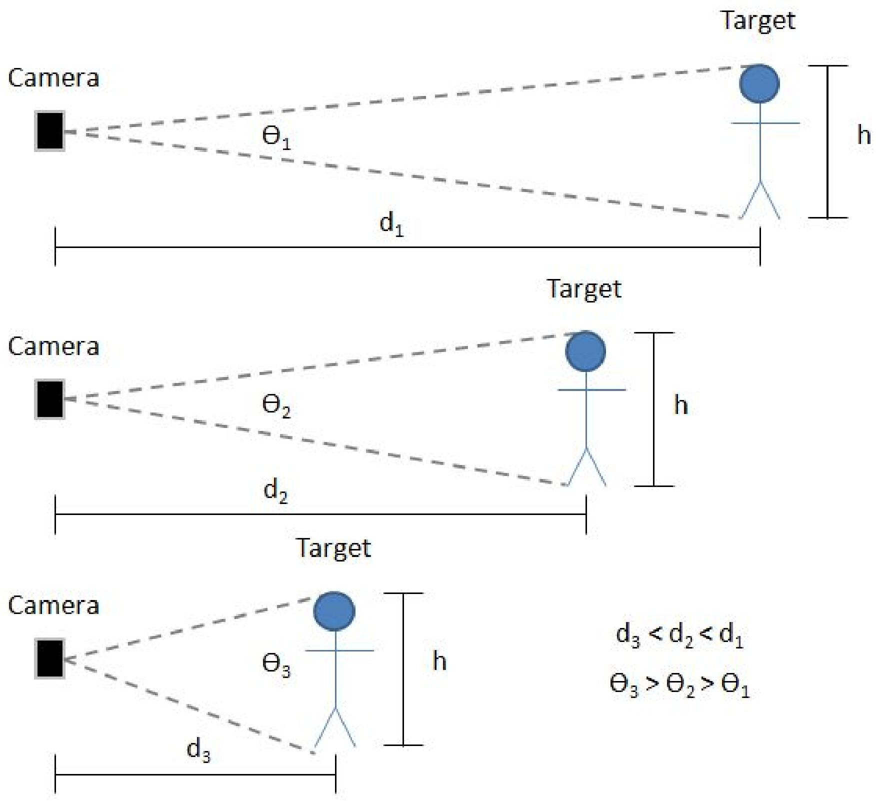
Learning object scale from multiple view geometry: Final Report

Ahmed AlMutawa(ahal4066), Austin Anderson (amanders), Rohit Raje (rora9362), John Stechschulte (jost6111), Qingjun Wu(qiwu5520)

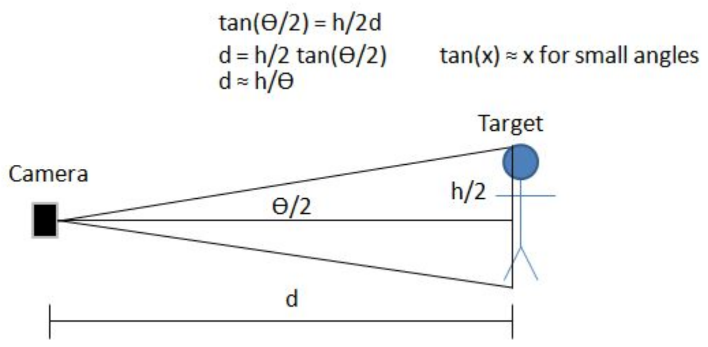
Project Goal
Robots navigating unknown environments must estimate the scale of the world around them. This can be done with active depth sensors, stereoscopic cameras, or fusion of data from multiple sensors. The goal of this project is to explore an alternative approach to learning environment scale by using a single camera and machine learning algorithms trained to estimate object scale from RGB images. At this stage, only learning depth from pre-segmented, labelled objects will be addressed. The identification and segmentation of objects from an image is left for future work.

The NYU Depth Dataset V2 was used to train several regressions--one for each object class--to predict the depth of an object segmented from an image. The segmentation given in the labelled data was used to isolate individual objects from an image. Features and the average depth were extracted from the segmented RGB images and depth data, respectively. A multivariate linear regression was trained using the extracted features as input and average depth as output. Software was written in MATLAB as the NYU Depth Dataset and its companion toolbox are in MATLAB.

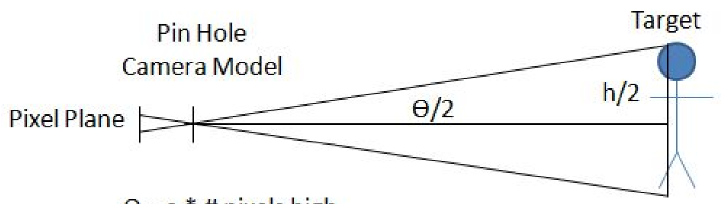
Project Background
Certain classes of objects will all have a similar size. For example, the height of humans is well described by a normal distribution so that a large portion of the population will be within one or two standard deviations of the mean. If a robot sees a human the odds are that person is around 5 ft. tall and not 10 ft. tall or 2 ft. tall. If the robot can estimate the size of a known object, it can estimate the distance to that object. Using cameras the scenario is illustrated below:



Here a camera is viewing a person of a fixed height as they move closer. As the height remains constant, the angle the person occupies in the image increases. Using trigonometry, the distance can be estimated from the angle and the assumed height:



Here if we assume small angles the relationship becomes very straightforward. Using a pinhole camera model, the angle an object occupies in the plane of an image can be estimated as:



$\theta = a * \#$ pixels high
a –> scalar conversion from pixels to degrees
$a \approx$ Camera Field of View/Total Pixel Height
For Kinect:
$a_{vert} \approx 43°/480$ pixels
$a_{horiz} \approx 57°/640$ pixels
$d \approx h/a_{vert} * \#$ pixels high

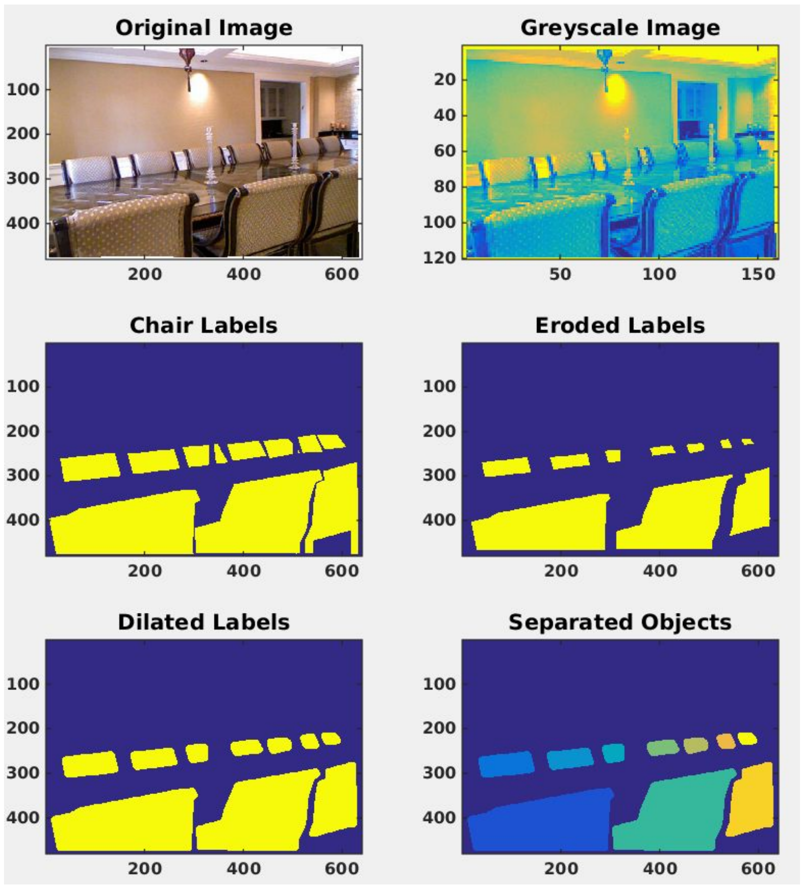
From: https://msdn.microsoft.com/en-us/library/jj131033.aspx

This illustrates that if the assumptions of similar object size and small angles hold, the distance to an object can be approximated by determining the pixel height and width of the object, inverting and applying a scale term. This very basic approximation will be used as a baseline classifier.

Implementation

Estimating distance to an object is a continuous prediction. The core estimator for this project is a linear regression that calculates weights for combining features extracted from the image data. The NYU dataset provides RGB-D images that have objects individually labelled and segmented. The dataset includes 894 different, labelled object classes. The full dataset is broken into a training set and a testing set for cross validation using a ¾, ¼ split.

For a single object, the relevant training and testing images are extracted, and a preprocessing step is applied to separate individual objects of the specified type in each image. This is done by generating a binary mask of the labelled object indices and applying an erosion and a dilation filter. The erosion filter eliminates small objects and connections between objects. The dilation filter grows the mask uniformly to return to the original size. A connected component analysis is then run to isolate individual objects. These isolated objects are treated as separate data points for training, labelling, and testing. This step is especially important for objects that appear multiple times in a single image, like chairs:



The code extracts the following features from the isolated RGB images:

MATLAB Feature Index	Feature Name	Description	Explanation
1	Bias	Bias term	Accounts for mean offset in data.
2	1/sqrt(npix)	Inverse square root of number of pixels attributed to the object	Taking an inverse and square root should linearize the area trend for increasing distance.
3	dx	Maximum pixel width	Pixel width should be related to object width and decrease the farther away an object is.
4	dy	Maximum pixel height	Pixel height should be related to object width and decrease the farther away an object is.
5	x	Mean x position	No physical relation to object distance should just relate to offset from

			camera boresight, although may uncover an unexpected relationship.
6	y	Mean y position	No physical relation object distance should just relate to offset from camera boresight in elevation, although may uncover an unexpected relationship.
7	dx_inv	1/dx	Inverse of pixel width, should vary linearly with increasing distance given fixed size, small angle approximation.
8	dy_inv	1/dy	Inverse of pixel height, should vary linearly with increasing distance given fixed size, small angle approximation.
9	num_corn	Number of corners extracted from corner detection	The total number of corners found by a corner detector. It was assumed that the larger, more densely featured an object, the more corners it would produce, and therefore be related to distance.
10	pca_eigv	First eigenvalue of singular value decomposition	See discussion below.
11	gray_diff	Max gap between gray values	See discussion below.

The labelled depth is simply the average of all depth values associated with an isolated object.

Two major functions were developed one_obj_run.m and main_loop.m. One_obj_run.m runs an in-depth analysis of a single object type and can be used to tune preprocessing parameters to maximize fit performance. Main_loop.m runs a bulk analysis over all object types using a single preprocessing configuration. The results suggest which object classes warrant further analysis.

A variety of features, both hand-engineered and learned, were tested. Most of these were rejected since they did not improve the testing error, and instead served to enable overtraining. Rejected features include,

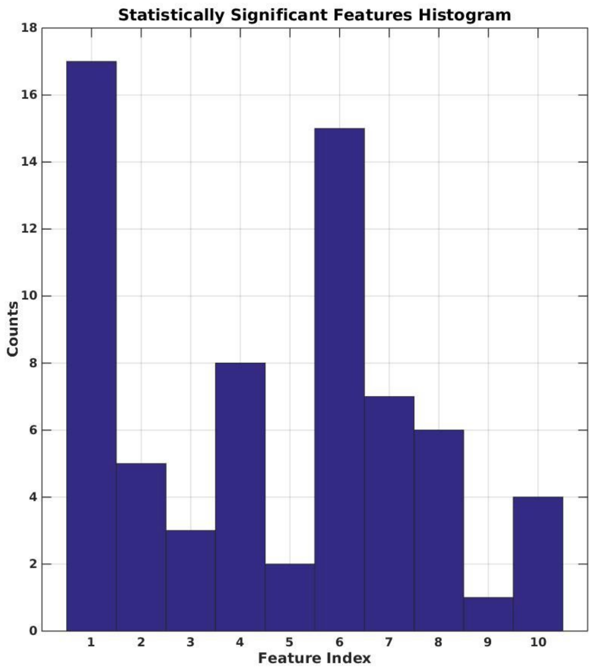
- Sparse filtering [1], an unsupervised feature learning algorithm that learns a linear transformation from a large number of input features to a small number of output features with advantageous sparse characteristics. Because a one-layer model was computationally intractable, a two-layer model was used. The first layer acted on image patches, generating intermediate features that were fed through the second layer, which had 20 output features.

- Principal Component Analysis (PCA). The largest component should carry information about object size and be more orientation invariant.
- Water filling [2]. This algorithm yielded statistics about max/min gray value of connected area, maximum ratio of connected area to gray value difference.
- Hough Transform. Detects lines and peaks within an image, which could be a good identifier of various objects. A variant of this technique, the Circular Hough Transform, was also tried.
- Edge detection. This generated too many features, and overlapped significantly with corner detection.

Results

As expected, the classifier works better for some object classes than others. For instance, the "monitor" classifier performs far better than the "wall" classifier. There are multiple reasons for this, such as variation within an object class (walls come in a wider range of sizes than monitors), and variation in an object's setting (walls are more likely to be occluded, or only partially in the image frame, than monitors). In the target application, the robot would only use more reliable objects for scale determination, and could also include a measure of uncertainty in the result.

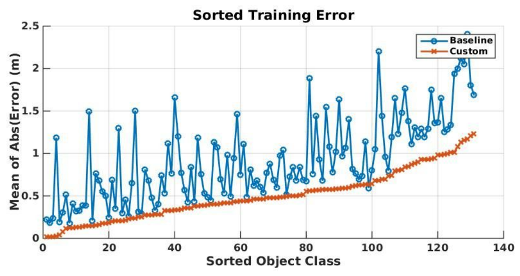
Initially, the whole data set was processed with the full set of features. Object classes with an insufficient number of training examples were discarded. For each acceptable regression, statistics on the estimated weighting coefficients were calculated. These statistics included a p-test to assess the statistical significance of the different feature weights. In the first run using all 10 features, regressions were run on 115 object classes. Based on these results the performance of individual features was assessed:

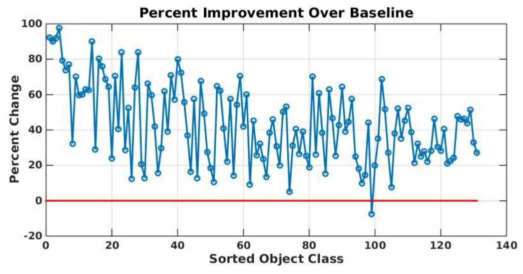


```
'bias'    'npix'    'dx'    'dy'    'x pos'    'y pos'    '1/dx'    '1/dy'    'num_corn'    'pca eig'
```

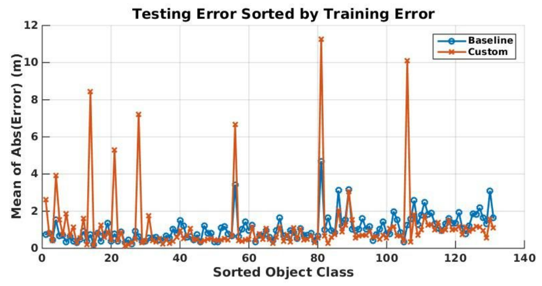
From this analysis it was determined that dx, x-pos, and num_corn were underperforming and were eliminated to reduce overfitting and processing time.

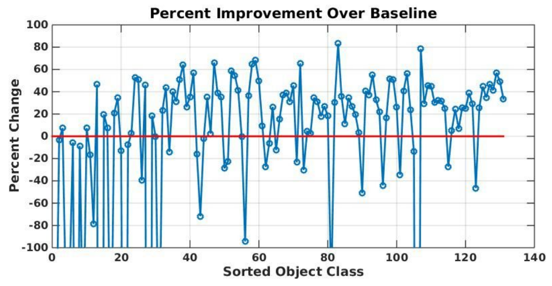
Using the reduced set of features the regression model was run against the full dataset. With fewer features, 131 object classes had sufficient data for the regression. The following plots show the resulting performance sorted by increasing error of the training set:





The lower plot shows the performance improvement over baseline. With the exception of the ‘decorative plate’ class the custom feature set outperformed the baseline in every object class. The testing error is shown in the following plot sorted again by training error:





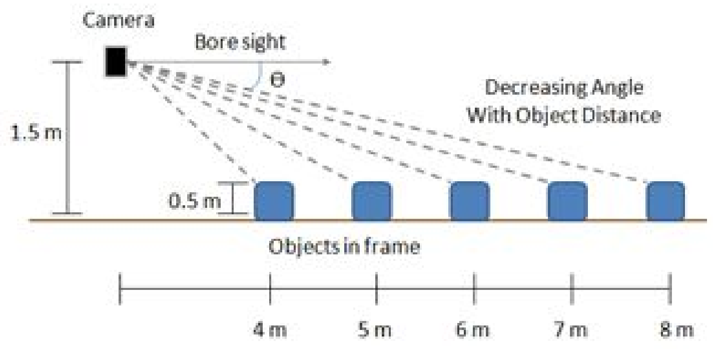
There is an increasing trend visible in the testing error that matches the training trend it was sorted by. There are a number of outliers for which the regression performs poorly. The lower plot shows improvement over baseline. While many test cases performed better than the baseline, there are now more cases of the baseline outperforming the custom classifier.

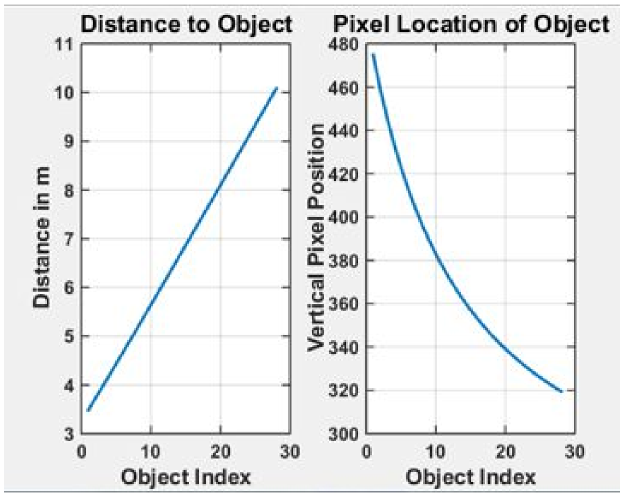
The tables below show the mean absolute error for the best and worst object classes based on testing error:

Best Object Classes	Testing Error (m)	Training Error (m)	# Training Images	# Testing Images	Best Feature	Worst Feature
‘toilet paper’	0.14	0.21	16	2	1/dx	y pos
‘tea kettle’	0.15	0.15	10	3	1/sqrt(npix)	1/dx
‘placemat’	0.19	0.14	23	17	y pos	dy
‘sink’	0.21	0.22	108	35	y pos	pca_eigv
‘paper towel’	0.24	0.28	32	11	1/sqrt(npix)	dy

Worst Object Classes	Testing Error (m)	Training Error (m)	# Training Images	# Testing Images	Best Feature	Worst Feature
‘pipe’	11.25	0.56	9	3	1/dx	1/sqrt(npix)
‘flower pot’	10.12	0.74	9	4	y pos	pca_eigv
‘classroom board’	8.44	0.15	10	4	dy	1/sqrt(npix)
‘mail shelf’	7.21	0.24	8	4	dy	pca_eigv
‘bicycle’	6.66	0.42	11	2	bias	1/dx

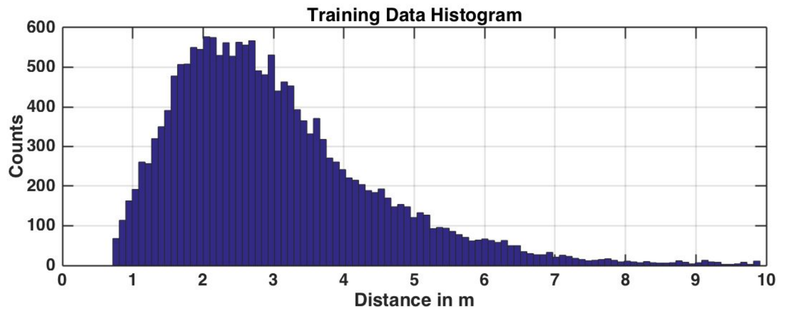
Objects like ‘toilet paper’ and ‘paper towel’ are typically shaped like cylinders, and so have well defined sizes and constant width at all viewing angles. Inverse pixel height performed well for toilet paper and the linearized number of pixels performed well for paper towels. Objects like ‘sink’ tend to be in well defined locations in a scene, specifically in a countertop that is mounted to the floor. The pixel y-position of objects tended to perform well for a wide variety of object types. This was counterintuitive at first; an analysis was performed to understand it.

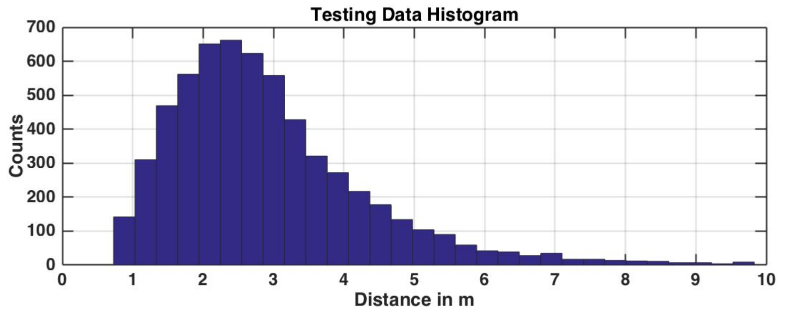




For a camera at a constant height, if an object moves along a plane such as the floor, it will appear to ascend in the vertical axis of the image. This explains the strong y-pos correlation observed throughout the data.

The strength of the bias term can be understood by considering the distribution of depths in the training dataset. The histograms below show the number of objects at various depths across all object classes, in the training and testing datasets.

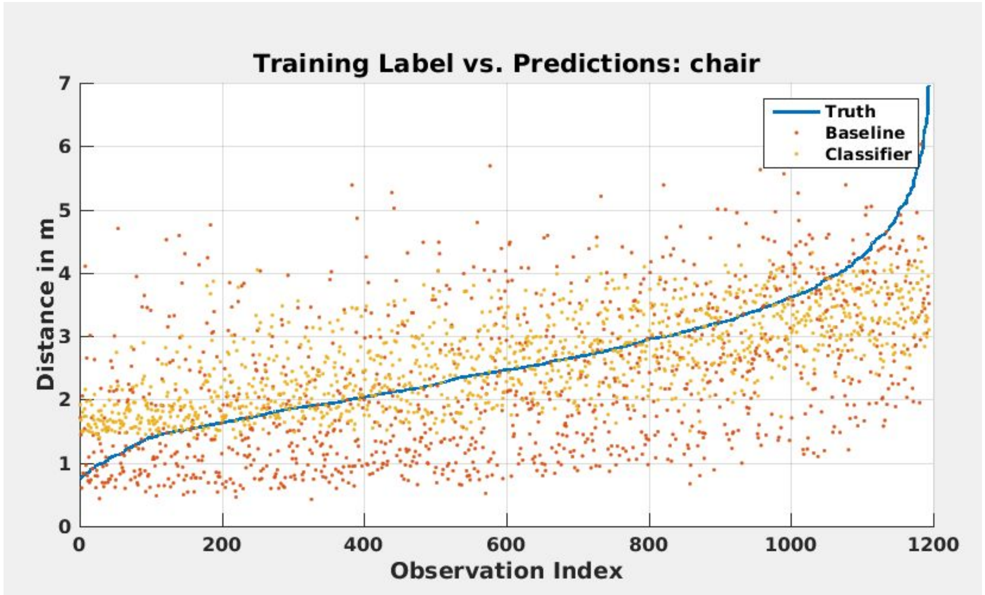


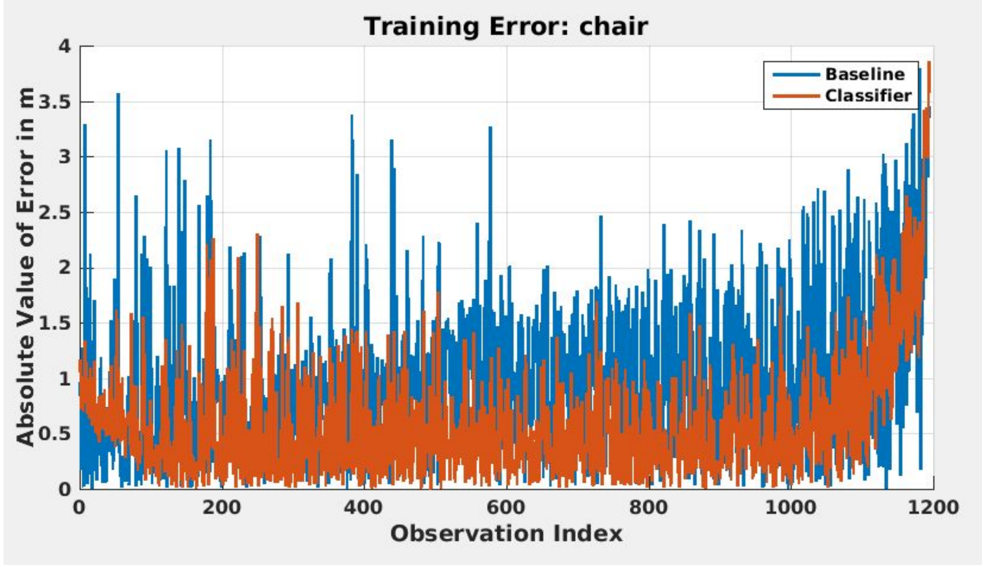


The bias term is strong because the depth distribution has a strong mode--with no other information, the regression does pretty well by guessing an object's depth is 2.5m. Ideally this distribution would be more uniform to improve the performance of the classifier in more generalized scenarios.

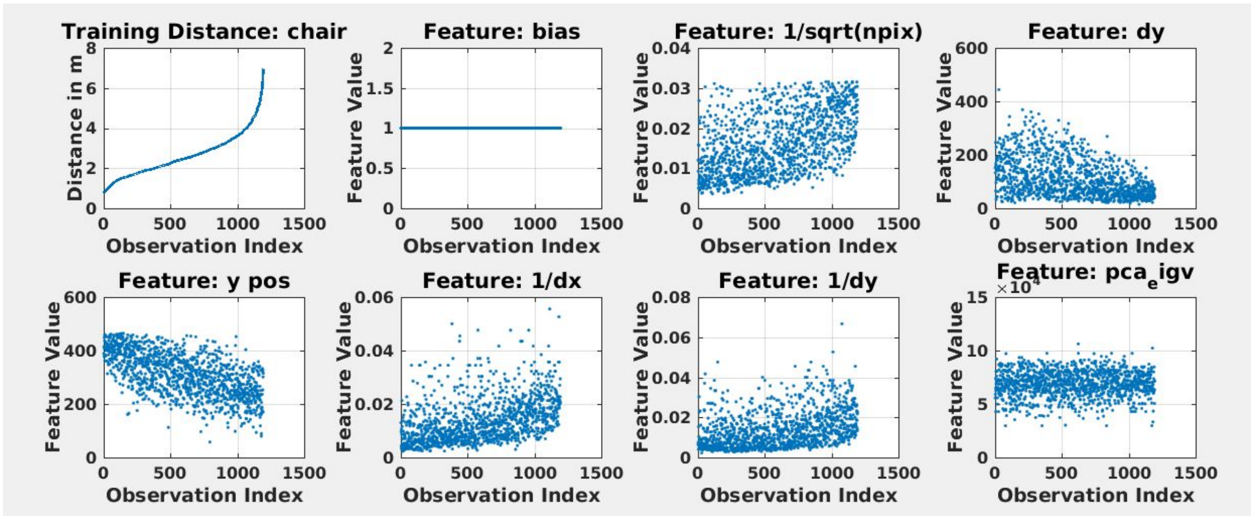
Error Analysis

An in-depth error analysis was performed on the 'chair' object class. Chairs were believed to be a good object type as they are ubiquitous in indoor settings and tend to have a standard size. Using the full data set and the reduced feature set, the individual observation fits were computed and plotted for the training data:





Higher error is present below 2 m and beyond 4 m. This can be attributed to the distribution of the label data, discussed in the previous section. Individual features are also plotted for this test case, sorted by labelled distance:



It is clear that the regression is fitting to very noisy signals. Trends are visible in the y-pos and 1/dx plots, but both features are still noisy. The pca_eigv and dy on the other hand show no clear trends.

An explanation for the noisy behaviour can be found in the image segmentation figure. There, 10 individual images of chairs are generated from a single scene. All chairs are the same type, but their label masks look very different. Some chairs are partially occluded by the table, some are broken up by other objects, and some are clipped by the edge of the frame. This makes it difficult for the regression to observe “true” sizes of the object. As it’s calculated now, the pixel height and width features only correlate to true object height and width for unobstructed, upright objects viewed straight on. Similarly, objects with high aspect ratios will cause difficulties as rotating such objects will result in different pixel widths.

Conclusions

Across hundreds of object classes, the testing error varied by almost two orders of magnitude. Some classes of object have greater size variance, and factors such as occlusions and viewing angle can affect the result significantly, as well. This is to be expected, and is acceptable: in the target application, only the more reliable objects need be considered, and by aggregating information over many observations the effect of outliers is mitigated.

As is often the case in the application of machine learning, this work uncovered as much insight about the training data and how it was collected as it did about the problem. For instance, the effectiveness of the y-coordinate in predicting depth was unexpected, but makes sense in retrospect considering the data was collected by a human carrying a camera at a fairly consistent height and angle to the ground. This, of course, limits the usefulness of the particular results obtained here: images captured looking down from a quadcopter or up from a small ground vehicle would have very high error in this regression. However, the overall method is still valid, and could be repeated with new training data that is representative of the desired application.

Future Work

There are a variety of ways this work could be enhanced or extended. The goal application of scale determination would require also implementing object segmentation and recognition, likely through use of deep convolutional nets. Within the problem domain addressed here, improvements might be possible by augmenting the training data to make the regression more robust, and allowing parameters to be tuned for each object type and further determining how to tune them in a statistically rigorous and automatic manner.

Individual Contributions

Ahmed AlMutawa - Data preprocessing and data engineering

Austin Anderson - Initial regression architecture, image segmentation, statistical analysis of features

Rohit Raje - Feature extraction including corner detection and contour detection

John Stechschulte - Architecture of full processing loop, design of API for refined regression analysis, baseline performance design and analysis, researching and selecting dataset

Qingjun Wu - Feature extraction including PCA and water filling algorithms

Citations

[1] Ngiam, Jiquan, et al. "Sparse filtering." *Advances in Neural Information Processing Systems*. 2011

[2] Xiang Sean Zhou, et all. “Water-Filling: A Novel Way for Image Structural Feature Extraction”