Flying Drone for Package Delivery: Identifying Landing Area at a Home Address

Anshul Rai, Harsh Lal, David Ramirez, Sudhanshu Biyani
Fulton Schools of Engineering
Arizona State University
Tempe, Arizona USA

Abstract—**With the cost of human labor on the rise and the cost of drones dropping, a number of companies are exploring the use of drones for their last-mile of package deliveries. Current methods being explored include lowering packages via a parachute or string. Alternatively packages are delivered to address with a large backyard. The objective of this project was to develop a proof-of-concept using computer vision technology to detect a safe portion of a driveway at a landed house for the delivery of packages by a drone. Our final approach was to use convolutional neural network for semantic segmentation to identify three different classes: driveway, houses, and area of no interest. Once over the target house, our algorithm identifies the driveway closest to the house and iterates towards the driveway for landing. We used the Parrot Bebop drone for our final system. Our final demo successfully demonstrated the system in action.**

I. Introduction

For the past decade or more, flying unmanned aerial vehicles (UAV) have been used for military applications. Fixed winged vehicles used for surveillance, and later military strikes, were restricted from the private sector. More recently, flying vehicles with 4 or more helicopter like engines have emerged for consumer purchase as toys and for industrial applications. Both military vehicles and consumer vehicles with fixed wings or quad-rotor type layouts now all fall under the same name: drones.

The maturing technology and shrinking costs of such drones are opening new markets in commercial applications. One such application is for delivery to home addresses. A drone could be launched from a static location or mobile delivery automobile, and asked to delivery products. Such products might include online shopping, groceries, pharmaceuticals, or fast food. Large tech companies, like Amazon, and startups alike are researching and developing future systems for this application. This new application for drones comes with new challenges though. Although drones may have historically had simple autopilot programs, more advanced fully autonomous systems may be needed to make these technologies cost effective. By taking a dedicated human pilot or delivery person out of the loop, drones could be very cost efficient.

A full autonomous autopilot system would need to be very smart to overcome all possible obstacles out in the world. Testing and certifying the safety of such a system would also be very costly. For the scope of this school semester-long project, we chose to approach a particular and limited problem involved with drone home package delivery.

We have developed a system which can identify a delivery location at a home address and land to make such a delivery.

II. Approach

When a flying drone reaches a home residence, a delivery location must be identified. Where would the ideal delivery location be? Perhaps in a backyard, so as to protect the package from theft. A literature review showed that some prototype systems indeed try to drop parachute packages into large backyard areas [1]. But this might not be the best location if the backyard lot has dogs/pets or a watering sprinkler system which might damage the package. Traditionally, package deliveries from FedEx or UPS are made at the front door. Due to the high flight geometry of a delivery drone, identifying a front door can be difficult. A roof overhang commonly covers exterior doors. The choice was made that our system would make deliveries to an open driveway of a home address. This avoids possible property damage to the home. Additionally, a car in the driveway can also be avoided by a safe distance.

We choose an iterative approach for our flying drones perception and control algorithm. After a high-flying drone reaches the approximate location of the home, via GPS, it will then attempt to find the home. Our novel computer vision algorithm does a course semantic segmentation to differentiate the home from other objects in the environment. Next the drone looks for the closest driveway to said home. The drone then attempts to move itself directly above this driveway. With a clear line of sight on the driveway, the drone then descends slightly. The drone iteratively repeats this process to narrow in on the driveway: observe, identify, move, and descend. When the drone reaches a certain altitude, a secondary decision algorithm makes sure a safe landing area exists before the drone tries to land.

In this way the drone shows as a proof of concept that it can make a landed package delivery. Our operational drone is not actually capable of carrying objects though. To do so would require more powerful lift from the motors, as well as other aerodynamic concerns. Due to our limited budget, we choose very modest consumer off the shelf (COTS) drones for system development.

A. Drones

Our first development drone was a Parrot AR Drone 2.0. Parrot is a French company which has a large market share of

Fig. 1. The older Parrot AR Drone 2.0.

Fig. 2. The newer and higher quality Parrot Bebop 2 drone.

the consumer drone market. The AR Drone 1.0 was the first consumer drone with an HD video camera which could stream video in real-time to a pilots smartphone. The AR Drone combines this 720p front facing camera with a poor quality downward facing camera. This 480 by 360 pixel camera was initially the best we had for our computer vision algorithm. Both cameras had no stabilization, so the video feed was commonly very jerky. The AR Drones are no longer being produced by Parrot, but there is numerous open source libraries available for controlling these drones.

Dr. Yang was generous enough to purchase and loan a much more powerful drone for our final system. The Parrot Bebop 2 drone is noted as the current preferred computer vision research drone. This drone combines a static 180 degree field of view with 14 megapixel camera. This allows for visual capture directly below the drone and well above the drone. The drones video feed portions off a section of this fish-eye lens capture, which allows our drone to consistently look straight down. Instead of mechanical stabilization on the video feed, via gimbals, the drone video has real time digital stability postprocessing. These features were key in the successful completion of our prototype system.

B. *Perception Module*

Our initial computer vision research explored two methods: YOLO based object detection [2] and superpixel based semantic segmentation.

Object detection in an image involves placing bounding box identifier over one or more desired instances of your object classes. Current methods for object detection commonly involve using convolutional neural network (CNN) classifiers to make predictions on specific regions in the image. This process is repeated many times throughout the image at different proportions and scales. These traditional CNN object detection frameworks are extremely computationally inefficient.

Yolo is a state of the art object detection framework which greatly reduces this computation. First, an image for object detection is passes as a whole to a CNN. The CNN architecture used can vary, and many of the popular ResNet, Inception, etc have been tried. Instead of returning a binary or categorical class from the CNN classifier, the CNN instead returns a spatially relevant 3-dimensional feature-map. Two dimensions account for the image-like spatial information, and the third accounts for many outputs of different convolutional operations. This intermediate feature-map is then used by the Yolo system to produce bounding boxes. In this two stage system, the CNN detects relevant features related to the objects of interest, and Yolo sorts this information with a dense fully connected neural network. Yolo stands for You Only Look Once, a fitting acronym for this algorithm. As with any artificial neural network (ANN), the algorithm must be trained with related data. Most CNNs take an image as input, and output a vector with elements equal to the number of possible object types. For Yolo, as with other CNNs, the input vector is an image, but the output of a Yolo system is much different. The output vector must contain all the true bounding boxes of the object classes contained in the image, whereby Yolo uses a unique cost function for reducing the training error. For this reason, training Yolo with new data can be more challenging and more time consuming. Instead of simply hand labeling your training data with object class information, you must also hand draw all the bounding boxes for all object instances.

The second computer vision method which was investigated was semantic segmentation using superpixels. Different from CNN classifiers and object detectors, semantic segmentation attempt to assign object classes to all the pixels in an image. In general, this means more precise information than a bounding box. Instead of independently classifying every pixel in an image, we instead use a very coarse resolution. We use the term superpixels to describe the square groups of pixels used for semantic segmentation. These superpixels are used to train a CNN classifier with semantic classes based on the drone aerial imagery.

For training our CNN based computer vision methods, we used several sets of data during development. Initially, we used open source satellite imagery. Next we used low resolution video capture frames from the AR Drone. Finally, we used high resolution Bebop drone video frames. The performance of these methods on the different data types will be discussed in detail in the experimental results section. The final drone system design uses only the semantic segmentation method trained only with the Bebop drone imagery.

C. ROS System Setup

Robot Operating System (ROS) is used for autonomous control of our Parrot Bebop. We use an existing driver package called bebop_autonomy for communicating with the drone [3]. Publisher and subscriber nodes are developed to send commands to the drone using the topics provided by the driver node.

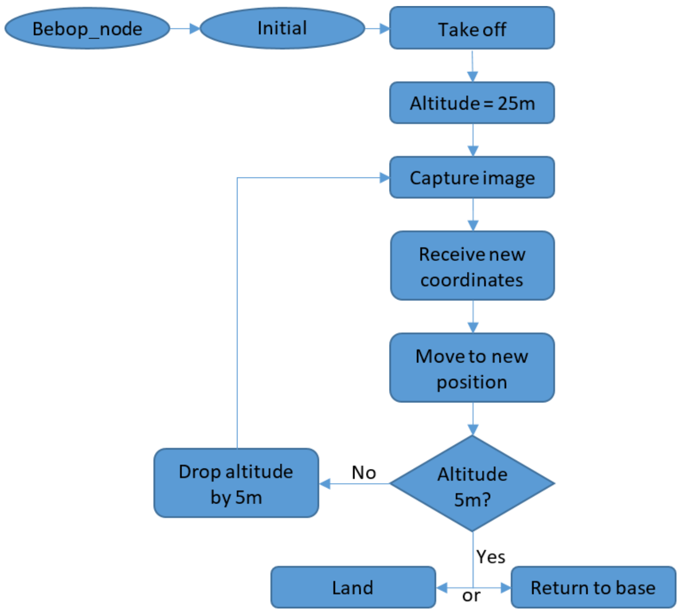


Fig. 3. Flowchart showing the ROS and decision algorithm process.

Figure 3 shows the process flow through the ROS nodes. The sequence starts with the bebop_node.launch which is the one of the drivers for the Parrot Bebop. This initializes the communication link between the ground station (PC) and the drone. It also calibrates the camera using a .yaml file, which is similar to an XML structured data file. We then run the initial.launch file which is used to perform safety checks and calibrate the sensors based on level ground.

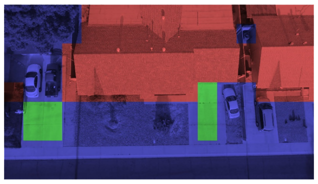
After the safety process is complete, the drone is commanded to takeoff and hover at a height of 1m from the ground. The Altitude node is then run to bring the drone up to an altitude of 25m. The control loop in this node allows the drone to hover at that altitude without causing much drift in the position. The node also outputs an image which is sent through the Semantic Segmentation algorithm which outputs new x,y-coordinates for the drone.

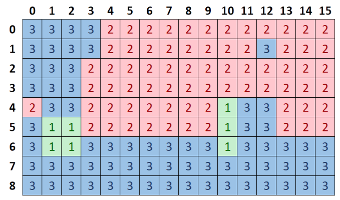
A sequence of Position loops are then run to bring the drone to a specified position which is based on these x,y-coordinates. Also, each position loop allows the drone to drop down by a height of 5 m. When the drone is just 5m above ground level, the drone is commanded to land without any change in its x,y-position.

D. Control and Decision Algorithm

The control and decision algorithm can essentially be broken into three parts. The entire process flow is shown in the flowchart on Figure 3.

The decision algorithm starts when the drone has reached the desired GPS coordinates. To ensure we detect the right driveway associated with the correct house, it is important to center the drone over the target house first. Figure 4 shows the classification output from the CNN that is used in the decision algorithm as a 9x16 array.

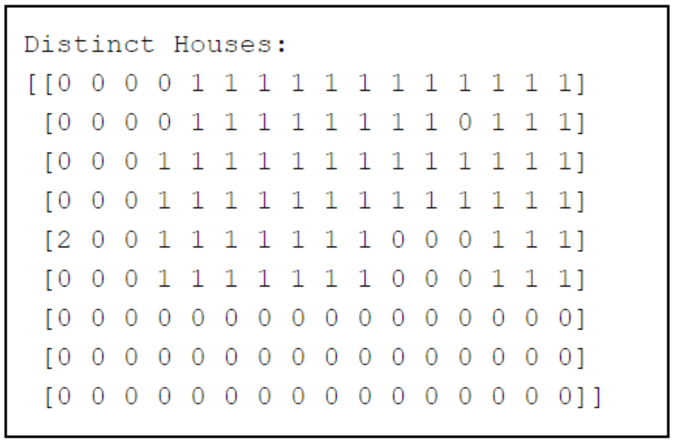




	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
1	3	3	3	3	2	2	2	2	2	2	2	2	3	2	2	2
2	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
4	2	3	3	2	2	2	2	2	2	2	1	3	3	2	2	2
5	3	1	1	2	2	2	2	2	2	2	1	3	3	2	2	2
6	3	1	1	3	3	3	3	3	3	3	1	3	3	3	3	3
7	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
8	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

Fig. 4. Left: Prediction of classes from drone when at GPS coordinates. Right: Equivalent array used for decision algorithm.

The algorithm first detects the number of distinct houses and labels them. In the example above, two distinct houses are detected and are labeled as shown in Figure 5.

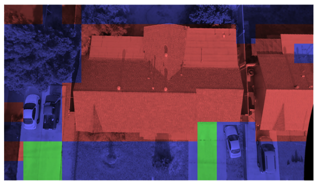


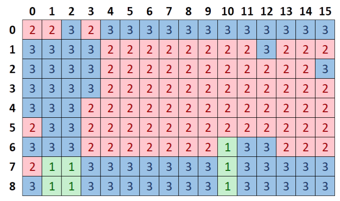
```
Distinct Houses:
[[0  0  0  0  1  1  1  1  1  1  1  1  1  1  1  1]
 [0  0  0  0  1  1  1  1  1  1  1  1  0  1  1  1]
 [0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1]
 [0  0  0  1  1  1  1  1  1  1  1  1  1  1  1  1]
 [2  0  0  1  1  1  1  1  1  1  0  0  0  1  1  1]
 [0  0  0  1  1  1  1  1  1  1  0  0  0  1  1  1]
 [0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
```

Fig. 5. Output of image with distinct houses labeled.

Next the center of mass for these labels are computed and the euclidean distance to the center of the image frame is calculated. The house closest to the center of the image is saved as the target_house and the relevant X,Y coordinates are written to a CSV file to be read by ROS.

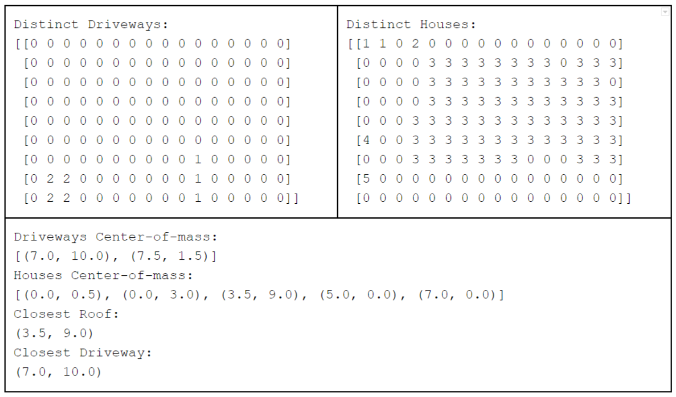
Once the drone is centered over the target_house, the second part of the decision algorithm starts. In a similar method as above, the distinct driveways and houses are labeled based on the CNN prediction as shown in Figure 6.





	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	2	2	3	2	3	3	3	3	3	3	3	3	3	3	3	3
1	3	3	3	3	2	2	2	2	2	2	2	2	3	2	2	2
2	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	3
3	3	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2
4	3	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
5	2	3	3	2	2	2	2	2	2	2	2	2	2	2	2	2
6	3	3	3	2	2	2	2	2	2	2	1	3	3	2	2	2
7	2	1	1	3	3	3	3	3	3	3	1	3	3	3	3	3
8	3	1	1	3	3	3	3	3	3	3	1	3	3	3	3	3

Fig. 6. Left: Prediction of classes from drone once centered to target house. Right: Equivalent array used for decision algorithm.



```
Distinct Driveways:
[[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 2 2 0 0 0 0 0 0 0 1 0 0 0 0 0]
 [0 2 2 0 0 0 0 0 0 0 1 0 0 0 0 0]]
```

```
Distinct Houses:
[[1 1 0 2 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 3 3 3 3 3 3 3 3 0 3 3 3]
 [0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 0]
 [0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3]
 [0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3]
 [4 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3]
 [0 0 0 3 3 3 3 3 3 3 0 0 0 3 3 3]
 [5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

```
Driveways Center-of-mass:
[(7.0, 10.0), (7.5, 1.5)]
Houses Center-of-mass:
[(0.0, 0.5), (0.0, 3.0), (3.5, 9.0), (5.0, 0.0), (7.0, 0.0)]
Closest Roof:
(3.5, 9.0)
Closest Driveway:
(7.0, 10.0)
```

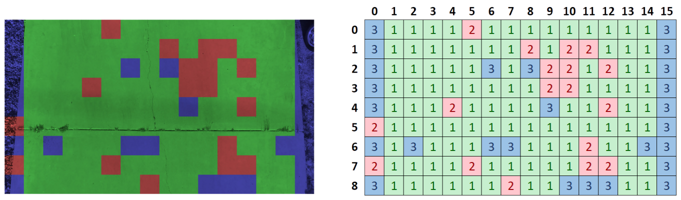
Fig. 7. Output of image with distinct driveways and houses labeled as well as their center-of-mass.

Once again, the center of-mass for each distinct house is computed and the house with the smallest Euclidean distance from the center of the frame is saved as the closest_house. Following that, the Euclidean distance between each driveway and the closest_house is computed. The driveway with the smallest distance is saved as the closest_driveway. The relevant X,Y coordinates are then written to the CSV file to be read by ROS.

After the drone has moved to the new X,Y coordinated and descended 5m, this process is repeated until the drone has reached an altitude of 5 meters.

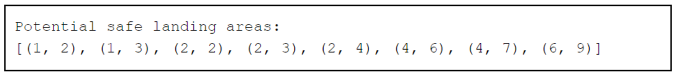
Once the drone reaches an altitude of 5 meters, the last part of the algorithm kicks in. The algorithm needs to determine if there is sufficient portion of the only_driveways is detected. This is achieved by using a 3x3 array sliding window. An array size of 3x3 was used as at 5 meters, this would equate to an area of approximately 1.6m x 1.6m on the ground, which was deemed to be sufficient clear space for the drone to land.

When sliding the window, the total of that 3x3 array is computed. If the total is 9 (i.e. all elements are 1 in the 3x3 array) then the index of that window is stored in a list of potential landing areas as shown in Figure 9.



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	3	1	1	1	1	2	1	1	1	1	1	1	1	1	1	3
1	3	1	1	1	1	1	1	1	2	1	2	2	1	1	1	3
2	3	1	1	1	1	1	3	1	3	2	2	1	2	1	1	3
3	3	1	1	1	1	1	1	1	1	2	2	1	1	1	1	3
4	3	1	1	1	2	1	1	1	1	3	1	1	2	1	1	3
5	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3
6	3	1	3	1	1	1	3	3	1	1	1	2	1	1	3	3
7	2	1	1	1	1	2	1	1	1	1	1	2	2	1	1	3
8	3	1	1	1	1	1	1	2	1	1	3	3	3	1	1	3

Fig. 8. Left: Prediction of classes from drone once at 5m altitude. Right: Equivalent array used for decision algorithm.



```
Potential safe landing areas:
[(1, 2), (1, 3), (2, 2), (2, 3), (2, 4), (4, 6), (4, 7), (6, 9)]
```

Fig. 9. List of potential safe landing areas of size 3x3.

Finally, the area closest to the center of the frame is determined using Euclidean distance and saved as the landing coordinates. These coordinates are saved to the CSV file to be read by ROS. If no clear area to land is found, then drone is commanded to return back to base.

Before saving any coordinates to the CSV file, the relevant array index need to be converted to real world X and Y distances which can be sent to the drone for movement. Since we did not have the focal length of the camera used on the Parrot Bebop 2 drone, we used a simple calibration process to derive the equation that would define the scaling required for a given altitude.

We used a 20.5cm x 20.5cm black image placed on the ground and flew the drone over it at various heights. After that, we measured the number of pixels that would represent the edge length at various altitudes and plotted the results in excel as shown in Figure 10.

E. Data Collection

Initially, to collect data for our Semantic Segmentation and YOLO training, we used downloaded high resolution satellite images from the U.S. Geological Survey (USGS) website [4]. However, this image was still far to course when it was zoomed in to simulate the drone at lower altitudes. We also collected some data from out first AR Drone. Unfortunately, the quality of the video was also quite poor and lacked detail at high altitudes. We also found that these two data types was not similar enough. When training our CNN methods, our validation performance quickly diverged from your training accuracy.

Luckily, we received the Parrot Bebop 2 drone. With its 16MP HD camera, and automatic video stabilization features, we were able to capture video with significant details. We used this data exclusively for both training and validation performance. To collect the data, we flew the drone over concrete footpaths, roads and fields on the ASU campus. Additionally, to get the roofs of houses, we flew the drone over a few house near one of the authors residence. Due to privacy concerns, we were only able to collect a limited variety of data samples. Because of our small sample size of houses, we do not expect our CNN methods would generalize well to new houses.

F. System Integration

Given the weights of the trained Image Segmentation Neural Network and an image frame of size 1080x1920x3 from the drone video we first take each of the 144 x (120,120, 3) and resize it to (128, 128, 3). This is done as our model is pre-trained on chips of size (128, 128, 3). After operating on the 144 (120, 120, 3) chips and making them a size of (128, 128, 3) we create a 4d array of size (144, 128, 128, 3) which is then sent to the model. Our resultant output is a (144, 1) vector where each unit has the value of the class that our model thinks is the most probable element in that chip.

This output vector is reshaped to a (9,16) array and sent to the heat-map and decision algorithm modules simultaneously. The output frames from the heat-map module are concatenated in a video object to create a video of heat-map output. The

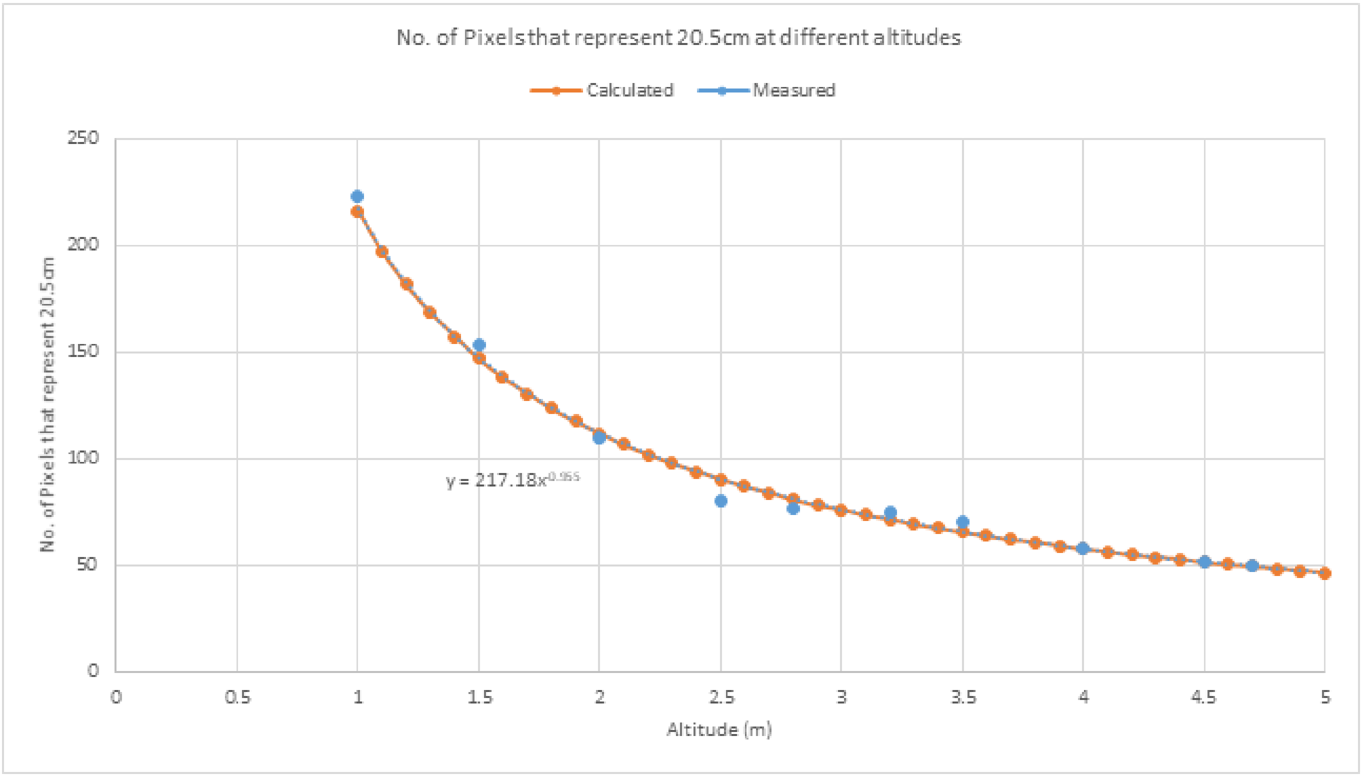


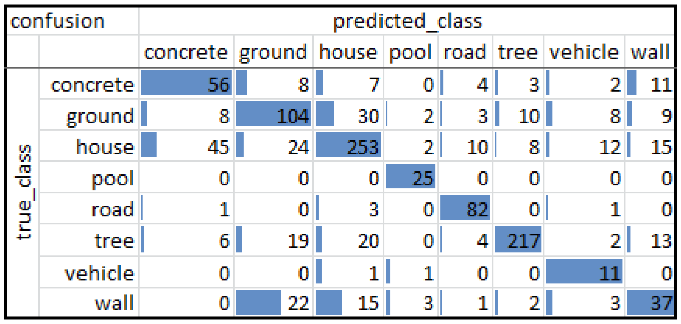
Fig. 10. Plot of number of pixels that represent 20.5cm at different altitudes.

ROS module operates independently from the rest of the system. Once we move to a new location, the ROS module sends us the next set of frames after which the entire process is repeated.

III. Experiments

A. Computer Vision Experiments

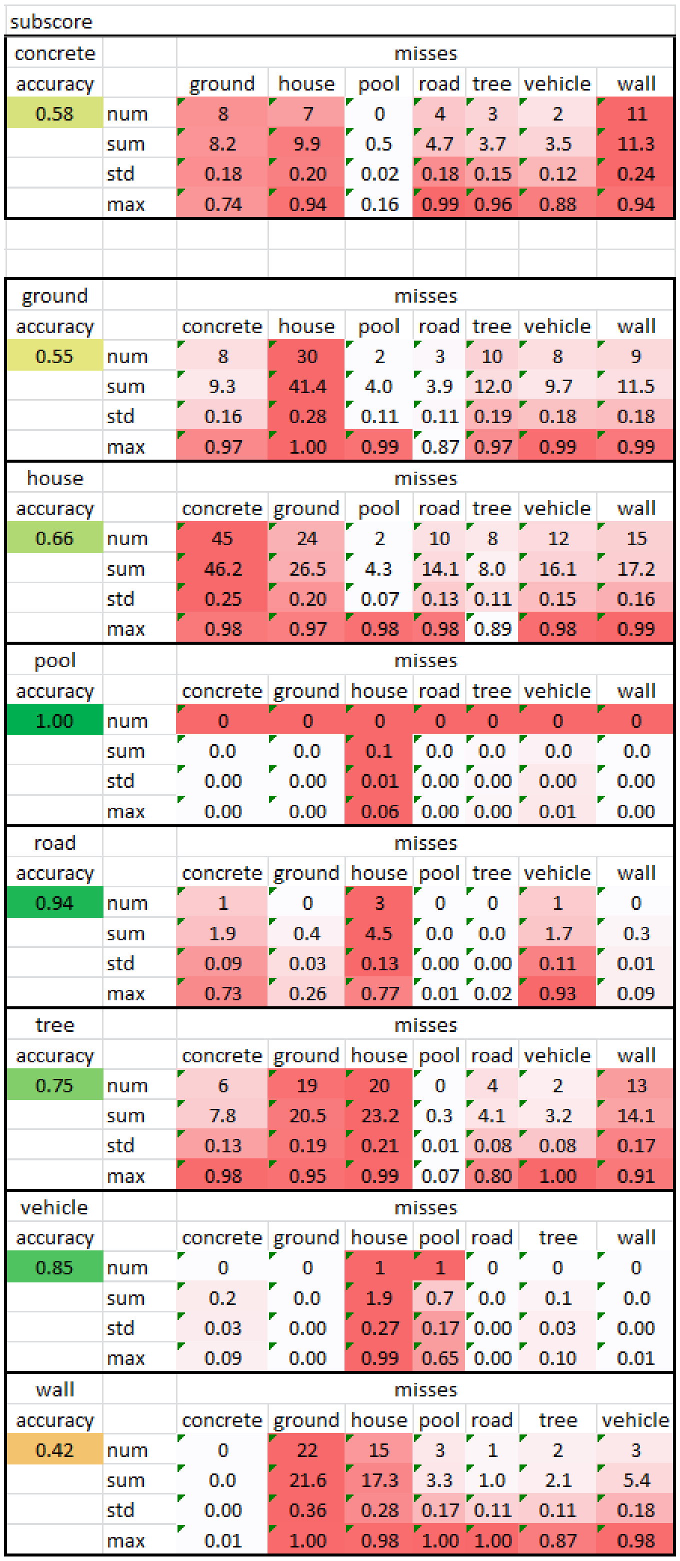
We initially experimented with both our Yolo Object Detection module and Image Segmentation module on manually annotated images that had 7 classes. These classes were: house/roof, concrete/driveway, cars, pools, trees, other ground cover, and walls. For the semantic segmentation method, our wall class had low performance, but all other classes performed well. See Figure [fig] for semantic segmentation performance. The Yolo method we tried was trained with similar classes: house/roof, driveway, cars, trees, and pools. The catch all ground cover class was not used in the Yolo method. We soon decided not to continue the Yolo method. We did not have enough labels for each unique class to correctly train a Yolo CNN. Though Yolo could recognize the presence of an object of interest in a segment of the image, it couldnt estimate and bound its position and size to a satisfiable degree. For these reasons we decided to continue iterating over our Image Segmentation approach.



confusion		predicted_class							
		concrete	ground	house	pool	road	tree	vehicle	wall
true_class	concrete	56	8	7	0	4	3	2	11
	ground	8	104	30	2	3	10	8	9
	house	45	24	253	2	10	8	12	15
	pool	0	0	0	25	0	0	0	0
	road	1	0	3	0	82	0	1	0
	tree	6	19	20	0	4	217	2	13
	vehicle	0	0	1	1	0	0	11	0
	wall	0	22	15	3	1	2	3	37

Fig. 11. Confusion matrix for first generation semantic segmentation CNN.

Once we fully committed to only a semantic segmentation method, our CNN was retrained with the Bebop drone chips. This was just as challenging to label as the satellite imagery, but not as difficult as the lower quality AR Drone chips. Of the dozen videos we had for training data, we pulled an image out out every several hundred frames. These were then chipped as described above. The resulting superpixels/chips were then labeled by hand into their classes. For our final design we only used 3 classes: house/roof, driveway, and everything else. We selectively avoided partial concrete examples for only those with concrete almost entirely covering the chip. This allows us to better avoid obstacles. In total we hand labeled approximately 40,000 chips in into these three classes. We made sure to include an even number of examples in each of our classes. We then split the data into a training and validation set. Our final CNN uses the fully convolutional MobileNet



subscore								
concrete		misses						
accuracy		ground	house	pool	road	tree	vehicle	wall
0.58	num	8	7	0	4	3	2	11
	sum	8.2	9.9	0.5	4.7	3.7	3.5	11.3
	std	0.18	0.20	0.02	0.18	0.15	0.12	0.24
	max	0.74	0.94	0.16	0.99	0.96	0.88	0.94
ground		misses						
accuracy		concrete	house	pool	road	tree	vehicle	wall
0.55	num	8	30	2	3	10	8	9
	sum	9.3	41.4	4.0	3.9	12.0	9.7	11.5
	std	0.16	0.28	0.11	0.11	0.19	0.18	0.18
	max	0.97	1.00	0.99	0.87	0.97	0.99	0.99
house		misses						
accuracy		concrete	ground	pool	road	tree	vehicle	wall
0.66	num	45	24	2	10	8	12	15
	sum	46.2	26.5	4.3	14.1	8.0	16.1	17.2
	std	0.25	0.20	0.07	0.13	0.11	0.15	0.16
	max	0.98	0.97	0.98	0.98	0.89	0.98	0.99
pool		misses						
accuracy		concrete	ground	house	road	tree	vehicle	wall
1.00	num	0	0	0	0	0	0	0
	sum	0.0	0.0	0.1	0.0	0.0	0.0	0.0
	std	0.00	0.00	0.01	0.00	0.00	0.00	0.00
	max	0.00	0.00	0.06	0.00	0.00	0.01	0.00
road		misses						
accuracy		concrete	ground	house	pool	tree	vehicle	wall
0.94	num	1	0	3	0	0	1	0
	sum	1.9	0.4	4.5	0.0	0.0	1.7	0.3
	std	0.09	0.03	0.13	0.00	0.00	0.11	0.01
	max	0.73	0.26	0.77	0.01	0.02	0.93	0.09
tree		misses						
accuracy		concrete	ground	house	pool	road	vehicle	wall
0.75	num	6	19	20	0	4	2	13
	sum	7.8	20.5	23.2	0.3	4.1	3.2	14.1
	std	0.13	0.19	0.21	0.01	0.08	0.08	0.17
	max	0.98	0.95	0.99	0.07	0.80	1.00	0.91
vehicle		misses						
accuracy		concrete	ground	house	pool	road	tree	wall
0.85	num	0	0	1	1	0	0	0
	sum	0.2	0.0	1.9	0.7	0.0	0.1	0.0
	std	0.03	0.00	0.27	0.17	0.00	0.03	0.00
	max	0.09	0.00	0.99	0.65	0.00	0.10	0.01
wall		misses						
accuracy		concrete	ground	house	pool	road	tree	vehicle
0.42	num	0	22	15	3	1	2	3
	sum	0.0	21.6	17.3	3.3	1.0	2.1	5.4
	std	0.00	0.36	0.28	0.17	0.11	0.11	0.18
	max	0.01	1.00	0.98	1.00	1.00	0.87	0.98

Fig. 12. Performance metrics by subclass for first generation semantic segmentation CNN.

architecture [5]. We used an adjustable width factor Alpha of 0.25 that of normal, and imported kernel weights pre-trained on ImageNet. Because of the similarity between video frames in our small sample of videos, the CNN trained very quickly. We monitored validation accuracy in order to avoid overfitting. The resulting CNN performed reasonably well on the test video shown in class. We do not have accuracy metrics for this video, as each and every frame was not labeled with ground truth data.

B. *Autonomous Control Experiments*

Our initial experiments with the Parrot ARDrone 2.0 caused a lot of drift in the position of the drone during takeoff. It would drift quite a lot if there was even a slight wind present which prompted us to use the new, more stable Parrot Bebop. The flight stability issue was taken care of by the Parrot Bebop.

The next problem encountered was with the ROS driver for this drone which kept crashing multiple times during flight. This issue was partly due to the fact that the drone assigns a different IP address to every PC its connected to and a different one to its controller. On correcting this problem in the .launch file of the driver, we were able to get a more stable driver with lesser communication losses during flights.

Except for the ROS driver, testing the Bebop indoors did not present a major challenge as we tested it in a zone of about 600 sq.feet in area and 50 ft. in height which was sufficient for testing the autonomous flight system.

Initial experiments involved sending manual commands to the drone to see how quick it was able to react. For position and altitude control, we used the cmd_vel topic to send 3 linear and 3 angular velocity commands to the drone. We kept the linear velocities well below the maximum limit to help us observe its behaviour in flight and also prevent any communication losses. We then created a positional loop within the Position and Altitude nodes to help the drone achieve the desired position causing very little drift. We varied the Kp (Proportional gain) values to control how fast and accurately the drone achieved its desired position, finally settling on Kp = 0.1 for x,y-positions and Kp = 0.5 for z-position. This caused the drone to settle on the required height much faster but, with greater possibility of error in the z-position compared to the x,y-position which we wanted to keep as accurate as possible for our application.

References

[1] "An amazon drone has delivered its first products to a paying customer." [Online]. Available: https://www.technologyreview.com/s/603141/an-amazon-drone-has-delivered-its-first-products-to-a-paying-customer/

[2] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[3] [Online]. Available: https://bebop-autonomy.readthedocs.io/en/latest/

[4] "Earthexplorer." [Online]. Available: https://earthexplorer.usgs.gov/

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861