Fly Sense

Team C – ILR09

21st March 2018

Joao Fonseca Reis

Shivang Bhaveja

Harikrishnan Suresh

Nick Crispie

Sai Nihar Tadichetty

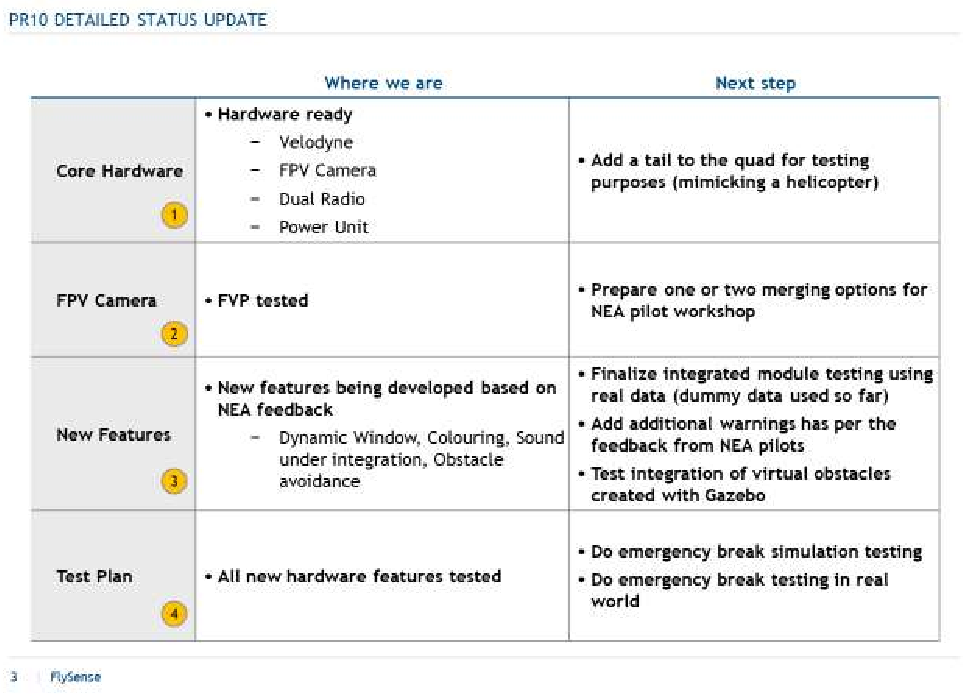
Work done these past 2 weeks

Over the past two weeks we have

a) Done multiple flights (dynamics calibration with different weigths, FVP video testin)
b) Re-written the code for the sound warnings (to be the input for the obstacle avoidance)
c) Integrated the code for coloring obstacles (parameters being fine-tuned)
d) Written the code for obstacle avoidance in Matlab (push done over the spring break).

Our hardware setup is final apart from the pending tail (which we are still debating internally whether to do a virtual or a real tail). The trade-off is between safety and "making it as real and tangible as possible".

The final component was the FPV video that was successfully deployed last weekend.



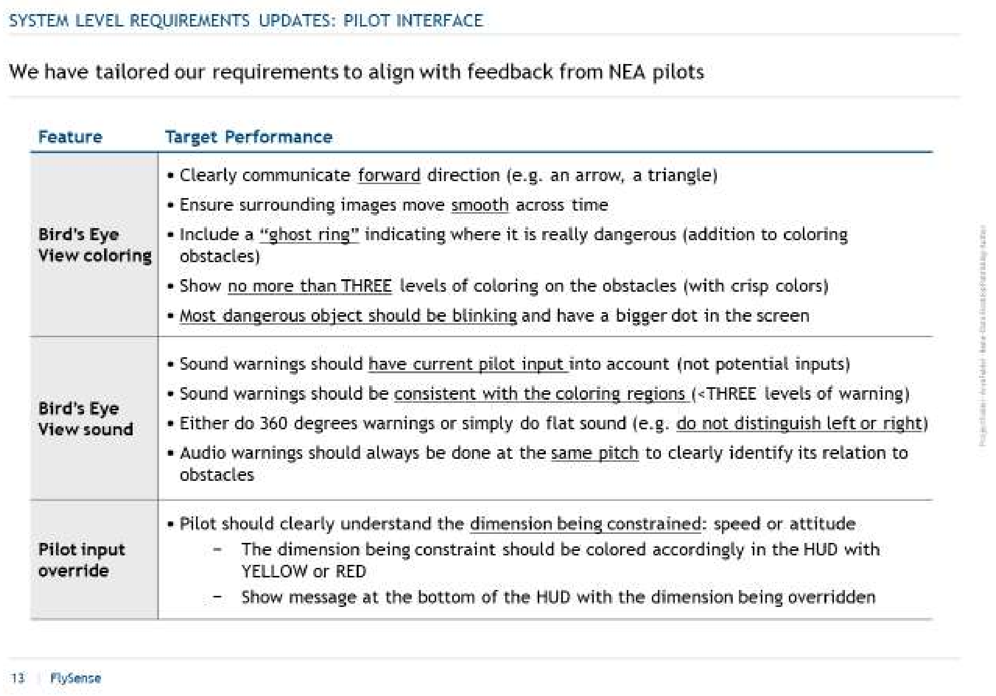
PR10 DETAILED STATUS UPDATE

	Where we are	Next step
Core Hardware 1	• Hardware ready – Velodyne – FPV Camera – Dual Radio – Power Unit	• Add a tail to the quad for testing purposes (mimicking a helicopter)
FPV Camera 2	• FVP tested	• Prepare one or two merging options for NEA pilot workshop
New Features 3	• New features being developed based on NEA feedback – Dynamic Window, Colouring, Sound under integration, Obstacle avoidance	• Finalize integrated module testing using real data (dummy data used so far) • Add additional warnings has per the feedback from NEA pilots • Test integration of virtual obstacles created with Gazebo
Test Plan 4	• All new hardware features tested	• Do emergency break simulation testing • Do emergency break testing in real world

3 | FlySense

Our next big milestone is the second NEA pilot workshop this coming Friday where we will be showing a few mock-ups and "finalizing the art-work"/setup parameters on what to deploy.

We got a lot of feedback in the first workshop, but we need to show a few options so that we close the design and start making the final programming:



SYSTEM LEVEL REQUIREMENTS UPDATES: PILOT INTERFACE

We have tailored our requirements to align with feedback from NEA pilots

Feature	Target Performance
Bird's Eye View coloring	• Clearly communicate <u>forward</u> direction (e.g. an arrow, a triangle) • Ensure surrounding images move <u>smooth</u> across time • Include a <u>"ghost ring"</u> indicating where it is really dangerous (addition to coloring obstacles) • Show <u>no more than THREE</u> levels of coloring on the obstacles (with crisp colors) • <u>Most dangerous object should be blinking</u> and have a bigger dot in the screen
Bird's Eye View sound	• Sound warnings should <u>have current pilot input</u> into account (not potential inputs) • Sound warnings should be <u>consistent with the coloring regions</u> (<THREE levels of warning) • Either do 360 degrees warnings or simply do flat sound (e.g. <u>do not distinguish left or right</u>) • Audio warnings should always be done at the <u>same pitch</u> to clearly identify its relation to obstacles
Pilot input override	• Pilot should clearly understand the <u>dimension being constrained</u>: speed or attitude – The dimension being constraint should be colored accordingly in the HUD with YELLOW or RED – Show message at the bottom of the HUD with the dimension being overridden

13 | FlySense

Individual achievements for the past 2 weeks

During the spring break I dedicated myself to write/review all the Matlab code of the algorithms being the project:

- Dynamic window (what is the region shown to the pilot)
- Circle of doom (what is really dangerous from what you see around)
- Coloring of the region shown to the pilot (what is the danger level of the different obstacles based on what the current state and what the pilot <u>can do</u> next)
- Sound warnings (and how to highlight to the pilot which point is causing the beeps based <u>on pilot input)</u>
- Obstacle avoidance (pilot control override working as an emergency brake when collisions are eminent)

I also worked with Hari on the deployment of the Sound and Coloring code in the Jetson, but we could not finish everything as we got held back by unforeseen factors. First the coloring code that worked in our laptops did not work in the Jetson (there was an overflow in one of the intermediate steps that did not overflow when the code run in the laptops) and Hari ended up having to reformat his computer as its performance was degrading ever more.

<u>Control Algorithm used</u>

Assuming a linear system, the position on a specific dimension is given by (x_i is the i coordinate of a three-dimension 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 xy 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 the attrition is the same for the x and y axis which is approximately true) and z axis (much more drag)

The speed is given by:

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

For obstacle avoidance we can easily determine the time for impact from zeroing the speed equation and introducing this time in the position equation:

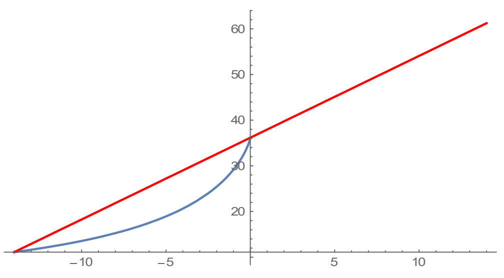
$$\Delta\text{X\_i} == \frac{m\text{v0X\_i}}{\text{AX\_i}} + \frac{m(\text{veqX\_i} * \text{Log}[1 - \frac{\text{v0X\_i}}{\text{veqX\_i}}])}{\text{AX\_i}} * \text{veqX\_i}/\text{veqX\_i}$$

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 using it (C++ is bound to be even worst).

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 then 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”:

Example:



Y is the computed range (output in m)
X is the equilibrium speed (input in m/s)

Blue is the “real” product log function

Red is the simplified control (please note that his is not a classic linearization about a single point, is a simplification that fits across two different points/regimes)

Note: this data uses the calibration from the flight with 2.9 kg total payload.

The result is that we will be 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”

The discussion above is for the direction(s) constrained by the existence of obstacles. In directions where there are no obstacles the pilot is by default in “Type 1” regime.

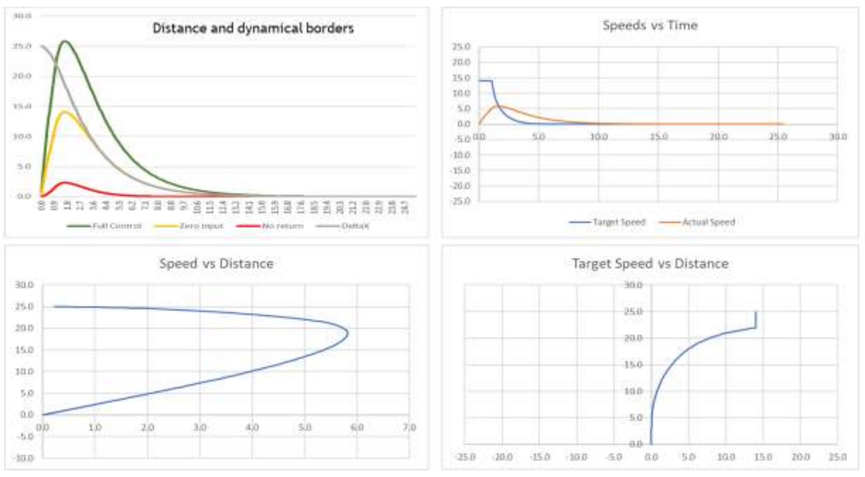
Obstacle avoidance test cases

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

- 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

TEST CASE 1: 1D FROM ZERO VELOCITY

Starting from 0 velocity with full input forward and obstacle at 25 meters



Note: Distances in meters, speeds in meters/second

14 | FlySense

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.

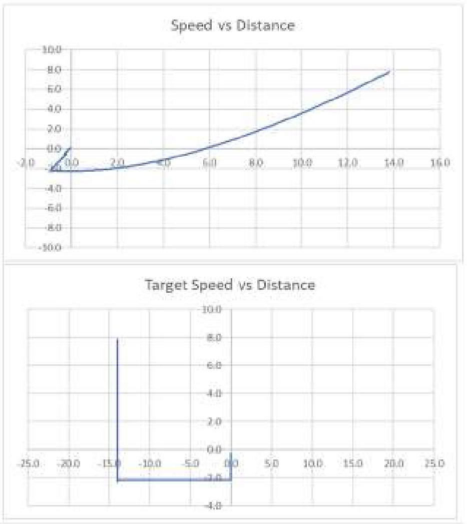
TEST CASE 2A: 1D WITH DIFFERENT TYPES OF DYNAMICS

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

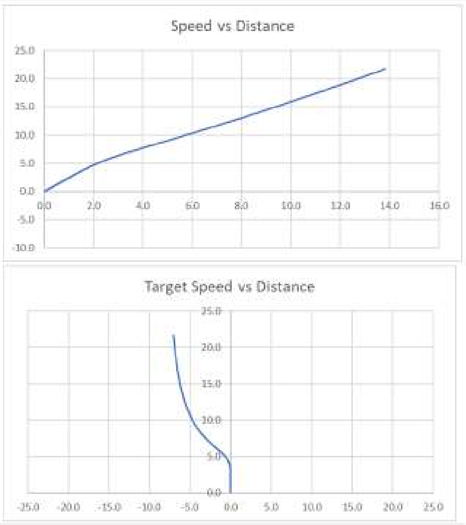
TEST CASE 2B: 1D WITH DIFFERENT TYPES OF DYNAMICS

Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s

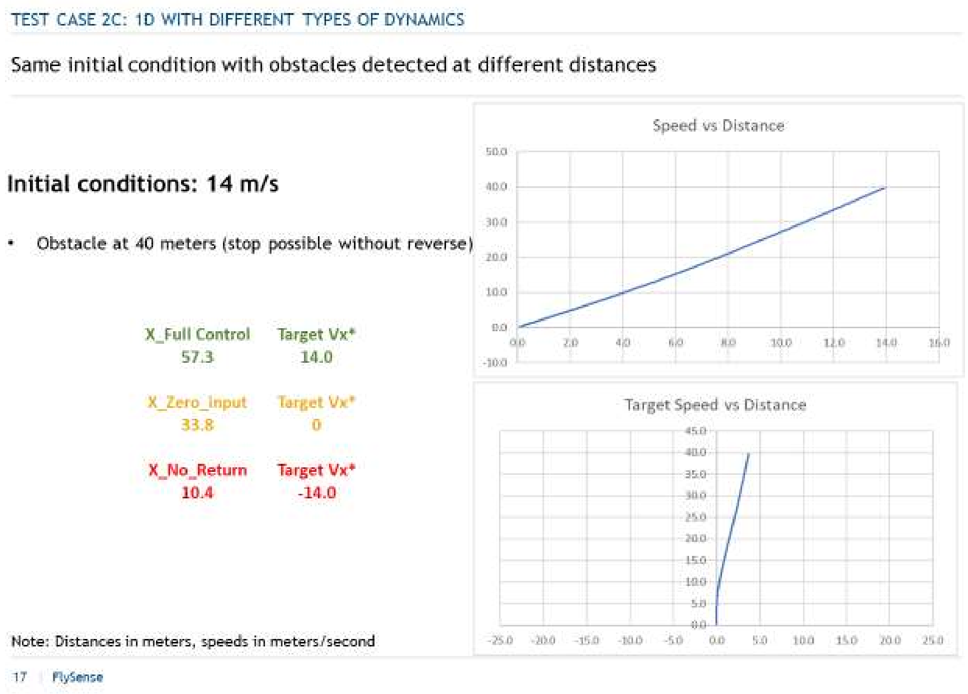
- Obstacle at 22 meters (stop possible using reverse)

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



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

Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s

- Obstacle at 40 meters (stop possible without reverse)

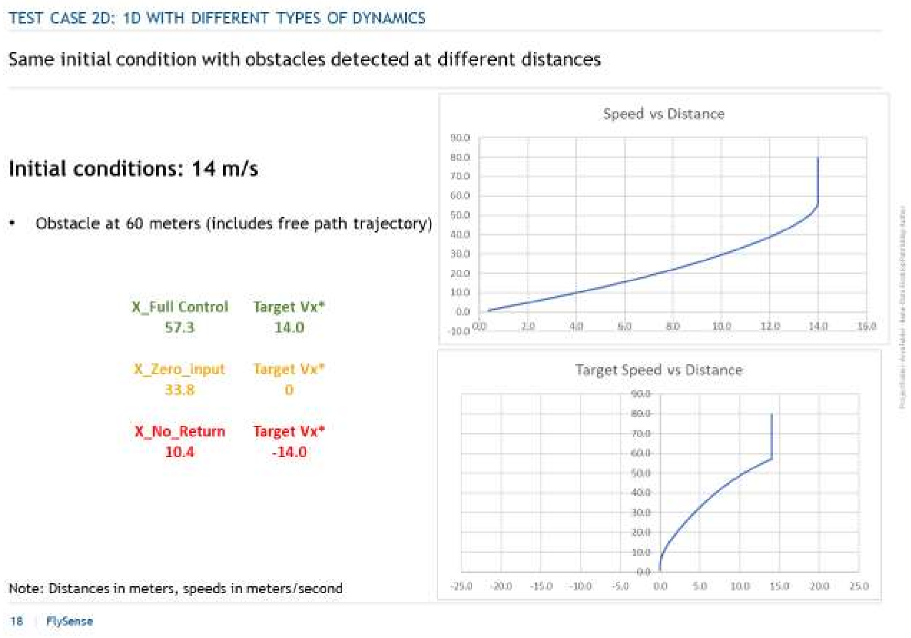
X_Full Control	Target Vx*
57.3	14.0
X_Zero_input	Target Vx*
33.8	0
X_No_Return	Target Vx*
10.4	-14.0

Note: Distances in meters, speeds in meters/second

17 | FlySense

Note: on the code implementation in the quad we define a minimum clearance distance. This will be the distance, measured from the obstacle, where the quad will stop.

- 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



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

Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s

- Obstacle at 60 meters (includes free path trajectory)

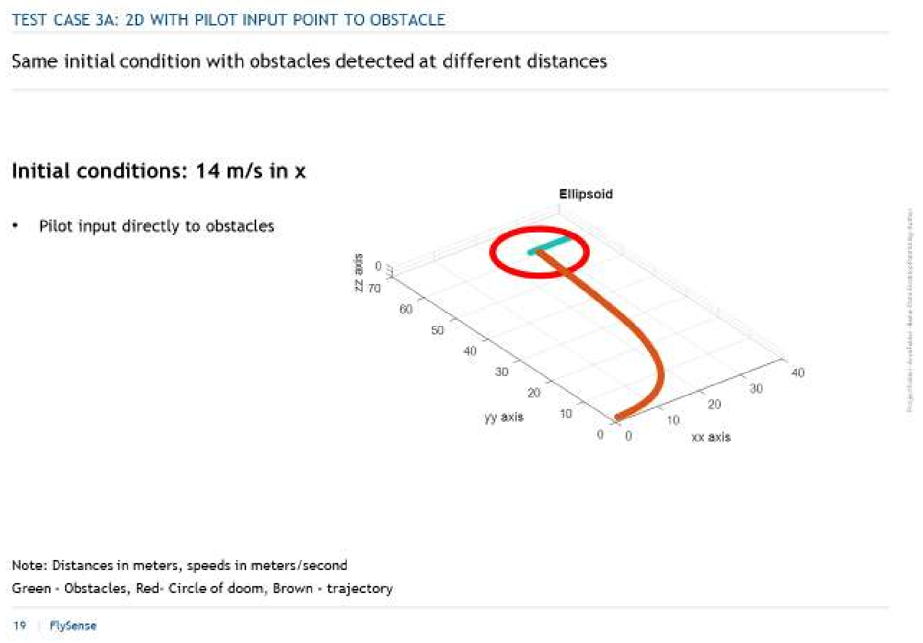
X_Full Control	Target Vx*
57.3	14.0
X_Zero_input	Target Vx*
33.8	0
X_No_Return	Target Vx*
10.4	-14.0

Note: Distances in meters, speeds in meters/second

18 | FlySense

We then initiated tests on 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 sopped the quad 1 meter before the obstacle (clearance distance defined in the simulation).



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

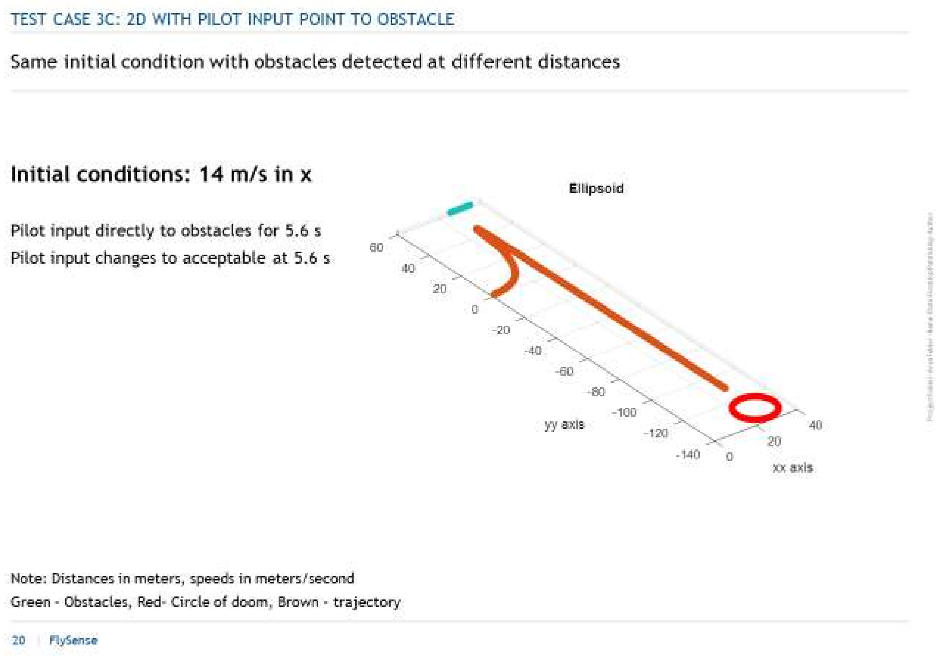
Same initial condition with obstacles detected at different distances

Initial conditions: 14 m/s in x

- Pilot input directly to obstacles

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

- 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



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

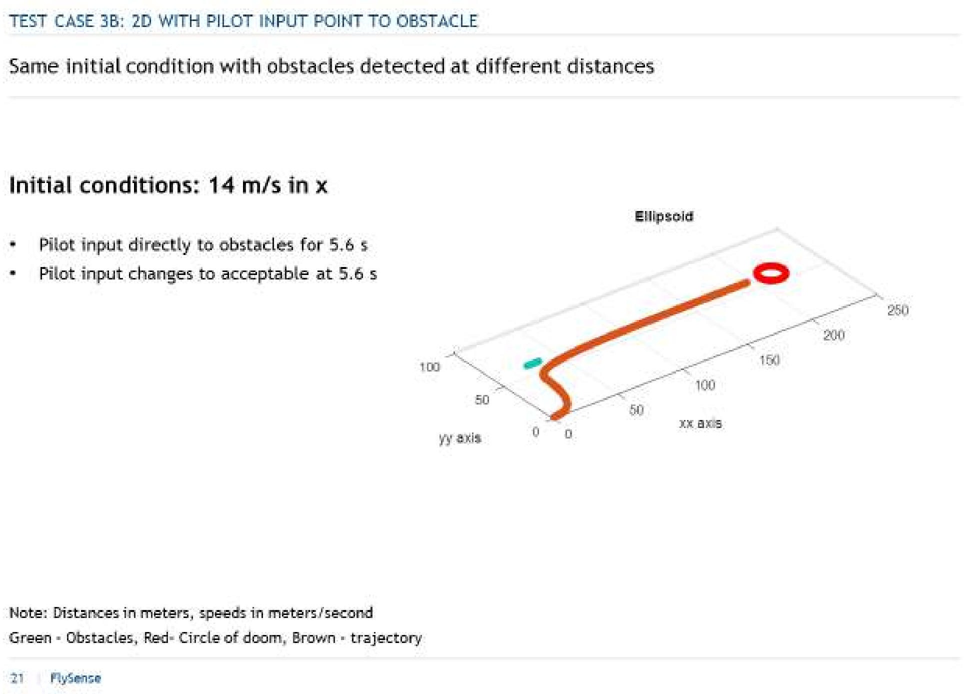
Same initial condition with obstacles detected at different distances

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

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

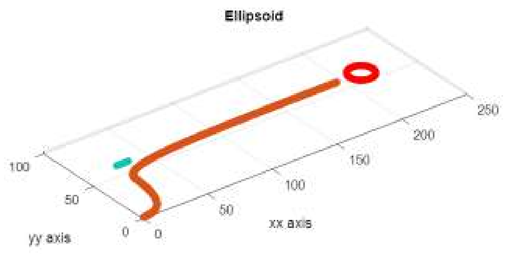


TEST CASE 3B: 2D WITH PILOT INPUT POINT TO OBSTACLE

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

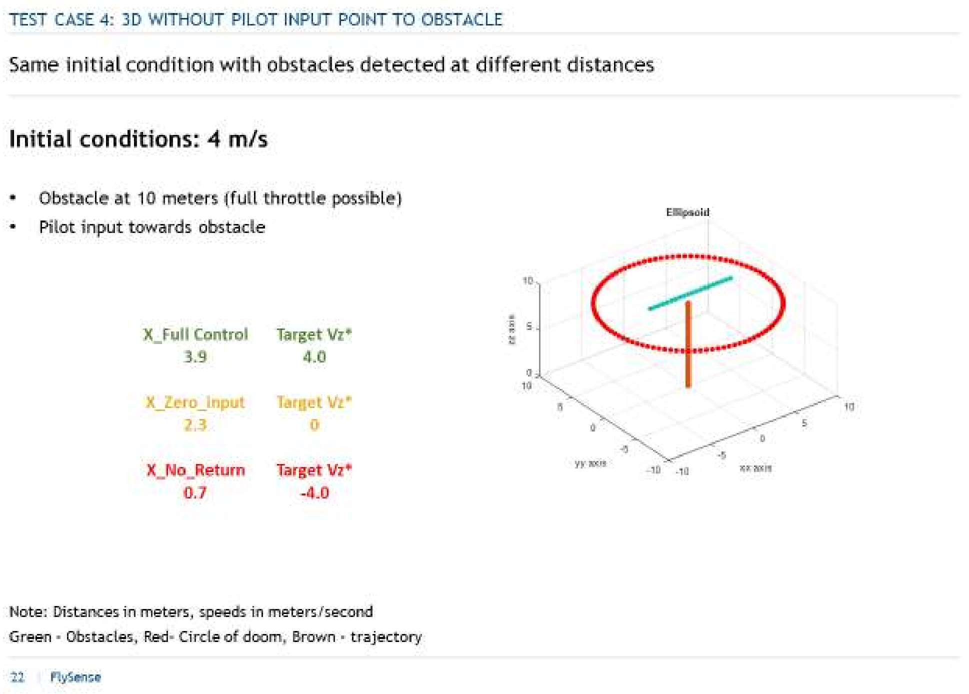


Note: Distances in meters, speeds in meters/second
Green - Obstacles, Red- Circle of doom, Brown - trajectory

21 | FlySense

The last set of tests were done in 3D

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



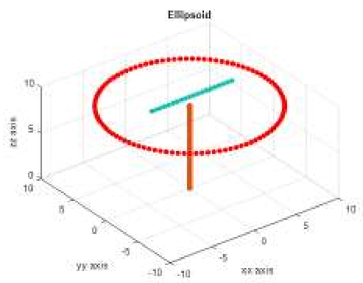
TEST CASE 4: 3D WITHOUT PILOT INPUT POINT TO OBSTACLE

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

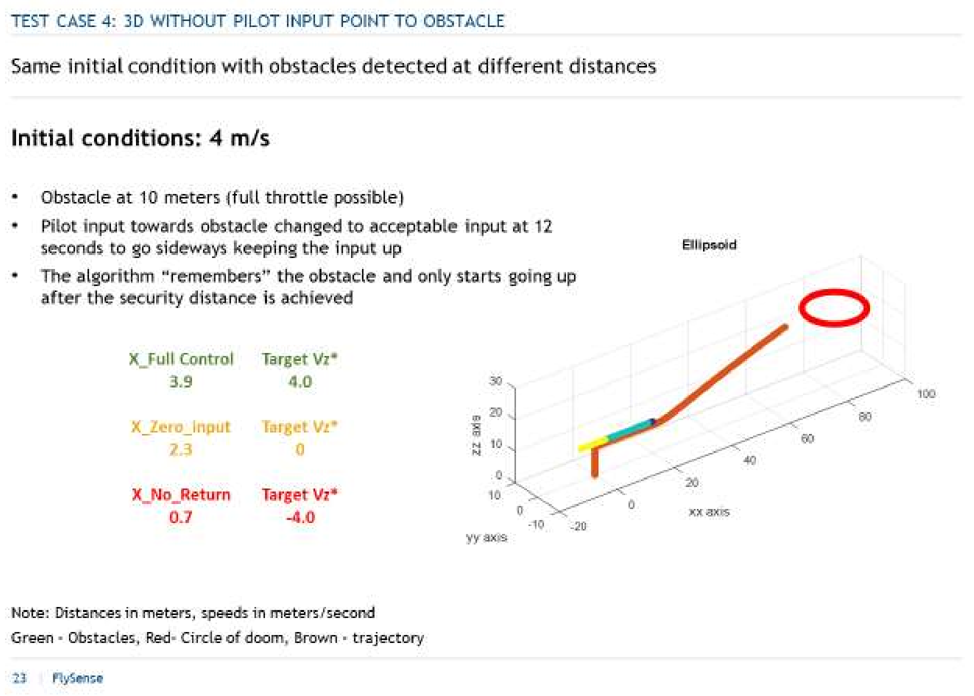
X_Full Control	Target Vz*
3.9	4.0
X_Zero_Input	Target Vz*
2.3	0
X_No_Return	Target Vz*
0.7	-4.0



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

22 | FlySense

- 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

23 | FlySense

This was the most difficult case as we found that in the first implementation we had done, once a direction was "forbidden" all subsequent pilot inputs in that direction were ignored. The algorithm was not forgetting the obstacle.

Milestones for next two weeks:

For the next PR I plan to have:

a) Prepare the NEA pilot workshop with the rest of the team
b) Work with Hari and Shivang in deploying/optimizing/reviewing all the code

Problems Faced these past weeks:

The biggest problem is, as always, a substantial load from all the different courses... made worst by the fact that this semester there is not only one group to interact with across three different courses, but rather a different group for each course.

Key risks:

Navigating through the heavy load coming from multiple assignments across all courses. I am beginning to regret the day I convinced professor Dolan to have 5 technical courses in a single semester (I had an MBA which made the business classes kind of redundant).