# Research on Model-Based Vision for Scene Classification

Jiuqi Wang, Moyang Chen, Sym Piracha, Emma Wu

Under the Supervision of Professor Joseph Vybihal

Department of Computer Science

McGill University

December 18, 2020

**Abstract— Recognizing scenes, where the robotic agent is in, is crucial for autonomous robot localization, mapping, and navigation. This report presents a study on a model-based vision system that aims to classify scenes based on input images. The system extracts features and properties that include dimension, lines, vanishing point, as well as the perceptual hash from both provided models and the input image. Then it measures the similarity between the image and the model features according to a self-defined cost function. The output is determined on the cost relative to the set threshold. We find that the lazy-training algorithm can attain a satisfactory level of accuracy only with ten model samples. The experiments show that although the attained accuracy is only around 56% on the validation dataset, the system with the final determined threshold achieves an accuracy as high as 87.5% on the test dataset. It is an encouraging discovery that we deem would open more possibilities for scene recognition of robots.**

## I. Introduction

Vision is one of the most used means that mobile robots perceive their surroundings and identify the scenes where they currently are. The visual data in general comes from the cameras that are attached and connected to the body of the agent. To enable scene identification, one typically needs to adopt a system that can classify vision data. Examples of classic image classifiers as of 2010 are SIFT feature-based methods[1] and support vector machines (SVM). The recent breakthrough of deep learning has revolutionized the field of image classification in terms of accuracy. The application of convolutional neural networks such as "AlexNet" [2] has reported an unprecedented performance in image recognition and classification. Nevertheless, they are all statistical learning methods that often require a considerable amount of data to train. In particular, convolutional neural networks typically require thousands of images for each class. In our project, we investigate a mode-based vision algorithm that only requires a small number of models as templates to classify scenes. The algorithm itself is lightweight and relies on much fewer samples for lazy training.

We obtained our models by manually sketching out the overall structure of our scenes. Then we apply the Canny Edge detector [3] on both the image and our models. After we have the edges, we then look for lines, vanishing points, and calculate the average hash [4] for both. These features are stored together and act as a "descriptor" for the model and the input image.

We compare the similarities between the model and the image by computing a cost function that takes in the features as input. In our experiment, we start with a binary classification problem, identify if a scene is a hallway or not. The algorithm outputs the result by checking if the cost is below a well-tested threshold.

The final test on a dataset that includes eight positive instances and eight negative cases reports an accuracy of 87.5%, which we believe is promising.

## II. Related Works

Feature extraction and comparison is a crucial component of our model-based vision system. The assumption is that the robot is capable of taking photos with an average quality camera. Hence, we take the problem out of the context of robot vision and work purely with static images as inputs. To extract the features and store them in a meaningful and comparable way, we refer to some related work in image processing and analysis: edge detection, vanishing point detection, and perceptual hash.

## A. Edge Detection

The Canny edge detector is a powerful edge detection algorithm proposed by John Canny in his paper *A computational Approach to Edge Detection, 1986*. It is still a widely used edge detector today because of its precision and robustness.

Before applying this algorithm, we smooth the image intensities with a gaussian filter to reduce the image noise. The algorithm first computes the gradient at each pixel position. For each coordinate, if the magnitude of the gradient is greater than the threshold, we consider it to be a candidate for edge point. Then it divides the directions into several sectors. For each candidate edge point, the algorithm checks if any of the neighboring positions has a larger gradient magnitude in the direction within the sector in which the gradient of the candidate point falls. If so, it removes the candidacy of that point. Otherwise, the position is a local maximum and will be considered as an edge point. This technique is called "non-maximum suppression," which tends to keep only those edge points whose gradient magnitudes are a maximum across an edge. Since the maximum often occurs very close to the exact location of the edge, the result of the Canny edge detection yields a high precision for most of the time.

The algorithm also implements two thresholds: lower threshold and higher threshold. For gradients with a magnitude smaller than the lower threshold, the points will not be considered edges. On the contrary, those with a magnitude larger than the higher threshold will have their position picked as edge point without conditions. In terms of those in the middle, they will pass only if their neighbors are edges: only those that can form a line with neighboring pixels will get picked by the algorithm.

## B. Vanishing Point Detection

The closely related work to our computation of a potential vanishing point cell is SZanlongo's project in their Github repo "Vanishing-Point-Detection." [5] Taking the result edges from edge detection as input, SZanlogo uses Hough Transform [6] to filter out trivial lines and connect lines that are close enough to each other, outputting a list of lines. After that, SZanlongo calculates the coordinates of all the intersections of these lines. Then, they use grid segmentation to divide the input picture into cells. SZanlongo loops through all the cells to output the cell that contains the maximum number of intersections, and it is this cell that will be considered containing a potential vanishing point.

Taking SZanlongo's algorithm for reference, we make some adjustments and improvements:

Firstly, we implement a line extending functionality that lengthens the detected lines, which will generate more intersections that will make the vanishing point stand out.

When using the concept of Hough Transform to detect lines from the edges, we refer to the Probabilistic Hough Transform to replace the regular one (cv2.HoughLinesP instead of cv2.HoughLines), which is an optimization that only computes a random subset of points to reduce the number of computations, which leads to better time efficiency.

The Probabilistic Hough Transform also directly returns the two endpoints of a line, whereas the regular Hough Transform only returns the parameters that define the line. Hence, optimization also results in more direct and cleaner codes.

Another adjustment we make when calculating the intersection of two lines is instead of working out the math to do the

computation, we turn the lines into LineString objects provided by the Shapely package and find the intersection point of the two LineString objects. By doing so, we get more concise codes and avoid potential errors.

A vanishing point implies the existence of intersections highly concentrated in a small area. Therefore, we significantly decrease the size of the grid to avoid counting intersections that are too far from each other and to filter out some irrelevant intersection points.

### C. Perceptual Hashing

Perceptual hashing is a technique that encodes the image information into a 64-bit integer fingerprint in an efficient manner such that similar images will have close fingerprints. Unlike cryptographic hashing, the perceptual hash values are not random. However, it does preserve the main property of hashing: each image generates a distinct but not unique hash value so that the same image always generates the same hash value, whereas the same hash value does not guarantee identical. In particular, it is useful when we compare the whole picture.

There are also different types of perceptual hashing, which depend on the hashing function. In our case, we adopt the average hashing to encode our images.

The first step of average hashing is to reduce the image size to 8x8, which ideally will preserve the structural information and discard details. Then it reduces the color space of the image by converting it into grayscale.

After the image processing step, the algorithm computes the mean value of the 64 intensities and uses it as the threshold.

Finally, the encoding step sets each bit based on whether the pixel intensity is higher or lower than the mean. This operation results in a 64-integer that captures the pixel information of the image.

To compare the similarity between the two pictures, one typically counts how many bits do the two hash values differ. A small difference implies a high resemblance and vice-versa. Since the computing power of embedded computers in robots is usually limited, perceptual hashing provides a fast way to encode and compare images irrelevant to the scale or aspect ratio.

## III. Method

Given an image, our goal is to classify it into one of the classes that we declare. Our approach is to apply a model-based technique that involves feature extraction and a cost function. This section will mainly focus on the detailed procedure of the algorithm.

### A. Model Preparation

We prepare our models by manually drawing white lines on a black background following the skeleton of the image (Figure 2). The purpose of sketching is to identify and abstract the general structure out of the scene with little or no noises. These models will act as templates that filter the input images, which is the foundation of our model-based approach.

Figure 1 Image of a hallway taken by cell phone camera.

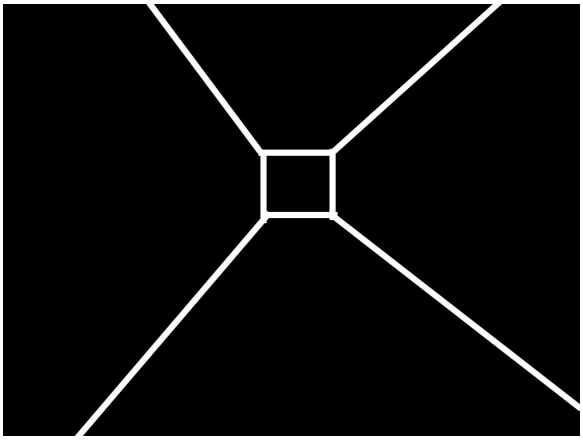

Figure 2 Manual sketch representing the structure of Figure 1.

## B. Edge Detection

We apply the Canny edge detector to detect edges for both the models and the input images (Figure 3). Before detection, we first smooth the input with a Gaussian filter with default window size = 9 and $\sigma = 2$ to reduce the noise. Through experimentation, we pick the lower threshold = 20 and the higher threshold = 60 to control the detector sensitivity.

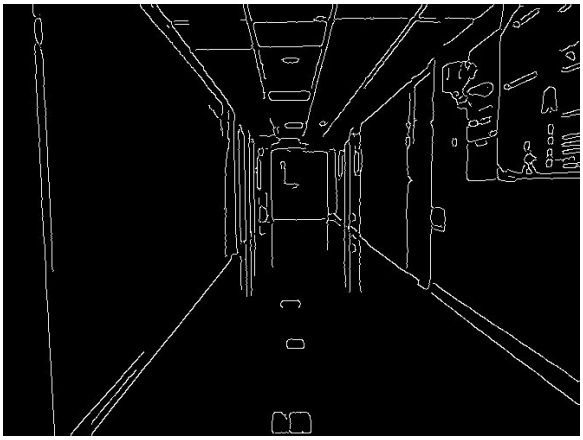

Figure 3 Detected edges of Figure 1 with the Canny edge detector and default parameters.

## C. Line Detection

After we have the edges, the next step is to detect the lines using them (Figure 4). We take advantage of the HoughLinesP function provided by the OpenCV library. The algorithm is based on the Hough Transform, a technique that iteratively votes for the target structure in the image space and takes the maximum.

The function also has two important parameters, namely minimum line length and maximum line gap. The first parameter sets a threshold such that the algorithm only outputs the lines with a length greater or equal to the threshold. The second one sets a threshold such that the colinear line segments will be connected as a single line if the gap between them is smaller or equal to the threshold.

Through experiment trials, we find that the minimum line length = 40 and the maximum line gap = 60 work well with our images. Thus, we set them as defaults. Besides, we also calculate the angle of each line in degree in the interval [0, 360).

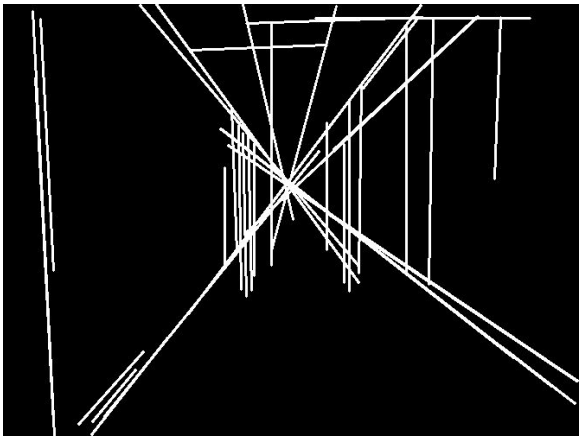

Figure 4 Lines extracted from Figure 3 using the HoughLinesP method and default parameters.

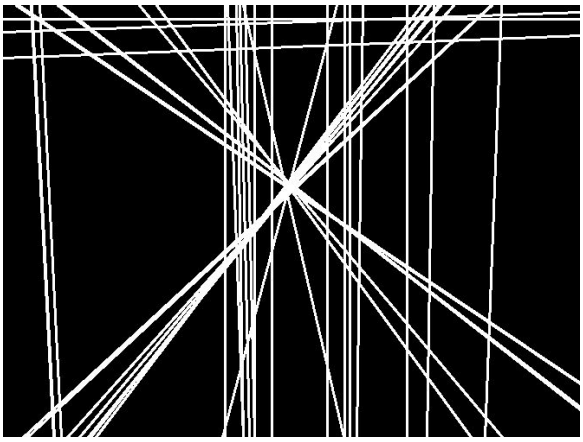

Figure 5 Lines extended from Figure 4.

## D. Vanishing Point Detection

Our algorithm for detecting vanishing points divides the image into square cells of equal dimension and outputs the cell that has the most intersections. Therefore, before applying the algorithm, we must find the line intersections. We first extend the lines so that they span the entire width or height (Figure 5). Then we compute the intersections based on where they cross.

One adjustable parameter in the vanishing point detection algorithm is the grid size. The larger the size, the more likely the output cell correctly contains the vanishing point and the more uncertain the exact position of the vanishing point. We choose 50 to be the default grid size.

## E. Image Hashing

We compute the average hashing for the edge graphs of both the models and the input images. The reason we hash the edge graphs instead of the original pictures is that we only intend to encode the structural information instead of the intensities. Thus, we discard the pixel intensities in the original images and only take the black and white edge graph for hashing.

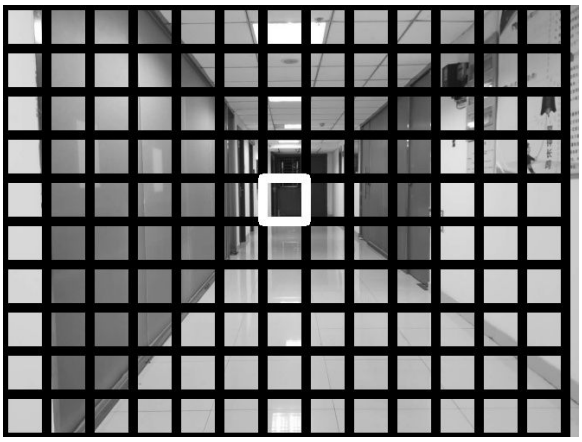

Figure 6 Vanishing point detection with grid size = 50 and intersections obtained from Figure 5.

## F. Line Cost Computation

In this section, we will mainly focus on our algorithm for computing the cost of lines. For two lines $L_1$, $L_2$ with lengths $l_1$, $l_2$, and angles $\theta_1$, $\theta_2$, the cost of them is defined to be the cost of angle plus the cost of length. The cost of angle is defined to be the absolute value of the difference between the angles over 360, where the two angles are converted to be in the interval [0, 180). Therefore, the angle cost is in [0, 0.5). The cost of length is defined to be the absolute value of $l_1$ over the diagonal length of image 1 minus $l_2$ over the diagonal length of image 2. Since the diagonal is the longest possible line within each image, the two ratios will stay in the interval (0, 1). Thus, the absolute value of their differences will also stay in the interval (0, 1).

To compare all the lines in the image to the model, we first divide both of them into four quadrants so that the lines in the image are only compared with those in the models that are spatially close to them. For each quadrant, we iterate through each line in the input image and

try to find one in the model that has the smallest angular difference from that line. After the closest line is matched in the model, the cost will be calculated before the line is removed from the corresponding model quadrant. If during the iteration, the model quadrant becomes empty, then we add up the cost of the length of each remaining line in the image quadrant. We sum up the cost of each pair in each quadrant and output the result.

### G. Vanishing Point Cost Computation

The computation of the cost of vanishing points is simple: The vanishing point detection algorithm will output the coordinate of the center of the cell that contains the vanishing point. The first step is to convert the coordinates into relative positions, namely converting absolute coordinate (x, y) into $(\frac{x}{width}, \frac{y}{height})$ so that the scale of the images will not affect the vanishing point coordinate. Then we add up the absolute values of the differences in both x and y coordinates between the vanishing point in the model and that in the image.

### H. Average Hash Cost Computation

Since the average hash algorithm already encodes the edge graphs into 64-bit integers. The cost of the hash between the image and the model is the number of different bits between the two integers.

### I. Prediction

The total cost between the input image and one model is the sum of the line cost, the hash cost, and ten times the vanishing point cost. We scale the weight of the vanishing point cost because they fall into the interval [0, 1], which is considerably smaller than the other two. We want to maintain a balanced weight among the costs.

To predict whether the input image belongs to the class, we first compute the cost of the image between each model in the dataset and outputs the minimum one. If the cost value is below the threshold, the system will accept it into this class. Otherwise, the system will reject it.

## IV. Experiment

### A. Dataset

We build up our dataset with the photos taken with our cell phones. The dataset is then divided into a validation set and a test set. The validation set includes 16 positive instances and 16 negative instances. The test set contains eight positive and negative cases, respectively. The validation set and the test set do not overlap.

### B. Threshold Localization

Since cost is unbounded and arbitrary, we need to pin down a threshold that optimizes the classifier's performance.

To achieve this, we peek at some typical costs from the dataset and use them as heuristics. Then we make a range of thresholds with equal intervals around the heuristics and run the algorithm with each one of them on the images in the validation set. We note down the accuracy and select the threshold that results in the highest accuracy. We keep repeating this step for a smaller interval around the current optimal parameter until the accuracy does not change much.

The accuracies on the validation set are around 56%, which is lower than what we expected. One possible reason that we can think

of is some of the staircase images have a very similar structure to our hallway pictures, which may result in a low cost produced by the algorithm.

### C. Final Test

After we pin down our threshold = 250, we run the algorithm once on the test set and report the final accuracy. To our surprise, although the algorithm has a bad performance when we are tuning the threshold on the validation set, the accuracy on the test set is reported to be 87.5%. A detailed report is shown in Table 1.

| | Positive | Negative |
|---|---|---|
| True | 7 | 7 |
| False | 1 | 1 |

Table 1 Outcome of the algorithm on the test set.

## V. Software Engineering

We organize our project into the source folder and dataset folders. The source code of our project is composed of three python files, namely Process_Img.py, Fit_Model.py, and Experiment.py. Each script is well commented and formatted.

- Process_Img.py implements the basic functionalities of the feature extraction. It provides functions such as image loading, edge detection, line detection, and vanishing point detection. It also includes a wrapper function that automatically wraps the features into a dictionary for ease of application.
- Fit_Model.py implements all the necessary functions that compute the cost. It provides functions that calculate the line cost, the vanishing point cost, the hash cost as well as the total cost. The prediction function also stays in Fit_Model.py.
- Experiment.py contains codes that are convenient to run the experiment. It loads the validation set and the test set as global variables. The function that finds the optimal threshold and the one that reports the final accuracy stay in Experiment.py.

## VI. Discussion and Future Work

During the process of developing the model-based vision system, we have tried several different approaches and made some adjustments along the way.

For instance, we initially tried a masking approach that did a logical AND operation on the input image with the model as the filter. Then we computed the ratio of the number of the pixels that passed the filter over the total number of pixels of the model itself. The performance was not encouraging as only a few pixels got through, and it was extremely sensitive to the position and orientation of the camera.

There is also something more that we want to experiment with but are limited by time and experience. Here we point out three future works that can be done to improve or extend the capability of the system.

### A. Multiclass Classification

In our study, we only experiment with the binary classification case, namely true or false. To go beyond this, we recommend the next group of researchers to test the algorithm on more scenes such as staircase, rooms, forests, streets and make corresponding adjustments to the current algorithm.

### B. Incorporate Camera Info

Our study deals with static images that are taken out of the context of the robot and camera configuration. Potential improvements can be made if such parameters are taken into consideration.

For example, one can take the camera position, orientation, and calibration information as knowns and utilize the information to transform the images so that their scale, translation, and rotations are aligned. We speculate that such transformations would improve the precision of the cost function as the lines and vanishing points will have approximately the same properties for the same scene.

### C. Object Detection and Classification

One of our initial ambitions is to apply the model-based approach to identify and classify both the scene and the objects inside it. To do so, one can typically look for objects by sliding a window of fixed size through the image and recursively call the algorithm. An alternative is applying image segmentation techniques that mark out those objects, then run the classifier on each one of them. Either way, the task is more complicated as the features become vaguer for smaller entities that generally have a lower resolution.

### D. Upgrade Feature Extraction and Cost Function

There is certainly much more space for improvements for feature extraction and cost function formulation. The current feature extraction techniques are very basic and sometimes intolerant to noises.

For example, the line detection algorithm always outputs some lines that do not belong to the structure of the model. Researchers can look into it and come up with a more robust and accurate method.

Additionally, one can also test and discover more features for comparison. One type of feature that we would like to try is the SIFT feature.

Besides the feature extraction, the cost function is also crucial in accurately reflecting the similarities between images. Our definition of the cost function is relatively coarse and lacks fine-tuning. Therefore, upgrading the cost function will likely improve the performance of the algorithm by a huge step.

## VII. Conclusion

We demonstrate that a model-based vision system is able to achieve high accuracy with a few model samples. Future works can focus on expanding the research on multiclass classification and object detection and classification. The system can also be enhanced by taking into consideration the camera information and upgrading the existing feature extraction functionalities and the cost function.

When the statistical learning algorithms are not suitable for robot scene recognition, the research on model-based vision techniques or the combination of both could open more possibilities of alternatives.

## References

[1] Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* 60, 91–110 (2004). https://doi.org/10.1023/B:VISI.0000029664.99615.94

[2] *Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. DOI:https://doi.org/10.1145/3065386*

[3] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[4] Krawetz, N. K. (2011, May 26). *Looks Like It - The Hacker Factor Blog*. The Hacker Factor Blog. http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html

[5] SZanlongo. (n.d.). SZanlongo/vanishing-point-detection. Retrieved December 19, 2020, from https://github.com/SZanlongo/vanishing-point-detection

[6] *Richard O. Duda and Peter E. Hart. 1972. Use of the Hough transformation to detect lines and curves in pictures. Commun. ACM 15, 1 (Jan. 1972), 11–15. DOI:https://doi.org/10.1145/361237.361242*