

Autonomous guidance and object classification of uavs.

Internship Iti Certh
Summer of 2025

***Presenter:
Aximiotis Dimitrios***



Goal of the project

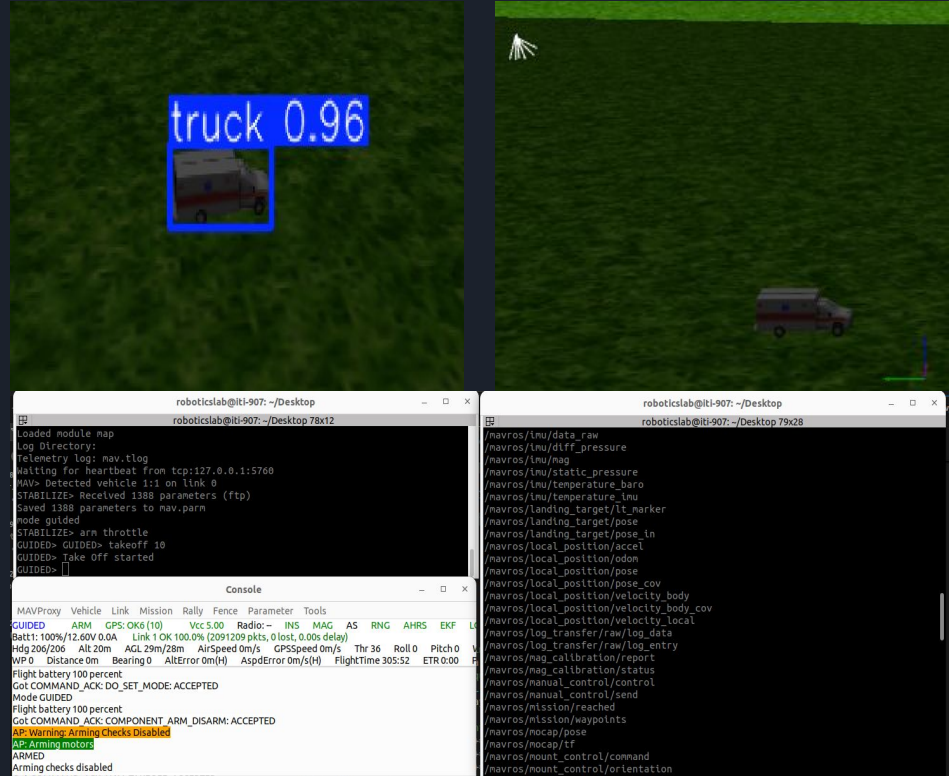
The following project examines approaches for drone tracking control methods using a simple rgb camera. Although it seems a simple project there are many problems we need to overcome to make a correct tracking in a real environment such as:

- finding a way to estimate the depth from the rgb camera,
- making a robust control algorithm that rejects disturbances,
- estimating unknown parameters for the interaction system between the camera and the classified object,
- providing acceptable control inputs to the drone so it won't overturn.

Working environment

To perform the simulations the following tools were used to resemble an actual tracking environment:

- **Gazebo 11** classic ,in which we perform the simulation
- **Yolo 8n** for object detection
- **Ros2** in order to read data from the simulation and publish the controller commands
- **Rviz2** to visualize the yolo detection
- **Ardupilot** to simulate realistic drone flight dynamics and monitor the drone's state.





Workflow

The solution of the problem can be divided into 2 connected topics,

- Classification and detection of the surrounding objects
- Autonomous driving of the drone through the rgb camera.

It is clearly obvious that the 2 topics are connected since the control methods that will be presented use the camera information of the drone .In particular to drive the drone we use the bounding box from the yolo detection.

Yolo detection

For the detection problem we used:

- An rgb camera 320x240 pixels with 1.05 field of view adapted in the uav,
- A modified gazebo model (ambulance).
- The yolov8n model, that we train with a large dataset for a better detection.

Logging results to runs/detect/yolo_retrain_single_class22
Starting training for 55 epochs...

Epoch	GPU mem	box_loss	cls_loss	dfl_loss	Instances	Size					
1/55	1.97G	0.9613	3.434	1.001	21	640: 100%		7/7	[00:04<00:00, 1.43it/s]		
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		1/1	[00:00<00:00, 1.61it/s]	
	all	30	30	0.00333	1	0.679	0.503				
Epoch	GPU mem	box_loss	cls_loss	dfl_loss	Instances	Size					
2/55	2.15G	0.904	2.484	0.9552	31	640: 100%		7/7	[00:02<00:00, 3.18it/s]		
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		1/1	[00:00<00:00, 2.69it/s]	
	all	30	30	0.00333	1	0.949	0.685				
Epoch	GPU mem	box_loss	cls_loss	dfl_loss	Instances	Size					
3/55	2.17G	0.8916	1.623	0.976	34	640: 100%		7/7	[00:02<00:00, 3.18it/s]		
	Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100%		1/1	[00:00<00:00, 2.42it/s]	
	all	30	30	0.00333	1	0.845	0.693				





We took pictures from the rgb camera of the drone from gazebo and we trained the yolo model under multiple angles in order to have a better confidence interval and greater stability for the interaction system.



Autonomous guidance

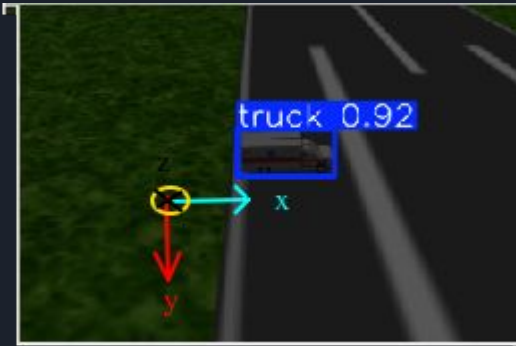
The methods implemented for this part of the problem are:

- Adaptive PID control, with depth estimation through yolo bounding box
- Adaptive PID control, with depth estimation through mlp depth estimation
- Lyapunov least squares estimator with sliding mode controller , using yolo bounding box to estimate depth
- Lyapunov least squares estimator with sliding mode controller , using a depth sensor

Coordinate transformations


It is essential to mention that in order to apply all the methods discussed above we need to transform the error in pixel from the camera to the world coordinates.

Camera Coordinates



Desired Coordinates




- 
- The first transformation matrix represents the rotation of the drone in respect to the camera .

$$R_{b_1c} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

- We will need an additional transformation in the y-axis (pitch) to reach the body coordinates of the drone. Assuming there is an angle of 30 degrees in the pitch, the second rotation applies this transformation.

$$R_{bb_1} = \begin{bmatrix} \cos\left(\frac{\pi}{6}\right) & 0 & \sin\left(\frac{\pi}{6}\right) \\ 0 & 1 & 0 \\ -\sin\left(\frac{\pi}{6}\right) & 0 & \cos\left(\frac{\pi}{6}\right) \end{bmatrix}$$



The topic for velocity commands in gazebo gives values to its coordinate system to transform them in body velocities we need three rotation matrices:

- The pixel rotation of the camera
- The pitch rotation in the y-axis of the drone
- Its own orientation in respect to the world coordinates Rob

So the final transformation is :

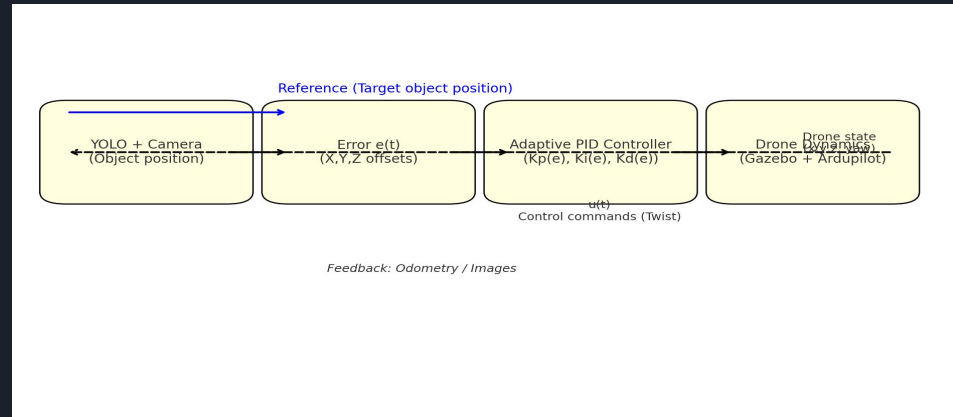
$$R_{total} = R_{ob} \cdot R_{bb_1} \cdot R_{b_1c}^T$$

,while the Rob transformation can be given from an odometry topic. This transformation is used to provide correct velocities commands. In that way the message to be published from a Twist topic is:

$$V_{final} = R_{total} \cdot V_{estimated}$$

Adaptive PID

In order for us to have correct velocity inputs to the drone we need full knowledge of the jacobian matrix (interaction matrix) between the error and the velocity. Since it's never possible to always have the actual system, this approach overcomes the need of an interaction matrix.





For the following equations we need to mention that

- The error term $e(t)$ is normalized between $[-1,1]$
- We smooth the control gains (K_d, K_e, K_i) with a sigma function for greater stability.

$$u = K_e e(t) + K_i \int e(t) + K_d \dot{e}(t)$$

$$K_e = \text{round}(e(t)/0.05) + \text{round}(\dot{e}(t)/0.05) + c$$

$$K_d = \text{round}(e(t)/0.05) + \text{round}(\dot{e}(t)/0.05) + c$$

The adaptation mechanism is only used for the proportional and the derivative term while for the interval term we used a constant gain. This approach manages to use magnitude of the error, so it can achieve greater control inputs.



Moving average

As we previously mentioned it is possible that error in pixels from the center of the bounding may have disturbances. We use a moving average filter in the error measurements so we can :

- Reject noise
- Overcome oscillations in the pitch angle of the drone
- Smooth the signal error and make continuous control inputs.

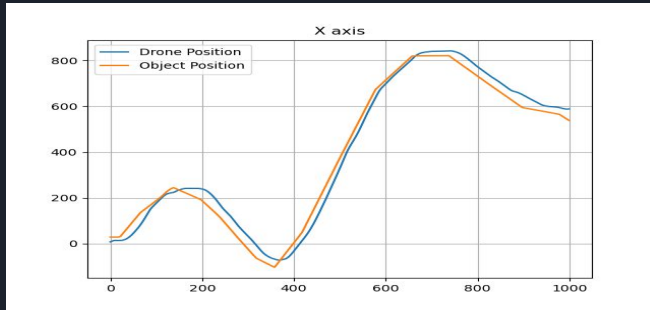
$$e_{smooth}[k] = b \cdot \frac{1}{N} \cdot \sum_{i=k-1-N}^{k-1} e[i] + (1 - b) \cdot e[k]$$

The smoothed error is the final input error for the velocity controller

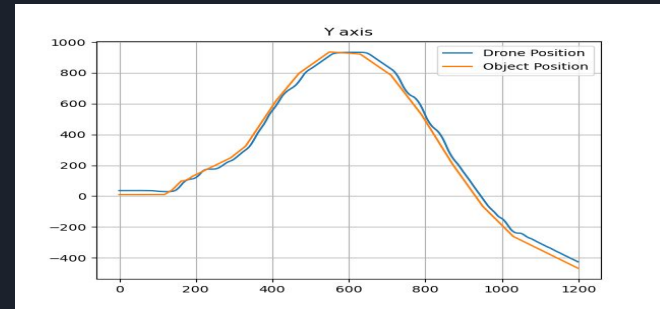
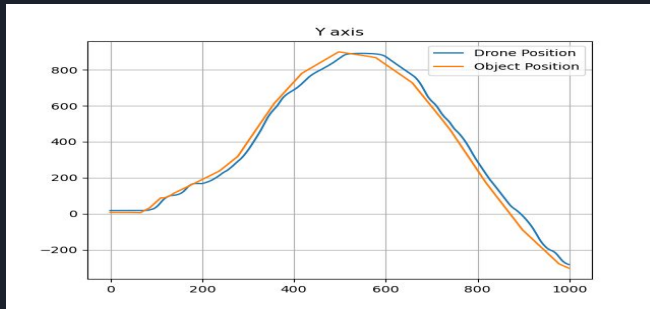
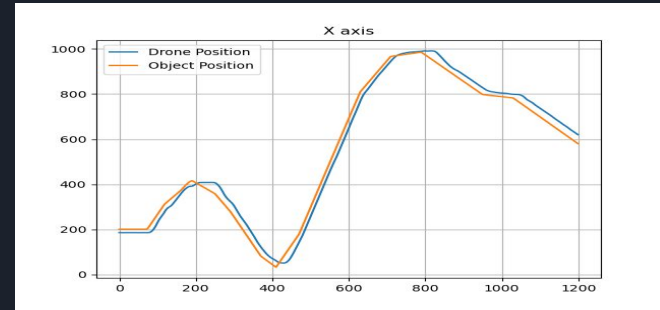
Simulations

Position output

No data filtering

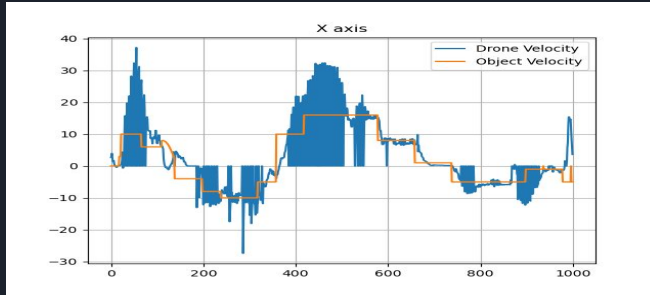


With data filtering

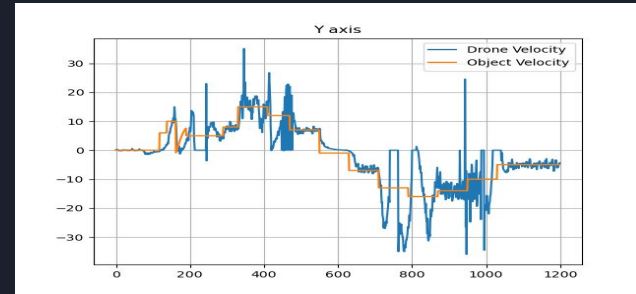
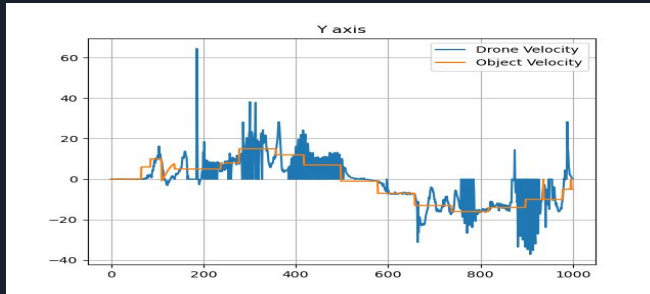
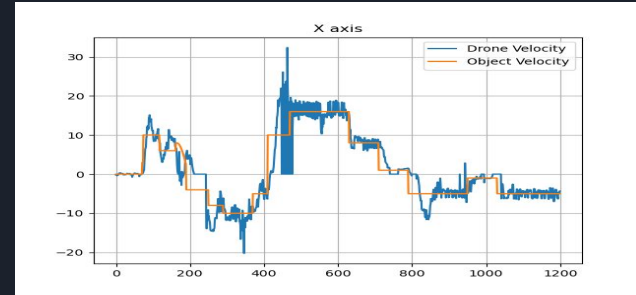


Velocity output

No data filtering



With data filtering



truck 0.96





Conclusions

- The drone follows the detected object at close distance ,without getting too close so it will be visible from the camera.
- Error filtering leads to smoother velocity outputs.
- Although the velocity of the drone does not adapt perfectly to the velocity of the object,the target is always in the view of the camera.




Least squares with sliding mode control

This is a Lyapunov based control method. From image based visual servoing there is an Interaction matrix connecting the error in pixels to body velocity commands of the drone.

$$\dot{e}(t) = L(x, y, Z) \cdot V_c$$

$$\dot{e}(t) + \lambda_1 e(t) + \lambda_2 \int e(t) = 0$$

$$V_c = -\lambda_1 L^T (LL^T)^{-1} e(t) - \lambda_2 L^T (LL^T)^{-1} \int e(t) - \rho L^T (LL^T)^{-1} \text{sat}\left(\frac{e}{N}\right)$$

- 
- The L matrix is the jacobian matrix connecting error in pixels to velocity commands both linear and angular
 - The gains (λ_1, λ_2) define the overdump and the response time of the system
 - The last term in the velocity controller with the saturation is a sliding term helping us to overcome non linearities since the the position of the target is not constant.



IIR Filtering

It is obvious that we still need a data filtering method in order to

- reduce the spikes of the LS method,
- make more continuous control inputs,

The mathematical representation is the following

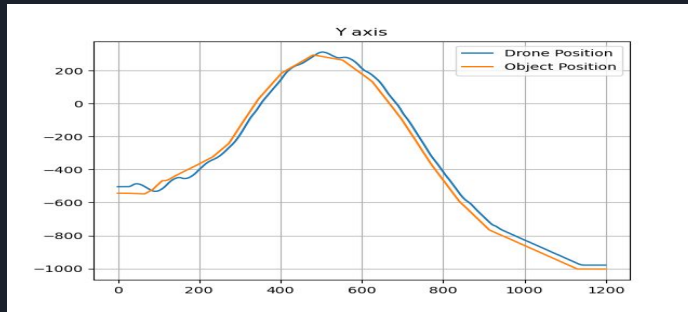
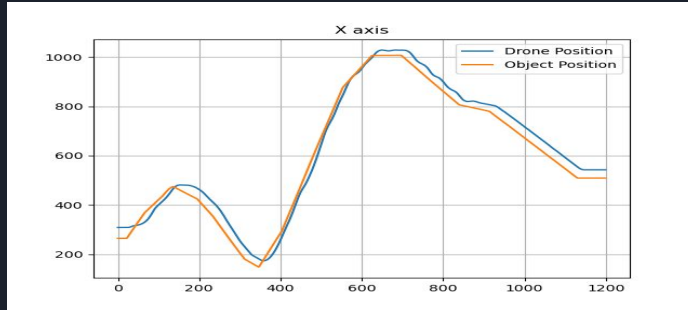
$$V_k = \sum_{i=1}^8 a_i V_{k-i} + b_k V_k$$



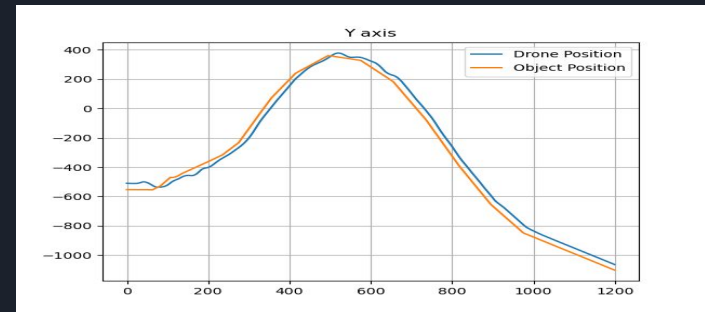
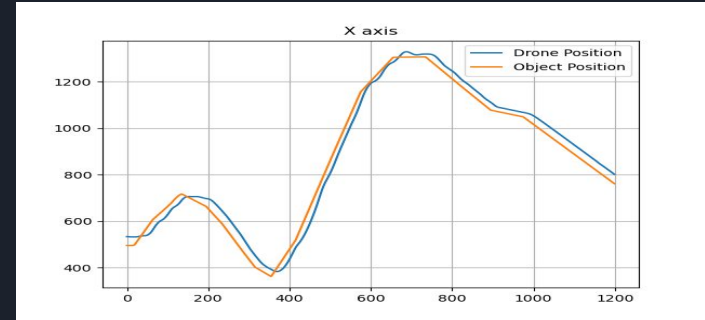
Simulations

Position output(no depth)

No data filtering

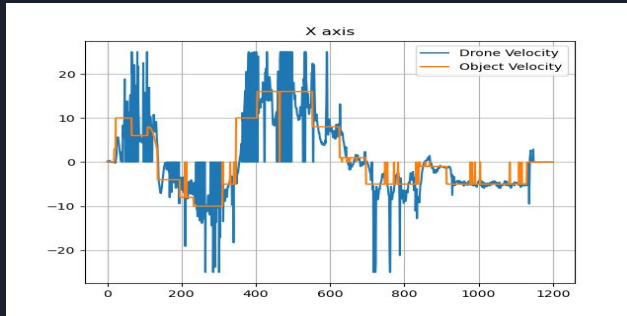


With data filtering

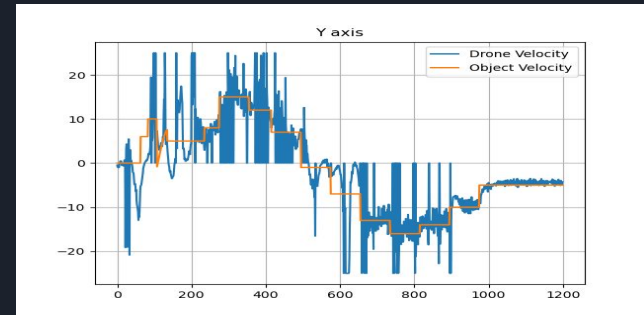
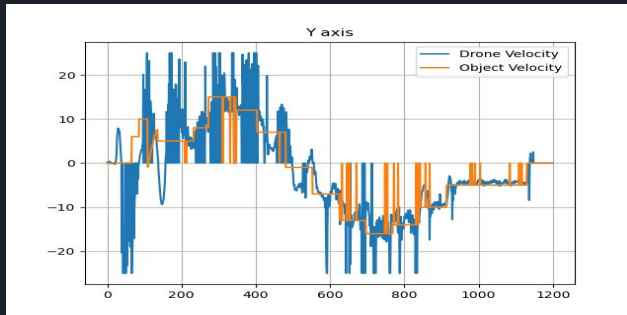
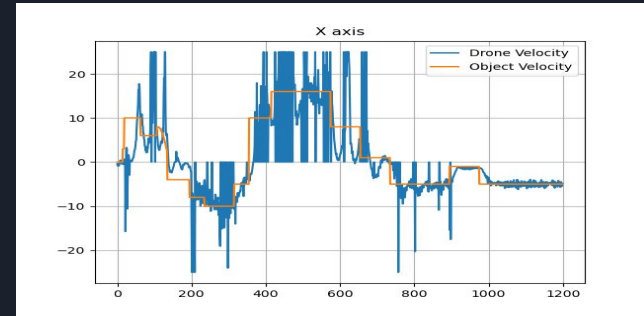


Velocity output

No data filtering



With data filtering



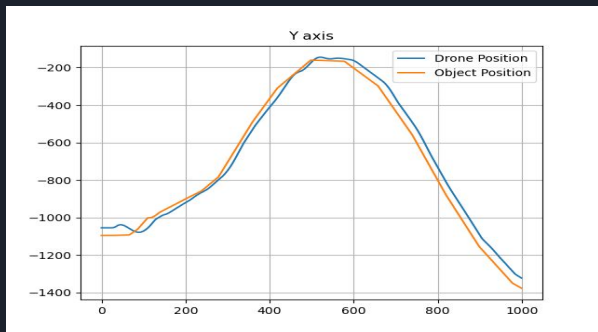
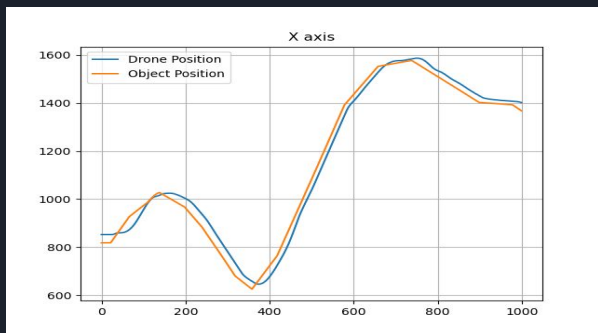
truck 0.89



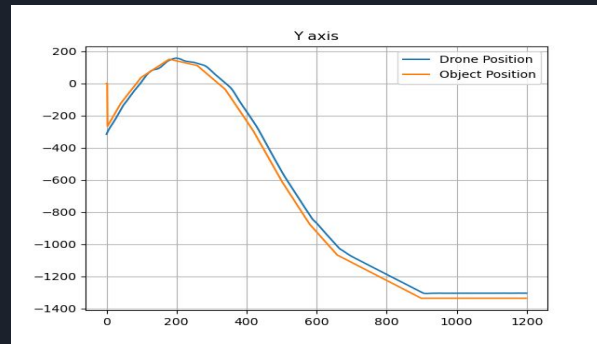
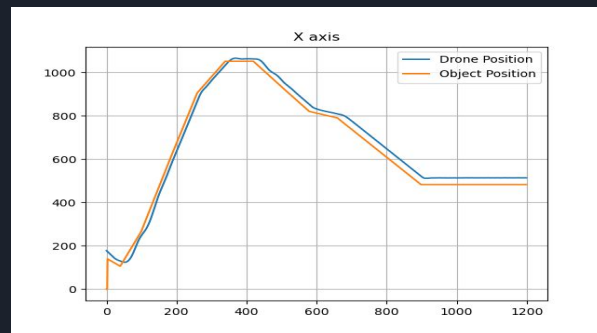


Position output (with depth sensor)

No data filtering

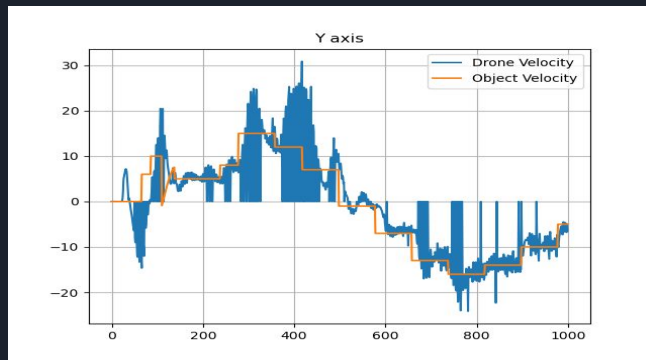
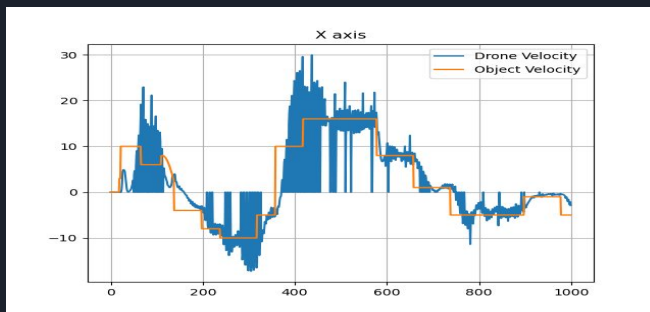


With data filtering

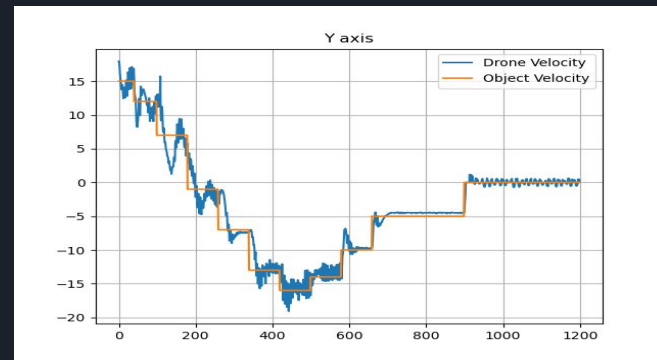
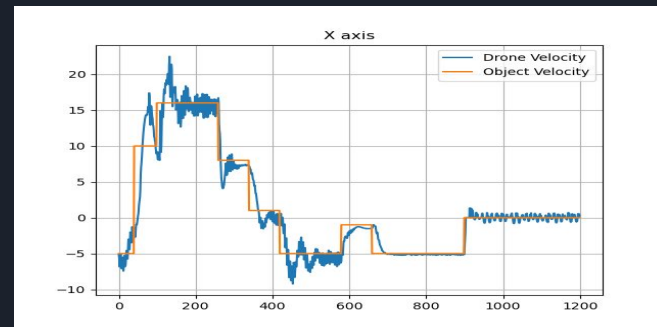


Velocity output

No data filtering



With data filtering



truck 0.93





Simulation results

From the three simulations presented above we extracted the R2 score for the controller input:

Simulation 1

X_position : 0.9853
Y_position : 0.9967
X_velocity : 0.7975
Y_velocity : 0.6736

Simulation 2

X_position : 0.9828
Y_position : 0.9952
X_velocity : 0.5483
Y_velocity : 0.6166

Simulation 3

X_position : 0.9852
Y_position : 0.9938
X_velocity : 0.76015
Y_velocity : 0.7400



Conclusion

- The third simulation provides better results since we have a R^2 score near 1 even for the velocity commands. Also from the graphs provided we can see that we have less oscillations in contrast to other simulations.
- The first simulation with the adaptive pid control also provides good result but there are some oscillations making poor results for the y-axis velocity.
- The second simulation ,although applies the same method as the third one,is using a depth estimation through bounding box.As it seems having a correct depth value can lead to greater results.



References

- Visual servo control, Part I: Basic approaches ,François Chaumette, S. Hutchinson.
- Visual Servoing Using Sliding-Mode Control with Dynamic Compensation for UAVs' Tracking of Moving Targets.
- <https://ardupilot.org/dev/docs/sitl-with-gazebo.html> ardupilot setup

Git repo: <https://github.com/Aximiotis/Visual-Servoing>

(The Simulation file contains the python code for the autonomous driving methods analyzed in this presentation)

Presentation:

<https://docs.google.com/presentation/d/1Lxy1NxUVzJwtYRjPao2yohfhKR1nKvRmcRtIXkuy3II/edit?usp=sharing>