
CAMEL: Learning Cost-maps Made Easy for Off-road Driving

Kasi Viswanath

Department of Electrical Engineering and Computer Science
Indian Institute of Science Education and Research Bhopal
Bhopal, India, 462066
kasi18@iiserb.ac.in

P.B. Sujit

Department of Electrical Engineering and Computer Science
Indian Institute of Science Education and Research Bhopal
Bhopal, India, 462066
sujit@iiserb.ac.in

Srikanth Saripalli

Department of Mechanical Engineering
Texas A&M University
College Station, Texas, 77843
ssaripalli@tamu.edu

Abstract

Cost-maps are used by robotic vehicles to plan collision-free paths. The cost associated with each cell in the map represents the sensed environment information which is often determined manually after several trial-and-error efforts. In off-road environments, due to the presence of several types of features, it is challenging to handcraft the cost values associated with each feature. Moreover, different handcrafted cost values can lead to different paths for the same environment which is not desirable. In this paper, we address the problem of learning the cost-map values from the sensed environment for robust vehicle path planning. We propose a novel framework called as CAMEL using deep learning approach that learns the parameters through demonstrations yielding an adaptive and robust cost-map for path planning. CAMEL has been trained on multi-modal datasets such as RELLIS-3D. The evaluation of CAMEL is carried out on an off-road scene simulator (MAVS) and on field data from IISER-B campus. We also perform real-world implementation of CAMEL on a ground rover. The results shows flexible and robust motion of the vehicle without collisions in unstructured terrains.

1 Introduction

Unmanned ground vehicles (UGV) are used in several terrains and off-road conditions for applications such as search and rescue, surveillance, inspection, exploration etc. In these unstructured environments with varying texture and slopes, achieving autonomous capability through planning is more difficult than structured urban environments. For autonomous traversal, cost function and maps are used to encapsulate the terrain features from the perception module(1). Many planning systems employ manually designed cost-maps and cost-functions (2) with successful demonstrations at the

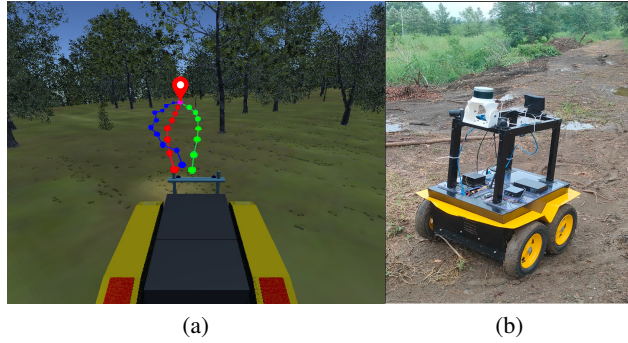


Figure 1: (a) Trajectories generated while navigating in a simulated off-road environment. **Trajectory generated by CAMEL**; **Trajectory generated from a finely handcrafted cost-map**; **Trajectory generated from coarsely designed cost-map**. (b) UGV used for real-world experiments.

DARPA Grand Challenge (3). Generally, these cost functions have obstacles inflated with respect to the vehicle size. The general weighting of costs to different perception sensors are handcrafted through numerous trials which is a laborious, time consuming and relies on detailed domain knowledge. Moreover, with different weighting can lead to the generation of different paths which is not desired.

Consider a surveillance application where a robot needs to navigate in a forest area (as shown in Figure 1a), the cost-map needs to estimate traversable regions coping with obstacles, trees, bushes, marshy area, puddle and irregular elevations. These scenarios introduce new challenges to the planning module and determining an optimal cost-map parameters through handcrafting would prove to be an inefficient strategy. In Figure 1a, we can see that the paths for the handcrafted costs is not optimal. This issue, motivates us to ask the question: Can we learn a cost-map taking the camera and LiDAR information directly such that the cost-map represents information similar to the expert human demonstration?

In this paper, we present an approach called CAMEL¹ that uses fully convolutional networks (FCN) to fit the multi-modal data and generate cost-maps through expert demonstrations. Deep Learning architecture enables learning high versatile, highly non-linear models necessary for complex and dynamic environments. Although, learning cost maps has been a topic of interest in the robotics community, however there are very few articles in this topic. In (4) and (5), deep reinforcement learning and deep inverse reinforcement learning techniques respectively are used to learn the cost-maps. However, in these approaches, initially a pre-trained CNN model trained on manually designed cost-map is used. The cost-map is further refined by the RL framework. This approach confines the model to the manually designed scene conditions. Due to the varying off-road scene conditions, this approach requires extensive amount of demonstration data to capture the complexity of off-road environment. In CAMEL, we bypass a pretrained model by directly learning to process the visual (camera) and point cloud (LiDAR) data from the perception module to generate trajectories that mimic human driving.

While traversing in off-road terrains, the vehicle stability and safety is essential and hence the visual and geometric features of the terrain such as vegetation, slope, soil stability need to be considered to plan robust and adaptive paths for the vehicle. These features are obtained by the sensors like camera, LiDAR and radar. Multi-modal datasets (6) (7) (8) provides reliable images and point cloud data of diverse scenes to develop algorithms for the outdoor domain. The CAMEL framework uses semantic projections, height map, slope and LiDAR reflectance intensity to generate a cost-map yielding the trajectories which is then compared to the actions of a human driver.

The main contributions of this paper are:

- A FCN based Deep Learning framework CAMEL able to predict 2D navigation cost-map for off-road terrains.

¹As camels can walk in different terrains, our proposed framework is applicable to different types of terrains and hence we named our architecture CAMEL.

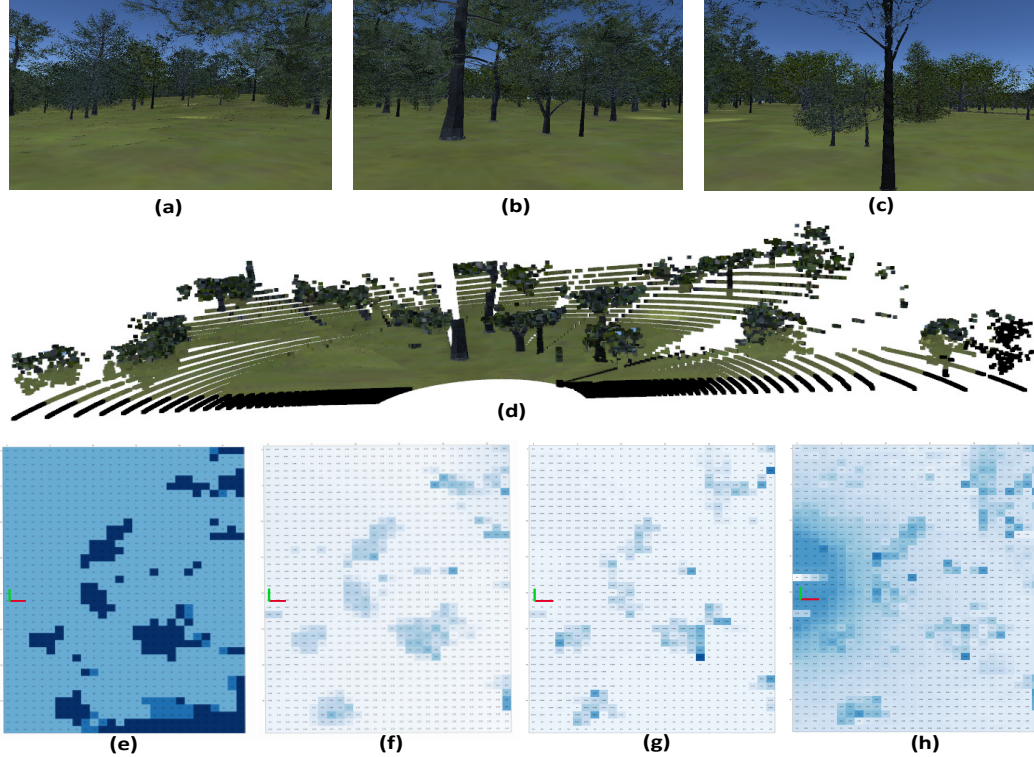


Figure 2: Figure shows the data generated from a single instance in simulation. (a), (b) and (c) are the images from left, forward and right camera respectively. (d) The three images projected to point cloud data. (e) The map of projected semantic segmentation on point cloud. The colours range indicates class hierarchy where blue represents traversable region and dark blue to obstacles. (f) The average height map extracted from the point cloud data. (g) The estimated slope map indicating the elevation of the terrain. (h) The LiDAR reflectance intensity map.

- A navigation stack module optimized to the generated cost-maps with the ability to handle intricate and dynamic topography. CAMEL being learned through human demonstrations, alongside the navigation stack generate trajectories that mimic human driving.
- Demonstration of the framework’s scalability from simulation training to real-world implementation (Sim2Real) with very few tuning parameters.
- Demonstration of the framework’s robustness in handling sensor miscalibration and system biases in the real-world experiments.
- We compare the predicted cost-map against a carefully handcrafted cost-map to show the efficacy of CAMEL.

2 CAMEL Methodology and Architecture

The CAMEL methodology consists of three stages: input data generation, model training and a navigation stack as shown in Figure 3a.

2.1 Input data generation

The input to the model consists of 4 different information coming from 3 cameras and a LiDAR as shown in Figure 2(a)-(d). These are (i) semantic projection (ii) average height map (iii) slope estimation and (iv) LiDAR reflectance intensity. We fuse all these four geometric and semantic information into a grid map of prescribed size and grid resolution. The dimensions and resolution of the grid map is decided based on the sensor ranges and the vehicle dimension. Assume the cost map size is $\ell \times w$, where ℓ is the length of the map and w is the width of the map and the map is discretized with resolution Δ . Each cell in the map is represented as γ . The vehicle’s position on the

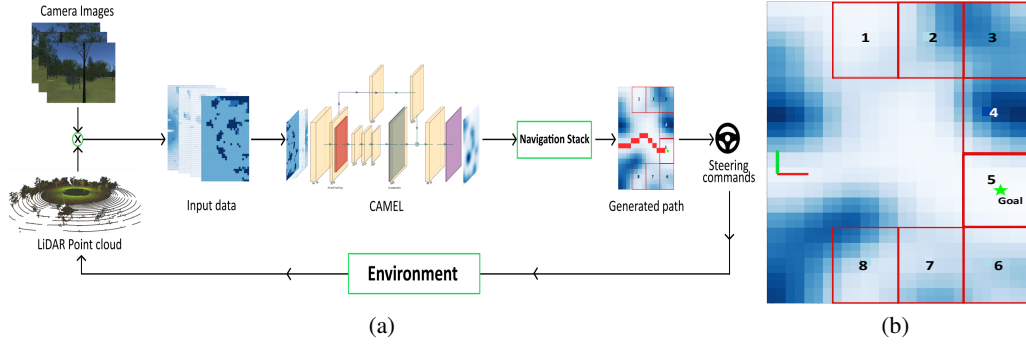


Figure 3: (a) The complete system framework consisting of data input, CAMEL, and the navigation stack (b) The kernel spaces used to find the local goal.

map is considered as the origin. The visual data and LiDAR data are fused to each of the cells in the grid. We now describe the generation of different inputs.

2.1.1 Semantic projection

For semantic segmentation we use Offseg(9) a semantic segmentation framework for unstructured terrains which generates a 2D mask for each pixels in the input RGB image classifying into different classes. The index of the classes is assigned based on an ascending risk factor (i.e, traversable to obstacle). The mask is then projected to the corresponding timestamped point cloud using calibration matrices for respective cameras. The projected points are then mapped onto the grid where for each cell value, the highest class index encompassed within the cell is assigned. Figure 2(e) is the resultant output using the camera input from Figure 2(a)-(c) and lidar input from Figure 2(d).

2.1.2 Geometric Characteristics

The LiDAR data provides spatial information as a point cloud. The sensor emits laser pulses which reflects off a surfaces of vegetation, buildings etc. The reflections are captured and in turn processed to provide the position of the surface in coordinate space. We compute different morphological and geometrical information of voxels generated from the point cloud.

Average Height map The information on the height profile of a terrain is necessary to safely navigate through the environment. This is extracted from the point cloud data. If the value of the cell is high implies there is an obstacle in the cell and if the value is low then it implies that its a safe cell to navigate as there is no obstacle in the cell.

To generate the height map, the average height of each cell H_γ is computed using voxels. Assuming n number of voxels in γ , H_γ is computed as the average height value of the voxels corresponding to cell γ . It is computed as

$$H_\gamma = \frac{1}{n} \sum_{\nu=1}^n H_\nu \quad (1)$$

Figure 2(f) shows the average height maps for an instance based on the information from Figure 2(d).

Slope Estimation By generating voxels, we take the neighbouring points in a cubic volume and do an averaging thereby giving us better estimates. Slope λ of a voxel is the angle between its surface normal η and the z-axis of world coordinate. It is computed by taking the cosine inverse of surface normal component to the z-axis of world coordinate. The slopes of the voxels are then averaged corresponding to their grid cells as

$$\lambda_\gamma = \frac{1}{n} \sum_{\nu=1}^n \arccos(\eta_\nu^z). \quad (2)$$

	Input	Block	c	k	s	Output
	$40 \times 28 \times 4$	Conv2D	64	5	1	$40 \times 28 \times 64$
feature extraction	output1	Conv2D	32	3	1	$40 \times 28 \times 32$
	output2	MaxPool2D	32	-	-	$20 \times 14 \times 32$
	output3	Conv2D	32	3	1	$20 \times 14 \times 32$
	output4	Conv2D	32	1	1	$20 \times 14 \times 32$
	output5	Conv2D	16	1	1	$20 \times 14 \times 16$
feature fusion	output6	Upsample	32	-	-	$40 \times 28 \times 16$
	output2	Conv2D	32	3	1	$40 \times 28 \times 32$
	output7	Conv2D	32	1	1	$40 \times 28 \times 32$
	output8+output6	Concatenate	64	-	-	$40 \times 28 \times 48$
	output8	Conv2D	1	1	1	$40 \times 28 \times 1$

Table 1: CAMEL uses standard convolutions (Conv2D) with two branches, a feature extractor and a feature fusion. The parameters c, k, s represents number of output channels, kernel size and stride parameters.

For better slope estimates k -dimensional tree ($k - d$ tree) is used to search the nearest neighbours. The radius for k -d tree is two times the voxel size with a maximum of 10 neighbours for better estimates. Figure 2(g) shows the slope map for the input given in Figure 2(d).

LiDAR reflectance intensity LiDAR intensity ι can be used to classify terrains and vegetation (10). The intensity by which the laser pulses are reflected depends on different morphological properties such as moisture content, roughness, range and surface composition. These characteristics can provide distinctive features on classes such as puddle, grass, asphalt apart from semantic segmentation. For each grid cell we compute the average intensity of the voxels as

$$\iota_\gamma = \frac{1}{n} \sum_{\nu=1}^n \iota_\nu. \quad (3)$$

Figure 2(h) shows the slope map for the input given in Figure 2(d).

2.2 CAMEL Architecture

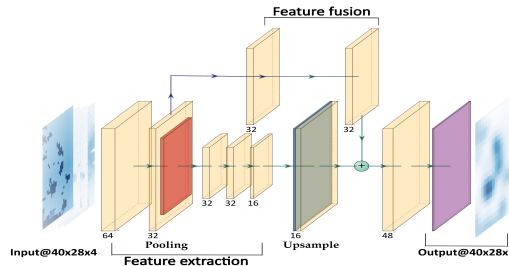


Figure 4: The proposed CAMEL architecture with feature extraction and feature fusion branches.

We develop a novel architecture based on Fully Convolutional Network called CAMEL to generate cost-map through demonstrations based on the perceived data from sensors. This could be considered as an imitation learning approach wherein a model tries to imitate decisions taken by human experts at a given instance. Inspired from Multi-Scale Fully Convolutional Networks(5), the architecture consists of three segments: a feature extraction, a feature fusion as shown in Figure 4 along with a navigation module to extract the steering values from the output cost-map.

For the feature extraction module, we employ four layers of convolutions and a pooling layer as listed in Table 1 to ensure the low-level feature sharing is valid. All convolutional layers are standard Conv2D with max-pooling layer. The convolutional layers employ a single stride, replicate padding followed by leaky-ReLU activation(11). The first layer has spatial kernel size of 5×5 and the other three with 3×3 kernel size.

A skip connection is introduced which takes the output of the second Conv2D layer before max-pooling and computes two convolutions with kernel sizes 3×3 and 1×1 following leaky-ReLU activation. This enables the model to preserve translational variant and invariant features. The output from feature extraction module is then upsampled by a factor of 2 which is concatenated with the skip connection. This enables the model to treat feature channels separately. A final convolution is applied on the concatenated output yielding the cost-map.

Using steering value $y \in [-1, 1]$ as the ground truth, a path from start to a desired goal is generated from the output cost-map yielding steering value \hat{y} which is used to compute the training loss. Mean squared error (MSE) loss function gives the loss between the target and predicted steering values backpropagated for weight updation. Adam optimizer(12) along with weight decay ($L2$ regularization) to avoid the exploding gradient of weights. The extraction of steering value from cost-map is discussed broadly in the next section.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2. \quad (4)$$

2.3 Navigation Stack

Along with the cost-map predictor we propose a navigation module compatible and optimized for the generated cost-map consisting of few parameters. The input data $I \in \mathbb{R}^{4 \times l \times w}$ is passed to the model giving an output cost-map. The cost-map C is then normalized to 1 i.e. $C \in [0, 1]$ for easier parametrization.

For navigation a path needs to be generated in C that has the least cost. This task is achieved by using a path planning algorithm like A^* (13) or Dijkstra(14) for a given start and goal. The start location is the current position of vehicle, however we need to provide a goal in C . Here we introduce a framework for selecting a favourable goal for local path planning from C . To compute the local goal, a kernel space of one sixteenth the map dimension is considered at different locations in C as shown in Figure 3b. For all the kernels, the mean and the least cost cell value within the kernel is computed and summed giving a traversability coefficient T as given in equation (5). Now based on the current orientation of the vehicle towards goal g in the global frame, a finely tuned weight is generated for each kernel where the kernel k oriented in the direction of g has least weight. Weights are then distributed across other kernels based on the euclidean distance from k . These weights are then multiplied with T to give the final kernel risk coefficients V .

$$T_k = \frac{1}{m} \sum_{i=1}^m x_i + \min(x_1, x_2, \dots, x_m) \quad (5)$$

where m is the number of cells in kernel k .

Considering the distance between two adjacent kernels as unit measure. Weights for kernel i can be calculated from:

$$W_i = W_k \times \text{dist}(k, i) \quad (6)$$

where W_k is the tuned weight of the kernel oriented towards g . The risk coefficients V are generated by multiplying traversability scores with the corresponding weights and the kernel. The kernel with least V is selected as the goal kernel k with the cell having least cost in the kernel as local goal point.

$$V_i = W_i \times T_i \quad (7)$$

for kernel k_i and

$$k = \text{argmin}(V) \quad (8)$$

where kernel k is the local goal kernel. Now with the start and goal we compute a path planning algorithm to find least cost path for traversing. From this path a series of steering and throttle values are generated for the vehicle to reach the goal. By employing the navigation stack, an optimal goal in the cost-map is found considering the occupancy of neighbouring cells, cost value of the goal, orientation with respect to the global goal thereby generating a low risk, least cost path. Further discussion regarding different parameters, kernel sizes, weights are described in the next section.

3 Experiments

The CAMEL framework is trained and tested on both simulation and real world scenarios. The simulations are done on Mississippi State University Autonomous Vehicle Simulator (MAVS) (15) for different scenes while for real world, we trained the algorithm on RELLIS-3D(6) dataset and tested on data collected in the IISERB-campus. Further, we have CAMEL implemented on a real robot. This section includes the experiments performed in the simulation framework followed by real-world implementation.

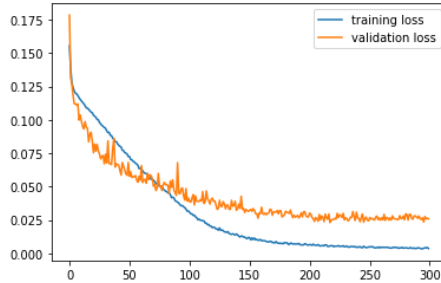


Figure 5: Training curves on simulation data. Loss over epochs for training and validation datasets are shown. The least validation loss of 0.0256 is achieved at the 217th epoch.

3.0.1 Simulation

Initial testing of the framework was performed on simulation which includes data collection of scenes, parameter tuning and navigation testing. The vehicle used is an inbuilt skid-steer model of Clearpath Robotics Warthog UGV with sensors mounted and integrated to ROS. In this section we will discuss on sensor setup, data collection, model training and navigation.

Sensor setup: We use three cameras with similar intrinsic parameters and a Velodyne HDL-64E LiDAR where the three cameras are positioned and oriented to give a 180° overlap field of view as given in Figures 2. The vehicle has a GPS, an imu sensor and an odometry sensor which provides position and orientation of the vehicle in world frame.

Data collection: For training CAMEL, we collect the sensor data while the vehicle is driven by a human expert. The driver inputs throttle $\epsilon[0, 1]$ and steering value $\epsilon[-1, 1]$ which is updated to the vehicles controls through teleoperation. The data was collected for three different scenes then processed to remove desynchronized data and outliers from the pool. This throttle and steering value is used as ground truth to train the model.

We constructed a grid of dimension 40×28 with the grid resolution being $0.3m \times 0.3m$. The grid resolution is taken approximately one third the vehicle width for a stable navigation. With these parameters we cover an area of $112.8 m^2$ in front of the vehicle. The point cloud belonging to this region is extracted and converted to voxels with voxel size of $0.15m$ (half of grid resolution). This gives a good estimate for point localization and surface normal estimates. These voxels are used to extract the geometric data grids such as average height map, slope estimation, LiDAR reflectance intensity map.

The semantic segmentation masks for the three cameras were generated through Offseg pretrained on RELIS-3D dataset and the indices arranged in a risk based ascending order. Offseg gives an average mIoU of 78% on the three scenes which ensures the segmentation data used for semantic mapping on LiDAR points are true. Offseg gives predictions in four classes namely sky, traversable, non-traversable, obstacle. These masks are then projected to the LiDAR voxels for generating the semantic grid map. The approaches (4) and (16) uses the semantic information only to coarsely differentiate the grid cells between traversable and non-traversable. Here we provide a grouping of grid cells based on semantics and hierarchy of classes (dependent on traversability) along with geometric information to yield better cost estimates by CAMEL.

These four grid maps of an instance is appended together to create a $R^{40 \times 28 \times 4}$ input data for the model to train. A total of 4000 data instances from the simulation were generated which was shuffled and divided between train, validation and test dataset in the ratio 7 : 2 : 1. For grid cells encompassing zero voxels, we perform a neighbourhood interpolation. This is because *nan* values in the data leads to huge loss of information while training CAMEL.

Model training: We conducted experiments of CAMEL on PyTorch framework using Python. The workstation used has a configuration of Nvidia RTX 3060 with CUDA 11.1 and CuDNN v8.1. We used 100 instances to report the average frames per second (fps) measurement.

ADAM optimizer with a learning rate of 10^{-4} and sequentially increasing weight decay is used to train the model for 300 epochs. Here the output of the model being a cost-map of dimension $40 \times 28 \times 1$ and the ground truth being a steering value, we need to extract a steering value from

the cost-map to compute the loss. We employ the navigation stack on tensors in order to maintain the gradients and extract the steering value. The MSE loss is then computed between the target and predicted steering values which is backpropagated to the model for weights updation.

The training and validation loss are shown in figure 5 where the model attains convergence with the least loss of 0.0256 on validation dataset whereas the test dataset gave a loss of 0.0213. The best model is then used to test on real time simulation where the vehicle explored the environment without collisions.

Navigation: The best epoch from training is used to test CAMEL in the simulation. For every instance the data was processed and used to predict the cost-map for the vehicle to navigate. This cost-map is then processed by the navigation stack which extracts the throttle and steering commands. For a smooth and jerk free motion of the robot, throttle and steering values are limited at 0.7 and $[-0.4, 0.4]$ respectively. Parameters such as weights for goal kernel and threshold for obstacle kernel were tuned through trials to get a robust motion of the vehicle. The threshold for obstacle kernel was found optimal at 0.625 for simulations.

While training we considered the costs of three adjacent grid cells to compute the least cost-map using Dijkstra thereby incorporating the size of the vehicle for steering clear from obstacle at safe distances. Figure 1a shows trajectories generated from CAMEL and handcrafted cost-maps. While testing we observed that the path generated by Dijkstra considering single grid cell cost is identical to the one generated by considering costs from three grid cells. This suggests that the vehicle specific characteristics also influences CAMEL in learning the cost-map.



Figure 6: GPS tracks of the vehicle while traversing autonomously in IISER-B campus during testing at two different regions. The terrain included steep climbs, puddles, dynamic obstacles where the vehicle performed robust planning. Blue circle \rightarrow starting location, blue star \rightarrow mission end. The vehicle traversed a total distance of 450 meters in the two tracks.

3.0.2 Real World scenarios

The models trained on simulation is used for testing at various locations of IISER-B campus. We use a skid-steer vehicle with comparable dimensions to the Clearpath Robotics Jackal UGV mounted with calibrated sensors and Jetson AGX Xavier platform for on-board computations.

Sensor setup: We use two Teledyne Flir Firefly S mounted at specific angles giving a 180° field of view overlap along with Velodyne VLP-16 LiDAR both calibrated. The vehicle uses PixCube Orange for flight controls consisting of GPS, compass and IMU sensor giving us the position and orientation of the vehicle.

Navigation: The best CAMEL model obtained from simulation was used for navigation in real-world. Offseg pre-trained on RUGD dataset(17) generated the semantic segmentation as it is found to produce acceptable results in IISER-B campus scenes. The computation time taken to process a single instance on average is 0.95 seconds. The output throttle and steering from the current instance is executed for 1 second as the maximum velocity of the vehicle is 0.5 m/s where the commands generated are valid for up to 1 meters in front of the vehicle. The track followed by the vehicle is shown in Figure 6. In this experiment, we generated only local goals based on the description given in the Navigation stack (Section 2.3).

Moving from simulation to real-world implementation, the vehicle performed robust maneuvers with modification only for maximum throttle and steering constraints. This showcases the ability of CAMEL for direct simulation to real world (Sim2Real) application. Figure 7 shows an instance from the ground testing in Figure 6(b). An imprecise calibration for the yaw angle of camera resulted in a complete failure of handcrafted cost-map 7(d). However, the learned cost-map as shown in Figure 7(e) is equivalent to the cost-map generated from calibrated data. CAMEL was able to differentiate the features despite perturbation in the system and predict a close to optimal path.

While navigating we observed at instances where Offseg gives subpar segmentation, the cost-map generated maintains its prediction quality suggesting well balanced weight distribution to geometric information from LiDAR during training. During experiments vehicle opted to traverse through grass patch over puddle indicating in authors view, the involvement of LiDAR intensity for vegetation prediction but more exhaustive study is required to ascertain the observation.

Unlike HDL-64E LiDAR in the simulation, the VLP-16 gives $4\times$ sparser point cloud. The framework still generates a cost-map that has acceptable throttle and steering commands. The algorithm could also classify negative obstacle from a distance assigning high grid costs hence steering away from the location. The inference speed of the whole framework with two Offseg predictions, a cost-map

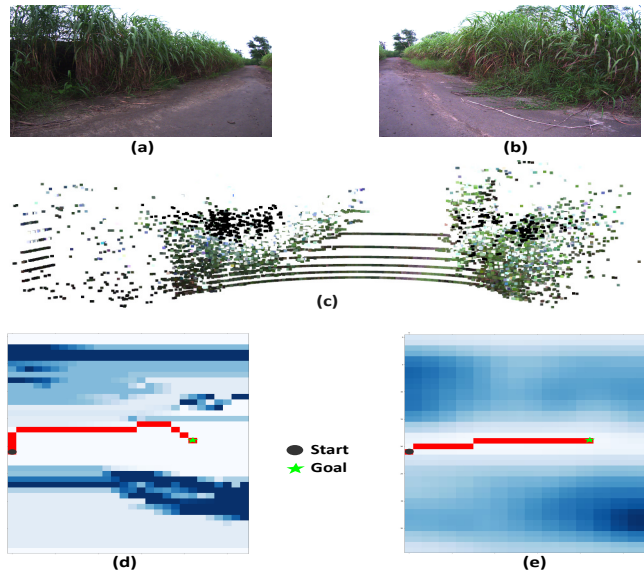


Figure 7: Real world implementation: (a) and (b) are the images from left and right camera. (c) Images projected on the VLP-16 point cloud data. (d) and (e) shows a comparison between the path generated from (d) manually handcrafted cost-map and (e) CAMEL predicted cost-map.

prediction and the navigation stack was tested on Jetson AGX Xavier. The Offseg with BisenetV2(18) gave an inference speed of 92 ms for each image whereas the FCN model has 4.5 ms inference speed. The whole data processing and navigation stack took on an average 0.95 seconds to process. This computation time can be reduced by multi-processing and optimization.

4 Conclusion and Future Work

In this work, we presented CAMEL a FCN based deep learning framework to navigate a UGV reliably in outdoor environments. Fusing RGB images and point cloud data we learn a cost-map from expert human demonstrations completely eliminating a handcrafted cost-map. Our approach is validated in simulation and real-world off-road terrains showcasing highly robust and adaptive motion by the UGV. The framework exhibited invariancy to perturbations in sensor calibrations and system biases making manually crafted cost-maps completely obsolete.

Future work will target extensive study of LiDAR intensity and learning vegetation features in different climatic conditions. Another interesting direction is the effective sensing of negative obstacles such as steep troughs which is difficult to classify due to low LiDAR reflectance.

References

- [1] T. Overbye and S. Saripalli, “G-vom: A gpu accelerated voxel off-road mapping system,” 2021.
- [2] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, May 2005.
- [3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, “Stanley: The robot that won the darpa grand challenge,” *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006. [Online]. Available: <http://dx.doi.org/10.1002/rob.20147>
- [4] K. Weerakoon, A. J. Sathyamoorthy, U. Patel, and D. Manocha, “Terp: Reliable planning in uneven outdoor environments using deep reinforcement learning,” in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 9447–9453.
- [5] M. Wulfmeier, D. Rao, D. Z. Wang, P. Ondruska, and I. Posner, “Large-scale cost function learning for path planning using deep inverse reinforcement learning,” *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1073–1087, 2017.
- [6] P. Jiang, P. Osteen, M. Wigness, and S. Saripalli, “Rellis-3d dataset: Data, benchmarks and analysis,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1110–1116.
- [7] A. Valada, G. Oliveira, T. Brox, and W. Burgard, “Deep multispectral semantic scene understanding of forested environments using multimodal fusion,” in *International Symposium on Experimental Robotics (ISER)*, 2016.
- [8] D. Maturana, P.-W. Chou, M. Uenoyama, and S. Scherer, “Real-time semantic mapping for autonomous off-road navigation,” in *Field and Service Robotics*. Springer, 2018, pp. 335–350.
- [9] K. Viswanath, K. Singh, P. Jiang, P. Sujit, and S. Saripalli, “Offseg: A semantic segmentation framework for off-road driving,” in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, 2021, pp. 354–359.
- [10] C. Reymann and S. Lacroix, “Improving lidar point cloud classification using intensities and multiple echoes,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 5122–5128.
- [11] A. L. Maas, “Rectifier nonlinearities improve neural network acoustic models,” 2013.
- [12] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [13] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968. [Online]. Available: <https://doi.org/10.1109/tssc.1968.300136>
- [14] E. W. Dijkstra, “A note on two problems in connexion with graphs,” *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.
- [15] C. Hudson, C. Goodin, Z. Miller, W. Wheeler, and D. Carruth, “Mississippi state university autonomous vehicle simulation library,” in *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium*, 2020, pp. 11–13.
- [16] T. Guan, Z. He, D. Manocha, and L. Zhang, “TTM: terrain traversability mapping for autonomous excavator navigation in unstructured environments,” *CoRR*, vol. abs/2109.06250, 2021. [Online]. Available: <https://arxiv.org/abs/2109.06250>

- [17] M. Wigness, S. Eum, J. G. Rogers, D. Han, and H. Kwon, “A rugd dataset for autonomous navigation and visual perception in unstructured outdoor environments,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5000–5007.
- [18] C. Yu, C. Gao, J. Wang, G. Yu, C. Shen, and N. Sang, “Bisenet v2: Bilateral network with guided aggregation for real-time semantic segmentation,” 2020.