SmartDrone

Swapnil Aggarwal
Department of Electrical and Computer Engg.
swaggarw@ucsd.edu

Akash Boghani
Department of Electrical and Computer Engg.
aboghani@ucsd.edu

Rishabh Sethunathan
Department of Electrical and Computer Engg.
rsethuna@ucsd.edu

Utkarsh Singh
Department of Electrical and Computer Engg.
usingh@ucsd.edu

Abstract

Drones have been increasingly useful in various applications, including critical situations that are dangerous and inoperable for humans. However, their pervasive use has been limited due the high skill requirement to accurately navigate a drone. The training for a drone pilot certification is very stringent, and thus to fully and effectively aid the society in complete drone integration, it is crucial to design a safe and intuitive way to interact with these systems. We propose a project that implements an object detection algorithm to recognize a specific target in the environment and then integrate a speech to text engine to automatically navigate the drone to the selected object. Real-time object detection is crucial for this application, and since Convolutional Neural Network (CNN) based object detection is computationally demanding, we propose accelerating the object detection by implementing the CNN framework on a GPU. The CNN we propose to use is tiny YOLO. Autonomous navigation and the simple voice based UI have the potential to improve ease-of-use in disaster relief, search and rescue operations and job site safety inspections.

1 Introduction

Drones have, as most technologies, a wide spectrum of usefulness. However, accurate navigation using drones requires extensive training and skill training. There are several stringent trainings, professional & technical licenses and years of experience required for successfully becoming a drone pilot. With so many restrictions, it becomes difficult to intuitively and safely interact as well as accurately navigate the drone. The Economic Impact of Unmanned Aircraft Systems Integration in the United States report shows the economic benefit of UAS integration. AUVSI's findings show that in the first three years of integration more than 70,000 jobs will be created in the United States with an economic impact of more than $13.6 billion. This benefit will grow through 2025 when we foresee more than 100,000 jobs being created and an economic impact of around $82 billion. However, in the immediate aftermath of hurricane Maria, drones were not immediately available because first responders did not have the skills necessary to pilot a drone.

However, If drones were more intelligent, their use would permeate more applications and assist the drone pilots in treacherous situations. One such application is disaster relief. Disaster relief is presently an expensive, personnel-intensive operation. With intelligent drones, we could potentially reduce the entry barrier to such operations, allowing more hands on deck.

In our project, we plan to use highly developed current technologies to make drones easier to accurately navigate. The drone should automatically be able to detect and recognize objects in the environment. We have used You Only Look Once (YOLO) real-time Object Detection algorithm to extract the information about the environment. We further plan to make the user select a detected object by 'speaking' to the controller. We have used Google Speech-To-Text API for extracting the drone command. Finally, the drone should automatically navigate to the selected object without losing context of the object.

The core technology used for our project includes GPU accelerated real-time object detection on video stream being transmitted from the drone simulator. Since, the CNN based object detection is computationally demanding, we propose accelerating the object detection by implementing the CNN framework on a GPU. The object detection algorithm would

run on the Jetson TX2 board and transmit back the environment information to the drone simulator. In addition to it, we are using speech processing for intuitive object selection. The user would be able to command the drone by speaking to the controller. It would be easier for anyone to fly the drone and even assist the drone pilots as well. Finally, the automatic navigation is achieved using the centering algorithm for the drone to fly to the coordinate of the selected object for ground-scene analysis and center the object in the middle of the screen.

2 Technical Material

2.1 Nvidia Jetson

Jetson TX2 is one of the fastest, most power-efficient embedded computing device. It's built around an NVIDIA Pascal^{TM}-family GPU and loaded with 8GB of memory and has 6 CPU cores (4x ARM Cortex A57 Cores, 2x Custom Denver Cores). The four A57 cores that support out-of-order execution to do all the heavy lifting, and Denver cores are a custom VLIW, in-order core made by NVIDIA to perform small tasks efficiently. The Pascal GPU is one of NVIDIA's most efficient GPUs, offering a high performance to power ratio. The Nvidia Jetson 5 operating modes. We chose to run it on Max-N which brings all cores on line with maximum clock frequency and maximum supply voltage, which would give us the best performance.

(a) Developer kit

Mode	Mode Name	Denver 2	Frequency	ARM A57	Frequency	GPU Frequency
0	Max-N	2	2.0 GHz	4	2.0 GHz	1.30 Ghz
1	Max-Q	0		4	1.2 Ghz	0.85 Ghz
2	Max-P Core-All	2	1.4 GHz	4	1.4 GHz	1.12 Ghz
3	Max-P ARM	0		4	2.0 GHz	1.12 Ghz
4	Max-P Denver	1	2.0 GHz	1	2.0 GHz	1.12 Ghz

(b) Operating modes

Figure 1: Nvidia Jetson

2.2 Object Detection

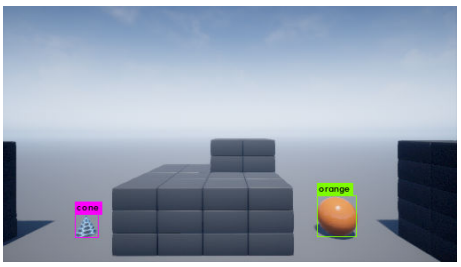
You only look once (YOLO) is a state-of-the-art, real-time object detection system. We are using tiny YOLO which is a lightweight version of YOLO to detect objects in the environment of the drone. Tiny YOLO is a small model, which uses 13 convolutional layers as opposed to the 75 convolutional layers of YOLO, which works well in constrained environments. For a better result, we trained the model to detect oranges and cones which we placed in this environment for the purpose of integrating the system.

Custom training on YOLO requires several hundred images with the information of the co-ordinates of the bounding boxes for each class being detected. Thus, we acquired 200 images from the Airsim environment using the inbuilt camera recorder and individually marked the bounding boxes using Yolo_mark. We trained the YOLO network for 200 epochs before we end up with the final trained weights for the YOLO's CNN network.

We can also extract object data from tiny YOLO in a JSON object. This JSON object contains a list of all objects detected with their name, id, confidence, relative-xy positions and height and width of the bounding box. These details are used for centering the object, details of which are mentioned in a later section.

2.3 Object Tracking

We were worried about context loss of the detected object, so we also tried Deep SORT, a real-time Object tracking algorithm, which utilizes the same weights as YOLO. We were successfully able to implement it on laptop achieving a frame rate of 4.5 FPS. After which, we moved to Jetson TX2, however, the frame rate was dismal hovering around 1 FPS. Moreover, more often than not, the algorithm would make the GPU on board run of out of memory and freeze the Jetson board. Due to the limitation of the hardware, we decided to continue with YOLO Object detection algorithm and come up with Object Centering algorithm so that mostly object remains in the center of the frame.



(a) Bounding Boxes


```
{
  "frame_id":4542,
  "object":[
  {"class_id":1, "name":"orange", "relative_coordinates":
  {"center_x":0.704870, "center_y":0.799382, "width":0.061748,
  "height":0.114636}, "confidence":0.999094},
  {"class_id":0, "name":"cone", "relative_coordinates":
  {"center_x":0.283606, "center_y":0.857091, "width":0.039221,
  "height":0.078673}, "confidence":0.999528},
}
```


(b) JSON object

Figure 2: Object Detection with YOLOv3

2.4 Drone Simulator

During the course of the project, we tried following drone simulators. We have discussed in detail various drone simulators that we tried, what worked/didn't work and what we concluded from the above experiments.

2.4.1 DJI Simulator

DJI Simulator is the official flight simulator software provided by DJI for training. It offers a fairly comprehensive demo in the free version, and features rich environments and realistic controls.

The main disadvantage of the DJI Flight Simulator is that it does not provide any support for automated control in its free version. Moreover, controlling the drone by using simulated key-presses seemed too slow and prone to bugs with respect to accurate drone control. We then dropped the DJI Flight Simulator due to lack of customization options.

2.4.2 ArduPilot

We also tried ArduPilot, which is a free open source drone simulator software available on github. It provides a set of interfacing tools and commands to perform SITL (Software In-The Loop) Simulation of a 3D Drone Simulator, such as FlightGear

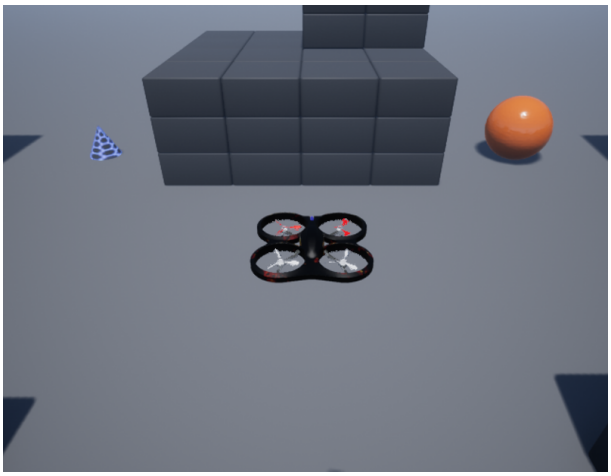
This ArduPilot Command System can be mapped to simulate drone controls in the FlightGear simulator, using the arducopter library. Arducopter provides a simple library and drone model for drone control, with basic commands such as arming the throttle, takeoff, setting velocity vector, etc. However, not only is the support of Ardupilot+Flightgear very poor, but also the commands are not generic and were creating various roadblocks while trying to interface it with the Jetson board. Hence, we also dropped the Ardupilot+Flightgear combo.

2.4.3 Microsoft AirSim

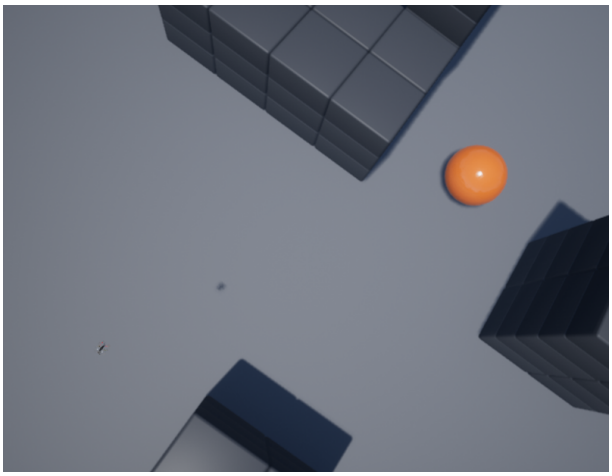
Microsoft AirSim (Aerial Informatics and Robotics Simulation) is a relatively newer, freely available simulator for drones, cars and more. Developed for and by open source developers, AirSim provides a unique and seamless 3D simulation interface using the Unreal Engine 4 (with an experimental Unity release also available). It is cross platform and supports hardware-in-loop with popular flight controllers, such as PX4, for physically and visually realistic simulations.

The platform interfaces with common robotic platforms, such as a Robot Operating System (ROS), and comes pre-loaded with a commonly used multirotor robotic model (drone), a generic sports utility vehicle for autonomous driving simulation, and several sensors (IMU, LiDAR, GPS, Barometer, etc). In addition, the platform enables high-frequency simulations with widely supported protocols, such as MavLink. Its cross-platform (Linux and Windows) and open-source architecture is easily extensible to accommodate diverse new types of custom autonomous vehicles, hardware platforms, and software protocols. This architecture allows users to add custom autonomous system models and new sensors to the simulator quickly.

Machine learning has become an increasingly important artificial intelligence approach in building autonomous and robotic systems. One of the key challenges with machine learning is the need for massive data sets—and the amount of data needed to learn useful behaviors can be prohibitively high. Since a new robotic system is often non-operational during the training phase, the development and debugging phases with real-world experiments face an unpredictable robot.



(a) Drone View



(b) Aerial view

Figure 3: Microsoft AirSim

AirSim solves these two problems, the need for large data sets for training and the ability to debug in a simulator, by providing a realistic simulation tool for designers and developers for seamless generation of the amount of training data they require. In addition, AirSim creates accurate, real-world simulations by leveraging current game engine rendering, physics, and perception computation. Together, this realism, based on efficiently generated ground-truth data, enables the study and execution of complex missions that are time-consuming and/or risky in the real-world. For example, collisions in a simulator cost virtually nothing, yet provide actionable information for improving the design of the system.

Moreover, AirSim exposes APIs so you can interact with the vehicle in the simulation programmatically. You can use these APIs to retrieve images, get state, control the vehicle and so on. The APIs are exposed through the RPC, and are accessible via a variety of languages, including C++, Python, C# and Java.

These APIs are also available as part of a separate, independent cross-platform library, so you can deploy them on a companion computer on your vehicle. This way, one can write and test their code in the simulator, and later execute it on the real vehicles.

For the purpose of this project, we utilized AirSim to simulate a multirotor vehicle (drone) inside the Blocks environment, which is provided as a default environment with the AirSim source code. The blocks environment consists of a few objects which can be utilized to test out our autonomous drone functionality. We made use of two objects, an Orange ball and a color changing Cone.

For drone control, we used Python APIs to interface the drone with the automated control system, which runs as a Python script. The Python APIs can be installed by downloading and installing 'airsim' using the python package installer - pip. The APIs provide many functionalities such as current state of the drone (including sensor values and current position), arm/disarm, takeoff/land, and various drone movements using position, velocity, angle, etc. We made use of velocity based movement using the 'moveByVelocityAsync' functionality call.

2.5 Flask

Flask is a micro web framework written in Python, based on Werkzeug and Jinja 2. Werkzeug is a utility library for the Python programming language, in other words a toolkit for Web Server Gateway Interface (WSGI) applications, and is licensed under a BSD License.

It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Jinja is a template engine for the Python programming language and provides for the templates that are evaluated in a sandbox. Applications that use the Flask framework include Pinterest, LinkedIn and the community web page for Flask itself.

We utilize flask in order to setup a MJPG image stream from the drone's first person camera output to the custom trained TinyYoloV3 running on the Nvidia Jetson TX2. The Python script which runs for the video stream specifies the frame and the type of response object returned by the MJPG stream. We also utilize OpenCV for encoding the AirSim camera output to JPEG format before sending it to the Flask streamer.

2.6 Speech Recognition

During the course of our project we tried the following speech recognition:

2.6.1 Pocket Sphinx

PocketSphinx is one of Carnegie Mellon University's open source large vocabulary, speaker-independent continuous speech recognition engine. It works offline since the acoustic model is fairly compact and takes only 97.8 MB disk space. Moreover, it is free to run on embedded platforms. However, it achieves an accuracy of 45% on Common Voice data which is bad considering SmartDrone project finds usage in critical applications like Disaster Relief.

2.6.2 Mozilla DeepSpeech

DeepSpeech is an open source Speech-to-text engine, using a model trained by machine learning techniques based on Baidu's Deep Speech. It uses Google's Tensorflow library in the backend to implement the deep speech architecture. It achieves an accuracy of 80% on Common Voice data which is impressive, however, the acoustic model is pretty large (greater than 2 GB) which makes it infeasible for the scope of embedded system application.

2.6.3 PicoVoice Cheetah

Cheetah is is an on-device speech-to-text engine developed using Picovoice's proprietary deep learning technology. It is offline and runs locally without an internet connection, which helps in protecting users' privacy. Cheetah achieves higher accuracy around 81% on Common voice data, and the acoustic model takes only 71.3 MB space which is ideal for embedded system application. However, the embedded model (cross-compiled libraries for ARM processor) requires commercial license and due to budget constraints, it was not feasible for the scope of our project.

2.6.4 Google Voice API

Since, none of the above mentioned speech to text engines were working efficiently on the Jetson TX2 board due to ARM compatibility and lower accuracy, we have used Google Voice API for accurately recognizing drone commands and maneuvering the drone. Google Voice API is an online Speech-to-Text engine which converts audio to text by applying powerful neural network models in an easy-to-use API. The synchronous recognition performs recognition on audio data provided within a gRPC bi-directional stream. Streaming requests are designed for real-time recognition purposes, such as capturing live audio from a microphone and it retrieves results after all audio has been sent and processed. The 'speech.recognize' method utilizes JSON for providing the configuration information (i.e. information to the recognizer that specifies how to process the request) and audio object name. The result obtained is also a JSON object having information about sequential list of transcription results corresponding to sequential portions of the audio source.

Moreover, we are using FLAC to encode the audio data to send to the API. FLAC stands for Free Lossless Audio Codec, an audio format similar to MP3, but lossless, meaning that audio is compressed in FLAC without any loss in quality. This is similar to how zip works, except with FLAC you will get much better compression because it is designed specifically for audio.

The speech_recognition.Recognizer() method is used to recognize the audio from the microphone. Initially, we start of by calibrating the recognizer energy threshold for ambient energy levels from the source using adjust_for_ambient_noise(source) method. Once, the threshold values have been set for ambient noise, then we forward the audio output from the Recognizer() method and request the Google speech-to-text API to transcribe the audio file. The output received from Google voice API is an JSON object from which we extract the text and process it into a list.

2.7 Object Centering

After extracting positional data from the json object obtained from YOLO, we calculated the xy displacement from center, and decide the directions it is supposed to move in. We decided that to be considered at the object, the object's bounding box should cover 70% of the screen. By this metric we decide if the drone is supposed to move forward or backward. After deciding the directions we are moving in, we provide directional velocities to the drone. The velocity

is calculated by $velocity = (height + 0.3) \times \nu$, where ν is the velocity factor i.e., the maximum velocity that can be attained by the drone. This makes sure that the velocity of the drone is between ν and zero.

3 Milestones

Task	Details	Timeline	Status
Laptop demonstration of object recognition on video	YOLO object recognition was successfully demonstrated on laptop showing a frame rate of 7.5 FPS. For tiny YOLO, we achieved an FPS of 30.	04/23	Completed on 04/23
Comparative evaluation	Compared different speech processing algorithms and flight simulators and tested the Jetson board.	04/27	Completed on 05/02
Demo of real time object detection on Jetson	YOLO object recognition was successfully demonstrated on the jetson board showing a frame rate of 5 FPS. For tiny YOLO, we achieved an FPS of 30.	05/02	Completed on 05/02
Hardware acceleration of speech processing on Jetson	We had issues while setting up PicoVoice for speech processing on Jetson due to licenses, so we changed the task to run it as a standalone component, and successfully ran it on a linux environment.	05/09	Completed on 05/09
Integrating object recognition with speech processing	Same as 'drone control using voice commands' below.	05/14	Completed on 06/09
[NEW] Integration of object detection and drone simulator	We successfully set up live streaming from the simulator, transmitted it to the Jetson board, and ran object detection on it in real time.	05/14	Completed on 05/14
Perform post-detection image centering on target object	Used reliative-xy coordinates of object from json output of Yolo to develop an algorithm for centering the object in the frame of drone feed.	05/16	Completed on 06/05
Interfacing the controls on drone	Interfaced controls of drone using python API.	05/23	Completed on 05/13
[NEW] Demonstration of Object tracking on laptop	DeepSORT object tracking was successfully demonstrated on the latptop showing a frame rate of 4.5 FPS. For the Jetson, we achieved an FPS of 2.	05/23	Completed on 05/22
Drone control using voice command	Integrated google voice API for speech processing with AirSim for drone control using voice commands.	05/30	Completed on 06/09
[NEW] Training Tiny YOLO for custom dataset	Trained tiny YOLO to detect oranges and cones to improve performance.	06/06	Completed on 06/06
Complete system integration	Complete system integration.	06/06	Completed on 06/09

4 Conclusion

In conclusion, we were successfully able to prototype the idea of being able to let anyone fly the drone without proper training and even assist the experienced drone pilots to tackle difficult situations. We were successfully able to accelerate the YOLO Object Detection algorithm on real-time stream video on Jetson TX2 board. In addition to it, we tried several speech processing architectures, however, we found out that running deep neural network based Speech-To-Text engine on an embedded board having limited hardware resource is not the most efficient solution, so we moved to an online alternative i.e. Google Voice API. Moreover, we implemented Object Centering algorithm to navigate the drone and maintain the selected object in the frame. Since, the Object detection algorithm was not able to recognize the object when the drone closes on the object and the drone jitters due to sudden direction change, we trained tiny YOLO to detect oranges and cones to improve the performance. Finally, the system integration of each individual components such as real-time Object Detection on streaming video through Flask, Drone Control using voice command, post-detection Object Centering on target object took most of our last few weeks of the quarter.

The major learning from the project is that systems integration with off-the-rack solutions is less efficient than building a solution from scratch. Moreover, there are limited options for offline speech processing specifically for embedded boards having ARM processor that give acceptable accuracy. In addition to it, we found out that training on specific dataset significantly improves performance. In spite of these challenges, we were able to achieve a working proof of concept of Smart Drone because of effective use of existing technologies.

There remain several areas in which the project could improved on. The most obvious improvement would be to build optimized CUDA code from the ground up, rather than using off the rack technologies. This would improve the performance significantly. The object tracking algorithm could be trained for realistic environment with humans. In addition to it, collision avoidance algorithm could be incorporated with the project. The navigation could also be improved by using a well planned algorithm rather than rudimentary form of Object Centering algorithm. Moreover, running the speech detection offline by cross-compiling it for ARM processor could also one of the future work. Finally, we have provided proof-of-concept on software using drone simulator, however, it could easily be implemented on commercially available drones, as all we need is video feed from the drone and compatibility with a programmable controller.

5 Acknowledgements

We would like to sincerely thank Professor Ryan Kastner and Peter for giving us an opportunity to work on the above topic, as well as providing us the correct insight and learning as we progressed throughout the quarter.

We would also like to thank Eric for helping us out with setting up the drone simulator. It was a crucial challenge to complete the integration of our drone and the suggestions have gone a long way in the achievement of our success.

Last but not the least, we would also like to thank Prof. John Eldon for lending us the Jetson TX2 board which provided as the base of our primary YOLO algorithm.

Special thanks to the CSE department at UC San Diego for giving us an opportunity to work on embedded prototyping in CSE237D.

6 References

- Joseph Redmon, Santosh Divvala, Ross Girshick & Ali Farhadi, **"You Only Look Once: Unified, Real-Time Object Detection"**, `www.pjreddie.com/media/files/papers/yolo.pdf`
- Joseph Redmon & Ali Farhadi, **"YOLOv3: An Incremental Improvement"**, `www.arxiv.org/pdf/1804.02767.pdf`
- Shital Shah, Debadeepta Dey, Chris Lovett, Ashish Kapoor & Jim Piavis, **"Microsoft AirSim (Aerial Informatics and Robotics Simulation)"**, `microsoft.github.io/AirSim`.
- **"Real-time Multi-person tracker using YOLO v3 and deep_sort with tensorflow"**, `www.github.com/Qidian213/deep_sort_yolov3`
- **Nvidia Jetson SDK**, `developer.nvidia.com/embedded/jetpack`
- Carnegie Mellon University, **"PocketSphinx"**, `cmusphinx.github.io/`
- **"Cheetah by Picovoice"**, `www.picovoice.ai/products/cheetah.html`
- Google Cloud, **"Cloud Speech-to-Text"**, `cloud.google.com/speech-to-text`

- Alexander Mordvintsev & Abid K, **"OpenCV-Python"**, opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html.
- **"DJI Flight Simulator"**, www.dji.com/simulator
- **"ArduCopter"**, www.ardupilot.org/copter
- CRN.com **"The Challenges Of Using Drones In Puerto Rico After Hurricane Maria"**, https://www.crn.com/news/channel-programs/video/300106456/the-challenges-of-using-drones-in-puerto-rico-after-hurricane-maria.htm