Door sign Corner Detection

Suhas Dara[1] and Lucinda Nguyen[2]

[1,2]Department of Computer Science at University of Texas at Austin
[1,2]Gates-Dell Complex, 2317 Speedway, Austin TX-78712
Email ids: [1]suhasdara@utexas.edu, [2]lucinda.onguyen@gmail.com

***Abstract*— Robots need to have an understanding of their environment, including semantic awareness, to perform navigation and localization tasks. Robots can utilize Simultaneous Localization and Mapping (SLAM) in order to construct a map, and the PRISM system automates the typically manual job of semantics labelling. The goal of this project is to offer a more flexible door sign corner detection algorithm that can be utilized by PRISM. To accomplish this goal, two neural networks are utilized to first perform door sign binary classification and then corner detection.**

I. Introduction

In order to accomplish common tasks in robotics, a robot is often required to travel to a specific location or explore an area safely. As a result, it is useful if a robot can construct a map with labels of significant landmarks marked on it for navigation and localization. While SLAM algorithms allow for an automatic mapping of an environment, the Pose Registration for Integrated Semantic Mapping (PRISM) system at University of Texas at Austin (UT) automates the process of localizing door signs (a common landmark) and annotating their locations on a map.

PRISM currently uses a heuristic approach to recognize door signs and detect the corners of door signs. A door signs corners can be used to calculate its homography, a 2D projective transformation that can be thought of as the image of a rigidly transformed 3D plane that has been projected into 2D space. Currently, PRISM can only detect one type of door sign in the UT Gates-Dell Complex (GDC). In our approach, we present a more flexible door sign corner detector, which uses deep learning to detect corners of one type of door sign. But, this approach can be trained to recognize multiple types of door signs in the future. The result of this project can be utilized as a plugin for the new version of PRISM. This approach can be very practical because of the consistency of door signs' design and shape, combined with the flexible nature of neural networks.

II. Background

Our project has two parts: detection of door signs and detection of the door sign's corners.

Note: In this paper, we commonly refer to two different types of door signs in our datasets, and in our environment. Type 1 door signs are just room numbers as seen in Fig 1. Type 2 door signs have room numbers along with a room name.

A. Corner detection

Currently, PRISM does not use deep learning to recognize door signs, and instead uses a heuristic approach. The PRISM classifier filters color and extracts any edges visible within the image. If the edges represent a deformed or transposed rectangle, then text detection will be performed to verify if the detected object is a door sign or not. Currently, the PRISM classifier can recognize only type 1 door signs. Similarly, we implemented a door sign corner detection algorithm that can recognize only type 1 door signs for now. But, because of the flexibility of deep learning, our algorithm could be trained to recognize type 2 door signs as well.

B. Object detection

Many tasks require object detection, which is an important application in robotics. As said by Zhou et al., deep learning, specifically convolutional neural networks (CNN), have quickly improved and shown great results in the field of object detection. Deep learning is a branch of machine learning that deals with learning data representations to learn useful features [9]. Because of features such as shared weights and local connectivity, CNNs are able to form better generalization on classification and detection problems. Convolution occurs when a kernel/filter is used to slide over an input image to create a feature map. With multiple convolution layers, and with the addition of activation functions that introduce non-linear features into the neural network, CNNs have high versatility. [3].

In order to binary classify door sign images, we chose to use an existing object detection classifier and retrain it with a custom data set of door signs from the GDC building. Two common approachs to object detection are Regional CNNs (R-CNN), and You Only Look Once (YOLO). R-CNN takes in an image, proposes multiple bounding boxes, and then uses a classifier to see if any objects are in a bounding box. After this, R-CNN will tighten the coordinates of bounding boxes, eliminate duplicate objects, and re-score the objects. However, R-CNN is rather slow because it has to train multiple models separately and it needs to send every single proposed bounding box through a CNN, which spans multiple layers [5].

Another popular approach is You Only Look Once (YOLO). YOLO's detection system is framed as a regression problem and it uses a CNN on a single image to produce a bounding box for each detected object in the image along

with its confidence in the prediction. Splitting an image into grid cells, YOLO immediately calculates the bounding boxes' coordinates and probabilities for multiple objects. Bounding boxes are proposed when the center of an object is within a grid cell. One limitation of YOLO is its spatial constraints. Since a grid cell can only have one class, but possible overlapping bounding boxes, detecting nearby objects will be difficult. Another difficulty is recognizing objects in non-standard shape or weird positions [5].

Out of the two approaches for object detection, we chose YOLO as it is much faster than an R-CNN, and easy to manipulate and train custom datasets. For this project, the first limitation would be less likely to apply since multiple door signs will usually not be in vicinity of each other. Considering the second limitation, the robot will usually be facing the door sign directly from the front (or at a slight angle) and from a fixed height as the camera is mounted on the robot. Hence, the door signs will not be in extremely skewed positions.

C. Corner detection

Other than PRISM, which is under research at UT, there are some other corner detection algorithms, such as the Harris Corner detection and the Features from Accelerated Segment Test (FAST) algorithm. The Harris Corner algorithm is more popularly used than others. Harris corner detection, similar to PRISM, first detects edges in the image and then detects collisions between the edges, which it defines as a corner. The algorithm also performs extra optimization such as looking at the color gradients in the vicinity of detected edges and corners to determine whether the predicted corners are actually corners or not. There have also been adaptations of the of Harris Corner detection such as one done by Ye et al. [8].

The FAST algorithm classifies a pixel as a corner when it is darker by a certain threshold amount than a set of contiguous pixels in an area, or when it is brighter by a certain threshold amount than a set of contiguous pixels [6].

There are also some projects that use neural networks for corner problems. For example, Dias et al. implemented a neural network based corner classifier for binary and gray scale images. After edge detection is performed on an image, several sub-images containing edges are sent to a neural network, which binary classifies the images as containing or not containing corners [2].

For the corner detection, we chose to implement a neural network, which builds upon the output of YOLO. The bounding boxes will be utilized to tighten the scope of corner detection. We chose to use a fully convolutional neural network (FCNN), which is a CNN that does not utilize fully connected layers at all in the entire network. Instead, FCNNs commonly use transposed convolutions which unsample the output of the convolutional layers to restore the original image resolution. FCNNs are often used for image segmentation and image classification problems [4].

III. Method

The detection of the corners of door signs can be divided into four stages. In the first stage, binary classification is performed to detect if an image contains a door sign or not. If there is a predicted door sign, the second stage will involve pre-processing the image in order to be an input for the neural network. The third stage is corner detection where we run the neural network on the image and output a black image with four white blobs that marks the corners of the door sign. Because our neural network currently outputs large blobs for the corners, there is a fourth stage where the centroids of the blobs are obtained and made the predicted coordinates of the corners.

A. Training YOLO

In order to create a binary classifier, we retrained YOLO to only detect door signs. YOLO has a neural network spanning 24 convolutional and two fully connected layers for detecting objects. YOLO version 2 (YOLOv2) can detect 80 different types of objects, and furthermore can be trained with training datasets to detect custom objects [5]. In order to retrain YOLO, we had to gather a custom data set of images with positive and negative instances of door signs and also label their bounding boxes by hand if there were door signs.

The ASUS Xtion Pro Openni2 camera was used to capture all images for the dataset. Instead of capturing individual images, the camera was attached to the BWI robot and driven around in the posterior half of Human-Robot Interaction (HRI) lab while capturing a rosbag of the rostopic /camera/rgb/image_raw published by the camera. This rosbag was converted to images by using a ROS utility at a conversion rate of 5 frames per second. This conversion resulted in a dataset containing 2338 images.

The images were hand-labelled by carefully drawing bounding boxes around door signs in the images using a tool called BBox-Label-Tool-Master. This tool produced text files corresponding to each image that contained information (supervised labels) about the bounding boxes drawn for the particular image. We then had to use a python script to convert every label in the text files into the format required by YOLO.

The dataset was then split into two sets: one for training and one for testing. 90 percent of the images were used in the dataset for training. After setting up the YOLO configuration files, YOLO was trained with the training dataset in order to retrieve the weights for the door sign classifier [7].

After completion of training, the classifier was tested against the original training dataset, the testing dataset, and a completely new sample of images (another dataset) collected from the anterior part of the HRI lab captured with the same method as before. The analysis of the results is under the Evaluation section.

B. Pre-processing for corner detection neural network

During YOLO's training and testing process, the dimensions and location of the bounding boxes predicting the door signs were also retrieved as text files. We then used the

Fig. 1: An Example of a door sign prediction by YOLO

text files to crop their respective images to retrieve new smaller images that contained mostly the door sign and some surrounding background. This was the first step of pre-processing the images for our neural network. However, after retrieving the cropped images, we have a dataset of two different types of door signs. For the purpose of training our neural network, we decided to use only door signs of type 1. Type 1 images were handpicked from the cropped dataset.

Another obstacle before the dataset became training worthy was the size of the cropped images. Because we plan to use our project's output with PRISM, where the robot is driven to directly face the door signs, our neural network does not have a requirement to detect door signs that are too small. Hence, extremely small images unsuitable for training were filtered by a simple algorithm: if neither of the image's dimensions were greater than 80 pixels, then the image was excluded from our neural network's training dataset. After these pre-processing steps, our training dataset contained 240 images, with the largest image width at 223 pixels, and the largest image height at 159 pixels.

Lastly, the images were grayscaled. Grayscaling helps eliminate recurring color patterns from the images, preventing the neural network from learning erroneous data. Bui et al. showed that object classification using grayscale images had higher accuracy compared with color images among different types of classifiers. This was another reason for grayscaling the images [1].

After this processing, the cropped images from YOLO's output needed to have their corners labelled in order to be used as the training dataset for our neural network. The corners of each image were carefully hand-labelled to the best extent possible with a tool created utilizing the OpenCV library and mouse-click event handlers. This tool produces a text label corresponding to each image that contained eight floating point values, representing the X and Y coordinates of the four corners in clockwise order from the top-left to the bottom-left corners. For our neural network design we required image labels, and not text labels like the aforementioned. So, the text labels were then converted to image labels. Each image label is a canvas of black pixels with four white pixels representing the four corners of the door sign in the respective locations.

Lastly, since large datasets and examples are necessary for the neural network to perform well, data augmentation was performed on the training dataset in order to double our training dataset. The training dataset's size was increased from 240 images to 480 images by including horizontally flipped versions of the door signs. The text labels and image labels were also flipped accordingly to match the flipped versions of the door signs.

C. The neural network

Our neural network model is fully convolutional, and utilizes Pytorch, a deep learning library, for its architecture. We wanted our neural network to be able to detect the corners of the door sign passed in. For this purpose, we made each pixel of the image a single classification problem of being a door sign corner or not. As stated before, our input is a cropped version of the bounding box outputted by YOLO that has been grayscaled and needed to meet a size requirement of at least 80 pixels in either dimension. The expected output of the neural network is a black image with four white pixels to denote the corners (refer bottom-left and top-right of Fig 3).

We designed the neural network to follow a squeeze-expand pattern (as seen in Fig. 2), where the image is squeezed through a network of convolutional layers and then expanded through a network of transposed convolutional layers, to get back an image of the original size, that can be compared with the image labels created for supervised training. The purpose of this was to extract important features for the corner detection as the inputs convolved.

1) Additional pre-processing: The pre-processing of the neural network involves a few more steps than what is done through other procedures. The grayscale version of the image is read in as a PyTorch CUDA float tensor, along with the image label. When loading in the tensors, the images are padded with black pixels to make the size of the image 223 x 159 pixels (as these are the largest dimensions in our training dataset.) This ensures that all the tensors in our training dataset are of the same size, and can be collated into random batches for better performance while training.

2) Architecture: For the first half of the architecture of the neural network, we have four alternating two-dimensional convolutional and two-dimensional batch normalization layers. The convolutional layers expand the channel dimension of the tensor, while reducing the actual height and width dimensions. All the convolutional layers utilize a kernel size of 5, with a stride size of 2. The stride size of 2 is used to pick up less, but more important features from the image, because of the very high sparseness of the four corners. Additionally, the batch normalization layers normalize the input over the channel dimension, using learnable parameters, sampled from a uniform distribution from 0 to 1: U(0,1).

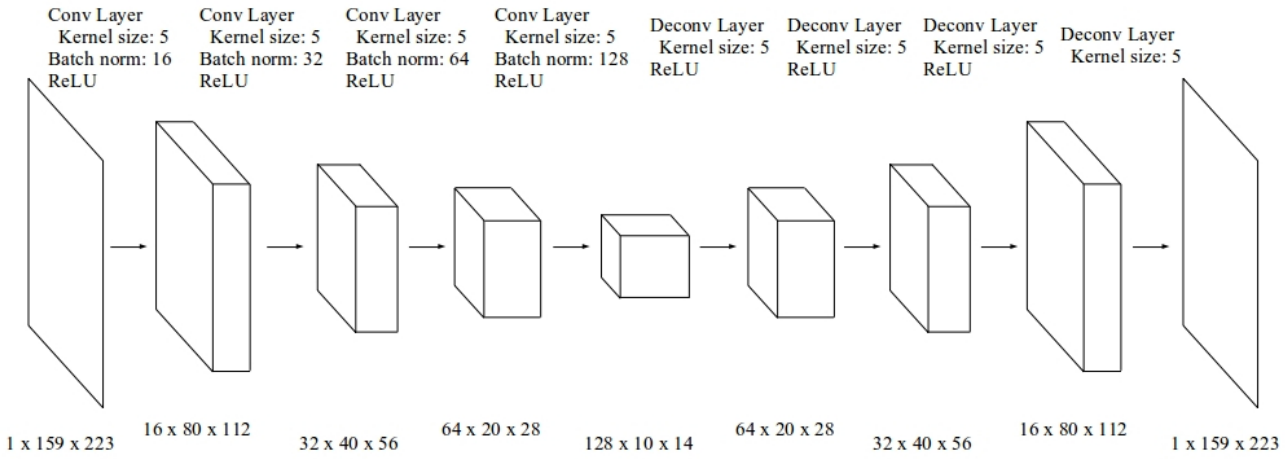

Fig. 2: The corner detection neural network has 4 convolutional layers and 4 transposed convolutional layers. Additionally, batch normalization layers were utilized to normalize the inputs to non-linearities in the outputs of the previous layers in the convolutional phase of the neural network. For the activation function, leaky ReLU was utilized. The neural network was designed in a squeeze-expand manner, to pick up the most important details, before expanding to the original image size again.

Additionally, after every batch normalization, the output is run through a leaky ReLU activation layer that does not completely eliminate negative values like a regular ReLU layer would. It minimizes negative values' effect on the weights instead of completely eliminating it as follows:

$$\phi(x) = \begin{cases} x & x \geq 0 \\ 0.01x & x < 0 \end{cases}$$

For the second half of the architecture of the neural network, we have four alternating two-dimensional transposed convolutional layers. These transposed convolutional layers reduce the channel dimension of the tensor, while increasing the actual height and width dimensions, or more simply, perform the opposite reshaping compared to the two-dimensional convolutional layers. After every transposed convolutional layer, leaky ReLU activation layer is applied, except on the last layer's output, as we want the final output not tampered with. This final output is of the same dimension as the initial input to the convolution layer, so that it can be compared to the image label for the calculation of loss for back propagation.

In neural networks, the loss function calculates how far away the neural networks output is from the target. This loss is used for back propagation which calculates the gradient of the loss function, and the optimizer will update the weights of the neural network as it is being trained. For the loss function, we utilize the PyTorch library's BCEWithLogitsLoss loss function which is a Sigmoid Cross Entropy function with an addition of the logit function. This loss function is stabler compared to a plain sigmoid function, a binary cross entropy loss function, or a manual combination of both. BCEWithLogitsLoss combines a sigmoid function with a binary cross entropy layer. The function computes loss as follows:

$$l(x, y) = reduction\_func(l_1, l_2, ..., l_N)$$

$$l_n = -w_n[p_n y_n \log \sigma(x_n) + (1 - y_n) log(1 - \sigma(x_n))]$$

This function utilizes the log-sum-exp trick to cancel out the large gradient of the sigmoid function for near-zero values (point of inflection). This allows for a smoother loss function.

We provide a positive weight that is applied to every cell in the input. This positive weight will have no effect on black pixels (as color black is represented by 0), but white pixels will get a much higher weight. The positive weight is calculated as follows:

$$pos\_weight = \frac{neg\_samples}{pos\_samples}$$

Each pixel is a single classification problem: black or white. In each image, there will be only 4 positive samples (the white pixels) and all the black pixels will be negative samples.

We set the loss's reduction function such that the loss function's output for each classification problem (each pixel) will be summed. If the reduction function is mean, the loss function would favor white pixels, and the loss function will try to maximize bringing out white pixels, by ignoring other pixels, which would result in a completely white output. Taking the sum instead of the mean allows for all pixels' respective weights to be taken into consideration while computing the loss.

A suitable optimizer will allow good results to arrive quicker during training. Our neural network utilizes the Adam optimizer with a learning rate of 0.001, which is an extended version of the Stochastic Gradient Descent (SGD) optimizer. The key advantage Adam optimizer offers over the classic SGD optimizer is its adaptive gradient algorithm, which allows a per parameter learning rate that improves the performances on image problems.

3) Training: The training dataset of 480 images of grey, cropped door signs and their respective image labels was used to train the neural network. For training, a batch size of 12 images was used, and the model was trained for 2000 epochs. This amounted to a total of 80000 iterations.

4) Testing: The original secondary dataset that was utilized for the testing of YOLO, was also utilized for testing our corner detection neural network. However, images that contained type 1 door signs were handpicked once again, as our neural network only detects corners for door signs of type 1. After running the handpicked images through the pre-processing, the testing dataset comprised of 49 images. These images were run through the trained neural network and the blob detection algorithm (refer following section III-D) to pinpoint the corners of door signs in the 49 images, and were evaluated for accuracy.

D. Corner blob detection

Our neural network currently outputs sparse white blobs for corners instead of four white pixels. So, we utilize a simple blob detector to pin down the corners. When the neural network successfully detects four blobs for the corners, the blob detector will output the X, Y coordinates for the center of each blob, which would represent a door sign's corner. In order to implement this, the neural network's output is binary inverted, and a threshold image is retrieved, with a threshold value of 220 pixels to filter out only the brightest spots in the image (indicating the white blobs). The SimpleBlobDetector utility from OpenCV library is utilized to detect the blobs. However, the blobs are still sparse and the blob detector may not detect the blobs.

The threshold image is run through some processing to highlight the blobs better. Erosion, a commonly used morphological operation, computes a local minimum over a small area of the image and replaces the pixels under that area with the minimal value. Hence, any areas with dark pixels (as the inverted blobs are black) will become larger, and make it easier for the blob detector to detect the blobs. The blob detector filters by black color and a minimum area of 25 pixels, but not by convexity, circularity, or inertia. These parameter choices were made because some of the blobs outputted by the neural network were not completely circular and were concave. The blob detector ignores blobs smaller than 25 pixels to eliminate small noise picked up by the neural network.

If less than four corners are detected with the blob detector, a dilation is performed on the image before the detector is rerun. Dilation, the sister operation to erosion, computes a local maximum over a small area of the image and replaces the pixels under that area with the maximal value. This allows bright regions (the white background surrounding the black blobs) to grow larger, and allows us to segment any blobs that fused together because of erosion. If at least four blobs are still not detected, the image is ignored, and deemed as having undetectable corners. Using the detected blobs (refer bottom-right of Fig 3), their centroids are extracted, which are the final corners of the door sign outputted by the neural network.

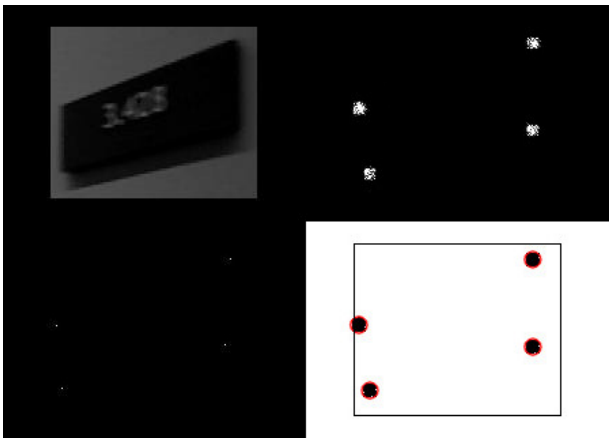

Fig. 3: An Example of corner detection process for a given door sign. Top-left shows a cropped, padded door sign. Bottom-left shows the hand-labelled door sign converted to an image label (only needed in training phase). Top-right shows the output of the neural network, with white blobs in corner vicinity. Bottom-right shows the detected blobs on the inverted version of neural network output.

IV. Evaluation

Analysis of the project's success contains two parts: the performance of retrained YOLO that performs binary classification, and the accuracy of the neural network that performs corner detection on images of door signs.

Evaluation of the retrained YOLO was done by running two different test sets through the YOLO detection system and gathering data on how accurate, consistent and confident YOLO was.

Evaluation of the neural network was done by running the training dataset and testing dataset through the neural network and calculating how accurate the predictions were. Accuracy was tested by calculating the Euclidean distance between the center of the blobs outputted by the neural network and the actual corners that were hand labelled, and then averaging the four distances per image.

A. YOLO re-training evaluation

YOLO has a set threshold for outputting whether it sees a particular object in a given image. This threshold is set at 25 percent by default for YOLOv2. Any prediction made by the classifier that has a confidence of less than 25 percent is automatically disregarded by the classifier. Before the classifier was run on the testing datasets, it was run on the training dataset again to check for overfitting. The average confidence level was between 75 and 95 percent, and not close to a 100 percent very often, indicating that the classifier did not overfit.

The classifier was then run on the primary testing dataset, a mutually exclusive subset of the original dataset. The average confidence level given by the classifier was slightly lesser than that of the training dataset, ranging from 70 to 95 percent, with very occasional predictions of a lower

confidence. The reason for such high confidence even on the primary testing dataset can probably be attributed to the origin of the images. Images in the training and primary testing datasets were part of the same video footage and were probably distinguishable only by a margin of a few pixels. This is why a secondary testing dataset was chosen that did not contain the same door signs as the other datasets.

The secondary testing dataset consists of 818 images, of which 466 images contain instances of door signs. The classifier was run on the secondary testing dataset, and out of the 466 images, door signs were identified in 369 of them. Although only 369 out of 466 door signs were identified, several of the images were instances of the same door signs. Despite the classifier not identifying all the door signs in their first appearances, it did identify every door sign eventually. Also, there was not a single image where the classifier detected a door sign when it did not exist (i.e. no false positives). The average confidence level of the prediction given by the classifier was also quite high at 84.85 percent, with a standard deviation of 8.64 percent. There were a couple of instances where the classifier only had a confidence level between 50 and 60 percent, but these are very unlikely instances as these values are more than three standard deviations away from the mean.

B. Corner detection neural network evaluation

There are two ways of evaluating the corner detection neural network. The first way is to look at the loss value during and after the completion of the training of the neural network. The second and better analysis is to compare the neural network output with the labels with math (refer section III-C.4).

1) Evaluation of loss: The value of loss was evaluated using the TensorBoard utility from the TensorFlow library.

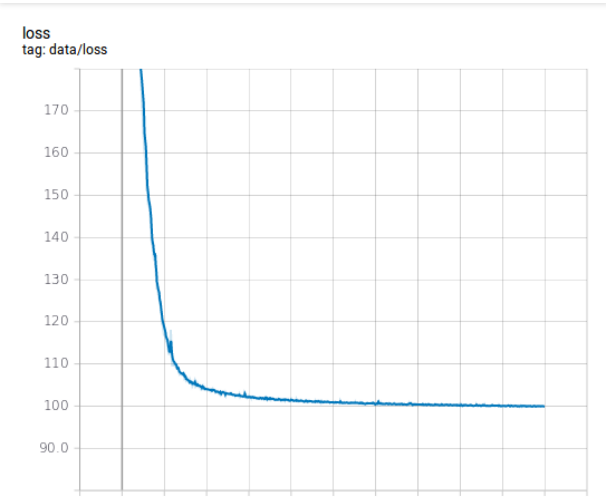

Fig. 5: TensorBoard's representation of the loss for the 2000 epochs of training

Loss was recorded after every epoch of training (refer Fig 5). The value of loss started very high, above 7000, but soon fell below 1000. The value of loss stagnated at around 100 after about 900 epochs. The value of loss at the end of training was 99.8. The value of loss is quite high, which explains why there are big, sparse blobs of white pixels in the neural network output instead of small, dense blobs or a single pixel. However, the overhead of such a large value of loss was mostly eliminated through blob detection and centroid calculation.

2) Evaluation of neural network output: We evaluated the performance of our neural network by following the methodology in section III-C.4. Euclidean distance between predicted and hand-labelled corners was calculated to be used as a measure of the error in prediction.

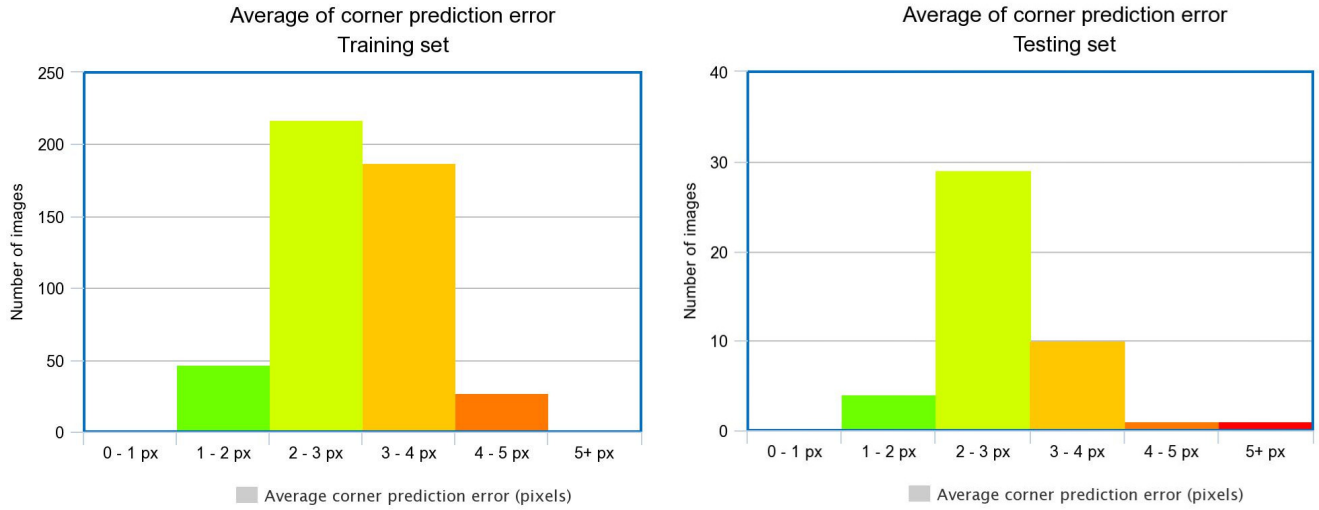

Fig. 4: Each histogram shows the average of absolute value of corner prediction error for each image in the given dataset. The left histogram shows the average error when the trained corner prediction neural network was run on the training dataset of 480 images. The right histogram shows the average error when the trained neural network was run on a testing dataset of 49 images (The histogram cumulative is 45 images as the neural network could not find corners in 4 images)

To evaluate the performance of our neural network, before it was run on the testing dataset, it was first run on the training dataset of 480 images to test for overfitting, in a similar way the retrained YOLO was tested for overfitting. The average Euclidean distance between a predicted corner and a hand-labelled corner was 2.862 pixels for the training dataset (refer to left histogram in Fig 4). Out of the 480 images, the majority of images had an average prediction error of 2-4 pixels, with 217 images having a deviation between 2 and 3 pixels, and 187 images having a deviation between 3 and 4 pixels. Only 2 images had average deviation less than 1 pixel. This data along with a high loss value shows that the neural network is not overfitted, while producing decent results.

However, the actual evaluation of the neural network is with a testing dataset. The original secondary testing dataset of 818 images was reduced to 49 images after pre-processing for the neural network. Out of these 49 images, the neural network combined with blob detection was unable to detect corners on 4 images. For the remaining 45 images however, the average Euclidean distance between a predicted corner and a hand-labelled corner was 2.729 pixels (refer to right histogram in Fig 4). Out of the 45 images, the majority of images had an average prediction error of 2-3 pixels, with 29 images falling in the category. This data shows that the neural network is effective whenever it does detect corners, but it is unable to detect corners consistently.

Upon closer evaluation of the images on which door sign corners were not detected, and the images that had a high prediction error, a few observations were made. Two of the four images were very dimly lit compared to the other images. For these two images, it was difficult even for the human eye to distinguish corners. The other two images however had good amount of lighting. The issue with these images was that one or multiple corners were very close to the edge of the image. Although the corners close to the edges were detected by the neural network, they were represented by quite sparse and concave blobs. The blob detection algorithm was either unable to recognize blobs shaped as aforementioned, or if recognized, had a larger margin of error. This shows that the blob detection algorithm needs to be optimized further.

V. Future Plans and Improvements

The short term goal of this project was to accomplish door sign corner detection, which was accomplished up to a large extent. However, the long term goal of the project is to be able to calculate a homography of the door sign, to be able to accurately position the location of the door sign onto the robot's map autonomously instead of manually. The autonomous approach currently used is PRISM, but using neural networks adds flexibility to what door signs can be detected and could possibly increase the accuracy of corner detection. To achieve the level of accuracy that the calculation of homography demands, a lot of improvements will be required. Although an average error of only 2.729 pixels is good, it is not good enough for calculating an accurate homography.

The best way to improve the accuracy of the corners is by improving the architecture of the neural network. The addition of pooling layers and/or linear layers could increase the effectiveness of the neural network largely. Also, with a more sophisticated loss function, which takes into account a normalized gradient for the positive weights instead of a single value, the corner blobs outputted would likely be smaller and denser.

Other improvements would involve a better way to detect blobs in the output of the neural network. A modified version of the current approach could find edges in the image and then find the pixel from a blob that is closest to an edge. A completely different approach to blob detection could be the OpenCV's built in K-Means algorithm, which can detect blobs in the image, as long as the number of blobs are known. However, for K-Means to work effectively, noise has to be eliminated from the output of the neural network to the best extent possible. This is to prevent K-Means from picking up noise as one of the four corner blobs.

Once sufficient changes to the entire neural network architecture are made, a homography for the door sign can be calculated. PRISM already supports the calculation of a homography if it is given corners. Hence, for the calculation of a homography, the corners outputted by our neural network can be used instead of the corners retrieved with PRISM's heuristic approach. Once all of the aforementioned is achieved for a single door sign (type 1), the architecture can be expanded and replicated with other door signs as well.

VI. Conclusion

This project utilizes a retrained version of YOLO to detect two different types of door signs in a given image. Any detected type 1 door signs will be pre-processed and passed into a neural network that can detect corners of the door signs. Currently, the neural network is incapable of outputting the exact pixel where a corner is located. Hence, blob detection is performed to obtain the centroid of each blob which is then labelled as one of the corners.

Overall, our approach utilizes two different convolutional neural networks in order to perform binary classification and then corner detection. Hence, there exists flexibility in two different places to make the door sign corner detection even more accurate. It also definitely provides more flexibility over the current heuristic approach used for the PRISM project, as a neural network can be stretched to also incorporate non-rectangular door signs, while the heuristic approach probably cannot because of its rigidity.

Acknowledgments

We thank the Peter Stone Laboratory for providing an opportunity to work as a part of the ongoing BWI project. We are immensely grateful to Dr. Justin Hart for providing insight and expertise during the research, Jamin Goo, and Rishi Shah for their assistance throughout the project. We are also thankful to Shivam Patel for providing direction to the project and mentoring us.

REFERENCES

[1] Hieu Minh Bui, Margaret Lech, Eva Cheng, Katrina Neville, and Ian S Burnett. Using grayscale images for object recognition with convolutional-recursive neural network. In *Communications and Electronics (ICCE), 2016 IEEE Sixth International Conference on*, pages 321–325. IEEE, 2016.

[2] PGT Dias, AA Kassim, and V Srinivasan. A neural network based corner detection method. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 2116–2120. IEEE, 1995.

[3] Ujjwal Karn. An intuitive explanation of convolutional neural networks., August 2016. Last Accessed 13 September 2018.

[4] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[6] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[7] Timebutt. How to train yolov2 to detect custom objects, May 2017. Last Accessed 25 October 2018.

[8] Zhinyong Ye, Yijian Pei, and Jihong Shi. An adaptive algorithm for harris corner detection. In *2009 International Conference on Computational Intelligence and Software Engineering*, Wuhan, China, Dec 2009.

[9] Zinyi Zhou, Wei Gong, WenLong Fu, and Du FengTong. Application of deep learning in object detection. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, Wuhan, China, May 2017.