AA 274: Principles of Robotic Autonomy
Problem Set 2

Name: Ashar Alam
SUID: 06265091 ashar1

February 13, 2019

Problem 1: Camera Calibration

No need to include images in writeup

Problem 2: Line Extraction

Table 1: Paramater values

Parameter	Parameter values
MIN_SEG_LENGTH	0.03
LINE_POINT_DIST_THRESHOLD	0.02
MIN_POINTS_PER_SEGMENT	3
MAX_P2P_DIST	0.4

Plots:

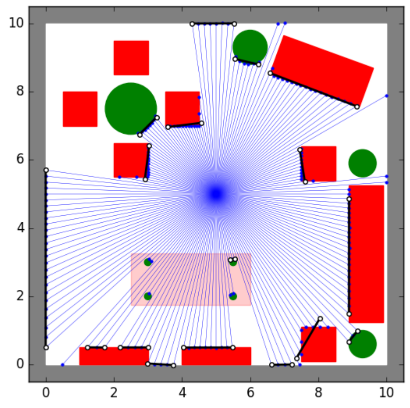


Figure 1: rangeData_5_5_180.csv

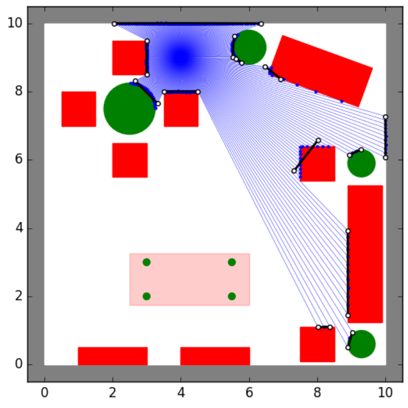


Figure 2: rangeData_4_9_360.csv

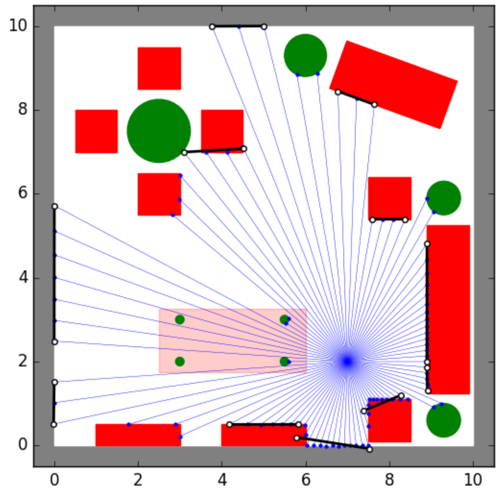


Figure 3: rangeData_7_2_90.csv

Problem 3: Tensorflow and HOG+SVM Pedestrian Detection

(i) Let's start with the figure in Page 9:

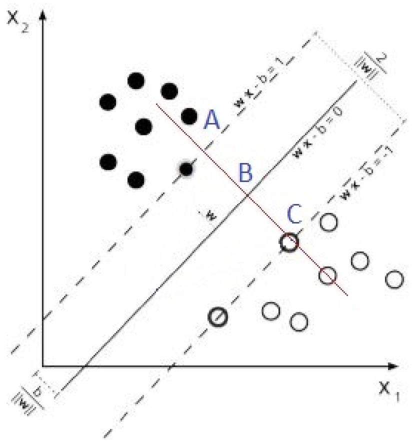


Figure 8 in Problem Statement

Let's assume point C lies on the hyperplane $\mathbf{wx} - b = -1$, which means $\mathbf{wx_0} - b = 1$.
To find the distance between $\mathbf{wx} - b = 1$, and $\mathbf{wx} - b = -1$ (AC); we need to find the perpendicular distance from x_0 (C) to hyperplane $\mathbf{wx} - b = 1$ (AC)
Since, $\frac{w}{||w||}$ is a unit normal vector for the hyperplane $\mathbf{wx} - b = 1$; we get:

$$\mathbf{w}(\mathbf{x_0} + \mathbf{AC}\frac{\mathbf{w}}{||\mathbf{w}||}) - b = 1 \tag{1}$$

as $(x_0 + AC\frac{w}{||w||})$ is a point in hyperplane $\mathbf{wx} - b = 1$

Let's expand equation (1)

$$\mathbf{w x_0} + \mathbf{AC}\frac{\mathbf{ww}}{||\mathbf{w}||} - b = 1$$

$$\implies \mathbf{w x_0} + \mathbf{AC}\frac{||\mathbf{w}||^2}{||\mathbf{w}||} - b = 1$$

$$\implies \mathbf{w x_0} + AC||\mathbf{w}|| - b = 1$$

$$\implies \mathbf{w x_0} - b = 1 - AC||\mathbf{w}||$$

$$\implies -1 = 1 - AC||\mathbf{w}||$$

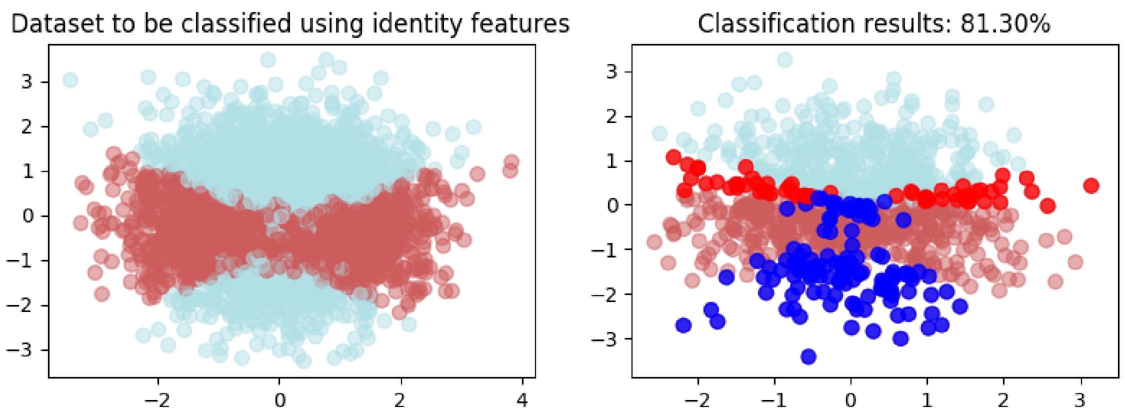
$$\implies AC = \frac{2}{||\mathbf{w}||}$$

Thus, the perpendicular distance between the two planes (AC) = $\frac{2}{||\mathbf{w}||}$ and minimizing $||\mathbf{w}||$ is equivalent to maximizing the margin as decreasing $||\mathbf{w}||$ increases $\frac{2}{||\mathbf{w}||}$

(ii) TensorFlow uses a dataflow graph to represent computation in terms of the dependencies between individual operations. It combines the computational algebra of optimization techniques for easy calculation of many mathematical expressions.
In Python, the Python runtime executes each operation directly and immediately discards any intermediate information used in the process. For such a simple procedure, there isnt really any intermediate information thats worth holding onto.
Lets look at an example: w = (j x k)/(l+m)
Then let's say computations are x_1 = j x k and x_2 = l + m $\implies w = x_1/x_2$. By storing the graph rather than simply executing its component operations, TensorFlow can help us to do operations x_1 and x_2 simultaneously. We can also analyze and optimize the graph and do things like compute an operations derivative. This would not be possible if we only kept the result of an operation and not a representation of the operation itself. Thus, this paradigm is also suitable for robots using vision with large dataset of images.

(iii) No need for write-up

(iv) The plot is featured below:



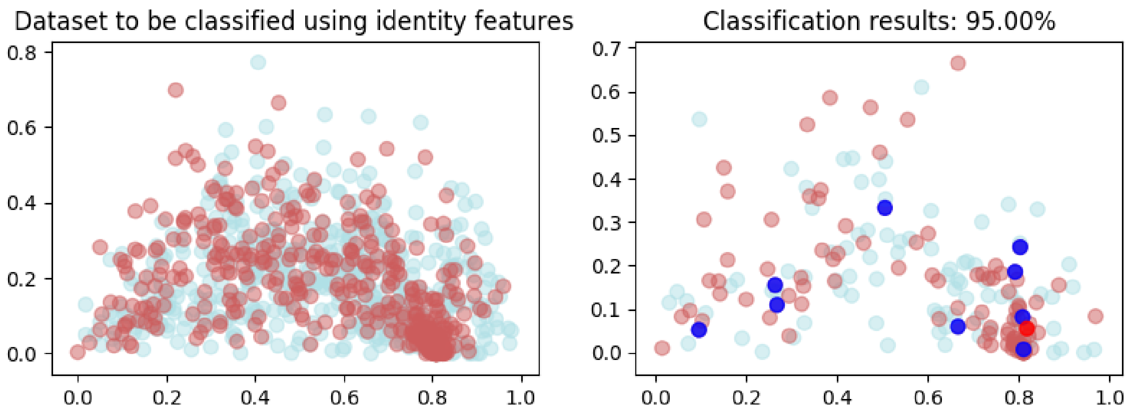
(i) Plot with Estimator

We can only get to an accuracy of about 81 %. As seen in the figure this is mostly because of the blue data points towards the bottom. This happens, because, we can only learn a separation line in the middle of data plane with the classifier and that results in misclassification of blue data points below that line.

(v) The identity features we have used account for linear fitting of the data. For two-dimensional cases, we may need non-linear, quadratic features for better fit and a better accuracy. Since, new features include non-linear terms x_1^2, x_2^2 & x_1x_2, I think this would **improve** our classification accuracy as we can get better fit for the data set using the new features.

(vi) No need for write-up.

(vii) I plotted a figure by copying the code for the plot given for Part (iv) to obtain a graphical representation below:



(i) Plot with Estimator 'Hog'

I got the accuracy of the model as 95 %

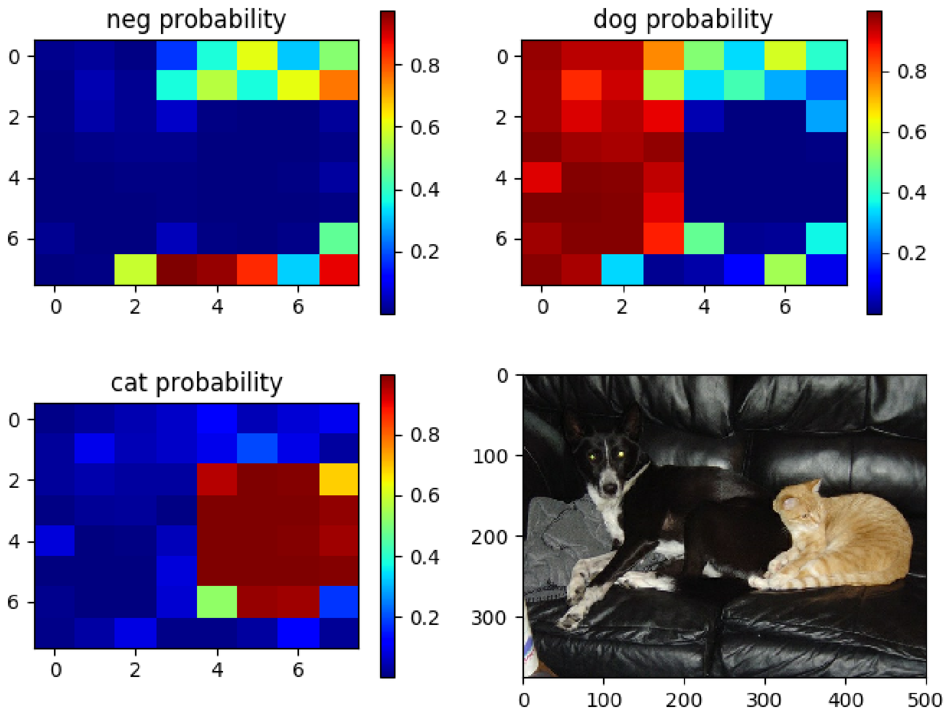
Problem 4: Classification and Sliding Window Detection

(i) The dimension of each "bottleneck" image summary is **2048**. For n-images, we will have image summary of a dimension n x 2048.
We have weight matrix of dimension 2048 x 3 and bias matrix of dimension 3 x 1 (essentially, a vector). Thus, the number of parameters we are optimizing in this retraining phase are (weights + biases) = (2048 x 3 + 3) = 6147

(ii) No need for writeup.

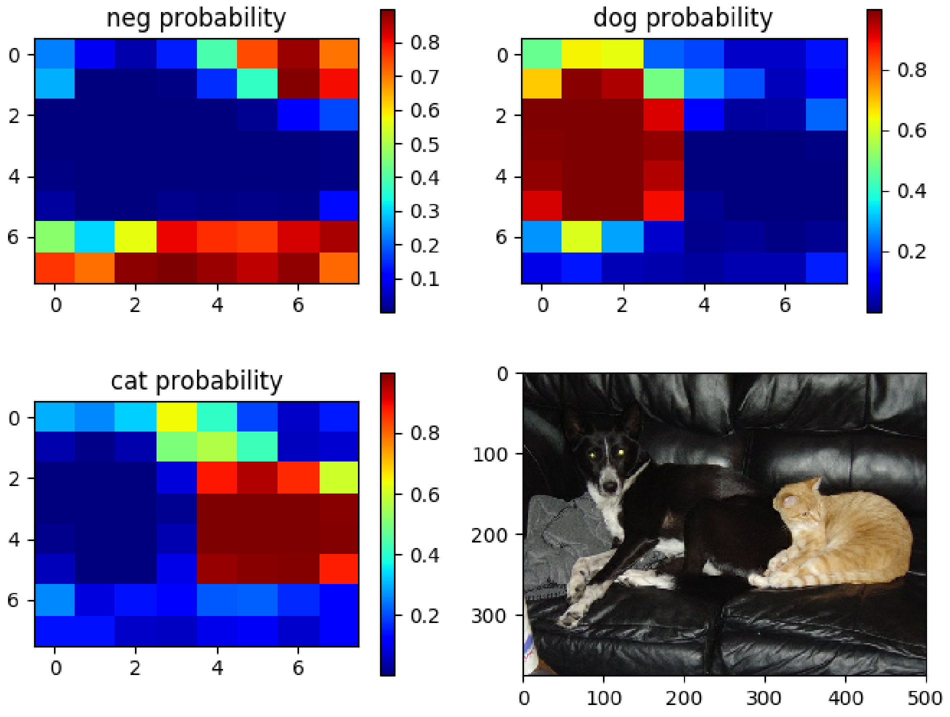
(iii) I chose my favorite image as:

Image # 009763 in 'catswithdogs' folder



(i) Detection plot for my favorite image using brute force classification .

(iv) Using convolution we get the following plot:



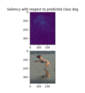
(ii) Detection plot for my favorite image using convolution.

(v) The final convolution layer feeds into the average pooling operation. The feature vector for the image as a whole is computed as an average of the feature vectors in the sub-region.

(vi) No need for writeup.

(vii) In the graphs for saliency, I observed that the highlighted pixels are used to identify the object as they represent the most distinct features of the class (here the animals). These highlighted pixels try to identify parts of the animal's body like head, body etc. using the saliency value for each pixel to identify its contribution to classifying images.
For the correctly classified images, we can see that the highlighted pixels in saliency graphs mark the silhouette of the most important features of the animal.
For the incorrectly classified images, we can see that highlighted pixels are agin used to identify the animals but this time, due to variation in background; the CNN mistook the cat for a dog because of highlighted features of human or the beams in the two images, respectively.

Result of compute_and_plot_saliency function implementation:



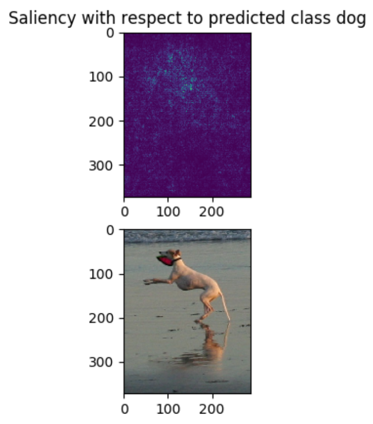


Figure 4: Image correctly classified as dog

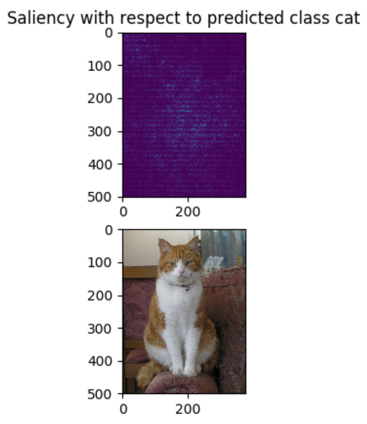


Figure 5: Image correctly classified as cat

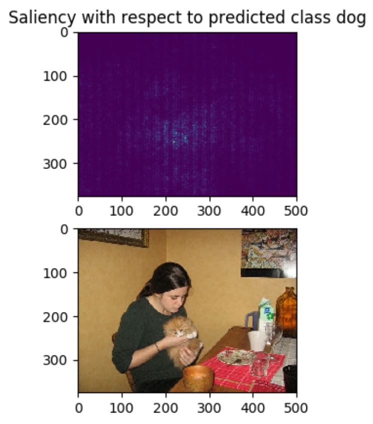


Figure 6: Cat image incorrectly classified as dog

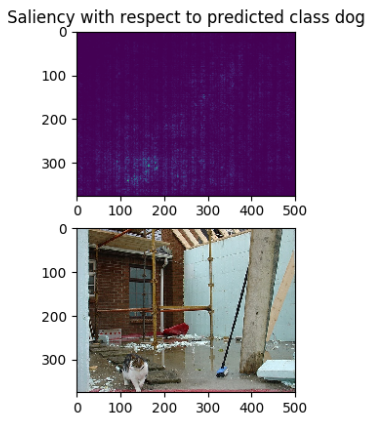


Figure 7: Cat image incorrectly classified as dog

Problem 5: Stop Sign Detection and FSM in ROS

(i) Superviser.py publishes the **current desired pose** to the pose controller as we can see in pose_controller.py. The type of the message is Pose2D

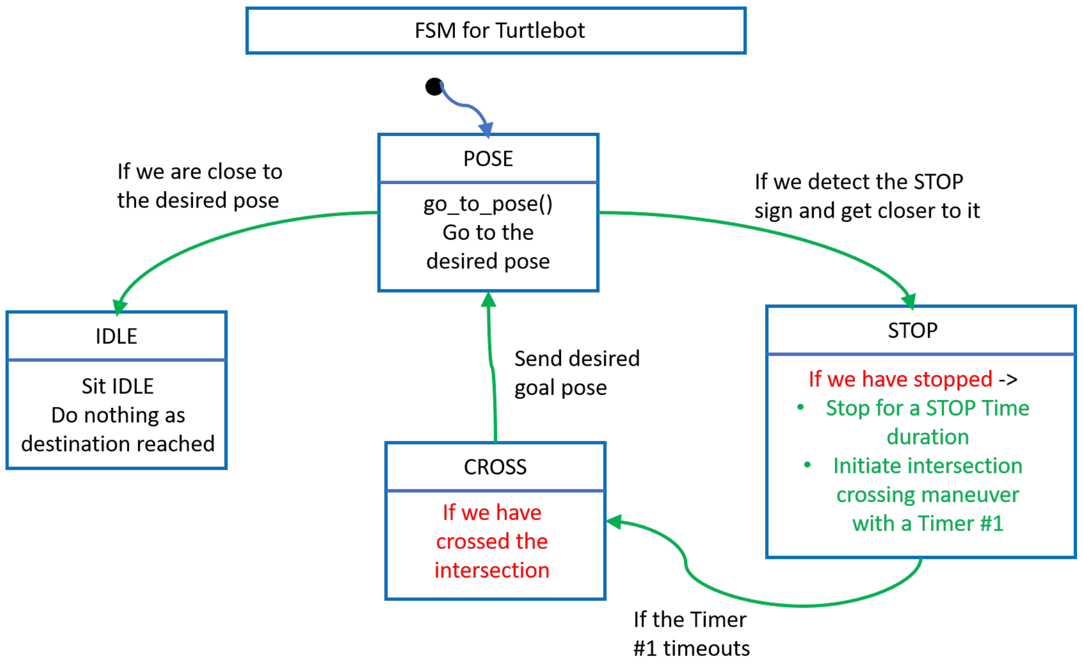
(ii) No need for writeup

(iii) No need for writeup

(iv) No need for writeup

(v) No need for writeup

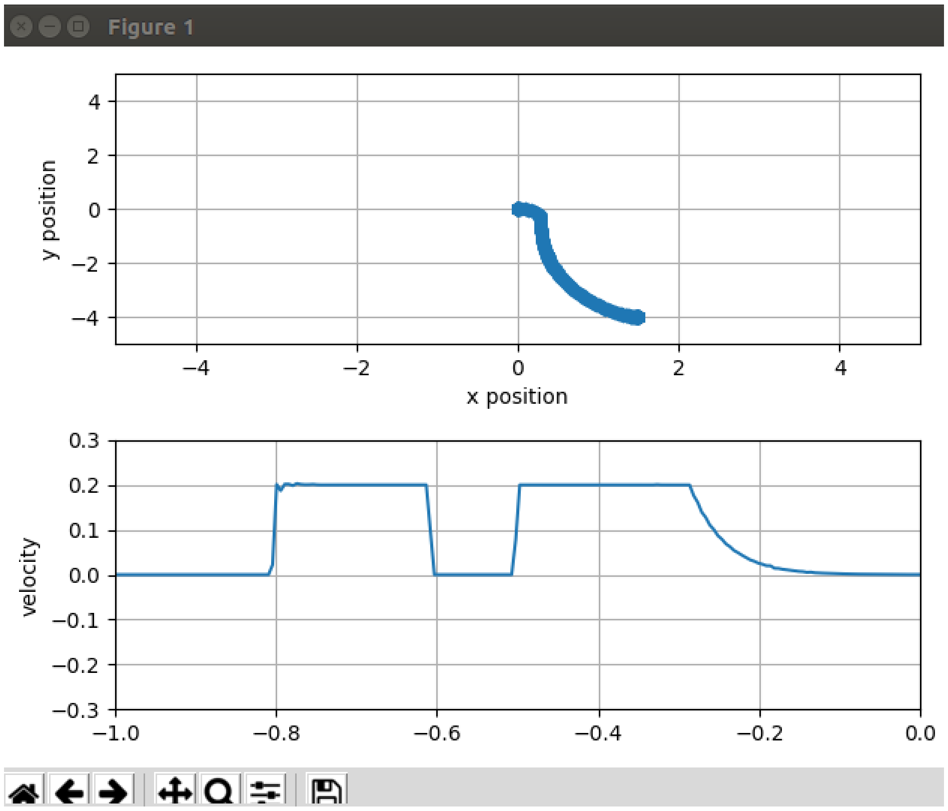
(vi) The FSM is drawn below:


FSM for Turtlebot
POSE
go_to_pose()
Go to the desired pose
If we are close to the desired pose
If we detect the STOP sign and get closer to it
IDLE
Sit IDLE
Do nothing as destination reached
Send desired goal pose
STOP
If we have stopped ->
• Stop for a STOP Time duration
• Initiate intersection crossing maneuver with a Timer #1
CROSS
If we have crossed the intersection
If the Timer #1 timeouts

FSM for Turtlebot

(vii) No need for write-up

(viii) The screenshot for path and velocity profile is included below:



Path and Velocity Profile for the Turtlebot