Hacettepe University

Computer Engineering Department

BBM 480 End of Project Report

Project Details

Title	Active Object Tracking via Simulation
Supervisor	Özgür Erkent

Group Members

	Full Name	Student ID
1	Batuhan Ayvaz	21726971
2	Bekir Semih Tekeli	21889919
3	Batuhan Orhon	21727556

Abstract of the Project

The popularity of Computer Vision is increasing with the advancement of technology. With the evolvement of cameras and embedded systems, the fields of computer vision problems have also expanded. One of the problems of computer vision is the object trackers. We can summarize the object tracking problem as tagging one or more objects from consecutive image sequences. The general approach of the trackers is to find the object by distinguishing the background and foreground. We decided to work on object trackers for this project. For the object tracking algorithms, a labeled data set is essential for the training and testing phases. However, it can be tough to find such a dataset for a particular problem. When we look at the usage areas of object trackers, we can see that they are used widely in autonomous vehicles. However, developing an object tracker in an autonomous vehicle is very costly. At the same time, the testing phases of the object tracker can be long and dangerous. As a result of the observations we mentioned, developing an object tracker in the simulation environment is a wise approach. With the development of technology, the quality of simulation environments has also increased. Simulation environments developed in various themes and environmental conditions can be used for object tracking. In this project, we developed a drone to follow a car with an object tracker in a simulation environment. At the same time, the data of the cameras on the drone and the car were recorded. In this way, these labeled image sequences can be used as datasets for further and other projects. In the future, the developed object tracker can be tested in a real environment. At the same time, deep learning-based object trackers can be developed on powerful computers. Details about future work are discussed under the heading 'The Impact and Future Directions'.

Introduction, Problem Definition & Literature Review

In recent years, unmanned aerial vehicles (UAV) became more popular in some industries such as security and supply chain. Since the drones see the world from bird's eye perspective, tracking objects is both necessary and easy for a UAV. In our work, we created several UE4 environments to implement an active object tracking algorithm to track a land object moving on the surface. UE4 is the most realistic simulation environment for this project, and the Airsim plugin and its python API provided by Microsoft are used to create, control, and record position values of the drone and the land vehicle.

Tracking an object by using only visual data has some challenges. First, we need to detect the object in a taken photo shot. Second, distance between the car and the UAV must be estimated by using the camera's focal length and the position of the car in the image. Third, a velocity vector must be calculated to track the car in a correct way. Each of those steps has its challenging problems.

In the literature, deep reinforcement learning models which learn to track and move UAV together provides the state-of-the-art result. However, training those kinds of models demands lots of data, computation, and systems to train such models. Currently, Airsim doesn't have a reinforcement learning support or at least a support we could use. Additionally, deep learning models takes a lot of time to track the object, and using those models in real-time was not possible with our devices. So, we began the project with a tracking algorithm that works with GPS data of the car using the Airsim API. Such a tracker works perfectly since we calculate the ground truth velocity and location information of the car. In the second term, we used basic trackers such as KCF, MIL, and CSRT. We also used a deep learning model named DaSiamRPN. Out of those four, CSRT works the best considering tracking accuracy and performance. After detecting the car in camera output, we need to measure a velocity vector to track the car. To do that, first, we need to calculate the distance between the car and the UAV using the camera's focal length which we calculated using calibration.

Here are some of the papers we used to get more familiar with the field:

In this article [1], Luo, Wenhan, et al. have developed a tracker that generates a camera control signal using a video sequence. In previous studies, tracking and camera movements worked separately, but in this study, they were able to train both together. They propose an end-to-end solution through deep reinforcement learning that adopts a ConvNet-LSTM function approximation for direct-to-framework prediction. They tried the simulations they made in Unreal Engine in many different environments, so they were able to achieve a good generalization (a good generalization is more successful in real-world scenarios). They also tested their tracker in the VOT (Visual Object Tracking) competition, and in their tests, they saw that their tracker was suitable for use in the real world. In figure 1, you can see the pipeline of active tracking.

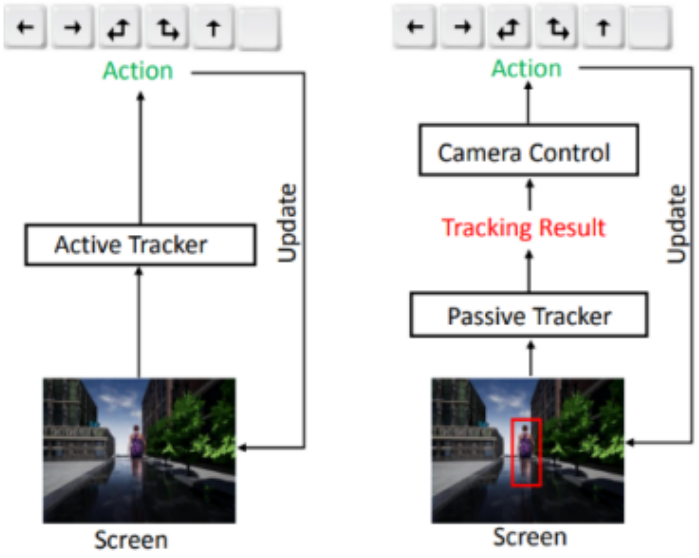

Fig. 1. The pipeline of active tracking. Left: end-to-end approach. Right: passive tracking plus other modules[1]

The authors of the article[1] published a new article [2] in the following years. They developed the generalization feature of the tracker by testing the simulations they made on the Unreal engine in developed environments. At the same time, they tried their tracker in the real world and achieved successful results.

In this article [3], Chen, Gang, et al. mentioned that the cost of training deep learning networks in the real world can be reduced by using simulations effectively. However, networks learned in simulations should also be successful in the real world. Therefore, they designed a model to solve real-world problems of mesh networks training in simulations. They propose a visual information pyramid (VIP) model to systematically explore a practical representation of the environment. They designed a representation with spatial and semantic information synthesis. In Figure 2 you can see the elevated VIP theoretical model for image-based navigation and their environmental representation.

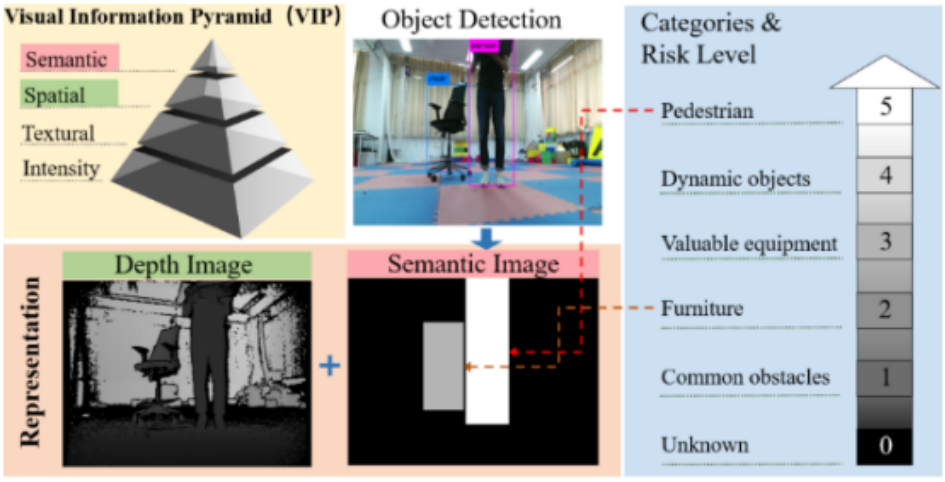

Fig. 2. The raised VIP theoretical model for vision-based navigation and their environment representation.[3]

Article [4] presents a new lightweight, real-time, embedded object tracking approach with multiple inertial detection data. Chen, Peng, et al. proposed to extract feature points from the video frame using the oriented FAST and rotated binary robust independent elementary features algorithm. It then adopts a local difference binary algorithm to generate the image binary descriptors. The k-nearest neighbor method is then used to match the image descriptors. You can see their object detection framework in figure 3.

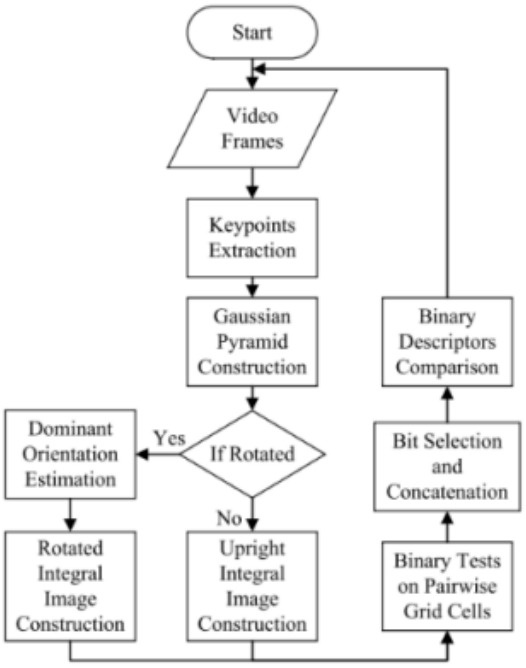

Fig. 3. Article [4]'s object detection framework

In article [5], Erkent, Özgür, et al. offer a network model for multi object training. We couldn't create time for multi-object tracking in this project. However, this paper might be useful for the future directions of the project. In figure 4, you can see the general architecture of the approach.

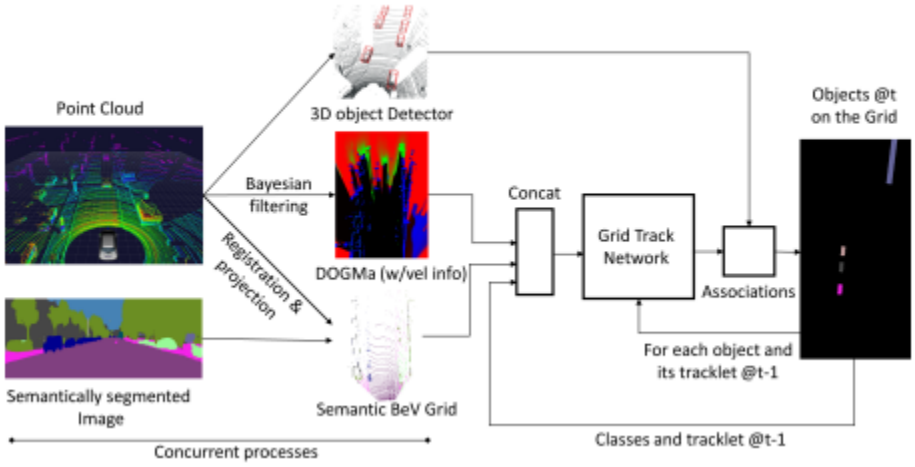

Figure 4. General Overview of The Approach of Article [5]

In article [6], DaSiamRPN architecture is introduced. Zhu, Zheng, et al. are inspired by the DaSiamese Network and SiamRPN Network architectures and offer the DaSiamRPN model which outperforms previous approaches in tracking purposes. This model worked as the second best option in our project considering performance and accuracy.

Methodology

Tools

AirSim
In this project, we will use the AirSim plugin. AirSim is a plugin for drones, cars, and more built on Unreal Engine. It is open-source, cross-platform, and supports software-in-the-loop simulation with popular flight controllers such as PX4 & Ardupilot and hardware-in-the-loop with PX4 for physically and visually realistic simulations. It is developed as an Unreal Engine Plugin that can simply be dropped into every unreal environment. The main goal is for developing AirSim as a platform for AI research to experiment with deep learning, computer vision, and reinforcement learning algorithms for autonomous vehicles. For this purpose, AirSim also exposes APIs to retrieve data and control vehicles in a platform-independent way.

Unreal Engine
Unreal Engine is open-source which is a great advantage because it makes development easier and more efficient. It has great blueprints which work like building blocks so even artists without any programming knowledge can use the engine to assemble and adjust basic things. Also, this feature is really great for prototyping.
In Unreal Engine, the whole environment will be built according to the project's tests.

Python: High-Level Programming Language
Python became so popular since it's easy to pick up and has a huge open-source community constantly developing tools for it. It is typically very easy to look at well-written Python code and understand what's going on. Python is very popular with autonomous vehicle engineers because there are comprehensive libraries for math, science, data visualization, machine learning, AI, deep learning, etc. The disadvantage with Python is that it's a large, compiled language. This makes it unsuitable for very high-performance applications and can eat up memory.

Robotic Operation System (ROS)
ROS is an ecosystem of software libraries for robot development. Since autonomous vehicles are just large, wheeled robots, this tool makes developing autonomous vehicles significantly easier than it would otherwise be. While ROS supports a few operating systems, it is commonly run on top of Ubuntu. People typically program ROS is C++ and/or Python. ROS is great for all the tools it includes for autonomous vehicles, but it is still a clunky platform to run the autonomous vehicle stack on. It is typically considered a "prototyping tool" until the industry standardized on different hardware and software that will be more deterministic, cheaper, scale-able, and less power-hungry.
We didn't need ROS as we were controlling a single vehicle, but for a larger scale project, ROS could be useful.

Simulation and Programming Language

In order to realize the project, a simulation environment was needed first. We had two different options, Gazebo and UnrealEngine. However, as a result of some research, it was realized that a plugin called Airsim works with UnrealEngine and meets all the desired features. 3 different environments were set from 3 different computers and all tests continued from three different environments. As a software language, we used Python and its libraries for most of the Project. By connecting to AirSim's API via Python client, we performed our tests and studies in the prepared environment in Unreal Engine. Some limitations brought by AirSim's API caused some changes in the project.

First Term

In the first part of the project, tracking processing was completed using position information. At this stage, the speed of the vehicle was calculated using the change of position in the two-time intervals. For the tracking process, the difference between the positions and the instantaneous speed of the vehicle was formalized and the tracking process was performed.

```
velocity_v = (math.sqrt(
    (x_v - x_v_prev) ** 2 + (y_v - y_v_prev) ** 2 + (
            z_v - z_v_prev) ** 2) / (current_time - previous_time))

x_dif = math.sqrt((x_d - x_v) ** 2 + (y_d - y_v) ** 2 + (z_d - (z_v - 30)) ** 2)
v_desired = velocity_v + (x_dif / ((current_time - previous_time) * 50))
```

Figure 5. Velocity Calculation in position-based tracking

Camera Calibration

An environment was created in the unreal engine for camera calibration. A chessboard was placed on the floor. The vehicle was moved with a square-shaped route so that drone can always see the chessboard. From the matrix which is the result of the calibration, the focal length information of the camera, separately from the x and y axes, was obtained. Using these focal lengths, the pixel data taken from the camera was converted to the real distance on the unreal engine.

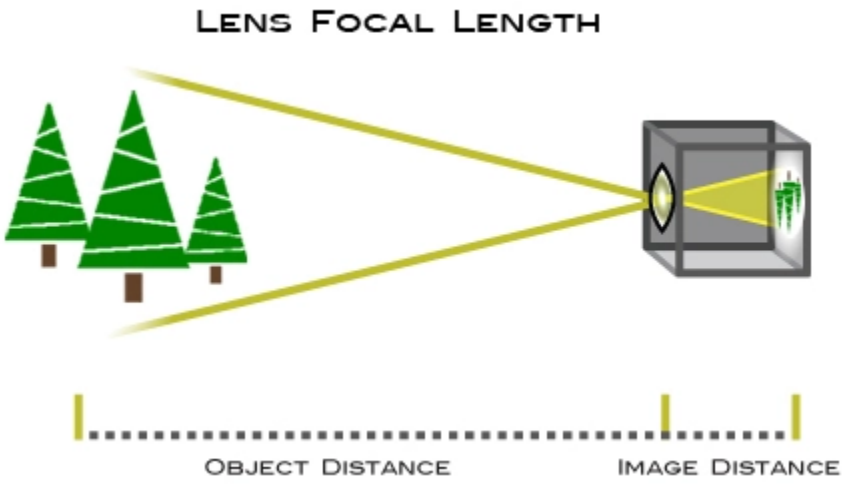

Figure 6. Triangle similarity used to find the relation between pixel distances and real distances

$$Object\ distance/Image\ distance\ =\ Object\ Measurement\ /\ Image\ Pixels$$

Second Term

In the second part of the project, the use of position information for vehicle tracking was abandoned. Only camera data was used for tracking. The distance between the midpoint of the drawn bounding box and the midpoint of the camera was found separately on the x and y-axis using focal lengths. Since the vehicle's motion function works in forward-only mode, the value we found from the x-axis was used only for the vehicle's yaw change instead of using it for roll. The value obtained from the Y-axis was used to calculate the speed to be given to the vehicle. A regression model was trained for the speed to be given to the vehicle, and the speed for each y-value was calculated using the weights obtained from this regression. A linear regression model was also used for the X value. A threshold has been set for the yaw change. The yaw rate was calculated for each x value outside this threshold.

While testing the accuracy of the written code, the graphics obtained after each trial were used. First of all, it was checked whether the tracker started correctly or not on the camera display screen during the test. If the bounding box was properly drawn around the vehicle, the test was continued. The project development was continued by reading and interpreting graphics such as the distance graph between the vehicle and the drone, the value read from the distance sensor, and the ground truth value graph of the height difference between the vehicle and the drone, the velocity graph of the vehicle and the drone. The planned project was completed by giving priority to the correction of the error data obtained from these graphics.

Tried Trackers

DaSiamRPN

A distractor-aware Siamese framework for accurate and long-term tracking is proposed in DaSiamRPN. During offline training, a distractor-aware feature learning scheme is proposed, which can significantly boost the discriminative power of the networks. During inference, a novel distractor-aware module is designed, effectively transferring the general embedding to the current video domain. In addition, the proposed approach for long-term tracking by introducing a simple yet effective local-to-global search strategy is extended. The proposed tracker obtains state-of-the-art accuracy in comprehensive experiments of short-term and DaSiameseRPN long-term visual tracking benchmarks, while the overall system speed is still far from being real-time.

KCF(Kernelized Correlation Filter)

The target-tracking task is studied and an improved KCF-based tracking algorithm that can be implemented in real-time is proposed. In the proposed approach, the KCF tracker is combined with the GM(1,1) grey model, the interval template matching method, and the part-based model, which enables the proposed algorithm to track the target in challenging environments containing occlusion and object scale variation successfully. These performances of the proposed approach above are superior to the traditional KCF-based tracker. The experimental results show that the proposed method outperforms the other trackers in terms of distance precision and success rate. In future work, the tracking speed of the proposed approach should be further improved and some novel intelligent methods will be studied to improve the performance of the tracking algorithm.

MIL Tracker

This tracker works similar to the BOOSTING tracker. BOOSTING tracker has a simple algorithm similar to template matching. MIL tracker additionally uses the neighborhood information of the object being tracked.

CSRT Tracker

This tracker uses Discriminative correlation filter with channel and spatial reliability[6]. The algorithm uses spatial reliability map that provides enlarging and localization of given rectangular or non-rectangular region. The filter is trained on a bigger region than similar filters which improves the filter's discriminative power.[6]

In Figure 7, there is a flowchart for the algorithm.

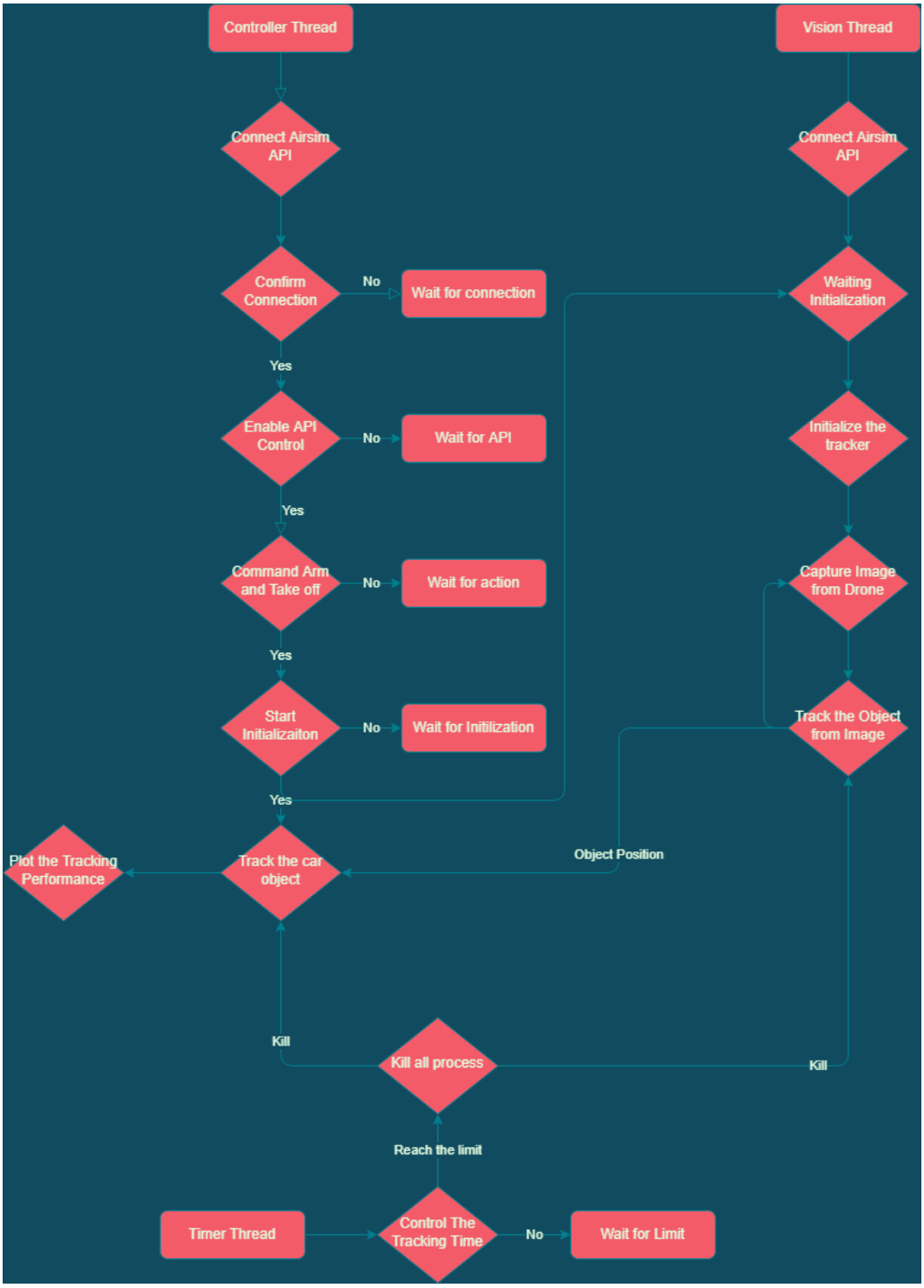

Figure 7. *Algorithm Flowchart*

Results & Discussion

Before sharing the project results, it would be more appropriate to describe the tests of the key points of the project.

Drone Altitude Measurement Test

Measuring the distance between the drone and the ground is substantial for both controlling the drone and calculating the distance between it and the tracked vehicle. Therefore, sensors that can be used to measure altitude were found and tested. In order to decide on the sensor from which we will receive the altitude data, we drew a square route to our vehicle on a flat surface. After a flight of about 1 minute, we drew the graphs using the data we received from the sensors. Although the sensor with the best data was the GPS sensor, we later decided that GPS was not suitable for our problem. In our problem, the drone has to maintain its height relative to the vehicle. In order to do this, we would need the GPS data of the moving vehicle below, so we gave up the use of GPS. We also found a connection between the lidar and the distance sensor with the vehicle's pitch and roll movements. Due to the pitch and roll movements of our vehicle, while it is in motion, we have decided that the most accurate sensor for us is the distance sensor, which will provide us with data in the same direction as our camera.

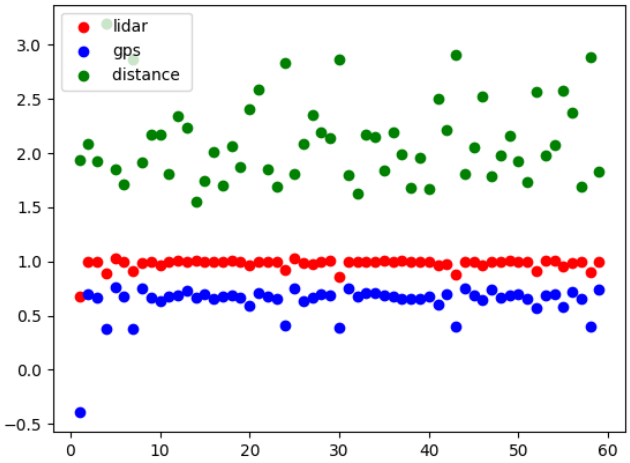

Figure 8. Result of Sensors for altitude

Drone Control Tests

As we mentioned earlier, the distance between the drone and the car is calculated from the camera data. The problem in the control part is how the drone can catch the car using this distance. The formula we talked about in previous documents failed the tests. The camera is very much affected by the drone pitch's movements, and it incorrectly estimates the speed. Therefore, the formula we used doesn't propose any solution to our controller problem. After this test, we suggest a simple and effective controller algorithm and we have managed to track the moving car. Investigating the distance between the car and

the drone during the object tracking period, we trained a regression model to generalize the complete drone controller problem. On the other hand, reinforcement learning can be used for solving this problem properly. Although Airsim has reinforcement support, however, it is a brand-new technology, and it is too slow for training the reinforcement model.

Gimbal Test

In the control tests, we mentioned that the pitch movement of the drone negatively affects the computer vision problems. For this reason, a camera that tries to correct itself according to the drone movements could solve this problem. For this reason, we added a gimbal to our camera, but the Airsim gimbal extension was not developed very carefully. There is a parameter ranging from 0 to 1 that determines how successful the gimbal will be. If we choose this parameter as 1, it gives an unrealistically perfect gimbal performance. For this reason, we started to reduce the parameter, but we realized that the parameter did not actually work as we wanted. This parameter change gives the probability that the real image and the corrected image will be displayed. For this reason, a video sequence that is too distorted to process data emerges.

Limiting Pitch Angle

We talked about the problems of the drone's pitch movement being too aggressive in the previous chapters. For this reason, we wanted to alleviate the problems we experienced by limiting the pitch angle of the drone. However, we noticed that Airsim, the simulation kit we use, does not have such a feature.

Trackers Test

Many new trackers are available in the library with the latest versions of OpenCV. We wanted to try the trackers in OpenCV first and then test the deep learning-based trackers. On the other hand,although we tested it on computers with powerful graphics cards, we experienced low FPS even on low computational trackers. We tried various methods to solve this FPS issue, but we couldn't make any significant improvement as we had to use AirSim's API which is connected to Unreal Engine. For this reason, we tested the trackers in the OpenCV library and decided to use the best one among them.

DaSiamRpn

Siam-based trackers have been used a lot in recent years. For this reason, we wanted to try DaSiamRpn, a Siam-based tracker, in our environment. This Siam-based tracker works based on deep learning and already has a trained model. We performed the necessary tests using this already trained model. When we looked at the test results, we saw that the trackers work slower than the other trackers and are very much affected by environmental changes. It failed in our test because it lost the vehicle that was followed at the very beginning of the tracking path.

MIL Tracker

Another tracker we tried was named Multiple Instance Learning (MIL). Although this viewer seems successful in the article, it loses its target in the sudden movements of the drone in our tests. For this reason, this tracker was unsuccessful in our test results.

KCF (Kernelized Correlation Filter) Tracker

This tracker has an algorithm that works faster than others. However, it is very much affected by light changes, so it failed in our tests.

Discriminative Correlation Filter Tracker (CSRT)

In our tests, the viewer named CSRT gave the best performance. It was able to successfully tag the car that was followed along the path we were considering for the presentation. It is a robust tracker against problems that may arise from sudden light changes, changes in the visible part of the car, and changes in the size of the car. That's why we decided to use this tracker.

Results

As a result of our tests, we decided on the optimal controller and tracking algorithms. We tested these choices on a route identified as the main problem. This route starts out as a straight road at first. Then there is an inclined road, and the tracked vehicle makes an accelerated movement there. Meanwhile, the visible part of the vehicle changes, sometimes a shadow falls on the vehicle, and sometimes the area of the vehicle on the camera changes because the road changes. At the same time, this mentioned road contains sharp bends. The algorithms we used in our experiments were able to successfully complete the route we mentioned. You can see a wide perspective view of the route mentioned in the photo below.

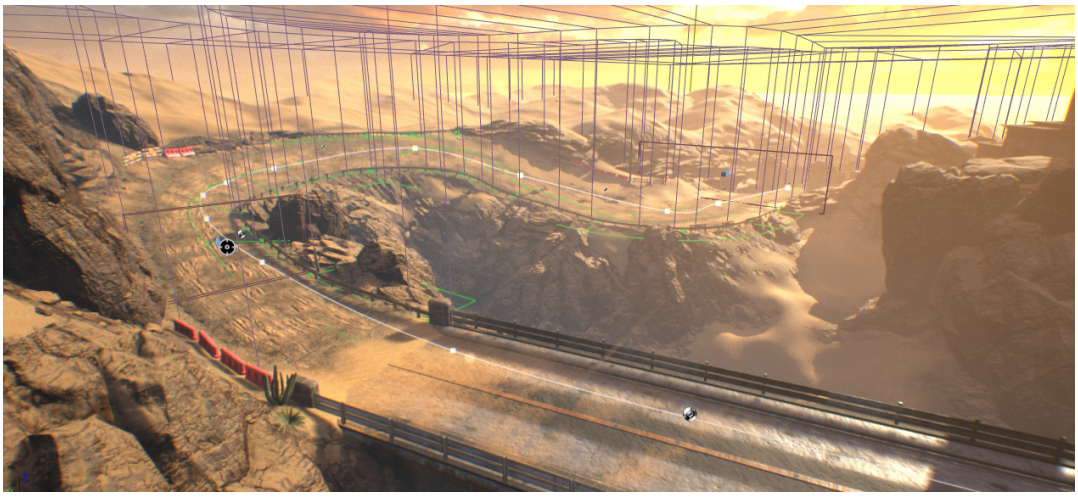

Figure 9. Environment Path

Real Distance vs Camera Distance

The photo below shows the graph of the actual distance (between the drone and the car) and the distance calculated using the data from the camera. As can be seen in the graph, realistic values have been estimated from the camera data. The distance mentioned here is the size of the vector created by the x and y coordinates between the drone and the car.

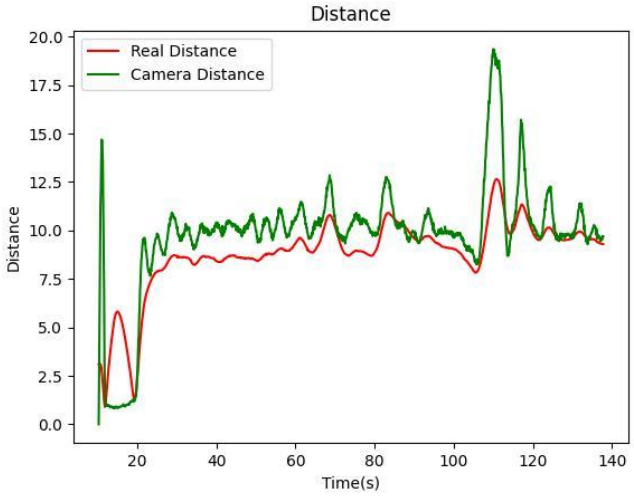

Figure 10. Distance Graph

Distance Sensor Results

Below is the graph of the values measured by the distance sensor during this time. Since the drone was following the vehicle from behind, a gap occurred between the measured values and the actual values on the winding and inclined road.

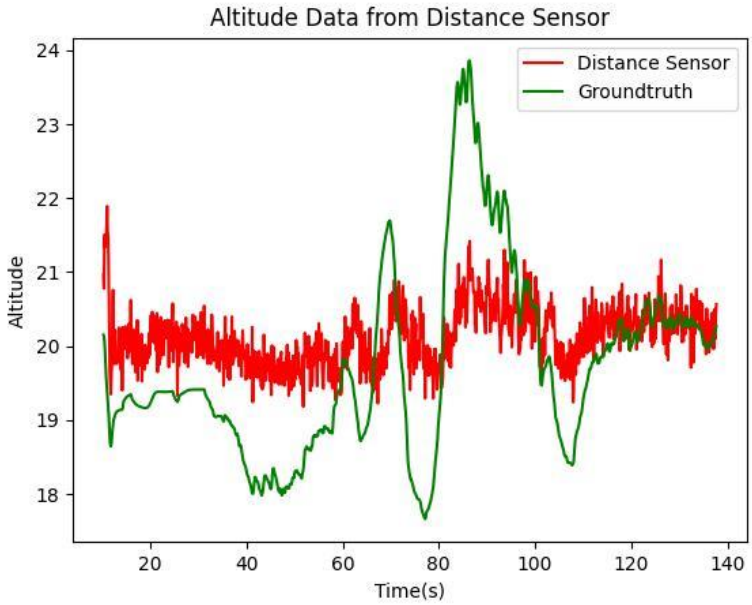

Figure 11. Distance Sensor Test

DEMO

In the image below, an image was taken while tracking the object.

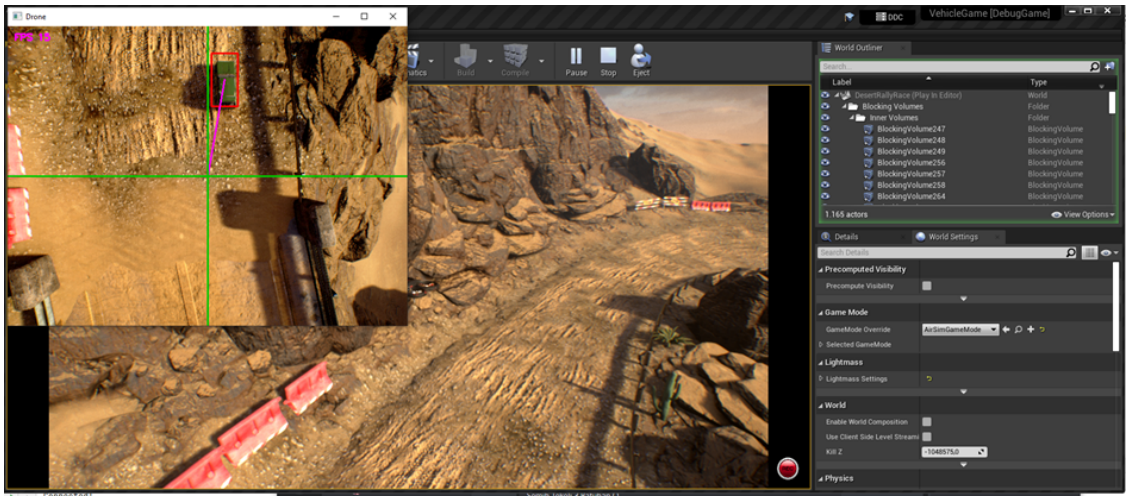

Figure 12. Tracking

In the image below, you can see some different kinds of image types captured from the drone. The most left one is the image captured from depth wise camera, the middle one is the image captured from segmentation camera and the most right one is the image that was captured from raw camera. We only used raw data in the object tracking task, as each image would have a separate computational cost. Although we are working on the last generation GPUs, our frame rate has dropped a lot with the load of simulation. Other camera datas can also be used for object tracking in non-simulation environments or more powerful computers.

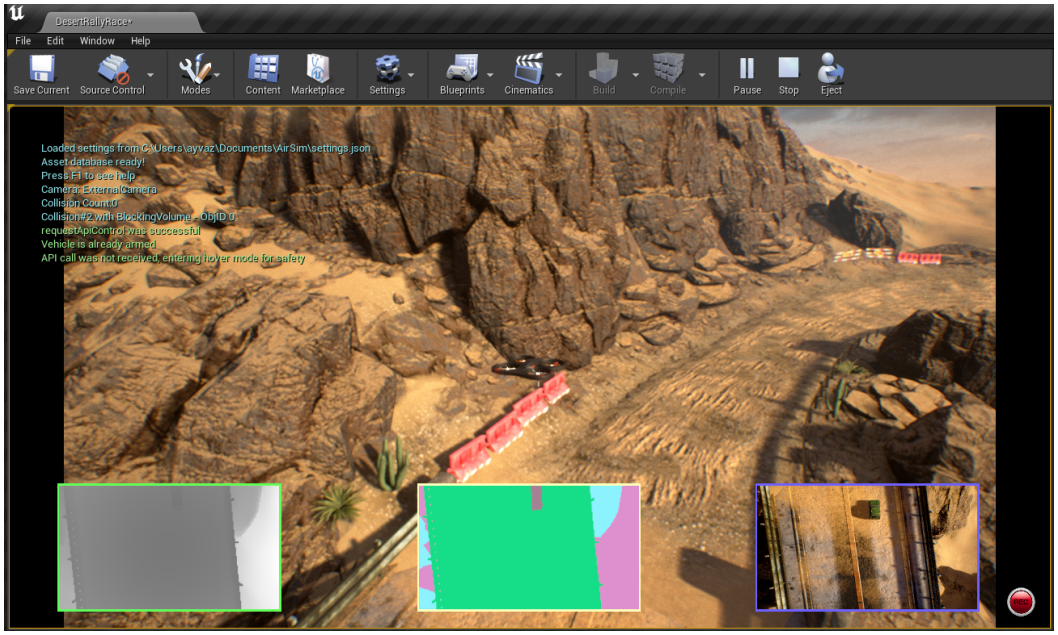

Figure 13. Different Camera Types

The Impact and Future Directions

The use of drones in sectors such as the supply chain, defense industry, and entertainment is increasing. Drones provide great convenience and advantages in cargo deliveries, human and vehicle tracking, and photography industries. "Follow me" modes in new generation drones and autonomous drones in the military are among the concepts that can be put on top of tracking. In the simulation environment we created, different trackers can be tested in different environments. Real-life weather conditions, lighting effects, and many similar factors can be added to the simulation environment. Thus, we make the simulation environment as close to reality as possible. Since we get outputs close to real-life outputs, we will have predictable results when the project starts to continue on a real drone in the future. When the project transitions from the simulation environment to real life, the tests made in the simulation environment are repeated and an optimum tracker is determined again. The drone we used in the simulation environment could only be changed through the sensors. By making hardware improvements on a real drone, trackers that are much more complex and give better results can be tested. By using methods such as ONNX and TensorRT, the optimization of these newly tested trackers can be brought to a much better level. In addition, some of the problems we have experienced in the simulation on real drones can be easily solved. For example, we encountered many problems because we could not change the pitch rate of the drone in the AirSim API we used. Changing a parameter on a real drone would be enough to limit this pitch rate. In this way, the probability of the object we track leaving the frame will be much lower. In addition, a reinforcement learning algorithm can be used for the movements of the drone regardless of simulation and reality. Thus, the movements of the drone while tracking the object can become more accurate and smooth.

References

[1]Luo, Wenhan, et al. "End-to-end active object tracking via reinforcement learning." *International conference on machine learning*. PMLR, 2018.

[2] Luo, Wenhan, et al. "End-to-end active object tracking and its real-world deployment via reinforcement learning." *IEEE transactions on pattern analysis and machine intelligence* 42.6 (2019): 1317-1332

[3] Chen, Gang, et al. "Learning to Navigate from Simulation via Spatial and Semantic Information Synthesis with Noise Model Embedding." *arXiv preprint arXiv:1910.05758* (2019).

[4] Chen, Peng, et al. "Real-time object tracking on a drone with multi-inertial sensing data." *IEEE Transactions on Intelligent Transportation Systems* 19.1 (2017): 131-139.

[5] Erkent, Özgür, et al. "GridTrack: Detection and Tracking of Multiple Objects in Dynamic Occupancy Grids." *International Conference on Computer Vision Systems*. Springer, Cham, 2021.

[6] Zhu, Zheng, et al. "Distractor-aware siamese networks for visual object tracking." *Proceedings of the European conference on computer vision (ECCV)*. 2018.