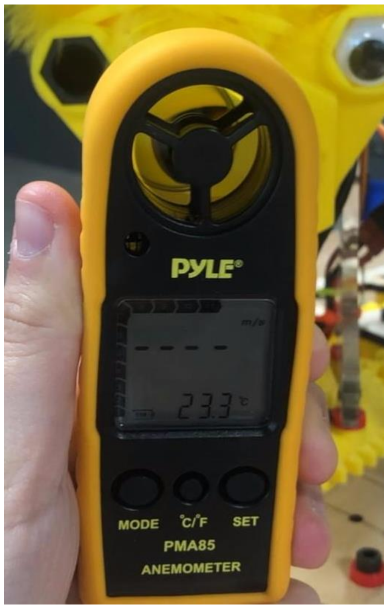


Figure 8.a: Upper Nozzle Output Velocity

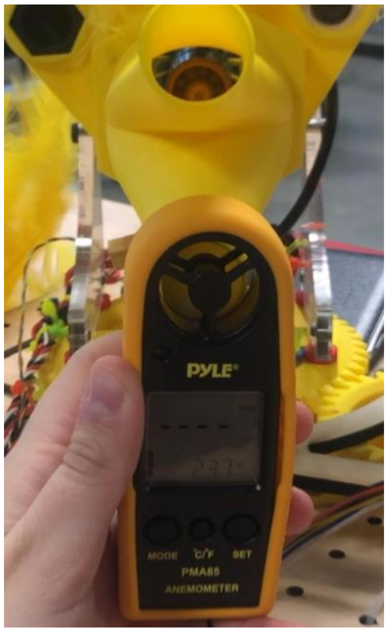


Figure 8.b: Lower Nozzle Output Velocity

Lazy Susan

Our IR Camera Sensor is located on the upper corner of the nozzle which rotates on our 3D printed turret. The turret is constructed similar to that of a lazy susan. The design of the main revolving gear includes a small track in order to capture a set of 3 roller bearings that act as a low friction surface. This main turret is used to search the field by pivoting the nozzle and IR Camera at fixed angles instead of rotating the robot with numerous drive functions. Using the turret to poll fire locations would accumulate less error overall. Neither the base-bot drive system or turret rotation is an ideal system. Turret rotation will accumulate a degree or two of error over multiple repeated polling sequences. However, according to our practice runs, we noticed that checking fire with the base-bot drive systems accumulated more error. Therefore we decided it was optimal to isolate the fire polling to a separate rotating turret system *(Figure 9).* Our lazy susan design consists of 3 layers. The bottom layer is a 3D base piece that attaches the 2nd layer of 3 ball casters to the wooden base. Finally, the 3rd, top layer is shaped like a gear and is the base for the nozzle.

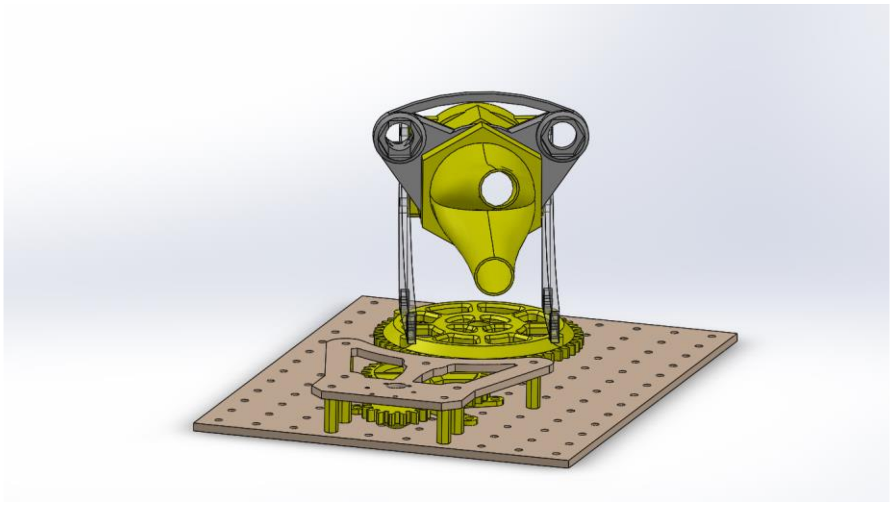


Figure 9: Overview of the Lazy Susan

We used a three-stage gearing with the ratio 50:15 and a 20-degree pressure angle *(Figure 10)* to get the revolution we wanted from our third motor. By this gearing, we are able to rotate the IR Camera and the nozzle all at once so when we detect the flame we can initiate the proper case statement and turn on the fan.

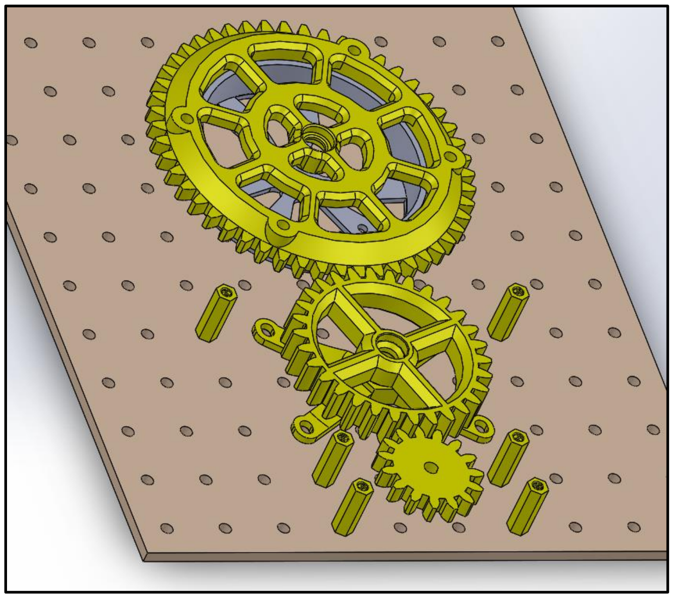
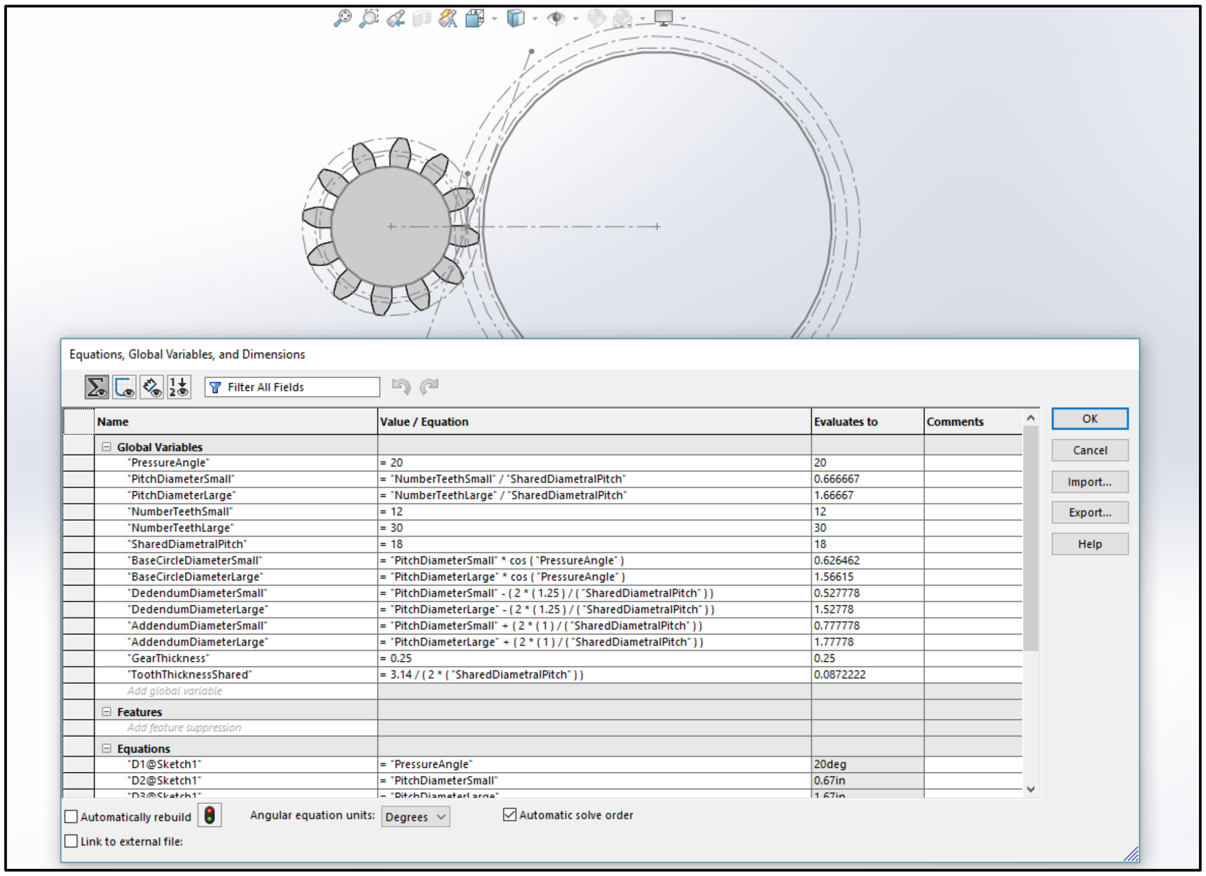


Figure 10: Lazy Susan Gearing

Gear Generator



Name	Value / Equation	Evaluates to	Comments
Global Variables			
"PressureAngle"	= 20	20	
"PitchDiameterSmall"	= "NumberTeethSmall" / "SharedDiametralPitch"	0.666667	
"PitchDiameterLarge"	= "NumberTeethLarge" / "SharedDiametralPitch"	1.66667	
"NumberTeethSmall"	= 12	12	
"NumberTeethLarge"	= 30	30	
"SharedDiametralPitch"	= 18	18	
"BaseCircleDiameterSmall"	= "PitchDiameterSmall" * cos ("PressureAngle")	0.626462	
"BaseCircleDiameterLarge"	= "PitchDiameterLarge" * cos ("PressureAngle")	1.56615	
"DedendumDiameterSmall"	= "PitchDiameterSmall" - (2 * (1.25) / ("SharedDiametralPitch"))	0.527778	
"DedendumDiameterLarge"	= "PitchDiameterLarge" - (2 * (1.25) / ("SharedDiametralPitch"))	1.52778	
"AddendumDiameterSmall"	= "PitchDiameterSmall" + (2 * (1) / ("SharedDiametralPitch"))	0.777778	
"AddendumDiameterLarge"	= "PitchDiameterLarge" + (2 * (1) / ("SharedDiametralPitch"))	1.77778	
"GearThickness"	= 0.25	0.25	
"ToothThicknessShared"	= 3.14 / (2 * ("SharedDiametralPitch"))	0.0872222	
Add global variable			
Features			
Add feature suppression			
Equations			
"D1@Sketch1"	= "PressureAngle"	20deg	
"D2@Sketch1"	= "PitchDiameterSmall"	0.67in	

Automatically rebuild [ ] Angular equation units: Degrees [✓] Automatic solve order

Link to external file: [ ]

Figure 11: SolidWorks Gear Generator

The turret's gearing was created using a generator the team developed. The generator is driven by Solidworks geometric relations. Using the built-in equations functionality, we can produce the significant features of a standard involute gear. With these relationships, we can input a given number of teeth, diametral pitch, and pressure angle. With these inputs, the base circle, dedendum circle, and addendum circle are derived and visualized through construction sketches as seen above in *Figure 11*. The involute curve is derived geometrically from the base circle. The involute curve is commonly visualized as the path made by the end point of a taut string being unwound from a spool as seen below in *Figure 12*.

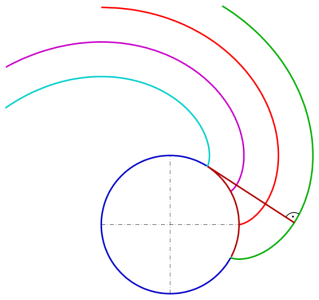


Figure 12: Example Involute Curve

This relationship is approximated in our generator. Geometrically the involute curve is formed by an infinite collection of points, these points are taken at the end of a moving tangential line. This line begins with a length of zero directly on the edge of the base circle this can be seen in *Figure 12* where the red curve intersects the base circle. As the tangential line is rotated around the circumference of the circle, the tangential line grows in length so that it is always equivalent to the arc length of the base circle that it has passed. To approximate this relationship, we sample this five times. Five equally spaced arc length distances are constructed. There are five tangential lines extending from the end of each equally spaced arc length. As stated before, the tangential lines are as long as the arc lengths they extend from. The five points which exist at the end of each line are mapped in 2D space. These points are used to define a generic spline. This spline acts to dampen the error in the involute approximation. From here we can use basic modeling techniques to mirror and pattern the teeth of each gear.

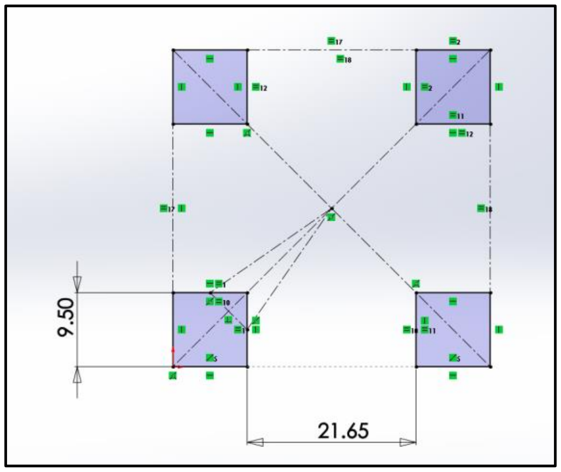


Figure 13: Aiming From a Node

Our strategy is to search the field at approximately 45-degree angles as shown in *Figure 13*. The lazy susan can rotate roughly 300 degrees. Our routing of wiring restricted the turret's full rotation. The ESC power cabling, ESC signal cabling, and IR camera I2c communication cables are all routed externally around the frame of the robot. These cables are tied in a fashion that keeps them away from the gear mesh while also allowing them to run in open space. The turret will only be used to poll a range of 270 degrees (4 directions offset by 90 degrees each). Therefore the turret will have more than enough rage to successfully poll its required positions.

Code

The variable cityscape poses a real challenge. Navigation and search patterns must be refined to assure every fire location is checked. Our strategy was based upon an important statement in *Playing Field Description* on page four under *Building/Fire Placement*. The statement noted that "the location of the fire will be chosen such that there is no direct line of sight from the robot's starting position to the fire". From this statement, we have decided to begin our scan from fire station Beta. While in beta, the exterior flame locations of A, B, C, D, and G will not be active. Therefore only the interior flame locations will have to be searched.

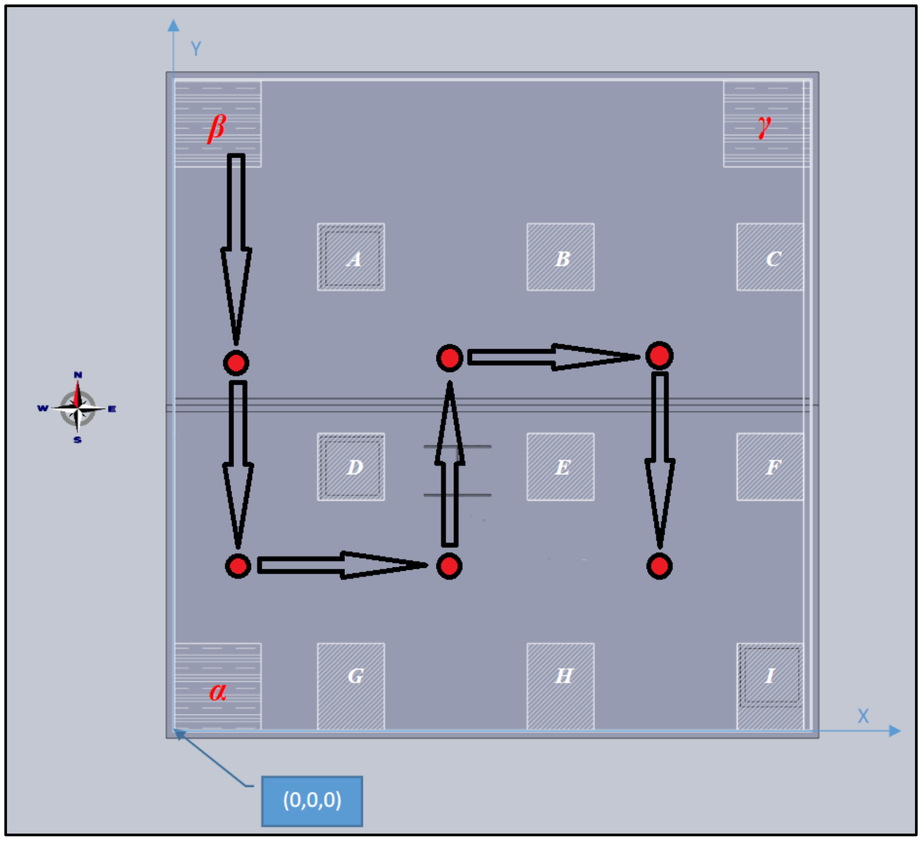


Figure 14: Strategy

Above, *Figure 13* depicts the predetermined path that we have set our robot to navigate. The red nodes represent stopping locations. The last four nodes in the predetermined pathway are used for polling the inner windows. From these nodes, we poll in four directions 90 degrees apart. Each poll checks all four windows in every direction. With numerous tests, we have determined that from 10 inches above the ground, the IR camera can see flames in both windows as long as it is 16.5 inches away. From these node locations, each flame we detect is roughly 19 inches away and therefore in our range of detection

One of the largest challenges we faced was traversing the cityscape. We were assigned to develop a driving algorithm that would include the functionality of the IMU to assure our robot stays on heading. The given *driveInterpolationDegrees* method only uses the encoders of each motor to detect raw displacement of the robot. Due to this limited sensing environment, it becomes very common for inaccuracies to occur in the form of heading inconsistency and drift. With our algorithm, we are constantly polling the IMU for directional data while also checking the raw displacement of the motor encoders. Using these three sensors we can create an environment that we, the programmers, can manipulate. A PID was created to proportionally affect the written velocity of each motor. This is done with respect to the error which exists in our active heading. In parallel to the heading PID, there is a second PID which proportionally affects the written speed of each motor relative to the overall error that exists in our linear displacement. In this second PID, we create a cap velocity to assure the robot moves slowly and consistently. Therefore the displacement PID will only act as a velocity ramp down near the setpoint displacement. This velocity cap keeps the robot at a controllable speed. Without it, the wheels are written initially to a high velocity and experience a loss of friction causing an inaccuracy in our final displacement.

Our state machine was located in StudentRobot.cpp. It started in a *buttonhit* state that constantly polled for the start button before jumping into the first driving state. Once it detected a push, it moved forward 925 millimeters. This was a measured movement to drive us out of the fire station. The next step in the state machine was to check for any obstacle in its way. Although our roadblock code did not work perfectly, we were still constantly checking for obstacles before every forward movement. If no obstacle was detected, the next state isone, and it drove forward a unit, which was a constant of 840 millimeters that we measured to get from one node to another. One was the first node we wanted the robot to search. It was directly in front of fire station alpha and, although we did not need to search the outside windows, it helped us find inner flames and lowered the chances of the robot not detecting some of the fire locations. After driving, it went into a *turn90* state, which rotated the robot 90 degrees and reset the theta heading to a new location. This helped our PID use the IMU to maintain on course. The *turn90* degrees state jumped to the turret state. In the turret state, we used the third motor to move the turret to 45-degree angles. We found the correct values to send to the motor through trial and error. We used the same motor PID we made for 2001, as it worked for us then. Once the motor determined it was at the correct value, the state switched to a fire detecting state. This state read the IR camera 4 times every quarter of a second. This ensured that we would not miss the flame if it flickered. If a flame was detected, the robot would report the address and floor of the flame. We did this using an accumulating variable, which increased everytime the turret moved. Since we knew every location the turret would turn at, and we knew how many times it would turn, we could report the address of the flame based on how many times the turret was already truend. We reported the floor by looking at the Y position of the flame If it was above 350, then the flame was on the second floor. The location was then sent to Bowler Studio using the coms packaged and bowler studio reported the location of the flame using the text-to-speech function. Once the flame was detected, the state machine switches to a blow state, which sent a signal to the EDF fan to turn it on for seven seconds. Once the time was up, the fan was turned off and the state switched to a drive home state. This state was never finished, but given more time, it would have directed the Robot home based on the current state it was in which would have been recorded with a variable. If a flame was not detected, it would return to the turret state and move it to the next 45-degree location. If it checked all 4 buildings without detecting a flame, it proceeded to the next state, which moved the robot to the next node using out unit distance. This process repeated itself until the robot searched the whole field. If the robot got to the end of the course without detecting a flame, it would enter the halting state.

Results

The robot was successful in completing the challenge. Our state machine had the robot navigate the cityscape in a fixed pattern. Our robot traversed the course, halted at the correct node locations, and polled the proper windows. The software properly reported the flames elevation and street intersection to the Bowler Studio terminal. Due to imperfections in heading manipulation, our roadblock avoidance methods would run into errors when re-entering the predetermined state machine.

Our unique design of the nozzle allowed us to successfully produce enough force to blow out a flame at either location. We needed a minimum force of 3.5m/s to extinguish the flame, and we obtained more than 30 m/s of airflow at the end of our experiments. We used steady-state flow to determine our minimum input velocity and nozzle diameter. We used circular outlets to reduce the amount of turbulence in the nozzle. Because the speed of the air flow was faster than the anemometer range, we were not able to calculate if the output velocities were identical or not. Because we cannot create an ideal system easily, we believe that there was actually turbulence in the nozzle, but increasing the power of the fan compensated for our loss, so we were able to extinguish the fire easily.

The lazy susan proved to be a great method for us to rotate the turret. The ball casters provided three points of contact, and therefore a low friction planar surface to rotate about. Mechanically, the system was sturdy. The gear teeth generated were involute and therefore had a consistent pressure angle. This assures smooth rotation, near to no backlash, and continuous contact while actively driving the system. After about an hour of testing, our refined PID gains allowed the turret to turn consistently and accurately to our desired distances. The collective agreement between mechanical repeatability and properly tuned PID gains provided us with consistent angular displacements. This allowed us to locate and examine the four buildings around an intersection from the node point, which saved us a good amount of time. This played nicely into our strategy as it was more efficient than checking every possible building location.

Discussion

Since we had a limited time to completely search the whole field, it was very important to efficiently navigate the city. Our fixed pattern scan allowed us to travel to four crucial nodes that would give us sight of all possible flame locations. This allowed us to quickly and successfully search the field. Since the node distance was equidistant from each other, we could plan out single unit movements and turns. This allowed for simpler code than other strategies and saved us a good amount of time required to search the field for the flame.

Our flame detection method was also unique and played an essential role in our success. Most teams had to motorize their IR camera to look at the top and bottom floors of each building. Since we looked at buildings from the intersections, our tests showed that the IR Camera would be able to see both the top and bottom floor. We could then report what floor the fire was on by comparing the Y-Axis value of the IR Camera to a constant that we knew split up the first and second floor. After a few tests, we saw that it was value needed to be 300. Once we could detect the flame and the floor the flame was on, reporting the address was easy. We knew how many times the turret would have turned at each point on the field. This allowed us to set up an accumulator that kept track of how many locations the turret has looked at. Based on where the turret was looking, we could use the accumulating variable to report the address of the building that was being searched. This information was then sent to bowler studio where it reported the location of the flame upon detection.

Our roadblock avoidance was not as successful as we would have liked. In theory, the algorithm for avoiding the roadblock would work very well for our strategy. Once a roadblock was detected, we could simply avoid it to the right or to the left in order to get to the next intersection, and from there jump back into the state machine. However, our team encountered problems when it came to implementing the code using the IMU. We mapped our IMU to limit the number of times we would have to deal with the 0-359 jump. However, no matter how we avoided the roadblock, we would still have to face this problem at some point. We did not have the time to problem solve and debug this part of our code. This is something that we would like to fix in the future, as we believe it would work perfectly with our searching algorithm given more time to debug and implement properly.

The combined lazy susan and turret mechanism worked very well in this small robot application. Luckily, the ball bearings were not required to undergo a large amount of force. Their physical properties allow for smooth and repeatable rotation. However, one oversight was the wire routing to electronics onboard the turret. There are three main electronic components that exist on the turret. The electronic speed controller, brushless motor, and IR flame recognition camera. In hindsight, it would have been much safer and effective to place the wire routing through the center column of the turret. With a center running wire column, the turret would be able to rotate freely with very little restriction to rotational displacement. The only physical concern with a center column would be an excessive amount of torsion. This would only exist if the turret was programmed to freespin excessively in a single direction. This situation can be avoided programmatically so that the turret always returns to a zeroed position.

Our overall nozzle design worked fine. We were able to extinguish the flames on both levels without a problem. Instead of increasing the power to compensate for the error in calculations and friction/ turbulence within the nozzle, we could have bathed the nozzle in acetone to smooth the inner walls more. We could have also extended the double extrusion in a way that the input diameter of the fan would be divided perfectly in half. However, because we had a very strong brushless motor that controlled the EDF fan, we preferred to increase the speed and consider the values of turbulence and friction as negligible factors in the system. In other words, we treated our system as an ideal system during our calculations and then tried to make up for it with an excessive power rate.

Conclusion

Our team designed a robot that could successfully and consistently complete the challenge. Our unique nozzle design allowed us to blow out both top and bottom flames simultaneously, which helped save us an actuator. Furthermore, the nozzle design and strength of the fan allowed us to blow out the flame from the center of the intersection. This was a huge benefit for us, as it eliminated the robot’s needs for movement, which could build up error. It also allowed us to maintain a simple search algorithm that could easily cover the field under the five-minute restraint. All of these successes don't come without a few failures. We were unable to consistently avoid a roadblock. Future work and time dedicated to this project would consist mostly in problem-solving the roadblock. After a limited amount of debugging, we decided the problem in the code was coming from our *Thetaheading* value and how we mapped the IMU. In future renditions, it might be smart to reset the value of the IMU after each turn or remap it more efficiently to prevent complicated and hard to debug code. We would also like to implement the return home function. We had plans to keep track of where the robot is located to help it return home after the flame was smothered, but never had the time to implement and debug the code. Our team was very pleased with the performance of our robot and how we successfully worked together as a team.

References

[1] Putnam C. (n.d.). *RBE 2002 D19*. Lecture presented at WPI Robotics Program in

Massachusetts, Worcester.

[2] Pyle. “Digital Anemometer - Instruction Manual .” *PyleMeters - PMA85 - Tools and Meters - Temperature - Humidity - Moisture*, Pyle, www.pyleaudio.com/sku/PMA85/2-in-1-Digital-Anemometer-and-Thermometer---Air-Velocity-(Wind)-and-Temperature-Meter.

Appendix

Figures:

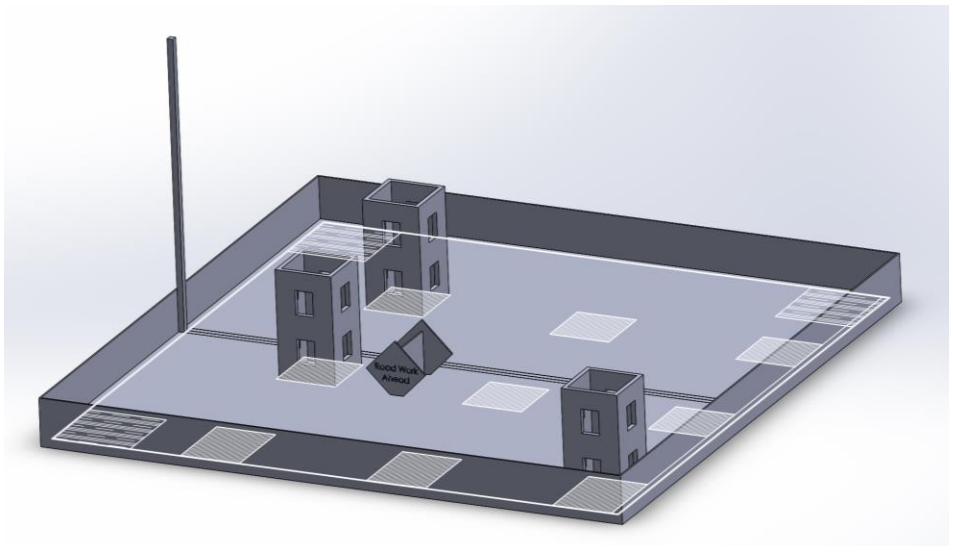


Figure 1: Render of the Game Field

$$T_{in} := 170ozf \cdot in$$

$$T_{Stall} := T_{in} \cdot \left(\frac{Gear_{Driven}}{Gear_{Driving}} \right) = 31.875 \cdot lbf \cdot in$$

Figure 2: Calculating Stall Torque

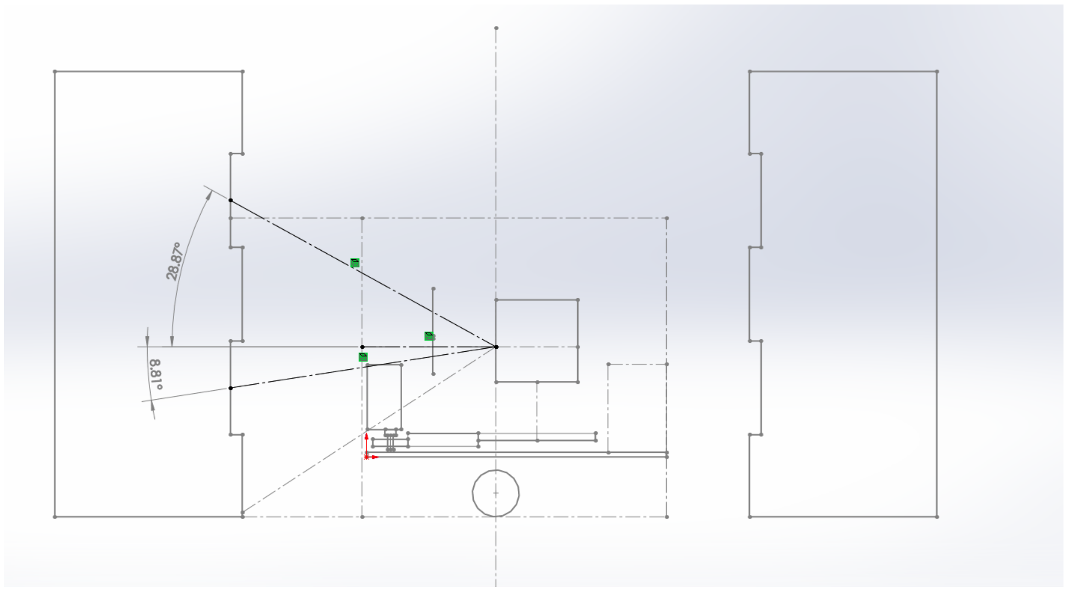


Figure 3: Vertical Nozzle Angles

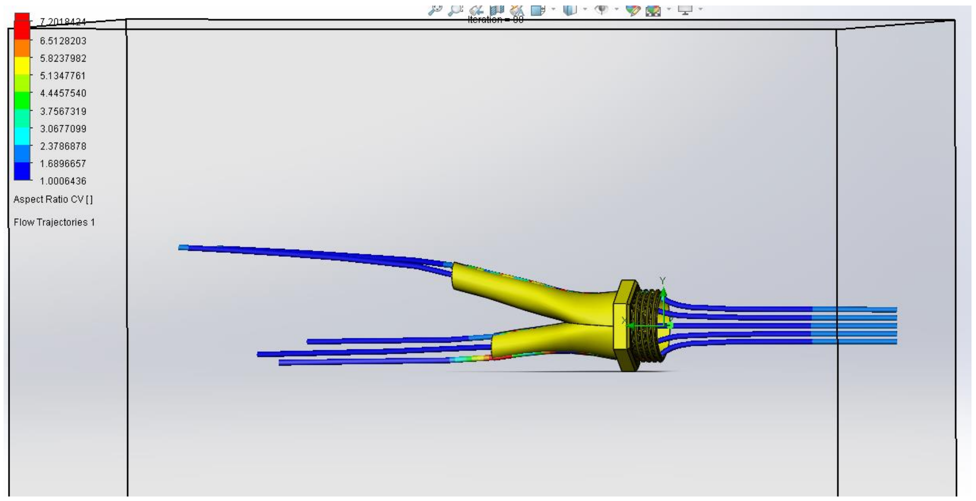


Figure 4: Flow Simulation

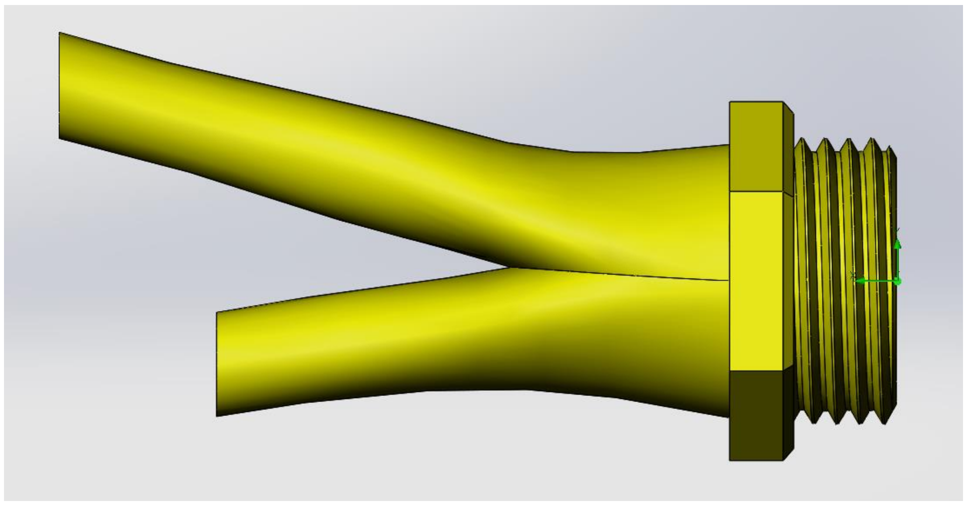


Figure 5: Threaded Nozzle Head

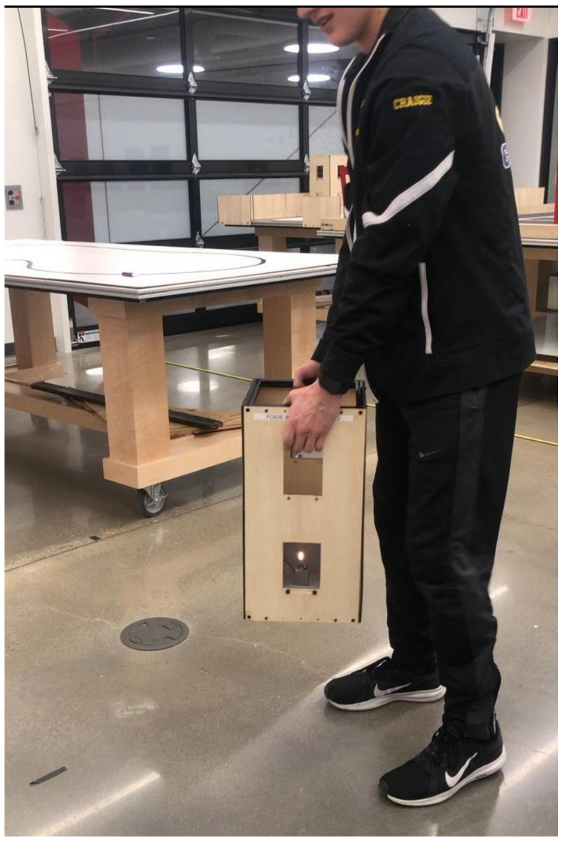
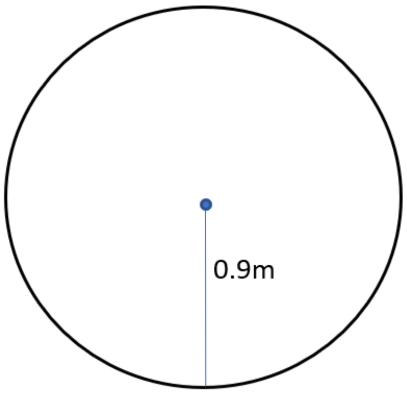


Figure 6.a: Setup for Velocity Experiment



$90^{o} \rightarrow 0.4$ sec

$360^{o} \rightarrow 1.6$ sec

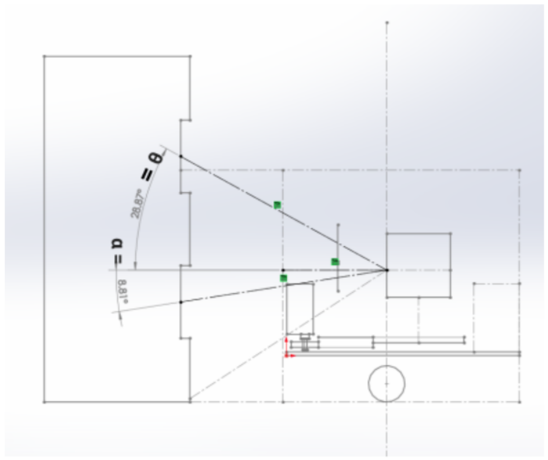
1 rev $\rightarrow$ 1.6 sec

$$\frac{60}{1.6} = 37.5\, rpm$$

$$1\, rev = 2\pi(0.9) = 5.652m$$

$$\frac{5.652m}{1\, rev} * rpm * \frac{rev}{min} = \frac{(5.652)(37.5)}{60} = 3.5325\, m/s$$

Figure 6b: Air Velocity Calculations



A_{in}=0.0031258002 m^2

$V_{upper}\cos\theta \geq 3.5325$ m/s

$V_{lower}\cos\alpha \geq 3.5325$ m/s

$(0.0031258002\ m^2)\ V_{in} = A_{upper}V_{upper}\cos\theta + A_{lower}V_{lower}\cos\alpha$

$(0.0031258002\ m^2)\ V_{in} = A_{upper}V_{upper}\cos(28.87) + A_{lower}V_{lower}\cos(8.81)$

$(0.0031258002\ m^2)\ V_{in} = A_{upper}V_{upper}\ (0.8757) + A_{lower}V_{lower}\ (0.9882)$

$V_{upper,min}$=4.0339 m/s

$V_{lower,min}$ = 3.5747 m/s

$A_{upper} = A_{lower} = A_{in}/2 = 0.001563\ m^2$

$V_{in,min}$= 3.533 m/s

Figure 7: Full Air Flow Calculations

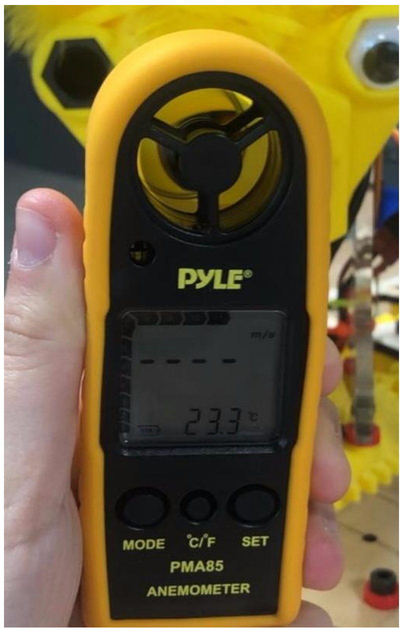


Figure 8.a: Upper Nozzle Output Velocity

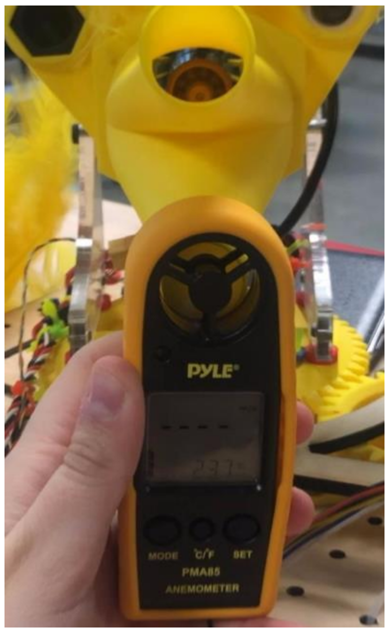


Figure 8.b: Lower Nozzle Output Velocity

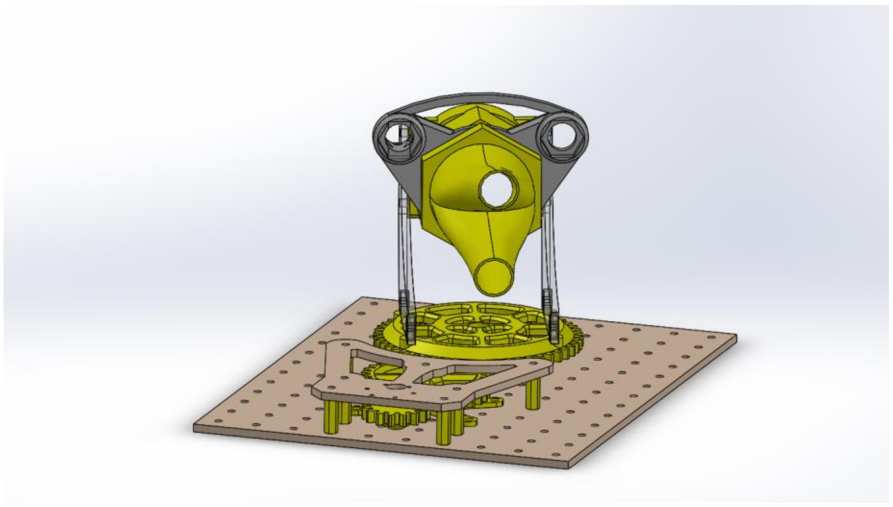


Figure 9: Overview of the Lazy Susan

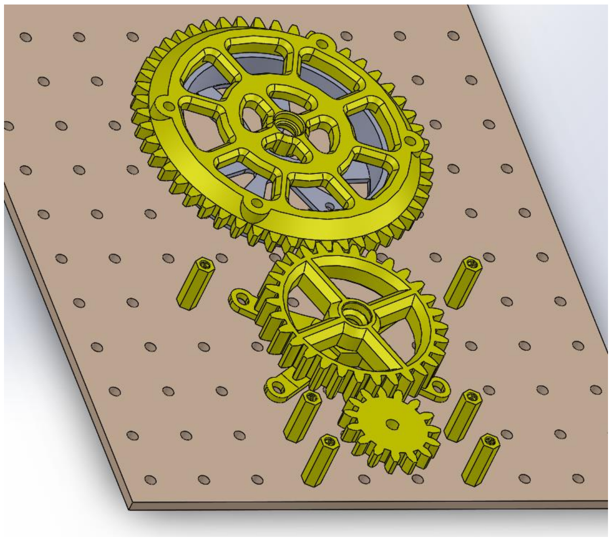
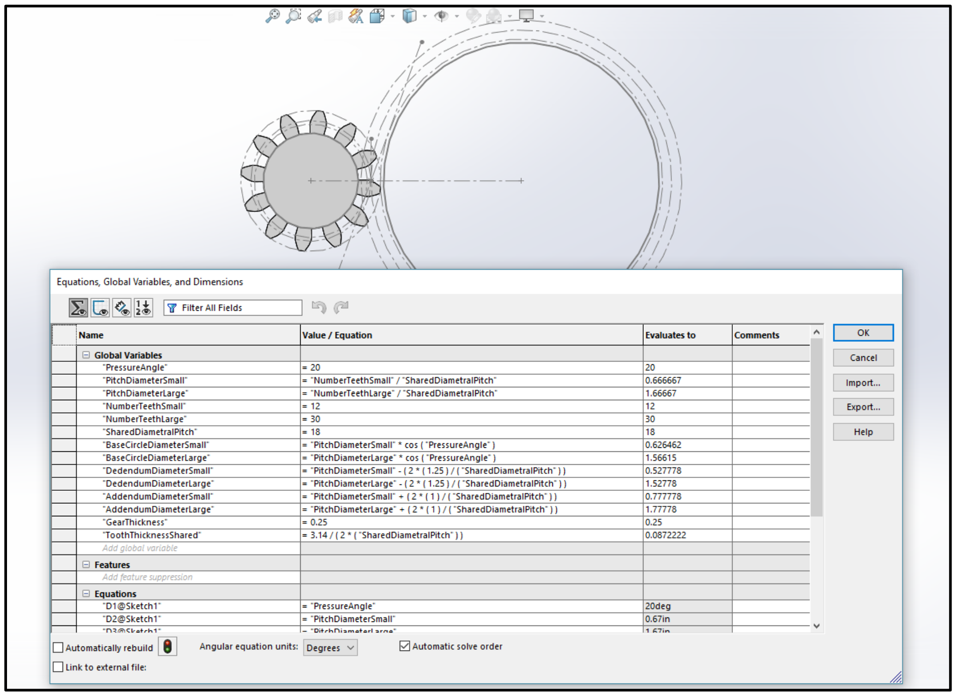


Figure 10: Lazy Susan Gearing



Equations, Global Variables, and Dimensions

Name	Value / Equation	Evaluates to	Comments
Global Variables			
"PressureAngle"	= 20	20	
"PitchDiameterSmall"	= "NumberTeethSmall" / "SharedDiametralPitch"	0.666667	
"PitchDiameterLarge"	= "NumberTeethLarge" / "SharedDiametralPitch"	1.66667	
"NumberTeethSmall"	= 12	12	
"NumberTeethLarge"	= 30	30	
"SharedDiametralPitch"	= 18	18	
"BaseCircleDiameterSmall"	= "PitchDiameterSmall" * cos ("PressureAngle")	0.626462	
"BaseCircleDiameterLarge"	= "PitchDiameterLarge" * cos ("PressureAngle")	1.56615	
"DedendumDiameterSmall"	= "PitchDiameterSmall" - (2 * (1.25) / ("SharedDiametralPitch"))	0.527778	
"DedendumDiameterLarge"	= "PitchDiameterLarge" - (2 * (1.25) / ("SharedDiametralPitch"))	1.52778	
"AddendumDiameterSmall"	= "PitchDiameterSmall" + (2 * (1) / ("SharedDiametralPitch"))	0.777778	
"AddendumDiameterLarge"	= "PitchDiameterLarge" + (2 * (1) / ("SharedDiametralPitch"))	1.77778	
"GearThickness"	= 0.25	0.25	
"ToothThicknessShared"	= 3.14 / (2 * ("SharedDiametralPitch"))	0.0872222	
Add global variable			
Features			
Add feature suppression			
Equations			
"D1@Sketch1"	= "PressureAngle"	20deg	
"D2@Sketch1"	= "PitchDiameterSmall"	0.67in	
"D3@Sketch1"	= "PitchDiameterLarge"	1.67in	

☐ Automatically rebuild Angular equation units: Degrees ☑ Automatic solve order
☐ Link to external file:

Figure 11: SolidWorks Gear Generator

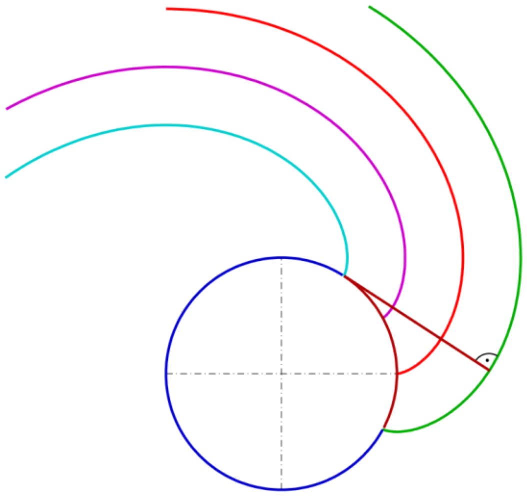


Figure 12: Example Involute Curve

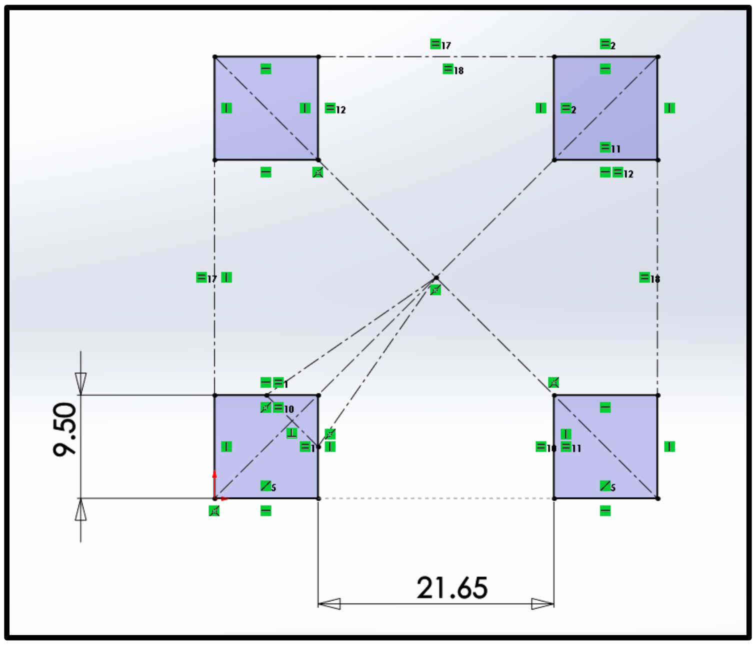


Figure 13: Aiming From a Node

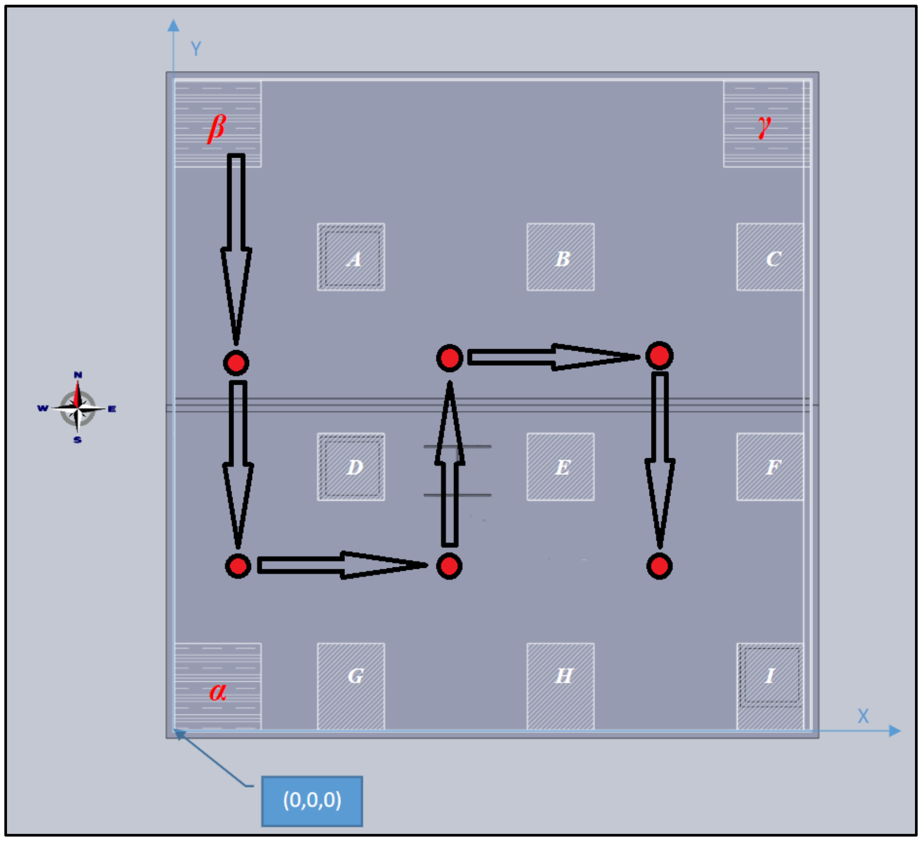
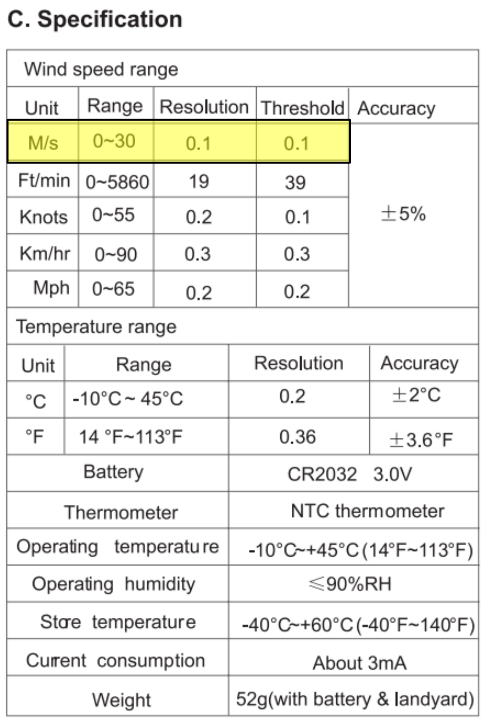


Figure 14: Strategy

Tables



C. Specification

Wind speed range				
Unit	Range	Resolution	Threshold	Accuracy
M/s	0~30	0.1	0.1	±5%
Ft/min	0~5860	19	39	
Knots	0~55	0.2	0.1	
Km/hr	0~90	0.3	0.3	
Mph	0~65	0.2	0.2	

Temperature range			
Unit	Range	Resolution	Accuracy
°C	-10°C ~ 45°C	0.2	±2°C
°F	14 °F~113°F	0.36	±3.6°F

Battery	CR2032 3.0V
Thermometer	NTC thermometer
Operating temperature	-10°C~+45°C (14°F~113°F)
Operating humidity	≤90%RH
Store temperature	-40°C~+60°C (-40°F~140°F)
Current consumption	About 3mA
Weight	52g(with battery & landyard)

Table 1: Specifications of Pyle PMA85 Anemometer