Robot Car Auto Navigation and Mapping for an Indoor Environment

ME 740 Project report

Weipeng Wang

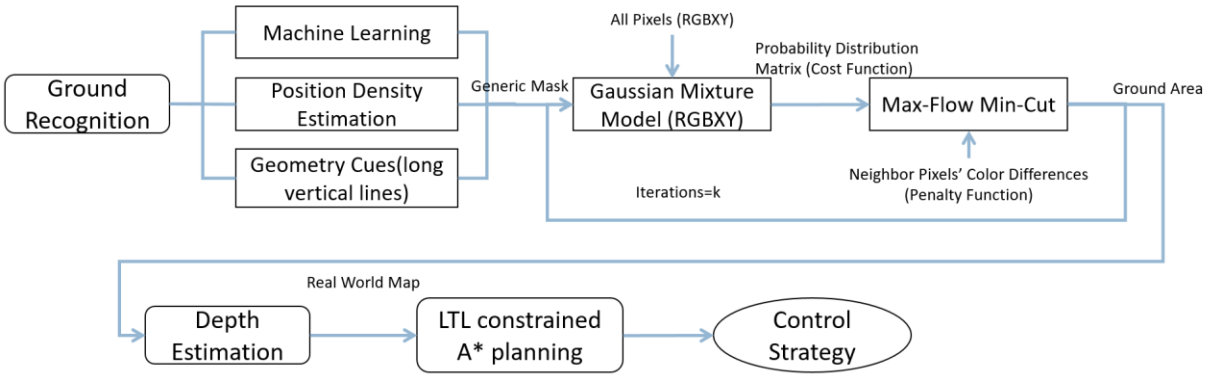
TABLE OF CONTENTS

INTRODUCTION

ROBOT CAR

(Figure 1) A rear wheel drive car called 'SunFounder Smart Video Car' which I bought on Amazon, with size 9.9 x 5.2 x 2.9 inches with a single camera at the front. The movement of the wheels is controlled by two motors and a servo motor. The camera has two degrees of freedom controlled by two servo motors. Using this camera, the project aims to guide the robot car to navigate in a rather complex indoor environment by processing the images taken by the camera.

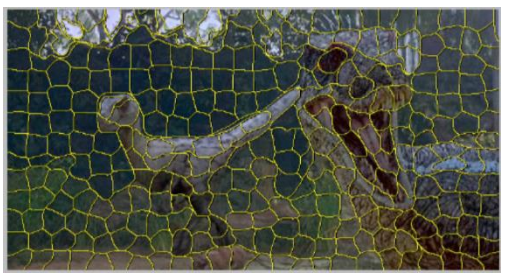
PROCESS OVERVIEW



The whole process can be roughly divided into 4 sections: Generate the generic mask, Smoothing and Optimizing, Depth Estimation and Path Planning. The first three all serve to give a very accurate map of the passable ground areas and the last one works to find an optimal path for the robot car to navigate and map the environment efficiently as well as generating control inputs for it. Detailed information of all the blocks in the process can be found in later sections.

SUPERPIXEL

As will be used as the basis of generating the generic mask, it is better to explain the idea of superpixel here.

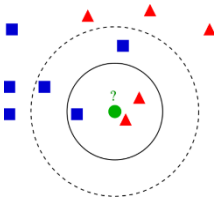


(Figure 2) Superpixel is a group of contagious pixels which carry similar information. It is a very good basis for many image processing problems since it reduces the 480*640 (say) pixels to hundreds of superpixels, yet not destroying the major information of the image. The algorithm called SLIC (Simple Linear Iterative Clustering) [1] used in this project adapts a k-means clustering approach to efficiently generate superpixels.

SECTION 1: GENERATE GENERIC MASK

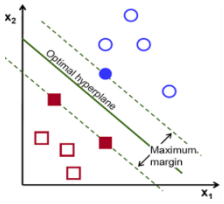
MACHINE LEARNING

Machine learning is a popular method nowadays, which collects certain amount of data based on the feature vectors defined by the designer (dataset), train this dataset to get a model of classifier. For example, the simplest example using K-Nearest Neighbors (KNN) model is show in the following image:

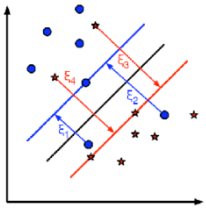


(Figure 2)To classify the green circle to be in the class of red triangles or the class of blue rectangles, the most obvious way is to compute the distance to samples in the two classes and pick the closet one. KNN chooses the closest k neighbors and compare the sum of distances in the two classes. The smaller the sum is, the more possible the green circle belongs to that class.

In this project, a calibrated Support Vector Machine is used as the model and below is a introduction of Support Vector Machine (SVM):



(Figure 3)The basic idea of a SVM is trying to draw a straight line (or hyperplane) with largest minimum distance to the training samples and separate them into two regions.



(Figure 4) Of course there will be some misclassifications and there are methods to optimize this problem, which is not in the scope of this report. The project uses a calibrated SVM model which is able to give a probability of classification rather than directly classify a testing sample.

Dataset and Learning Model Preparations

300 images of indoor environment were collected from the MIT LabelMe public collections and their ground areas are manually labeled as ground (1) and not ground (0) area. Just for the record here, a more precise training image collection is possible through taking photos from the robot car's camera of different indoor environments, though it will take a lot of time and efforts. These training images are segmented into SLIC superpixels [1] first, where every superpixel is a group of adjacent pixels containing similar information. Each image will have around 300 superpixels. Compared to 480*640 pixels in each image, it will dramatically reduce the time of training the model (details in introduction).

A feature vector is defined for each superpixel with a combination of human-defined features shown as normalized scalar numbers. These features are:

- Color: The average RGB color (3 cues) and the average HSV color (3 cues)
- Hue: The 5-bins histogram and entropy (6 cues), the 3-bins histogram and entropy (4 cues)
- Texture: Mean abs responses of 12 Gabor filters with their mean value, max value and (max-min) value (15 cues)
- Position and Shape: The 10th, 50th, 90th percentile of the pixels' location (x and y) and mean value of the locations in a superpixel (8 cues), the number of pixels (1 cue) and the shape as the

difference between the length and width of a superpixel (1 cue, for estimating the shape of a superpixel, which is more similar to a square at large area of ground)

The dataset, essentially, is a matrix of all the feature vectors, where each row represents a feature vector of a superpixel. Feed this dataset to the calibrated SVM, a model of classifier can be created, which is ready to be used by input feature vectors of a test image.

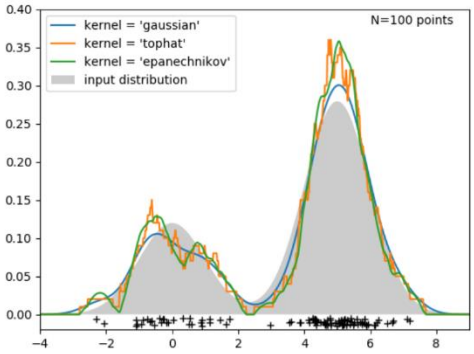
Working Process and Output

Apply the same process to the test image, get SLIC segmentation and feature vectors. The model will output a normalized score for every superpixel in the test image indicating how likely it is to be ground. Call it $Wa\_score(sp)$.

POSITION KERNEL DENSITY ESTIMATION

Position is an important clue for the robot car to identify the ground area. It should be a 'common sense' that the ground area in your sight is normally at the lower part of the scene, even to a machine. Although the machine learning algorithm already has the position cues, it is worth to look at the position separately.

So, where does this 'common sense' comes from? The answer is memory and frequency.



(Figure 5) The position of the ground area in a single one image does not mean anything, but when it becomes 300 images or more, it starts to be representative. The 300 training images' superpixels' positions (mean, 10th, 50th, 90th percentile xy values) that belong to the ground area are fit into the kernel (Gaussian) density estimation [2][3] model, which, with any sample superpixel positions input into the model, there is going to be a possibility for it to be ground.

The kernel density estimation (KDE) method uses a neighbor-based approach to construct a probability density map of the two-dimensional position locations of floor regions based on the trained dataset. The kernels from the KDE function corresponds to discretized regions of the test image. KDE uses a non-parametric approach to smooth the density function by not only considering locations where events occur, but by assigning a density distribution to each event based on the kernel density function. In this context an event is defined as a region defined as floor from the training dataset. The equation for the KDE distribution at each kernel on the image plane is shown below:

$$\rho K(y)=\sum_{i=1}^{N} K(\frac{y-x_i}{h})$$

Where $\rho K(y)$ is the probability of floor at a kernel, y, based on the general density function, K, h is the bandwidth parameter and x_i is the location where an event occurs.

The kernel function is scaled by a bandwidth parameter which determines kernel size. Bins from histogram density estimation are congruent to the kernel size in KDE estimation. A reasonable value for the bandwidth parameters is determined by testing different bandwidths and choosing one that corresponded to smooth and expected results. Optimizing the bandwidth parameter by minimizing the mean integrated square error is left for future work.

As standard of density distributions, the sum of the area under the kernel density function for each event is equal to one. A gaussian distribution fits the kernel density function. Since the Gaussian distribution is unbounded and the test image area is bounded, some probability data is lost to areas outside the bounds of the test image. The kernel density function for the Gaussian distribution is shown below and used simultaneously with the KDE distribution equation to construct a probabilistic map of the test image floor region:

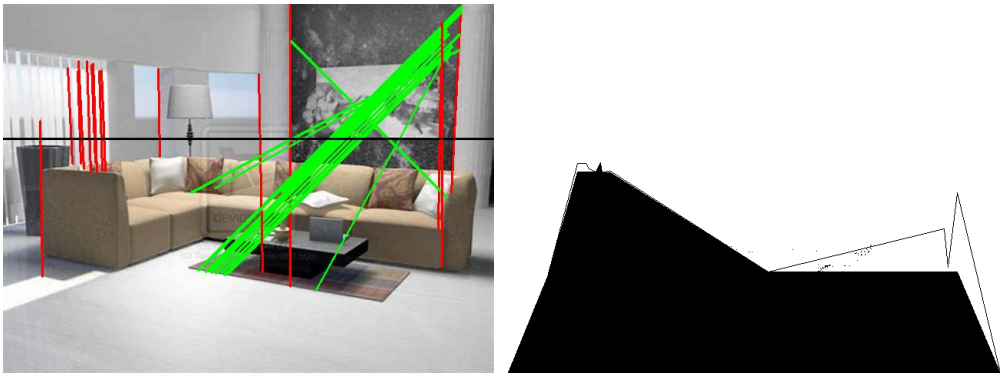
$$K(x;h) \propto \exp(-\frac{x^2}{2h^2})$$

Process and Output

Input all the superpixels' position cues (mean, 10th, 50th, 90th percentile) of the test image into the model and get a normalized score (possibility) for each of them, divide their summation by 4. Call this output $Wb\_score(sp)$.

GEOMETRY CUES (LONG VERTICAL LINES)

In a typical indoor image, objects on the floor are normally perpendicular to the ground plane and their edges are normally straight lines. By finding the long vertical edges' lower endpoints, a set of points on the ground plane can be obtained, which is important reference for finding the ground plane. So, how did I do this specifically? I used Canny Edge detection [4] to extract edges in the test image, applied Probabilistic Hough Transform [5] to get the long lines with length larger than 70 pixel length. These lines were classified into vertical and inclined lines, with corresponding slopes 0-5 and 20-65 degrees. By getting the average y values (see as a threshold line) of midpoints of all the inclined lines and filtering those vertical lines which have most of the length above the threshold line, the set of vertices of a mask for the rough estimation of the ground area is available. Below are two images showing the idea:



(Figure 6 and 7): The black line is the threshold line, the green lines are inclined lines, the red lines are vertical lines. The right image shows the mask by connecting the valid endpoints.

Output

To make the output aligned with the two outputs above in superpixel level, the formula below is used:

$$Wc\_score(sp) = \frac{Num\ of\ pixels\ in\ superpixel\ in\ mask}{Total\ num\ of\ pixels\ in\ superpixel}$$

If all the pixels in a superpixel belongs to the mask, the score is 1, if none, 0, if half, 0.5.

GENERIC MASK

By combining the above three output by giving different importance to them and set a threshold of the result, we can have the candidate superpixels that are decided by the system to be ground.

$$P(generic) = a * Wa\_score + b * W\text{b}\_score + c * W\text{c}\_score \quad (\text{a} + \text{b} + \text{c} = 1)$$

SECTION 2: SMOOTHING AND OPTIMIZING

GAUSSIAN MIXTURE MODEL

Mixture models are a composition of a finite number of independently distributed subpopulations. A Gaussian Mixture Model (GMM) is a population consisting of a set of a finite gaussian distributions. When a multivariate GMM is considered, the model may not appear Gaussian, it may rather appear as a combination of the number of n Gaussian distributed subpopulations with n peaks assuming the peaks lie sufficiently far apart. The equation for a realization of a GMM is established using the equation below to compute the probability density of the Gaussian mixture distribution.

$$F(x) = \sum_{i=1}^{n} w_i P_i$$

Where P is the corresponding probability distribution of each subpopulation's distribution and w is the weight of each distribution in the GMM such that $\sum_{i=1}^{n} w_i = 1$.

A realization of a mixture distribution is deterministically obtainable from a fully defined set of subpopulations which comprise the mixture distribution, however, obtaining the parameters of a subpopulation set from a mixture distribution requires the use of statistical inferences. The reasoning behind using statistical inferences to partition multivariant mixture models into subpopulations is that in these such mixture models, data points from multiple subpopulations may overlap, making it difficult to deterministically compute the individual distributions. An iterative approach to statistics is a mathematically and computationally reasonable method of determining parameters of subpopulations, such as mean and covariance, which fully define the distributions of the subpopulation given a realization of a mixture distribution.

Unsupervised clustering of trained data provides accurate values of the parameters of the Gaussian distributed subpopulation within a GMM. One common method of clustering data is the k-means clustering method which assigns data points to clusters via hard partitioning. Although k-means is a valid method for determining spherical gaussian subpopulations from multivariant GMM's, it is not an effective method of partitioning multivariant GMMs with subpopulations with variance and covariance. In contrast, the expectation maximum (EM) clustering method uses probabilistic values to partition data points and enables variance and covariance parameter computation of gaussian distributions in addition to centroidal parameter.

The EM algorithm process estimates parameters of Gaussian distribution from univariate GMMs as they converge asymptotically over a series of iteration. Before the iteration process begins, initial values of the subpopulation parameters are initialized. A common method of initializing subpopulation parameters for the EM algorithm is to us the k-means method of assigning data to clusters using distance in Euclidian space to determine a realization of k spherical Gaussian distributions. Although not as accurate of a method of initializing as k-means, the EM algorithm will also converge effectively by initializing all subpopulation normally distributed about the origin. Once the initialized distributions are set, the probability of each data point belonging the initialized distributions is calculated deterministically. The set of data points with probabilities from the initialized Gaussian distributions is the initialized posterior distribution. With the prior knowledge of posterior distributions, the EM algorithm uses the maximum a posterior (MAP) method to estimate the centroidal and covariance parameters. The equation below manifests the MAP estimation for a given parameter.

$$\hat{\theta}_{MAP}(X) = \arg\max_{\theta} f(\theta|X)$$

Where $\hat{\theta}_{MAP}(X)$ is the distribution of the parameter of interest and $f(\theta|X)$ is the posterior distribution of the parameter given a dataset X.

MAP estimation gives a new, more accurate representation of the subpopulations. The posterior distribution of the data from the results of the first MAP estimate is calculated so another MAP estimation can enhance the results of the subpopulation distributions further. Repeating the process of determining the MAP estimate and calculating the posterior distribution iterates until the parameters of the distributions converge asymptotically at which point the EM algorithm is complete.

Apply GMM to Image Processing

From Section1, the generic mask is given by a set of candidate ground superpixels that have higher score than a threshold. The RGBXY values of pixels inside this mask are combined to form a GMM. The algorithm for creating a GMM from a scored dataset of a discretized image is shown below.

1. Set a threshold value, T.
2. For each data point, D $\longrightarrow$ If $D_i > T$ set $D_i \varepsilon\, S$, else discard D_i.
3. Plot RGBXY values of pixels within D_i to the GMM.

Where S is the set of data points which have a high probability of being ground.

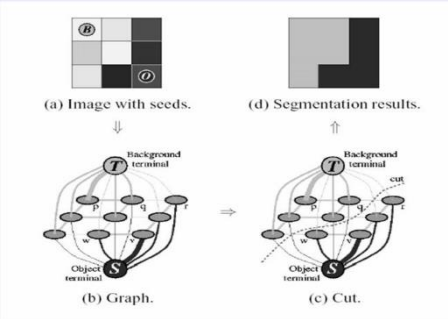
Super pixels are considered when comprising the training dataset because it speeds up processing time over a very large set of data. When analyzing the ground regions of a single test image individual pixels are analyzed because it is important not to compromise the accuracy of the results. The EM algorithm is

employed to estimate the parameters of the GMM for an image with resolution 480*640 at the pixel level. Then a test image is broken into pixels and the probability of each pixel being ground is assigned a score based on the kernel density split GMM, where all the scores form a 480*640 matrix and this matrix is further input to the Max-Flow Min-Cut algorithm to get a final optimization. Also, using the scores, a threshold can beset and the pixels above the threshold can be marked, creating a mask of the ground area for now. The results of the ground detection software up until this point are shown in the results section.

MAXIMUM FLOW MINIMUM CUT

A maximum flow minimum cut theorem [6] as an optimization problem. Incorporating a source and a sink into a Markov Random Field (set of random variables following Markov Property, here are the pixels) creates a flow network. Much like in a fluids flow analogy, assuming the model of the flow network is at equilibrium, the flow from the source will equal the flow to the sink over any given time. The purpose of the maximum flow minimum cut algorithm is to cut a path in the flow network that will maximize the difference in flows between the two sections created by the cut. More precisely the minimum cut will maximize the flow in at one section and maximize the flow out in the other section.

The following image shows the idea of the algorithm:



(Figure 8) Each pixel is connected to the source (representing the ground) and sink (opposite) node, as well as the 4 neighboring pixels. The strength of connection between pixels and the source and sink node is given by the result from the GMM. The strength of connection between neighbouring pixels is represented by the RGB color difference, named penalty function, written in formula as:

$$V = \frac{k}{Z * e^{2Z}}$$

Where Z is the normalized color distance and k is a scaling factor

High values for GMM probabilities are strongly connected to the source and likewise low values of GMM probability are strongly connected to the source. These strong connections create high flow values for these pixels which, will result in a highly unlikely choice of cut contingent on the penalty function not dominating the GMM probability function. Each pixel's flow due to the penalty function is determined by considering all the penalties that connect to the pixels, or in the context of the MRF all the edges that connect the nodes. Since the penalty function is inversely related to the scalar distance, values with high low color gradient will be assigned a high penalty and will not be likely to be cut. The flow due to the GMM probability and the penalty function are summed together to create an energy function. This summation is shown below.

$$E_{ij} = V_{ij} + P_{ij}$$

Where V_{ij} is the sum of the connected penalty values at each pixel and P_{ij} is the GMM probability of each pixel.

The algorithm works to cut through the edges with the minimum sum of energy, which can be seen as the most possible border between the ground and non-ground areas.

The iterative process as is shown in the flowchart in Introduction Section, means feeding back the result from the Max Flow Min Cut algorithm to the GMM, which gives a better estimation of the probability distribution of the ground area and then cut again. The iterations k can be set manually and the result should eventually converge with sufficient number of iterations

Section 3: Depth Estimation

Optical Flow

This report gives a rough estimation of the depth information of a test image based on the absolute value of optical flow of different regions in it.

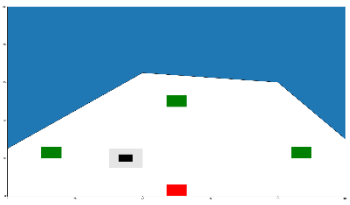
Optical Flow denotes the apparent motion of the objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. By applying Gunner Farneback's algorithm [7], it is possible to get the optical flow for all the points in the frame. Consider it as a simple triangulation problem, when moving the camera, the farer the object is, the larger displacement between two consecutive images and the larger the optical flow value. Although there is distortion issues of images, if we see the optical flow value of a point 1 meter away from the camera as a standard, we can get the distance between the boundaries of ground area and the camera in meters. Since the edges between the ground and non-ground area normally have a fairly large color difference, it is rare that one cannot get any optical flow value near the boundaries.

Section 4: LTL Constrained A Planning*

Linear Temporal Logic (LTL)

In path planning, linear temporal logic is expressed as a formulae, which encodes the conditions for a path to satisfy based on the time line. For example, the eventually operator '<>', shows that in the future, the path must meet the following condition. Other operators include always ('[]'), until ('U'), next ('N'), and ('&&'), or ('||'), negation ('!'), etc.

For the project, assuming the following map is given:



(Figure 9) The red square is the start area, the green squares are the areas for the robot car to go to, the blue area is non-ground area and the black area is the obstacle. Call the start area 's', the goal areas 'g1', 'g2', 'g3', the blue and black areas 'obs'.

It is wanted for the robot car to go to goal 1, 2 and 3 to have a further look at the environment make sure there is no ground areas left undetected. After been to the three goal areas, the robot car always wants to go back to the starting point and starting to navigate the area on the opposite side of the

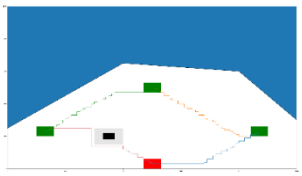
current map. While navigating, the obstacles always need to be avoided. So, the following formulae is defined:

[]<>s && <>g1 && <>g2 && <>g3 && []!obs

A* SEARCH ALGORITHM

Imagine a robot in an unknown environment. It has no idea of where the obstacles are but have the location of the goal. What is then a good method to find an optimal path from current location to the goal? A* algorithm, is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that appear to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node [8].

Given the map as is in Figure 8, make a 100*100 grid containing 10000 nodes to run the A* algorithm. The algorithm can find optimal path between any two squares, result shown as follow:

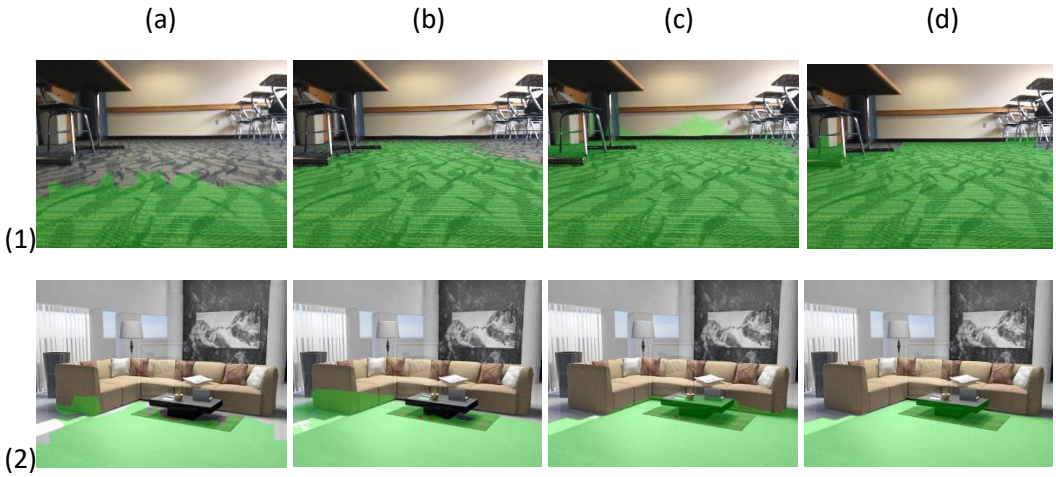


(Figure 10) The path given by A*, in the form of movement in a grid. In the result section later, by defining the robot car's kinematics, a better and feasible path will be given.

RESULTS

GROUND RECOGNITION IN DIFFERENT STAGES

To trade off between the optimal result and the computing speed, the iterations of running GMM and Max-Flow Min-Cut is set to be 2.



(3) 

(4) 

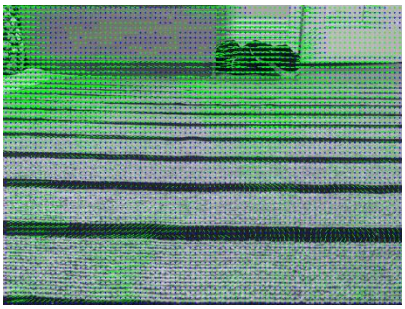
(Figure 11) (a): The generic mask; (b): The first GMM result; (c): The second GMM result after passing through the Max-Flow Min-Cut Algorithm once; (d): The final result by applying Max-Flow Min-Cut algorithm the second time on the mask from (c)

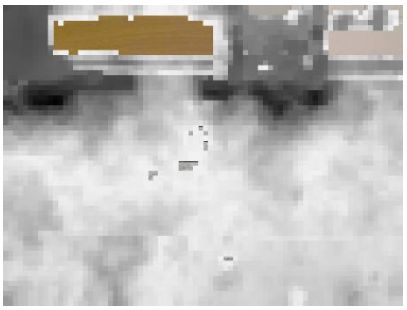
The program was running on my laptop with an Intel i7-6650U@2.20GHz CPU and the following table is the processing time spent:

	Image (1)	Image (2)	Image (3)	Image (4)
Generic Mask	15.58s	15.14s	15.49s	15.85s
Smoothing	18.69s	16.31s	10.89s	14.04s
Total	34.27s	31.45s	26.38s	29.89s

DEPTH ESTIMATION

Works are still needed to analyze the depth information, as it is a huge topic in image processing. Here is just a simple result showing the idea described in previous section. The camera (of the robot car for this example) takes one picture at the beginning, and moves to the right for 10cm and takes the second image, the optical flow is calculated based on the two images. Compare the result with the standard optical flow described in the 'Optical Flow' section, the depth value in meters can be given. Shown here in Figure 12, the darker for the color, the deeper for the distance.





(Figure 12 and 13) The optical flow shown by the vectors and depth estimation in darkness of the color. Notice that there are some regions without color differences, and so they do not have obvious optical flow.

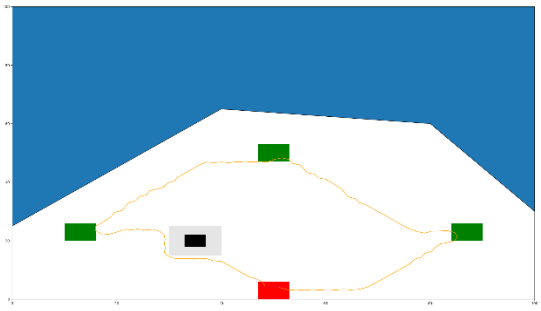
PATH PLANNING

The robot car's kinematics is defined as following:

$$\begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \varphi_{t+1} \end{pmatrix} = \begin{pmatrix} x_t \\ y_t \\ \varphi_t \end{pmatrix} + \begin{pmatrix} \tau v \cos\theta \\ \tau v \sin\theta \\ \vartheta \end{pmatrix} + uncertainty$$

Where it has a constant speed v, sampling period τ, head direction φ, turning angle ϑ and some uncertainty of the movement or environment.

Let the robot car, under the constraint of the kinematics, track the path given by the LTL constrained A* algorithm.



(Figure 14) The final path given by the path planning algorithm with control input given to the robot car every sampling period

CONCLUSION AND FUTURE WORK

This project is not finished in some ways, because the depth estimation just gives a rough result and further analysis is needed to give the real world map, and of course the ultimate goal has to be run the whole thing on the robot car. However, the most important and hardest part, recognizing the ground area in an image is finished and relative good results are given. With a good estimation of the depth estimation, together with the path planning algorithm, it is very easy to put everything onto the robot car and let it navigate the environment automatically while building the map of it.

There are several possible future works for the followers who are interested in this project to do:

- The balance between the cost function and penalty function (the scale factor for the penalty function essentially) can be improved by an adaptive algorithm to decide the scale factor K for different environment so that the Max-Flow Min-Cut will not cut at the wrong places.
- A better data set is possible if someone is willing to take images of different indoor environment from the robot car's viewpoint.

- A reinforcement learning is possible by feeding the testing results back into the training model. After certain iterations, the machine will have a very good knowledge what the ground in this certain environment should be like.
- Optimize the code or some of the algorithm to make it run faster. Now it takes about 30s on my computer and 2 times more on raspberry pi. If anyone can reduce that time to 0-10s, it will be a great auto robot product.

All the code and training models are on my GitHub Page, take a look if interested:

https://github.com/Robertwwp/Ground-Recognition

REFERENCES

Mentioned in this Report:

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274-2282, Nov. 2012.

[2] Rosenblatt, Murray. “Remarks on Some Nonparametric Estimates of a Density Function.” The Annals of Mathematical Statistics, vol. 27, no. 3, 1956, pp. 832–837.

[3] Parzen, Emanuel. “On Estimation of a Probability Density Function and Mode.” Ann. Math. Statist., vol. 33, no. 3, 1962, pp. 1065–1076.

[4] Canny, John. “A Computational Approach to Edge Detection.” Pattern Analysis and Machine Intelligence, IEEE Transactions On, PAMI-8, no. 6, 1986, pp. 679–698.

[5] Hough, P.V.C. Method and means for recognizing complex patterns, U.S. Patent 3,069,654, Dec. 18, 1962

[6] G. B. Dantzig and D. R. Fulkerson, “On the Max-Flow MinCut Theorem of Networks,” in “Linear Inequalities,” Ann. Math. Studies, no. 38, Princeton, New Jersey, 1956.

[7] Farnebäck, Gunnar. “Two-Frame Motion Estimation Based on Polynomial Expansion.” Scia13, 2003, pp. 363–370.

[8] Hart, Peter, et al. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths.” Systems Science and Cybernetics, IEEE Transactions On, vol. 4, no. 2, 1968, pp. 100–107.

Some Great Papers I Followed:

[9] Hoiem, D., et al. “Geometric Context from a Single Image.” Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference On, vol. 1, 2005, pp. 654–661.

[10] Aggarwal, Sanchit, et al. “Estimating Floor Regions in Cluttered Indoor Scenes from First Person Camera View.” Pattern Recognition (ICPR), 2014 22nd International Conference On, 2014, pp. 4275–4280.

[11] Fu, Jie, et al. “Optimal Temporal Logic Planning in Probabilistic Semantic Maps.” 2015.