Decision Assist For Self-Driving Cars

Sriram Ganapathi Subramanian*, Jaspreet Singh Sambee, Benyamin Ghojogh, Mark Crowley

Department of Electrical and Computer Engineering, University of Waterloo, Canada
(s2ganapa, jssambee, bghojogh, mcrowley)@uwaterloo.ca

Abstract. Research into self-driving cars has grown enormously in the last decade primarily due to the advances in the fields of machine intelligence and image processing. An under-appreciated aspect of self-driving cars is actively avoiding high traffic zones, low visibility zones, and routes with rough weather conditions by learning different conditions and making decisions based on trained experiences. This paper addresses this challenge by introducing a novel hierarchical structure for dynamic path planning and experiential learning for vehicles. A multistage system is proposed for detecting and compensating for weather, lighting, and traffic conditions as well as a novel adaptive path planning algorithm named **Checked State A3C**. This algorithm improves upon the existing A3C Reinforcement Learning (RL) algorithm by adding state memory which provides the ability to learn an adaptive model of the best decisions to take from experience.

Keywords: Autonomous cars, path planning, obstacle avoidance, reinforcement learning, weather estimation, machine learning

1 Introduction

Building **Autonomous Driving Systems (ADS)** is a major goal of artificial intelligence research. The standard approach breaks this problem into six levels defined by SAE International [2] ranging from no automation (level 0) up to full automation (level 5). Path planning could be used for routing a self-driving car which needs to be undertaken given the coordinates of its starting and ending points. However in practice, this routing needs to be dynamic in a self-driving car with changes necessitated for obstacle, road blockage, bad lighting, bumpy roads, etc. Under harsh weather, darkness, or heavy traffic, the car might face difficulty in obstacle detection or even path traversal. The goal of this work is to fill a gap in the literature for enabling a self-driving car to find the most appropriate path between given starting and ending points not merely in terms of shortest path but in terms of different weather, lighting, and traffic jam conditions and considering the abilities and strengths of the driver under those conditions, be it human or automated.

* Note that the three first authors contributed equally to this work.

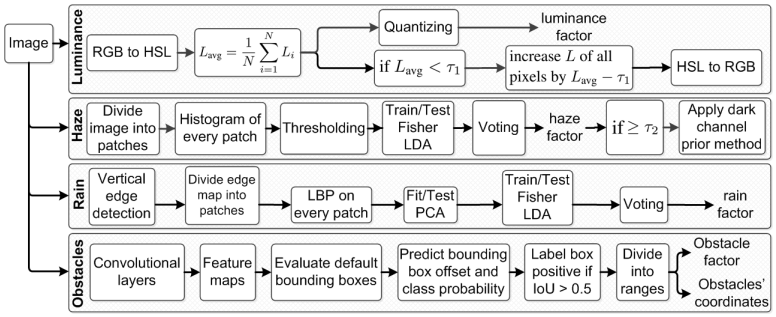

Fig. 1: Low-Level Processing: Image Condition and Obstacle Detection module.

A complete system for a fully autonomous vehicle capable of mapping and localization and low-level navigation planning was proposed in [7]. The key challenges resolved in such complex systems include navigating narrow roads, cross walks, and traffic light recognition and following. However, these systems do not include high-level path planning to dynamically consider the complete alternate routes to the goal points. Our proposed system using Deep Reinforcement Learning (RL) can handle road condition changes dynamically since during training, the assessment of conditions and planning are integrated into a single model learned. This adaptability would be very expensive to obtain in a classical path planning system using shortest path algorithms such as the Dijkstra algorithm.

The proposed system is composed of a (low-level) image condition and obstacle detection module and a (high-level) path planning module. An input image frame is fed to the low-level module which estimates and outputs the amounts (factors) of luminance, haze, rain, and obstacles, namely pedestrians and cars, present in the image. This module also enhances images to normal conditions when necessary. The high-level module applies a Deep RL algorithm to the low-level outputs in order to decide for the best possible path the self-driving car should traverse between starting and ending points. In the rest of this paper, the proposed system is explained and then verified by the experiments.

2 Image Condition and Obstacle Detection

The low-level module estimates four road condition factors directly from images: **luminance**, **haze**, **rain**, and **obstacles** as well as **enhancing** images if necessary (luminance enhancement for night scenes and haze removal for foggy or snowy conditions). These four factors are mapped to integer levels ranging from 1 (for bright, not hazy, not rainy, no obstacles) to 5 (too dark, too hazy, too rainy, too many obstacles) which are then fed to path planning module (see Fig. 1). The obstacle factor ranges are: 1: no obstacle, 2: 1-2 obstacles, 3: 3-5 obstacles, 4: 6-8 obstacles and 5: 9 obstacles or more.

Luminance: The image color model is changed from RGB (Red, Green, Blue) (if gray-scale, taking intensity for all channels) to HSL (Hue, Saturation,

Luminance) to extract luminance of each pixel i. The luminance factor is obtained by quantizing the average luminance of N-pixel image. If luminance is less than a threshold τ_1 (is dark), the luminance of all pixels are increased.

Haze: Histograms of intensities for different patches of a hazy image, which were empirically observed to have similar patterns, are found and after thresholding, a 256×1 feature vector is used for feeding to Fisher Linear Discriminant Analysis (LDA) with classes hazy and normal. By voting among the recognized patches of an image, haze factor is the number of hazy patches normalized by the total number of patches. The dark channel prior method for haze removal [3] is used if the haze factor reaches a threshold τ_2 (is hazy).

Rain: We consider both rain and snow streaks similarly as in literature [1]. These are mostly vertical even in strong winds and thus are captured well by a 3×3 vertical Sobel kernel. The Local Binary Pattern (LBP) method is used to extract texture features on patches of image, which then have Principal Component Analysis (PCA) applied for feature extraction. Fisher LDA is then used for classification on the resulting features and voting among recognized patches of an image determines the rain factor.

Obstacles: Obstacle detection is done in images having pedestrians and cars using the Single Shot Detector (SSD) approach [5]. SSD is a Deep Learning object detection approach in which the input image is fed into a set of convolutional layers which yields feature maps of varying scales. SSD consists of default bounding boxes and these bounding boxes are evaluated using a 3×3 convolutional filter which is placed at the end of the main convolution chain. For each of these boxes, a prediction on the bounding box and the class probability is made and the Intersection over Union (IoU) is applied on the ground truth labels and the predicted labels. The box which has an IoU greater than 0.5 is chosen and the rest ignored. The Tensorflow Object Detection API [4] was used to perform the SSD-based object detection and their base checkpoints were trained for further 7000 steps using the images manually labelled by LabelImg tool [11].

3 Path Planning

For the high-level task of deciding which paths to take we now introduce **Checked State A3C (CS-A3C)**, a Deep RL algorithm (see Algorithm 1) which is a modification of **Asynchronous Advantage Actor-Critic (A3C)**, a state-of-the-art policy gradient method [8]. We use a global network of workers, which is composed of convolutional layers and a Long-Short-Term-Memory (LSTM) layer. The function of the convolutional layers is to process spatial dependencies and the work of the LSTM layer is to process the temporal dependencies between the different input images.

A Deep Q Network (DQN) network represents the action policy as defined in [9] with 7 worker agents used. For simplicity we assume here the car cannot move in the reverse direction. A discounted reward model is used with the rewards in the range [-1,1]. The reward function assigns +1 for reaching the goal point, for each factor from the low-level module the rewards are normalized to the

Algorithm 1 Checked state A3C

Initialize thread step counter t ← 1
while $T > T_{max}$ **do**
  Reset Gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$
  Synchronize thread- specific parameters: $d\theta' = \theta$ and $d\theta'_v = \theta_v$
  $t_{start} = t$
  Get state s_t and check if it is a checked state s*
  while terminal s_t or $t - t_{start} == t_{max}$ **do**
    Perform a_t according to policy $\pi(a_t|s_t;\theta')$
    Receive reward r_t and new state $s_t + 1$.
    Discount factor : $\gamma = 0.9$ for s* and 0.1 for s != s*
    Propagate the rewards until the nearest s*.
    Apply learning rate $\alpha_t = 1$ for s = s* and $\alpha_t = 0.0001$ for s != s*
    $t \leftarrow$ t+1
    $T \leftarrow$ T+1
  $R = 0$ for terminal s_t
  $R = V(s_t, \theta'_v)$ for non-terminal s_t
  for $i \in t-1, \ldots, t_{start}$ **do**
    $R \leftarrow r_i + \gamma R$
    Accumulate Gardients w.r.t. θ and θ_v
  Perform asynchronous update of θ and of θ_v

range [-0.5,+0.5] using its factor, +0.1 for reduction in euclidean distance to the goal point and -0.2 for any increase in distance. We make three important changes to the A3C algorithm from [8]: (I) **Checked states:** Some states are more important to learn about than others. When the car chooses a path at a road intersection, the resulting state (travelling until the next intersection) needs to be updated with rewards from subsequent, more minor states resulting from driving and lane keeping actions. We call this a **checked state** and weight it more strongly in reward propagation. (II) **Selective Learning:** The discount factor (gamma) is different for checked (0.9) and non-checked states (0.1) making learning selective. (III) **Pseudo states:** All states apart from the checked state are pseudo states as the action is completely deterministic.

4 Experimental Results

Results of Low-Level Module: Figure 2 includes several results of the low-level module. Figures 2a and b respectively have luminance factors of 2 and 4 (dark). The result of luminance enhancement for Fig. 2c is shown in Fig. 2e. The dehazing result of Fig. 2d having haze factor of 5 (too hazy) is in Fig. 2e. The rain factor of Fig. 2f is 5 (too rainy) and the obstacle factors of figures 2g, h, i, and j are 2, 3, 4, and 5 (too crowded) respectively. The curve of training loss of object detection versus training steps is shown in Fig. 3a.

Overall Results and Comparison: In this work, the Oxford robocar dataset [6] is used for experiments. Linear recurrence random goal points [10] are selected on the Oxford area for the training and results are reported for about 72 hours of training (Fig. 3b). A car was made to virtually traverse the domain from a given starting point. Camera images are loaded from the dataset based on the spatial location of the vehicle. The CNNs, along with the output from the low-level layer, process the next free location in the image where the car can

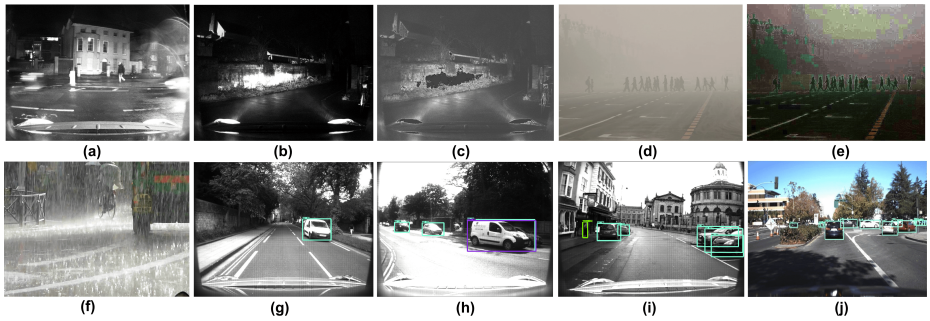

Fig. 2: Several results of image condition and obstacle detection.

move to. Once this is done, the corresponding image is loaded from the dataset. This continues until the car reaches the goal point. For training CS-A3C, the rewards are increasing over a period of time which implies that the agent is doing better with time (Fig. 3b).

For comparison, a weighted graph is made by averaging the low-level factors for each image in every edge in the map. The Dijkstra algorithm is run on this weighted graph using 15 different goal points set at equal distance intervals from a fixed starting point. Distance intervals are about 100m, ranging from 0 to 5000m. The car is made to virtually traverse the given path in simulation and the average time taken is compared to the path returned by CS-A3C after training times of 12, 24, 36, 48, 60, and 72 hours. Figure 3c shows that average time taken by Dijkstra to reach the goal point is less than CS-A3C trained for 12, 24, 36, and 48 hours. However, CS-A3C performs better after 60 hours. This also corresponds well with the graph seen in Fig. 3b where the accumulation of rewards drastically increases at about 60 hours of training. The training of CS-A3C is stopped within 72 hours as at this stage it comfortably beats Dijkstra. The CS-A3C algorithm tunes the weights of the different factors that determine the edge weights of the graph at training time, based on the time to goal point and the learned policy differentiates the edges based on experience. The Dijkstra algorithm, on the other hand, considers only an average edge weight. Thus, CS-A3C ultimately beats the Dijkstra algorithm after sufficient training in terms of time taken to reach the goal point.

5 Conclusion

The new CS-A3C algorithm is a highly efficient hierarchical path planning system with clear advantages over the traditional Dijkstra approach: (I) Our system requires a map only in the training phase and not at test time while Dijkstra needs a new map for every attempt. (II) If the conditions (luminance, weather, and traffic) of the road change during path traversal Dijkstra needs to be run again on a new graph. In contrast, our system needs to be trained only once

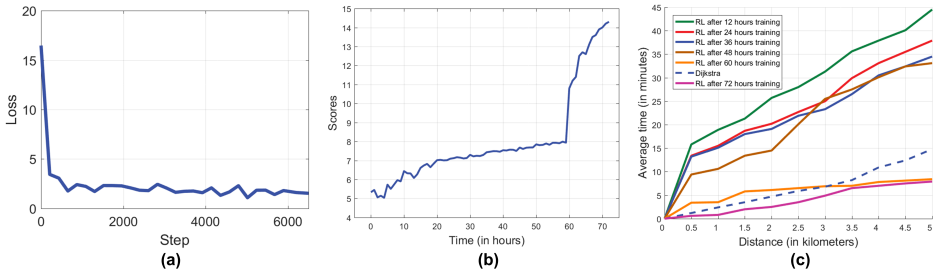

Fig. 3: Overall results: (a) training loss of object detection vs. training steps, (b) accumulation of rewards vs. training time, (c) comparison to the Dijkstra.

using a variety of conditions to learn a robust path planning model that applies under many changes in conditions.

References

1. Barnum, P.C., Narasimhan, S., Kanade, T.: Analysis of rain and snow in frequency space. International journal of computer vision 86(2), 256–274 (2010)
2. Committee, S.O.R.A.V.S., et al.: Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems. SAE Standard J3016 pp. 01–16 (2014)
3. He, K., Sun, J., Tang, X.: Single image haze removal using dark channel prior. IEEE transactions on pattern analysis and machine intelligence 33(12), 2341–2353 (2011)
4. Huang, J., Rathod, V., Sun, C., Zhu, M., Korattikara, A., Fathi, A., Fischer, I., Wojna, Z., Song, Y., Guadarrama, S., Murphy, K.: Speed/accuracy trade-offs for modern convolutional object detectors. Honolulu, Hawaii (2016)
5. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. In: European conference on computer vision (ECCV). pp. 21–37. Springer, Amsterdam, Netherlands (2016)
6. Maddern, W., Pascoe, G., Linegar, C., Newman, P.: 1 Year, 1000km: The Oxford RobotCar Dataset. The International Journal of Robotics Research (IJRR) 36(1), 3–15 (2017)
7. Milford, M.J., Wyeth, G.F.: Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In: Robotics and Automation (ICRA), 2012 IEEE International Conference on. pp. 1643–1649. IEEE (2012)
8. Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. pp. 1928–1937 (2016)
9. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (2015)
10. Tausworthe, R.C.: Random numbers generated by linear recurrence modulo two. Mathematics of Computation 19(90), 201–209 (1965)
11. Tzutalin, D.: LabelImg annotation tool. `https://github.com/tzutalin/labelImg`, accessed: 2017-18-10