Flipkart

GRiD 2.0

Autonomous Indoor Drone - Round 3 Template

Team Name: Safety Hazard

Institute Name: BITS Pilani, Hyderabad Campus

Team Member Details

Name	Batch	Interests & Experience
Aditya Chopra	2019	Interested in Machine Learning and Computer Vision applications. Worked with Neural Networks and OpenCV on personal projects. Briefs and code available on GitHub.
Vishwas Vasuki Gautam	2019	Interested in Control Systems and Electronics. Worked on several Arduino and Raspberry Pi based projects. Currently working on Localisation and Mapping algorithms along with Control systems concepts. Projects available on GitHub.
Abhijit Pranav Pamarty	2019	Relevant interests include Computational Fluid Dynamics, Computer-aided modelling, and Structural Simulation. Experienced with Fusion 360, ANSYS, and SimScale. Other projects available on Github.
Digvijay Singh	2017	Specializations include Circuit Design, Machine Learning and Automation. Prior experience building drones for water sampling/surveillance related applications.

Introduction & General Information

We chose to work with a Hexacopter as the Autonomous Indoor Drone. The six thruster (bldc motor + propeller) configuration will require less individual thrust from each motor, giving us more latitude while choosing thrusters. The components of the Hexacopter are chosen to make it versatile, as seen in the future slides.

The end to end length of the Hexacopter is 850 mm and the height is 164 mm keeping it well within the dimensions of the gate.

The Hexacopter will weigh around 3.5-4 kilograms. Considering 2:1 weight to thrust ratio, the maximum thrust per motor comes out to be 1.4 kilograms. This is achievable as seen in CFD and according to the datasheet of the thrusters.

We estimate the max flight time to be 45 min to 1hr(based on theoretical calculation and a complete usage of the available battery capacity which is not advisable).

The future slides and work will give a detailed information regarding the Hexacopter and our proposed solution to the problem statement.

CAD File with Payload & Drone Components

The CAD model is available in native f3d archive, .step and .iges formats.

The top plate was modelled initially and was used as the reference to other components like drone arms, etc.

Most of the on-board components such as the Raspberry Pi, Pixhawk, Propeller and various other sensors were found online on GrabCAD. But the BLDC was modelled using the datasheet.

The landing gear was modified from the round 2's version, to account for more strength.

The plates needed to be extended in order to account for a larger propeller width than what is usually recommended for the F550 model.

Link to the model can be found here.

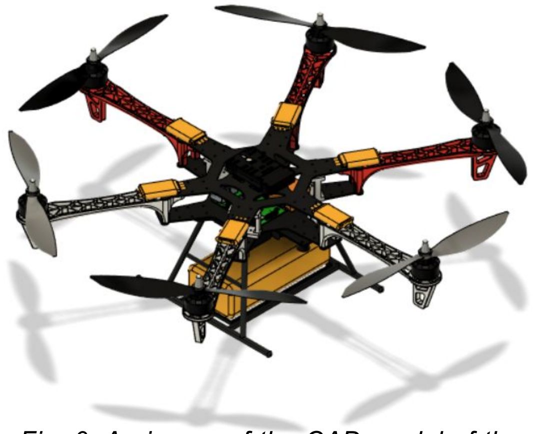


Fig. 0, An image of the CAD model of the Hexacopter.

Aerodynamic & Payload Calculations

It was decided that the aerodynamics of the drone would be split into two different parts; the first part involving the effect of a payload being carried on the horizontal and vertical pressure forces on the drone, and the second involving the observation of the propeller wake and roughly confirming the thrust developed by the propeller was in accordance with the values provided by an external thrust calculation website.

The objective of the first CFD simulation run was to compare the horizontal forces and the wakes between the drone carrying the payload and the drone which wasn't carrying the payload in a wind of 5 m/s velocity magnitude parallel to the ground, descending at the rate of 1 m/s. By measuring the forces and moments on the drone surfaces, it was proven that for this particular scenario, there was only a marginal increase in force in the direction of the wind, and thus this was deemed as acceptable.

The software used for the simulation runs was Simscale, by SimScale GMBh. The reasons for selecting SimScale were the free-to-use policy, and the ability to solve of problems on the cloud, thus saving the team both time and money. Furthermore, SimScale also allowed the team members to view the simulation results quite easily, as the solutions were available on the internet and didn't require members not working on aerodynamic and structural simulation to download and install any additional software.

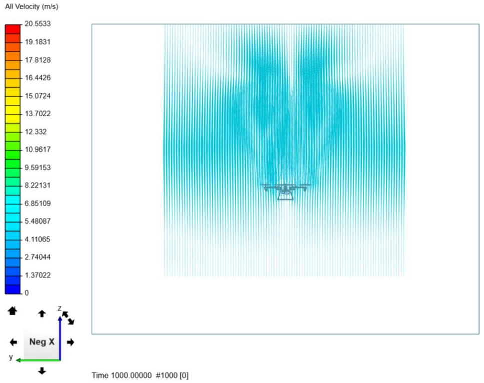


Fig. 1, the streamlines of the drone not carrying the payload viewed from the front. Note that there are no rotating zones employed in this scenario, and the reason for this will be explained in the subsequent slides.

Aerodynamic & Payload Calculations

Cleaning of the CAD model [Mid-flight resistance to moderate wind]

In contrast to standard CAD modelling, where more detail is better as the focus is on solid geometry, for computational fluid dynamics, in order to reduce small gaps which could increase the complexity of the problem and thus decrease the time for a CFD solution, while having a minimal impact on the results, a simple model is more effective.

Excess features were reduced in fusion 360's simulation environment using the simplify tool, and intersections were removed in order to reduce the chance of meaningless results. The simplified model was then exported as a step file, as SimScale did not have the ability to upload files in fusion 360's native file format. After the geometry was uploaded, the simulation was now ready to begin.

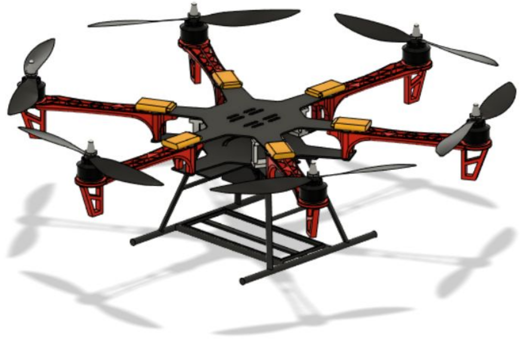


Fig.2, the cleaned CAD model of the drone. Most of the electronics have been removed, and unnecessary small features such as mounting holes, extrusions, and other small parts have been either reduced or completely removed. Removal of these features would simplify the problem, and allow for a faster solving time.

Aerodynamic & Payload Calculations

Incompressible flow [Mid-flight resistance to moderate wind]

The mach number is the ratio of the speed of the moving gas to the speed of sound in the gas at a given temperature and pressure. As the wind velocity was 5 m/s, or a mach number of 0.015, the simulation was approximated to be incompressible. This was justified from the equation shown on the right hand side, where M is the mach number, V is the velocity and ρ is the density. It is observable for our simulation requirements, the change in density would be so minimal that it could be ignored. Thus, the simulation model chosen was incompressible flow.

$$-M^2\frac{dV}{V} = \frac{d\rho}{\rho}$$

Sub.Fig.1, the equation. Note the change in density of the fluid increases proportionally with the square of the mach number.

The k-omega SST model is one of the most commonly used models for CFD simulation. There are two variables which allow for the quantification of the characteristics of the turbulent flow in the simulation model, namely k and ω. It is known that the k-omega model predicts well for near-wall regions, and the k-epsilon model predicts well for the regions which are far away from the boundary. The SST model which combined both of these characteristics gave satisfying results, and thus was chosen for the simulation.

The SIMPLE (Semi-Implicit Method for Pressure Linked Equations) algorithm developed in 1972, is an iterative algorithm for the solving of Navier-Stokes equations via a method which does not directly account for the pressure correction on the velocity correction equation. A direct consequence of this approach is the reduction of computation time. As this was a relatively simple problem, although the SIMPLE solver can lose a good amount of the velocity field for complex simulations, this was deemed as adequate for this simulation.

Aerodynamic & Payload Calculations

Initial and boundary conditions [Mid-flight resistance to moderate wind]

The initial global gauge pressure was set as o Pa. Assuming the environment would be the inside of a building, this was a fair assumption to make. The initial velocity vector was [5,0,0], as the drone was assumed to begin descent instantaneously as the simulation began, but to be at rest at the time of start of the simulation. The global turbulent kinetic energy and the rate of dissipation of turbulent kinetic energy were left at their default values in the solver, at 0.00375 m^2s^2 and 3.375 s^{-1} respectively. Turbulent kinetic energy is defined as the mean kinetic energy per unit mass, and is characterised by the measured RMS velocity fluctuations.

The velocity inlet was given a fixed-value velocity of [5,0,1] in order to simulate the drone descending in a wind tunnel. The drone's rotors were made static, as the simulation run's total time for the computations was nearing the max allowed for the free trial version. As the purpose of this simulation was to only characterise the difference in flow characteristics caused by an addition of a payload, this compromise was made. The outlet was given a face normal velocity of [5].As the normal vector out of the velocity outlet plane was pointing in the direction of [1,0,0], this translated to the velocity vector near the outlet would be moving only in the x direction.

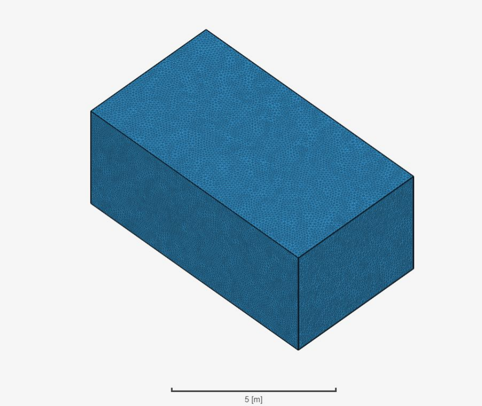


Fig. 3, an isometric view of the finished mesh. The face facing the bottom right corner of the image is the velocity inlet.

Aerodynamic & Payload Calculations

Initial and boundary conditions [Mid-flight resistance to moderate wind]

The walls were given a no-slip velocity condition. This essentially meant that near-boundary characteristics of the fluid would be that it would have zero velocity relative to the boundary. As the adhesive forces near a boundary between the fluid and the solid are far greater than the cohesive forces between the fluid and the fluid near the boundary, the fluid would not move. This, however, is not observed in real life, but was a good approximation for this problem.

The numerical controls of the simulation were left as default, except for the number of non-orthogonal correctors. The mesh generated was mostly orthogonal, and thus, a low value of 2 was chosen for both the simulations.

As the problem required that the forces on the drone surfaces because of the fluid flow be known, the only solution control applied was the one of forces and moments, during both the simulation runs. The meshing was done with the standard algorithm, and the fineness was left as default. The number of cells ended up being 8.6 M for the drone without the payload, and 8.4 M cells for the drone with the payload. The max aspect ratio for the without payload simulation was found to be 64, and the max aspect ratio for the with-payload simulation was found to be 49. As these were acceptable values, the meshing quality was deemed to be good, and the simulation began.

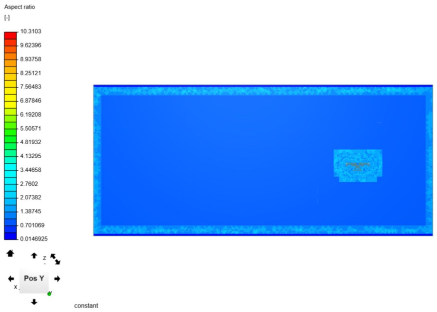


Fig.4, a side view of the mesh. The cutting plane passed through the centroid of the drone. It can be noticed here that the max aspect ratio at this position is around 10, which is an excellent value. The aspect ratio of a mesh cell is defined as the ratio between the length of its largest side to the length of its smallest side.

Aerodynamic & Payload Calculations

Results, overview [Mid-flight resistance to moderate wind]

The pressure force on the x-direction was found to be 0.733 N at time T = 1000 for the payload condition and 0.622 for the no payload condition at time T = 1000. This essentially meant a 117 % increase in force in the x direction was observed when the payload was carried by the drone.

The forces and moments graphs were analysed, and the difference between the geometries of the wake was observed.

The force in the z direction was nearly equal for both the conditions, at approximately around 0.47N. There was no considerable force in the y direction. As the drone could easily an extra 0.111N of force, the current design was determined to be efficient and further testing proceeded.

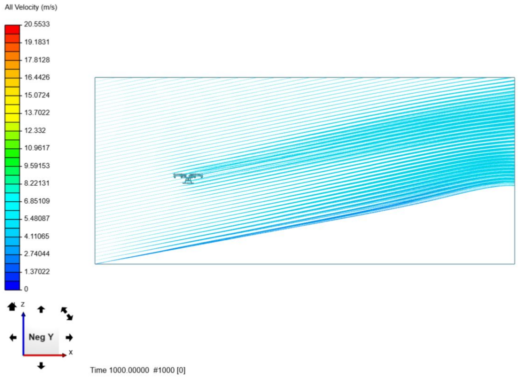
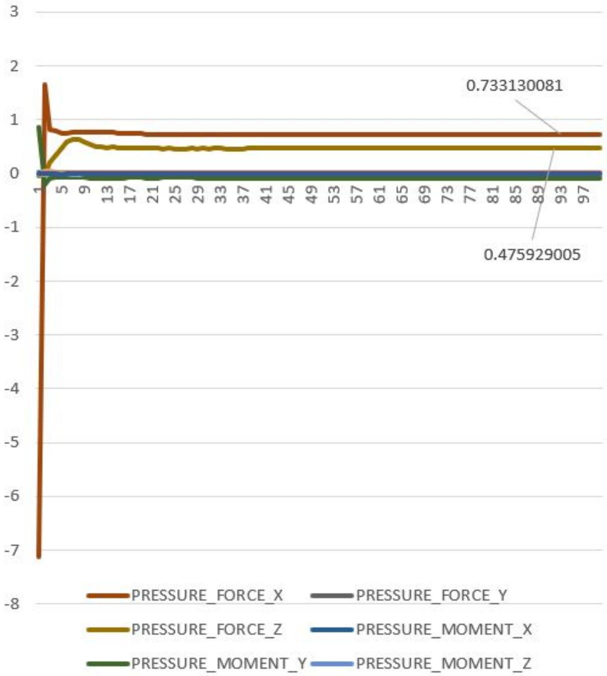


Fig. 5, A side view of the streamlines of the drone. Note that this simulation was the equivalent of putting the drone in a wind tunnel, with a stream of air blowing towards the top and to the right. This meant that there would be a large near-zero velocity region near the bottom.

Aerodynamic & Payload Calculations

Results, forces and moments graphs [Mid-flight resistance to moderate wind]



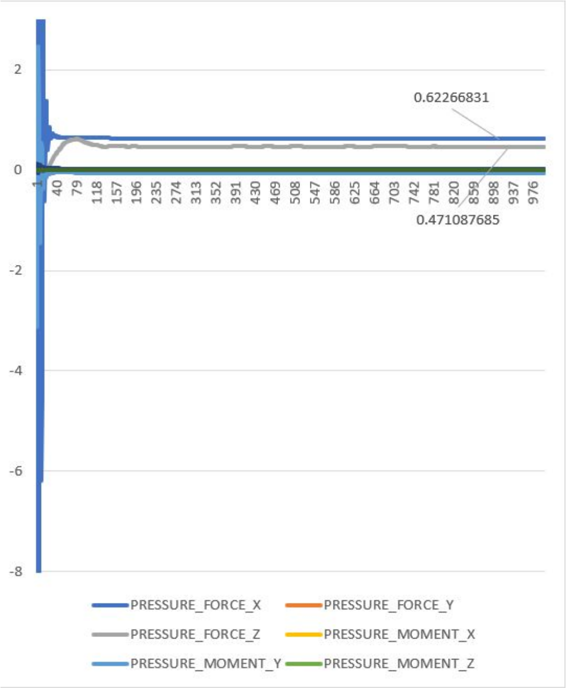


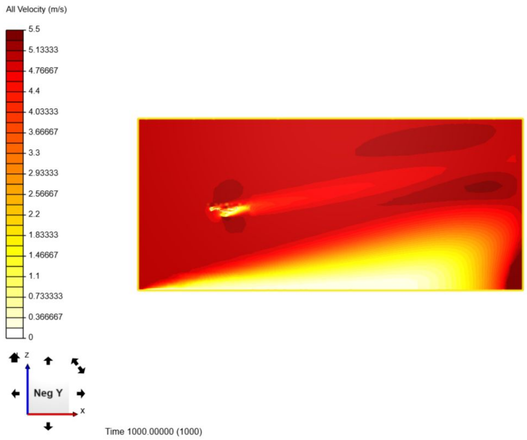
Fig 6.1 (left) and Fig. 6.2 (Right): The forces and moments graph proves to be a useful tool for measuring the effects of wind on the body. Figure 6.1 represents the graph for the drone carrying the payload, and Figure 6.2 represents the graph for the drone not carrying the payload.

It is observable from the graphs that the force in the positive X-direction is slightly larger in graph 6.1 than in graph 6.2. This is to be expected, as the payload bay is not covered, and the addition of a payload results in a greater forward facing surface area, which results in a marginal increase in the pressure force. For this wind speed, the pressure force is not too large that it can severely knock the drone off course.

The pressure force in the z-direction is nearly equal for both the scenarios.

Aerodynamic & Payload Calculations

Results, wake analysis at centre of drone [Mid-flight resistance to moderate wind]



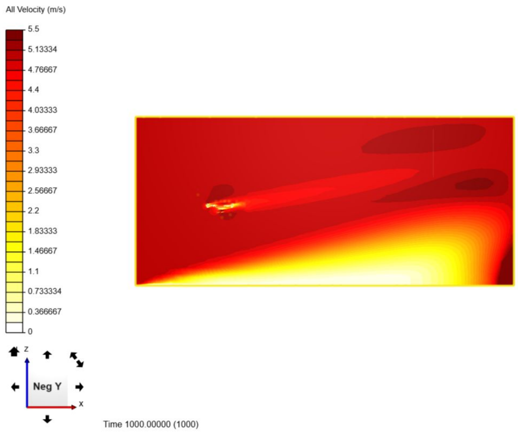


Fig 7.1 (left) and Fig 7.2 (right): Figure 7.1 represents the velocity gradient of the drone carrying the payload viewing a cutting plane through the midpoint of the mesh, and 7.2 the same for the drone not carrying the payload.

We can observe that the wake of the drone carrying the payload is slightly larger than the tail of the drone which is not carrying the payload. Furthermore, there is a larger blob of low-velocity near the landing gear of the drone when it is carrying the payload. Both of these results are what one should expect when adding a new surface which impedes the flow of air.

The large area at the bottom where there is a significant reduction in the velocity of the fluid is a result of the simulation parameters and not the geometry of the drone itself. Refer to fig. 5 for more details.

Aerodynamic & Payload Calculations

Wake analysis of propeller

In order to quantify the thrust generated by the propeller and to observe the wake, a quick CFD simulation run was done with MRF rotating zones. The purpose of this simulation run was to ensure that the external experimental results came close to the results of the CFD simulation runs.

As both the motors and the propellers are off-the-shelf and have been tested extensively by other enthusiasts and engineers, there would not be a need to optimise any of the parameters which would go into designing the propeller.

The reason for using MRF rotating zones will be explained in the following slides, along with the assumed initial conditions and the results. The target value of thrust for each propeller was 1.2 kg force of thrust, which the propellers could achieve at 733 rad/s angular velocity according to the experimental results from the source. The CFD results were found to closely match the same.

Only one clockwise rotating propeller was simulated at a time, both for efficiency and time-saving reasons.

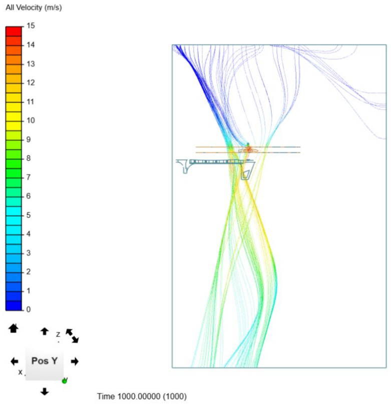


Fig. 8, the streamlines of the propeller. Notice that the cap of the velocity has been set to 15 m/s in the legend, in order to show more variation.

Aerodynamic & Payload Calculations

MRF rotating zones [Wake analysis of propeller]

For simulating the fluid flow around a rotating body, like an impeller, a turbine, or in this case a propeller, MRF rotating zones are used. The mesh is not physically rotated, and the rotating zone simply gives a steady state approximation of the transient rotating motion at a point in time.

The CAD model requires special preparation if rotating zones are to be used, that is, a cylindrical boundary must be drawn around the rotating volume of the mesh. Unlike the previous simulation, the fluid volume was generated in fusion 360's solid modelling environment.

The rotating zones follow the standard right hand rule, and the vector needed to be into the plane of rotation (or towards the bottom of the mesh) to simulate clockwise rotational motion.

Performing MRF simulations is not as computationally demanding as simulating transient motion, and since MRF provides a good approximation of the rotational flow, this was chosen for the simulation.

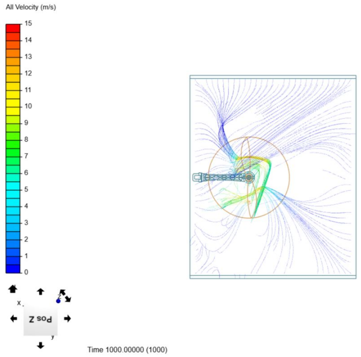


Fig. 9, the streamlines of the propeller, viewed from above. The cap of the velocity has again been set to 15 m/s in the legend, in order to show more variation. The rotating zone is projected as a circle in the middle of the mesh.

Aerodynamic & Payload Calculations

Initial and boundary conditions [Wake analysis of propeller]

The initial global gauge pressure was again set as o Pa. The initial velocity vector was [0,0,0], as the drone was assumed to be at rest (or in a perfectly stationary hovering state) at the start of the simulation. The global turbulent kinetic energy and the rate of dissipation of turbulent kinetic energy were left at their default values in the solver, at 0.00375 m^2s^2 and 3.375 s^{-1} respectively.

The pressure inlet-outlet velocity surfaces were given a total gauge pressure of 0 Pa, and the Turb. Kinetic energy type and the specific dissipation rate type were left at their default, or zero gradient. The drone walls were assigned a no-slip velocity condition.

The number of orthogonal correctors was increased to 4, from the default of 2.This was to ensure that the solver would reach an accurate solution for the mesh created by the standard mesher algorithm of SimScale, and was recommended by the company.

The simulation control parameters were left as default.

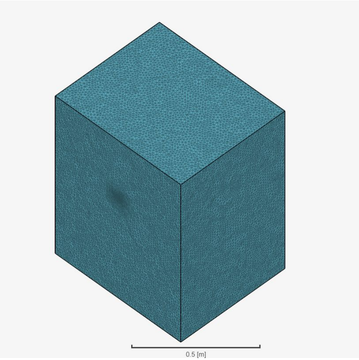


Fig. 10, An isometric view of the finished mesh. All the 6 faces have been assigned the “Pressure inlet-outlet velocity condition, and the drone faces have been assigned a no-slip wall condition.

Aerodynamic & Payload Calculations

Initial and boundary conditions [Wake analysis of propeller]

The rotating zone was assigned as a separate cell zone which was part of the mesh, rather than being a separate mesh by itself. The mesh was generated at the time of simulation and not separately, as at this point in time one could be reasonably confident that the meshing quality would be good.

This was confirmed after the simulation, when the max aspect ratio of the mesh was determined to be 476, which was deemed as acceptable for a rough simulation of this kind. However, this had a small chance of resulting in a large error in the results. This however, was not the case, and the simulation proceeded smoothly and gave reasonable results. The total number of mesh cells ended up being approximately 3.3 million.

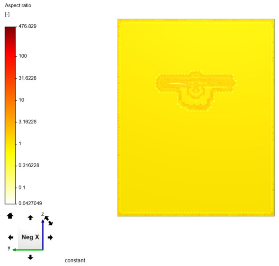


Fig. 11, A side view of the mesh displaying the aspect ratio. The aspect ratio around the propeller is roughly around 5, and this is a reasonably good value.

Aerodynamic & Payload Calculations

Results, overview [Wake analysis of propeller]

In order to calculate the thrust generated, the propellers were assumed to be a solid disk, across which the pressure would suddenly rise. The area of the disk multiplied by the pressure would be equal to the thrust generated. This could further be simplified into a set of equations with the velocity, the density of air, and the area as parameters.

The inlet velocity was assumed to be 0 m/s. And the outlet average velocity was assumed to be 16.715m/s, by averaging all the pixel values of an area in the cutting plane below the propeller.. The density of air was taken to be 1.225 kg/m^3 . The area was the area of the circle having 282 mm as its diameter, or 2.82 * 10^{-1} m. Thus, the area was equal to 0.0625 m^2 . Substituting these values in the formula, the thrust was approximated to be around 10.695 N, or 1.091 kg force, while the website reported 1.288 kg force, although this value was a projected value rather than an experimentally verified one. This translated to an error of about 0.1529. For the prototype, this error was deemed as acceptable.

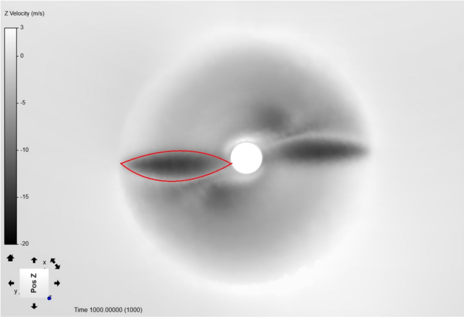
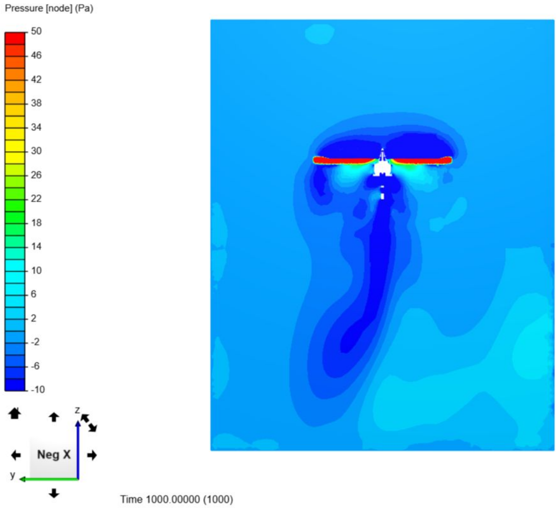


Fig. 12, A top view of the Z-velocity of the fluid flowing directly beneath the plane of the propeller. Note that, as MRF rotating zones were used instead of a transient analysis, the average velocity of all the points in the cylinder model could be assumed to be equal to the average velocity of the points directly below the propeller.

Aerodynamic & Payload Calculations

Results, pressure gradient analysis at centre of prop [Wake analysis of propeller]



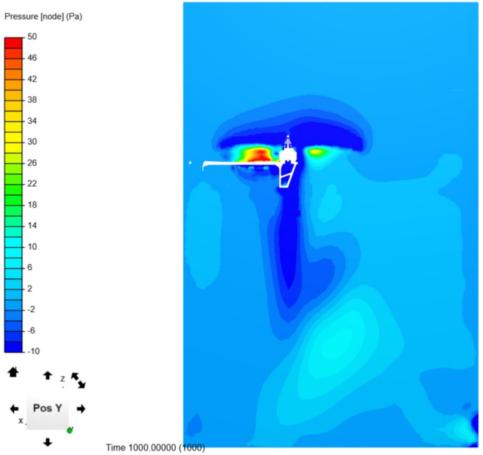
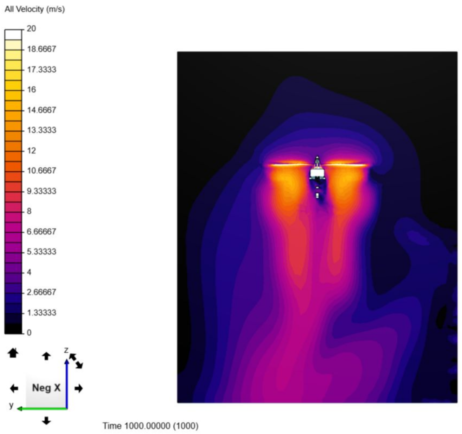


Fig. 13.1 (left) and Fig 13.2 (right), figure 13.1 is the pressure gradient viewed from the negative X direction, and figure 13.2 is the pressure gradient viewed from the positive Y direction. Note the high pressure zones near the arm in 13.2, and the high pressure zones on the prop in figure 13.1. The areas likelier to have higher velocity have lesser pressure, which is to be expected.

Aerodynamic & Payload Calculations

Results, velocity gradient analysis at centre of prop [Wake analysis of propeller]



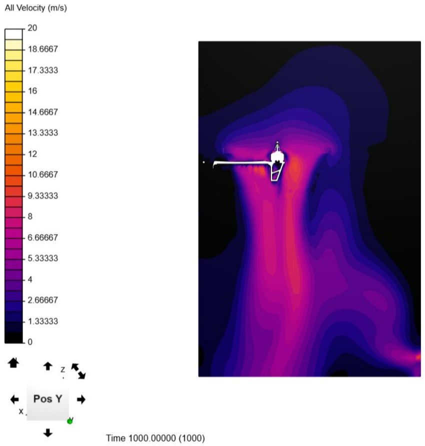


Fig. 14.1 (left) and Fig 14.2 (right), figure 13.1 is the velocity gradient viewed from the negative X direction, and figure 13.2 is the velocity gradient viewed from the positive Y direction. The velocity is greatest near the propeller, and eventually slows down and becomes 0 towards the bottom.

Aerodynamic & Payload Calculations

Payload Calculations

The max recommended dimensions for the payload are 80 X 180 X 50 mm, or a volume of 0.00072 m^3. The max volume means that for a payload weighing 2kgs, there is a minimum payload density in order to be able to carry the entire weight. This density is about 2777 kg/m^3, or slightly more than the density of aluminium.

The payload is secured via straps to the landing gear, allowing for a wide variety of shapes to be carried by the drone. In order to keep the center of mass constant, smaller payloads will need to be secured inside a larger box and strapped to the drone.

As it is expected that there will be a wide variety of shapes which the payload can assume if the drone is to be employed in a warehouse, it was determined that having a payload bay with a flexible volume was of utmost importance.

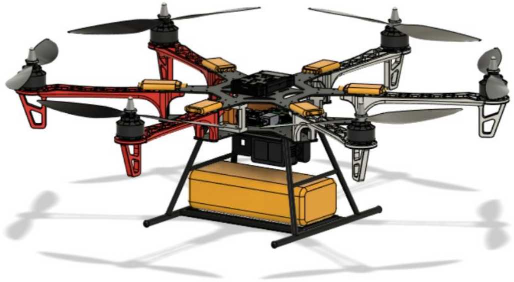


Fig. 15, the drone carrying a standard rectangular payload very close to the max acceptable volume. The straps are not shown in the image, but they would go over the payload and secure it to the bottom.

Structural & Stability Analysis (with & without payload)

Design overview

In order for maximum design efficiency to be achieved, a few of the drone's parts were selected to be off-the-shelf components, or COTS. This allowed for resources and time be used for other purposes, and also reduced the time needed for the optimisation of the parts. As the F550's frame could not support 11 inch diameter propellers without the risk of two adjacent propellers colliding with each other, the frame of the drone was extended radially from the round 2 design.

This meant that the frame would need to be manufactured out of aluminium 6061-T6, from an external sheet metal vendor. It was found out that manufacturing of the frame wouldn't raise costs as to go over budget.

Vertical I-shaped supports were also designed to make sure that the addition in material wouldn't cause structural deformation in the frame because of the internal forces generated mid-flight.

The landing gear was designed in-house, and thus would require slightly more rigorous testing than the arms. A drop-test was intended to be simulated for round 3, however, because of time constraints, it was pushed further into the future.

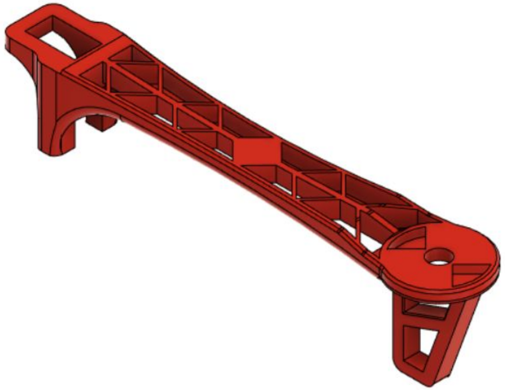


Fig.16, The COTS F550 drone arm model used for simulation. Note that this model has been simplified by removing the mounting holes and reducing other detailed geometry.

Structural & Stability Analysis (with & without payload)

Model parameters, Materials and Boundary Conditions (Drone Arm)

The global gravity was left at 0 m/s, as the only factor to be analysed was the effect of the motor exerting a force upwards. The material assigned to the drone arm was polyamide, or standard nylon polymer, as this is what all of the arms manufactured were made out of.

The young's modulus of the PA was 8e+8, the poisson's ratio 0.39, and the density 1030 kg/m^3, which was all standard for the material. The material behaviour was set as linear elastic, which was the default.

The faces attached to the plates were given the fixed support boundary condition. This boundary condition meant that those faces would resist both rotational and translational movement.

The face on which the motor was mounted was given a force boundary condition, directed in the positive z direction, and with a magnitude of 12.28 N. This value was selected as it was the greater of the two results, derived from the CFD analysis and the website.

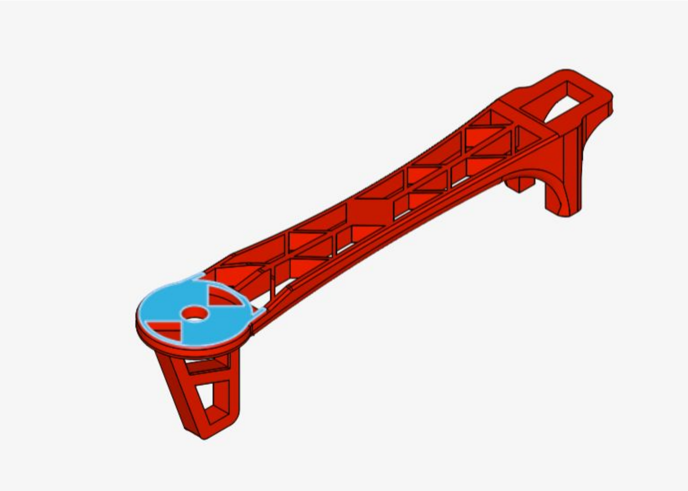


Fig. 17, the drone arm with the face on which the force is applied highlighted in blue.

Structural & Stability Analysis (with & without payload)

Mesh and simulation parameters (Drone Arm)

The meshing algorithm, the sizing of the cells, the fineness were all left as default. Second order elements was turned off After meshing, the final mesh cell count was around 190 thousand cells.

The meshing quality was determined with the help of an aspect ratio isovolume. As expected for a simplified body, the maximum aspect ratio was found to be around 26. This was deemed as excellent, and thus the simulation progressed.

The intention of the simulation run was to measure von-mises stress and average displacement of points on the drone arm during the max-thrust condition. There were no contacts required for this simulation.

As it was very unlikely that the drone arm would face a situation in which one end was fixed, it was a reasonable assumption to make that if the arm could survive this test, it could survive anything encountered in the real world.

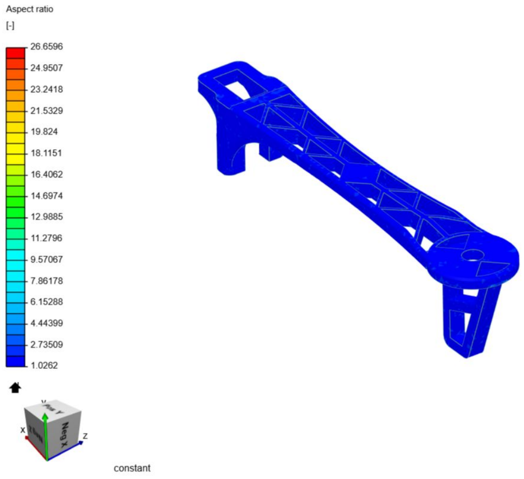
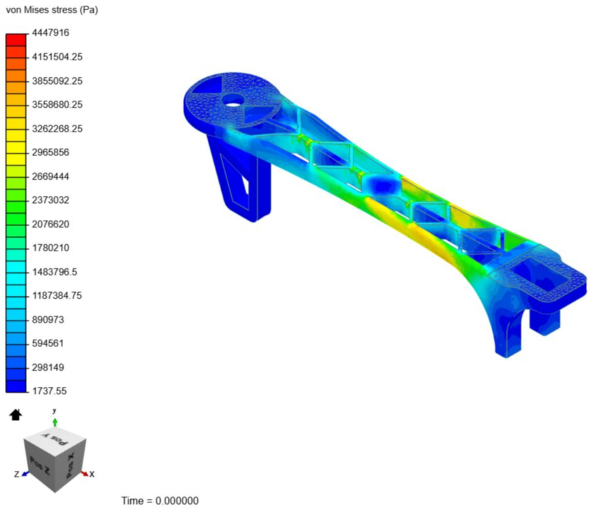


Fig. 18, the aspect ratio isovolume of the drone arm. Note that most cells have an aspect ratio very close to 1.

Structural & Stability Analysis (with & without payload)

Discussion of results (Drone arm)



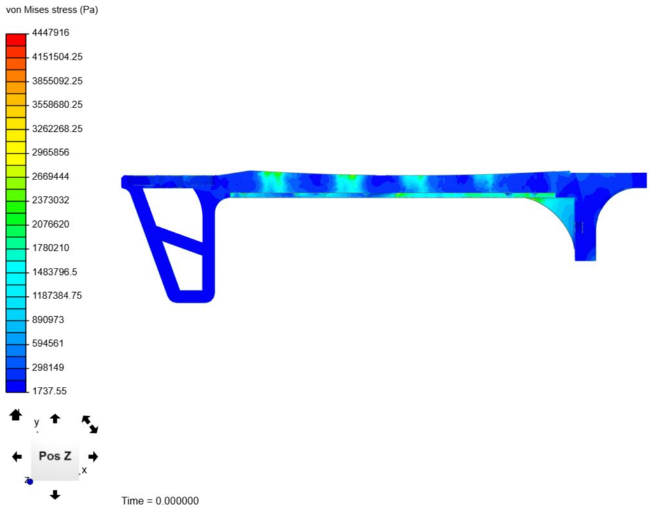


Fig 19.1 (left) and Fig 19.2 (right), figure 19.1 illustrates the von mises stress isovolume from above, and figure 19.2 illustrates the von mises stress inside the arm using a cutting plane with a normal perpendicular to the length of the arm. The von mises stress serves as a useful parameter to analyse the tensile stress of a material and can be used to predict the yielding of a material.

The stress observed in both the images is lesser than 40 MPa at most locations, which is the lower limit for the tensile yield strength of Nylon.

This is expected as the part is an off-the-shelf part.

Structural & Stability Analysis (with & without payload)

Discussion of results (Drone arm)

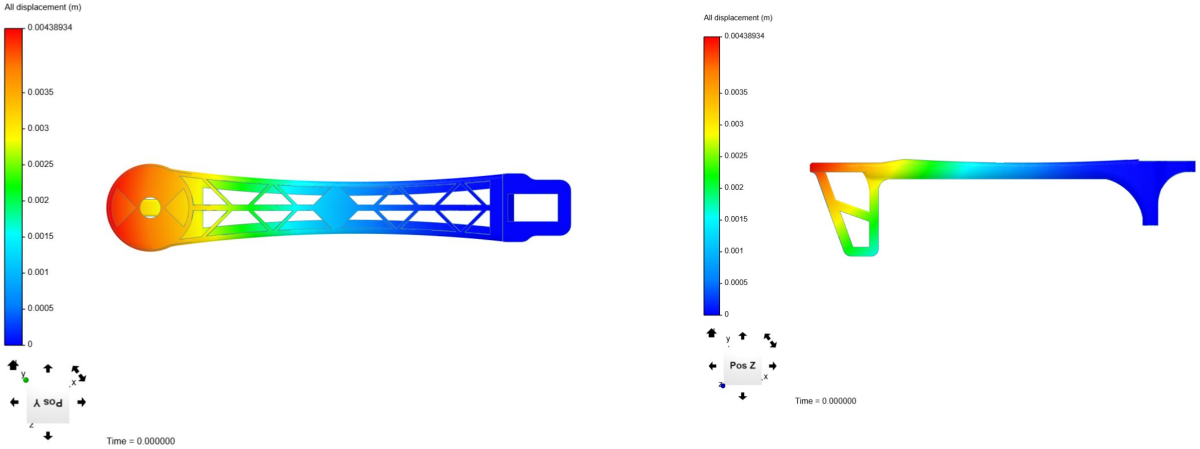


Fig 20.1 and 20.2, these images shows the displacement in all directions of each point on the arm. As expected, the largest displacement occurs near the end of the arm. The displacement however is very small.

Structural & Stability Analysis (with & without payload)

Model parameters, Materials and Boundary conditions (Landing gear with and without payload)

The global gravity was once again left at 0 m/s. The material assigned to the landing gear was PLA, or poly-lactic acid for ease of manufacturing via 3-D printing. The fill-density of the 3D printed material was assumed as 100%.

The young's modulus of the PLA was 3.5e+9, the poisson's ratio 0.36, and the density 1250 kg/m^3, which was all standard for the material. The material behaviour was set as linear elastic, which was the default.

The two rods at the bottom were given the fixed support boundary condition. This meant that the drone was being held by its landing gears, and the only forces acting were the weight of the drone itself, and the weight of the payload.

The face on which the plates were mounted was given a force of magnitude 19.6 N, towards the ground, to simulate the weight of the drone when it was placed on the ground. In the no payload condition, no other forces were present on the landing gear other than this. However, in the with payload condition, an additional 19.6 N of force was exerted on the payload frame.

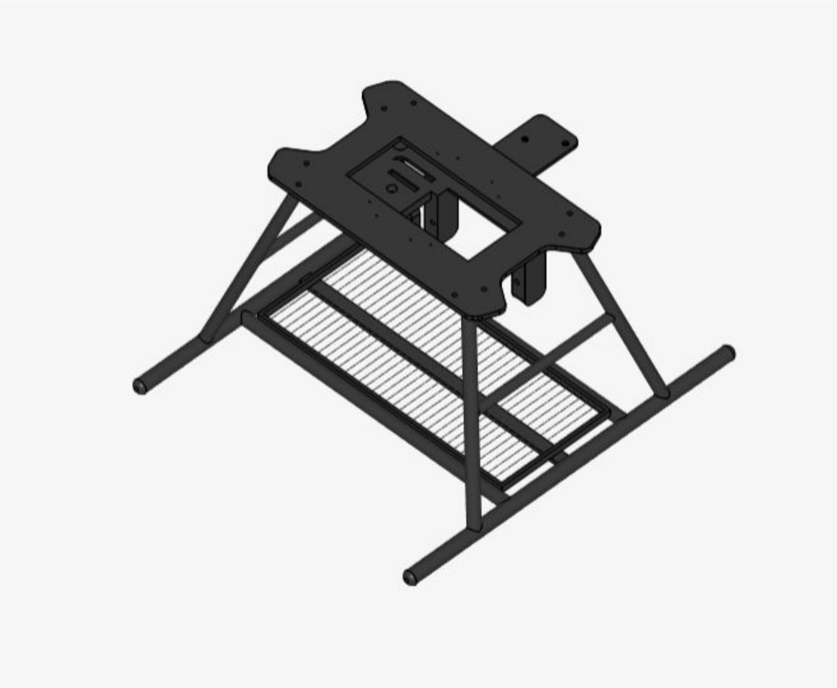


Fig. 21, the landing gear.

Structural & Stability Analysis (with & without payload)

Mesh and simulation parameters (Landing gear with and without payload)

The meshing algorithm, the sizing of the cells, the fineness were all left as default, once again. Second order elements was turned off After meshing, the final mesh cell count was around 181 thousand cells.

The meshing quality was once again determined with the help of an aspect ratio isovolume. As expected, the maximum aspect ratio was found to be around 46. This was deemed as fair, and thus the simulation could now be progressed.

The intention of the simulation run was to measure von-mises stress and average displacement of the landing gear in both the scenarios, and pull a comparison between them. The stress study was done at 0 external acceleration.

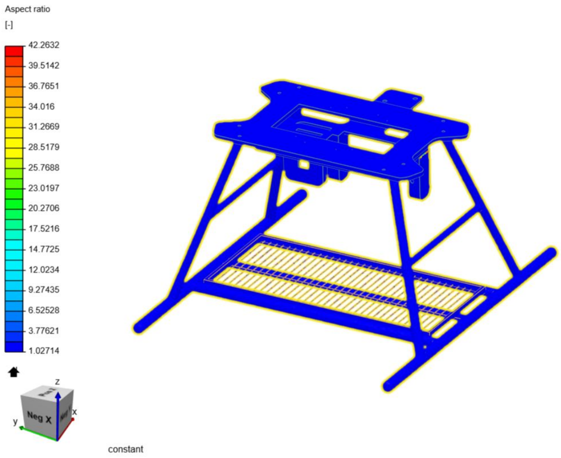
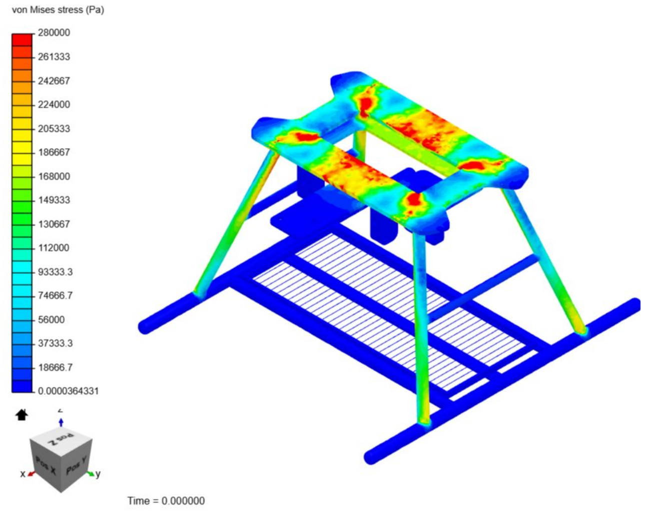


Fig. 22, the aspect ratio isovolume of the mesh.

Structural & Stability Analysis (with & without payload)

Discussion of results (Landing gear without payload)



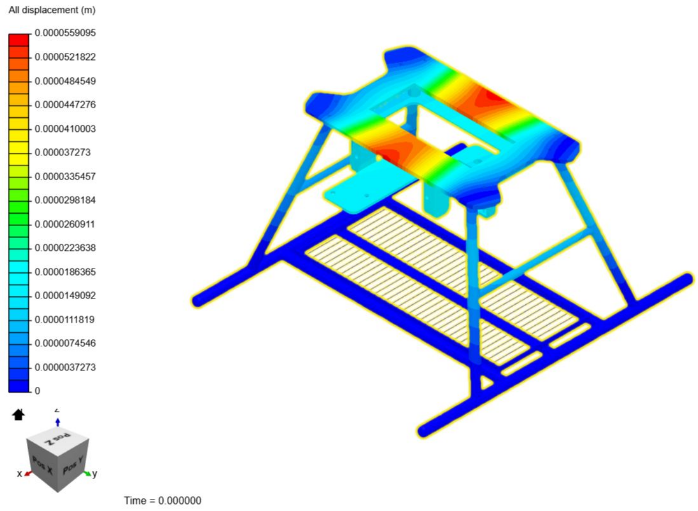
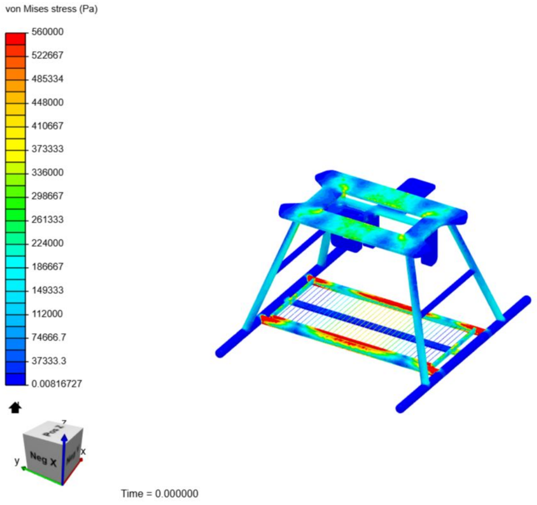


Fig. 23.1 and 23.2, show the von mises stress and displacement respectively. The scaling for the stress results is that the reddest region is 1/5th the yield strength. The displacement is again very negligible.The results indicate that the landing frame is reasonably strong without the payload.

Structural & Stability Analysis (with & without payload)

Discussion of results (Landing gear with payload)



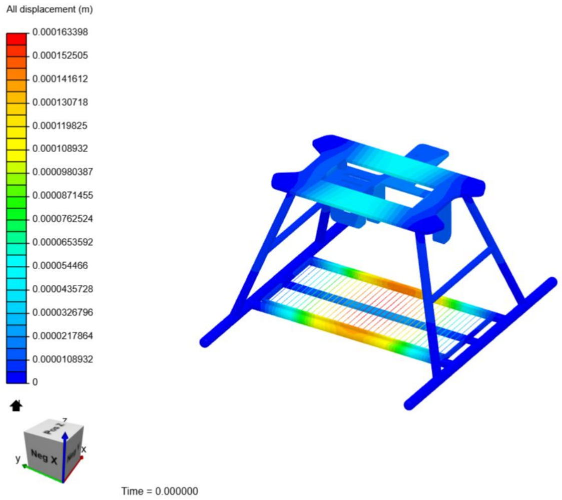
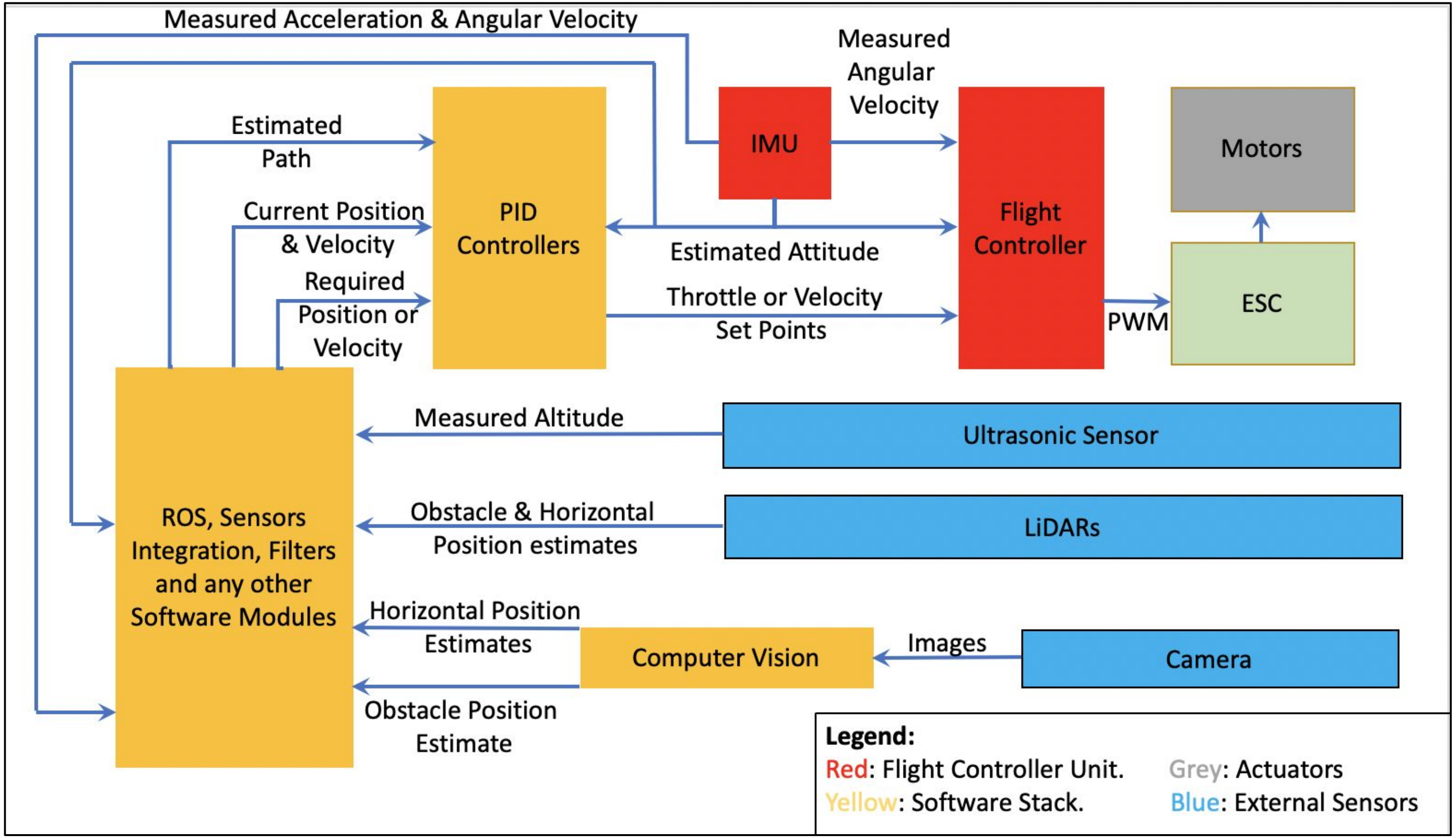


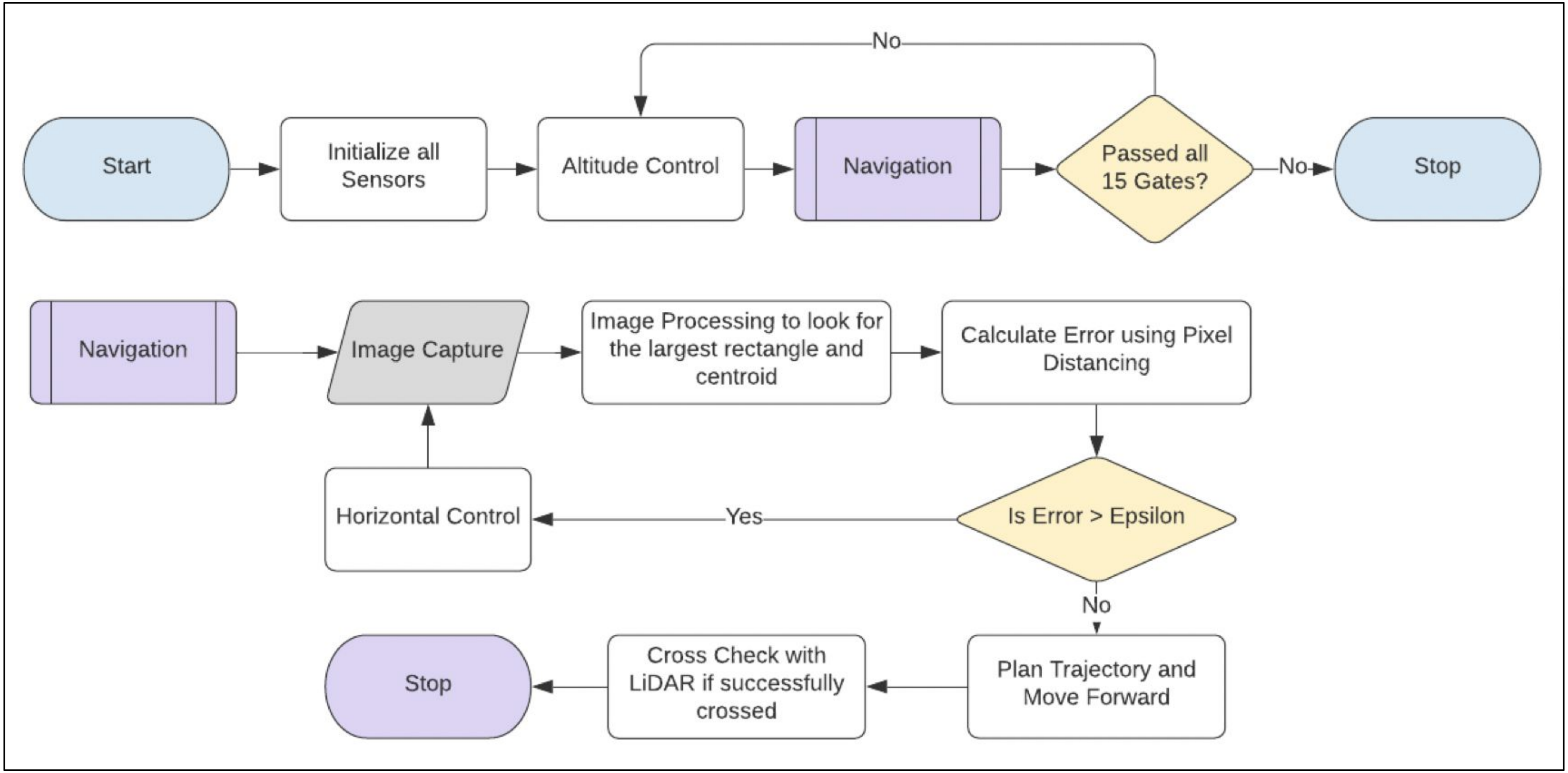
Fig. 24.1 and 24.2, show the von mises stress and displacement respectively with the load attached. The scaling for the stress results is that the reddest region is 1/ 2.5-th the yield strength. The displacement is once again very negligible.The results indicate that the landing frame is reasonably strong with the payload as well. The displacement scaling parameters have also been changed.

Architecture Overview


Measured Acceleration & Angular Velocity
Measured Angular Velocity
IMU
Flight Controller
Motors
ESC
PWM
Estimated Path
PID Controllers
Current Position & Velocity
Required Position or Velocity
Estimated Attitude
Throttle or Velocity Set Points
ROS, Sensors Integration, Filters and any other Software Modules
Measured Altitude
Ultrasonic Sensor
Obstacle & Horizontal Position estimates
LiDARs
Horizontal Position Estimates
Computer Vision
Images
Camera
Obstacle Position Estimate

Legend:
Red: Flight Controller Unit. Grey: Actuators
Yellow: Software Stack. Blue: External Sensors

Control Overview



Start
Initialize all Sensors
Altitude Control
Navigation
Passed all 15 Gates?
No
No
Stop
Navigation
Image Capture
Image Processing to look for the largest rectangle and centroid
Calculate Error using Pixel Distancing
Is Error > Epsilon
Yes
Horizontal Control
No
Plan Trajectory and Move Forward
Cross Check with LiDAR if successfully crossed
Stop

This is just an overview, a detailed description of the Control will be provided in later slides.

ROS + Gazebo Introduction

Robot Operating System along with the Gazebo 3D simulator was used to make the software framework for simple integration with hardware along with the simulation of the entire model.
ROS is a distributed structure that is robust and flexible, which allows easy access for upgrades and replacements without having to alter the entire architecture.

A ROS Package was made for the Hexacopter simulation needed in Round 3. The same package will upgraded as we progress. It contains the temporary simulation model of the Hexacopter along with the scripts that allow the control of the Hexacopter.

ROS plugins and messages were widely used to make the package robust for upgrades. To name a few, the *LiftDrag* plugin was used to define the properties of the propellers like attack angle, surface area, etc, the *Camera* plugin was used to get a live feed of the simulation environment. Messages such as *geomatry_msgs* was used for representing the common geometric primitives like velocities, positions, accelerations, etc. More information regarding these will be provided in the future slides.

The Hexacopter model was defined as an *Universal Robot Description Format* (urdf) package, which is a XML format for representing the robot model.

To integrate and simulate the model in Gazebo, *.gazebo* & *.launch* files were specified with the required external environments and the robot controllers to simulate the motion.

Our ROS package was inspired by this project.

Robot Model & Environment in ROS & Gazebo

As mentioned earlier, the robot model needs to be a urdf package in XML format.

A simplistic Hexacopter was designed in Fusion360 software for temporary testing and learning purposes. The fusion2urdf exporter was used to convert the design to the urdf format. Several modifications were made to these files as preparation for simulation.

The urdf package consisted of *.xacro* XML files that described every component(link) and its relationship with other components(joints).

The description of the components includes the material, colour, visual, inertial & collision properties. A joint is a relationship between two links, this relationship can be fixed or movable with some constraints. The joints between each of the arms, plates, camera, motors, Arduino board(just for representation purpose) were fixed immovable joints. The relationship between propellers and the motors was described as continuous rotation joints. Transmission elements was added to each of these rotation joints to allow motion.

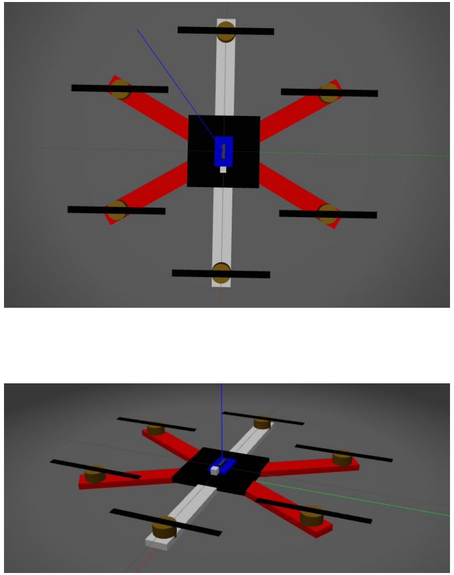


Fig 25: The robot model that was imported, as seen in Gazebo.

Robot Model & Environment in ROS & Gazebo (Contd.)

The urdf package consists of the *.gazebo* file as well. All the links defined previously referenced here. In addition, it contains the ROS plugins such as LiftDrag Plugin and the Camera Plugin.

The LiftDrag plugin defines the properties of the propeller such as surface area, attack angle, center of pressure, drag coefficients, etc. Some assumptions were made at this point (clearly mentioned in future slides) which may not be true to the real model. But since these are just values, they can be upgraded to the real model very easily.
The Camera plugin specified the camera properties like distortions, refresh rate, etc.

This ends the Robot Model Description. Now, we shall move on to the environment description.

The environment is described as a *.world* XML file. The building editor in Gazebo was used to make the frame, gates and the walls. Several such worlds were created for testing and demonstration purposes. Dimensions as specified in the problem statement was strictly followed.

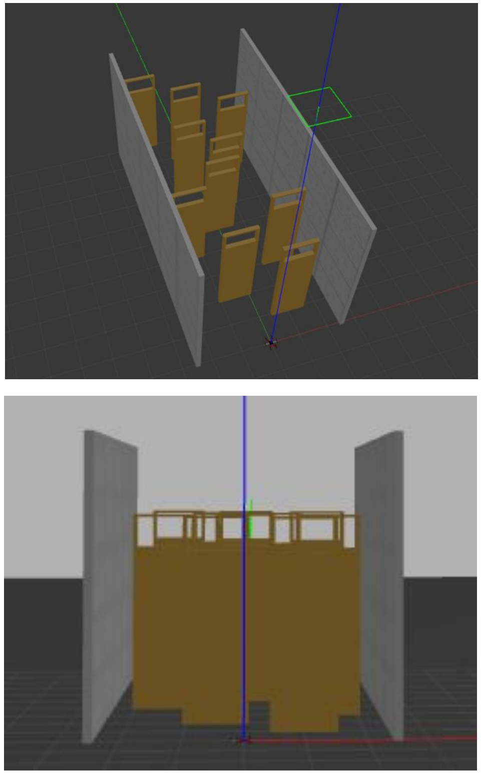


Fig 26: The world created for simulation purposes.

Robot Model & Environment in ROS & Gazebo (Contd.)

The commands to the motors were sent using ROS Controllers.

These Controllers are set of packages which are used to command the robot to work in a specific way, either by specifying its position, velocity or by applying some force or torque. This control over the robot can be achieved by publishing messages with the required parameters and arguments to the Controller. These act like commands to the Controller and the robot behaves as specified.

We widely used the *Joint Group Velocity Controller* to send 6 commands to each of the motors to rotate at a specific velocity. The velocities of each motor was calculated using the motor mixing algorithm, and messages were published to the Controller.

Other Controllers will be used as we progress when more or different form of control of the Hexacopter is required.

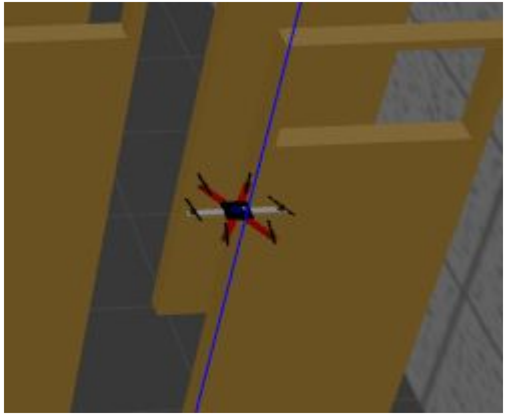


Fig 27: The Hexacopter mid flight.

Assumptions In Simulation

Hexacopter Simulation Assumptions:

Mass of Hexacopter	1.5kg
Length of Frame	0.45m
Width of each Arm	0.0315m
Model Height	0.01m
Mass of Propeller	0.0055m
Length of Propeller	0.16m

Assumptions in Environment:

- The path was well lit
- The stands/base of the gates were made solid
- 10 gates were made for presentations
- 2 gates were used to show Hexacopter Simulations

Propeller Property Assumptions:

Attack Angle (a0)	0.1
Air Density	1.2041
Area	0.762
Center of Pressure (cp)	0 0.5 0
Attack Angle at stall (alpha_stall)	0.2
Coefficient of Lift/Alpha slope (cla)	0.1
Coefficient of Drag/Alpha slope (cda)	0.01
Coefficient of Lift/Alpha slope after stall (cla_stall)	0.02
Coefficient of Drag/Alpha slope after stall (cda_stall)	1.0

Hexacopter & Multirotor Basics Used

A. Multi Rotor Basics:

Since the Hexacopter has 6 degrees of freedom, effective control over the motors is essential to keep it stable. Simple newtonian physics is employed to move the hexacopter to the required position.

When the propellers rotate with same angular velocity according to their configuration, the direction of the thrust vector is aligned with gravity and the Hexacopter can move up or down. But when the angular velocity of the propellers aren’t equal, a torque is produced causing the Hexacopter to rotate. Now the thrust vector is no longer aligned with the gravity vector, resulting in non vertical acceleration thereby moving the drone.

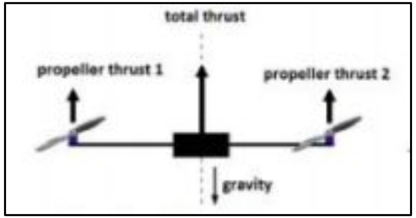


Fig. 28A.a: The thrust is aligned with gravity, causing vertical motion.

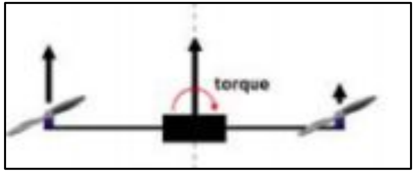


Fig. 28A.b: Different angular velocities cause torque.

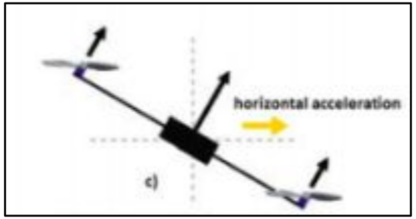


Fig. 28A.c: Horizontal acceleration due to non alignment of thrust & gravity.

B. Motor Mixing Algorithm & Configuration:

In the Hexacopter in X configuration, there are 2 motors per side on the pitch axis, but there are 3 motors per side on the roll axis. This results in a faster roll than pitch. To counterbalance this, the roll output is reduced to 50% at A, B, D & E.

In total, the pitch, roll & yaw axis are all balanced keeping the resultant zero.

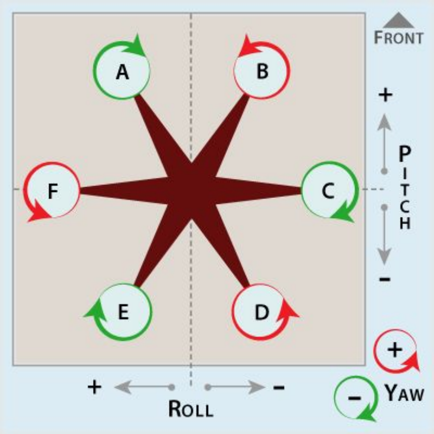


Fig. 28B.a: Hexacopter Rotor info in X configuration.

Control Theory Basics Used

The Hexacopter position can be controlled by sending thrust and attitude (roll, pitch, yaw) setpoints to the autopilot. These setpoints can be obtained using a cascade of *Proportional Integral Derivative* (PID) controllers that run on the Raspberry-Pi board.

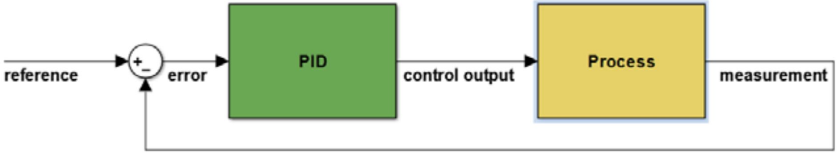
The basic theory behind a PID controller is to minimize the current error, where the error is the difference between the actual value and the setpoint value. A closed loop is formed as the output of this minimization is used as the next control signal as well. This controller will be used frequently during the flight of the drone. We may eliminate any of the terms I or D when deemed unnecessary by making the gains 0.

Proportional term(P): This term is involved in the direct scaling of the error. The value of proportional gain **K$_p$**, is modified to make the system more responsive at the cost of lower stability and overshoots. The expression for **P** is:

$$P = K_p \cdot e(t)$$

Integral term(I): This term is involved in the accumulation of differences in the previous errors. The value in integral gain **K$_i$**, is modified to eliminate the steady-state error at the cost of saturation (integral wind-up) and overshooting. The expression for **I** is**:**

$$I = K_i \cdot \int e(t)dt$$



Derivative term(D): This term is proportional to the derivative of the error. The value in derivative gain **K$_d$**, is modified to predict the future and eliminate any instability and overshoots at cost of additional noise. The expression for **D** is:

$$D = K_d \cdot \frac{d}{dt}e(t)$$

$$\implies u(t) = K_p \cdot e(t) + K_i \cdot \int e(t)dt + K_d \cdot \frac{d}{dt}e(t)$$

Autonomous Flight Algorithm Details

I. Take Off & Landing Logic (Altitude Control)

The purpose of this controller is make sure the Hexacopter reaches a desired height using altitude measurements from the onboard ultrasonic sensor and the velocity and acceleration measurements from the IMU.

Initially, a PID controller is used to output the desired velocity in the z-direction using the altitude measurements. This can be directly fed into the FCU to obtain the throttle values. But in our simulation we used an additional PID controller to obtain the acceleration and throttle values.

The PID controller ensures the smooth change in the throttle values to eliminate error and any overshoots.

Take Off	**Landing**
The desired altitude in our simulation was set to 3.25m. While taking off, the error is large, so the throttle values will reach saturation and Hexacopter will rise. As the drone closes in on the desired altitude the throttle values decrease and will eventually reach a hover value.	A very similar operation to Take Off, here the desired altitude will be 0m, the error will be largely negative thereby decreasing the throttle values and the drone will steadily lose height.
Video links: [1] [2] [3]	Video Links: [1] [2] [3]

Note: Drive link to all videos and additional ones is provided in the last slide.

Autonomous Flight Algorithm Details

I. Take Off & Landing Logic (Altitude Control) Contd...

Pseudo Code (implemented using python in ROS package):

```
1.  Input Desired Altitude    // x, y setpoints will remain 0
2.  Obtain Current Altitude using ROS Messages
3.  e = desAltitude - currentAltitude
4.  Define & Tune the PID gains to obtain velocity
5.  desVelocityZ = Kp*e + Ki*(I + e*dt) + Kd*(e - ePrev)/dt.
6.  Obtain Current velocities using ROS messages
7.  e' = desVelocityZ - currentVelocityZ   // This operation isn't required
    when we use a FCU
8.  throttle = Kp'*e' + Ki'(I' + e'*dt) + Kd*(e' - ePrev')/dt
9.  Calculate individual motor velocities using motor mixing algorithm.
10. Command the motors using ROS controllers
11. Repeat 2.-10. Till error is minimized.
12. Hover
```

PID Equations:

$PID\ for\ velocity,$

$$u_z(t) = K_p . e_z(t) + K_i \int_0^t e_z(\tau)d\tau + K_d . \frac{d\, e_z(t)}{dt}$$

$$where, \qquad e_z(t) = z_{ref}(t) - z(t)$$

$$e - error,\ z - height,\ t - time$$

$PID\ for\ throttle,$

$$u_{\dot{z}}(t) = K_p' . e_{\dot{z}}(t) + K_i' \int_0^t e_{\dot{z}}(\tau)d\tau + K_d' . \frac{de_{\dot{z}}(t)}{dt}$$

$$where, \quad e_{\dot{z}}(t) = \dot{z}_{ref}(t) - \dot{z}(t) = u_z(t) - \dot{z}(t)$$

$$\dot{z} - velocity\ in\ z\ direction$$

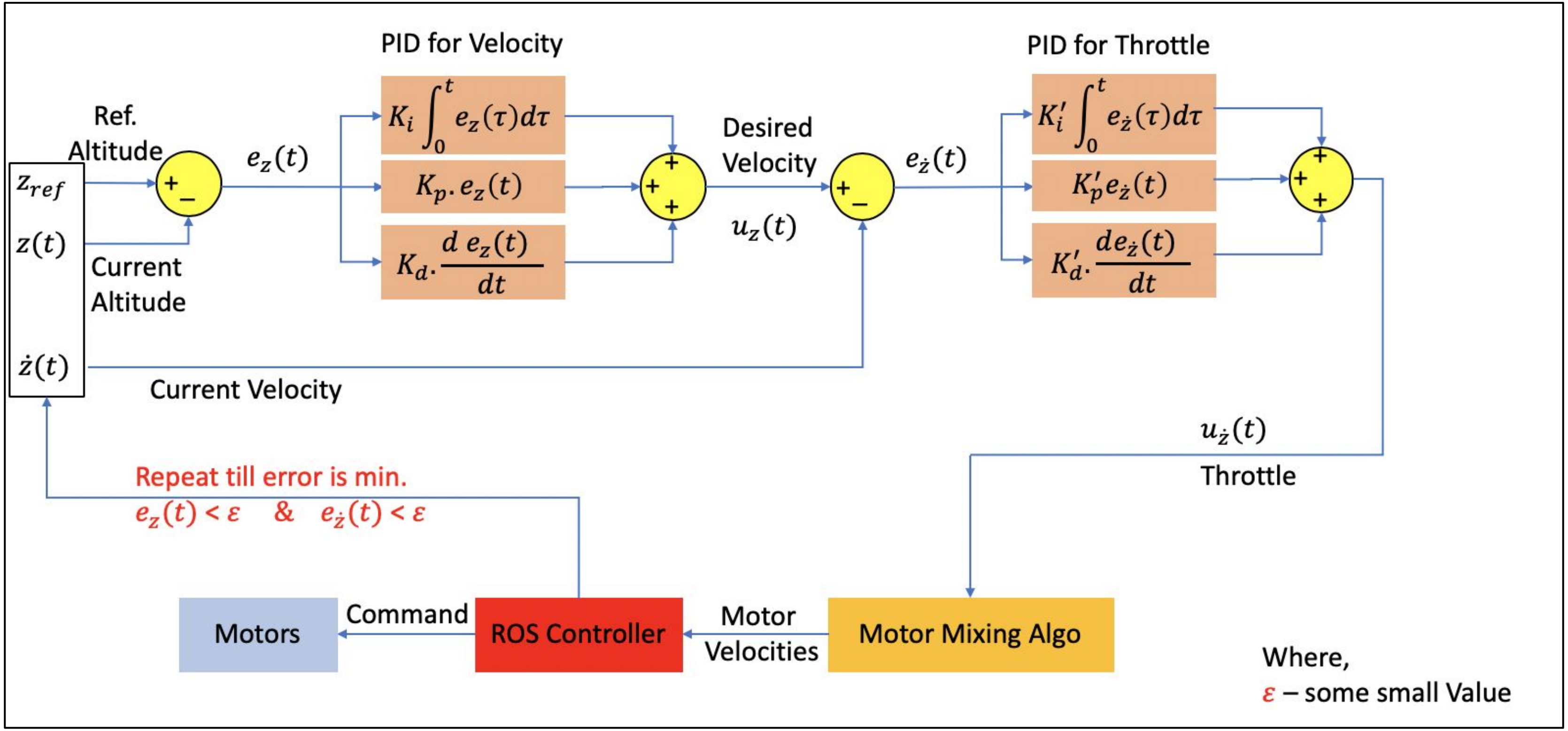
The on-board ultrasonic sensor will be used for depth measurements while the in-built IMU will assist the autopilot of the FCU to change the throttle values to the motors.

As mentioned earlier, we obtain the position of the Hexacopter by using geomatry_msgs ROS message and ROS Controllers to command the motor velocities.

Autonomous Flight Algorithm Details

I. Take Off & Landing Logic (Altitude Control) Contd...

Control system architecture



Autonomous Flight Algorithm Details

II. Flight Control (Horizontal Control)

In order to navigate through the gates, it is essential to have control of the Hexacopter over the x-y plane while maintaining a desired steady altitude. This can be achieved by a controller that controls the horizontal x-y position using the *frame detection algorithm* to get a desired horizontal position. The error in horizontal position is used by the PID controller to obtain a desired horizontal velocity. Similar to altitude controller, this desired value can be directly fed to the FCU to obtain the pitch & roll setpoints and accordingly change the thrust values. But for the purpose of simulations, we used a PID controller to get the horizontal acceleration from velocity. To obtain the pitch & roll setpoints from horizontal acceleration we applied some mathematical approximations. Yaw always remained 0 as the Hexacopter each axis individually.

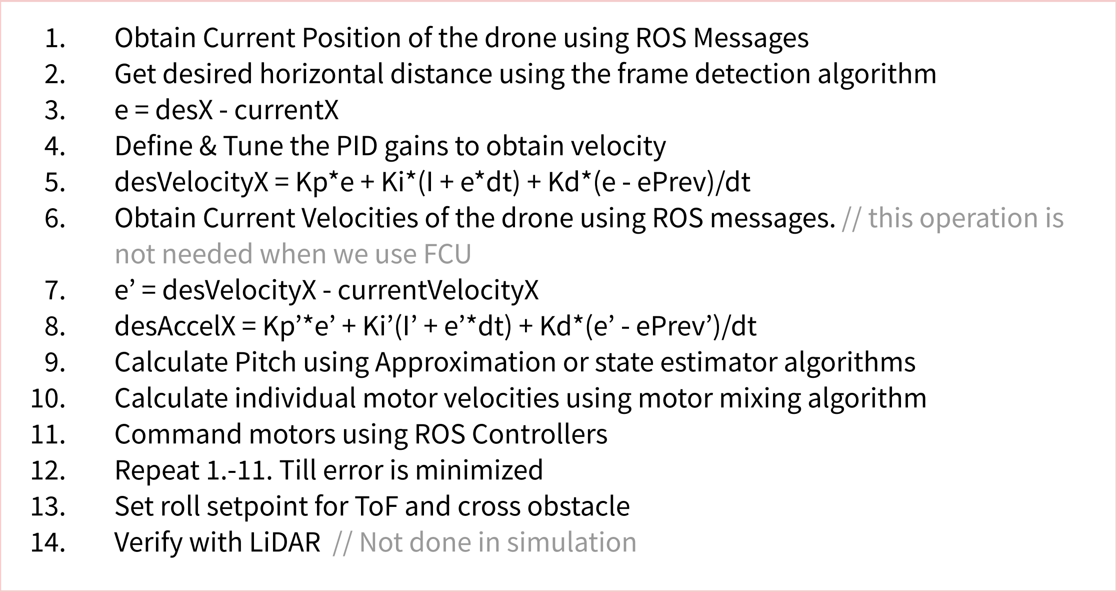
x - Control	**y - Control**
The error in x direction is obtained from the *frame detection algorithm.* This error is fed into the two PID controllers to obtain the pitch setpoints. This is fed into the the motor mixing algorithm to change x position of the Hexacopter. For demonstration purposes, we placed the gates at 2m, 1m & -1m as seen in the videos.	Once the desired x position is acheived, the Hexacopter needs to pass the gate. Now a roll setpoint is obtained for a certain Time of Flight (ToF) to make sure the gate is passed (verified by the LiDAR). For demonstration purposes, we varied the ToF and setpoints to see how the our model behaves.

Video Links: [1] [2] [3] [4] [5] [6]

Autonomous Flight Algorithm Details

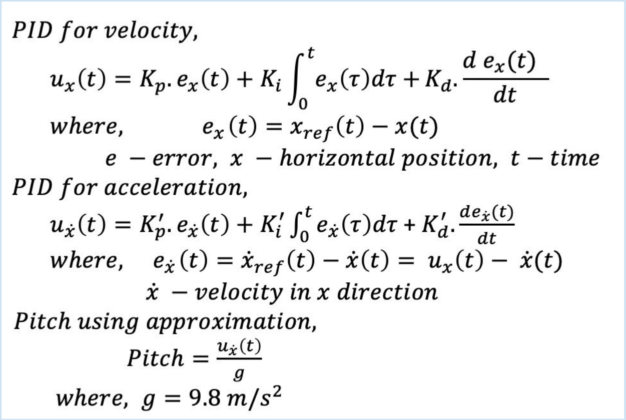
II. Flight Control (Horizontal Control) Contd...

Pseudo Code (implemented using python in ROS Package):



```
1.  Obtain Current Position of the drone using ROS Messages
2.  Get desired horizontal distance using the frame detection algorithm
3.  e = desX - currentX
4.  Define & Tune the PID gains to obtain velocity
5.  desVelocityX = Kp*e + Ki*(I + e*dt) + Kd*(e - ePrev)/dt
6.  Obtain Current Velocities of the drone using ROS messages. // this operation is
    not needed when we use FCU
7.  e' = desVelocityX - currentVelocityX
8.  desAccelX = Kp'*e' + Ki'(I' + e'*dt) + Kd*(e' - ePrev')/dt
9.  Calculate Pitch using Approximation or state estimator algorithms
10. Calculate individual motor velocities using motor mixing algorithm
11. Command motors using ROS Controllers
12. Repeat 1.-11. Till error is minimized
13. Set roll setpoint for ToF and cross obstacle
14. Verify with LiDAR  // Not done in simulation
```

PID Equations:



PID for velocity,

$$u_x(t) = K_p . e_x(t) + K_i \int_0^t e_x(\tau) d\tau + K_d . \frac{d\, e_x(t)}{dt}$$

$$where, \qquad e_x(t) = x_{ref}(t) - x(t)$$

$$e\ - error,\ x\ - horizontal\ position,\ t - time$$

PID for acceleration,

$$u_{\dot{x}}(t) = K_p' . e_{\dot{x}}(t) + K_i' \int_0^t e_{\dot{x}}(\tau) d\tau + K_d' . \frac{de_{\dot{x}}(t)}{dt}$$

$$where, \quad e_{\dot{x}}(t) = \dot{x}_{ref}(t) - \dot{x}(t) = u_x(t) - \dot{x}(t)$$

$$\dot{x}\ - velocity\ in\ x\ direction$$

Pitch using approximation,

$$Pitch = \frac{u_{\dot{x}}(t)}{g}$$

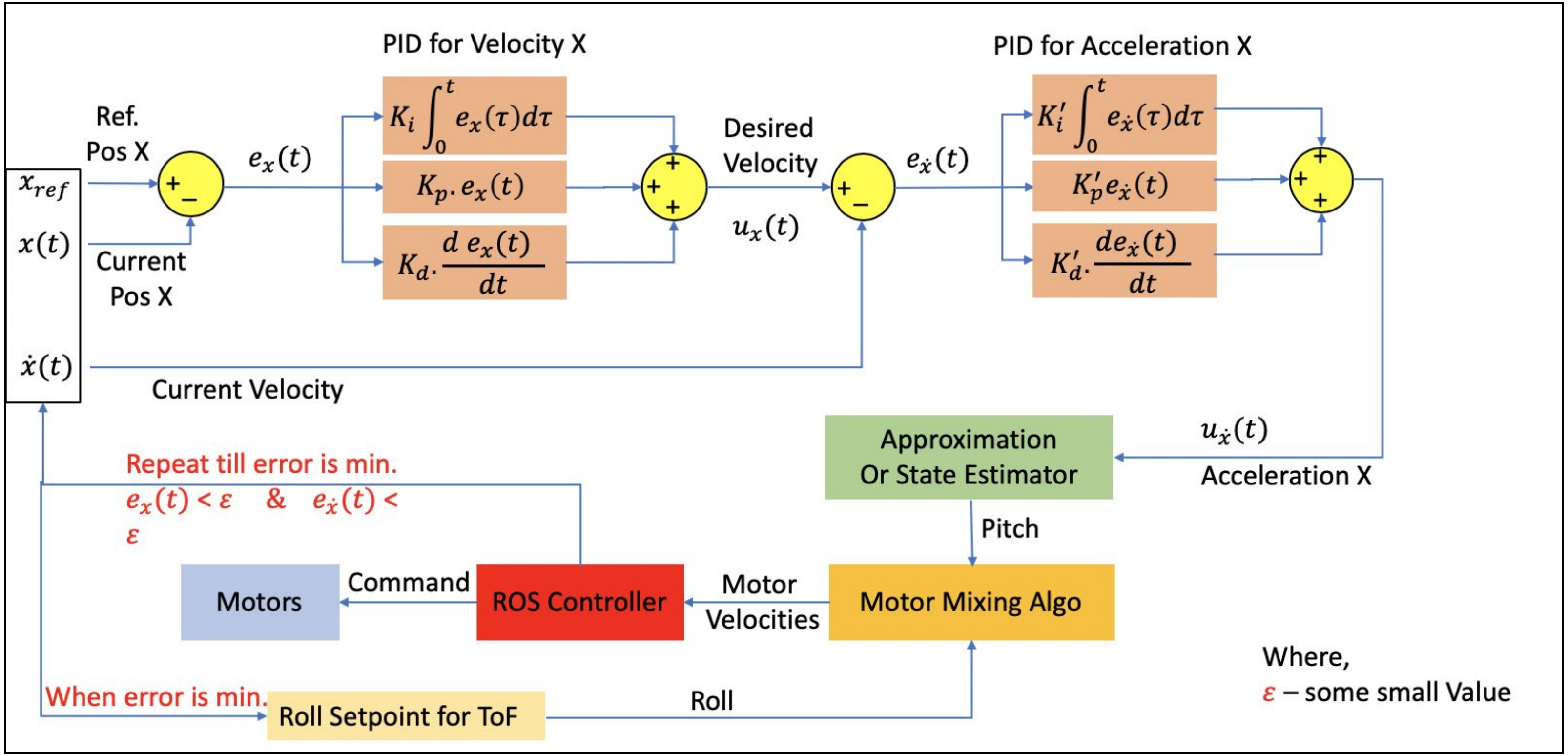
$$where,\ g = 9.8\, m/s^2$$

Roll, Pitch & Yaw can be calculated using the accelerations and angular velocities using state estimators like Kalman filters or simpler filters like Complementary filter.
But for simulation purposes we chose to do some approximations.
The filters have already been implemented[link] separately and just require integration with the final model.

Autonomous Flight Algorithm Details

II. Flight Control (Horizontal Control) Contd...

Control system arch



Autonomous Flight Algorithm Details

III. Trip management

Initially the Hexacopter is placed at the start position. Some amount of time will be taken in calibrating the sensors and the IMU. Once the initial calibration is done, the Hexacopter is ready to fly.

The ultrasonic depth sensor calculates the altitude of the Hexacopter when on ground, Take Off is initiate to reach the desired altitude. Eventually the desired altitude is reached with minimized error in desired altitude and current altitude.

Now, the Frame Detection Algorithm is employed and the horizontal distance between the center of the frame and the center of the largest frame is noted. This initiates the Horizontal Control to move along x-y plane to the reach the desired horizontal position. The Hexacopter moves along X direction first, then along the Y direction to pass through the gate.

Once the gate is passed, it is verified by the LiDAR.

This process in repeated for rest of the gates. And once all of them are complete, Landing is initiated and the Hexacopter lands.

Frame Detection: Introduction

1. The Frame detection task involves identifying the closest gate in proximity of the drone, and determining its center(C) along its x-axis.
2. The drone would use this to compute the error between this obtained center and the center point of the fisheye camera’s POV(C’), viz C-C’ in the horizontal direction.
3. Since all the frames are at the same height, and the drone calibrated to maintain a constant altitude, the need to determine the error in the z-axis (vertical axis) has been eliminated.
4. Detection of this frame involves a lot of assumptions, due to us not having access to true video recordings of the drone mid flight, and we plan on tuning the parameters once we have the bot on hand.
5. Presently, we are using screen captures of a custom world generated using Gazebo, which simulates a pseudo-realistic environment.

Frame Detection: Choice of Algorithm

Several different approaches can be used to detect these frames, each having its own merits and bottlenecks. We considered the following two approaches:

- **Neural Network:** A Deep Convolutional Neural Network, based on a pre trained Faster R-CNN ResNet-50 architecture using PyTorch. The approach, although novel and having much better accuracy would require a much more expensive On Board Computer, similar or better than the Jetson Nano to deliver at required speeds again adding higher power requirements and reducing the flight time, and increasing the budget. This approach was inspired by the following Project.

- **Computer Vision:** An OpenCV based script that uses various Computer Vision algorithms such as HSV based thresholding, Adaptive Thresholding, Morphological Transformations, Canny Edge Detection, Contour Detection algorithm among others (described in detail in the proceeding slides) to create a bounding rectangle on the closest frame, and determine its center in the x-axis. This approach was comparatively much less computationally intensive, consumed much lower power, and still performed reasonably well in our test environments.

Choice of Algorithm

The computer vision algorithm vision is basically an infinite loop that continuously polls for frames from the video feed of the on board camera and performs mathematical operations on it. We experimented with various operations that are known to give high accuracy of object detection, and removed those which did not aid to the performance. The bare-bones sequence of operation looked as follows:

1. Gaussian Blur
2. Hue-Saturation-Luminance Based Thresholding
3. Morphological Operations: Erosion/Dilation/Opening
4. Adaptive Thresholding: Adaptive Gaussian Thresholding vs Adaptive Mean Thresholding
5. Contour Search

The proceeding slides provide a detailed overview of filters considered.

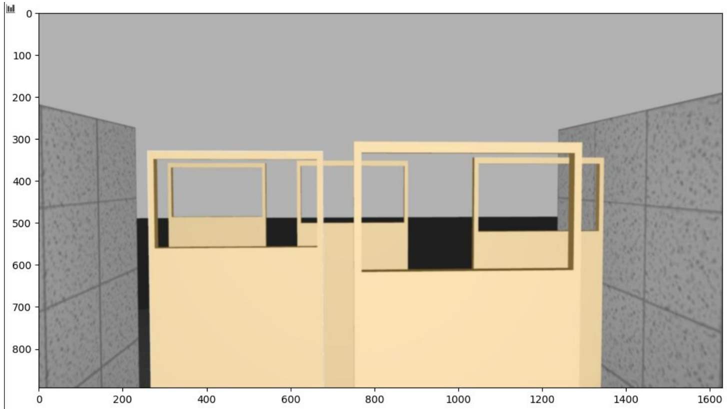


Fig 29a. Sample Frame used for Tuning

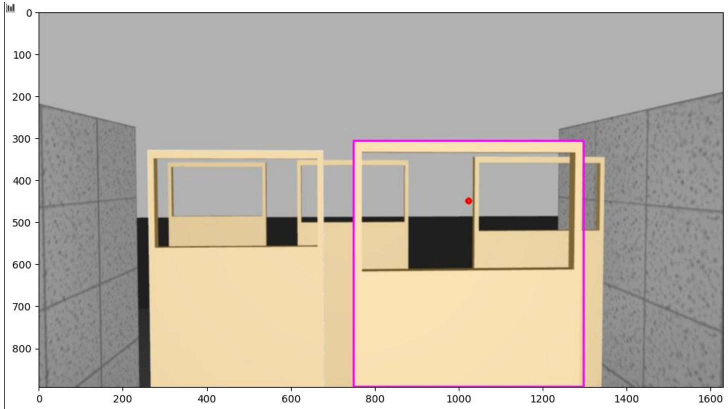
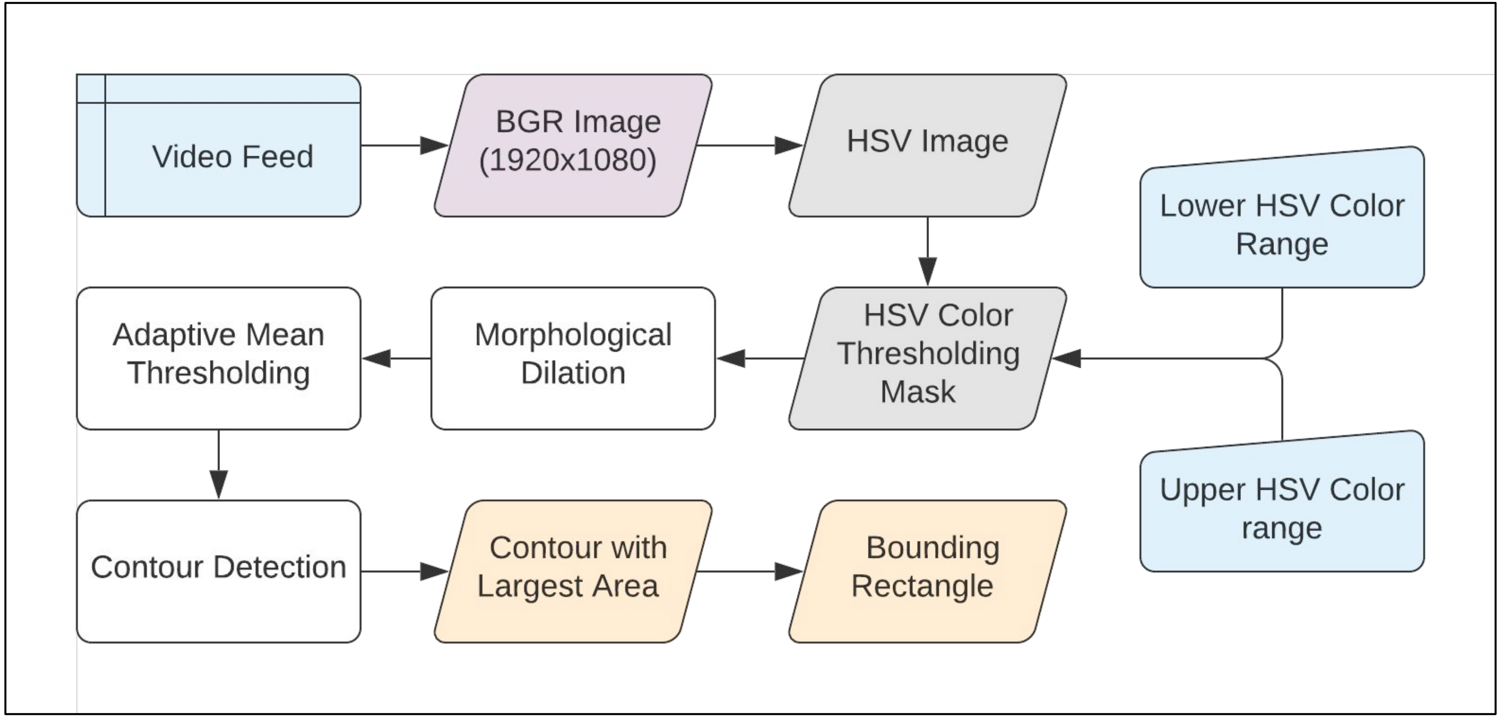


Fig 29b. Frame Detection on Sample Frame

Simplified Algorithm Overview



Color based Thresholding

Our object of interest, the frame has a specific color. This property can been used to create an image mask, effectively removing any pixel whose color does not lie within a given margin of this color. The color can be specified in various formats, or color spaces: HSV, RGB, YC_BC_R, CIELAB Color Space. In our particular case, the frames are easily isolated in the HSV Color Space, and hence that was used. This can be verified using Principal Component Analysis.

The first step in this process was to find a suitable values of the Hue, Saturation and Luminance for the Upper and Lower Margins. The Color Thresholder App from the Computer Vision Toolbox in MATLAB provides a simple interface to determine these optimum values. For the simulation, these values lie in a very small range, however for real world dataset it will be tuned separately. One of the major issues is that the Luminance values are highly dependent on the lighting conditions. To tackle this problem we plan on having a flashlight system. This would also result in uniform and greater illumination of gates right in front of them, and hence increase the luminance values to the required region.

This filter simplifies the problem greatly removing a lot of the unnecessary features.

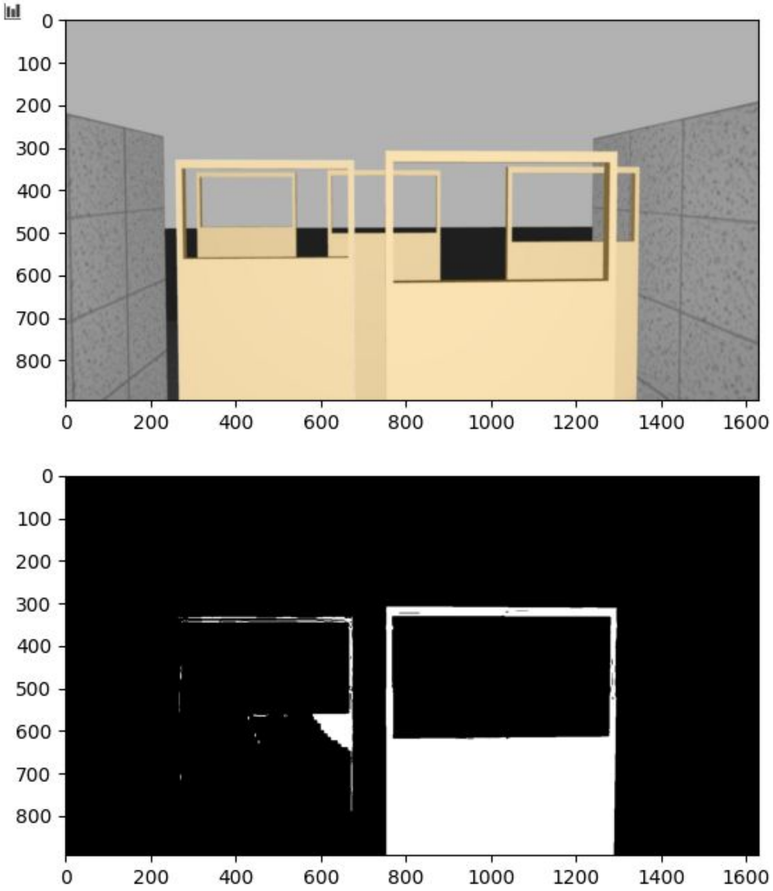


Fig 30. Result of HSV Mask on the Sample Image. Notice how most of the frames in the back have been eliminated due to lower luminance

Morphological Operations

If you notice carefully, the output of the HSV Masking Operation has some resultant noise, and small cavities in the frame shape. Morphological Operations are the perfect solution for dealing with these artifacts. The two basic operations erosion and dilation, fills in small holes and remove small objects respectively. A Morphological Opening is an erosion followed by a dilation and it removes small objects while preserving the dimensions. A morphological closing is a dilation followed by an erosion, and it fills small gaps while preserving dimensions.

Over a series of experiments, we observed that the Closing operation consistently performed better at removing the artifacts; This can mainly be attributed to the fact that gaps in the bigger gates are a bigger problem than noisy artifacts when creating the contours.

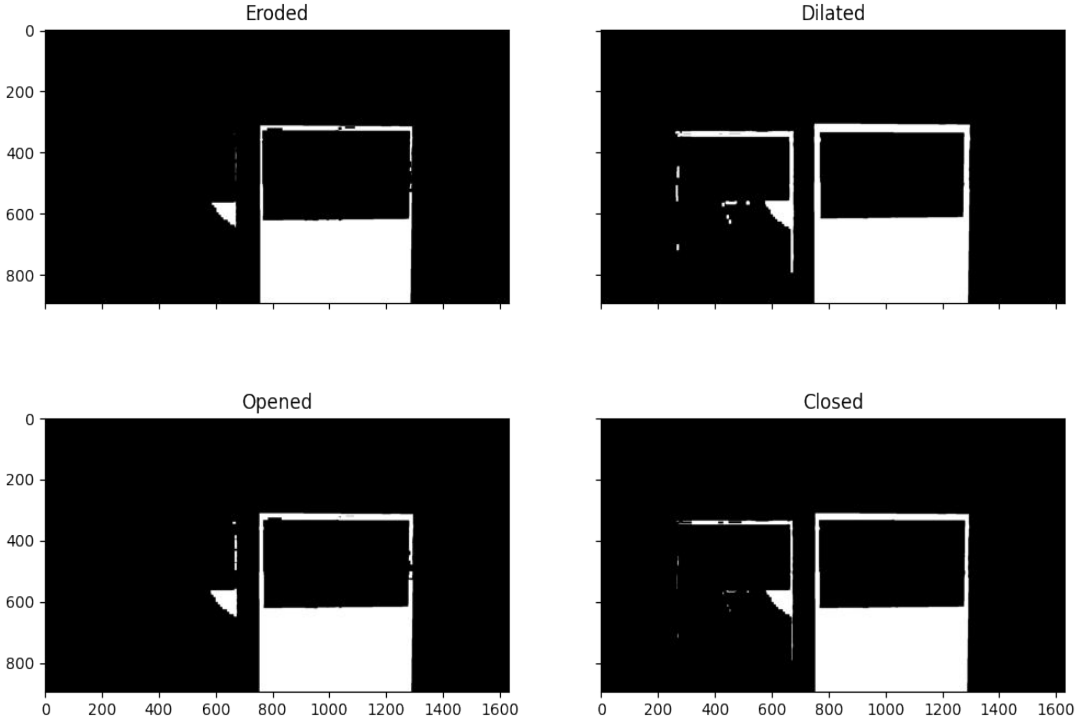


Fig 31. Erosion removes most Artifacts, but also creates new ones in the primary frame. Dilation reduces the gaps in the frames, but increases noise. Opening removes noise and fills gaps to some extent. Closing fills gaps but not very useful against noise

Other Operations

1. **Gaussian Blur**

We skipped the Blurring Procedure owing to the fact that the screen caps used were of a simulation and free from noise, leaving the blurring redundant. We expect to use it in the final version

2. **Adaptive Thresholding**

Adaptive Thresholding segments the image by comparing a pixel value with its neighbours. We tested Adaptive Mean Thresholding and Adaptive Gaussian Thresholding, both giving very promising results. However, our model was able to perform almost as well without it and was hence removed as it is quite a costly operation.

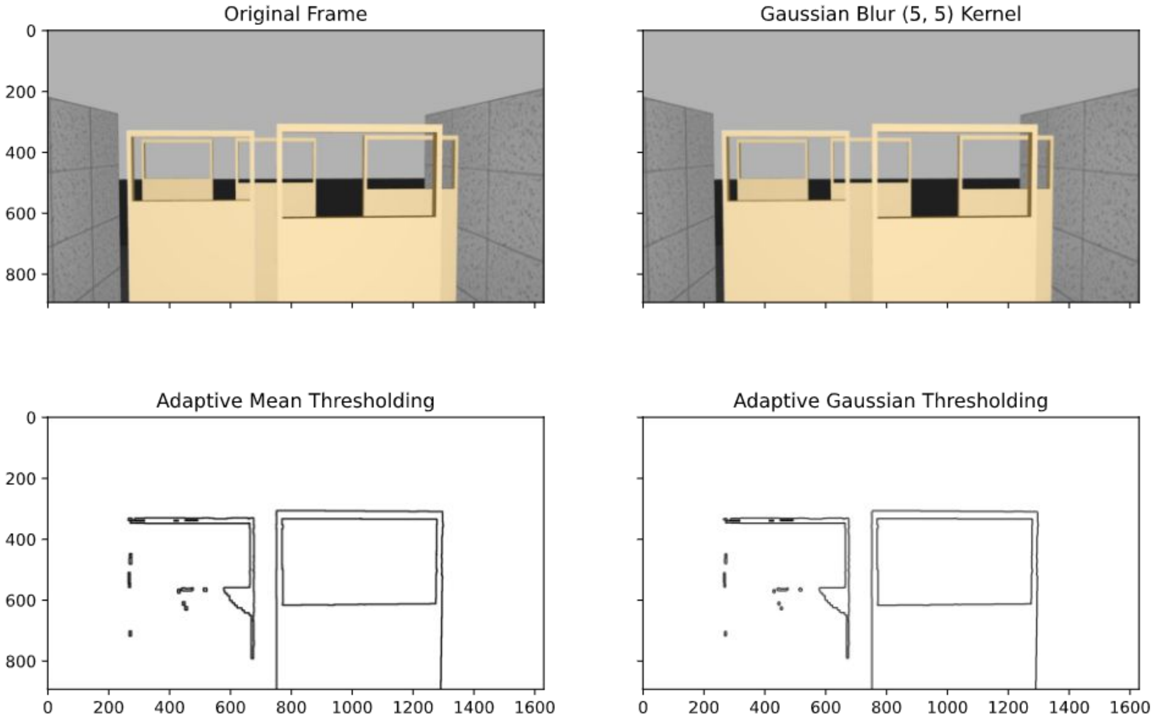


Fig 32.

A. *The difference between blurred and original frame is not very obvious, and unnecessary*

B. *Excellent boundaries generated by adaptive thresholding. Direct Feature extraction possible*

Final Operations

The final task of the pipeline is finding the contours and determining the one we need. We found that almost always the biggest contour was the frame. After identifying the correct contour we draw a bounding box around it, which gives us the necessary horizontal position of the center of the frame. A sample codebase can be found on one of our member’s github.

Improvements

1. The models we currently use, extend beyond the main window and hence it is not possible to determine the z error.
2. Often times, the bounding box becomes the entire frame. We can also impose a maximum and minimum range of areas the contours can have, resulting in even lower errors.
3. Once a better frame model is designed, we can also impose ratio restrictions on length and width of frames; This would eliminate false positives even further.

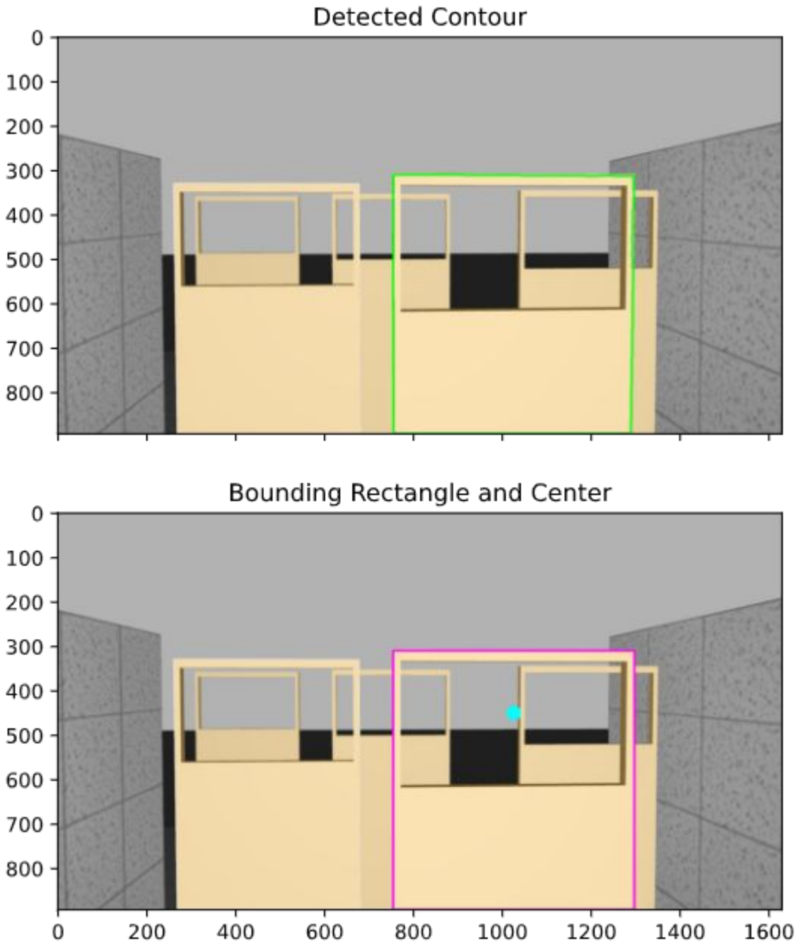


Fig 33. Contour and Bounding Rectangle on the Captured Frame.

Eventual Problems that we may encounter

Mechanical:
The top and bottom plates that house the electronics has to be fabricated. We will have to approach vendors who offer CNC services to fabricate the plates in Aluminium.

Although the mounting brackets are designed in CAD, for prototyping purposes we plan on using double sided tape. This is a temporary solution, and to make it production quality we will have to manufacture the mounting brackets.

Electronics:
With the 8000mAh battery, we estimate the flight time to be 45 minutes based on simple mathematical calculations. We are confident that it will be enough to meet the requirement, but skeptical about long flights.

Image Processing:
Since we plan of using a fisheye lens to cover the entire environment, corner rectangles might appear skewed. We plan on tackling this problem hands-on once we get a fair idea about the image through the fisheye lens.

Component Details

1. FCU - PIXHAWK 2.4.8: It is the Flight Controller Unit that allows use to integrate the hardware with the software stack. It also contains an Inertial measurement Unit that will be used to detect the movements of the drone.
2. Raspberry Pi: This onboard computer is capable of running the entire software stack seamlessly.
3. Raspi Camera: It’s compatibility and functionality with Raspberry Pi make it the first choice.
4. LiDAR - TFMini Plus: This sensor is Time of Flight sensor that is capable of measuring distances up to 12m. It adds additional functionalities to the Hexacopter that make it versatile to perform various other tasks. In the current scenario, it will be used to check if the gate is passed successfully or not.
5. MB1000-LV Ultrasonic Sensor: An ultrasonic sensor that can measure upto 6.5m with an accuracy of 0.025m. This will allows us to measure the altitude of the Hexacopter as the Barometer in the FCU will not work effectively in indoor environment.
6. 30A ESC: As the name suggests, the Electronic Speed Controller will be used to vary the speed of the BLDC.
7. EMAX MT3110 700KV Brushless DC Motor: EMAX MT3110 700KV Brushless DC Motor is specially designed for moderate-heavy lift applications while being highly efficient and reliable to get the desired amount of thrust.
8. Hexacopter Frame: The arms will be ordered. But as we modified the metal plates, we plan on fabricating it contacting a vendor.
9. Orange 8000mAh 4S 25C/50C: The 4 Cells in Series will give us enough room for changes in rpm of the motor, and 8000mAh capacity battery will give a prolonged flight time.
10. Propeller 11x4.7: These 11 inch propellers along with the BLDC will provide enough thrust to lift the weight of the payload.

Component Sources, Price & Total Cost

Component	Qty.	Total Price	Source
FCU - PIXHAWK 2.4.8	1	₹5,950.00	[1] [2]
Raspberry Pi 4 - 2GB RAM	1	₹3,190.00	[1]
Camera - Raspi Cam v2	1	₹1,999.00	[1]
LiDAR - TFMini Plus	1	₹3,399.00	[1]
MB1000-LV Ultrasonic Sensor	1	₹2,210.00	[1]
LiPo - 4S battery	1	₹7,799.00	[1]

Component Sources, Price & Total Cost (contd.)

Components	Qty.	Total Price	Source
ESC - 30A	6	₹1,950.00	[1]
BLDC - MT3110/BR3508/MN3110 700kv motors	6	₹16,848.00	[1] [2] [3]
F550 arms	6	₹900.00	[1]
LiPo Battery Charger	1	₹3,600.00	[1]
Propellers 1147(11X4.7)	6	₹423.00	[1]
Fabrication + Miscellaneous	-	₹2,500.00	-
Total		₹50,768.00	

Execution plan with timelines

Deadline	Goals & Execution Plan
Aug 17	● Import CAD Models into ROS World ● Extend the simulation solution for all 15 gates, refine the image processing solution. ● Unify the ROS Package and Image Processing Simulation and refine the pipeline
Aug 25	● Further develop the ROS package to incorporate autopilot software with mavros and other hardware support packages for other sensors. ● Complete a software run and simulation of the entire package. ● Finish the event simulation for the drop from a height in SimScale.
Sept 1	*Prototyping phase begins:* ● Mechanical assembly, Fabrication, 3D printing of all the components. ● Build and use a Thruster Stand to calculate the practical thrust. ● Build prototype gate and frame for testing purposes. ● Record actual footage for tuning of Image Processing Pipeline
Sept 8	● Set up the connections and test all electronics. Setup ROS package on the on-board computer. ● Test the sensors and the camera individually with the unified ROS Pipeline. Changes imminent to the ROS package at this stage.
Sept 15	● Final integration of the ROS package with the hardware. ● Test runs, final tuning and any small changes to make the final product.

Note: The Timeline has been drawn assuming the funding is processed, and parts delivered by 1st September. However, owing to the present scenario viz-a-viz covid-19 pandemic, the deadlines might have incumbent delays including the delays of the final round.

Current Progress

- We have a complete CAD model along with the results of the CFD simulations. Which can be used during the time of assembly.

- The basic control system architecture has been implemented in the ROS package. We will build on that as we progress by incorporating sensors, other hardware, etc.

- An image processing solution has been implemented using OpenCV and Python. We plan on refining it further and optimizing its performance.

- The structure of the Gazebo simulation is complete, the solution needs to be extended to all gates and the urdf model needs to be modified to the real CAD model.

Drive Link & Source Codes:

- CAD Model
- All simulation videos
- Image Processing & Target Detection Source Code
- ROS Package & Simulation Source Code

Please note that changes might have been made to the source code at the time of evaluation, so this is just for the purpose of reference and not for usage.

References

CFD & Mechanical:

1. Role of Mach numbers in compressible flows
2. SST k-omega model - [1] [2]
3. Difference between k-omega, k-epsilon, spalart-allmaras turbulent models
4. Simple solver for driven cavity flow problem
5. SIMPLE algorithm
6. Rotating Zones
7. Propeller thrust
8. Propeller thrust calculator form
9. Von mises yield criterion

Image Processing & Control Systems:

1. Yuntian Li, Matteo Scanavino, Elisa Capello, Fabrizio Dabbene, Giorgio Guglieri, Andrea Vilardi, *“A novel distributed architecture for UAV indoor navigation”. International conference on air transport - INAIR 2018*
2. Johan Fogelberg*, “Navigation and Autonomous Control of a Hexacopter in a Indoor Environment”, Lund University.*
3. *D. Sharipov, Z. Abdullaev, Z. Tazhiev and O. Khafizov, "Implementation of a mathematical model of a hexacopter control system," 2019 International Conference on Information Science and Communications Technologies (ICISCT), Tashkent, Uzbekistan, 2019, pp. 1-5, doi: 10.1109/ICISCT47635.2019.9011842.*
4. *H. Chen, "Monocular Vision-Based Obstacle Detection and Avoidance for a Multicopter," in IEEE Access, vol. 7, pp. 167869-167883, 2019, doi: 10.1109/ACCESS.2019.2953954.*
5. *Badrloo, Samira & Varshosaz, Masood. (2017). VISION BASED OBSTACLE DETECTION IN UAV IMAGING. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. XLII-2/W6. 21-25. 10.5194/isprs-archives-XLII-2-W6-21-2017.*
6. *Chi Zhang and P. Wang, "A new method of color image segmentation based on intensity and hue clustering," Proceedings 15th International Conference on Pattern Recognition. ICPR-2000, Barcelona, Spain, 2000, pp. 613-616 vol.3, doi: 10.1109/ICPR.2000.903620.*