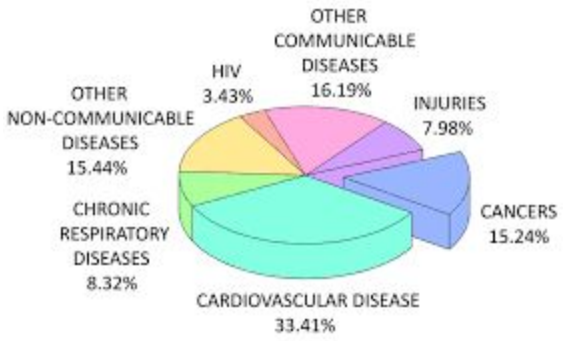
A Novel Intelligent Drone Navigation System for Safe Autonomous Flight Using a Convolutional LSTM Neural Network for Emergency Medical Assistance

By: Anant Bhatia and Allen Ye

Lynbrook High School

Introduction and Problem:

Heart disease is the leading cause of death for people in the United States, and around 320,000 out-of-hospital cardiac arrests occur annually. 90% of these victims die, and these deaths could have been avoided if victims were able to receive immediate medical treatment and medicine, such as defibrillators or epinephrine. This project implements a novel application of using unmanned aerial vehicles for emergency medical assistance by having them carry lifesaving medical devices and medicine to people in need. Currently, traveling through crowded cities and streets pose a challenge to current drone navigation systems that rely on GPS for pathfinding and external sensors, such as infrared, for obstacle avoidance. These hard-coded methods are not robust or adaptable, which makes them unreliable to use. We designed an autonomous drone navigation system that is both robust and adaptable as well as an user interface in the form of a mobile app to be implemented towards emergency medical assistance.



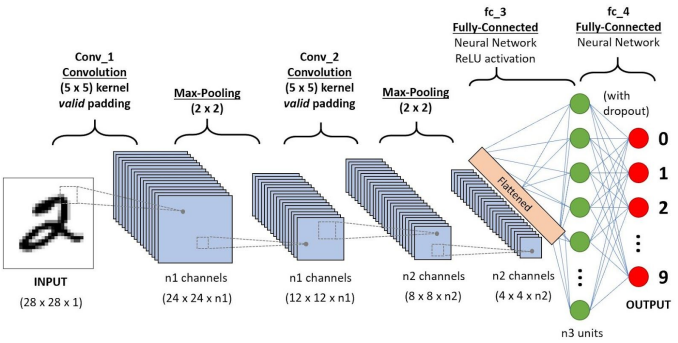
Statement of Purpose:

Our goal for this project is to design an autonomous drone navigation system to deliver emergency medical assistance to those in need. We will use deep learning to train drones and allow them to have the capability to safely navigate themselves without the need for human supervision. These autonomous drones will then be able to quickly travel through crowded cities and areas since they are not limited to the ground or roads, and deliver swift emergency medical assistance. We will also create a mobile app that allows users to request immediate help by sending their location and their needs to a real time database.

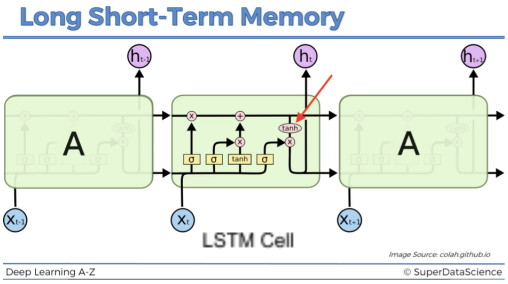
Background Information:

Kernel: A kernel is any matrix A, that, when multiplied by another matrix B, transforms B in a way that it highlights a certain feature. For example, some kernels can find features in images that include vertical lines, 45-degree lines, or circles. Kernels are called filters when they are convolved across an image.

Convolutional Neural Network: A convolutional neural network, or CNN, is a type of machine learning algorithm that takes an image and outputs a condensed version of that image (usually a vector or matrix). This is done by multiplying different kernels across the image and constantly improving these kernels through a process called gradient descent.



Long Short Term Memory Neural Network: A long short-term memory neural network, or LSTM, is a type of machine learning architecture that uses previous predictions and occurrences as a basis for predicting current input. This is useful because it learns temporal information, such as movement. For example, an LSTM can predict current frames in a video using previous frames (something that humans do as well).



Cost function: All machine learning algorithms learn by minimizing a cost function. This is equivalent to learning based on taking actions that give you the "best score". The cost function we used in this project was Mean Squared Error. Consider y to be the correct label and the predicted label to be $\hat{y}$, then $MSE(\hat{y}) = (y - \hat{y})^2$.

Images as functions: One important precursor to note is a different view of an image. An image can be considered as a 3-dimensional function, with the image as the X and Y plane, and the brightness as the z-value or third dimension. Thus, we can use calculus for image processing.

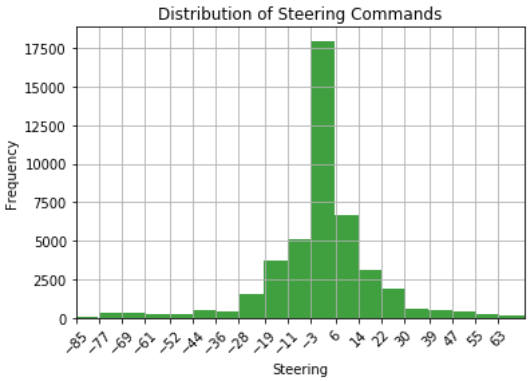
Materials:

A Parrot Bebop 2 Drone

For this project, we decided to purchase a 200 dollar Parrot Bebop 2 drone. The reason why we chose this specific model is because of its compatibility with Python. Python is essential for our project because of its numerous machine and deep learning libraries. The libraries we used include TensorFlow, Keras, Numpy, and Pandas for our machine learning models, OpenCV and PIL for image and data preprocessing, and PyParrot for interfacing between our 7n computer and the drone. Because we used supervised learning, we had to find a dataset to train our drones on. We decided on a self-driving car dataset consisting of 45,567 images, each labeled with the steering angle that a car takes. This dataset is perfect for our project because the environment its images were taken from is going to be extremely similar to environments our drone will be flying in. One important note is that the best in our dataset was

Example of an image from the dataset

collected from flex sensors on the steering wheel, which will be important later.



Range of steering angles provided by the dataset

Methods:

Data Preprocessing:

Data Preprocessing is an important step in the machine learning process because it transforms data that is useful for humans into data that is useful for a computer. Our data preprocessing methods include augmentation, blurring, and edge detections.

Augmentation: Data augmentation adds certain shifts, jubilations, and rotations to images before seeing it into a CNN. This makes the model more robust by allowing the model not to be too dependent on certain features, such as the angle at which the photo was taken, or the brightness in the environment

Blurring: Consider the functional representation of an image, as previously introduced. Since an image is discrete (made of discrete pixels), the function representing the image is often very jagged. This makes it hard for the computer to do certain computations that require calculus, which will be important in the next preprocessing step. To smooth out the function, we use Gaussian Blurring. Gaussian Blurring is a method that involves a kernel across an image, similar to a convolution neural network, such that each pixel is a weighted average of its neighboring pixels. We used a blurring kernel of size (5,5), with a standard deviation of 2, thus giving it the equation: $\frac{1}{4\pi}e^{\frac{-(x^2+y^2)}{2}}$.

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

This is the kernel when multiplied with an 2D image array, blurs the image

Edge Detection: To help our model determine important information, such as the curvature of roads and positions of cars, we overlaid the original image with another image containing highlighted edges. This allows the model to find edges and make decisions based on them in an easier way. We use Canny, an edge detection algorithm that calculates the gradient of the functional representation of the image to decide where edges are in the image.

$$\nabla f = G\left[f(x,y)\right] = \begin{bmatrix} f_x \\ f_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f(x,y)}{\partial x} \\ \frac{\partial f(x,y)}{\partial y} \end{bmatrix}$$

$$Edge\_Gradient\,(G) = \sqrt{G_x^2 + G_y^2}$$

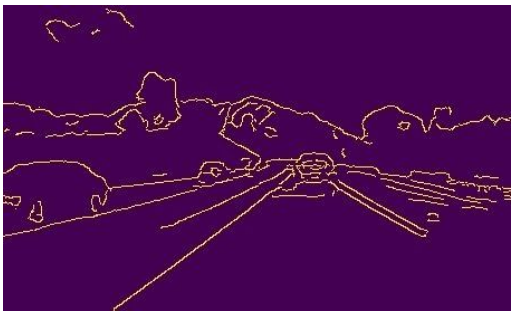
$$Angle(\theta) = \tan^{-1}\left(\frac{G_x}{G_y}\right)$$

Formulas for our Canny Edge Detector

Once the preprocessing was done, we ended up with a 128*128 pixel blurred image, with important edges highlighted in red. Shown here below is how our program processes every image it receives from the drone.

Original

Blurred Image

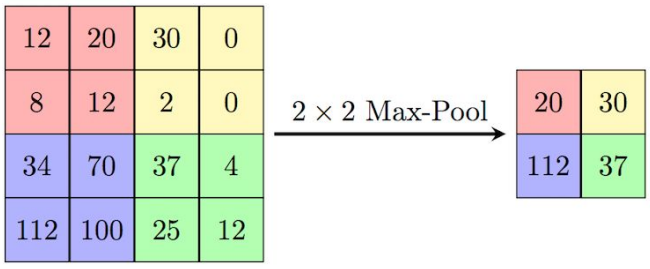


Edge Detection Image

Steering Model: We use a novel CNN-LSTM architecture to predict the steering angle of our drone. Since the drone's predictions are based on temporal information, such as movement, we must use an LSTM. LSTMs often don't work well with images though. Thus, we use a CNN to reduce the dimensionality of the image (128 by 128) into a vector representation that an LSTM can understand. The architecture looks like this:

Convolutional layer: Here we convolve certain kernels across the image to condense the spatial information in the image. This is the defining feature of a CNN

Max Pooling: This is a standardization algorithm meant to decrease the variability in an image so that it is easier to compute optimized kernels for the computer.



Example of Max Pooling

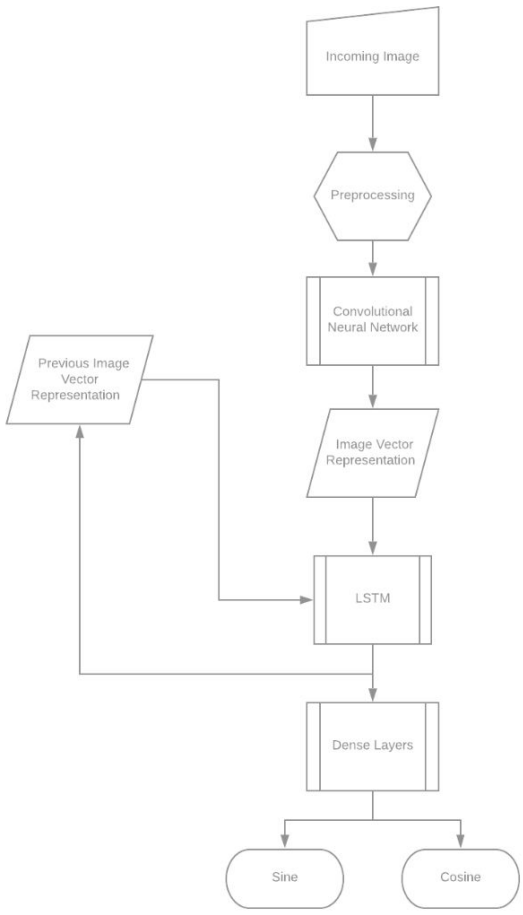
Batch Normalization: This standardized variability in batches of images so that different weights in the neural network don't shrink or blow up.

Dropout: Dropout blocks different nodes in the neural network. Oftentimes, a neural network can become too dependent on a specific edge in a graph, therefore dropout allows the weights in any given layer of the network to be relatively distributed. Thus, this will make the model more robust

LSTM: This is the layer that learns temporal information based on previous frames in the video. Previous outputs and inputs are taken into account and then used to predict future outputs.

Consider our cost function, Mean Squared Error. Due to the nature of angle, 360 deg and 0 deg are the same. Furthermore, 320 degrees and 1 deg are off by only 40 deg. We want to give a low mean squared error for these instances. Our model, though, with the first example, would calculate a mean squared error of $(360 - 0)^2$, which is really high. Thus, instead of predicting the angle itself, we predict the sine and cosine of the angle and use the inverse tangent function to predict the angle. Just to be clear, we created two different models for sine and cosine, although they could have been done in 1.

A problem that arises from this situation is that that inverse tangent has its maximum derivative, and thus its maximum variability, at x = 0, which is approximately the steering angle most used during driving (driving straight). This high variability was countered by the fact that our data was 0 centered, which meant it learned how to predict things near 0 better. Thus we ended up with a highly robust CNN-LSTM model.



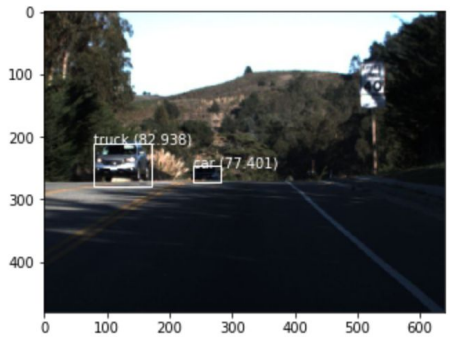
Our Steering Model Architecture

Speed Model: We evaluate what speed the drone should travel at depending on its probability of colliding with an obstacle in its surroundings. This being said, since we didn't have a labeled dataset to do this, and since finding the probability of colliding into something isn't precisely predictable, we used unsupervised learning methods. Specifically, we drew bounding boxes around important objects and then mapped these boxes to a probability of collision.

MobileNet: To predict the probability of colliding into something, it is often useful to know how far away it is. Unfortunately, because we only have one camera, we cannot perform depth perception (humans have 2 eyes for that exact reason). Instead, we decided to draw bounding boxes around all important objects - cars, trees, trucks, and people - in an image, and map these boxes to a probability. To draw said boxes, we used a pre-trained CNN called MobileNet. Training our own bounding box CNN was infeasible and unnecessary because MobileNet took months to develop by the University of Washington.

Instead, we used a technique called transfer learning, to make their architecture applicable to our problem.

```
truck 82.9379677772522
car 77.40051746368408
X is  0.1230754187696977  and y is  0.16071192493346592
X is  0.10247133310990375  and y is  0.06031537036457072
Probability of collision:  0.15010502712128193
```



Probability functions: Since predicting the probability of collision at any given state is relatively subjective (if I asked 2 people on the street what their probability of colliding into an object was, one might say 60% and one might say 70%), there isn't any ground truth we could use to train a model. Thus, we created our own mathematical model based on certain assumptions: objects in the lower half of the picture are closer to the drone, and objects near the edges of the image are not as likely to hit the drone. We also take into account the steering angle of the drone, since if the drone is turning to the left, it is more likely to hit an object in the left half of the image. Thus, our mathematical models look like the equations to the right. A visualization of these models is displayed on the left, where white signifies a higher weighted area black signifies a lower weighted area. Consider a box b. Then, the probability of colliding into the object in b is $p(b) = \iint_b W(x) * W(y)\, dx\, dy = \int_b W(x)\, dx \cdot \int_b W(y)\, dy$. This double integration (an application of Fubini's theorem), though, can make the probabilities too small, thus, we redefine the probability to be $p(b) = max(\int_b W(x)\, dx, \int_b W(y)\, dy)$. We then take a weighted average to obtain the formula to the right.

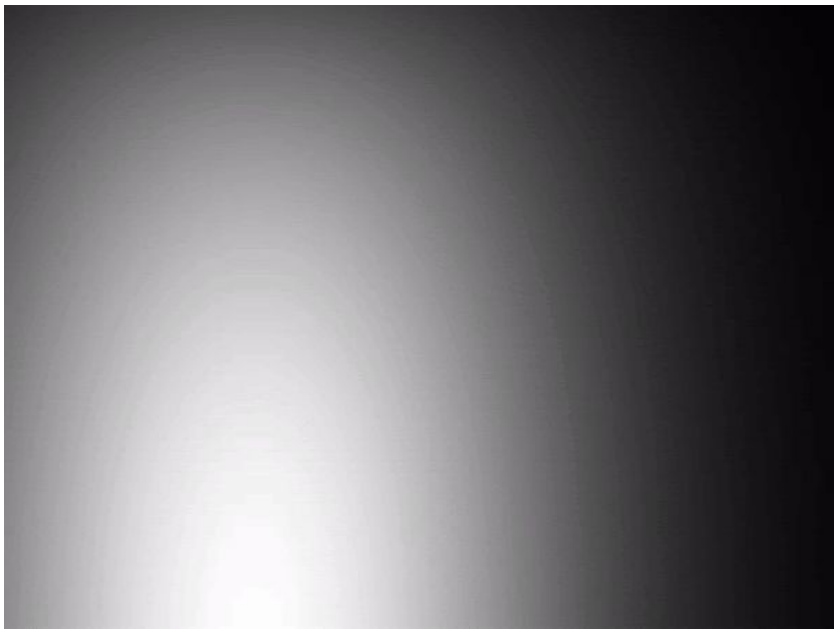
In this manner, we can robustly and quickly efficiently predict the probability of the drone colliding into any obstacles at any given time. When running the program, the user can set a probability threshold α, such that is the probability of collision is greater than α, the drone follows a user-defined function f (in our case, f is just flying the drone upward).

$$P_{final} = \sqrt{\sum_{box}(max(\frac{\int_b W(x)dx}{\int_I W(x)dx}, \frac{\int_b W(y)dy}{\int_I W(y)dy})^2}$$

$$W(x) = A \cdot e^{\frac{-4 \cdot log(I) \cdot (x - w - \frac{w}{2.5} cos(\theta))^2}{w^2}}$$

$$W(y) = \frac{-A \cdot (BM-1) \cdot (1 + \sqrt{1 - \frac{B-1}{BM-1}})^2}{h^2 BM} y^2 + \frac{2A \cdot (BM-1) \cdot (1 + \sqrt{1 - \frac{B-1}{BM-1}})}{hBM} y + \frac{A}{BM}$$

Our Novel Math Formulas to Determine a Probability Map



A Visual Representation of our Weighted Importance Map

How to interpret this map:

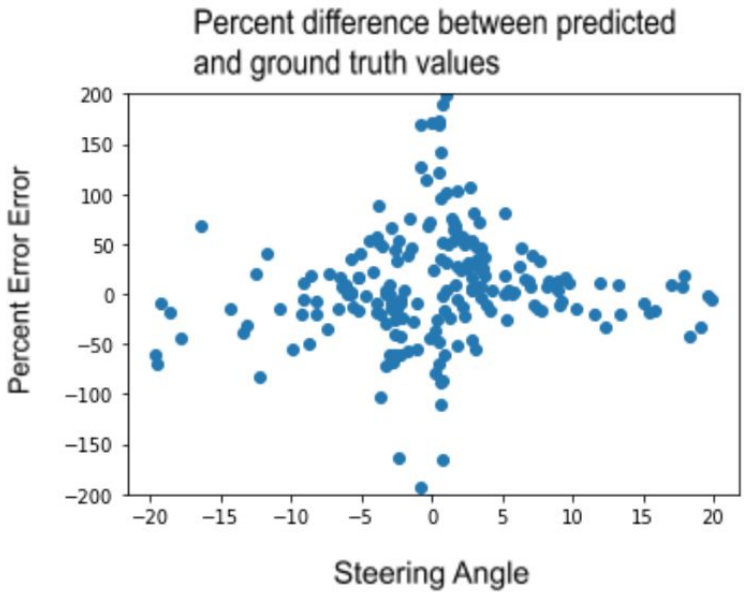
Lighter areas are areas of higher importance, so obstacles that are located in a white-gray region will be taken into account with a higher importance factor than obstacles located in darker regions. As you can see, the map is not static and is continuously shifting. This will be due to the direction our drone will be steering in.

Results and Conclusion:

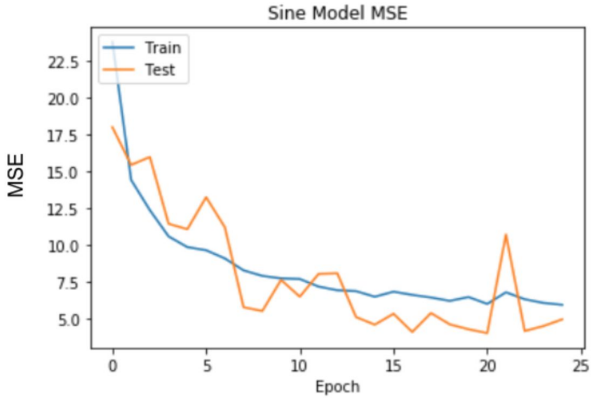
CNN-LSTM: Due to the largely unsupervised nature of our project, the extent to which we can analyze our results is limited. This being said, we can analyze the association between our predictions and the ground truth in the dataset. After running our CNN-LSTM for 25 epochs, we got the results given on the right (don't mind the epoch numbers, the API we had an error). It is important to note that all sines and cosines were multiplied by 20 to increase the gradient of the cost function and help our algorithm converge. Our Mean Squared Error after 25 epochs of training is 2.579/(20^2) ~ 0.01 on a scale from 0 to 1. Our Mean Absolute Error is 0.03. These values are extremely good in terms of the correlation between the predicted and actual values.

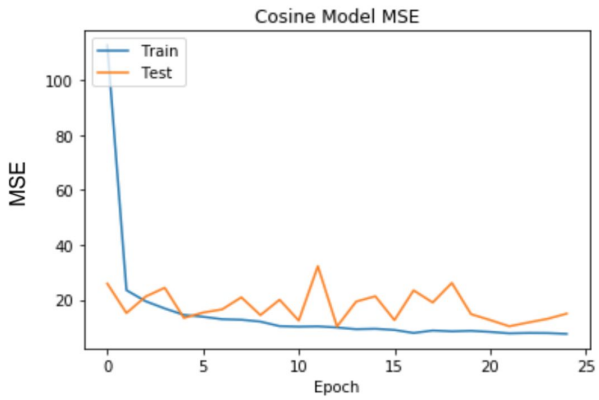
```
Epoch 3/25
909/908 [==============================] - 248s 273ms/step - loss:
18.8684 - mse: 18.8734 - mae: 1.9847 - val_loss: 5.6445 - val_mse:
25.0869 - val_mae: 2.2263
Epoch 4/25
909/908 [==============================] - 252s 278ms/step - loss:
5.4210 - mse: 5.4256 - mae: 1.0071 - val_loss: 0.6156 - val_mse: 4.
3514 - val_mae: 0.8377
Epoch 13/25
909/908 [==============================] - 248s 272ms/step - loss:
5.0564 - mse: 5.0503 - mae: 0.9637 - val_loss: 2.8872 - val_mse: 8.
9852 - val_mae: 1.3467
Epoch 14/25
909/908 [==============================] - 242s 267ms/step - loss:
3.1622 - mse: 3.1649 - mae: 0.7764 - val_loss: 0.7300 - val_mse: 2.
9505 - val_mae: 0.6702
Epoch 25/25
909/908 [==============================] - 244s 269ms/step - loss:
3.0568 - mse: 3.0596 - mae: 0.7725 - val_loss: 0.3792 - val_mse: 2.
5794 - val_mae: 0.6312
```

The picture above shows the progression of our MSE over 25 epochs of training



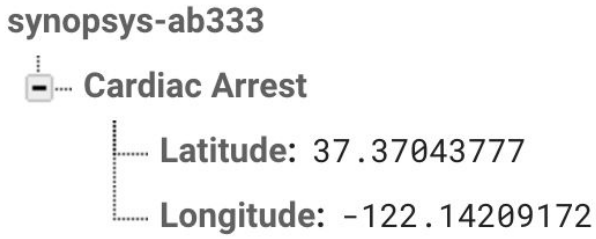
We also tested the amount of time it takes to predict an image through our machine learning pipeline, and on average it takes 0.236 seconds with a standard deviation of 0.057. This is fast enough for real-time implementation. To the left is also a graph of the percent error between the predicted angle and the ground truth. There is a spike near 0 because even small deviations in prediction near 0 lead to a high percent error (for example the percent error between 0.01 and 0.1 is 1000% even though the predictions are really close). The two graphs below show the progression of our MSE over time.





User Interface App:

In addition to creating a deep learning model for our drone to navigate itself, we also developed a mobile app to implement the real life applications from theory to reality. Using Swift, we developed an iOS app that allows users to immediately request for help with the press of a button. To do this, they first choose an option of whatever emergency is occuring to themselves or someone near them (i.e. cardiac arrest), and they upload this information along with their location in the form of longitude and latitude coordinates to Firebase, a real time database, with just a click of a button. Simplicity is key, since during real emergencies every second of time is extremely valuable. From the data uploaded to Firebase, emergency responders will be able to figure out exactly what and where emergency devices or medicine are needed to be sent.



Example of uploaded data to our real-time database. From this data, we are able to recognize that the drone must carry a defibrillator as well as Atropine to the specific given location.

Challenges:

We faced many challenges while creating our project, and two of the main problems was controlling our drone with a computer and optimizing runtime of our models.

Interfacing: The Parrot Bebop 2 drone is meant to be controlled with a mobile app that the Parrot company officially designed. However, there was no clear-cut way to control a drone with our computer, and finding a way to perform this took a lot much more effort and time than we had ever anticipated. We attempted using many libraries and APIs, the main ones include Bebop Autonomy, Olympe, and PyParrot. We ended up only being able to use PyParrot in the end.

Bebop Autonomy: We first tried using Bebop Autonomy, a robot operating system(ROS) driver specifically for Parrot drones, and found that it wasn't compatible with either of our computers. In addition, its installation guide was extremely complicated and after compiling error after error we decided to try another library.

Olympe: We discovered Olympe, which was a Python controller programming interface for Parrot drones and computers. We were able to get this working, but we found that runtime of its scripts took too long and the level of complexity of its scripts was a little too much for us to handle since neither of us was familiar with using this library.

PyParrot: Finally, we found PyParrot and compared to Olympe the scripts were much easier to write and the API was overall a lot more user-friendly with a wide range of

examples. Not only that, but it was open source meaning we could directly edit built-in functions, which was extremely convenient.

Runtime: After compiling and running all our models and scripts, we had a runtime of roughly 120 seconds. Obviously, a runtime this long would not allow our program to be applicable in real life.

YOLOv3: Before we used the MobileNet CNN in our speed model, we started off with another object detection CNN called YOLOv3. We sourced most of the runtime to YOLOv3's image labeling method, which sacrificed runtime in order to increase the accuracy of predicting and labeling exactly what an object was. However, this level of accuracy was not needed for our project, for example crashing into a tree or a car results in the same thing: failure. YOLOv3 also required a non-maximal suppression algorithm which ran in $O(n^2)$.

After Optimizations: After switching to MobileNet and performing many math optimizations in our speed and steering models, we were able to get the runtime down to 0.29 seconds as a lower bound and 1.03 as an upper bound. The average runtime was 0.66 seconds and the standard deviation was 0.18 based on 150 trials. This meant that we increased our efficiency by more than 160 times.

Future Improvements:

Despite the many challenges we faced, we were able to create a finished working product. However, we believe that there are still a few things that we can do to further improve upon.

GPU or Better Computer: To further decrease runtime, we believe using GPU acceleration or a better computer will allow the program to run even faster. This then would allow the drone to fly faster, increasing its usefulness.

More Training: Training the model on a larger and more varied dataset would improve the drone's flying and adaptability, making it applicable in more situations. With our current program, if you want the drone to work in another environment all you need to do is just find a dataset for that environment.

Processing: Currently, the drone uses its own wifi signal to communicate with the computer. This means that the drone could only fly as far as its wifi range allows it to. If we could move all the processing from our computer onto the drone however, it would give the drone an unlimited range to fly in making it even more adaptable in situations.

App: Currently, our GUI for our mobile app is pretty plain, and we hope to later implement and create a better app with a better and easier to use design.

Further Application:

As you can imagine, a self-flying drone is applicable in nearly an unlimited amount of applications. We propose to use our drones, in addition to emergency medical assistance, for conservation, data gathering, natural disaster relief, and package delivery. For conservation, our drone could be implemented to gather data on animals by tracking them in their habitat without human interference. As for natural disaster relief, drones could scout and take risks that volunteers are unable to, due to debris and unstable infrastructure. We hope that our drone navigation program will be useful for many future applications.

Selected References

Cheryl, Fryar D, et al. “Prevalence of Uncontrolled Risk Factors for Cardiovascular Disease: United States, 1999–2010.” *NCHS Data Brief*, www.cdc.gov/nchs/data/databriefs/db103.pdf.

“CPR Facts and Stats.” *Cpr.heart.org*, cpr.heart.org/en/resources/cpr-facts-and-stats.

Fazli, Pooyan. “Learning to Navigate Like Humans.” Learning to Navigate Like Humans, Arxiv.org, 2018, pooyan.perlab.info/publications/MHamandi-RSS18-LAIR.pdf.

He, Kaiming. “Deep Residual Learning for Image Recognition.” Deep Residual Learning for Image Recognition, Arxiv.org, 2015, arxiv.org/pdf/1512.03385.pdf.

Loquercio, Antonio. “DroNet: Learning to Fly by Driving.” DroNet: Learning to Fly by Driving, University of Zurich, 2018, rpg.ifi.uzh.ch/docs/RAL18_Loquercio.pdf.

Löf, S., et al. “Drone Delivery of an Automated External Defibrillator – a Mixed Method Simulation Study of Bystander Experience.” *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine*, BioMed Central, 1 Jan. 1970, sjtrem.biomedcentral.com/articles/10.1186/s13049-019-0622-6#abbreviations.

Wu, Tung-Cheng, et al. “Navigating Assistance System for Quadcopter with Deep Reinforcement Learning.” Navigating Assistance System for Quadcopter with Deep Reinforcement Learning, Arxiv.org, 2018, arxiv.org/pdf/1811.04584.pdf.

Yang, Yinchong. “Tensor-Train Recurrent Neural Networks for Video Classification.” Tensor-Train Recurrent Neural Networks for Video Classification, Arxiv.org, 2017, arxiv.org/pdf/1707.01786.pdf.