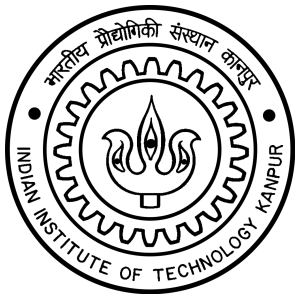
EE392A

Intelligent ground vehicle motion planning using demand-based mapping

Shubh Gupta (14670)

under the guidance of

Prof. Mangal Kothari, AE
Prof. S. R. Sahoo, EE



Indian Institute of Technology, Kanpur
Undergraduate Project Report

Contents

1 Introduction **1**

2 Motivation **1**

3 Problem Formulation **2**

4 Fundamentals **3**
4.1 Sensor modelling 3
4.2 Ground to image projection 4

5 Methodology **6**
5.1 Classifier initialization 6
5.1.1 Clustering 7
5.1.2 Morphological operations 7
5.1.3 Patch generation 8
5.2 Motion Planning 8
5.2.1 Dijkstra's Algorithm 8
5.2.2 A* Algorithm 9
5.3 Mapping 9
5.4 Classifier update 10

6 Experiments and Results **11**
6.1 Data 11
6.2 Experimental Setup 11

7 Results **12**

8 Conclusion and Future work **13**

9 Acknowledgement **13**

List of Figures

1 Lane image with patch level and pixel level ground truth 1
2 Patch-wise confidence-measurements 3
3 Camera Projection model 5
4 Camera Calibration using chessboard 6

5 Data generation 7
6 Input image 12
7 Dijkstra's Algorithm 12
8 A* Algorithm 12
9 Object detection using LiDAR 13

Abstract

The project aims at developing a pipeline for combining motion planning and ground-plane semantic mapping, with both assisting each other. Map generation queries are performed on image patches using online learning classifiers to accommodate for concept drift. The implementation is done in ROS framework, with code parts in both C++ and python.

1 Introduction

Motion planning problem is to produce a continuous path that connects a start configuration and a goal configuration while avoiding passing over obstacles. Since there might exit multiple of such paths, algorithms for motion planning aim to optimize towards a certain policy, such as minimizing the path length or maximizing information gain. These algorithms require a configuration space, or a set of all possible robot configurations in the environment, from which it can choose a set of reachable configurations as a part of the path.

This configuration space is usually determined via mapping of the environment through sensors, such as a camera or a LiDAR, in an online or offline manner. The aim of this project is to couple mapping and motion planning together, in order to generate a path faster, as well as to help the map generation become more robust.

2 Motivation

Figure 1: Lane image with patch level and pixel level ground truth

Motion planning is an integral part of developing Intelligent ground robots, since they are expected to autonomously make decisions regarding navigating the

environment without any controller intervention. Generation of motion planning path is usually preceded by a mapping procedure, and thus the time taken for path generation is the sum of the times taken for both of these procedures. Since ground robots have to run in realtime, the rate of map generation is a bottleneck.

In comparison to classical image processing methods, machine learning based methods have become the popular choice for robot vision and perception due to their success in modelling non-linear properties and incorporating latent variables. These methods, however, require a lot of input-output data which might not be easy to obtain for the given environment. Hence, there is a need to improve the performance of systems which can work with lower levels of supervision, which is easier to obtain. (Figure 1)

Furthermore, in real world robot problems, the state of the environment may not be consistent throughout the robot execution, and may change due to lighting changes, shadow, or simply geographical reasons. Hence, online learning algorithms are the method of choice as they are able to adapt to these changes. Online learning algorithms also fit in very well with the aforementioned requirement of needing very less initial data, and can pick up and learn from the more data acquired as the robot moves. However, the challenge here is to be able to provide correct and useful data to the algorithm during the run. This undergraduate project aims to address these requirements by combining online learning, mapping and motion planning into a single pipeline seamlessly.

3 Problem Formulation

The camera is mounted on a ground robot and takes images with sufficient frames per second such that no point on the ground fails to be captured because of the duration between two consecutive frames. Let the set of images be denoted by $I_0, I_1, ...I_n$. Let $G = [G_1, G_2, ..G_n]$ be an n-voxel grid overlaid over the ground plane in 3D world, where $G_l \in \mathbb{R}^2$ is a 2D point representing the center of l^{th} voxel of the grid. We define occupancy map $m = [m_1, m_2, ..m_n]$ as a set of values signifying the occupancy status of the voxel. Hence, $m_i \in [0, 1]$ is 0 when a voxel is a sure traversable point and 1 when voxel is a sure non-traversable point, with values in between specifying various degrees of confidence between these two extremities. The aim is to estimate map m and simultaneously obtain paths from each frame $\pi^k = [\pi_1^k, \pi_2^k ... \pi_n^k]$ such that π_i^k and π_{i+1}^k are adjacent and $\pi_i^k \subset G$. Furthermore, let F be the projection function which projects pixels in the current frame to ground

plane voxels, such that $F(I_k) \subset G$ giving us a tighter constraint on planner path $\pi^k \subset span(F(I_k))$.

By the nature of definition of m_i, it can be interpreted as the probability of a voxel being non-traversable, with low probability values denoting higher probability of the voxel being traversable since these cases are mutually exclusive. We use this value to calculate the cost c_i of traversing voxel i such that c_i is a function of m, and use this for generation of the planner path by minimizing the total cost over a path. Hence, optimal path is given by

$$\pi^{k*} = \underset{\Pi^k}{\operatorname{argmin}} \sum_{P \in \pi^k} c_{arg(P)} \tag{1}$$

4 Fundamentals

4.1 Sensor modelling

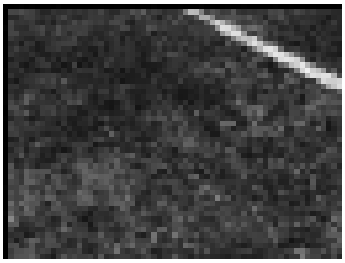


Figure 2: Patch-wise confidence-measurements

Value of m_i for a voxel G_i can be determined using an online learning based feature classifier by treating the classifier as a pseudo-sensor providing measurement z_k^i and using the classification confidence as the measurement belief $p(m_i|z_k^i, t_k)$, given a camera configuration during k^{th} frame as t_k. We denote a set of past measurements for i^{th} voxel as $z_{0:k}^i = [z_0^i, z_1^i, ...z_k^i]$ and further for all voxels $z_{0:k} = [z_{0:k}^0, z_{0:k}^1, ...z_{0:k}^n]$ and similarly for camera configuration $t_{0:k} = [t_0, t_1, ...t_k]$ to give a compact equation for belief b_k^m on map m [1]

$$b_k^m = p(m|z_{0:k}, t_{0:k}) \tag{2}$$

Collecting marginals based on the assumption that voxel pdfs and measurements are independent,

$$b_k^m \equiv (b_k^{m^i})_{i=1}^n, \qquad b_k^{m^i} = p(m^i|z_{0:k}^i, t_{0:k}) \tag{3}$$

Applying Bayes rule,

$$\begin{aligned} b_k^{m^i} &= p(m^i|z^i_{0:k}, t_{0:k}) \\ &= \frac{p(z^i_k|m^i, z^i_{0:k-1}, t_{0:k})p(m^i|z^i_{0:k-1}, t_{0:k})}{p(z^i_k|z^i_{0:k-1}, t_{0:k})} \end{aligned} \tag{4}$$

Furthermore, by assuming measurement independence on history,

$$p(z^i_k|m^i, z^i_{0:k-1}, t_{0:k}) \approx p(z^i_k|m^i, t_k) \tag{5}$$

Applying Bayes rule to this equation gives us the inverse sensor model

$$p(z^i_k|m^i, t_k) = \frac{p(m^i|z^i_k, t_k)p(z^i_k|t_k)}{p(m^i|t_k)} \tag{6}$$

Plugging into (4), we get

$$b_k^{m^i} = \frac{p(m^i|z^i_k, t_k)p(z^i_k|t_k)p(m^i|z^i_{0:k-1}, t_{0:k})}{p(m^i|t_k)p(z^i_k|z^i_{0:k-1}, t_{0:k})} \tag{7}$$

which can be simplified further by the assumption that camera position does not affect actual map

$$b_k^{m^i} = \frac{p(m^i|z^i_k, t_k)p(z^i_k|t_k)p(m^i|z^i_{0:k-1}, t_{0:k})}{p(m^i)p(z^i_k|z^i_{0:k-1}, t_{0:k})} \tag{8}$$

4.2 Ground to image projection

We use the perspective camera model to map 3D points in universe to 2D points in camera pixels.[2] Figure 3 shows the pinhole model of a camera. O is the center of projection of camera and the image plane is always at a distance of focal length f from O. A point in 3D $P(X, Y, Z)$ (in cameras frame) is viewed in image plane as $Pc(u, v)$ (here we assume that image plane origin O' is the point where principal axis intersects image plane). By the property of similar triangles,

$$f/Z = u/x = v/y, \qquad u = fX/Z \quad v = fY/Z \tag{9}$$

The above equations can be represented in homogeneous coordinates as

$$P_h = \begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = KP \tag{10}$$

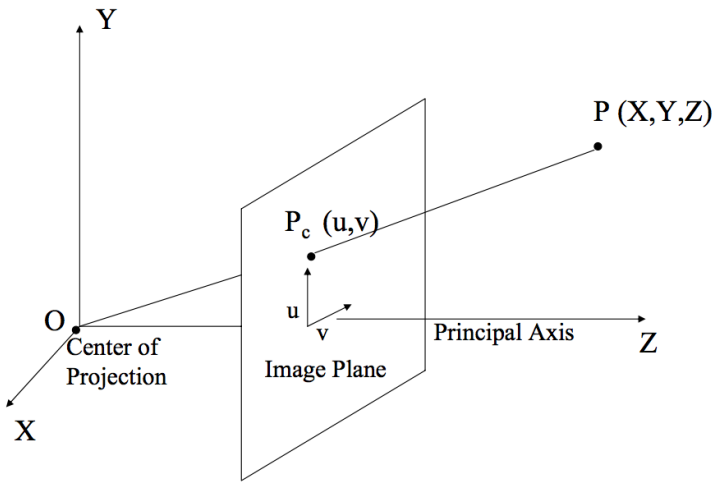


Figure 3: Camera Projection model

src: https://prateekvjoshi.com/2014/05/31/understanding-camera-calibration/

$$P_c = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} u'/w \\ v'/w \end{bmatrix} \tag{11}$$

Now, incorporating for when the image plane origin O' does not coincide with the point of intersection of principal axis with image plane, P_c can be adjusted as

$$P_c = \begin{bmatrix} fX/Z + c_x \\ fY/Z + c_y \end{bmatrix} \tag{12}$$

where (c_x, c_y) is the coordinates of the intersection point in the frame of image plane origin O'. Incorporating for conversions from MKS to standard systems with multiplicative factors (m_x, m_y),

$$P_h = \begin{bmatrix} u' \\ v' \\ w \end{bmatrix} = \begin{bmatrix} m_x f & 0 & m_x c_x \\ 0 & m_y f & m_y c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = KP \tag{13}$$

Since the cameras coordinate frame is usually not aligned with vehicles coordinate frame then we have to perform rotation and translation to align the above two frames as well.

$$P_h = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} m_x f & 0 & m_x c_x \\ 0 & m_y f & m_y c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{14}$$

Given a set of object points and their image correspondences, it is possible to solve for the intrinsic and extrinsic parameters. This process is called camera calibration. A key issue in calibration is to establish object to image point correspondences, that is to identify the point on the image corresponding to a certain point on the object, whose position is known in the object frame of reference. To simplify this process, it is common practice to choose objects with a well defined pattern, such as a chessboard (Figure 4).

Figure 4: Camera Calibration using chessboard

5 Methodology

The input to the system is an image feed through onboard camera on a ground robot, along with the robot's position as well as the camera's position with respect to the robot. An outline of the algorithm is as follows:

- **Classifier initialization:** Train a classifier with initial estimates of traversable and non-traversable paths.
- **Motion planning:** Generate queries of specific gridcells for map updation and cost determination to plan a path.
- **Mapping:** Update the map voxel beliefs based on classifier response.
- **Classifier update:** Update the classifier to incorporate for concept drifts.

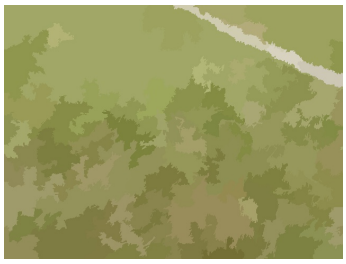
5.1 Classifier initialization

We begin with taking few initial images of the environment and autonomously generating a dataset of image patches from them to train a classifier. To that end, we use the following pipeline:

- **Clustering**

- **Morphological operations**
- **Patch generation**

5.1.1 Clustering



(a) Mean Shift Clustering

(b) Lane extraction

Figure 5: Data generation

Due to the presence of noise in natural images, we cannot directly divide the image into patches belonging to traversable and non-traversable regions by means of thresholding. Hence, we apply clustering on the image to combine regions belonging to same region. The clustering algorithm used here is Mean-Shift Clustering [3] for its simplicity, robustness as well as the ability to incorporate for a variable number of clusters. The algorithm builds upon the concept of kernel density estimation (KDE), and works by trying to find modes in a distribution by placing a kernel (weighting function) on each point in the data set. The kernel used in this project is a Gaussian kernel. The translation $m(x)$ for every kernel window for each iteration can be calculated by

$$m(x) = \frac{\sum_{i=1}^{n} x_i g(\|x - x_i\|^2/h)}{\sum_{i=1}^{n} g(\|x - x_i\|^2/h)} - x \tag{15}$$

where x_i is the i^{th} point in distribution and x is the kernel mean. g is a gaussian distribution with mean 0 and variance 1.

5.1.2 Morphological operations

The non-traversable region is determined by applying threshold on the clusters. A threshold interval $[t^k_{min}, t^k_{max}]$ is determined for each image using

$$t^k_{min} = 0.7 \max_{i,j}(I^k), \quad t^k_{max} = \max_{i,j}(I^k) \tag{16}$$

where I^k is the k^{th} image being used for data generation. To incorporate for noise generated during thresholding, morphological operation [4] of closing is performed on the mask of the segmented region to fill in gaps, followed by erosion to make the boundaries tight in order to make the data generation more accurate.

For generating data of traversable region, an opening operation is performed followed by dilation on thersholded region, and the mask is inverted at the end (Figure 5(b)).

5.1.3 Patch generation

Sliding window approach is used for generating patches from the obtained masks. A square window of 10X10 pixels is moved over the masked image and the patches with all non-zero pixel values are appended to the corresponding dataset.

This data hence obtained from the first few frames is used to initially train a feature classifier in an offline manner. The dataset used in the project contained 50,000 samples randomly chosen from from 10 images with a 50:50 ratio of traversable and non-traversable samples.

5.2 Motion Planning

The objective of motion planning is to generate a continuous motion that connect a start configuration S to a goal configuration G while avoiding obstacles. In this project, the motion planning cost-map is generated dynamically on the basis of motion planner queries of specific voxels. Due to this, it is difficult to pinpoint a valid G beforehand which is reachable from S. To overcome this issue, goal configuration is taken as a circle centered at a point at a fixed distance away in the direction of the motion of the ground robot. Hence, the planning algorithm generates a path by connecting S to any reachable point belonging to the circle of G, generating a path for moving forward. In this project, two grid-based search algorithms are experimented with,

- **Dijkstra's Algorithm**
- **A* Algorithm**

5.2.1 Dijkstra's Algorithm

Dijkstra's Algorithm [5] is a systematic search algorithm that finds feasible plans by finding single-source shortest paths in a graph, and is a special form of dynamic

programming. The algorithm works by growing a graph from the start configuration S to the goal configuration G. A list of nodes is maintained, and the distance value of each node n_i is calculated as the shortest traversal cost sum from S to n_i V_i as a solution to the local minimization problem

$$V_i = \underset{\alpha_j}{\operatorname{argmin}}(\sum_{j=0}^{n} \alpha_j V_j + \|x_i - \sum_{j=0}^{n} \alpha_j x_j\|) \tag{17}$$

where $\sum_{j=0}^{n} \alpha_j = 1$. The nodes are visited systematically by removing the node with the shortest distance from S in the list in each iteration and calculating the distances for its adjacent nodes and updating them if obtained to be lesser than the previous value. The path with the shortest distance from S to G is hence returned as a solution.

The benefit of this approach is that it explores most of the traversable region before reaching the goal configuration G, while not having to check for too many non-traversable regions and non-reachable traversable regions. Hence, the map generated is denser. The downside is that it is computationally more expensive since it triggers a check for many voxels.

5.2.2 A* Algorithm

A* algorithm [6] is quite similar to Dijkstra's algorithm in structure, except for the fact that it also includes a notion of heuristic regularity for updating nodes for path-finding. The heuristic used in this project is a function of physical distance between the goal configuration G and i^{th} node n_i. Hence, the distance values stored for each node n_i are a combination of min-cost sum $V_i = V_j + c_i$ (where n_j is the parent node from which n_i was queried) and heuristic cost $H_i = dist(n_i, n_g)$ where n_g is a set of nodes belonging to the goal configuration G.

The advantage of this approach is that the number of nodes to be checked decreases considerably depending upon the heuristic, making the planner output computation time much faster. A drawback, though, is that the costmap created is sparser.

5.3 Mapping

The queries from the motion planner trigger a classification process over a patch extracted from the current image frame, mapped from the ground plane using camera

projection matrices. The classification confidence is used to update the corresponding voxel belief as per equation (8)

$$b_k^{m^i} = \frac{p(m^i|z_k^i, t_k)p(z_k^i|t_k)p(m^i|z_{0:k-1}^i, t_{0:k})}{p(m^i)p(z_k^i|z_{0:k-1}^i, t_{0:k})}$$

$p(z_k^i|t_k)$ depends on a patch being present or not in an image. We take the value to be either 0 or 1, ignoring the noise in determining the camera position. Furthermore, we can take $p(m^i)p(z_k^i|z_{0:k-1}^i, t_{0:k})$ as a constant assuming that classification based measurements do not depend on measurement history. Hence, the equation can be simplified as

$$b_k^{m^i} = \eta p(m^i|z_k^i, t_k)p(m^i|z_{0:k-1}^i, t_{0:k}) \tag{18}$$

given that projections of G_i are present in the image. Using the fact that initial values of $p(m^i)$ are 0.5, η is taken as 2 for normalization.

To avoid repetitive computations, a threshold is used for making the above update to probabilities based on the idea that only the values with less confidence should be updated. The threshold interval is chosen to be [0.3, 0.7], and any values lying outside this region are assumed to be sure-traversable or sure-non-traversable. This system is also able to handle situations where the patch for a voxel from an image is ambiguous, and hence it is updated from the next image where it is more easily discernible.

5.4 Classifier update

Since the state of the environment may change over time, it is necessary that the prediction generated by the classifier are also able to vary to accommodate it. This is known as *concept drift*, and hence online learning is used to train the classifier during run time.

Online algorithms operate in rounds. On round i the algorithm receives an instance $x_i \in \mathbb{R}^d$ to which it applies its current prediction rule to produce a prediction $\hat{y}_i \in (-1, +1)$. It then receives the true label $y_i \in (-1, +1)$ and suffers a loss $l(\hat{y}_i, y_i)$. The algorithm then updates its prediction rule and proceeds to the next round. The prediction rule f_w used is a linear classifier with input x and weight w

$$f_w(x) = sign(w^T x) \tag{19}$$

The confidence c_w in prediction is obtained as

$$c_w(x) = |w^T x| \tag{20}$$

The update condition for training the classifier in online algorithms is given by

$$w_{i+1} = w_i + \alpha_i y_i x_i \qquad \alpha_i > 0 \tag{21}$$

For the Confidence-weighted linear classification [7] scheme used in this project, updates are performed as PA updates,

$$\alpha_i = max\left(\frac{1 - y_i(w_i^T x_i)}{\|x_i\|^2}, 0\right) \tag{22}$$

Since the online learning scheme models parameter confidence, $w \sim N(\mu, \Sigma)$ where N is a normal distribution with mean μ and variance Σ. From (21), they are updated using a confidence parameter ϕ as

$$\mu_{i+1} = \mu_i + \alpha_i y_i \Sigma_i x_i \tag{23}$$

$$\Sigma_{i+1}^{-1} = \Sigma_i^{-1} + 2\alpha_i \phi diag(x_i) \tag{24}$$

To generate the labels for data during runtime, a heuristic of neighborhood property is used - any voxel with more than 75% neighbors being sure-traversable is labelled as traversable and vice-versa. Hence, a query for any voxel G_i also checks its neighbors $Adj(G_i)$, and sends the corresponding label, if exists, to the classifier along with the patch image. The classifier uses the label for training, and returns the prediction from the updated classifier which is used to fill the voxel value.

6 Experiments and Results

6.1 Data

The data for experimentation is obtained from the on-board camera of a NEX 0x-Delta 4-wheel drive ground robot in a grassy terrain with lanes drawn with chalk over grass being the non-traversable region. The camera was mounted front-facing with no rotation between the robot heading and camera. The camera position information was determined manually to avoid sensor noise.

6.2 Experimental Setup

The system was implemented in ROS (Robot Operating System) framework, with the classifier portion in python and rest in C++ for runtime speedup. The images obtained we undistorted and then published over ROS topics along with camera position information.

7 Results

Visualization of resulting probabilistic map as well as path is shown in Figures 7 and 8 for the two motion planning algorithms on image (Figure 6), where the ground robot is assumed to be positioned at the start of the map.
Table 1 shows performance of the proposed algorithm against classification on full image, for an empty map on the input image.

Figure 6: Input image

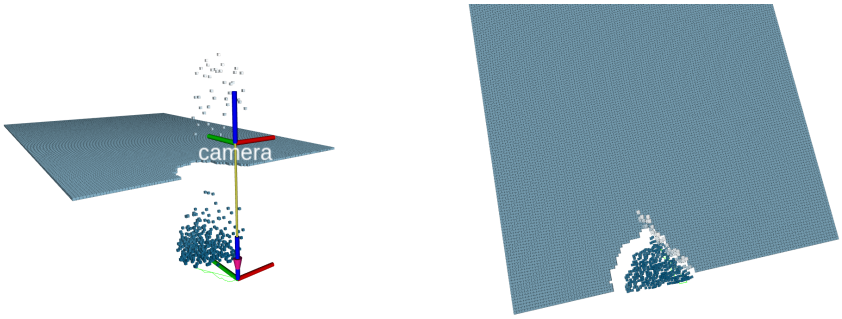


Figure 7: Dijkstra's Algorithm

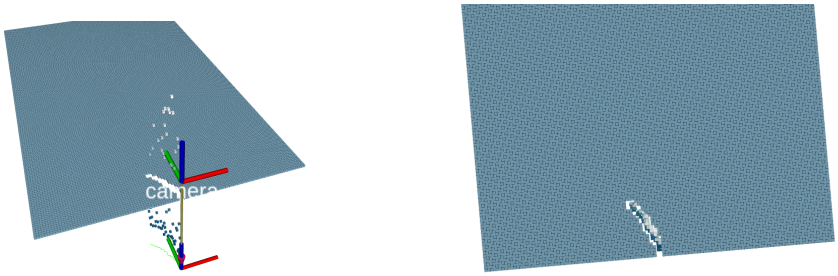


Figure 8: A* Algorithm

Algorithm	Time (seconds)
Online classifier (Full Image)	4.7
Offline classifier (Full Image)	21.2
Dijkstra's Algorithm	**3.4**
A* Algorithm	**0.6**

Table 1: Algorithm Performance

8 Conclusion and Future work

- **Incorporating for uncertainty in position of camera:** We hope to account for sensor noise in determining odometry values, and accordingly generate a map as well as plan better paths. Towards that end, we are experimenting with allowing a full range of $[0, 1]$ to $p(z_k^i|t_k)$ during mapping step.

- **Non-ground plane objects:** We are working on extending the system to be able to work with objects placed on ground as well. For that, we are using a 2D LiDAR to determine points belonging to an object, project them at varying heights onto the image to obtain object patches, and train the classifier to also detect ground-object boundaries (Figure 9). The mapping system can then use that in a similar manner as lanes.

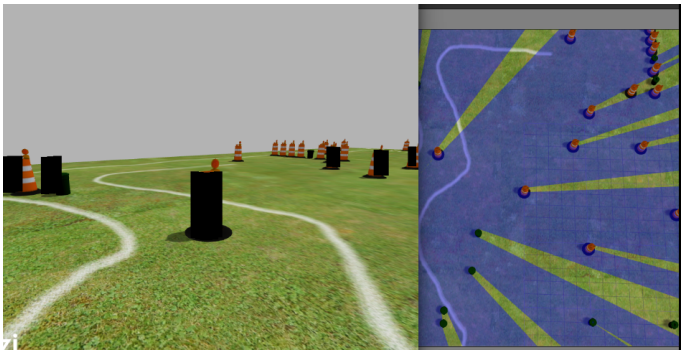


Figure 9: Object detection using LiDAR

9 Acknowledgement

I thank Prof. Mangal Kothari, Dept. of Aerospace Engineering and Prof. S. R. Sahoo, Dept. of Electrical Engineering for their valuable support, guiding me from

time to time and giving advice whenever it was needed. I also take this opportunity to thank the OpenCV community for their well documented and useful libraries.

References

[1] Ali-akbar Agha-mohammadi. SMAP: simultaneous mapping and planning on occupancy grids. *CoRR*, abs/1608.04712, 2016.

[2] Prof. Gaurav Pandey. Lecture slides. *EE698G*, Probabilistic Mobile Robotics, 2016.

[3] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, Aug 1995.

[4] Jean Serra. *Image Analysis and Mathematical Morphology*. Academic Press, Inc., Orlando, FL, USA, 1983.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, December 1959.

[6] W. Zeng and R. L. Church. Finding shortest paths on real road networks: The case for a*. *Int. J. Geogr. Inf. Sci.*, 23(4):531–543, April 2009.

[7] Mark Dredze, Koby Crammer, and Fernando Pereira. Confidence-weighted linear classification. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 264–271, New York, NY, USA, 2008. ACM.