Title	Real-Time Human Action Recognition for Drone Control
Mentor and Company	Rick L. Hudson Sr. Project Manager School of Data Science UNC-Charlotte
Dates of Internship	Jan 20th to May 10th, 2021
Student Name	Abhijeet Pokhriyal
Semester	Spring 2021
Course	DSBA6400

Table of Contents

Executive Summary

In this project we built an embedded real-time human action detection and classification system which was deployed on a Nvidia Jetson Nano Microcomputer and subsequently mounted on a drone. Main objective was developing an end-to-end fail-safe technology that allows a pilot to control the drone even if it loses contact with the remote control using only mounted camera and gestures. This opens possibilities for the drone to fly beyond the remote-control range and acts as a capability demonstration for building more complex systems on top of existing product. Major expected outcomes were that (i) We should have an acceptable classification accuracy, (ii) Real-time performance, (iii) Memory and Compute Efficiency. We were reasonably able to accomplish all three of these expectations even though risks and issues were anticipated. For example, we faced a resource constrained hardware with only two gigabytes of RAM and needed to operate out of it memory intensive computer vision models. Also, he Dataset we had was of only 170 videos each being around 2-5 seconds in length. This was expected to prove to be too small when developing deep learning models like LSTM. In total there were four dynamic gestures to be recognized. Each gesture was a movement of hands that was to be detected and then classified into the correct category.

Approach implemented was a 2D Discriminative Bottom-Up method. In this approach, Waveshare Camera module was used to provide input image stream using Nvidia JetCam interface. We built a data pipeline using NVIDIA GStreamer to pre-process the images and move them from the source to sink, which in this case were a sequence of modules. Pose estimation module was used to extract key-point positions like that of wrists, shoulders, elbows etc. On the extracted positions we then performed exploratory analysis to search for patterns and then moved onto feature engineering. Our final feature set included angles between body-parts, like shoulder-elbow-wrist for each frame of the camera stream. When building the LSTM model, we experimented by changing the window size (number of continuous frames) and the number of hidden units. We found that the model performed very poorly, primarily due to a very low sample size and considerable decrease in information present in each video when reduced to via feature extraction and engineering.

After failure of the LSTM Model, we switched to Tensorflow lite and Static gestures instead of dynamic movement-based gestures. This allowed us to move away from LSTMs to simpler rule-based model which used relative positions and distances between extracted key-points to determine which gesture was being performed.

Once we had our model ready, to interface with a drone we used dronekit-sitl (Software in the loop) for drone simulation and dronekit-python, for drone vehicle control. For visualizing drone movement, we leveraged Mission Planner. We developed a client-server architecture where Jetson Nano acted as a server, identifying the gestures and sending then onto the client (laptop) which also acted as ground control for the drone. We were successfully able to control the drone to perform simple flight manoeuvres in real-time.

Introduction

Project Objective:

Goal of the current project was to build an embedded real-time human action detection and classification system deployed on a Nvidia Jetson Nano chip used to control a drone using gestures.

Our main reasons for pursuing this goal were:

1. **Create a fail-safe mechanism:**

In case of weak signals or complete loss of communication signals, gesture control will allow the pilot to operate the drone using only cameras without depending on the remote control.

2. **Search and Rescue Missions:**

Because of gesture controls relying only on the cameras, drone will be able to operate in areas beyond the range of remote control. This has direct application in critical missions like search and rescue where the rescue team might be stationed remotely.

3. **Capability Demonstration:**

Pursuing this goal lets us build a Minimum Viable Product (MVP) that can then be used as a base for a more complex system.

From a data science perspective, we wanted to leverage transfer learning to incorporate computer vision technologies in developing a controller to help facilitate the overall goal of autonomous drone flights.

Major Technical Expectations were

i. **An acceptable classification accuracy**

We wanted a system that minimizes the number of false positives so as to avoid improper navigation commands being sent to the drone controller.

ii. **Real-time performance**

We wanted a system that performed in real-time because only then could we efficiently use it to control the drone.

iii. **Memory and Compute efficient system**

Since Jetson Nano was known to be resource constrained, our system had to be efficient both memory and compute wise. From compute perspective we wanted to keep the number of pieces in our pipeline minimum also to maintain a decent framerate for real-time performance. accuracy in real-time. We will be demonstrating the system with a successful test flight of the drone.

Literature review

Human action recognition can be practiced primarily with three methods: Using

i. Glove-based wearable devices
ii. 3-dimensional locations of hand keypoints and
iii. raw visual data

The first method comes with the obligation of wearing an additional device with which lots of cables come even though it provides good results in terms of both accuracy and speed. The second, on the other hand, requires an extra step of hand-keypoints extraction, which bring additional time and computational cost.

Lastly, for (iii), only an image capturing sensor is required such as camera, infrared sensor or depth sensor, which are independent of the user.

In current project we will be adopting (iii) methodology and within (iii) we will be focusing on what has been described as 2D Discriminative Bottom-Up approach.

2D Discriminative Bottom-Up approach

Top-down methods start from high-level abstraction to first detect persons and generate the person locations in bounding boxes. Then pose estimation is conducted for each person. In contrast, bottom-up methods first predict all body parts of every person in the input image and then group them either by human body model fitting or other algorithms. Note that body parts could be joints, limbs, or small template patches depending on different methods. With an increased number of people in an image, the computation cost of top-down methods significantly increases, while keeps stable for bottom-up methods. However, if there are some people with a large overlap, bottom-up methods face challenges to group corresponding body parts.

The main components of bottom-up HPE methods include body joint detection and joint candidate grouping. Most algorithms handle these two components separately. The library that we first used – OpenPose - uses a nonparametric representation, which they refer to as Part Affinity Fields (PAFs), to learn to associate body parts with individuals in the image. This bottom-up system achieves high accuracy and realtime performance, regardless of the number of people in the image. They demonstrate that a PAF-only refinement rather than both PAF and body part location refinement results in a substantial increase in both runtime performance and accuracy.

The model that we shifted to later for memory and computer efficiency adapts multi-person pose estimation architecture to use it on edge devices. Their network design and optimized post-processing code runs at 28 frames per second (fps) on Intel® NUC 6i7KYB mini PC and 26 fps on Core i7-6850K CPU. The network model has 4.1M parameters and 9 billions floating-point operations (GFLOPs) complexity, which is just ~15% of the baseline 2-stage OpenPose with almost the same quality.

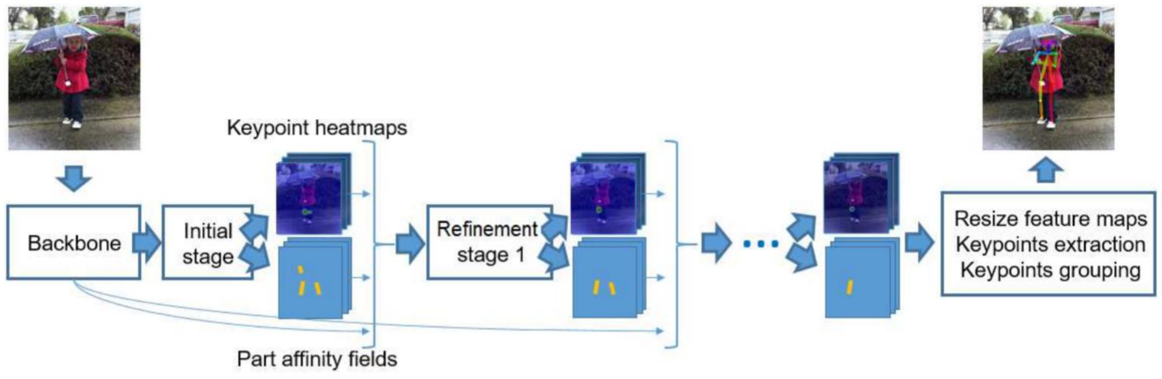
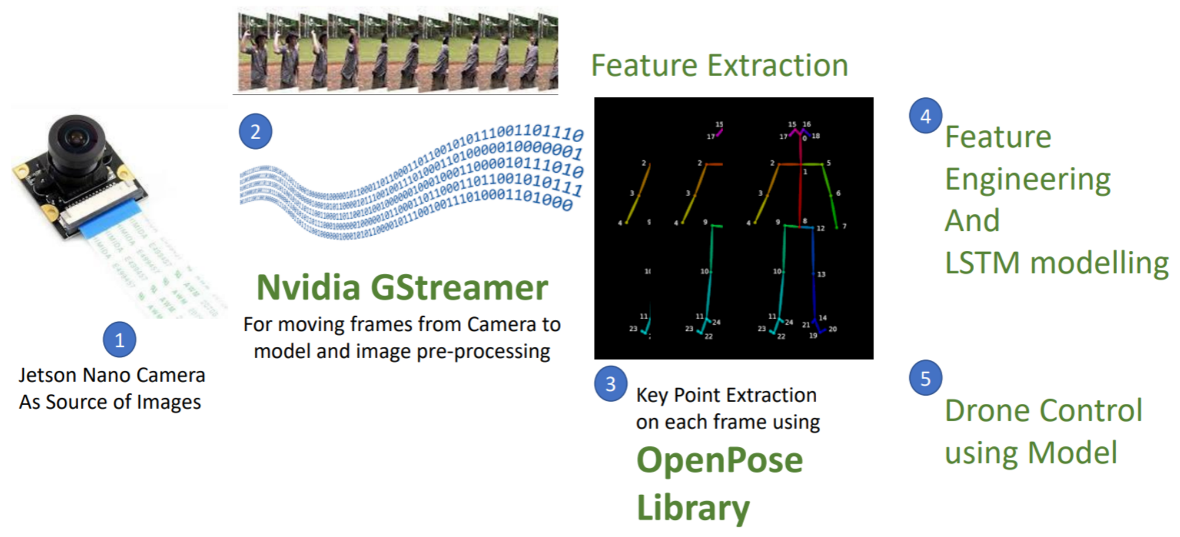


Figure 1: OpenPose pipeline.

Methods


1
Jetson Nano Camera
As Source of Images
2
Nvidia GStreamer
For moving frames from Camera to model and image pre-processing
Feature Extraction
3
Key Point Extraction on each frame using
OpenPose Library
4
Feature Engineering And LSTM modelling
5
Drone Control using Model

Overall, the method could be divided into five parts

1. Setting up the hardware and camera
2. Building Data Pipeline using NVIDIA GStreamer
3. Feature Extraction and Engineering
4. Model Building
5. Drone Controller Development

1. Setting up the hardware and camera

Jetson Nano comes with pre-built image that can be burnt onto a SD card and installed onto the board. But the problem is that many of the software on the image are either outdated or not needed by our approach and therefore unnecessarily taking resources on an already resource constrained environment.

We took a base ubuntu image for ARM architecture and installed all the required libraries needed. This led to some issues for us because quite a few of the prerequisites had to be built from source for which we needed extensive list of build tools installed and updated.

We also faced issued getting the camera interfaced with the board due to versioning issues of OpenCV, Python and Jetson Nano Camera API. These issues were a glimpse into the difficulty of building embedded AI systems because these systems are borderline Application Specific Architectures and therefore don't have implicit support from the Open Source community.

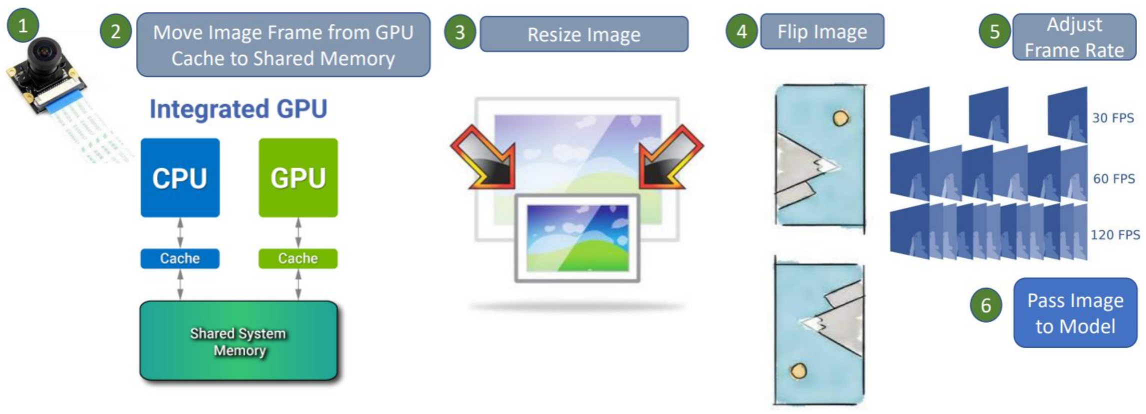
2. Building Data Pipeline using GStreamer

Once we had our hardware setup and the camera working, we then built a data pipeline using NVIDIA GStreamer. GStreamer is a pipeline-based multimedia framework that links together a wide variety of media processing systems to complete complex workflows. For instance, GStreamer can be used to build a system that reads files in one format, processes them, and exports them in another. The formats and processes can be changed in a plug and play fashion.

GStreamer processes media by connecting a number of processing elements into a pipeline. Each element is provided by a plug-in. Elements can be grouped into bins, which can be further aggregated, thus forming a hierarchical graph. This is an example of a filter graph.

Elements communicate by means of pads. A source pad on one element can be connected to a sink pad on another. When the pipeline is in the playing state, data buffers flow from the source pad to the sink pad. Pads negotiate the kind of data that will be sent using capabilities.

NVIDIA has built on top of GStreamer to provide support for its Embedded AI Chips.



We used GStreamer for

1. Moving Image Frame from GPU Cache to Shared Memory

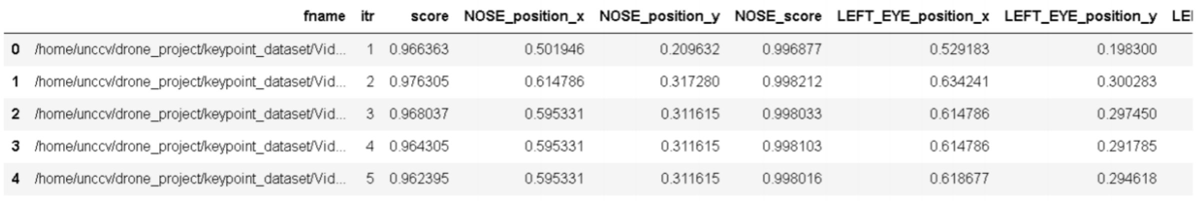
    a. Image frames once captured from the camera are moved into the GPU cache. For our processing we moved it to Shared System memory to allow it to be read by our python programs

2. Image Pre-processing
    a. We then pre-processed the image stream by resizing them to the size needed by the machine learning model.
    b. We also added steps for flipping the image and adjusting the frame rate.
    c. Frame rate was adjusted because the models we were using were not meant for high framerates that the camera supported like 60fps or 120fps.

3. Feature Extraction and Engineering

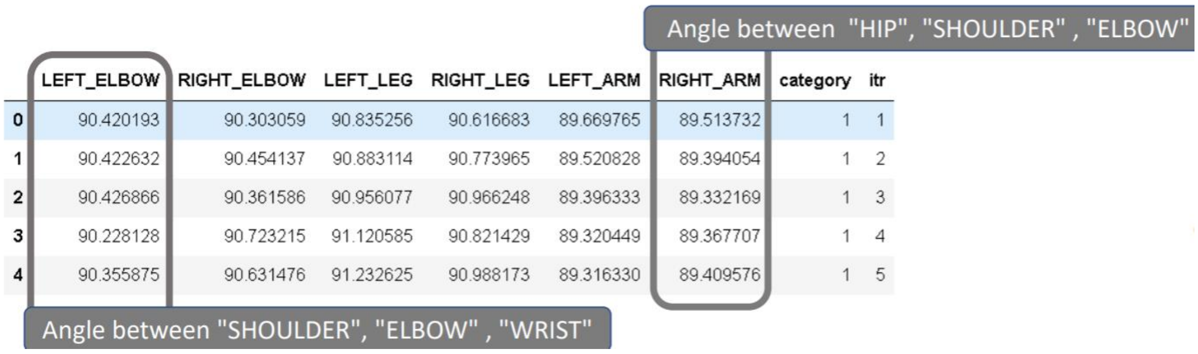
Once we had our data pipeline ready, we then integrated into it the pose estimation model. We had first decided to go with OpenPose library. But we soon realized that it consumed too much RAM and led to Out of Memory Exceptions. We then switched to Tensorflow Lite which provided a slightly less accurate model but one that was significantly smaller and only needed around 100MB RAM to operate.

Using the pose estimation model we were able to extract keypoints representing different body parts in real-time for each frame along with the confidence score about the overall accuracy of those points.



	fname	itr	score	NOSE_position_x	NOSE_position_y	NOSE_score	LEFT_EYE_position_x	LEFT_EYE_position_y	LE
0	/home/unccv/drone_project/keypoint_dataset/Vid...	1	0.966363	0.501946	0.209632	0.996877	0.529183	0.198300	
1	/home/unccv/drone_project/keypoint_dataset/Vid...	2	0.976305	0.614786	0.317280	0.998212	0.634241	0.300283	
2	/home/unccv/drone_project/keypoint_dataset/Vid...	3	0.968037	0.595331	0.311615	0.998033	0.614786	0.297450	
3	/home/unccv/drone_project/keypoint_dataset/Vid...	4	0.964305	0.595331	0.311615	0.998103	0.614786	0.291785	
4	/home/unccv/drone_project/keypoint_dataset/Vid...	5	0.962395	0.595331	0.311615	0.998016	0.618677	0.294618	

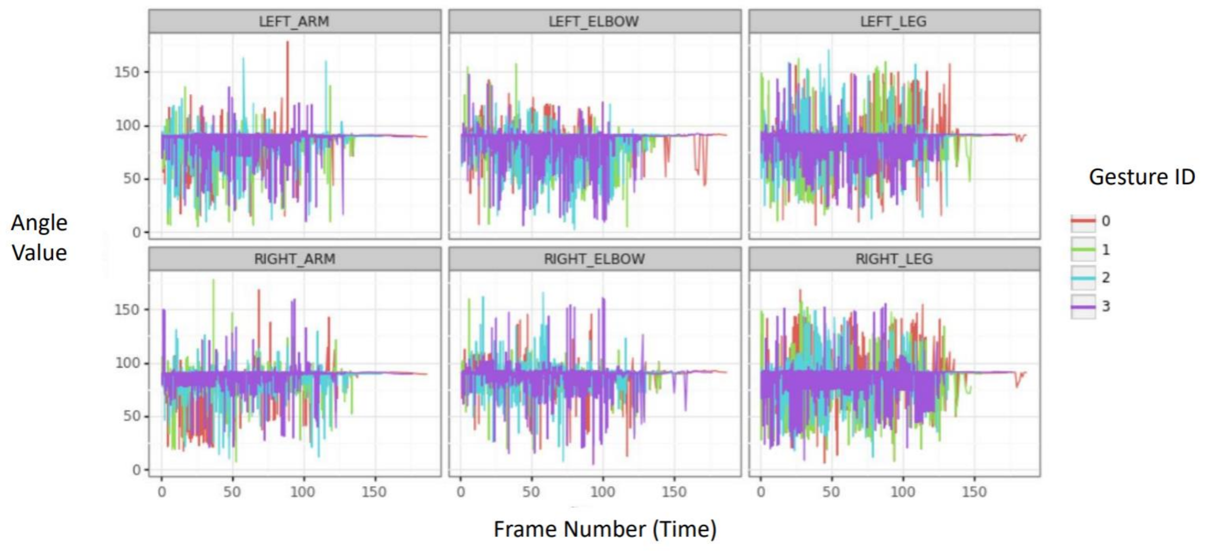
We then performed feature engineering and transformed keypoint positions into angles between different body parts. For example points Shoulder,Elbow and Wrist were used to come up with one value representing the angle between these three body parts.


Angle between "HIP", "SHOULDER" , "ELBOW"

	LEFT_ELBOW	RIGHT_ELBOW	LEFT_LEG	RIGHT_LEG	LEFT_ARM	RIGHT_ARM	category	itr
0	90.420193	90.303059	90.835256	90.616683	89.669765	89.513732	1	1
1	90.422632	90.454137	90.883114	90.773965	89.520828	89.394054	1	2
2	90.426866	90.361586	90.956077	90.966248	89.396333	89.332169	1	3
3	90.228128	90.723215	91.120585	90.821429	89.320449	89.367707	1	4
4	90.355875	90.631476	91.232625	90.988173	89.316330	89.409576	1	5

Angle between "SHOULDER", "ELBOW" , "WRIST"

We then performed some exploratory analysis to see if there were any patterns of significance on how these angles changed with time as each gesture was performed.

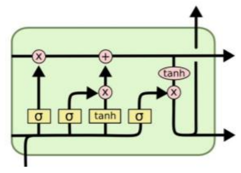


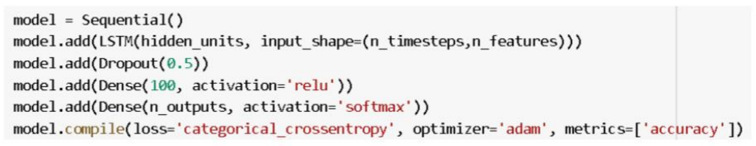
We could see that quite a few of the angles fluctuated around 90 degrees during the duration of the gesture. While some angles like left and right leg, had angles that were obtuse at times for some of the gestures.

4. Model Building

Once we had extracted the features, we proceeded with the modelling bit. We then split the dataset into timesteps. This was done by grouping on each video and then extracting WINDOW_SIZE number of continuous frames as part of one sample. In our experimentation we varied the WINDOW_SIZE from 50 to 100 in steps of 10. Total dataset size formed in this way reduced to only around 600 samples (depending on the window size), each sample being of the shape (WINDOW_SIZE , 8) , where 8 was our feature set. We then created a simple Sequential Model using Keras python API. This model included LSTM layer with varying hidden units followed by two Dense Layers with 100 and 4 hidden units respectively. We then split our dataset into Training and Test sets and performed experiments.

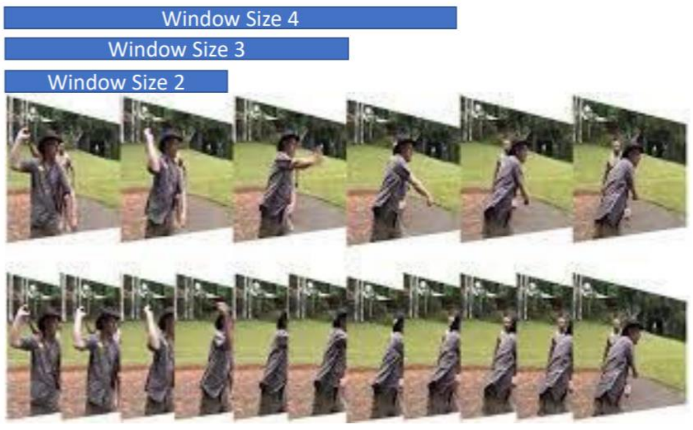
LSTM Model





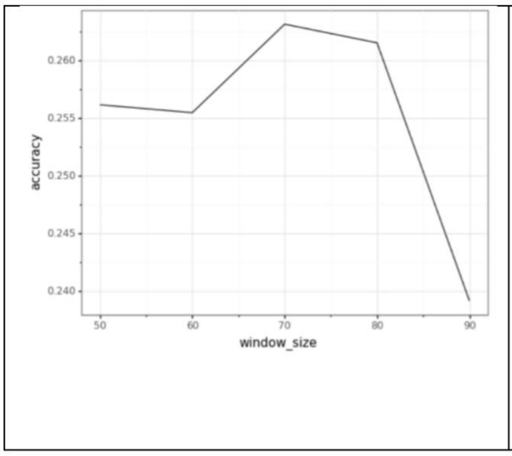
```
model = Sequential()
model.add(LSTM(hidden_units, input_shape=(n_timesteps,n_features)))
model.add(Dropout(0.5))
model.add(Dense(100, activation='relu'))
model.add(Dense(n_outputs, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

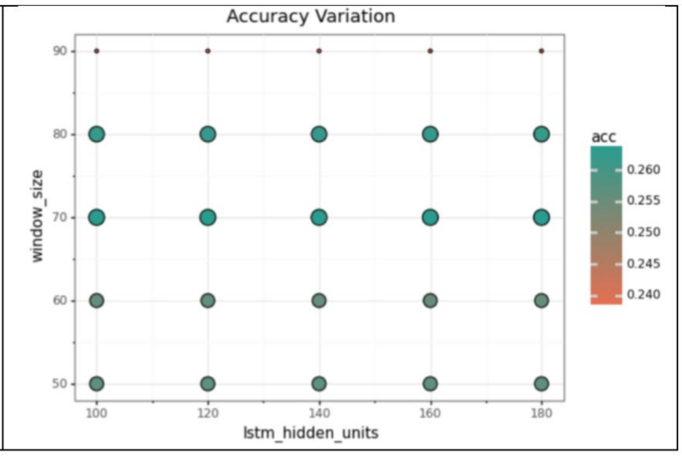
Experimentation
- Varying Window Size
- Varying Number of Hidden Units



Results

Our model performed very poorly; low performance was expected but our empirical performance were too low for any practical use. Part reason for this poor performance was the measly size of the dataset. Max performance achieved was 0.26 was only slightly better than randomly deciding 1 of 4 categories. We tried data augmentation by adding gaussian noise but even that did not change the performance level and we were forced to tweak our approach.



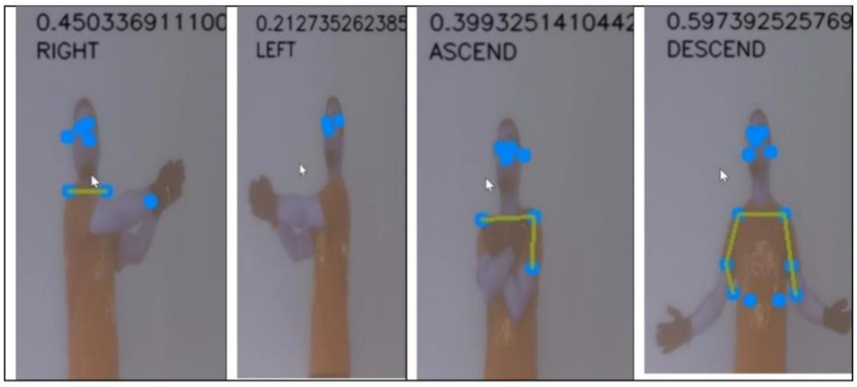


Even with larger window sizes we did not see any improvements.

Redesign

Since we had anticipated this problem of small dataset, we decided to pivot to an approach which -

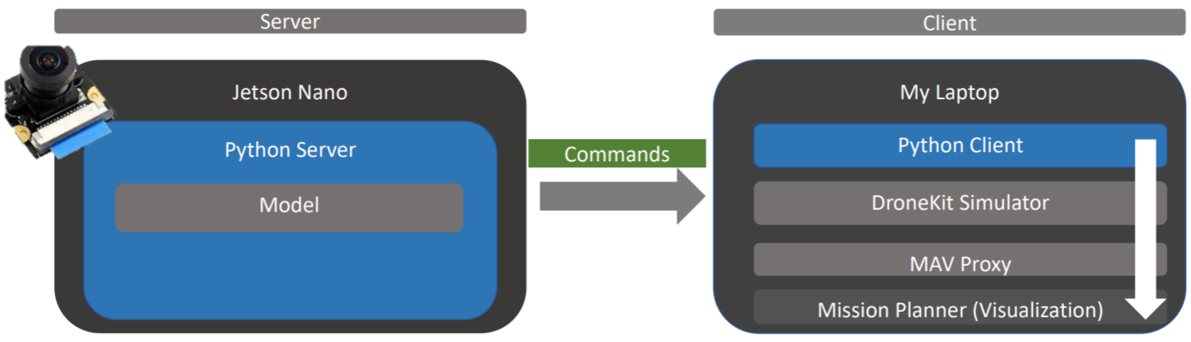
1. Used static gestured instead of dynamic hand movement based gestures.
   a. These static gestures were chosen such that even a simple model can easily distinguish between them



2. Used a Rule Based Model instead of deep learning LSTM
   a. Since we had too few observations in our sample, it made no sense to continue with deep learning based models which require a lot of data to learn even basic patterns.
   b. We decided it was pragmatic to switch to a Rule Based Model because this allowed us to circumvent need for curating a new dataset.

Drone Control

Once we had our model ready, we moved onto its integration with the Drone. For this purpose, we used two python libraries one for Drone Simulation (drone software in the loop) and another for Drone Control. Along with these, we also leveraged Mission Planner to visualize drone movements.
We went for a client-server architecture to interface between our model and the drone (simulated)



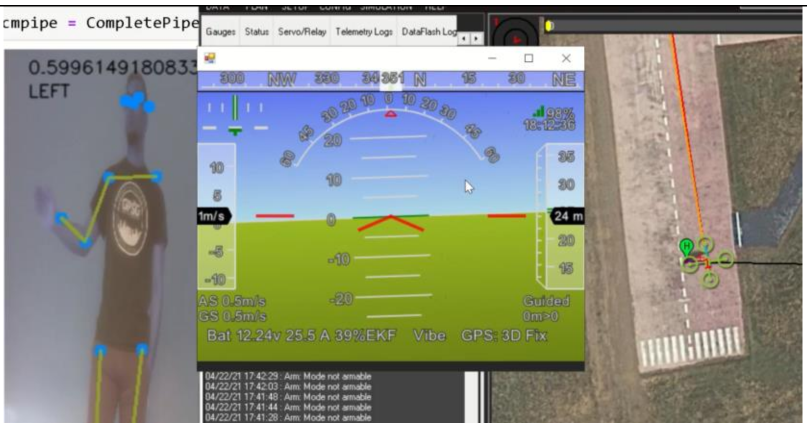
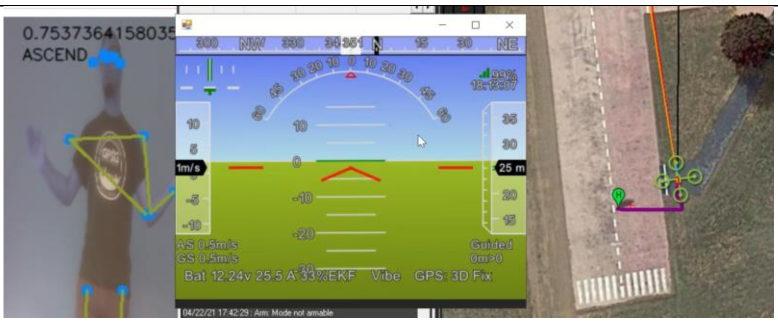
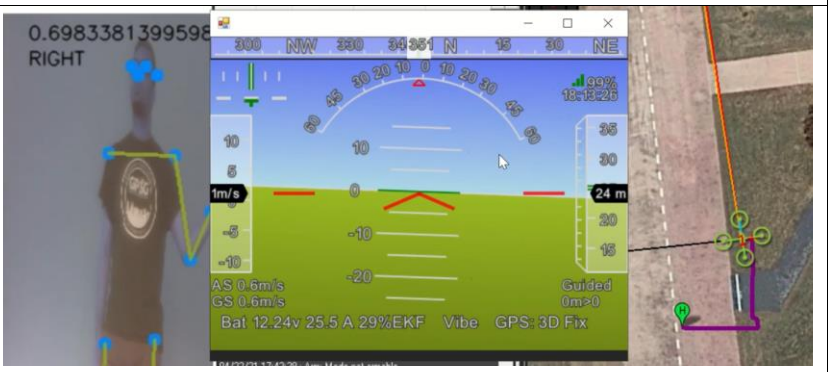
Server:

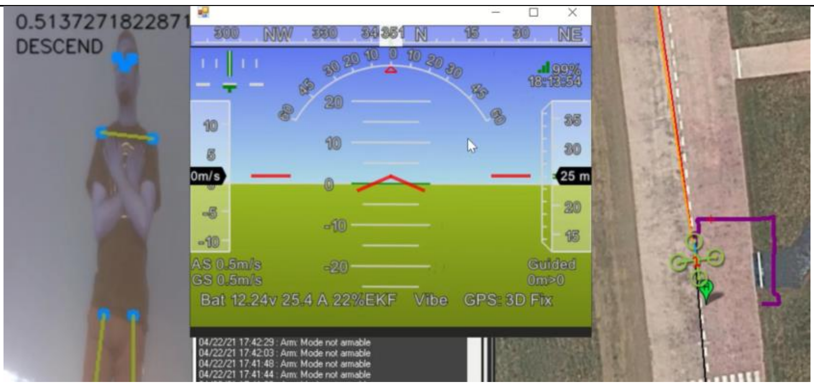
Jetson Nano with the model running on top of it acted as a server and sends the identified gestures to the client.

Client:

A Laptop acted as a client with drone controller script and Mission Planner running on it. Drone controller script accepted the stream of identified gestures and translated them into control commands for the drone.

Below is a series of screen grabs from drone being controlled using gestures. On the right side of the screen grab you can see the position of the drone with the black line indicating the direction it's heading in.

Left Gesture	 cmpipe = CompletePipe 0.5996149180833 LEFT Bat 12.24v 25.5 A 39%EKF Vibe GPS: 3D Fix 04/22/21 17:42:29 : Arm: Mode not armable 04/22/21 17:42:03 : Arm: Mode not armable 04/22/21 17:41:48 : Arm: Mode not armable 04/22/21 17:41:44 : Arm: Mode not armable 04/22/21 17:41:28 : Arm: Mode not armable
Ascend Gesture	 0.7537364158035 ASCEND Bat 12.24v 25.5 A 33%EKF Vibe GPS: 3D Fix 04/22/21 17:42:29 : Arm: Mode not armable
Right Gesture	 0.6983381399598 RIGHT Bat 12.24v 25.5 A 29%EKF Vibe GPS: 3D Fix

Descend Gesture	

0.5137271822871
DESCEND
300 NW 330 34 351 N 15 30 NE
99%
18:13:54
0m/s
25 m
AS 0.5m/s
GS 0.5m/s
Guided
0m>0
Bat 12.24v 25.4 A 22% EKF Vibe GPS: 3D Fix

```
04/22/21 17:42:29 : Arm: Mode not armable
04/22/21 17:42:03 : Arm: Mode not armable
04/22/21 17:41:48 : Arm: Mode not armable
04/22/21 17:41:44 : Arm: Mode not armable
```

We had met all the technical expectations:

i. **An acceptable classification accuracy**

Once we moved to the rule-based model with static gestures were able to estimate a rough performance of around 60% to 75% accuracy in our gesture recognition.

ii. **Real-time performance**

We were able to maintain 20-25 FPS performance which was acceptable for maintaining control of the drone in realtime

iii. **Memory and Compute efficient system**

We had managed to squeeze in the entire pipeline onto a Jetson Nano enabling deployment on an embedded system.

Role of DSBA Coursework

In Spring 2020, I had taken Computer Vision course under Stephen Welch and that coursework was extensively relied on during the entirety of the project.

From Image pre-processing to Deep learning and Rule Based models, all topics were covered in Professor Welch's class and proved to be of immense help in this project. We also had guest lectures in that course from Industry experts who facilitated knowledge transfer on operating and maintaining machine learning models in a production environment. This was especially relevant considering the deployment of the model onto Jetson Nano in our case.

Other DSBA courses that were relevant were:

1. Applied Machine Learning, which laid the groundwork for scientific approach to model development. From requirement elicitation to data gathering and cleaning and then to model training and evaluation.
2. Knowledge Based Systems- this course helped understand the importance of robust end-to-end pipelines. Building data transformation pipeline and integrating it with other pieces while keeping the performance metrics above suitable thresholds was leveraged in this project.
3. Visual Analytics – While performing feature engineering and exploration, principles taught in this course proved to not just relevant but also crucial for efficient exploration and presentation.

Learning Experience

Personally, this internship has been a great learning experience and has clearly augmented my learnings from the DSBA program. Few of the key takeaways for me have been:

1. **Operationalizing Models**
   This is one thing I feel is not tackled in the program, we are taught how to build and interpret the models but not how to deploy and maintain those models.
   This internship has been an uphill battle in this regard because not only are we deploying the model, but we are doing so in a resource constrained environment which has its own short-comings and challenges.
2. **Model Integration and Pipelines**
   In DSBA program we are usually taught model building in isolation, in the sense that once the model is done it’s an independent piece on its own. But in the internship the model formed part of a bigger system and hence had to interact with other sub-systems of the overall product. This provided insight into how each piece has to be built efficiently so that optimal performance can be extracted.

Discussion and Conclusions

Impact

This project was one in a series of projects which have been going on since 2019 Fall under Rick. So far, teams had solved part of the problem and documented their approaches. The dataset we inherited was also an artifact of a previous team. But the problem was yet to be solved completely. Previous teams stopped short of either deploying the model on Jetson Nano or integrating the model with the drone for a successful test flight. This time around we got an end-to-end working system that meets our standards and requirements and completes one long outstanding goal.

Future Work

We are going to package and document our system so that the next team can build on top of it and move on to more challenging goals of drone control and autonomous navigation.

New avenues of inquiry

Working on this project introduced me to the world of edge inference and embedded AI. This is one part of the IoT ecosystem where machine learning algorithms are running on locally owned computers or embedded systems as opposed to on remote servers. While the algorithms might not be as powerful, we can potentially curate the data before sending it off to a remote location for further analysis. One prime example of Edge AI is your common smart speaker. Combined with model compression and lightweight machine learning platforms like Tensorflow Lite, we can create powerful machine learning applications on the edge without needing expensive cloud services. For me this has opened possibilities of home automation projects that can serve me personally but can also be scaled as part of an enterprise.

Appendix

Socialization experiences

Even though the internship was remote but that in itself was a learning experience because it was easier to setup time with external teams/people especially with regards to Drone Controller development. We could get advice and recommendations on our approach much more freely and quickly as people could quickly hop on to a web call which allowed us to communicate our progress and get feedback.

Art of mentorship

Working with Rick was a pleasure, I had previously worked with him on another project where we were tracking COVID cases using WiFi data. As always Rick was always asking if there were any roadblocks or if there were any help that I needed with the project. He would arrange for calls with other groups and also a regular cadence for us. His understanding of Drones and Jetson Nano was also of great help in bootstrapping the project.