# Flyover: Aerial Mapping from Drone Footage

Alex Yue
acyue@princeton.edu

Victor Zhou
vzhou@princeton.edu

Bharath Srivatsan
bharaths@princeton.edu

## ABSTRACT

Drones have become ubiquitous in both public and private contexts. Used for everything from pleasure flights to military reconnaissance, these aerial vehicles have the potential to fundamentally transform many industries.

In this paper, we introduce Flyover, a state of the art aerial mapping technology. Flyover is able to generate detailed overhead maps from post-flight footage produced by widely available drone hardware, without the need for any GPS or EXIF flight path data. Existing commercial solutions are either unable to generate these flyover maps without expensive supplementary data or simply generate low-quality outputs. A quantitative evaluation comparing Flyover to the only commercially available, post-flight, footage-based system, MapsMadeEasy, showed that our system was widely favored among a surveyed audience of 20 classmates, scoring 3.4 on a scale of 1-4 (versus MapsMadeEasy at 1.2).

Furthermore, we show that the aerial maps generated from dynamic drone footage is of a sufficient quality to be used in many post-processing applications. We demonstrate the viability of one such application by building a deep-learning based classifier to identify cars in our output maps. It achieves precision and recall accuracy of 0.70 and 0.94, respectively.

## 1. PROJECT DESCRIPTION

The prevalence of affordable, unbox-and-fly drones has led to an explosion of drone-based video footage. In industrial and commercial settings, this has brought about a host of startups aiming to use (often proprietary) drones to create high accuracy spatial mappings and metric calculations. These services can involve extraordinarily high sticker costs. For commercial jobs like site mapping and surveying, clients often opt for high-end, purposed vehicles that can cost tens of thousands of dollars. For civil jobs like urban zoning and traffic engineering, municipalities can pay similar amounts to acquire critical data.

We set out in this final project to investigate whether the amateur videos generated by off-the-shelf drones can be used to create large scale stitched maps. To this end, we hope to use two immensely popular drones, the DJI Spark and Phantom 3,[1] to create state of the art overhead maps. Software to build high-resolution maps from cheaply available drones would be useful not just for surveying and mapping projects, but also, more crucially, in post-disaster scenarios to estimate damage and identify key points quickly.

Finally, we use the maps generated by this footage to build and test a car classifier. Such an application would be relevant for municipalities and other civic organizations looking to conduct traffic engineering or perform occupancy analyses.

## 2. PREVIOUS WORK

There exist a variety of competitors to our work on this project in both the research and commercial sectors. However, none of the other solutions to this problem produce high-quality aerial stitching outputs from only post-flight flyover footage.

Some drone mapping packages, including 3DR Site Scan and WebODM, require hands-on pre-flight involvement. They require either direct drone control via their apps, or a large amount of post-flight data to help identify movements and absolute positioning. These images, furthermore, can still be susceptible to warping and are often identifiably stitched.

The main software vendor that offers a truly competitive feature set is MapsMadeEasy. MapsMadeEasy advertises online tools that can assist users in utilizing their drones to generate aerial images. While it appears to be similar in functionality and form to Flyover,

This paper is a written report on a final project submitted for the Fall 2017 offering of COS 429 at Princeton University. It was written in accordance with University Regulations.
Signed: Alex Yue, Victor Zhou, and Bharath Srivatsan

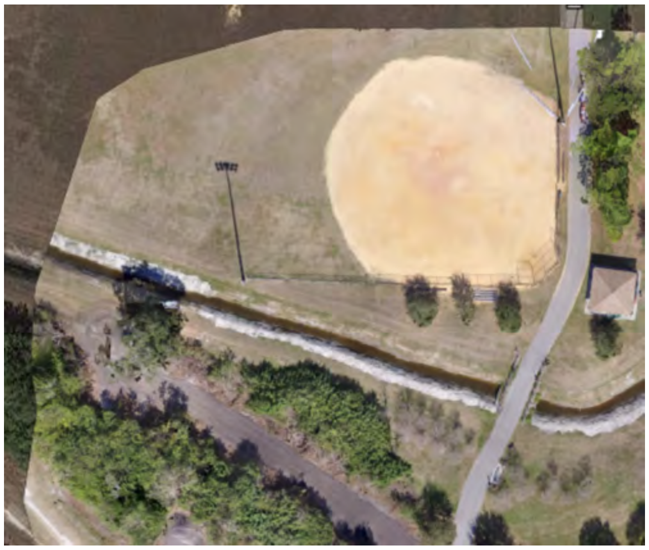


Figure 1: Sample map output from 3DR Site Scan, with evident warping and stitching present

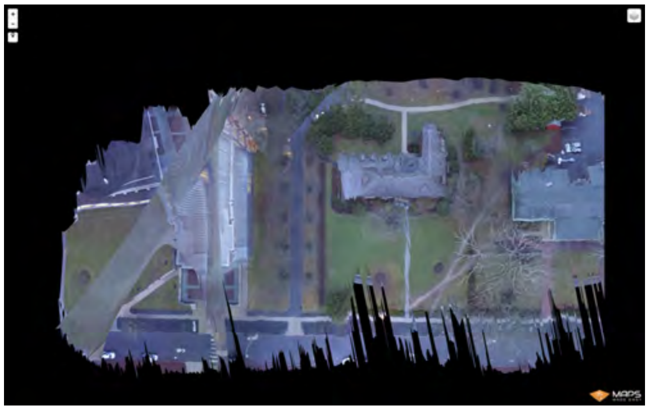


Figure 2: Sample map output from MapsMadeEasy, demonstrating terrible stitching presence, jagged edges, and warped buildings.

MapsMadeEasy requires users to upload either a flight path or to generate one using their online interface in order for their technology to work optimally. This software package does not support post-flight, dynamically shot flight footage very well without this supplementary flight data (see Figure 2 above). Other software solutions such as Pix4D offer similar offline functionality, but are incredibly expensive (with software licenses costing upwards of \$8,700[2]) and are typically designed to be used with flight path data.

We leaned on a variety of techniques learned in class for this project as well. We dynamically subsampled videos using a difference of Laplacians approach for detecting blurriness. We used the SIFT algorithm to identify points of interest and RANSAC to match them across frames. A lot of the content we learned near the beginning of the semester (about creating panorama footage, etc.) came into use for stitching. Finally, our neural network-based approach to building a car classifier relied on techniques described at the end of the course.

## 3. APPROACH & EXPERIMENTS

The Flyover stitching software is comprised of many different modules that work in conjunction to create the final aerial image. The main steps involved in generating these images are: dataset collection, frame subsampling, and image stitching. The approach to each individual part of this final project is covered in the following subsections. Finally, we describe the system design and implementation of our car identification deep-learning classifier.

### 3.1 Dataset Collection

The dataset that was utilized for image stitching consisted solely of drone footage captured by DJI drones owned by members of this team. Specifically, these were the DJI Phantom 3 and the DJI Spark, both of which retail for under \$600.[3] These drones were flown under various weather and temperature conditions in order to gather a wide sample of possible landscapes for our aerial stitching software.

Our team took several precautions in order to ensure that the captured drone footage was of sufficient quality to be processed by our other modules. Particularly, we focused on capturing purely translational footage, ie. with no rotation or changes in height during video capture. Initially, we were concerned that our drones would not be able to fly in a pattern that could guarantee vertical and rotational stability; failure in this regard would increase the complexity required of our movement model as it would grow to include an affine transform model to support rotation, translation, and scaling. However, we were able to configure settings within the DJI app itself in order to minimize vertical altitude changes and prevent rotations. This included changing the IMU (inertial measurement unit) and gimbal pitch speed, and helped us prevent significant warping issues with the capture data. This also demonstrates that beginners, with a brief tutorial on changing these settings in the app, can capture adequate footage for our software.

These two drones were flown by amateur human operators (members of the group), and captured footage with the camera field of view directly parallel with the ground, at altitudes varying from 70 feet to over 400 feet. This was done to ensure that we had plenty of data to work with. We also captured footage with the camera field of view perpendicular to the ground in order to generate aerial imagery of the sides of buildings.

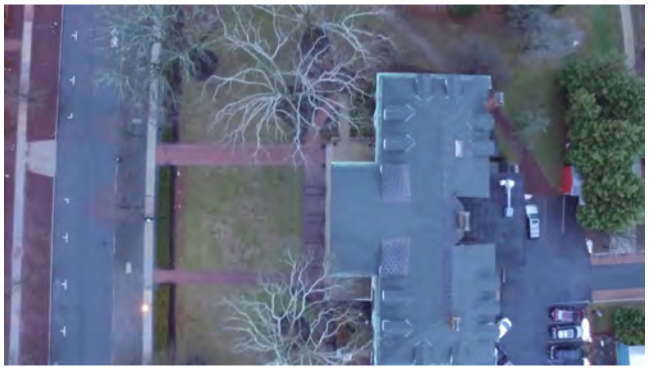


Figure 3: Sample Aerial Frame

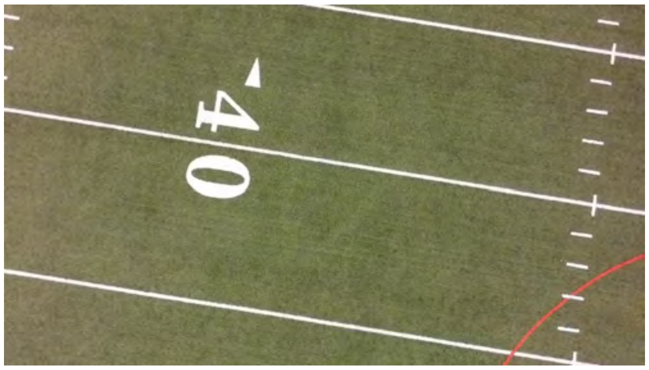
We flew our drone in various locations around the municipality of Princeton to capture a complete set of testing data to use with our system. Our dataset consisted of numerous flight patterns like Hilbert curves, linear paths, and overlapping paths. The final footage was stored in either .MP4 or .MOV format.

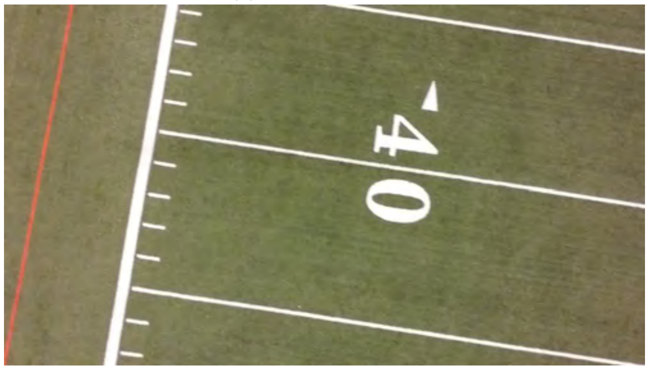
## 3.2 Frame Subsampling

Before we stitched our drone footage, we used a number of methods to subsample the video's frames intelligently. We aimed to reduce redundant calculations and speed up the overall stitching process; typically, each input video consisted of thousands of high definition frames (1980x1200 resolution), which posed a significant processing challenge for conventional computers.

Drone footage sometimes contains imperfections - for example, the camera occasionally attempted to refocus on a new target or adjusted the ISO sensitivity automatically in changing light conditions. This resulted in moments where the footage is either blurry because the ground is not in focus or where there are changes in the color temperature of the same surface over time. One filter that we applied on all frames was a blur detection filter used to remove frames that weren't sharp enough and that would degrade our final output.

The blur detection filter is based on the variance of Laplacians convolved with each individual frame. We chose this algorithm based on the research presented by J. Pech-Pacheco and G. Cristobal in "Diatom Autofocusing in Brightfield Microscopy: a Comparative Study".[4] As we learned in class, the Laplacian can be used for both edge and blob detection.[5] Essentially, the Laplacian is able to measure the change in intensity of each edge - blurry photos are less likely to result in large changes in intensity because they contain less distinct edges. Thus, the variance of the Laplacian for a blurry photo would also be smaller than that of a crisp photo allowing us to easily filter out frames by setting a minimum threshold variance. Taking the variance of the Laplacian, then, enabled us to control our tolerance for blurry photos with a single threshold and was fast to compute for each given frame.


(a) In Focus Frame


(b) Blurry Frame

We also filtered out adjacent frames that resulted in minimal displacement from one another by calculating the L2-norm offset distance between the two. This was done through a combination of the SIFT algorithm (used to identify keypoints in each frame) and a brute force matcher based on the L2-norm provided by OpenCV (to match keypoints across frames). This is somewhat computationally expensive; however, with a limited number of keypoints and prior filtering, we were able to restrict runtime to a reasonable length. If the L2-norm offset between two adjacent images fell below a given threshold, the latter image was discarded from the frame subsample. This, in turn, saved a lot of unnecessary computation later on, as a lot of frames involved small movements (less than 2 pixels in Euclidean distance) due to the high frames per second (FPS) capture rate.

Other filters that were implemented but removed from our final code included a similarity metric that compared the mean structural similarity index between two frames. This essentially measured the difference in image quality between two similar images. However, we chose not to use this measure as it was difficult to have a baseline to compare each frame to.

## 3.3 Image Stitching

Rendering a complete aerial image from a flyover requires stitching together hundreds of individual frames from a trimmed set of images. From the subset of images that remain, we ran the SIFT (Scale Invariant Feature Transform) algorithm on each frame to obtain a set of keypoints. Then, we tried to use numerous models including linear models in order to compute the pixel offset between two frames. We discovered that the best approach was to use brute force matching in order to determine invariant keypoints across frames. Finally, we used RANSAC to obtain the estimated relative offset between two adjacent frames.

Once the relative offsets between two adjacent frames was computed, we needed to join these frames together. To do this, we tried a variety of techniques - our initial approach involved joining adjacent images iteratively, immediately after relative offsets were calculated. This, though, caused issues in stitching footage from more complex, partially overlapping flight paths like U and P shapes. When partial overlapping occurred in non-adjacent frames, marginal rounding errors in relative offsets sometimes added up to ensure that these later frames would be skewed or displaced, causing jarring lines and repeated features.

We ultimately settled on a technique called composite image generation to preserve perspective through the entire aerial view. We implemented this by computing the relative offset of each new frame and only updating the edges of the proposed composite image iteratively. Only at the conclusion of the offset calculations do we convert relative positions to absolute positions for stitching. This ensured that only the most recent frame data for each pixel is shown in the final composite image, thereby preserving perspective without causing distortion or warping. In Figure 12, this technique is demonstrated: newer frames are rendered on top of older frames, ensuring that the image perspective of object flyovers remains constant (especially for tall objects like trees and buildings).

This technique was not the only one we tried; however, it outperformed techniques like image blending and min-cut stitching by far. This was because composite frame rendering handled changes in perspective with ease. Other techniques for image stitching failed to account for the relatively low altitude that drones fly at versus satellite imagery.

## 3.4 Extension: Car Detection

To perform car detection on our stitched composite images, we used the Cars Overhead With Context[6] (COWC) dataset made publicly available by members of the Computer Vision group at Lawrence Livermore National Laboratory. The dataset consists of 256 x 256

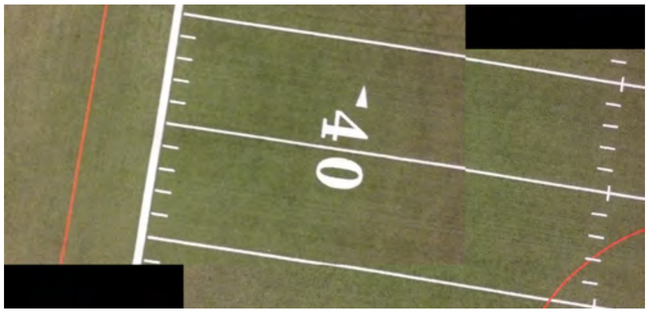


Figure 5: Image Stitching of Two Frames

patches of aerial (usually satellite) images taken in various cities around the world resized to 15 cm per pixel so that a standard car would fit within a 48 x 48 pixel bounding box. Each patch was tagged as either *car* or *neg* based on whether or not the central 48 x 48 patch of pixels contained a car - the remaining image data surrounding the central patch was meant to be used as context to improve car classification. The cars featured in these central 48 x 48 patches were of many orientations, colors, and models.

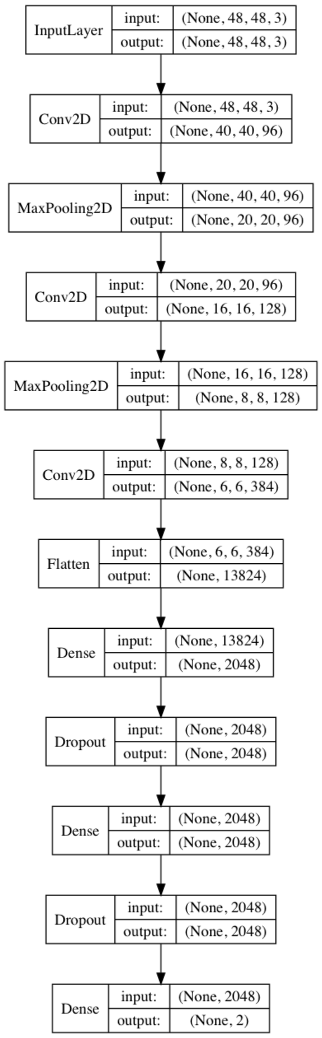
Using Keras with a Tensorflow backend, we built and trained several deep convolutional neural networks to perform binary classification (*car* versus *neg*) on an aerial image patch. The network architectures we experimented with were based on our past experience but also drew inspiration from the well-known AlexNet and VGG-16 networks. Our networks used convolutional layers, max-pooling layers, fully-connected layers, and dropout layers. We chose to use dropout layers to prevent overfitting to the training dataset, which was especially important to avoid given how characteristically similar (lighting, altitude, perspective, contrast, etc) the training patches were to each other. We trained our networks using various subsets of the COWC data, including:

- the Columbus Surrogate Unmanned Aerial Vehicle data from the United States Air Force Research Lab, which contained black-and-white samples only
- the International Society for Photogrammetry and Remote Sensing data from Potsdam, Germany and Toronto, Canada
- the Land Information New Zealand data taken in the city of Selwyn

After training a car classifier, we used a fixed size sliding window mechanism with non-maximum suppression to apply it to a test stitched composite image to detect cars.

### 3.4.1 Version 1

The first iteration (Version 1) of our neural network architecture had 3 convolutional layers mixed with 2 max-pooling layers, 3 fully-connected layers, and 2 dropout layers. Each successive convolutional layer used increasingly higher numbers of filters but increasingly smaller kernel sizes. The final layer of the network had 2 neurons with softmax activation. The network took in an input of size 48 x 48 pixels with 3 color channels and output a vector of length 2 representing the probabilities of the input being a car and not being a car, respectively.

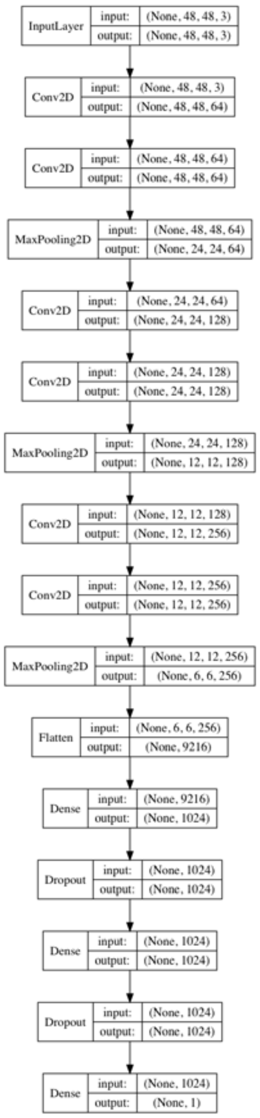


InputLayer input: (None, 48, 48, 3) output: (None, 48, 48, 3)
Conv2D input: (None, 48, 48, 3) output: (None, 40, 40, 96)
MaxPooling2D input: (None, 40, 40, 96) output: (None, 20, 20, 96)
Conv2D input: (None, 20, 20, 96) output: (None, 16, 16, 128)
MaxPooling2D input: (None, 16, 16, 128) output: (None, 8, 8, 128)
Conv2D input: (None, 8, 8, 128) output: (None, 6, 6, 384)
Flatten input: (None, 6, 6, 384) output: (None, 13824)
Dense input: (None, 13824) output: (None, 2048)
Dropout input: (None, 2048) output: (None, 2048)
Dense input: (None, 2048) output: (None, 2048)
Dropout input: (None, 2048) output: (None, 2048)
Dense input: (None, 2048) output: (None, 2)

Version 1 of our car classifier

Unsurprisingly, Version 1 trained on the black-and-white dataset performed relatively poorly when tested on grayscale versions of our stitched composite images, so we quickly discarded that dataset and focused only on datasets with color images. After quickly (< 2 hours per training) experimenting with a few different color data subsets, we quickly found that Version 1 seemed to be especially prone to false negatives. This was most likely due to a major discrepancy between our test images and the training images we used: whereas the COWC image patches were all taken from satellites or UAVs from extremely high altitudes and cropped to low resolutions, our test images were taken from off-the-shelf drones from heights of no more than a few hundred feet. The difference in altitude especially contributed to a noticeable perspective difference of the cars in our stitched images versus the cars in the training images.

### 3.4.2 *Version 2*



InputLayer input: (None, 48, 48, 3) output: (None, 48, 48, 3)
Conv2D input: (None, 48, 48, 3) output: (None, 48, 48, 64)
Conv2D input: (None, 48, 48, 64) output: (None, 48, 48, 64)
MaxPooling2D input: (None, 48, 48, 64) output: (None, 24, 24, 64)
Conv2D input: (None, 24, 24, 64) output: (None, 24, 24, 128)
Conv2D input: (None, 24, 24, 128) output: (None, 24, 24, 128)
MaxPooling2D input: (None, 24, 24, 128) output: (None, 12, 12, 128)
Conv2D input: (None, 12, 12, 128) output: (None, 12, 12, 256)
Conv2D input: (None, 12, 12, 256) output: (None, 12, 12, 256)
MaxPooling2D input: (None, 12, 12, 256) output: (None, 6, 6, 256)
Flatten input: (None, 6, 6, 256) output: (None, 9216)
Dense input: (None, 9216) output: (None, 1024)
Dropout input: (None, 1024) output: (None, 1024)
Dense input: (None, 1024) output: (None, 1024)
Dropout input: (None, 1024) output: (None, 1024)
Dense input: (None, 1024) output: (None, 1)

Version 2 of our car classifier

To try to reduce false negatives, we set aside Version 1 and tried a different approach (see Figure 18). Version 2 exclusively used 3 x 3 filters in convolutional layers but stacked 2 convolutional layers before each max-pooling layer to increase the true reach of each input pixel - two successive 3 x 3 filters has the effective range of a 5 x 5 filter. In addition, we changed the output layer from 2 neurons with softmax activation to be a single neuron with a sigmoid activation.

When trained on the same color datasets we used in Version 1, Version 2 generally had a lower false negative rate on our test images. Although the false positive rate was still significant, Version 2 seemed to have better performance across the board when compared to Version 1.

## 4. DOCUMENTATION

### 4.1 Code Modules

This project is broken up into two main sections: the stitcher and the classifier. For each, we provide a brief explanation of the individual files that comprise the codebase and their functionality.

#### *4.1.1 Stitcher Module*

The stitcher module is comprised of these main files:

- `stitcher.py` - The core logic used to generate keypoints through SIFT, generate matches between keypoints in two adjacent frames, and compute the frame offset using RANSAC.
- `frame_manager.py` - Business logic for parsing and converting the input drone footage into a format used by this module.
- `image_model.py` - Responsible for storing the final stitched image and for maintaining the relative offsets of each frame from the initial one.
- `main.py` - Launcher script to run the stitcher.
- `point_manager.py` - Keeps track of all keypoints on a global level across frames to ensure that small errors in the offset calculations don't result in alignment issues for overlapping sections in the final stitched image.

#### *4.1.2 Classifier Module*

The classifier module is comprised of these main files:

- `train.py` - Launcher script to run the TensorFlow training session to generate the H5 model file.
- `model.py` - Makeup composition of the Convolutional Neural Network used to build the model.
- `detect_cars.py` - Detector that applies classifier using a sliding window and non-maximum suppression to render the final image.

### 4.2 Dependencies

This project could not have been completed without the work of numerous contributors who have provided a vast amount of modules available through the `pip` (Pip Installs Packages) package management system. Packages that were used in this final project include:

- OpenCV
- Numpy
- scikit-image
- TensorFlow
- Keras

## 5. RESULTS

Below, we've included a variety of stitched maps generated by Flyover.

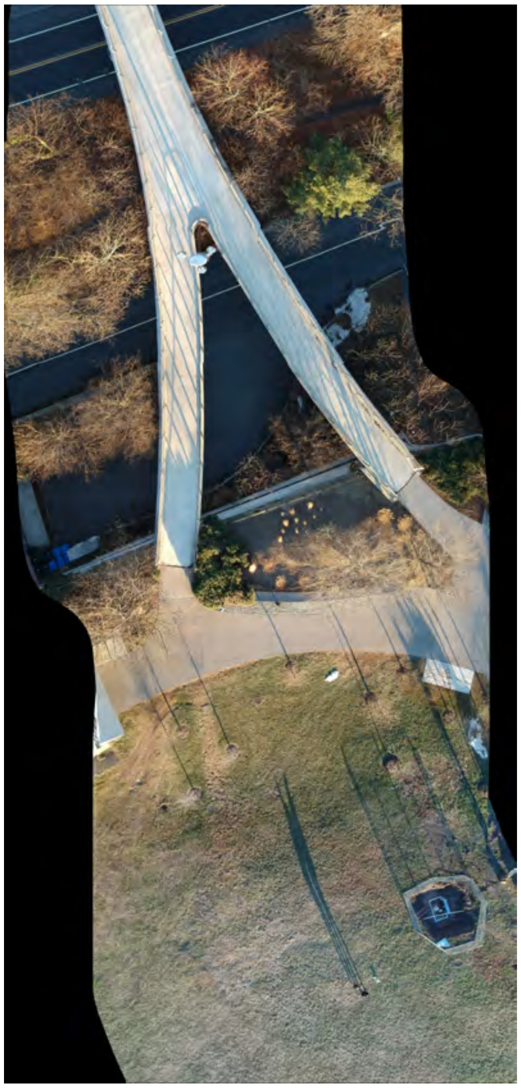


Figure 8: Streicker Bridge: Demonstrates performance on varying terrain styles and unconventional flightpaths

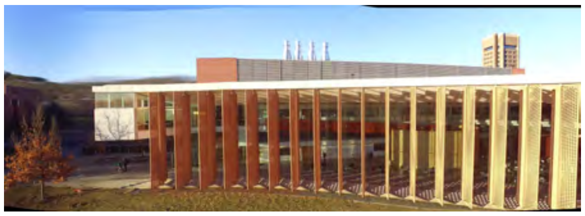


Figure 9: Icahn Labs: Demonstrates vertical footage stitching capability

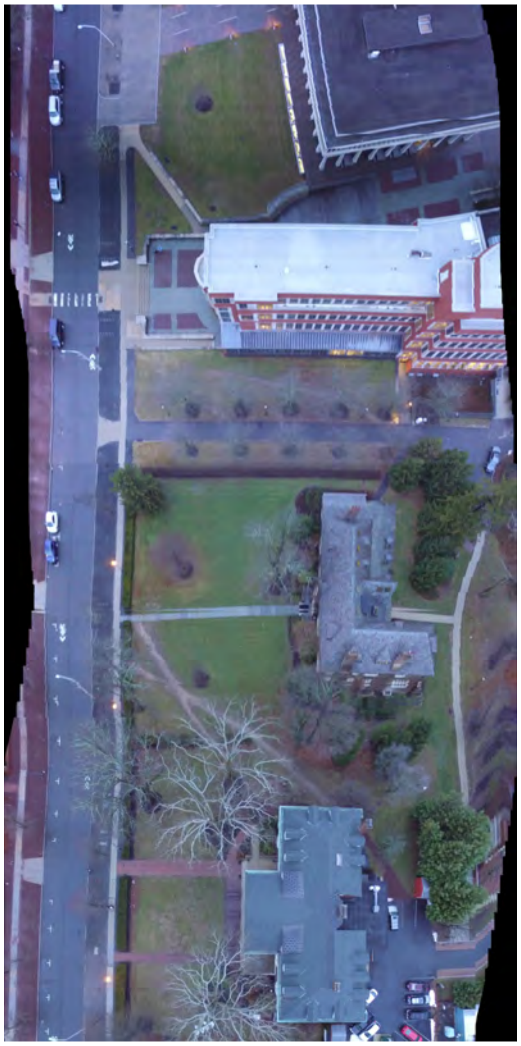


Figure 10: Prospect Avenue North side buildings: Demonstrates resilience to in-flight drone speed changes

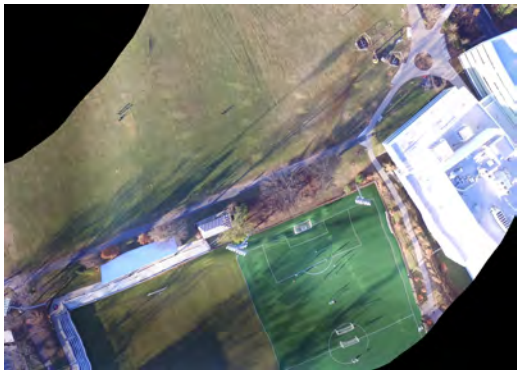


Figure 11: Roberts Stadium: Demonstrates stitching capabilities at high altitude (>130m)

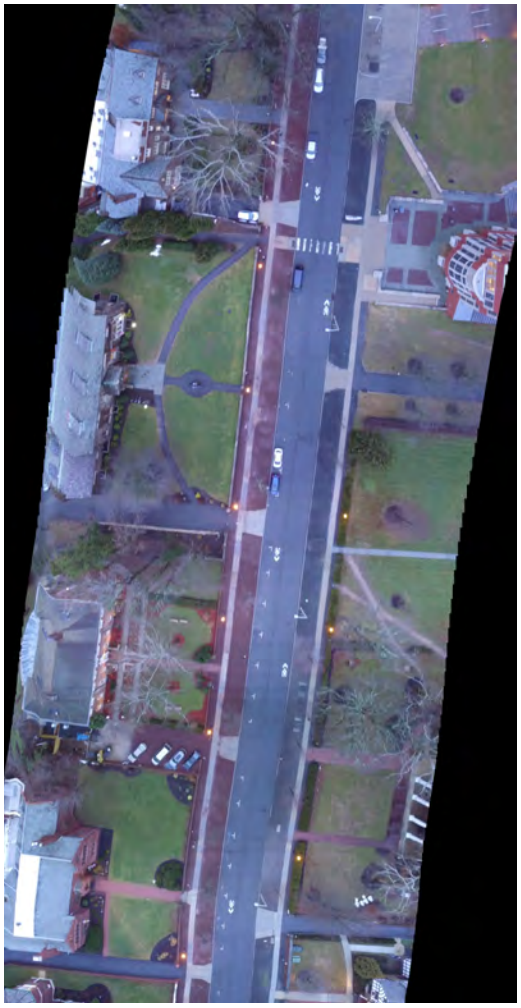


Figure 12: Prospect Avenue: Demonstrates resilience at low altitudes to tall trees and buildings causing potential perspective warping

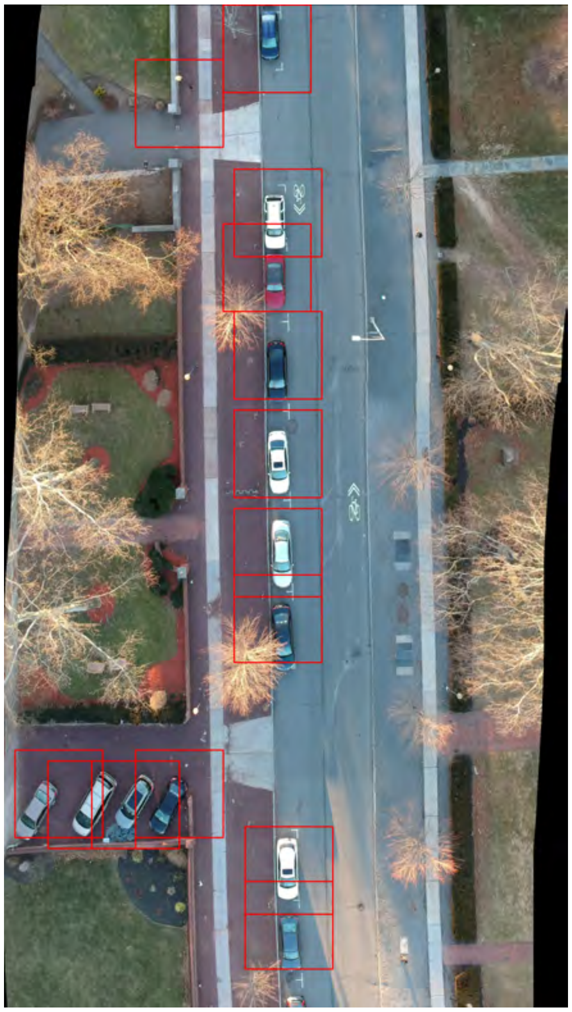


Figure 14: An example of our car classifier using Version 2 of our net. This generated 0 false negatives and 1 false positive

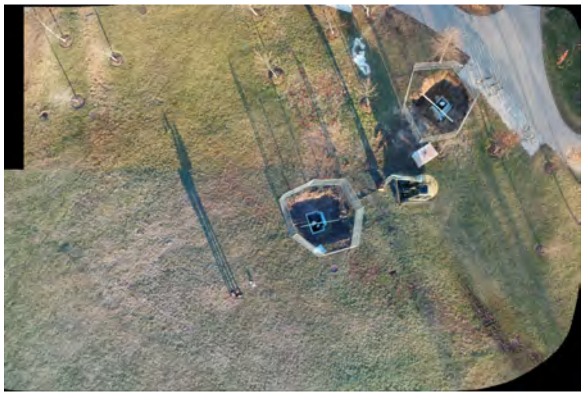


Figure 13: Poe Field: Demonstrates performance on overlapping flyovers of same ground and for surveying applications

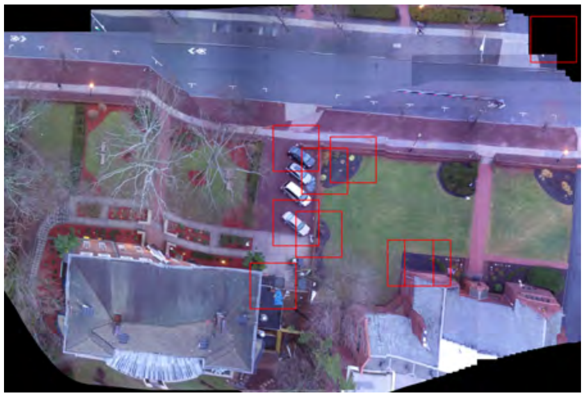


Figure 15: Another car detection example with slightly poorer performance: 1 false negative, 6 false positives

## 6. EVALUATION

We evaluated our project in three stages. First, we qualitatively evaluated the performance of our code. Then, we quantitatively evaluated these results by surveying classmates about their preferences for these maps. Finally, we quantitatively evaluated our extension project by generating precision and recall figures for our cars classifier.

### 6.1 Qualitative Evaluation

We felt, subjectively, that our system was easy to use - a parameter defined at the top of the main python script set the video location, so stitching a new video required only one change and one call to this method. Runtime could be further optimized - stitching a 500-800 frame video took between 20-40 minutes - but still easily outperformed our benchmark MapsMadeEasy, which required an hour for even a 45-frame sequence. In particular, stitching speed was slowed down for grassy shots that could generate many more keypoints due to tiny foliage inflections.

We then evaluated our maps qualitatively. Each of the maps was high-definition; the stitching process didn't affect the sharpness or image quality of the footage. We found that the stitching also, correctly, output a superset of all frames shot - no areas shot were ignored or absent from the resulting maps. Finally, we noted that the quality of maps generated was invariant to lighting conditions (as stitching was successful in bright light, low light, and at sunset) and terrain styles.

We then examined the performance of our stitching output across the variables we played with in creating our dataset. Our generated maps were successful across a variety of speeds - the video for Figure 10 alone was shot at a range of speeds between 5 and 20 mph. Figures 8 and 13 demonstrate the success of our technique for a range of flight paths. 8 involved an unusual boundary path, while 13 incorporated repeated flyovers of the same ground. Figure 9 demonstrated our success in a different shooting orientation (eye-level versus birds-eye). Finally, figures 14, 10, and 11 represented progressively higher shooting altitudes, with only minimal performance degradation and image warping even in the tough wind conditions over 130m.

Finally, we assessed performance in the presence of various complicating factors. First, we noted that the image perspective issues we had expected for tall objects like buildings or trees (namely, that the maps would unnaturally display multiple perspectives on the same artifacts) were largely resolved by our relative-offset approach. Because we stitched together our composite image only once, at the end, more recent images could supersede older ones, meaning that only one flight perspective was rendered. We then noted that object movement (ie. cars and pedestrians) also didn't largely affect our stitched output. This was because the frame sampling approach we used dynamically selected partially overlapping frames - by ignoring frames too close to one another, we reduced the probability of capturing the same moving object multiple times in small offsets. Finally, we noted that performance in cases of radial movement was still imperfect. In the high-altitude shots of figure 11, harsh winds caused the drone to rotate slightly. This caused slight warping of lines in the bottom right quadrant of the image.

### 6.2 Quantitative Evaluation: Flyover

Since there doesn't exist a 'true' baseline for us to algorithmically compare our stitching output to (satellite maps would be occluded in different ways and contain different features), we aimed to compare our maps to those generated by the only competitor: MapsMadeEasy. To do so, we built a series of maps by running Flyover and MapsMadeEasy on an overlapping set of videos. Then, we presented these compiled maps in random, shuffled order to 20 classmates. Each survey respondent was asked to rank each of the images from 1 (poor) to 4 (excellent), without knowing which map belonged to which class.

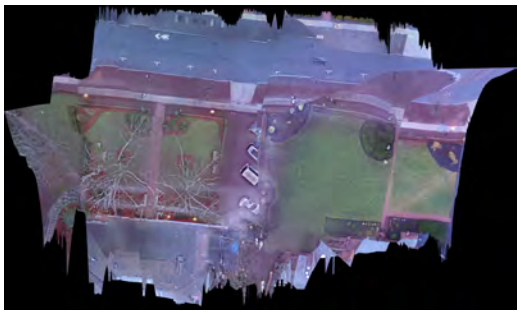


Figure 16: A stitching sample output from MapsMadeEasy. One respondent, Samhita Karnati ́18, called this rendering "a joke."

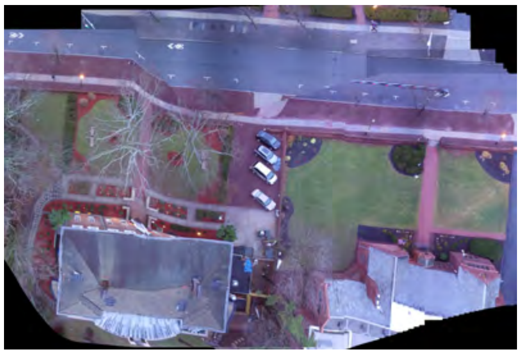


Figure 17: For comparison, the output of Flyover on the same footage as above

Ultimately, we found that the Flyover-generated maps vastly outperformed those made by MapsMadeEasy.

- Flyover maps had an average **rating of 3.4**
- MapsMadeEasy had an average **rating of 1.2**

This resounding difference left us confident of the success of our algorithm. Given that we had generated a mapping system that could handily outperform the only commercial competitor in the space, we concluded that we were achieving high-quality results.

## 6.3 Quantitative Evaluation: Car Classifier

To evaluate our car classifier module, we calculated precision and recall figures for the number of cars identified. This allowed us to get a sense for the efficacy of our system.

- Our classifier had average **precision of 0.70**
- Our classifier had average **recall of 0.94**

Obviously, in an ideal world, these figures would each approach 1.0. However, given the dataset constraints we faced (explained below) and the limitations of our computing resources, we found that this was acceptable performance. Our recall figures were especially high, representing our success at minimizing incidences of false negative reports - almost every car in the maps was identified. Precision was markedly lower, in part because of misidentified walls and roofs in low-light renderings.

## 7. DISCUSSION

We found that Flyover performed surprisingly well across a wide variety of videos in all sorts of lighting, weather, and wind conditions. We discuss some salient characteristics below:

## 7.1 Strengths

Our algorithms performed optimally, as expected, in well-lit settings that included various unique, identifiable objects. Our flyover of Prospect and Olden Avenues, for example, worked well for this reason - that footage contained cars, trees, and buildings that resolved to easily distinguishable keypoints for stitching. In low-light conditions, Flyover was effective at stitching together maps, but the movement of long shadows and rapidly changing light conditions sometimes resulted in stitching evidence in the output. Flyover was also quite effective across a variety of altitudes but performed best in the middle-range of flight heights (40-60m). Speed didn't affect stitching accuracy, as our dynamic frame subsampling approach took care of disproportionately slow or fast flyovers. More broadly, we saw that our system would be particularly useful in high-definition surveying and monitoring applications for which repeated flyovers of busy (feature-rich) areas at medium-to-low altitudes are required.

A variety of post-processing applications are also possible as the maps generated by Flyover were of a sufficiently high quality. Our car classifier performed well across many of the images used as input. In particular, it was strong at identifying colorful cars at any orientation against a traditional asphalt background.

## 7.2 Weaknesses

### *7.2.1 Image Stitching*

Flyover still has room for improvement, though. The software responds poorly to height and rotational variance, both of which often occur at high altitudes when wind speeds can rise and fall dramatically. This is because our relative offset approach anticipates translational movement, so rotations and altitude changes can result in incorrect keypoint offsets and misplaced frame positions. As such, maps stitched from high-altitude drone footage (ie. above 100 meters) can end up being slightly warped.

### *7.2.2 Car Detection*

The main weaknesses of our car detection models were false positive rates and performance. While later iterations on our model were able to produce low false negative rates, they had a tendency to mistake lots of features that weren't cars for cars. In addition, our detection pipeline currently is not even close to being able to run in real-time. Although a real-time detection system was never one of our goals and is arguably overkill for our applications, there is still much to be done to improve the performance of our detector.

## 7.3 Key Contributions

Most existing drone mapping software is very expensive and cannot be used post-flight. Those that do allow for post-flight map stitching often require in-depth flight path metadata submitted via GPS coordinates or EXIF files (which may be difficult to obtain).

Flyover works off-the-shelf, post-flight, is cheap, and only needs flight footage. It outperforms the only competitor with a similar offering, MapsMadeEasy, by a significant margin, and so represents the state of the art in this space. Finally, our car classifier module demonstrates the viability of satellite image-trained neural net models for identifying and classifying features from (relatively low-altitude) drone videos.

## 7.4 Next Steps

### *7.4.1 Image Stitching*

Flyover is a great baseline platform for generating aerial flyover images. One key innovation that we would like to focus on in the future would be to support affine transform motion models that would allow for the drone to have slight imperfections in its flight path (like variations in height and orientation). This would make our system more robust for stitching images in the real world. An affine transform motion model would allow us to account for scale, translation, and rotation versus just the linear translation model implemented currently in our system.

Furthermore, we would also like to integrate more image post-processing capabilities into our system. Over time, as each drone captures video, the camera varies its exposure and ISO settings to record the best image possible. However, when stitching, this will sometimes lead to small color discrepancies over long flight paths. We would like to develop and implement algorithms in our code to self-correct these discrepancies, thereby generating more seamless renderings.

#### 7.4.2 *Car Detection*

The main weaknesses of our car detection system stem from 2 major issues: our dataset and our computing resources.

As we mentioned in earlier sections, the discrepancy between the low-resolution training patches taken from extreme heights and the high-resolution test patches taken from our low-altitude drones was likely a major driver of errors in our classifier. While the UAVs and satellites that took the training images had the same perfectly top-down perspective of all the cars, our test images often had varying perspectives even for different cars in the same stitched image. If we had access to a sufficiently large training set more suited for our application, our results would most likely be significantly better. It's unlikely that a large enough publicly available dataset of cars taken from drone heights exists (otherwise we would've found it), so if we wanted to pursue this we would probably have to capture our own training footage and label the data ourselves.

Even if we were restricted to the COWC dataset that we had, we most likely could have produced better results if we had access to more powerful computing resources (specifically Nvidia GPUs). We trained all of our models on a standard desktop computer with an AMD GPU, meaning we couldn't take advantage of the GPU-enabled version of Tensorflow. Thus, we had to restrict our dataset and model structure to have a reasonable training time. If we had more powerful computing resources, we could try to expand the input size from 48 x 48 to the full 256 x 256 offered in the training data and thus take advantage of the context (cars tend to be near roads, sidewalks, other cars, etc) provided in the image patches to improve accuracy. We could also then train on more images - we weren't using the entire CWOC dataset because training would've taken too long. For reference, while most state-of-the-art convolutional nets train on millions of images, we were only training on tens of thousands of images. More training images would mean that our classifier would likely generalize better and be more robust.

## 8. CONCLUSION

Flyover is a significant technical step towards the generation of 2-D aerial imagery. This software package transforms cheap, off-the-shelf drones into capable systems that can be used everywhere from disaster reconnaissance to municipal planning. This aerial imagery generated is generally clear and usable, even to classify cars; such a feature would be useful in urban planning and traffic engineering.

We would like to thank Professor Russakovsky and our advisor, Kyle, for their help with this project. We would also like to thank Professor Ferencz and Riley for a great semester.

## Notes

[1]https://www.dji.com/
[2]https://cloud.pix4d.com/store/
[3]https://store.dji.com/
[4]http://optica.csic.es/papers/icpr2k.pdf
[5]Lecture 4
[6]https://gdo-datasci.ucllnl.org/cowc/

STATISTICAL LEARNING
Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel s too high. Get some experts and minimze the structural risk in a new one. Rework our loss function, make the next kernel stable, nbiased and consider using a oft margin
NEURAL NETWORKS
STACK MORE LAYERS
LAYERS
LAYERS

Figure 18: Oddly relevant

From: https://i.redd.it/5193db0avbey.jpg