–Independent Work Report Fall,2013–

Detecting Objects Using Google Street View Data

Brian Matejek

Abstract

Over the last decade, new advances in computer vision have led to improved object detection and recognition software. During this time, Google has collected large amounts of image and LIDAR data using Google Street View cars. This paper describes new algorithms that combine the existing object recognition software and Google Street View data sets to produce the locations of certain objects in New York City.

1. Introduction

Combining Google Street View (GSV) data and existing object recognition software, this paper addresses two new methods for determining the real locations of objects at a street intersection. Focusing on traffic lights and stop signs, the goal of this independent project is to improve on current object recognition software by implementing a Hough voting algorithm and defining a system of linear equations. Further, these created programs should differentiate between unique objects so that a traffic light seen in two different images is recognized as one traffic light.

I evaluate existing software and the new programs by determining the average precision (AP) for segments of New York City. In this paper, recall corresponds to the percentage of unique objects that have been found in a particular run. If there are three traffic lights at an intersection, and four images all see that same traffic light, the recall is 33.3%. This extends from the paper's goal of identifying unique objects at a street intersection.

This paper focuses on two algorithms to identify objects in New York City. The first implements a Hough voting procedure that allows images to corroborate what they see. The existing object detection software produces probabilities that certain pixels captured an input object. However, there is more information available with the Google Street View data. It is possible to determine all of the locations that the pixel possibly could have captured using the camera viewpoint and direction. If multiple pixels all have a high probability of having captured an object of interest, and they all “see” the same location, it is highly probable that location corresponds to the location of the object, and that all of these pixels saw the same object.

The second algorithm creates a large system of linear equations to define the object recognition problem. Every pixel in every image and every location in a grid space becomes a variable. There are smoothing equations to prevent large jumps in probabilities between adjacent pixel and grid variables. If a pixel could possibly have “seen” some set of grid locations, the sum of the probabilities of those grid variables equals the probability that the pixel captured an object. The probabilities from the existing object detection software provide a starting value for the pixel variables. After solving the system of linear equations, the grid variables with higher values have a higher probability of containing the object.

2. Existing Research

2.1. Google Street View Data

Google Street View cars drive around various cities taking pictures of the surrounding area and collecting LIDAR data with a pair of scanners perpendicular to the car path. Eight cameras create a panorama around the car while a ninth camera takes pictures facing skyward. Figures 1 and 2 show a New York City intersection as captured by the LIDAR scanners and a camera respectively.

The data I worked with corresponded to 3.35 square kilometers of Manhattan and contained 150,525 images. These images are split into 20 runs corresponding to different times that the GSV car collected the data. The GSV car has a set of GPS trackers, accelerometers, and inertial meters that try to locate the precise location of the car as it travels along its path. However, interference from the New York City buildings and errors in the integration of the accelerometers produce a path that varies from the car’s true route. There are a series of programs created by programmers at Google and in Princeton’s Graphics Lab that align the car’s path by solving a system of nonlinear equations given input correspondences between the images and the LIDAR data. Since collecting

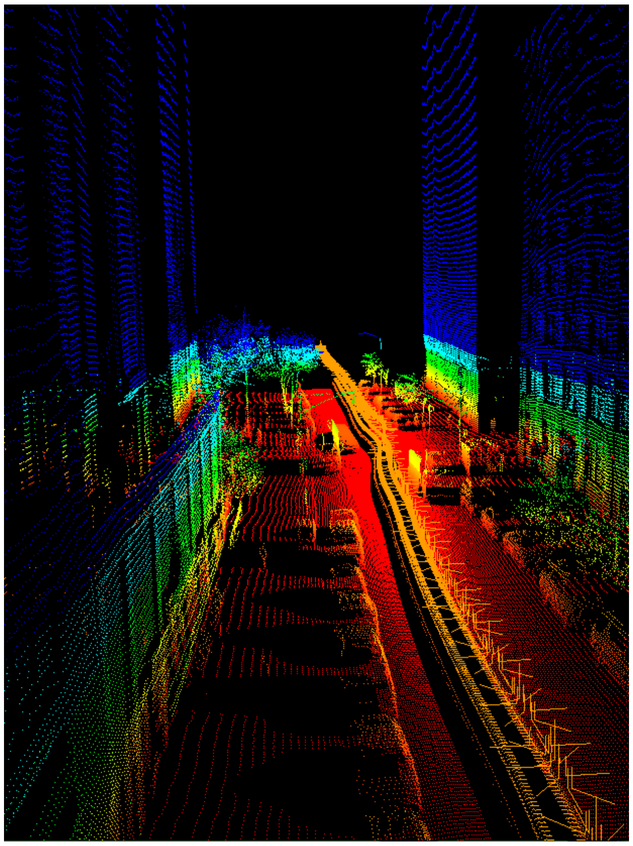


Figure 1: Representation of the LIDAR data for one small segment of New York City.

Figure 2: A typical GSV image that creates part of a panorama.

truth data to align the path is very time intensive, only some of the runs have been aligned.

Although the New York City data set provided by Google contained 20 different runs corresponding to different times and places the driver entered certain parts of Manhattan, I focused on only 14 of the runs and used seven of them to train and validate the algorithms discussed in this paper and the remaining seven to test the algorithms. These fourteen runs contain 45,315 images, 440 traffic lights and 70 stop signs. Runs 1, 3, 9, 10, 14, 17, and 19 trained and validated the algorithms in this paper, and runs 0, 2, 4, 6, 7, 11, and 15 tested the algorithms. Table 1 contains the statistics for these fourteen runs.

Run	Images	Volume (m^3)	Cross Sectional Area (m^2)	Traffic Lights	Stop Signs
0	10,260	23,636,043.74	313,734.61	59	12
1	4,644	51,480,860.37	800,174.40	85	3
2	7,236	38,171,330.88	311,508.03	41	20
3	1,287	5,333,674.03	85,056.49	10	9
4	1,854	5,225,889.79	86,737.13	13	8
6	3,285	10,110,584.21	120,402.14	16	2
7	3,618	12,340,787.56	134,224.64	26	2
9	621	2,137,314.64	41,130.64	13	2
10	279	764,490.48	16,269.72	4	0
11	2,484	52,375,588.77	677,378.31	67	0
14	4,284	32,521,359.20	591,221.33	54	8
15	3,303	21,458,602.94	299,776.48	31	0
17	927	2,088,379.96	41,081.57	7	0
19	1,233	3,335,317.20	71,759.23	14	4

Table 1: The data corresponding to some of the GSV runs in New York City.

2.2. Object Recognition in Images

Many computer scientists have researched how to detect and classify objects in two dimensional images. Although the field has progressed greatly in the last decade, it is still a difficult task that is far from perfect. To detect objects in the GSV images, I adapted the object detection software of Pedro Felzenszwalb, Ross Girshick, et. al. that creates a deformable parts based model.[2][3][4] The work won the PASCAL VOC "Lifetime Achievement" Prize in 2010 and has greatly influenced the object detection field. Throughout the rest of this paper, the software is called VOC for visual object classifier. The VOC software takes a series of training and validation images and returns a model that can identify objects in new unseen images. The program returns a series of predicted bounding boxes in these images and each box receives a score between -1.0 and 2.0. A score of 2.0 indicates a very strong match in the image to the model. I used the VOC software to train traffic light and stop sign models. The software then created prediction bounding boxes for 14 of the runs in New York City.

The VOC algorithm takes n images and n corresponding XML annotation files that record tight truth bounding boxes on all of the objects in each image. Some of these images must not contain the object so that the software learns to identify false positives. Half of the images train the generated

Pos. Images	Neg. Images	Average Precision
25	25	0.00
50	50	26.69
75	75	31.86
250	10,000	37.93

Table 2: The results of the VOC model when trained with varying numbers of positive and negative images.

models and the other half validate the parameters in the models to find the best possible model. I created truth bounding boxes for the traffic lights in run 10 in New York City to test how well the VOC produced models performed with a variable number of input images. Table 2 shows the results of the VOC models with a varying number of positive and negative input images.

As the number of training images increases, the model produces higher scores for true matches. A true box and a predicted box were considered matched if the area of the intersection over the union was greater than 0.7. The final table entry indicates the approximate number of images used to train the models for this paper. Only scores above 0.0 were considered when creating this table. With only 25 positive and negative images, the model could not predict a single true box with that score. This partly derives from the problem that scores are lower if the model trains on fewer images. With fewer images, the model cannot say with certainty whether or not the image captured an object of interest. The images used to test these models included some traffic lights that were distant and difficult to predict. Although the best model only predicted 37.93 of the traffic lights correctly, every unique traffic light was predicted by at least one box.

2.3. C++ Libraries

I used a series of C++ libraries that enable the creation and manipulation of images, grids, rays, and other three dimensional and two dimensional geometric objects. These files work with the GSV libraries to track projections from 3D space into 2D GSV images as well as other features. I also utilized CSparse, a preexisting linear equation solver that uses sparse matrices and Cholesky decomposition. Timothy A. Davis created the concise sparse matrix package in 2006.[1]

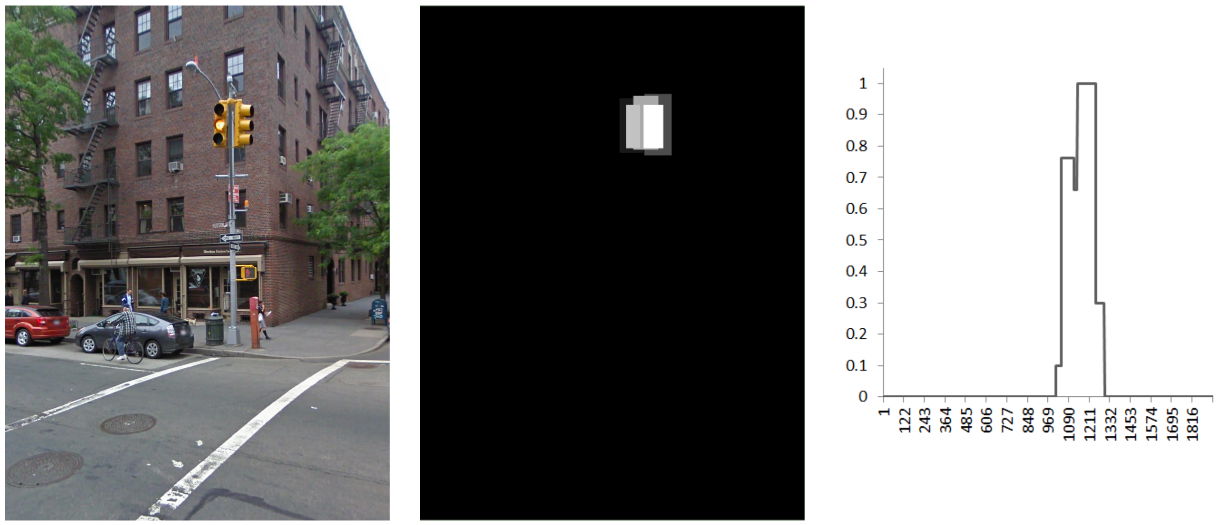


Figure 3: The center probability map corresponds to the predictions made by the VOC detection software when input the image on the left. The function on the right is the horizontal scanline created when projecting the center distribution into one dimension.

3. Creating Probabilities from VOC Predictions

The VOC detection software returns bounding box predictions based on a trained model for a given object. These boxes have corresponding scores in the range of -1.0 to 2.0. Scores close to -1.0 have nearly no probability of belonging to an object and scores above 1.0 have a high probability of belonging to an object. Thus, the procedures discussed in this paper use a high and a low threshold to map these scores to probabilities between 0.0 and 1.0. Any score below the low threshold receives a probability of 0.0 and any score above the high threshold receives a probability of 1.0. All scores between the thresholds are linearly assigned probabilities based on the distance between the low threshold and the score. Figure 3 shows a typical GSV image and a corresponding probability map based on the VOC returned predicted bounding boxes.

4. Projection into Two Dimensions

Since the number of images per run and the area covered per run are so large, all of the discussed algorithms in this paper contract the z-axis. A one dimensional function represents the image probabilities and the three dimensional real space collapses into two dimensions. I created a grid space with a sample every half meter. For the largest considered New York City run, there would

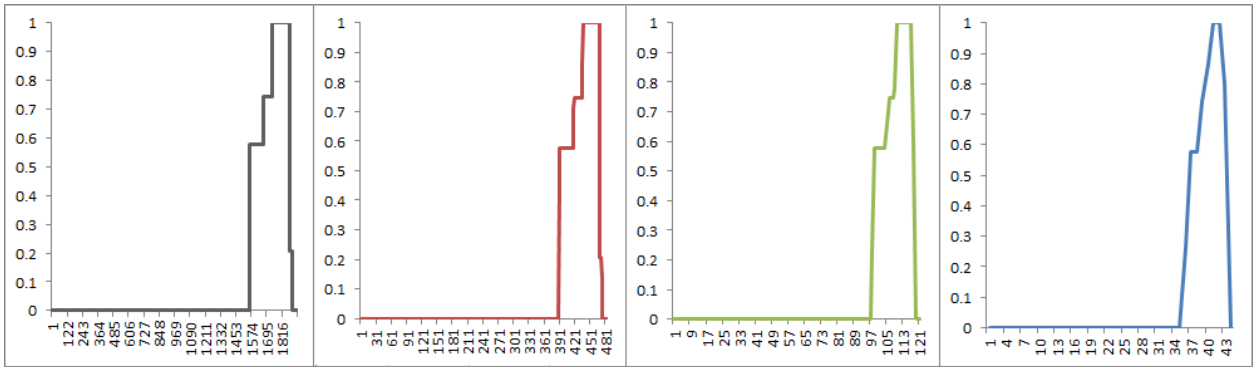


Figure 4: Various probability functions with different sampling frequencies. The sampling frequencies from left to right are: every 1 pixel, every 4 pixels, every 16 pixels, every 44 pixels.

be $511,485,829,120$ pixels and $189,088,350$ grid samples. These numbers produced infeasible memory and time demands. When projecting into two dimensions, this run reduces to $19,863,360$ image samples and $1,254,938$ grid samples, just 0.004% of the number of variables in the three dimensional problem.

The new image samples are determined by taking the maximum along the vertical scanline for every x coordinate in the image. For each image I and corresponding horizontal scanline H, each value of x in the scanline is determined by:

$$H_x = \max_y I_{x,y}$$

Figure 3 shows a probability graph created by the Felzenszwalb code and the corresponding vertical scanline function. Sometimes a scaled down version of the function is used (when there are still too many variables for a given algorithm). In these cases, an averaging filter is applied over the scanline and samples along the function are taken uniformly by a scaling constant. Figure 4 shows the different functions corresponding to different sampling frequencies.

For simplicity, in the rest of this paper the one dimensional samples along the probability function will still be referred to as pixels.

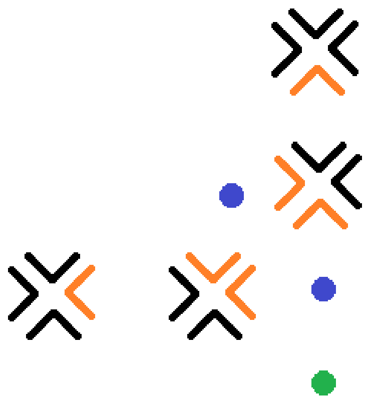


Figure 5: The orange cameras captured the blue objects of interest, and the black cameras did not. The green dot is occluded by the blue dot and is only possibly seen by the two lower cameras.

5. Evaluation Metrics

5.1. VOC Predictions in Three Dimensions

The truth data collected contained three dimensional locations for traffic lights and stop signs in the fourteen New York City runs that were analyzed. I projected these truth data points into all of the images to get an (x, y) image location. By iterating through all of the VOC bounding box predictions in decreasing order of score, I created precision and recall curves for each object for each run. I calculated the average precision from these curves. A bounding box that contained the projected point counted as a true positive, and a bounding box that did not correspond to any projected points indicated a false positive. Projected points that fell outside the bounding box within some margin of error were considered true positives to account for variations in the car's alignment. If a single bounding box contains multiple projected truth points, the closest truth point is considered a match. The one exception to this occurs when multiple truth points are within three quarters of a meter to account for traffic lights that align in such a way that the bounding boxes overlap but both objects are visible. Figure 5 illustrates this. The orange cameras captured the blue objects of interest, but only the lower cameras can possibly see the green dot. Although the green dot would project to the same location as the above blue dot in the upper images, it will not count as seen because it is occluded by the blue dot. In the precision recall curve, the recall measures the number of unique objects the series of images see. A high average precision indicates that most of the highest scores corresponded to real traffic lights and that all the real traffic lights were in fact predicted by a high scoring bounding box. A distant traffic light that does not receive a bounding box in one image does not affect the average precision if another image happens to capture that particular object. Table 3 contains the results for the VOC detection software in three dimensions for unaligned runs. The precision and recall curves for traffic lights are in Appendix A and the precision and recall curves for stop signs are in Appendix B.

5.2. VOC Predictions in Two Dimensions

Since the following algorithms search for objects in a two dimensional plane due to memory and time constraints, another metric tests the VOC prediction boxes in two dimensions. True positives occurred everywhere the projected image x coordinate of a truth point fell in a predicted bounding box's x range. Thus, the y and z values of the truth points and the y range of the bounding boxes do not influence the results. As above, the recall measures the number of unique objects that the set of images captured. Table 3 contains the results for the VOC detection software in two dimensions for unaligned runs. The precision and recall curves for traffic lights are in Appendix A and the precision and recall curves for stop signs are in Appendix B.

Run	Traffic Light (3D)	Stop Sign (3D)	Traffic Light (2D)	Stop Sign (2D)
0	54.22	76.45	69.31	88.79
1	35.39	4.92	51.33	41.82
2	27.43	43.20	32.76	59.82
3	32.76	19.57	65.80	76.80
4	34.36	16.67	41.54	49.20
6	59.98	100.00	89.10	100.00
7	26.32	13.94	47.63	83.33
9	48.15	83.33	68.24	100.00
10	89.29	–	100.00	–
11	65.53	–	70.82	–
14	65.30	43.19	79.86	47.08
15	30.14	–	57.20	–
17	30.66	–	79.48	–
19	42.32	34.01	88.35	83.04

Table 3: The average precision for the VOC detection software on 14 of the New York City runs. The evaluations on the left used the three dimensional metric and the evaluations on the right used the two dimensional metric.

5.3. Probability Grid Evaluation

I projected into two dimensions all of the three dimensional locations of traffic lights and stop signs collected from the 14 New York City runs. This created a series of truth grids with grid samples every 0.5 meters. The algorithms described in this paper produce probability grids with samples every 0.5 meters as well. A function takes in a probability grid and a truth grid. All of the grid samples are ordered by value from high to low. If a grid prediction is within 1.5 meters of a true location, it is considered a positive hit. The value 1.5 was chosen to account for misalignments in the car's path that compounded inaccurate results when multiple misaligned images agreed on a deviated location. The metric produced a precision versus recall curve and the average precision was calculated for each run. Appendices A and B have the precision and recall curves for traffic lights and stop signs respectively.

6. Hough Voting

6.1. Theory

I implemented a Hough voting procedure to create high probability regions for certain objects in the real world. Each pixel of every image has a weight corresponding to the probability that that pixel captured part of an object of interest. Consider a pixel p with probability weight w in an image with a camera viewpoint V. The ray from V through p is labeled R. The ray R starts at V and travels through the grid, intersecting multiple grid samples. Every grid sample that the ray intersects receives a vote of weight w from p. By iterating over all of the pixels in all of the images, grid samples accumulate weighted votes giving some value to that location. Locations corresponding to grid samples with a large value are more likely to contain the desired object.

Unfortunately, following the above procedure will produce a natural bias towards locations where camera viewpoints are located. Figure 6 illustrates this problem. Consider two cross sections of the field of view that are perpendicular to the camera's direction (drawn in blue). Cross section C is closer to the viewpoint than cross section C'. The distance from C to V is h and the distance from C' to V is h'. The length of C is b and the length of the C' is b'. The sum of the weights of the pixels in this image is W. The weight of votes distributed over the length of all perpendicular cross sections is W. The density of the weight of the votes over the cross section is the total weight of all the pixels in the image divided by the length of the cross section. This density should remain constant regardless of the distance from the viewpoint to a given cross section since the pixels provide no additional information on the distance to the object it captured. δ is the density along C and δ' is the density along C'.

$$\delta = \frac{W}{b}$$

$$\delta' = \frac{W}{b'}$$

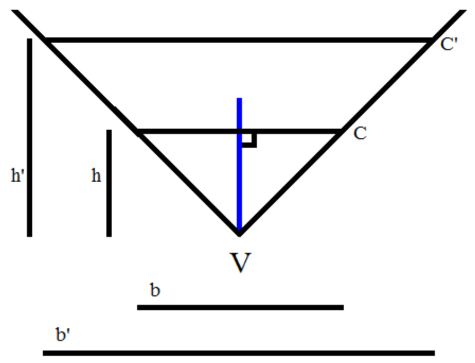


Figure 6: The density along the cross section C is greater than the density along cross section C'. This causes a bias for locations that are closer to the viewpoint.

δ should equal δ' but here they differ by some constant.

$$\delta = \alpha\delta' \rightarrow \frac{W}{b} = \alpha\frac{W}{b'} \rightarrow \frac{1}{b} = \alpha\frac{1}{b'}$$

Since the triangles are similar, $\frac{b'}{b} = \frac{h'}{h}$. Thus, the constant $\alpha = \frac{h'}{h}$. By choosing the distance $h = 1$ as the ideal density to match, the bias can be eliminated by increasing the weights of pixel votes along the ray R proportionately with the distance h'.

6.2. Modifications

6.2.1. Global Maxima Voting

Figure 7 illustrates one of the problems of implementing the above Hough voting algorithm. Consider the street intersection at the bottom of the images that has two traffic lights. When the Hough voting procedure concludes, there are certain regions that receive a high number of votes only because of their location. Every forward facing image before reaching the intersection should see two traffic lights and vote for both. Since the path is relatively straight, there are many nearly parallel rays voting through the intersection. When the car turns at the intersection, the backwards facing images should also see both traffic lights. All of these images contain pixels that produce rays that vote through the intersection, but these rays are nearly perpendicular to the previous set of rays. Call the traffic lights t_1 and t_2 and the path p_1 before the intersection and p_2 after the intersection. The pixels along p_1 that align with t_1 and pixels along p_2 that align with t_1 corroborate the location of t_1 when the rays intersect at t_1's location. However, since the rays extend through the actual position, the rays from the pixels along p_1 voting for t_1 will also intersect the rays from the pixels along p_2 voting for t_2. This creates a phantom traffic light that only receives votes because of its relation to t_1 and t_2.

In addition to this problem, there are residual votes for locations behind an object in relation to the group of rays along a straight path. Each of these rays votes for all of the locations that it passes, and there are certain regions in the grid that receive high votes before the rays that see the same object diverge. The blue circle in Figure 7 shows an example of this behavior. Since the rays might diverge slowly, some locations distant from the true object will receive a high number of votes. If many images see one object and not many images see another, the residual votes for the more widely recognized object can overpower the votes for the lesser seen true object.

To reduce these two effects, a secondary round of Hough voting occurs. The algorithm traverses along every ray from every pixel and determines the location along the ray that is a global maxima. That pixel then gets one vote along the entire ray and votes for the global maxima. The weight of this vote solely equals the probability that the pixels captured the object since the density along the cross section in the secondary round of voting is no longer a concern. This reduces the residual voting effects since the rays begin to diverge immediately after the intersection so the phantom locations have fewer total votes in the preliminary voting than the true locations. The image on the right of Figure 7 shows the result of voting only for the global maxima in run 19.

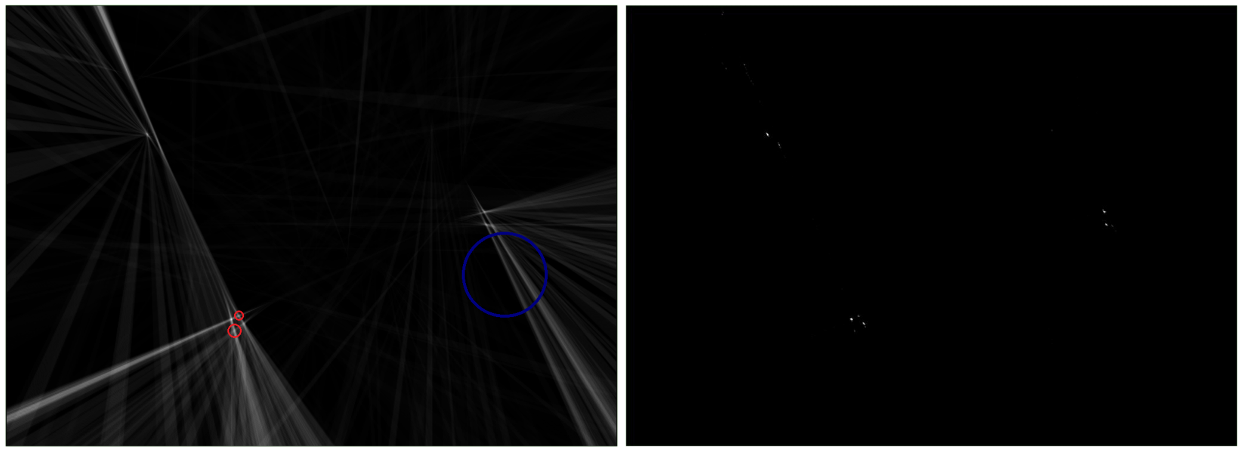


Figure 7: The left image shows the results of running the Hough voting algorithm described in the theory section. The red circles indicate regions that received a high number of votes based on their location relative to the actual objects. The image on the right shows the predictions when a pixel only votes for the global maximum along its ray.

6.2.2. Restricted Range Voting

In reality, there are certain object locations that a pixel cannot capture. An image cannot capture a stop sign if it is so distant that the bounding box would only be a few pixels. The probability of occlusion also increases as a ray traverses further from the viewpoint. In long runs where the grid space is large, every pixel should not vote for every location along its ray in practice. Most of these locations are so distant that even if a traffic light or stop sign occurred there it would not be visible in the image. This really becomes a problem when instituting the global maxima secondary round as discussed in the previous section.

Consider a straight path that a Google Street View car might take over the course of several blocks. At the end of this path, the car turns down a side street. Street intersection objects such as traffic lights and stop signs are only useful if they are easily visible and recognizable to humans. Thus, most of the objects are generally placed in the same location at every intersection. For example, there is almost always a traffic light hanging over the middle of the intersection. The car path mentioned above might travel under ten traffic lights before turning down a side street. These traffic lights are labeled $t_1, ..., t_{10}$, with t_1 corresponding to the first traffic light crossed and so forth. In the preliminary round of voting, the pixels that capture t_1 vote for all of the locations along their rays. However, there may be several traffic lights along this ray that are receiving votes for these pixels

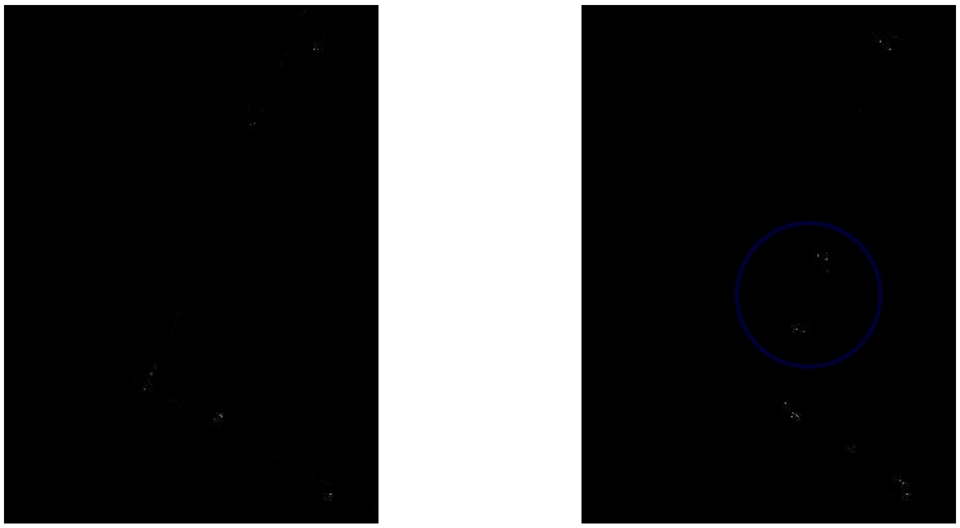


Figure 8: The image on the left shows the results when every pixel votes for every location along its ray. The image on the right restricts the locations a pixel can vote for based on distance.

that really only captured t_1. Since the votes increase proportionately with distance, t_{10} receives many highly weighted votes from pixels that simply could not have seen it. Figure 8 illustrates this problem for a portion of run 1. When the secondary round of voting occurs the pixels that truly captured t_1 may cast their vote for t_{10} simply because it received a disproportionately high number of votes based on its location at the end of a long straight run. In order to compensate for this error, in both rounds of voting, pixels only vote for locations within a certain distance of the viewpoint of the camera. Figure 8 shows the results when imposing a voting distance restriction on run 1. There is no closed form mathematical solution to the optimal maximum distance for which pixels should vote. Rather, I used training and validation runs to determine the optimal distance for traffic lights and stop signs.

6.3. Training and Validation

There are three parameters that can vary for each object. Since the VOC software returns bounding boxes with scores, the voting procedure needs to determine how to calculate a probability based on the VOC predictions. I employed a linear conversion from score to probability with a low threshold, θ_l where everything below was 0.0. There was also a high threshold, θ_h, and any score constituted a probability of 1.0. The maximum distance in meters, d, which a pixel could vote for also influenced the results of the Hough voting procedure. I tested all combinations from the sets:

$$\theta_l = \{-0.5, 0.0, 0.5\}$$

$$\theta_h = \{1.0, 1.5\}$$

$$d = \{\text{none}, 10, 12.5, 15, 17.5, 20, 22.5, 25, 27.5, 30, 32.5, 35, 37.5, 40\}$$

I found the best combination for each run in terms of average precision. The best combination for the object was determined by finding the combination whose average precision deviated least from the best combinations for each individual run. For traffic lights, $\theta_l = -0.5, \theta_h = 1.0, d = 25$. For stop signs, $\theta_l = -0.5, \theta_h = 1.0, d = 17.5$. Table 4 shows the results of the algorithm on the training and validation data. Appendices A and B contain the precision and recall curves for all of these runs. Detecting stop signs in run 1 appeared to be an outlier, since this combination of θ_l, θ_h, and d produced the best results for the other training and validation data.

Object	NYC 1	NYC 3	NYC 9	NYC 10	NYC 14	NYC 17	NYC 19
Traffic Light AP	64.20	70.11	87.79	100.00	89.71	94.81	100.00
Traffic Light Recall	96.47	80.00	92.31	100.00	100.00	100.00	100.00
Stop Sign AP	2.25	95.36	100.00	–	87.12	–	87.50
Stop Sign Recall	66.67	100.00	100.00	–	100.00	–	100.00

Table 4: The results on the training and validation data.

6.4. Testing and Results

Runs 0, 2, 4, 6, 7, 11, 15 were used to test the Hough voting procedure discussed above using the parameters determined by the training and validation data. The results of these runs are in Table 5. Appendices A and B have all of the precision and recall curves for all of these runs.

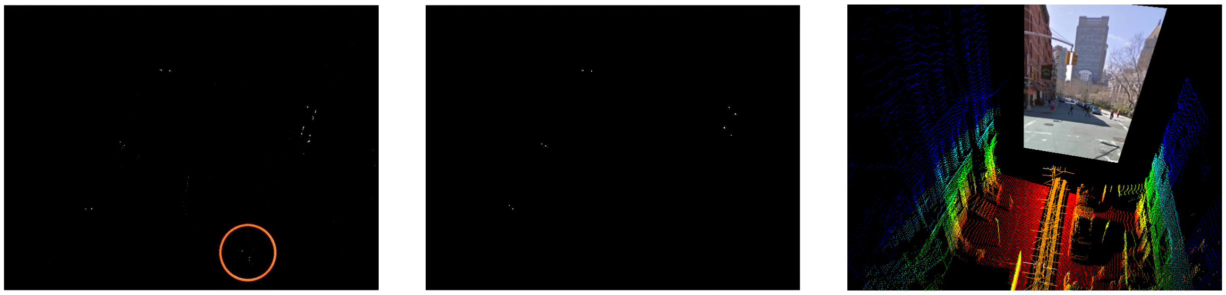


Figure 9: The three predicted traffic lights in the orange circle cannot appear in the truth data because the LIDAR scan did not capture that segment of the city.

Object	NYC 0	NYC 2	NYC 4	NYC 6	NYC 7	NYC 11	NYC 15
Traffic Light AP	86.78	54.89	66.26	60.20	68.41	85.21	79.77
Traffic Light Recall	98.30	97.56	100.00	93.75	100.00	100.00	93.55
Stop Sign AP	84.51	36.26	90.28	100.00	45.00	–	–
Stop Sign Recall	91.67	95.00	100.00	100.00	100.00	–	–

Table 5: The results on the data reserved for testing.

For the traffic lights, all of the runs except for NYC 6 performed better than the standard VOC object detection software in both three dimensions and two dimensions. However, NYC 6 presents an interesting scenario, as shown in Figure 9. The voting procedure described above predicts three traffic lights in the orange region. The corresponding points are not in the truth data. However, those traffic lights really do exist but the LIDAR does not extend far enough and thus no truth points could be created for those traffic lights. This represents a wider problem when analyzing street intersections using Google Street View data. Namely, most of the runs begin as a car leaves a street intersection. Since two LIDAR scanners perpendicular to the car's direction collect the LIDAR data, the LIDAR does not capture the objects in the intersection. However, since the images produce a panorama around the car, the objects in the intersection are seen on camera.

For the stop signs, NYC 2 and NYC 7 performed worse than the 2D and 3D evaluation metrics. In particular, NYC 7 failed when the car passed through a construction zone with scaffolding on both sides. If the VOC object detection model recognized the scaffolding as the shape of a stop sign, it might predict a weak scored bounding box. In this section of New York, it appears that enough of these weak scores corroborated what they saw to make a strong prediction of something that was

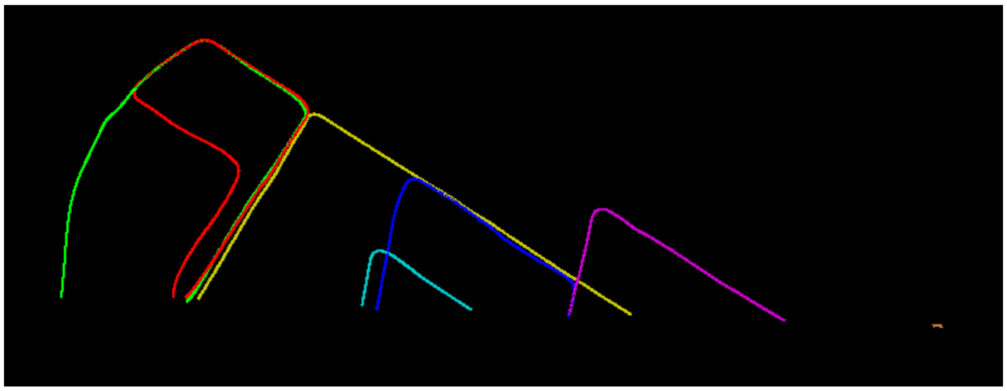


Figure 10: The various segments that make up run 2 in New York City. Most of the stop signs occur where the green, red, and yellow paths follow the same street.

simply misrepresented.

Run 2 contains several segments corresponding to different times the GSV car entered certain New York City streets. The above results used the unaligned GSV data. However, when using aligned GSV data, run 2 performs better with an average precision of 43.79 and a recall of 100.00. Figure 10 shows run 2 of New York City. The stop signs are nearly all located where the green, red, and yellow paths intersect. When the paths are not aligned, the pixels in the images vote for slightly different areas diluting the vote totals. Figure 11 shows a side by side comparison of one part of run 2. There is one stop sign in this location. The pixels from the images in the aligned segments vote for the same location rather uniformly creating one grid location with a very high probability of containing the object. The unaligned images scatter their votes creating four weaker locations that do not correspond to an actual true location.

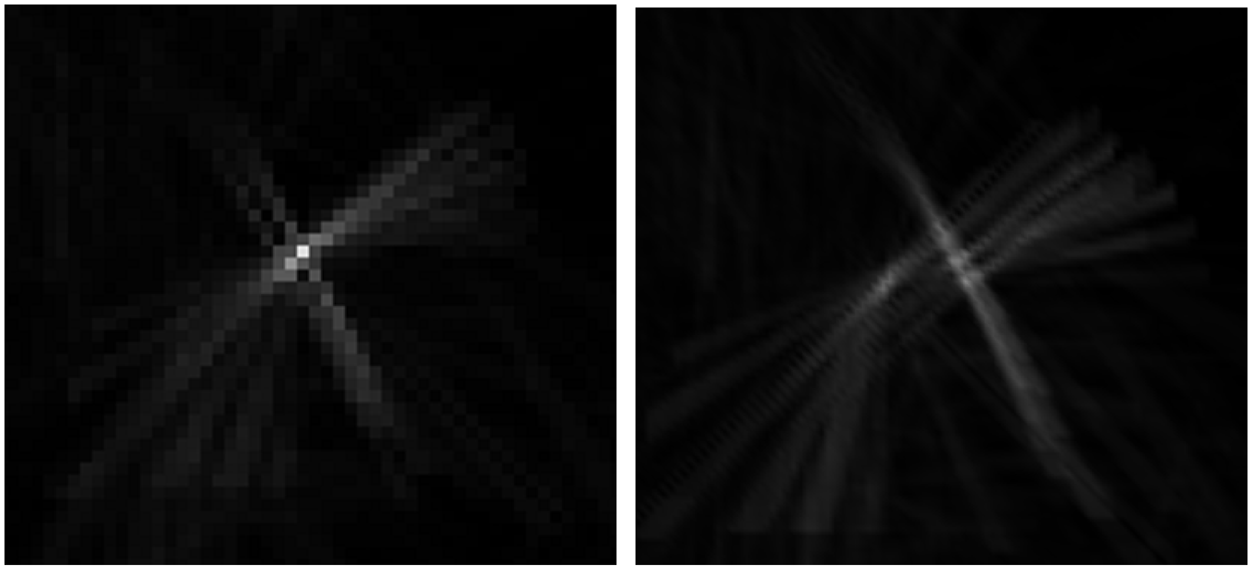


Figure 11: This part of New York City contains one stop sign. The image on the left shows the preliminary Hough vote when the segments are aligned. The image on the right shows the preliminary Hough vote when the segments are not aligned. When the segments are not aligned the votes scatter leading to a few locations with only medium expected probabilities.

7. Linear Equations

7.1. Theory

The next program defines a system of linear equations to model a city. $p_{i,j}$ represents the probability that the j^{th} pixel in image i corresponds to part of an object of interest in the image i. $g_{x,y}$ represents the probability that the location (x,y) contains an object of interest. For each image i, for each pixel j, there is a variable $p_{i,j}$. Every pixel (i,j) maps to a set of grid samples. These grid samples are determined by taking a ray from the viewpoint through the pixel and seeing which locations the ray intersects. The sum of the probabilities of the samples (call the set S) that map to a pixel (i,j) must equal $p_{i,j}$, since $p_{i,j}$ gives a prediction for how likely an object is seen by that pixel, and the samples correspond to the real locations that (i,j) could have possibly captured. There are also smoothing equations that make sure that adjacent pixels have the same probability as well as adjacent grid cells. This leads to the following set of equations:

For all pixels (i, j) with voxels $(x, y) \in S$ that map to (i, j):

$$\sum_{(x,y)\in S} g_{x,y} - p_{i,j} = 0$$

For all pixels (i, j):

$$\lambda \left(p_{i,j} - p_{i,j-1}\right) = 0$$

For all voxels (x, y):

$$\gamma \left(g_{x,y} - g_{x-1,y}\right) = 0$$

$$\gamma \left(g_{x,y} - g_{x,y-1}\right) = 0$$

The previous equations create a system of linear equations that define the problem of finding objects of interest using the results of the Felzenszwalb object detection algorithm. The algorithm finds the least squares solution using the CSparse library[1].

7.2. Modifications

7.2.1. Coarse-To-Fine Algorithm

Even after projecting the problem from three dimensions into two dimensions, there are significant memory constraints when using the CSparse linear equation solver. Consider, run 3, one of the smaller New York City runs. The cross-sectional area is $85,056.49m^2$, which leads to $340,228$ grid variables. There are $1,287$ images, and sampling at every pixel leads to $2,491,632$ pixel variables. This results in roughly $8,835,808$ equations. Assume there are n variables and m equations. The linear equation solver takes an n by m sparse matrix which in this run could easily contain tens of millions of nonzero elements. In order to reduce the number of pixel variables, I created many different functions with different sampling frequencies, as shown in Figure 4.

To even further reduce the memory problems, I implemented a coarse-to-fine algorithm. The coarse-to-fine algorithm had two levels, the first with grid samples every other meter, and the second with grid samples every half meter. The same set of equations explained above produced a linear

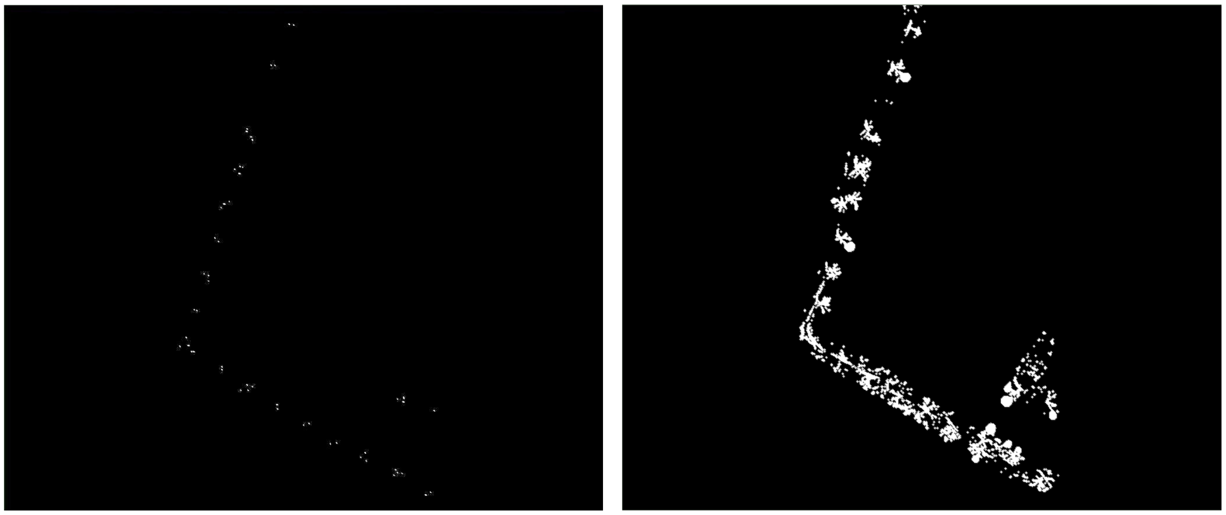


Figure 12: The image on the left shows the truth data for run 1. The image on the right is the result of the first level of the coarse-to-fine algorithm. Only one true traffic light's location from this level did not proceed to the next level.

system that the CSparse library solved[1]. The highest scoring 1.625% of grid samples "moved" on to the next level where the grid sampling frequency increased. The other 98.375% of grid samples were not considered at the higher sampling frequency.

Recall is the only metric that matters have the first level of the coarse-to-fine algorithm completes. If a traffic light is missed during the first-level and the corresponding grid location is not promoted to the second level, the second level cannot predict that location. Figure 12 shows the results of the first level of the coarse to fine algorithm when detecting traffic lights in run 1. This run had near perfect recall and only missed two traffic lights out of eighty five after the first level of the coarse to fine algorithm.

Going to the second level of the algorithm, the grid sampling frequency increases as the number of grid locations considered decreased so that the total number of variables remained nearly constant between the two levels. Figure 13 shows the results of the second level of the coarse to fine algorithm when detecting traffic lights in run 1.

7.2.2. Restricted Range Voting and Local Maxima Voting

As discussed with the Hough voting algorithm, there are many object locations that simply cannot correspond to certain pixels based on the distance between the location and the viewpoint of the

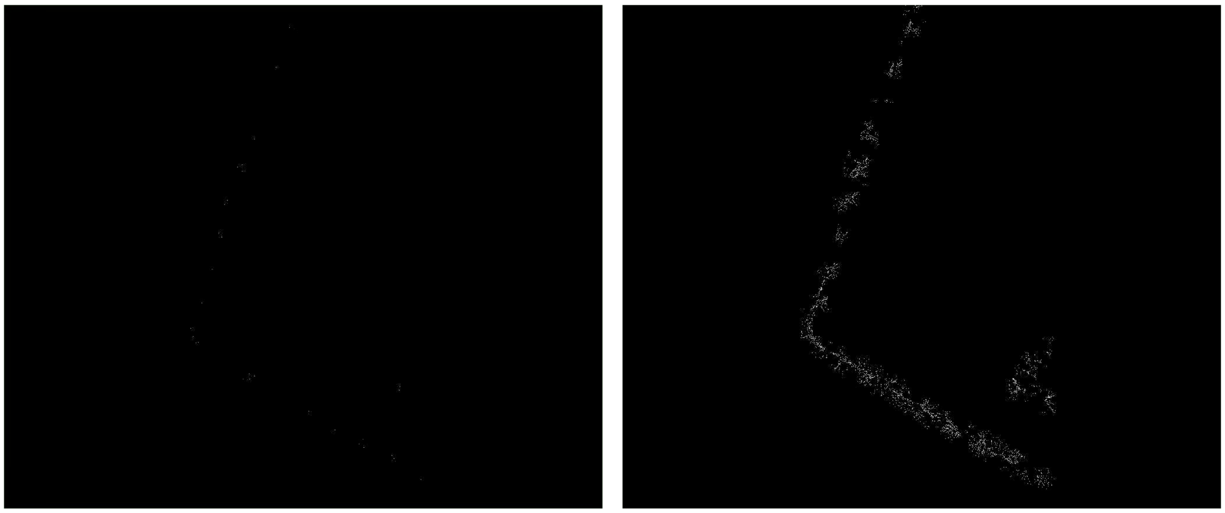


Figure 13: The image on the left shows the truth data for run 1. The image on the right is the result of the second level of the coarse to fine algorithm.

camera. When constructing the set of linear equations, it was important that the data term between the grid variables and pixels only included viable pairs. Otherwise, some of the weight of a pixel's probability would be spread to locations that the object certainly could not belong to. At least in Hough voting every possible location receives some vote. However, in the system of linear equations, a distant location could dominate the data equation taking most of the pixel's probability weight.

One side effect of the grid smoothing term is that many locations around the object receive some non trivial probability for containing an object. This can easily spread over several meters. To reduce this effect, I only consider probabilities that are local maxima and valued higher than all of their surrounding neighbors. Since there may easily be six to eight traffic lights or stop signs in an intersection, every location in the intersection could easily receive a high probability because of the number of objects around it. By only considering the local maxima, these other street intersection locations do not negatively impact the results.

7.3. Training and Validation

There are three variable parameters for this algorithm that I considered, λ, γ, and the maximum distance in meters, d, allowed between pixel and grid sample pairings. Pixel sampling frequency

Object	NYC 1	NYC 3	NYC 9	NYC 10	NYC 14	NYC 17	NYC 19
Coarse Recall	91.52	82.92	100.00	100.00	98.15	100.00	100.00
Traffic Light AP	53.20	53.20	78.50	86.11	80.02	89.09	98.15
Traffic Light Recall	97.65	100.00	92.31	100.00	98.15	100.00	100.00
Coarse Recall	33.33	88.89	100.00	–	100.00	–	100.00
Stop Sign AP	5.08	68.25	100.00	–	55.49	–	59.44
Stop Sign Recall	66.66	88.89	100.00	–	100.00	–	100.00

Table 6: The results of this algorithm on the training and validation data.

was not considered a variable but rather the algorithm always choose the highest sampling frequency within the memory constraints of the linear equation solver. For most of the runs, this resulted in a sampling frequency of 8. The variable parameters were chosen by running the program on the training and validation data with varying values for each. These parameters varied for both levels of the coarse-to-fine program since the first level focused on recall and the second level precision. $\lambda = 0.25$, $\gamma = 4$, $d = 24$ when detecting traffic lights at the coarser level and $\lambda = 0.25$, $\gamma = 8$, $d = 25$ when detecting traffic lights at the finer level. For stop signs, these parameters were $\lambda = 0.25$, $\gamma = 4$, $d = 12$ for the coarser level and $\lambda = 0.25$, $\gamma = 8$, $d = 19$ for the finer level.

Table 6 shows the results of the training and validation data for both objects. Occasionally the recall increases from the coarser level to the finer level. This is an artifact of how recall is calculated in the grid space for both grid sizes. When the grid samples are spaced every two meters, the predicted location has to exactly match the true location. When the grid samples are spaced every half meter, the predicted location can deviate slightly from the true position. When the recall increases, the true location of the object was only one sample away from a grid location that passed to the next level. Since there is a buffer region when determining recall at the finer level, that grid location can now count as a true positive.

7.4. Testing and Results

Table 7 shows the results of this algorithm on the testing data. This algorithm generally performs well for the first number of objects that it detects. However, as the recall increases, the precision drops off quite quickly, leading to a low average precision. Figure 14 illustrates this. The system of

Object	NYC 0	NYC 2	NYC 4	NYC 6	NYC 7	NYC 11	NYC 15
Coarse Recall	91.52	82.92	100.00	100.00	84.20	98.51	83.87
Traffic Light AP	76.33	35.35	51.93	62.20	58.00	64.28	50.33
Traffic Light Recall	97.65	100.00	92.31	100.00	98.15	100.00	100.00
Coarse Recall	83.33	80.95	87.5	100.00	100.00	–	–
Stop Sign AP	84.72	51.08	53.27	100.00	75.00	–	–
Stop Sign Recall	91.67	90.48	100.00	100.00	100.00	–	–

Table 7: The results of this algorithm on the testing data.

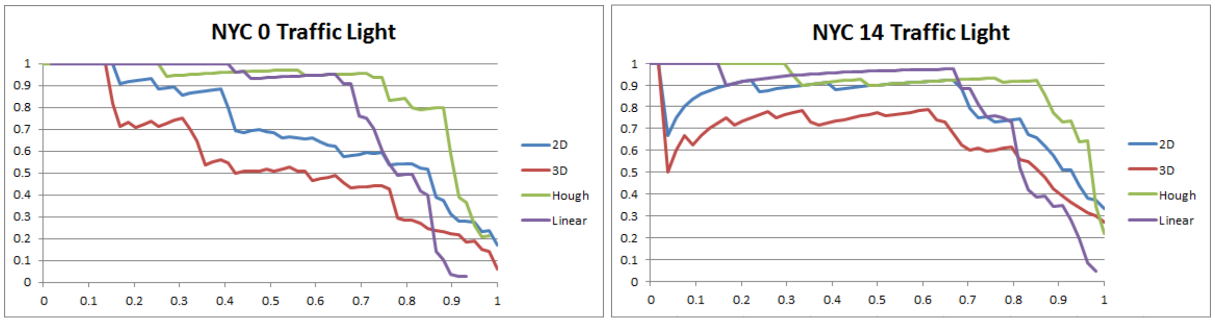


Figure 14: The precision for the linear algorithm remains high for most of the objects detected, but drops off quickly towards the end.

linear equations performs well, and even better than the other algorithms for most of the objects detected early on. However, the precision quickly falls leading to a lower average precision.

This system of linear equations had average precisions near the two dimensional object detection metrics for almost all of the runs. It performed better than the three dimensional object detection metric on all of the runs besides 10 and 14 when detecting traffic lights. With the Hough voting algorithm, the objects seemed to actually have optimal parameters. There was a lot more variation per run when tweaking the parameters on the training and validation data. Thus, every individual run had its own optimal λ, γ, and d but these were not necessarily equal across runs. Thus, the training data might not have actually helped choose the values of the variable parameters.

As mentioned above, run 2 has many segments that are not properly aligned in the GSV data this algorithm first tested on. Therefore, I also ran this program on an aligned version of run 2 in New York City. This program produced a probability estimate with an average precision of 60.50 with 90.00% recall. The average precision increased by nearly 10 points by using an aligned GSV path. Figure 15 shows the precision and recall curves when using the aligned and unaligned Google

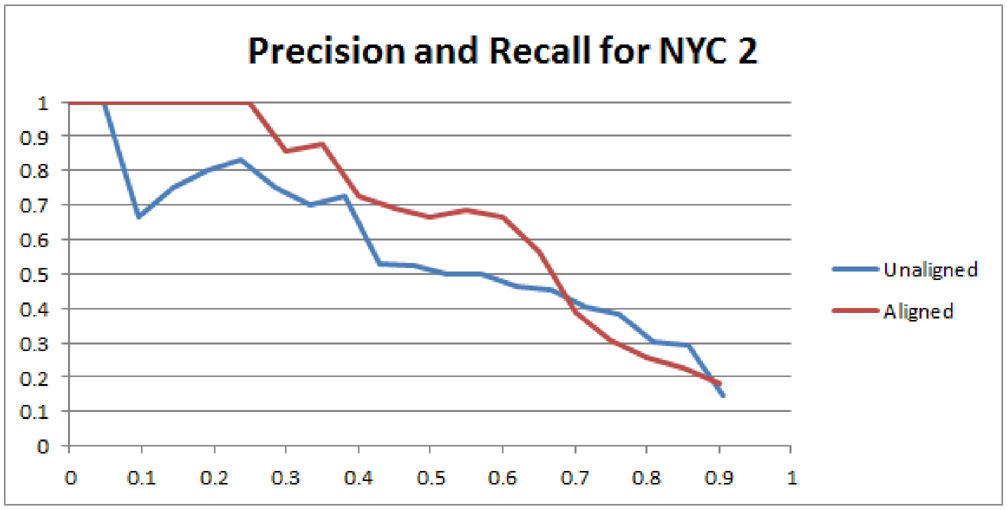


Figure 15: The average precision for the aligned run is nearly 10 points higher than the average precision for the unaligned run.

Street View data for stop signs in run 2.

The Hough voting procedure described above performed better than the system of linear equations on all runs except for detecting traffic lights in run 6 and stop signs in runs 0 and 2. I believe using unaligned data greatly impacted the effectiveness of the linear equation program. A slight misalignment in the path will cause the pixels to align with the completely wrong grid locations leading to a system of equations that does not truly define the problem. With Hough voting, pixels are providing strong support for all of the locations that they align with. Ideally, in the system of linear equations one grid sample will have probability 1.0 and all of the grid samples along the ray other than it will have a probability of 0.0. Thus, the pixel's "votes" should strongly support one grid location and provide near no support for the other grid locations. (Although the pixel is not technically voting, it is still providing some probability to the grid samples through the linear equation's data terms.) A slight misalignment in the calculation of the path could skew all of the probabilities in the grid. In the aligned run 2 data, the system of linear equations performs better than the Hough voting program. Although this does not confirm that the system of linear equations could perform better with aligned data, it encourages further investigation.

8. Future Research

There are several additions I would like to implement in the future. First, I would like to collect truth data for multiple different objects in the LIDAR data and train more VOC models. Eventually I would like to have models for one way signs, pedestrian crossing signs, street lamps, trash cans, walk signal boxes, among others. I could then run the above algorithms on these new objects to see on what range of objects the algorithms will work. I did not collect any truth data for the largest four New York City runs because of time constraints. I would like to eventually collect that data and see how the algorithms scale. On a similar note, many of these runs overlap, and as of now I am not utilizing that information. Eventually I would like to look at all of New York City as a continuous unit and not differentiate between runs. That way there is even more information to further refine the Hough voting procedure and system of linear equations.

Eventually I would like to further test how these algorithms perform with aligned data. As of now, the data is not perfectly aligned and thus small errors in the car's path can greatly affect the algorithms since they rely on traversing the ray from the viewpoint of a camera through the pixel. If the viewpoint is skewed, the ray can miss entirely certain object locations. After getting truth data for all of the aligned runs, I could then compare the results on the aligned and unaligned data to determine which procedures are the most robust to errors in the car's path.

Lastly, with a three dimensional object detection software, I could add in additional information to the linear equation solver. By first iterating over the LIDAR data with a three dimensional object model, I could determine beforehand the probabilities that a location contains an object. Then, I could link the pixels in images to the LIDAR locations and merge the data from the two dimensional object recognition and three dimensional object recognition to get a clearer picture of the scene.

9. Conclusion

By combining Google Street View data and preexisting object detection software, it is possible to locate the real positions of objects at a street intersection. The current object detection software does a good job at detection objects in images but cannot determine if two images are seeing the same object. By using GSV data, which includes camera viewpoints, it is possible to determine if two images captured the same unique object or two separate objects. Furthermore, the actual locations of these objects can be determined.

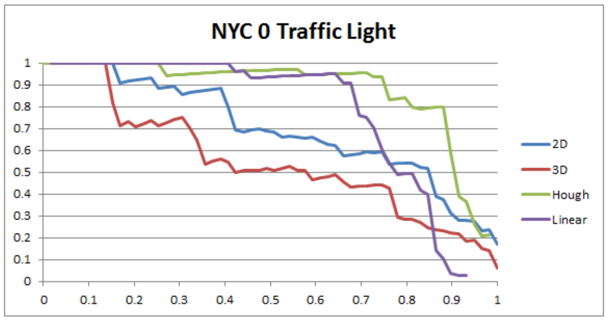
The first program that I created to find the actual location of objects in a street intersection relied on Hough voting techniques. Every pixel of every image cast a vote through the world to indicate possible places an object could be if the pixel did in fact capture it. By accumulating votes, certain locations return a high probability of containing an object. After modifying some of the theoretical Hough voting procedures to simulate the real world, the Hough voting program performed better than the traditional object detection software.

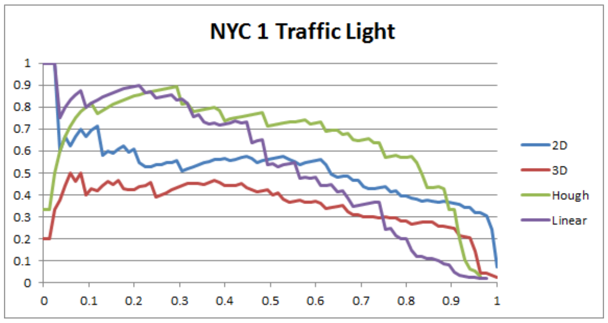
The second program generated and solved a large system of linear equations that equated the probabilities that pixels captured an object with the probabilities that a location in a grid corresponded with an object. I reduced the time and memory constraints by implementing a two step algorithm that performed a coarse run of the grid space and then a finer pass afterwards. This algorithm performed on par with the preexisting object detection software, although aligning the path may greatly improve its performance.

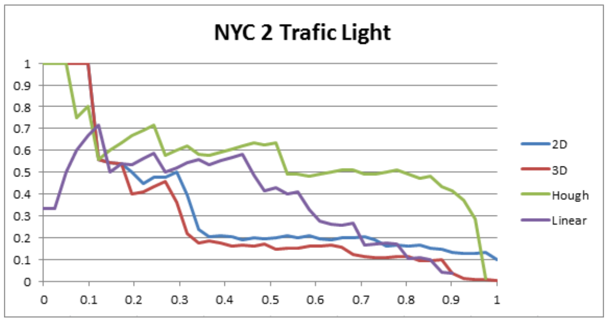
There are many opportunities for future research stemming from this paper. The alignment of the car path may have a nontrivial impact on the performance of these algorithms as discussed in previous sections. In addition to this, there are many other objects at street intersections that are worth testing the algorithms on. Testing these algorithms with multiple different objects is the only way to fully understand how well they can perform. Lastly, with three dimensional object recognition, there is even more information the programs can use to determine the real locations of certain objects.

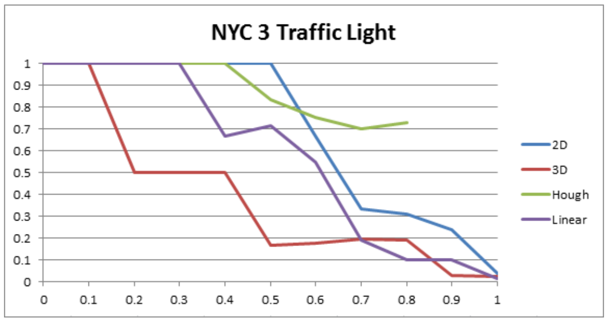
This paper represents my own work in accordance with University regulations. – Brian Matejek

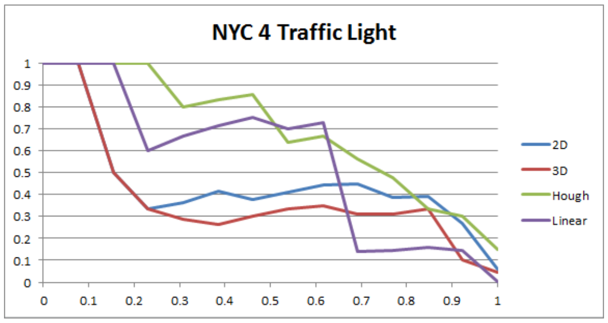
10. Appendix A: Traffic Light Precision and Recall Curves

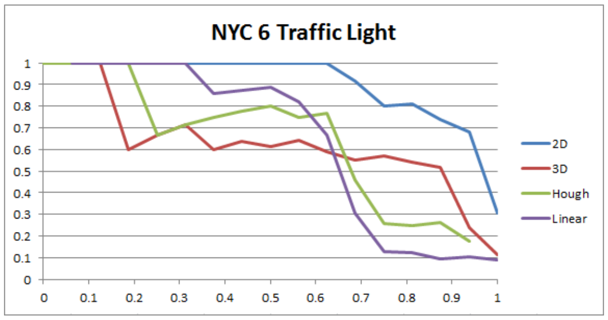


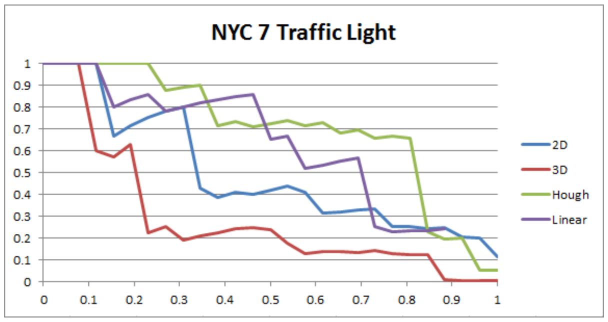


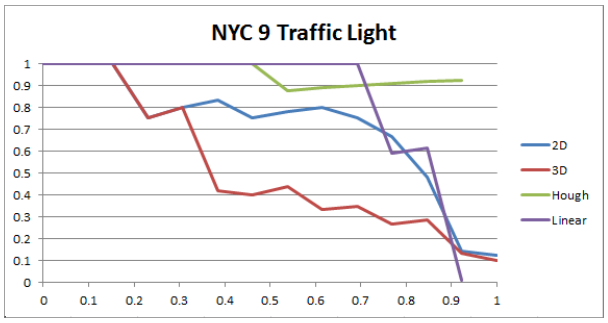


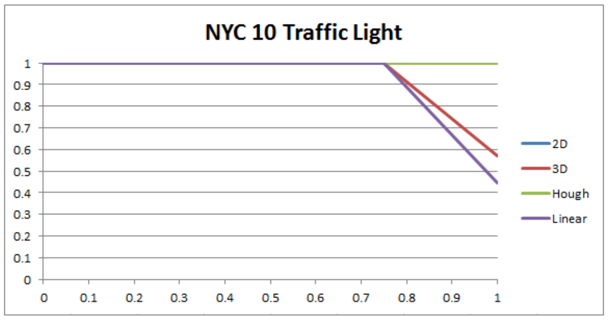


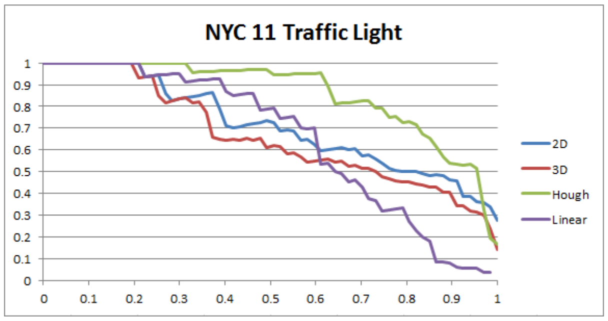


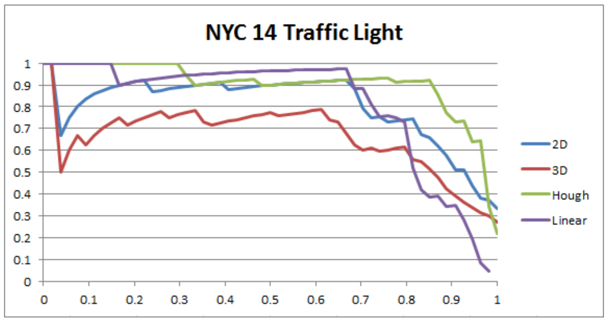


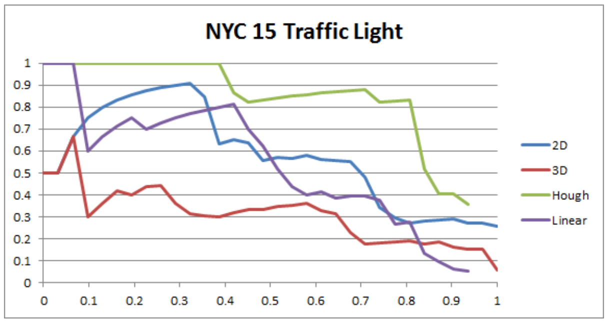


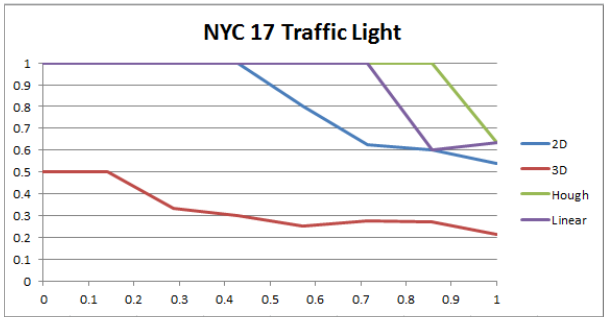


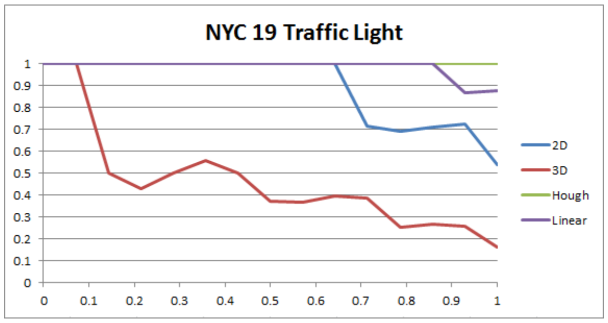

NYC 10 Traffic Light
2D
3D
Hough
Linear


NYC 11 Traffic Light
2D
3D
Hough
Linear

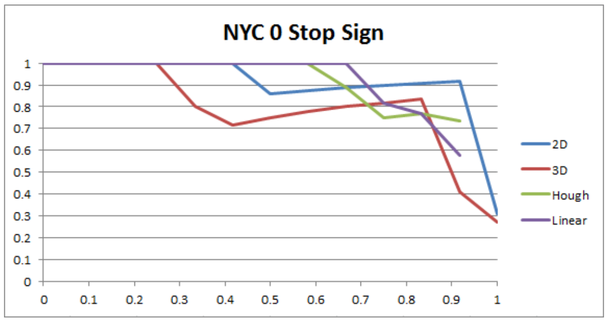

NYC 14 Traffic Light
2D
3D
Hough
Linear

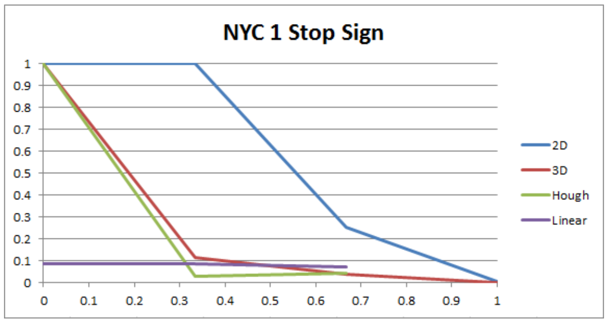

NYC 15 Traffic Light
2D
3D
Hough
Linear

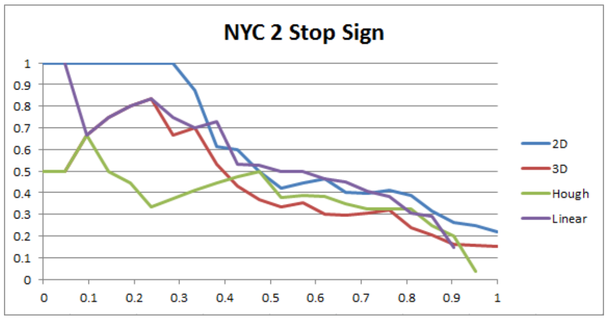

NYC 17 Traffic Light
2D
3D
Hough
Linear

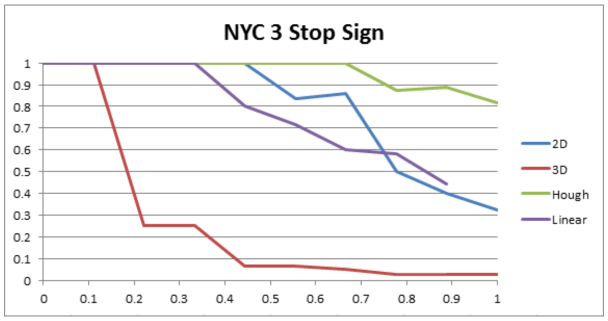

NYC 19 Traffic Light
2D
3D
Hough
Linear

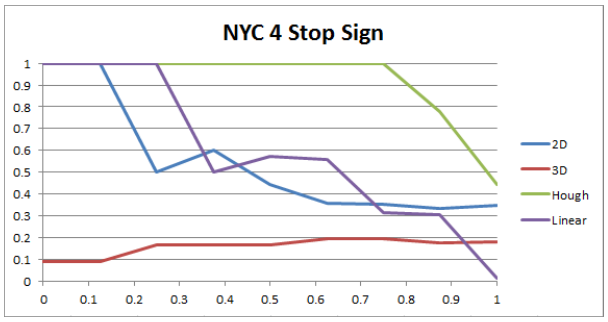
11. Appendix B: Stop Sign Precision and Recall Curves

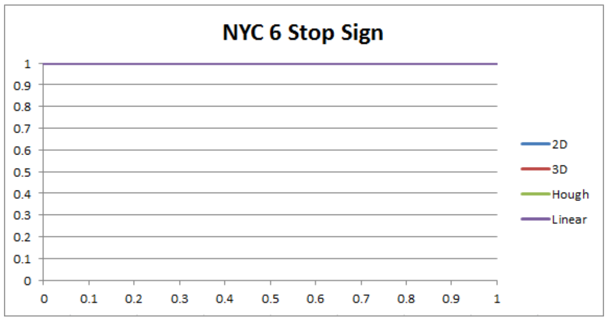


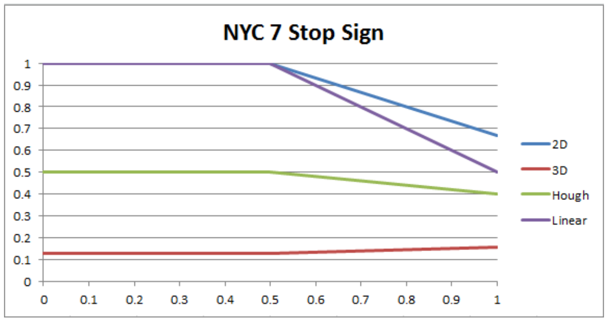


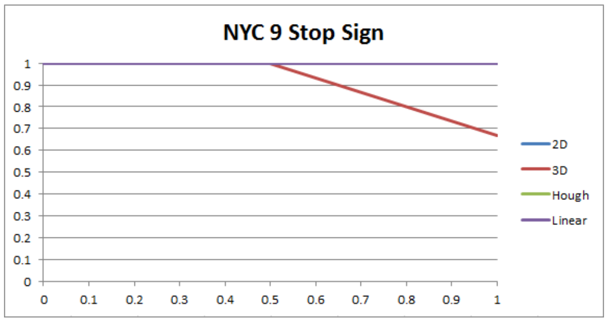


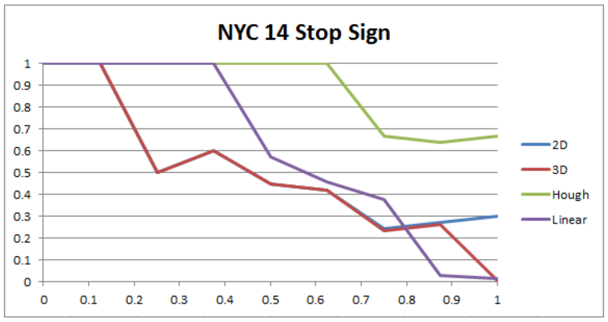


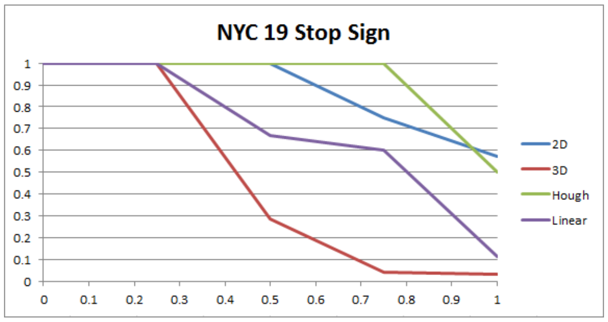









NYC 14 Stop Sign
2D
3D
Hough
Linear


NYC 19 Stop Sign
2D
3D
Hough
Linear

References

[1] T. A. Davis, *Direct Methods for Sparse Linear Systems*, ser. SIAM Book Series on the Fundamentals of Algorithms. SIAM, September 2006.

[2] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[3] R. Girshick, "From rigid templates to grammars: Object detection with structured models," April 2012.

[4] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Discriminatively trained deformable part models, release 5," http://people.cs.uchicago.edu/ rbg/latent-release5/.