FlySense

"Augmented Reality Assisted FPV navigation for aerial vehicles"

FINAL PROJECT REPORT

Team C
Shivang Baveja
Nicholas Crispie
João Fonseca Reis
Harikrishnan Suresh
Sai Nihar Tadichetty

Sponsor: Near Earth Autonomy

May 7, 2018

Abstract

This document extensive details of the project undertaken by MRSD 2018 Team C under MRSD Project course. MRSD Project course is a core element of the MRSD program at Carnegie Mellon University, and allows students to work in teams towards a common goal of developing a robotic system which can be used in an identified problem scenario. This document summarizes the activities carried out by Team C – FlySense from August 2017 to May 2018 to develop their Augmented Reality based assistive technology for aiding aerial navigation.

The report begins with a basic description of the project, highlighting the need for the assistive system, followed by a use case that clearly depicts how the system will be used. A list of system level requirements, both performance and non-functional are presented. The approach is graphically depicted in functional and cyber physical architecture. The key component selection is motivated by system level trade studies.

A detailed description of the system is then provided including sub-system design, modelling, testing and overall system evaluation during Spring Validation Experiment (SVE) and SVE Encore. Further, project management section provides details about schedule, budget and risk management. The report concludes with lessons learned, future work and references.

Table of Contents

Project Description 7

Use Case 7

System Level Requirements 9

Mandatory performance requirements 10

Mandatory non-functional requirements 11

Desired performance requirements 11

Desired non-functional requirements 12

Functional Architecture 12

System Level Trade studies 14

Cyber-Physical Architecture 14

System Description and Evaluation 16

Aerial subsystem 16

DJI Interface with custom flight mode for obstacle avoidance 17

Filter point cloud based on flight envelope 18

Obstacle mapping and Bird's Eye View generation 19

Sound warning generation

Gazebo simulation 19

User subsystem 21

Modeling, analysis, and testing 23

Aerial subsystem 23

User system 24

Flight test summary, learnings and progress 28

Spring Validation Experiment Evaluation 28

Strong and Weak points of the system 29

Project Management

10.1. Schedule 30

10.4. Budget 32

10.5. Risk Management 32

Conclusions 35

Lessons Learned 35

References 35

1. Project Description

Helicopter pilots have one of the toughest jobs in the world. Their jobs are usually task and sensory saturated, with limited ability to process new information and many different controls to be used in an instant. However, there aren't many aids for helicopter pilots that present useful information in a relevant way. The U.S. military has invested millions of dollars in state-of-the-art headsets for conveying all sorts of information to fighter pilots in real-time, but nothing close to that technology has been introduced in the commercial domain given the current price point and the focus on assisting firing and targeting systems.

Helicopter pilots face difficulties in different phases of flight and mission types. Some of these are low-altitude flights, landing in tight spaces with fixed structures and navigation in low-visibility scenarios. Out of the listed flight stages, one of the most critical is a flight at an altitude below 200ft AGL (Above Ground Level) where, unlike commercial airplanes, there are no autonomous piloting features in place to aid with the landing. Helicopter pilots resort to their instruments, but above all look for visual landmarks to understand their environment and judge how far they are from obstacles. This can be even more difficult when flying in unfamiliar environments, like in areas where the landscape is monotonous (e.g. desert, or a grass field) or in situations where it's hard to judge obstacles that can cause a crash (e.g. a pole near the tail rotor).

Through this project, we developed a pilot assistance system using Augmented Reality, that gives the pilots enhanced situational awareness in the least intrusive way. With our FlySense system, the pilot will be better equipped to handle the difficult flight scenarios mentioned above as he will rely on the visual and audio warnings informing him about the possibility of collisions. FlySense will offer a high level of assistance through mapping of surrounding obstacles and low-level autonomy to override bad decisions by the pilot.

2. Use Case

Lori is an EMS helicopter pilot operating out of the University of Pittsburgh medical center in Oakland. As soon as she comes into work for the day, an alert comes in from the dispatcher. "We have multiple severe injuries in a multiple vehicle collision on highway 30 near Clinton. I'm sending in the GPS coordinates now, we need an EMS helicopter for the victims as soon as possible!"

Figure 1: EMS helicopter pilot at the ready

Lori grabs her gear and heads out to the helicopter. Upon reaching the helicopter, she grabs her FlySense visor and turns it on. Within seconds the headset boots up. After a 10 second

calibration procedure, a couple options pop up. With a quick selection, Lori selects a trip planning view and enters in the GPS coordinates on the flight computer.

As she starts to take off, a member of the grounds crew sets off her obstacle alarms as he runs across the helipad. She couldn't even see him herself from her vantage point, so it was a good thing she had full coverage visually and with sound warnings from the Bird's eye view, which automatically popped up in the takeoff sequence.

Once Lori ascends to 200 feet, she engages the autopilot. This makes her job a lot easier, but her FlySense display is still active on "Heads Up Display" view. It gives her constant updates on her attitude, altitude, and a view of the horizon, even as she goes through a couple low hanging fog banks. It also shows her location of flying vehicles around her and what path to take to reach the destination all within the HUD mode. The flight was going smoothly until she reached the destination and had to find a good landing spot (**Figure 2**).

The accident she is responding too has swarms of EMS vehicles around the crash, and a set of power lines and multiple trees are surrounding the area, making it a little tricky to come in. As she descends, she looks at the Bird's eye view image on her screen to check her distance to the power lines, but those are only yellow, without any sound warnings, so she knows she is clear but to stay careful.

Figure 2: Lori trying to land on at a congested highway accident scene

As soon as she touches down, her medical crew springs to action and gets 2 patients aboard the helicopter in minutes. They need to make it back to UPMC as soon as they can since both patients have lost a lot of blood and they only have limited medical resources to cope with that on the helicopter. Lori guides the helicopter backwards outside of the area of trees and power lines, relying heavily on her bird's eye view to safely extricate her helicopter from the tricky situation, even as the wind starts picking up dramatically. Once safely up and preventing a dangerous crash, Lori guides the helicopter and the patients back to the medical center.

3. System Level Requirements

To accomplish the design, development and ultimately deployment of the system, the team is following a systems engineering approach. The core system requirements were defined at the beginning of the project after careful analysis, research and deliberations among the team and stakeholders. All the effort of the team is directed towards fulfilling these requirements.

Since the commencement of the project, a few requirements have been modified based on the feedback received from Pilots at Near Earth Autonomy (NEA). These changes were reported in the beginning of the spring semester. The finalized requirements have been stated in Tables 1-4 below along with their evaluation status.

Each requirement is mapped to a subsystem (Aerial or User). Also the requirements marked with asterisk were the ones modified.

3.1. Mandatory performance requirements - Table 1

Subsystem		Description	Evaluation Status
Aerial	M.P.1	Receive, and process point cloud data from one Velodyne VLP-16	Validated
User	M.P.2*	Recognize 5 voice commands with an accuracy of 90% without noise and 70% with noise	Validated during Fall but was deemed unnecessary. Feature removed from the final system.
Aerial	M.P.3	Detect obstacles in the flight envelope projected 5 seconds into future	Validated
Aerial	M.P.4	Detect obstacles of size greater than 2m x 2m located at distances less than 10m	Validated
Aerial	M.P.5	Generate Bird's eye view image in vehicle frame at a rate of at least 10 Hz	Validated
Aerial	M.P.6	Color obstacles in bird's eye view (red, yellow, green) based on pilot inputs and time to impact. Red corresponds to the lowest time to impact, followed by yellow and green.	Validated
Aerial	M.P.7	Override pilot commands to stop the aerial system at least 1m before the obstacle	Validated
User	M.P.8	Render all modes on the AR interface at refresh rate of at least 10 Hz	Validated
User	M.P.9*	Generate audio warnings based on time to impact to obstacle and current pilot input. Two levels of sound warnings with same pitch for both but varying duration of the beeps.	Modified based on pilot feedback, Validated

3.2. Mandatory non-functional requirements - Table 2

Subsystem		Description	Evaluation Status

User	M.N.1	Easily set up (within 1 minute) by a single operator	Validated
User	M.N.2	Feel natural to the pilot, i.e. Project images at focal distance up to 20 meters	Validated
User	M.N.3	Wearable like normal glass	Validated
User	M.N.4	Comfortable to wear headwear for long periods of time, i.e. should weigh less than 1 pound.	Validated
User	M.N.5	Displays information in a clear and simple manner.	Validated
User	M.N.6	Be non-intrusive to the pilot, i.e. pilot should be able to see through the projected images.	Validated
User	M.N.7	Be non-distracting for the pilot, i.e. pilot should be able to engage or disengage the system as and when desired and with simple voice commands	Validated
Aerial	M.N.8	Solution hardware is more affordable than available solutions (Cost below 5000 USD)	Validated

3.3. Desired performance requirements - Table 3

Subsystem		**Description**	**Evaluation Status**
User	D.P.1	Voice commands personalized to 3 users	Dropped
Aerial	D.P.2	Override pilot commands to maneuver around obstacle maintaining radial clearance of at least 2m	Dropped
User	D.P.3	First Person View (FPV) video overlay on the AR interface at frame rate greater than 10Hz	Validated
Aerial	D.P.4	Segment obstacles into 2 categories (Trees or building)	Dropped
Aerial	D.P.5	Recommend feasible trajectory to goal maintaining clearance of at least 1m from all obstacles, and display in AR interface	Dropped

3.4. Desired non-functional requirements - Table 4

Subsystem		Description	Evaluation Status
User	D.N.1	Easily customizable to include more features and widgets	Dropped
User	D.N.2	Ability to integrate with flight simulators to train pilots	Dropped

4. Functional Architecture

The updated system requirements were used to update the functional architecture, which is shown below in **Figure 3**. The complete system operation is depicted in a block diagram capturing functions and overall flow of information. Here, we have divided the system into 3 stages which are happening continuously and concurrently. These are:

a. Input:
   i. Pilot Inputs: Our system being an assistive technology always has a pilot-in-the-loop. The pilot operates the quadcopter by relying only on the FlySense interface
   ii. Onboard Sensors: They are primarily used for perception and state estimation.

b. Process:
   i. The raw velodyne point cloud data is filtered to extract the relevant points based on the flight envelope and pilot inputs.
   ii. The filtered point cloud data is sent into 3 different channels, one for obstacle danger classification and coloring, another for the sound warnings algorithm and final channel for the emergency braking for obstacle avoidance.
   iii. The obstacle coloring and sound warnings merge together into the Bird's Eye View (BEV) image.
   iv. The FPV video is merged with the BEV and published.
   v. The emergency braking code produces control commands that are sent to the flight controller.

c. Output:
   i. The sensor information is rendered as a Heads Up Display (HUD) and overlaid on top of the FPV and BEV video.
   ii. Sound warning are generated based on the dangerous obstacle to alert the pilot.
   iii. The flight controller executes control commands to force braking of the quadcopter depending on a potential collision.

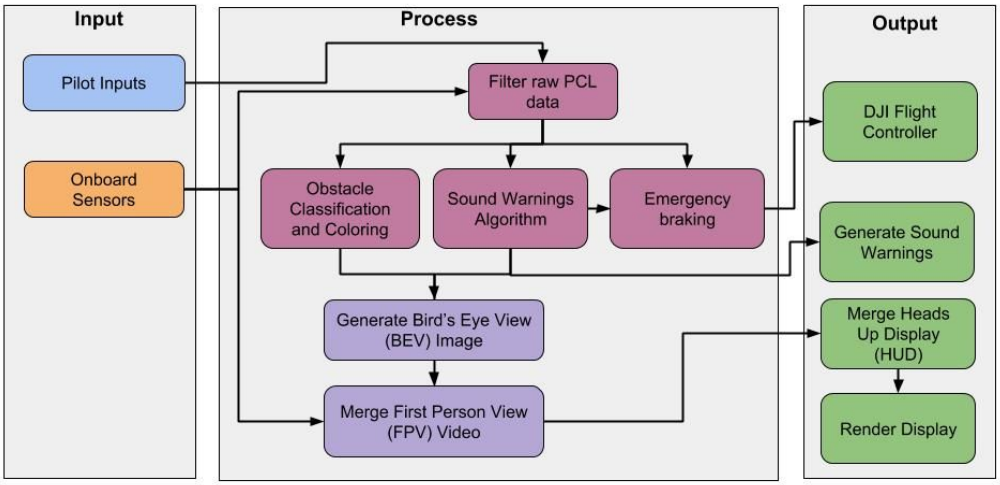


Figure 3: Functional Architecture

5. System Level Trade studies

Jetson Carrier Board

We needed to use a carrier board for the Nvidia Jetson TX2 since the development board was far too large to put on the quadcopter. We considered three different options, scoring each one with respect to the size, weight, which types of ports were available, and the logic circuitry which we would use to interface with the DJI.

Table 5

	Sprocket	**Orbitty**	**Elroy**
Size	87mm x 50mm	87mm x50mm	87mm x 50mm
Weight	28g	41g	35g
Ports	USB	HDMI, USB	HDMI, USB
Logic Circuitry	3.3V UART	RS-232	3.3V UART

Table 6

	Sprocket	**Orbitty**	**Elroy**
Size	5	5	5
Weight	5	4	4.5
Ports	2	5	5
Logic Circuitry	5	5	2
Total	4.25	4.75	4.125

Onboard power regulation:

As stated later in section 7, the aerial vehicle we had chosen is DJI Matrice 100 quadcopter which has been mounted with Jetson TX2, Velodyne VLP16 LIDAR and a FPV camera. The FPV camera is powered via USB from Jetson TX2. The quadcopter frame provides two power ports which allow powering the onboard components. The voltage output of these ports is 26V. Both the velodyne and Jetson operate at 12V, so a voltage step-down converter was required to power them. Table 7 describes the power modules we considered and the criteria used to select the one suitable for our needs.

Criteria/Power modules	CCBEC 10A PEAK 25V MAX INPUT SBEC	Step-down DC-DC Power Converter 25W	eBoot Mini MP1584EN DC-DC Buck Converter
Size	43x14x8 mm	46x50x20mm	22x15x2
Weight	21 grams	Not given	18
Current rating	6 A	2 A	3 A
Input Voltage	7V-26V	3.6V-25V	24 V

Table 7: CC BEC was selected as it provides higher current compared to the other two modules, is smallest and lightest compared to other two. Also, it is used by a lot of RC hobby pilots.

Communication system:

Reliable communication is important for us to provide real-time and accurate information of the surroundings to the pilot. We considered a lot of COTS communication radios which are used by the FPV community for transmitting video over long range. There were a few problems with them which are stated below:

1. But all these modules require a receiver in the aerial vehicle (which would add weight).
2. Also a lot of these radios transmit analog or digital video but the data is transmitted via serial protocol. This was a problem for us as the only way to establish connection between Epson (AR headset aka user system) and Jetson is via wifi.
3. These might cause interference with the DJI radio.

After considering these issues, we decided to stick to wifi based communication. During the Fall Validation Experiment (FVE), we faced a lot of issues with wifi communication. After testing we established the reason for these problems to the interference caused by DJI radio which operates at the same frequency of 2.4Ghz. To fix that issue we decided to operate at 5Ghz which had another benefit in that it allowed more bandwidth to transmit video.

We selected **Unifi AC-M wifi router along with UMA-D directional antenna** considering:

1. Possibility of attaching a directional patch antenna.
2. Transmission power greater than 20 dBm.
3. Supports the wifi protocol available in Jetson TX2 wifi module and Epson wifi module.

Augmented-Reality Headset:

Our system heavily depends on the quality of user experience, for this we absolutely need to make sure that there are no hiccups in the setup or visualization process. We have zeroed in on the capability and ease of programming as the most important factors.

Parameters	Weights	Microsoft Hololens	Google Glass	Vuzix	Meta 2.0	Recon Jet	Optivet ORA	Epson BT300
Capability	**25%**	5.0	2.0	2.0	4.0	3.0	2.0	2.5
Ease of Programming	**25%**	5.0	4.0	1.5	5.0	1.5	1.0	4.0
Cost	**10%**	4.1	0.3	2.5	0.3	3.4	0.0	2.5
Reliability	**10%**	5.0	4.4	3.3	5.0	5.0	2.8	4.4
Weight	**10%**	0.0	4.7	4.6	0.7	4.3	4.2	4.2
Hand Tracking	**10%**	5.0	5.0	0.0	5.0	0.0	5.0	5.0
Head Tracking	**10%**	5.0	2.5	5.0	5.0	0.0	2.5	5.0
Total	**100%**	**4.4**	**3.2**	**2.4**	**3.8**	**2.4**	**2.2**	**3.7**

Table 8: Trade studies performed for different Augmented Reality devices.

AR tread studies showed that Microsoft Hololens seemed best choice. But it turned out to have functionality issues that we did not anticipate. This made us consider the next best options; Meta 2.0 and Epson BT300.

- Meta 2.0 seemed to be a very good option but due to the huge gap between demand and supply we had to go ahead with the Epson BT300.
- The Epson turned out to be a lot better than we initially expected, it was very easy to program, robust to shocks, light weight and had good resolution. This sealed our search for the right headset.

6. Cyber-Physical Architecture

The functional architecture was used to update cyber physical architecture which is shown below in **Figure 4**. The cyber-physical architecture delineates the functions among different subsystems and goes into details of implementation on a higher level. It also explains the decisions taken based on the trade-studies to identify components, algorithms, etc.

Our system has 2 major subsystems (Aerial and User). The aerial subsystem is further divided into 2 key components - the onboard sensor suit and the onboard computer Jetson TX-2. The user subsystem consists of the pilot with the DJI radio controller and the Epson BT-300 Augmented Reality headset running the FlySense interface. Both the subsystems are interlinked by Wi-Fi communication subsystem to enable transfer of data.

Each of these components are described below:

a. Aerial Subsystem
    i. DJI M100 Quadcopter is the platform where all the algorithms are tested.

ii. Velodyne VLP-16 LIDAR gives the raw point cloud data for obstacle detection in 3D and 360°.
iii. FPV camera gives the frontal view of the quadcopter with a field of view of 80°.
iv. The state estimation is carried out using the onboard IMU and GPS.
v. Onboard computer Jetson TX2 is used for the following functions:
    1. Calculate Flight Envelope: The flight envelope is calculated from the received pose estimate and pilot inputs. This is the addressable area around aircraft where aircraft can reach in 5 seconds. This does not include sudden malfunction/crash.
    2. Point Cloud filter pipeline: The raw point cloud is passed through a series of filters that involve cropping, downsampling and outlier removal. The flight envelope calculated is used to extract out only the relevant data and get rid of the extra point cloud data. This is done to reduce required onboard processing.
    3. Obstacle Classification and Coloring: The filtered point cloud is then used to identify the obstacles in the flight path, classify them into different danger levels based on the maximum possible pilot input and time to impact and color them red/yellow/green based on the same.
    4. Sound Warnings: Among the obstacles detected, the obstacle with least time to impact is calculated using Newton's method and used to generate sound warnings.
    5. Bird's eye view: The obstacles detected along with the most dangerous obstacle given by the sound warnings code are combined to generate a bird's eye view image.
    6. The FPV video and BEV combined together and published at one frequency over Wi-Fi to the Epson.
    7. Relevant state information of the quadcopter is broadcast over Wi-Fi.
    8. Override pilot commands: The pilot commands are modified if the most dangerous obstacle is in the immediate path of the quadcopter. The algorithm publishes linear velocity commands for emergency braking.
vi. The onboard flight controller receives velocity commands and takes control of the vehicle to avoid collision.

b. User Subsystem:
    i. The pilot is the heart of our complete system. He provides commands using the DJI Radio Controller and FlySense interface to navigate the quadcopter safely.
    ii. Pilot inputs are part of all the algorithms in the software stack.
    iii. The Epson BT-300 headset is used to render FPV video, BEV and HUD based on the sensor information and video received from the onboard computer.
    iv. The sound warnings are given to the pilot through the headset as beeps.

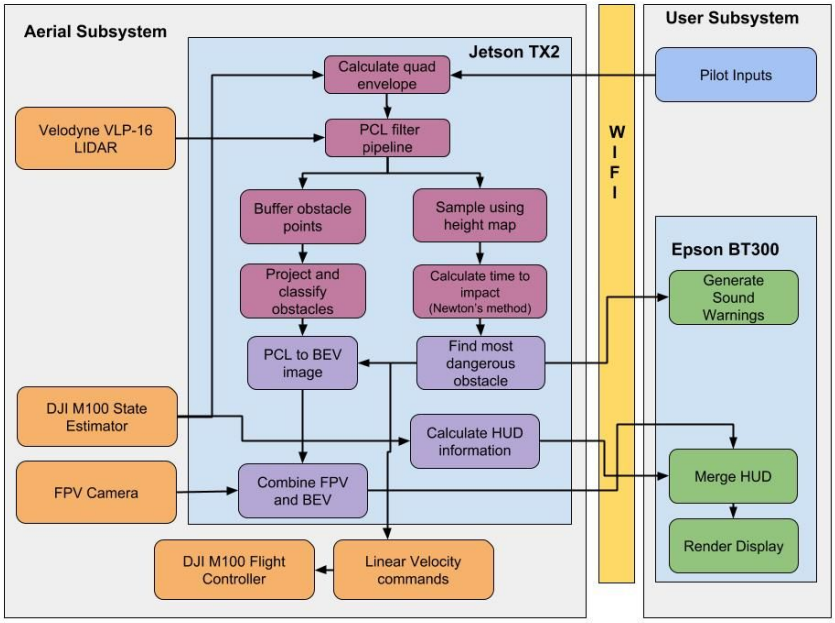


Figure 4: Cyber-physical Architecture

7. System Description and Evaluation

Our final system consists of 3 major subsystems as shown in **Figure 5**. These are:

- Aerial subsystem – A DJI Matrice 100 mounted with Velodyne VLP-16 LIDAR , FPV camera, Jetson TX2 onboard computer and CC BEC 2.0 (power distribution board).
- User subsystem – The Augmented Reality headset Epson BT 300 and another audio headset for sound warnings.
- Communication subsystem (COTS)– Unifi AC-M wifi router and UMA-D Directional antenna. The directional antenna is pointed manually while flying the quad.

Figure 5: Overall System diagram

The design, development and testing conducted for each of these sub-systems is described in the following sections.

7.1. Aerial subsystem

The Aerial subsystem consists of multiple hardware and software components. The hardware components are described in Modelling/analysis and testing section.

The software architecture for the aerial subsystem is shown in **Figure 6.** The software architecture has been developed keeping in mind the testing and safety constraints associated with flying an aerial system. We have setup a way to test the system by feeding in data from a recorded bag file. This allowed us to record the data once in our mission scenario and use it continuously to fix minor bugs which would be difficult to fix if we were to move on to live operation directly. It also helped us as a team, as individual developers were able to test software components with minimum dependencies.

The architecture is shown in **Figure 6** also includes all the software components (ROS nodes) that were created to validate all the system requirements. In essence, we are processing all the information onboard to minimize processing required in Epson. This is necessary as Epson is based on Android and its difficult to carry out certain tasks like image processing.

On a high level, following functions are done in each of these nodes:

1. Preprocessing node: Calculate the flight envelope and filter the received point cloud based on the that. It also publishes the tf tree and all the vehicle state information which to be displayed by the Epson.
2. Bird's eye view generation: In this node, the filtered point cloud is transformed to world frame. These point cloud frames are then buffered to increase the amount of points. The buffered point cloud is transformed back to body frame where the points are converted projected on a 2d plane to form an image. The pixels of the image are colored based on time to impact calculated for each of the 3d points and dangerous obstacle calculated in sound warning node.
3. Sound warnings: This node generates sound warnings based on pilot's control inputs, aircraft states and surrounding obstacles. The sound warnings are directly published to user interface. It also publishes information about most dangerous obstacle which is used in flight control node to override pilot control if necessary.
4. Flight control node: This node allows the option to switch on and off the emergency brake feature. When switched on, it received the dangerous obstacle from sound warnings node and then overrides the pilot inputs to prevent collision.
5. DJI Gazebo interface node: This node receives pose information from DJI SDK and moves the simulated velodyne LIDAR position in gazebo world. This helps us simulate virtual obstacles to test the emergency brake feature.

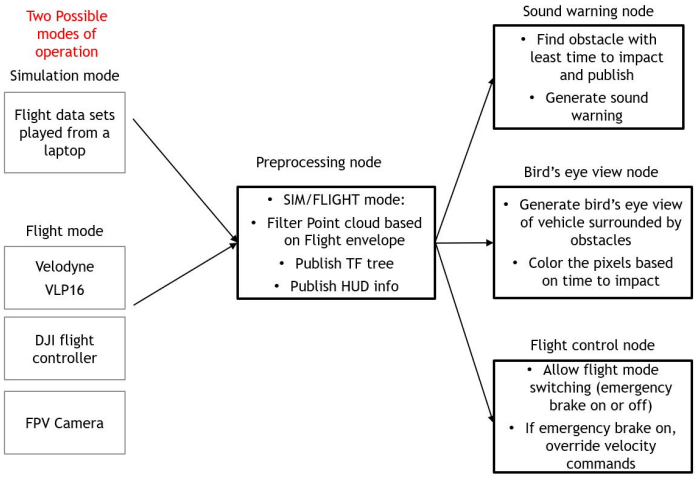


Figure 7: Aerial Subsystem Software Architecture

7.1.1. DJI Interface with custom flight mode for obstacle avoidance

DJI provides Onboard SDK which can be used to communicate with DJI flight controller to receive real-time data and to control the quadcopter. Our DJI interface is built around the ros wrapper of Onboard SDK which allows receiving data as ros topics and publish commands by subscribing to ROS services.

DJI provides information related to state of the aircraft which includes position, attitude, velocity, pilot commands, etc. A separate node was written for emergency brake functionality. This node subscribes to pilot control inputs and based on position of a switch on the radio controller it either enables or disables our custom flight mode. This custom flight mode converts pilot commands to velocity commands for the quadcopter. These velocity commands are constrained if pilot's commands are leading to a collision. Figure 8 shows the basic architecture of this node:

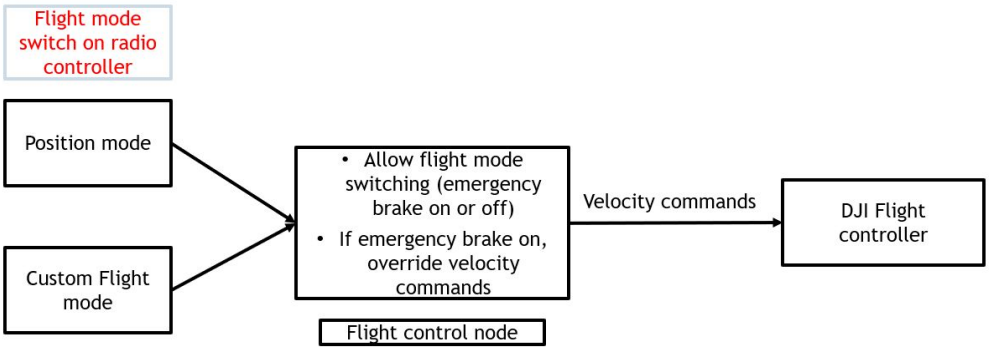


Figure 8: Pilot override

Motion Model for our quad

The obstacle avoidance algorithm is based on a feed-forward model. This model was derived assuming a linear dependency on drag and gravity, with the drag parameters fitted from actual flight data. Assuming a linear system, the position on a specific dimension is given by x_i (where x_i is the ith coordinate of a three-dimensional position state vector):

$$\text{x\_i}[t\_] := t * \text{veqX\_i} - \frac{e^{-\frac{\text{Ax\_i} * t}{m}} m(\text{v0x\_} - \text{veqX\_i})}{\text{Ax\_i}} + \frac{m(-\text{veqX\_i} + \text{v0x\_i})}{\text{Ax\_i}} + \text{x0\_i};$$

The formulas are the same for all axis, but the equilibrium speeds are different.

- In z, gravity plays a role with the zero-input mapped to a non-zero vertical thrust so that the pilot does not have to worry about keeping altitude constant.
- In x and y, the z Euler angle defines rotations that transpose the body frame inputs (left/right or forward/backwards) into the real world.
- The drag coefficients are substantially different across the XY (we assume that attrition is the same for the x and y axis which is approximately true) and z axis (much more drag).

The quad speed, at any given moment, is:

$$\text{vx\_i}[t\_] := (-\text{veqX\_i} + \text{v0x\_i}) e^{-\frac{\text{Ax\_i} t}{m}} + \text{veqX\_i};$$

We performed a calibration flight for our quad where we regressed the drag coefficients (1.2 in xy plane and 5 in the z axis) and validated the model.

Comparison between the real data and the closed form dynamics model:

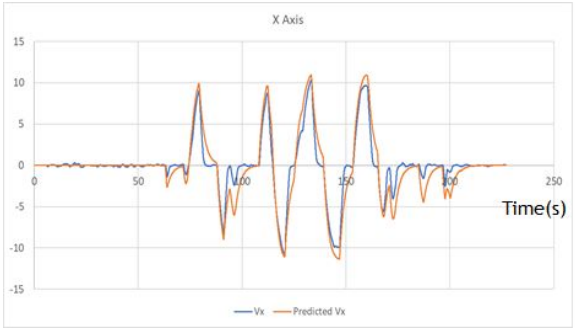


Figure 10: Comparison between predicted and actual velocities using the selected motion model.

Avoidance Control Algorithm used in FlySense

For obstacle avoidance we can easily determine the time to impact from zeroing the speed equation and introducing its value back into the position equation:

$$\Delta X\_i := \frac{m v0X\_i}{AX\_i} + \frac{m\left(veqX\_i * \mathrm{Log}[1 - \frac{v0X\_i}{veqX\_i}]\right)}{AX\_i} * veqX\_i/veqX\_i$$

This equation can be solved for the equilibrium speed that a pilot input should give, that given a non-zero initial speed at a particular point, would stop at the obstacle location. Nonetheless, the equation is transcendent (product log) and Matlab/Mathematica are really bad solving it (C++ is bound to be even worse).

Instead of doing this, we solve for two types of situations that are really simple: the "No Return" range (what is the distance the quad travels starting from a non-zero speed and having full reverse input) and the "Zero Input" range (the distance the quad travels with a zero input when starting from a non-zero initial speed).

We then fit a straight line between the two limit cases, simplifying the inverse dynamics and getting a control that reacts faster than the original function and should be more stable than a non-linear control with a function whose numerical computation is problematic. As a bonus, the straight line can be extended beyond the "Zero Input" as we no longer have a domain problem as we did in the "Product Log". The point where we can give maximum input in the direction of the obstacle and still stop on time is referred to as "Full Control":

With this approach we were able to operate seamlessly in four different regimes:

- Type I: The quad is in a state where the obstacle is further away from the "Full Control" and the pilot can do whatever he pleases
- Type II: the quad is in a state where the obstacle is further than the "Zero Input" range and the algorithm operates in the region of positive thrust inputs (which is a one to mapping to a target steady state speed) that tend gradually to zero as the obstacle approaches
- Type III: the quad is in state further away from the "No return range and the algorithms operates in the region of negative thrust inputs that tend gradually to zero as the object approaches
- Type IV: the quad is in a state closer than the "No return" range and only accepts inputs of "full reverse"

So, in directions where there are no obstacles, the pilot is by default in the "Type 1" regime. The detailed test cases can be found in the Annex 1 of this report. In figure 11:

- Y is the computed range (output in m), X is the equilibrium speed (input in m/s)
- Blue is the "real" product log function and
- Red is the simplified control (please note that this is not a classic linearization about a single point instead it is a simplification that fits across two different points/regimes)

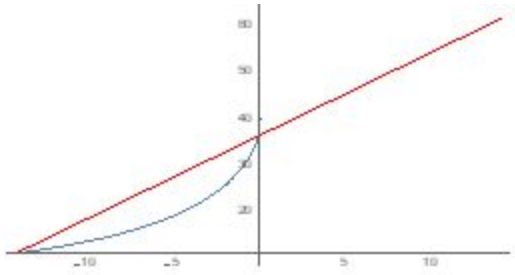


Figure 11: Product log versus "two-points straight line" approach (m=2.9 kg)

7.1.2. Filter point cloud based on flight envelope

Dynamic window

To reduce the processing load on the onboard computer, the point cloud received from the Velodyne LIDAR is filtered based on the flight envelope of the aircraft. We have defined flight envelope as the addressable area surrounding the vehicle where aircraft can reach in 5 seconds.

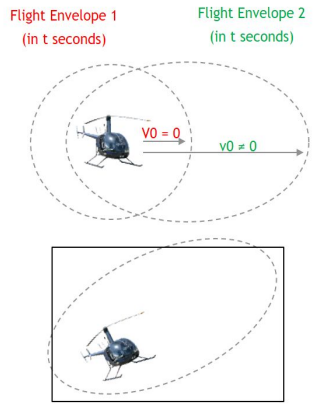


Figure 12: Dynamic flight envelope calculation diagram

The flight envelope at a given time instant is estimated based on the current state of the vehicle and the pilot inputs. Assuming a vehicle starting from rest, the addressable region changes from a circular envelope to an elliptical envelope. Taking into consideration a safety clearance and ease of implementation, the flight envelope has been approximated as a cuboidal envelope surrounding the ellipse. The flight envelope algorithm is depicted in **Figure 12.**

PCL filter implementation

The above algorithm was implemented using a cropbox filter from point cloud library. Another cropbox filter was implemented to filter the points captured from parts of the vehicle frame. **Figure 13** shows raw point cloud and filtered point cloud based on flight envelope.

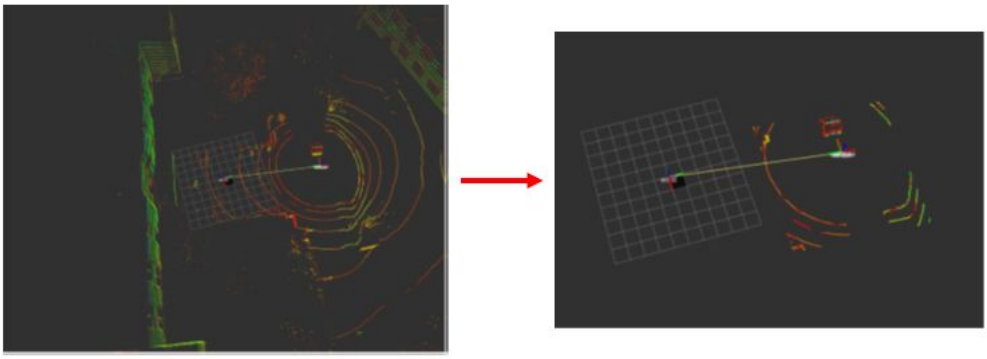


Figure 13: PCL processing before and after filtering

7.1.3. Obstacle Classification, Coloring and Bird's Eye View generation

The output of the PCL filter pipeline is further processed to identify the danger level of the relevant obstacles and color them accordingly. The point cloud is then registered in the world frame, buffered by 3 frames to capture all the relevant information and transformed back to the vehicle frame. By combining the current state of the quadcopter, the maximum possible pilot inputs in all directions and the buffered obstacles in the dynamic window, the space inside the window can be subdivided into ellipsoids of different sizes based on different parameters.

For our purpose, we used time to impact as the metric and used three cutoff time limits to obtain the regions. The obstacles are classified into these three danger levels based on their location inside the ellipsoid regions.

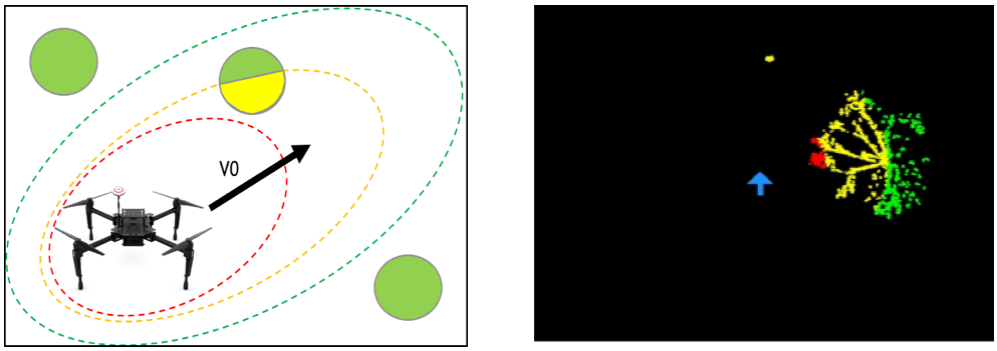


Figure 14: Coloring concept and actual result

The obstacles are then colored Red/Yellow/Green based on their danger levels, as shown in Figure 14a. All the obstacles are in the body frame, and give the pilot complete situational awareness. The obstacles are inflated and then rendered as a top view image to generate the BEV image. In addition, the most dangerous obstacle determined by the sound warnings algorithm flashes as a white dot in the BEV indicating the pilot what is causing the beeps. Figure 14b shows a sample output of the BEV where the arrow indicates the vehicle. The design and color combinations, along with the backend algorithm were all finalized based on the feedback given

by the NEA pilot, David Murphy. A variety of design options were suggested during the pilot workshop, and is briefly outlined below in Table 9.

Icon (Color + Shape)	•Icon options – Triangle, Simple Arrow, Arrow with Tail, Quad sample image •Color options – Blue, Brown	Simple blue arrow (scaled correctly)
Single Color / Multiple Colors	•Single color – ❑ Kd-tree with KNN approach to extract clusters and segment obstacles ❑ Metric – Euclidean distance •Multiple colors with no height influence (2D) •Multiple colors with height influence and opacity (3D)	Multiple colors with no height influence (2D) (inflated to look nicer, factor of safety)
Ground Points	•With ground •Without ground (passthrough filter when altitude <1m)	With ground

Table 9: Art concepts discussed at NEA workshops

The NEA pilot preferred the multi-colored obstacles in 2D as it always indicated a sphere of danger (red) while flying, and did not confuse him with too many colors. Figure 15 shows a comparison between the two options. During flight tests, we have reverted back to the 2D coloring based on pilot input. The 3D was deemed too distracting.

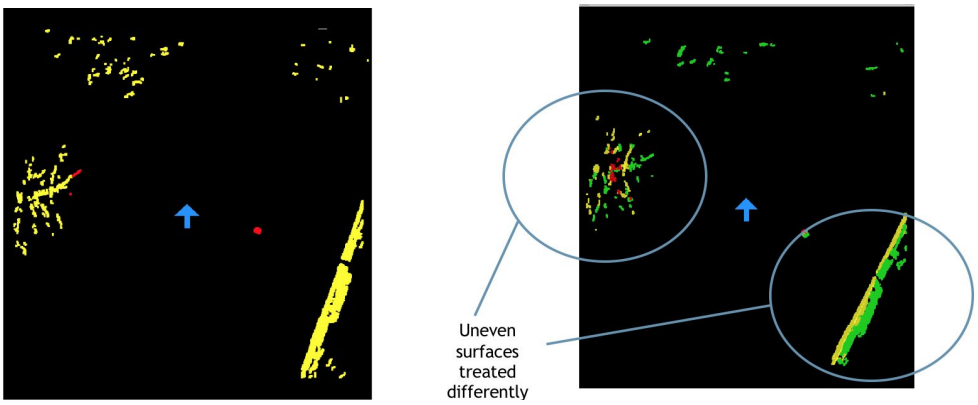


Figure 15: 2D coloring (left) and 3D coloring (right)

In addition, it was decided to keep the ground points as it helped the pilot relate better to the surroundings and understand where the ground was. The BEV would not give good contextualization across near ground or "high altitude flight" if the ground points are not present.

The simple blue arrow contextualizing the quad location and orientation in the Bird's Eye View was introduced to clearly distinguish between what is happening in front and what is happening behind the quad.

7.1.4. Sound warning generation

The output of the PCL filter pipeline is processed parallely in the sound warnings generation node to give the pilot audio feedback, which is the most direct and influential form of feedback for humans. Here, based on the current state of the quadcopter and the current pilot input, the obstacle points are put into a Newton-Raphson iteration cycle to determine the most dangerous obstacle with respect to time to impact.

Since the time to impact calculation for sound warnings was done using the conventional Newton-Raphson method, the sound generation could have been really slow with the heavy processing. To select only the relevant points, the velodyne height map package backend was modified to give obstacle points in 3D. It involved a 2D grid-based approach where all the points in a particular cell are replaced by the centroid and the height is taken as the mean height of all points in the cell.

A grid resolution of 0.2 m was chosen with a total grid size of 300 to give 30m coverage on all sides of the quadcopter. This helped to reduce the number of points from 4000 to around 1000 and is seen clearly in Figure 16(A)

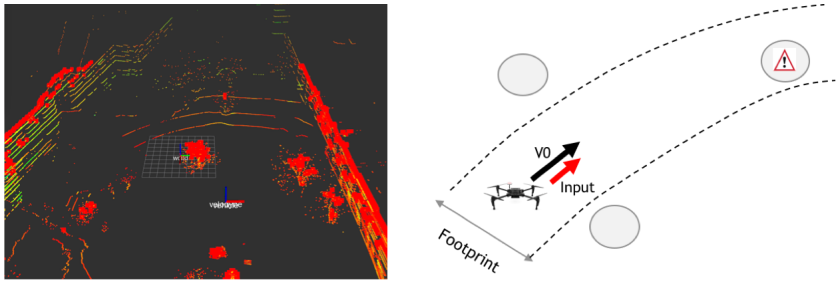


Figure 16: Sound warning generation from point cloud data (left, A) and a diagram of when to generate sound warnings (right, B)

A few important points to note in the sound warnings code are:

- It is processed parallel to the obstacle coloring code, and provides an additional input to perform the blinking in the coloring code
- The time to impact depends on the current pilot input and warnings appear only if the pilot tries to fly towards an obstacle, as shown in Figure 16 (B). The quadcopter is given a footprint padding of 1m on all directions to account for safety.
- The cut off thresholds used for our purpose are 1s, 1.5s and 2s in the decreasing order of danger. Sound warnings are produced if the time to impact is less than or equal to 1.5s. The code was written to have multiple levels of beep frequency depending on the danger level, but for our purpose we found having only one level to be non intrusive.

The equations used to define coloring and sound warnings is described in detail on Annex 3.

7.1.5 Gazebo simulation

In order to test our development of emergency stop functionality, we built a gazebo simulator to emulate our full system. We first created a virtual sensor model of the Velodyne LIDAR and placed this in a environment populated with virtual obstacles to allow us to interact with an environment and generate the same type of data that we receive from the LIDAR in real life. We next wrote a Gazebo C++ plugin to move the Velodyne LIDAR around the virtual environment based on the control inputs from the DJI PC simulator, which emulates the behavior of the DJI M100 in the air and can be controlled using the joysticks.

The plugin reads off the current position, orientation, and velocity values of the quadcopter from the DJI Onboard API and updates the Gazebo model at a specific rate in order to keep the correct physics of the quad maintained in the virtual environment. This way, we could generate a custom flight path in any environment without physically flying the quadcopter.

Using this system, we were able to effectively test our emergency stop code and validate corner cases of the Bird's Eye View interface since we were able to fly in any environment we could create within Gazebo.

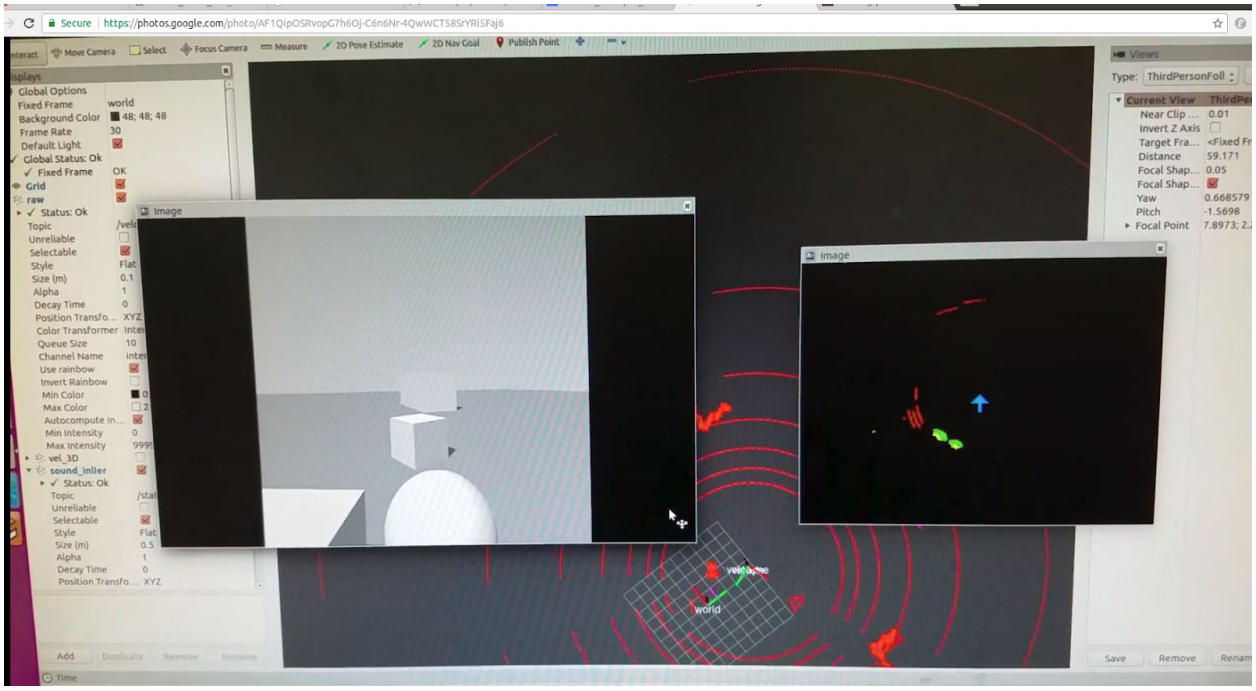


Figure 17: Introducing virtual obstacles with the Gazebo simulator

7.2. User subsystem

The user subsystem consists of the Epson BT 300 AR headset which displays information to the pilot without obstructing his view and an audio headset that provides sound warnings based on imminent danger.

Figure 18: User interface hardware

The user interface consists of three widgets namely the Heads-up display, Bird's eye view and the FPV video overlayed. The three widgets are strategically placed on the same screen such that the pilot can get an immersive experience of the flight without cluttering the field of view.

FPV:

The FPV shows the pilot what the quadcopter is seeing. This is to mimic the flight experience as that of a helicopter where the pilot can only see what is directly ahead of him without any other visual feedback.

Heads-up Display:

The heads-up-display gives only the relevant sensor information to the pilot. The information includes the vehicle's orientation in terms of roll, pitch and heading, ground speed, time to impact to the nearest obstacle.

Bird's Eye View:

The bird's eye view accurately depicts the environment around the vehicle. Bird's eye view is capable of showing the finer details of the surrounding environment like trees, buildings, etc.

Sound warnings:

Based on time to impact and location of the obstacle, sound warnings are generated in either left or right ear. If an obstacle is in left vehicle direction warnings are heard in left ear and if an obstacle is in right of the vehicle direction, warnings are heard in the right ear. This feature has been found to work reliably well on the ground. We will do more tests to check the performance in flight.

7.3. Modeling, analysis, and testing

The aerial subsystem and the user subsystem were tested independently and together.

7.3.1. Aerial subsystem

The aerial system consisted of multiple components mounted on a DJI M100 base kit. A critical goal in the design of the final aerial system was the overall weight. DJI recommends a max

takeoff weight of 3.6kg. With the DJI frame and the Velodyne LIDAR weighing in at nearly 2kg and 800g respectively, that did not leave much room to work with.

We modeled all of this weight in a spreadsheet to get an estimate of what were out fixed weights and where we could get rid of extra weight (see Figure 19). One main source of weight we wanted to eliminate was the long cable and interface box that comes with the Velodyne and houses a fuse and protection circuitry. We did this by splicing the Velodyne cable data lines straight to an ethernet connector. We then put an in-line fuse in series with the power line and attached that directly to our 12V line from our DC-DC power regulator.

Weight Plan

DJI frame (with Propeller fairings)	1924
Matrice 100 Propeller fairings	-200*
Battery	676
Velodyne without box and cable	800
Velodyne Cable	150
Velodyne Signal Box	100
Shave off Velodyne box and cable	-200
4 propellers	19.5
Jetson TX2	82
Heatsink with fan	67
Orbitty Carrier Board	41
Heatsink shaving	-20
LAN cable	8
Power cable + xt60 + fuse	16
Power to Lidar	10
Power to Jetson	10
TTL cable	5
Power module CC BEC	21
Camera	30**
Camera cable	10
Mounting Hardware	5
TOTAL	3554.5

Figure 19: Weight of the quad after deploying all the needed electronics.

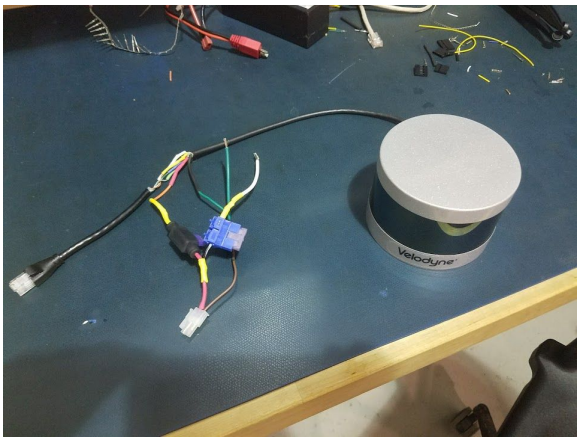


Figure 20: LIDAR weight reduction

In the design of the system, we also needed a very light and robust DC-DC power converter to allow us to power both the Velodyne and the Nvidia Jetson TX2. Power requirements dictated that we needed approximately 2 amps at 12V for the Jetson and 1-2 amps at 12V for the Velodyne. The original power module that we made could convert 24V to 12V with 5 amps output, so that would have worked well, except the existing design was on the order of 120 grams, which was prohibitively heavy. If we were to redesign the board to use light components, it would have been expensive and difficult to assemble without very a very good reflow setup, so we looked for commercial solutions. We found a Castle Creations programmable DC-DC converter from 24V to 12V that could support up to 10A of current draw, and was specially designed for use in drones. We used this to power both the LIDAR and the Jetson.

In order to test if quad would support the required weight, we ran a series of preliminary tests with weights added to the quad. We progressively added 200g weights and flew in roll, pitch, and yaw directions to make sure that the quad was stable before progressing to more weight. Once we were satisfied that the quad was safe to fly at the projected takeoff weight, we added all the designed components.

Testing obstacle avoidance in flight and simulation

After testing the algorithm in the Matlab simulation, we acquired the DJI PC simulator and integrated it with the Gazebo simulator to introduce obstacles in the environment. We did extensive testing of the algorithm. The flight dynamics model in the simulator was slightly different from reality (e.g. we could not change the mass of the vehicle), but this exercise was valuable to ensure the coordinate conventions were well converted across each of the code blocks. The algorithm inputs and outputs were converted to follow body frame conventions instead of world frame conventions. This helped us in linking the code seamlessly with Sound Warning node which provides the most dangerous obstacle at any given point.

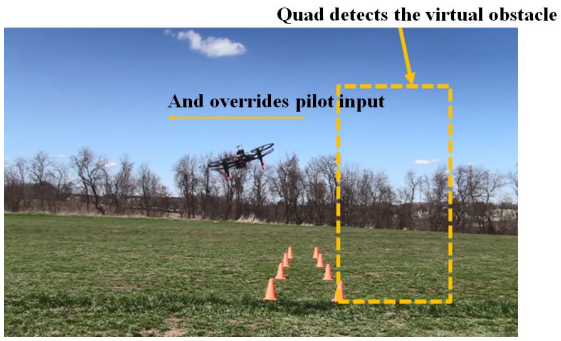


Figure 21: Testing obstacle avoidance with the introduction of virtual obstacles

Once the basic conventions were corrected and brake functionality was working, we started tuning the parameters to get stable braking behavior. We also tested the flight mode switch on radio controller to ensure the emergency brake feature can be switched off at any point if

required. After some testing, the brake functionality was working well in simulation and gave us the confidence to test in real world.

For testing in real world we had configured Gazebo simulator to spawn a Huge wall 20m in east direction from wherever the quadcopter takes off. The aircraft was flown towards the wall and braking behavior was seen. Initially the braking behavior was a little unstable and was therefore tuned on the field. After some further testing, emergency braking was found to work reliably. Figure 21 shows emergency brake functionality being tested in the field. The quad stops 1m before the virtual wall.

Testing coloring and sound warnings with ROS Bags and simulation

To test the coloring and sound warnings we have used ROS bag files of actual flights and also connected the code to the Gazebo and DJI simulators.

Coloring tests

The first set of tests on the coloring were simply checking if the color ellipsoids were consistent across time: they moved with the quad, the color ordering was correct and the shape was extending when the quad was travelling in one direction and the pilot input was constant. The opposite occurred when the input was in opposition to the motion. This part was the most difficult one, because it implied standardizing all the conventions across all the modules in the code (e.g. using body coordinates vs world frame coordinates, FLU instead of ENU, etc)

The second set of tests were related to checking the color cutoff areas which was done by measuring the distance to the obstacle and computing the time to impact. This was done by printing the algorithm inputs on the screen and manually introducing the data fed to the model in an Excel file that was created for this purpose. This Excel file would compute a small trajectory and the time to impact, that was compared with what the c++ code was outputting.

Sound tests

The first set of tests was done by opening RVIZ together with the Bird's Eye View and tracking the white dot that marked the point generating the sound warnings. The introduction of this feature was a pilot requirement (David Murphy said, "I need to know which obstacle is in my path given that the coloring only gives insight on what I might do, not what I am trying to do") and was also extremely useful for debugging. When we got the conventions correct, we could easily see if the projected point of collision "seemed correct" (e.g. an input to the right cannot generate a prediction of collision to the left or any other random direction).

The second set of tests was done in the same fashion as the coloring. We printed the inputs in the screen and introduced those values in an Excel file that was created for the effect. We then compared the values to check that the code was processing correctly those inputs.

The backend of the complete interface is shown in Figure 22, with the left image showing the BEV and terminal output for the sound warnings and the right image showing the simulated data and sampled point cloud in RViz.

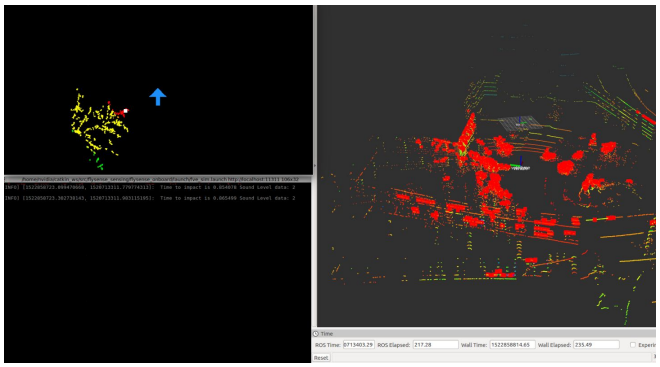


Figure 22: Generating the appropriate coloring for the user interface

The complete interface as seen by the pilot during one of the most dangerous flight test scenarios is shown in Figure 23. The pilot relies only on FlySense with no line-of-sight with the quadcopter and lands it exactly at the middle of two containers.

Figure 23: (A) HUD and Bird's Eye View as seen by the Pilot (B) Landing in between two containers

7.3.2. User system

A brief description of the major tests conducted during the SVE is given below:

Test to evaluate the HUD- The pilot sees the telemetry data coming from the vehicle. The pilot takes off flight and can see the system changing roll, pitch, heading, and the values being updated on the HUD.

Result- The telemetry information was updated on the HUD mode at a refresh rate of 10Hz

Test to evaluate the Bird's Eye View- The pilot takes off flight and sees the quadcopter marked as an arrow and only the obstacles around the addressable region. The operator follows a a flight sequence containing obstacles. The vehicle starts at a location greater than 5.5 seconds to impact away from every obstacle, and no warnings are heard. As the vehicle approaches the first obstacle, the sound warnings start when time to impact is less than 5.5 seconds. As the vehicle moves closer, the time to impact decreases and warnings become more frequent. Once the vehicle crosses the obstacle, the warnings stop. The same scenario happens when the quadcopter encounters this obstacle throughout its course.

Result- All the requirements in this test were successfully completed. The images were rendered at 10Hz, which proved very natural and real time to the pilot. The pilot was able to clearly distinguish through sound the location of the obstacle, and get a better understanding of the environment through the visuals.

7.3.3. Flight test summary, learnings and progress

The flight testing covered various phases of the development cycle: initial testing and characterization, unit testing of subsystems, then final integrated system testing. We flew for a total of ~4.5 hours over the course of our development, with the detailed flight log presented in Annex 2.

8. Spring Validation Experiment Evaluation

The spring validation experiment was performed at Nardo which essentially consisted of 2 tests each with different objectives.

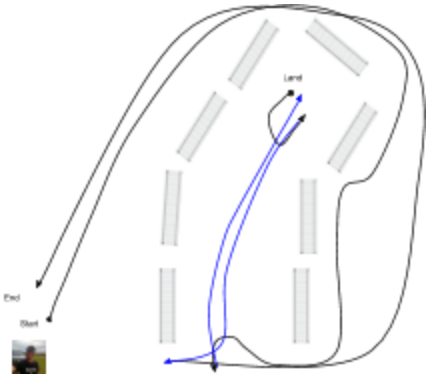


Figure 24: Nardo test plan

Item	Description
Objective	• Test Obstacle Avoidance of our Flysense system • Test Obstacle detection on Bird's eye view, sound warning generation, HUD display and FPV video
Equipment	• DJI Matrice 100 with FlySense Hardware

	• Epson BT300 AR Headset / Android tablet
Procedure	1. Pilot tries to fly the quadcopter into a virtual wall 2. Pilot will fly quadcopter along route, first around the containers **forward** 3. At opening of container enclosure, pilot yaws the vehicle around so the front (FPV side) is facing away from the back 4. Pilot navigates **backward** to back of enclosed area, and lands 5. Pilot takes off, turns 180 deg around, then flies **backward** out of enclosure 6. Pilot navigates vehicle **forwards** around the enclosure back to the start 7. Pilot navigating exclusively by FPV and BEV
Result	1. The quadcopter doesn't let the pilot fly **closer than 1 meter** to the virtual obstacle 2. The pilot was able to observe the HUD display updating values and horizon without any lag i.e, **at 10hz** 3. The pilot can visualize immediate obstacles around him based on the flight envelope and obstacles **less than 1 meter** away on the Bird's eye view. 4. Pilot uses FPV video to know where the quadcopter which technically is analogous to the pilot being on the helicopter. The FPV is being **updated at 10hz**. 5. The pilot can hear sound warnings based on the distance/time to impact between the nearest obstacle in case it is **less than 1 meter or 5 seconds**. 6. The pilot flies the quadcopter exclusively based on the feedback from the flysense system and lands it successfully in space constrained zones.

Table 10: Description of the test plan

9. Strong and Weak points of the system

The biggest overall strength of the FlySense system is the user system. All our system tests were performed with a professional pilot, our ultimate end user, and he was extremely happy with the FlySense interface. Quoting David Murphy from NEA, "The FlySense system was easy to use. I moved through the test card quickly in difficult windy conditions. I would not have even attempted to fly FPV without the FlySense technology. It was fun to use and I was eager to try it in many different applications."

Moving on to the strong technical points of the user interface, the images are very crisp and projected 20m in front of the eye. The visual maps are non-intrusive and provide a clear picture of surroundings. The images are streamed with a latency of less than 1s making it possible to use FlySense interface standalone while operating an aerial vehicle. The sound warnings are heard only when needed, and the flashing dot helps the pilot understand where the danger is. The red areas help the pilot be more careful while flying. The AR headset was lightweight and comfortable to wear for long periods of time.

The strong points in the aerial system are that it is able to capture all the relevant obstacle information considering the hardware limitations. The onboard flight controller is robust and

responds to RC commands with no latency. The localization of the quadcopter is very accurate due to the onboard RTK GPS.

There are a few weak points in both the user and aerial system. The limitations of Android development environment make it difficult to improve the rolling tapes in the HUD. The AR headset is affected by lighting making it difficult to see what is projected on the screen, due to which we had to tape the glasses. Another minor limitation is the lack of complete 3D perspective view, which is more of a nice-to-have.

The users of our system during the demo were seen to rotate their head and see how the FPV video changes. Since our quadcopter had only one camera, it was not possible to provide anything more than a frontal view. The aerial system had a few hardware limitations mainly due to the weight constraints of our vehicle. The LIDAR being 16 beam misses out on very thin but lethal objects like cables and transmission wires. This would require the use of camera and computer vision algorithms for detection and conveying the same to a pilot.

10.Project Management

10.1. Schedule

Over the course of the semester, we had mixed success with adhering to our development schedule. Many of the preliminary tests were done within a week of the intended deadline, but the FPV test and the stop functionality test were significantly delayed. This was due to a delay in obtaining hardware to integrate and limited support for doing additional tests in the case of the FPV functionality and delays in generating an accurate model in simulation for the stop functionality. Both timelines could have been improved with a better balance of resources.

The following schedule describes the test plan and development timeline for each subsystem component of the FlySense project.

Test	ID	Architecture	Planned	Completed
T01	Sub-system	Quad Flights with Weights	10-Feb	9-Feb
T02	Component	Power Module Test	12-Feb	12-Feb
T03	Component	Communications Test	10-Feb	22-Feb
T04	Sub-system	Log flight data to tune flight dynamics	13-Feb	18-Feb
T05	Sub-system	Flight with Velodyne and Jetson onboard	20-Feb	27-Feb
T06	Component	Test FPV Camera	14-Feb	10-Mar
T07	Sub-system	Flight with Velodyne, Jetson and FPV camera	28-Feb	18-Mar
T08	Sub-system	Flight to Evaluate Birds Eye View, sound warnings	7-Mar	11-Mar
T09	Sub-system	Flight to evaluate “Stop” functionality	21-Mar	23-Apr
T10	Sub-system	Flight with new video feed merging FPV and user widgets	14-Mar	6-Apr
T11	System	SVE Dry run 1 (Flagstaff Hill)	4-Apr	18-Mar
T12	System	SVE Dry Run 2 (Nardo)	6-Apr	6-Apr
T13	System	Mini SVE (Nardo)	26-Apr	26-Apr

Table 11: Overall Test plan

10.4. Budget

Budget Category	Amount
Communications	$237.82
Power Hardware	$169.93
Onboard computer	$475.00
Test Infrastructure	$828.03
Mech Hardware	$205.94
DJI hardware	$198.52
Sensing Hardware	$50.42
Data Storage	$83.98
User Interface	$799.00
Total	**$3647..64**

Table 12: Categorized Budget

In total, we spent approximately 75% of our total available budget. We were able to get a significant number of components sponsored, namely the big-ticket items of a Velodyne VLP-16 LIDAR ($9000) and DJI Matrice 100 Drone ($3000).

The overall budgeting process worked well, we were never in any danger of going over our allotted budget once we secured the donations of the most expensive items. We kept a close eye on how much we were spending to always stay within the allotted amount.

10.5. Risk Management

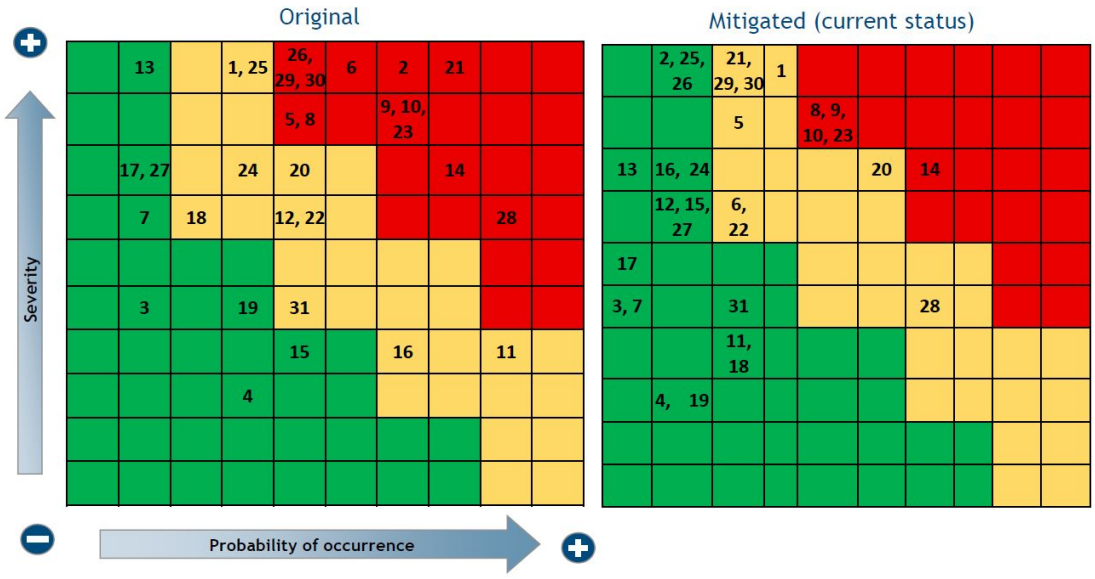


Figure 25: Risk diagram for the Spring Semester

Figure 25 shows the risks that we were tracking throughout the Spring Semester. Some notes on some major risks and their mitigation strategies:

- (14) related to the weight of the drone being too heavy for flight. This was a significant problem at the beginning of the Spring semester, with us projected to be 300 grams overweight, but through weight cutting measures discussed in previous sections, we were able to keep the weight at the maximum recommended of 3.6 kg. This was still a significant risk, since if we had to make any modifications to the hardware, the weight could go up, and potentially put the quad in danger, but we kept a close eye on the dynamic behavior to characterize the drone and limited any hardware changes in order to limit the risk of this problem.
- (21) was a risk associated with the WiFi communication. In the Fall semester, we had significant problems with our communication system, making it a huge risk for us. We purchased all new hardware to limit the occurrence of the problem, but we were still limited in the range of the antenna, so any long distance testing could trigger problems. We adjusted our flight plans accordingly.
- Weather (20) was a significant risk that we could not do much about. We limited the impact of this by planning ahead and testing as much as possible while we could, though we still got behind in our test plan over the course of the semester when work schedule did not match well with the weather schedule.
- (9) was related to availability of a testing ground. We did a significant amount of testing in Schenley Park, but people in the past had been hassled by park staff for flying. This did not occur to us until the day of SVE, but we had a backup flight area of NREC (through Dimi) that we could access if Schenley Park became unavailable to us.
- (10) related to the processing of FPV frames to keep the system real-time. This was mostly dependant on the WiFi connection (addressed earlier) and the amount of data passed through. We limited the amount of data as much as possible and had plans to downsample the number of buffered frames if the frame rate became a significant problem.

11. Conclusions

The team was able to develop a robust flight system which allowed quick deployment and testing. As mentioned above, the FlySense system was validated during SVE, SVE Encore and couple of times during flight tests at NEA Flight Test Facility at Nardo. The rigorous flight testing allowed the team to refine the system and ensure that it works properly in real world setting. The system successfully passed all the stated requirements.

The assistance features of the system were also appreciated by NEA Pilot who got really used to trusting the system when it came to flying non line of sight with obstacles around. On numerous

occasions pilot just had to trust the bird's eye view and sound warnings as obstacle was not even visible in the FPV stream, especially when flying backwards.

The team sees following possible improvements to the system:

- Continuous obstacle avoidance based on vector field potential.
- Point cloud registration to improve bird's eye view image.
- Use depth cameras or rotating hokuyo to replace Velodyne LIDAR.

12. Lessons Learned

- Testing a system is much more demanding than testing a single sub-system (e.g. network)
- Designing for a human is substantially different from designing for a robot (e.g. mapping)
- Sometimes the simplest possible solution works well (e.g. direct from LIDAR)
- Time is an extremely scarce resource that needs to be well managed from the beginning
- Cross-functional tasks need to be planned as early as possible to ensure work bandwidth
- Requirement ownership is crucial for success (demand vs "sell them to someone else")

13. References

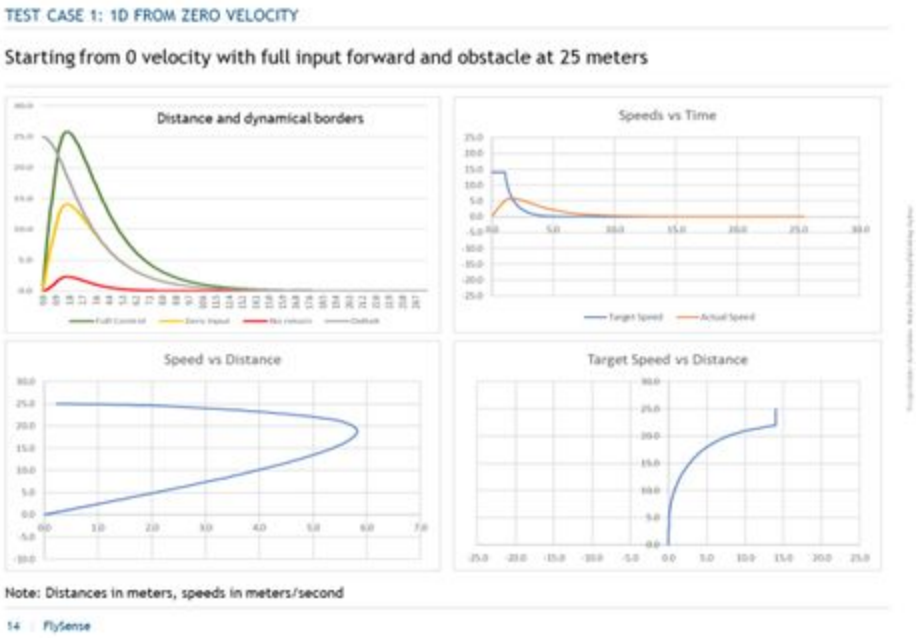
[1] Hornung, Armin, et al. "OctoMap: An efficient probabilistic 3D mapping framework based on octrees." *Autonomous Robots* 34.3 (2013): 189-206.
[2] Asvadi, Alireza, et al. "3D Lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes." *Robotics and Autonomous Systems* 83 (2016): 299-311.
[3] Luukkonen, Teppo. "Modelling and control of quadcopter." *Independent research project in applied mathematics, Espoo* (2011).
[4] wiki.ros.org/
[5] pointclouds.org/
[6] medscape.com
[7] nvidia.com
[8] dji.com
[9] velodyne.com

Annex 1: Obstacle avoidance test cases

Below are some of the test cases ran on the Matlab code used for obstacle avoidance.

1) We started with a very simple case with the vessel steading still, then initiating a 1D movement towards a stationary obstacle situated 25 meters away.

- The first graph shows the “Full Control” range, the “Zero Input” range and the “No Return” range
- The second graphic shows the actual speed and the target speed (this has a one to one mapping to the thrust inputs and corresponds to a steady state)
- The third graphic shows the actual speed vs the distance to the obstacle (please note that the algorithm first allows full acceleration before constraining the pilot)
- The last graphic shows the target speed vs the distance to the obstacle (please notice that all inputs are on the “positive” thrust side



2) Cases where the quad starts with non-zero initial speed (1D)

- Below the case where the algorithm could not prevent the collision (e.g. a moving obstacle that just crossed the path). It needed minimum 10.4 meters to stop but only had 8 meters available before crashing.

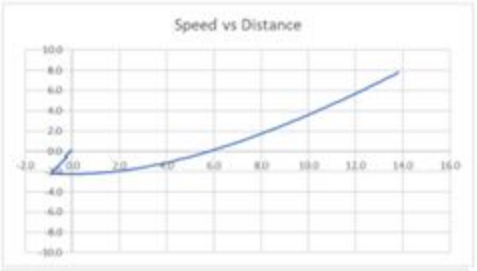
TEST CASE 2A: 1D WITH DIFFERENT TYPES OF DYNAMICS

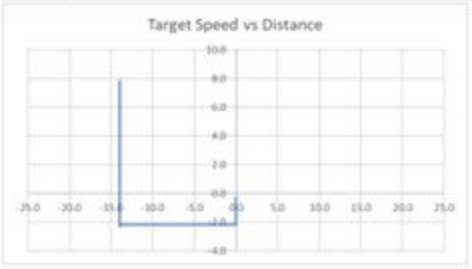
Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s

- Obstacle at 8 meters (impossible to stop)

X_Full Control	Target Vx*
57.3	14.0
X_Zero_input	Target Vx*
33.8	0
X_No_Return	Target Vx*
10.4	-14.0





Note: Distances in meters, speeds in meters/second

- Below the intermediate case where only negative thrust inputs can be issued

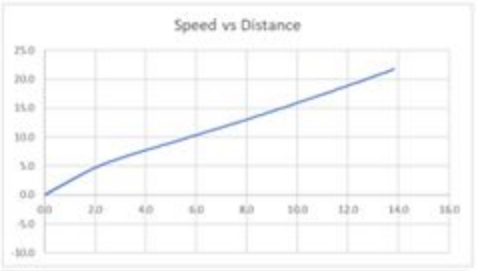
TEST CASE 2B: 1D WITH DIFFERENT TYPES OF DYNAMICS

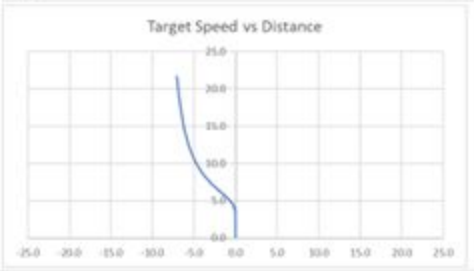
Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s

- Obstacle at 22 meters (stop possible using reverse)

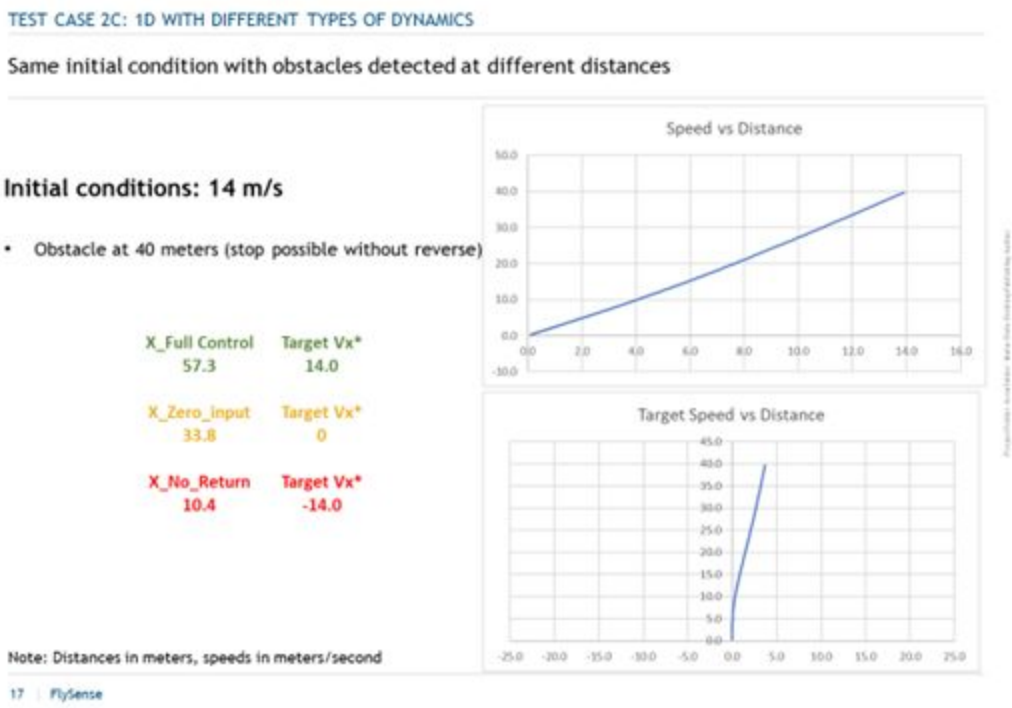
X_Full Control	Target Vx*
57.3	14.0
X_Zero_input	Target Vx*
33.8	0
X_No_Return	Target Vx*
10.4	-14.0





Note: Distances in meters, speeds in meters/second

- The case where the quad can prevent collision with positive thrust inputs

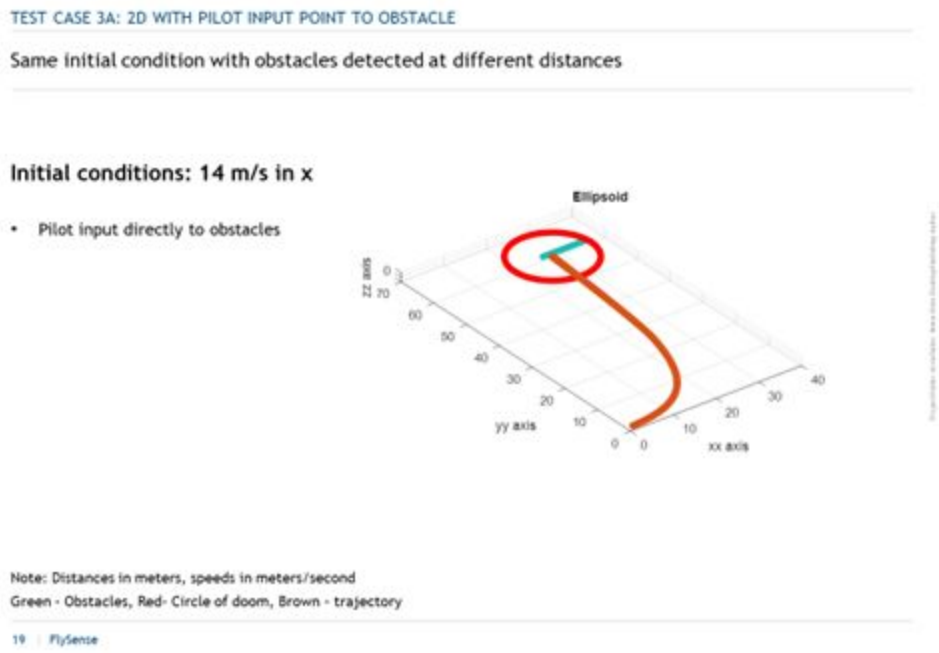


Note: on the code implemented in the quad we defined a minimum clearance distance so that the quad would not stop when physical contact was achieved with the obstacles, but slightly before.

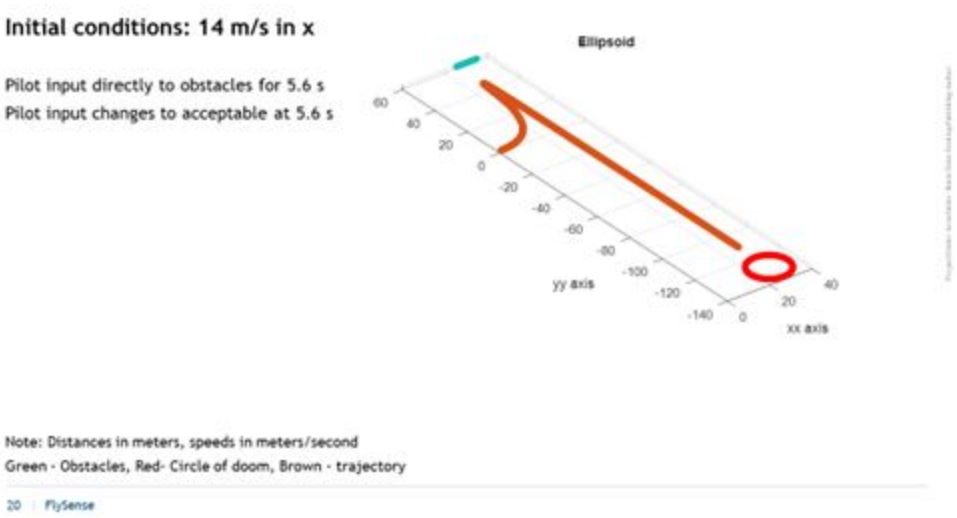
- The case where the quad can do any input for the first leg of the flight and gets gradually constrained as it approaches the obstacle

3) Tests in 2D

- Flight is initiated with a speed in X and the pilot gives a continuous input in Y in the direction of the obstacles. The algorithm successfully stopped the quad 1 meter before the obstacle (clearance distance defined in the simulation).



- Same as the previous case, but at 5.6 seconds the pilot changes its mind and introduces a continuous input in the Y direction away from the obstacle



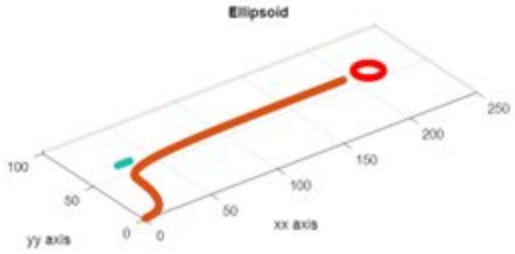
- Same as the first 2D case, but the pilot input is changed at 5.6 seconds to an input in the X axis

TEST CASE 3B: 2D WITH PILOT INPUT POINT TO OBSTACLE

Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s in x

- Pilot input directly to obstacles for 5.6 s
- Pilot input changes to acceptable at 5.6 s



Note: Distances in meters, speeds in meters/second
Green - Obstacles, Red- Circle of doom, Brown - trajectory

4) The last set of tests were done in 3D

- Pilot input is done continuously in the direction of the obstacle

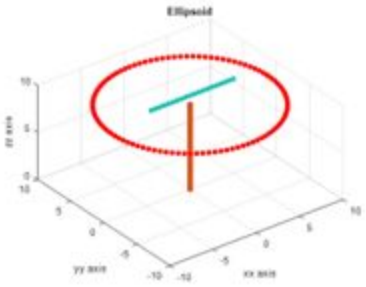
TEST CASE 4: 3D WITHOUT PILOT INPUT POINT TO OBSTACLE

Same initial condition with obstacles detected at different distances

Initial conditions: 4 m/s

- Obstacle at 10 meters (full throttle possible)
- Pilot input towards obstacle

X_Full Control 3.9	Target Vz* 4.0
X_Zero_input 2.3	Target Vz* 0
X_No_Return 0.7	Target Vz* -4.0



Note: Distances in meters, speeds in meters/second
Green - Obstacles, Red- Circle of doom, Brown - trajectory

- Same as the previous 3D case, but the pilot input at 12 seconds is changed to a side input AND continues to introduce an input in the direction of the obstacle

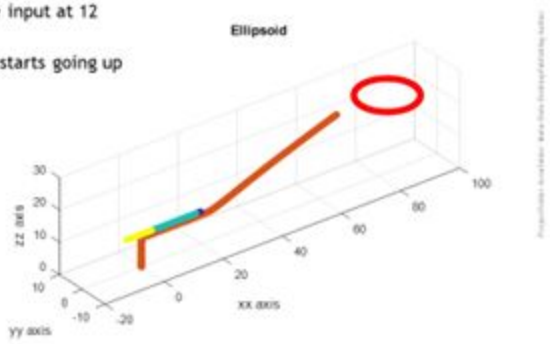
TEST CASE 4: 3D WITHOUT PILOT INPUT POINT TO OBSTACLE

Same initial condition with obstacles detected at different distances

Initial conditions: 4 m/s

- Obstacle at 10 meters (full throttle possible)
- Pilot input towards obstacle changed to acceptable input at 12 seconds to go sideways keeping the input up
- The algorithm "remembers" the obstacle and only starts going up after the security distance is achieved

X_Full Control	Target Vz*
3.9	4.0
X_Zero_Input	**Target Vz***
2.3	0
X_No_Return	**Target Vz***
0.7	-4.0



Note: Distances in meters, speeds in meters/second
Green - Obstacles, Red- Circle of doom, Brown - trajectory

Annex 2: Detailed flight log

Flight #	Date	Location	Flight Time	Summary
1	1/31	NSH	5	Quad pushed by wind
2	1/31	NSH	4	Quad pushed by wind, otherwise stable
3	1/31	NSH	5	Fairings removed, drift in position. IMU cal needed
4	2/2	NSH	8	3050g takeoff weight, position hold ok
5	2/2	NSH	14	3450 takeoff weight, position hold ok
6	2/9	Schenley Park	4	IMU error when landing, 3650g takeoff weight
7	2/9	Schenley Park	7.5	good agility, IMU error, 3650g takeoff weight
8	2/18	Schenley Park	5	Flight check
9	2/18	Schenley Park	5	vertical accel, full yaw, backward at max speed
10	2/18	Schenley Park	5	forward and back at max speed
11	2/22	Schenley Park	8	dynamics tested, data recorded, takeoff weight 3.6kg
12	2/22	Schenley Park	10	communications test, problems at higher alt, fine otherwise
13	2/27	Schenley Park	4	First flight with Velodyne, Jetson both onboard
14	3/4	Schenley Park	9	DJI, velodyne data recorded. Flew near obstacles
15	3/11	NSH	9	DJI, velodyne data recorded. Flew near obstacles
16	3/11	Cut	1	DJI, velodyne data recorded. Batt. temp warning
17	3/18	Schenley Park	8	DJI, velodyne, FPV data logged. Flew near obstacles
18	4/6	Nardo Airfield	10	Familiarization Flight, some flying near obstacles
19	4/6	Nardo Airfield	10	Flying near obstacles head in, windy
20	4/6	Nardo Airfield	10	Flying near obstacles backwards, windy
21	4/6	Nardo Airfield	10	Flying near obstacles backwards, windy
22	4/6	Nardo Airfield	10	Flying near obstacles, windy. Stayed low to the ground
23	4/6	Nardo Airfield	5	Flying near obstacles, windy. Stayed low to the ground
24	4/23	Cut	5	Evaluated stop. Quad very violent. Magnetometer error
25	4/23	Schenley Park	5	Evaluated stop. Quad very violent at higher speeds
26	4/25	Schenley Park	5	Testing pre-SVE. Stopped by park ranger. Stop functionality working
26	4/26	Nardo Airfield	5	latency in FPV. Pilot asked for coloring change
27	4/26	Nardo Airfield	10	new coloring, sideways inside container area, then back
28	4/26	Nardo Airfield	10	sideways + backwards flying near obstacles btwn containers

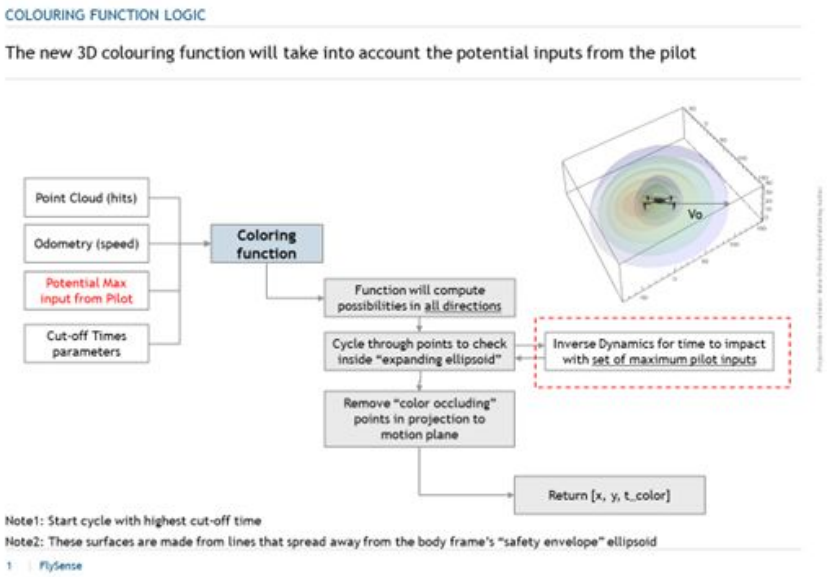
29	4/26	Nardo Airfield	10	takeoff inside container area and fly out
30	4/26	Nardo Airfield	10	backwards in, turn around, backwards out
31	4/26	Nardo Airfield	2	takeoff, distance limit triggered
32	4/26	Nardo Airfield	10	takeoff, flew to bee hives, come back
33	4/26	Nardo Airfield	10	flew around building, into containers, out around obstacle
34	4/26	Nardo Airfield	5	stop functionality demonstrated
35	5/2	NREC	5	stop functionality demonstrated
36	5/2	NREC	10	BEV functionality demonstrated

Annex 3: Bird's Eye view sound and coloring formulas

Coloring Algorithm

The coloring algorithm code works as follows:

- The collision surface is an ellipsoid that will expand across time based on the maximum quad dynamics (initial conditions and maximum pilot input in any direction)
- Frame of reference for function inputs will have
    - All speeds and coordinate inputs are in the body frame of the quadcopter
    - X axis pointing forward and aligned with Velodyne reference axis



Three cut-off times are selected (red, yellow and green). For each point is tested sequentially if the point can fit inside the ellipsoid defined by:

$$Test = \frac{1}{Tzmax^2+Txymax^2}\left(\frac{x1^2}{x2^2}+\frac{y1^2}{y2^2}+\frac{z1^2}{z2^2}\right)$$

Where:

z1 = (Az*Vz0+m*g)*(exp(Az_m*t)-1)+exp(Az_m*t)*(-Az*g*t+Az_m*Az*(z0-z0bs);
y1 = (Axy*Vy0+0)*(exp(Axy_m*t)-1)+exp(Axy_m*t)*(-0+Axy_m*Axy*(y0-yObs);
x1 = (Axy*Vx0+0)*(exp(Axy_m*t)-1)+exp(Axy_m*t)*(-0+Axy_m*Axy*(x0-xObs));

z2 = exp(Az_m*t)*(1-Az_m*t)-1;
y2 = exp(Axy_m*t)*(1-Axy_m*t)-1;
x2 = exp(Axy_m*t)*(1-Axy_m*t)-1;

Note: As per the input from NEA pilots, we removed the z component to avoid coloring an obstacle with too many colors (e.g. tree branches have multiple heights).

Whenever "Test" is returns a value between zero and 1, the object is inside the ellipsoid built with the selected cut-off time to impact. These ellipsoids are computed taking into account the current state and the maximum potential pilot input. For efficiency, we start allocating from the green area so that we do not need to run through a green point twice. From there, what fails goes through the yellow cycle and what is left is assigned by definition to red.

Sound Algorithm

The sound algorithm code works as follows:

- The collision surface is the body frame ellipsoid translated without any deformation across the projected trajectory (initial conditions and current pilot input)
- The output from this function is only the most "dangerous" obstacle, corresponding to the point that has the shortest positive collision time based on current pilot input
- The output of this function is reused an input in for the obstacle avoidance function
- Frame of reference for function inputs:
    - All speeds and coordinate inputs in the body frame of the quadcopter
    - X axis pointing forward and aligned with Velodyne reference axis

For each axis, the Newton method is used to search for a collision time between 0 and 30 seconds. Any collision time outside this interval is discarded. Same whenever the number of loops exceeds a pre specified threshold:

Increment (x axis):

```
xTest=xObs-(x0+1/Axy_m*(Vx0-VeqX)*(1-exp(-Axy_m*t))+VeqX*t);

dxTestdt=exp(-Axy_m*t)*(Vx0-VeqX)+VeqX;
```

Update (x axis):

```
F=xTest;dFdt=dxTestdt;

t=t+F./dFdt;
```

The process is done successively first for the x axis, followed by the y axis and the z axis. To remove false positives (when a collision time is found on one of the axis but does not generated collision when the other axis are taken into account), the projected collision time is substituted into the motion equations. If the distance between that point and the obstacle is below the clearance distance, it is considered a valid positive. If not, it is considered a false positive.