Jake Teeter, Andrew Nyland, Marcos Ordonez

Dr. Andy Nere

CS 497 – Computer Vision

May 2, 2023

Final Report – Gesture Controlled Drone

In this project we investigate implementing computer vision driven controls of a DJI Tello drone. While our initial plan was to pursue tracking a flight path via optical flow recorded on a separate video, we ended up implementing gesture-based control of the drone in flight. Our project heavily relies on implementation details and chaos of running a real, live device, but we also learned many of the troubles of dealing with situation specific training and controlling of a drone.

## Overview of our project

### Learning the drone

Spending time learning how the drone flies took longer than expected but it was fun to see the abilities it has. This also gave us additional ideas for expansive projects, but many of these lay outside of the time allocated. It showed us that its array of sensors were more accurate than expected, but unfortunately battery life was quite limited.

### Original plan

Our original goal was to record a flight path with a camera and then, temporally separately, have the drone follow this path. Optical flow was the chosen methodology but rotation speed and camera quality of the drone seemed to be an issue. Optimizing the optical flow parameters, as well as persistent tracking points becoming either irrelevant or causing lag in the processing, began to be an excessive problem. As we experimented with flights and live streaming we became more interested in seeing the drone fly by live command.

### First CNN approach

We wanted to build a system from scratch, and planned to use a CNN to recognize single hand gestures which would be represented as states. These states could be used with simple code to design control logic. Our goal was to be able to stand in front of the drone and conduct its movements, and ultimately possibly have it follow us.

In order to have generic person recognition and options for customization later in our process, we investigated recording our own training data. Scripts were built to record live training data as useful data structures on our computer, and data cleaning processes and tasks were designed and completed. After about 12000 images were recorded from the drone, then culled to only the images that we wanted, then preprocessed into a larger set of varied images (multiple approaches to generating color and environment agnostic datasets were attempted), we found this task to be more difficult than expected.

The lighting in the classroom that we took our training images of was not optimal, and processing speed became an issue. Air gusts in the building would randomly take the drone on paths that were not intended. We had recorded a few different sets of gesture commands, ranging from single hand movements to full body movements such as dancing and jumping, but sequential training and inference was a new problem. While we had moderate success, we decided to attempt implementing and customizing already existing models.

### Gesture recognition with MediaPipe

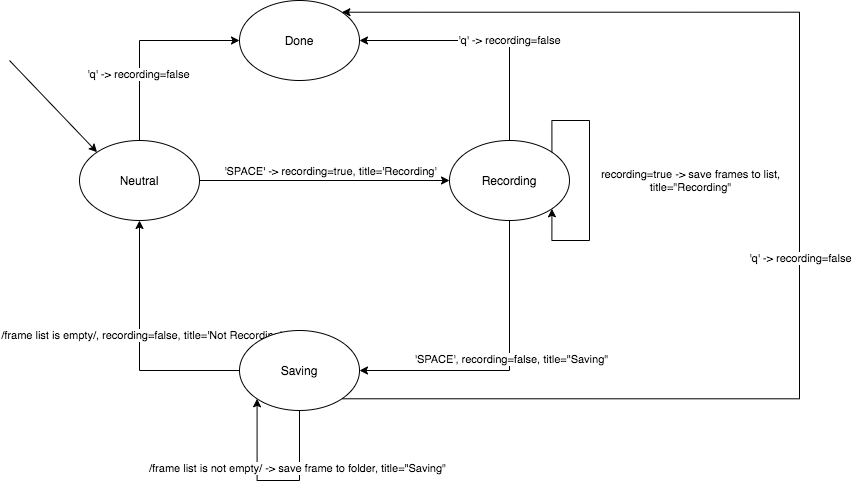
In our research we had found existing projects dealing with using American Sign Language to control the drone, so we decided to take our swing at customizing this approach. These models relied heavily on Google’s MediaPipe project, which as an extremely accurate pre-trained model for hand and posture gestures seemed like a viable option. It proved far more accurate than we had expected. However, module support and computational power again reared its ugly head.

Implementation

Much of our project necessitated building support scripts and considering the full process of what we were attempting. As mentioned, this included building scripts to handle cleaning of our data at multiple steps of the process.

### Cleaning and recording data

We created scripts create.py and preprocessing.py which allow us to extrapolate minimal recorded training images into multiple, as well as blurring and color correction. Our recording script saves images into single pictures for each frame for easy management later, especially without the drone. Below is a state machine diagram of our recording script.



To allow for easy recording (we wanted to run the script and only focus on aiming the drone for optimal training image acquisition), we kept track of a counter (in counter.txt) and each sequential recording run was saved to a folder with this number.

### Problems that arose

As is typical in engineering, logical and logistical errors and frustration showed up while we thought we were doing well.

#### Cleaning data

It turns out 30fps leads to many images existing, and while we wanted this so that we could consolidate the recorded data later instead of not having enough, this task was extensive. Convoluting the images into generic training data also involved more theory than we had time or energy for.

Debouncing

Asynchronistic behaviors of the drone’s API appeared to not return as expected (and as they did in our early testing), and this led to the controller computer and drone interpreting commands faster than we expected. This caused the drone to become confused and block our scripts from sending commands as we wanted, and no solid or repeatable solutions existed easily.

#### Battery Life

The drone’s limited battery capacity and capability certainly gave us a few problems. Both in testing and running live programs. Testing with solely the camera and no flight could last for a little while, but when the system began to fail it caused us a few headaches and wasted time (as well as conceptual progress and flow).

#### Software support

As our endeavors became more ambitious and we started to rely on or at least be inspired by existing published work, software compatibility on our old and various operating system computers became an issue. At first, macOS was our only fully viable option – for the best of reasons the Windows firewall decided to block connections to the drone (it’s an “untrusted network”, and only supports UDP) unless prayed to correctly. We found that it could work correctly with Windows, minus native Tensorflow, but no connections were viable into a Linux virtual machine. Mediapipe was trouble on our old devices, whether compiling natively or via package managers. Ultimately it was found that PyCharm could make everything work, but due to severe overhead it was still limited in real-time performance.

#### Physical strength

While very uncommon (only two), broken propellers gave us problems that were more impactful than they likely should have been. The significant one was due to testing hardware capabilities exposed by the API, but the other one delayed us significantly. Without any indication as to the cause, a propeller broke off by maybe a few millimeters on one end, and suddenly the drone failed to lift off from the ground for a long time. We attempted to diagnose almost every other possible cause, including restarting both the drone and commanding laptop.

#### Unstable air systems in our academic building

Early on we determined that the positional sensors provided with the drone are more accurate than they should be for the price and build quality, but uneven and unpredictable air distribution systems on the interior of our building still made their mark. As our gesturing to the drone requires the conductor to stand nearby to directly in front of the drone, random air pushes interrupted this. While occasionally inconvenient, it gave us interesting insight into how weird airflow in this building is.

Conclusion

In short, we learned how many struggles and options can exist in an applied project like this. While theory of such may have helped us with some problems, and certainly showed us a few paths we considered then determined were not worth investigating.

We tried our own methodology and approaches but these were more limited in performance than preferred and that which time allowed for, so we branched out and did some additional research. If this was a semester long project this scenario would be different, but as it were we ended up finding that our focus was on an implementation that consisted of physical interaction and hardware all the way to high-level software. Our learning ended up being not about the end goal but about the adventure to get there.

Sources

MediaPipe:

<https://google.github.io/mediapipe/>

<https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/>

<https://www.section.io/engineering-education/creating-a-hand-tracking-module/>

<https://github.com/kinivi/tello-gesture-control>

<https://github.com/Kazuhito00/hand-gesture-recognition-using-mediapipe/blob/main/README_EN.md>

<https://medium.com/analytics-vidhya/mediapipe-object-detection-and-box-tracking-82926abc50c2>

Watershed: <https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Watershed_Algorithm_Marker_Based_Segmentation.php>

Watershed Algorithm: <https://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_Image_Watershed_Algorithm_Marker_Based_Segmentation.php>

CNN:

<https://www.analyticsvidhya.com/blog/2022/03/understand-the-realtime-image-classification-cnn/>

<https://ocw.mit.edu/courses/18-065-matrix-methods-in-data-analysis-signal-processing-and-machine-learning-spring-2018/resources/lecture-32-imagenet-is-a-cnn-the-convolution-rule/>

YOLO:

<https://medium.com/@nnamaka7/detect-hand-gestures-with-yolov5-756ac6bef0bc>