High-Speed Autonomous Driving through Unknown Map

Xiaobai Ma
Department of
Astronautics & Aeronautics
Stanford University

Zhankai Wang
Department of
Astronautics & Aeronautics
Stanford University

Siyan Guo
Department of
Civil & Environmental Engineering
Stanford University

***Abstract*—In this work, we implemented Ritcher's[1] Bayesian Learning method to predict collision for ground vehicle moving in unknown and partially observable maps. We also modified the feature extracting function to seek for potential improvement. Section III will introduce the MDP model used in this High-Speed Navigation problem. In section IV, we will give description about the data set, feature extraction methods,and training procedure that we have adopted. Section V gives the learning method we used as well as the planning algorithm we implemented for testing, and section VI will show the simulation results of this method implemented for vehicle navigation. By comparing the performance of Bayesian Learning method using different feature extraction methods with the baseline method, we were able to find the best feature extraction methods implemented.**

***Keywords*—Bayesian Learning, MDP Planning, Autonomous Driving**

I. Introduction

In recent years, the rapid improvement of computational capability and sensing technology has made possible various applications in the field of autonomous driving and navigation. With safety being the highest priority, few researches have put the focus on high speed navigation systems. One of the pioneers is the Robust Robotics group at MIT, who has made an effort to develop a machine learning algorithm that allows high-speed driving in unknown maps with minimized chances of collision.[1] The objective is to train the robot the proper way of approaching and avoiding obstacles as human beings would, such as approaching a corner from a wide angle, while maintaining a high speed. The basic idea of the algorithm is to extract features from the vehicles configuration, the observed map, and the action the vehicle could take, and use these features to predict the collision probability of this action.

The input to our algorithm is the vehicle configuration, observed map, and the coordinate of the goal location. We the use a non-parametric Bayesian inference model with training data to output the "optimal" action which ensures collision free while trying to reach the goal in shortest time.

Since there are shared components between CS229 and AA228 projects, we write the work partition here. The model of the system including the state space, action space, and state update parts are shared with AA228. The training and testing are unique for CS229, while in AA228 we used reinforcement learning to achieve the same goal.

II. Related Work

The project is largely based on a study done by Ritcher et al., in which successful navigation in an unknown map with relatively high speed has been achieved by minimizing a specifically designed cost function.[1] Limited work has focused on this aspect of autonomous navigation, while the majority prioritizes safety over other features. A large number of studies have been conducted in regards to safety planning in static and dynamic environments. Fox et al. have examined the motion dynamics of the robot and their dynamic window approach achieved a vehicle speed of up to 0.95m/s.[2] Bekris and Kavraki have demonstrated a tree-based planner dealing with kinodynamic constraints with partial knowledge of the environment.[3]

The approach adopted in the project is fundamentally different from the methods proposed in this group of work in terms of design priorities. The absolute safety constraints are replaced with predictions of collision probability, which is similar to the concept proposed by Althoff et al.[4] The utilization of a collision penalty factor allows for the flexible trade-off between vehicle speed and algorithm conservativeness. Partially observable Markov decision process (POMDP) is a natural choice given the problem, as the vehicle doesnt have the knowledge of the surrounding environment in its entirety. They have been employed to ensure unmanned aircraft collision avoidance, as shown by Bai et al.[5]

III. Markov Decision Process(MDP) Model

The model used in this paper is the same as Charles'.[1] Here is a brief introduction of the MDP model used.

A. State Space S

A state $s = \{q \times m\}$ is composed with the vehicle configuration q and m that is the observation for the true map M from the vehicle position. The vehicle configuration, $q = \{x, y, \Psi, k, v\}$, contains the vehicle's position (x, y), heading angle Ψ, curvature (steering angle) k, and speed v. The local map m is a partial occupancy maps observed through radar range measurements at q. Both q and m are assumed to be measured perfectly.

B. Action Space A

The action space A is a pre-computed discrete action library spanning the vehicle's maneuvering capabilities. An

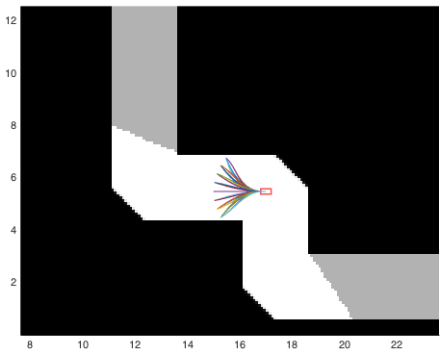

Fig. 1. Local map and possible trajectories in a hallway map with speed=1m/s and heading=180^o. The grey area indicates the unknown map

action $a = \{k1, k2, k3, v_c\}$ contains four parameters, where $k1$, $k2$, and $k3$ are three parameters determining the curvature profile which consists of two linear segments. v_c is the commanded speed. Since the vehicle is constrained by bounds on speed, acceleration, curvature, and curvature rate, not all parameters would generate a feasible action.

C. State Update

Given the vehicle configuration q_t at time t, an action a_t would determine the motion of the vehicle until the vehicle reaches q_{t+1} which is 2m away from q_t. Notice that the time between q_t and q_{t+1} could be different but the distance always remains 2m. Given q_t and a_t, a trajectory is then generated with 101 steps with same time interval by the methods introduce by Howard.[6] The local map m_{t+1} is obtained by simply perform another radar scan at q_{t+1}. By sampling target configurations on a 2m radius circle, we precomputed the possible actions and trajectories for for different vehicle configurations. Appendix B gives details of how we did the pre-calculation. Figure 1 gives an example of a local map and possible trajectories of a given vehicle configuration q in a hallway map.

IV. Date Set and Features

The training data is in the form of $\{(\vec{\phi}_1, y_1), (\vec{\phi}_2, y_2)...(\vec{\phi}_n, y_n)\}$, where $\vec{\phi}_i$ is the feature extracted for i^{th} data point, and y_i is the corresponding indicators of collision. A data set with n=50000 is generated from simulations on 5000 randomly generated hallway type maps with 10 data points on each map. The size of the map is 16mx32m with a resolution of 0.1m.

A. Feature Extraction

Given the action generated as described in III, we need to extract features from the trajectories resulting from the actions. Charles[1] uses a feature vector $\vec{\phi}(s, a) = [\phi_1, \phi_2, \phi_3, \phi_4]^T$ where ϕ_1 is the minimum distance between the trajectory and obstacle; ϕ_2 is the mean range to obstacle or frontier in the 60° cone ahead of the robot; ϕ_3 is the length of the straight free path directly ahead of the robot; and ϕ_4 is the speed of

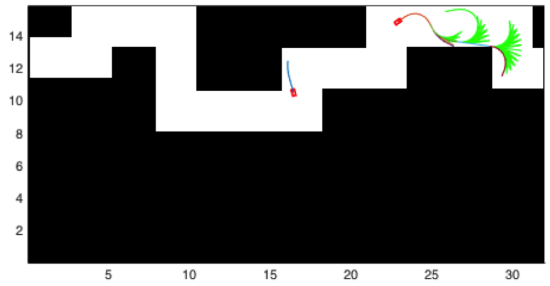

Fig. 2. Traing example: The left one indicates a y=1 (collison) case, and the right one indicates a y=0 (no collision) case

the vehicle at the end of the trajectory. All unknown area are treated as obstacles. Notice that ϕ_2 and ϕ_3 are extracted by averaging the two features of several points sampled from the trajectory. In this paper, we explored different ways of sampling including:(a)$\vec{\phi}_a$: 4 points uniformly spanned the whole trajectory; (b) $\vec{\phi}_b$:4 points uniformly spanned the second half of the trajectory; (c)$\vec{\phi}_c$: 4 points uniformly spanned the second half of the trajectory with higher weights for points closer to the end of the trajectory; (d)$\vec{\phi}_d$: One point from the end of the trajectory. While $\vec{\phi}_a$ seems to be a normal choice, $\vec{\phi}_b$, $\vec{\phi}_c$, and $\vec{\phi}_d$ are chosen with the belief that the points closer to the end of one trajectory has more relationship with whether the vehicle would come into collision in the future.

B. Training Procedure

At each training stage, before an action is performed, a feasible state $s_t = q_t, m_t$ and action a_t is first generated randomly in the training map. Feature $\vec{\phi}$ is extracted from s_t and a_t. Then if there exists any actions that could keep the vehicle from colliding in next three time steps, y is set to 0. Otherwise, y is set to 1. For a state where no actions could keep the vehicle from immediate collision, we call it $InevitableCollisionState$ with notation $ICS(q, m) = 1$.Figure 2 shows both a y=0 and a y=1 training case in one map. The pseudo code of this training procedure is shown as the following:

Algorithm 1 Train(M)

$q_t \leftarrow$ a random feasible configuration in M_i
$m_t \leftarrow Observe(q_t, M)$ update local observed map m
$A_t \leftarrow PossibleAction(q_t, m_t)$
$a_t \leftarrow$ random feasible action from A_t
$\phi \leftarrow ExtractFeature(q_t, m_t, a_t)$
$y \leftarrow LookForward(q_t, 3, M)$
Add (ϕ, y) to training data set D

Algorithm 2 $LookForward(q, depth, M)$

if $depth = 0$ **then**
  return 0;
$m \leftarrow Observe(q, M)$
if $ICS(q, m) = 1$ **then**
  return 1;
$A \leftarrow PossibleAction(q, m)$
for $a \in A$ **do**
  $q \leftarrow ConfigUpdate(q, a)$
  $y = LookForward(q, depth - 1, M)$
  if $y = 0$ **then**
    return 0;

V. METHOD

Similar to the procedure of the training algorithm, at each time step, a set of possible actions are generated based on the current configuration, observed map, and robot dynamics. Among these possible actions, the one that minimizes the cost function is selected as the next action to take:

$$a_t^*(s_t) = argmin_{a_t}\{J_a(a_t) + h(s_t, a_t) + J_c \cdot f_c(\vec{\phi}(s_t, a_t))\}$$

This specific cost function is composed of three terms representing respective time factors. The first term J_a represents the time duration of the current action a_t. The second term is the expected time-to-go characterized with a heuristic function, assuming the unknown map is obstacle-free and current speed is maintained until the destination is reached. The last one is a penalization term of the probability of a collision occurs as a result of the action. J_c is the collision penalty constant controlling the weights of collison possibility in action choosing, and $f_c(\vec{\phi}(s_t, a_t))$ is the learned collision possibility. The collision is modeled as a Bernoulli-distributed random event with beta-distributed parameter $\theta \sim Beta(\alpha, \beta)$, where α and β represent the prior collision pseudo count based on the feature $\vec{\phi}_d$. Then the collision of probability is learned as:

$$f_c(\vec{\phi}) = P(y = \text{"collision"}) = \frac{\alpha(\vec{\phi}) + \sum_{i=1}^{N} k(\vec{\phi}, \vec{\phi}^{(i)}))y^{(i)}}{\alpha(\vec{\phi}) + \beta(\vec{\phi}) + \sum_{i=1}^{N} k(\vec{\phi}, \vec{\phi}^{(i)}))}$$

where $k(\vec{\phi}, \vec{\phi}^{(i)}))$ is a kernel function measuring the proximity between query feature $\vec{\phi}$ and $\vec{\phi}^{(i)})$, which is:

$$k(\vec{\phi}, \vec{\phi}^{(i)})) = exp(-c \mid\mid \vec{\phi} - \vec{\phi}^{(i)}) \mid\mid^2)$$

For $N = 50000$ training data, we use $c = 4, N_{eff} = \alpha + \beta = 5$, such that $N_{eff} / \sum_{i=1}^{N} k(\phi, \phi^{(i)}) << 1$ in a familiar map and and $N_{eff} / \sum_{i=1}^{N} k(\phi, \phi^{(i)}) > 100$ in an unfamiliar map. The pseudo count α is calculated as:

$$\alpha(\phi) = \sum_{i=1}^{3} \frac{N_{eff} \cdot w_i}{1 + exp(7(\phi_i - 1.5D_{stop}(\phi_4))/(1.5D_{stop}(\phi_4)))}$$

where w_i is the weight parameters sum to 1. D_{stop} is the stop distance at current speed ϕ_4 and depends on the maximum braking acceleration a_{brake}:

$$D_{stop} = \frac{\phi_4^2}{2a_{brake}}$$

The pseudo code of the testing procedure is shown as the following:

Algorithm 3 Test(D,M,q_{start},q_{goal})

$q_t \leftarrow q_{start}$
$m_t \leftarrow Observe(q_t, M)$
$s_t \leftarrow \{q_t, m\}$
while $q_t \neq q_{goal}$ or $ICS(s_t) \neq 1$ **do**
  $A_t \leftarrow PossibleAction(q_t, m)$
  $a_t^*(q_t) \leftarrow argmin_{a_t \in A_t}\{J_a(a_t) + heuristic(q_t, a_t, q_goal, m) + J_c \cdot f_c(\phi(q_t, a_t, m), D)\}$
  $q_t \leftarrow ConfigUpdate(q_t, a_t^*)$
  $m_t \leftarrow Observe(q_t, M)$
  $s_t \leftarrow q_t, m$

VI. SIMULATION RESULTS AND DISCUSSION

In this section, we present simulation results from vehicle navigation in 20 randomly generated hallway maps with size of 16mx32m and resolution of 0.1m. We compared the performance of the baseline planner, where the collision probability is calculated just by prior using feature $\vec{\phi}_d$, and Bayesian Learning planners with different feature extraction methods talked in section IV-A in terms of success rate and average speed versus different collision penalty J_c. A simulation case at $J_c = 14$ using Bayesian Learning planner with different features and baseline planner is shown in 3. In this map, the baseline planner and the Bayesian Learning planner with feature $\vec{\phi}_d$ fails to navigate the vehicle to the goal location. Notice that both two planners use $\vec{\phi}_d$. This indicates that using ϕ_2 and ϕ_3 just at the end of the trajectory could not well characterize the vehicle's state. Since a short mean range to obstacle in 60^o ahead (ϕ_2) or a short straight free path length (ϕ_3) at the end of the trajectory does not necessarily mean there is a wall ahead, but could also mean unknown area ahead, in which case, a low collision possibility may be assigned.

The simulation results for success counts and average speed for all J_c tested in 20 random hallway maps are shown in Figure 4.

According to figure 4, as the collision penalty J_c increases, the success rate increases in general, and the average speed of all planners decreases. These results are as expected since J_c controls the scale of collision penalty. As J_c increases, the planner would become increasingly conservative. Due to computation limitation, the map number used to test each planner is 20 which rises the randomness in the test results. However, the trends shown in the results are clear. While planners using other feature vectors give flapping success rate, the success rate of Bayesian Learning using $\vec{\phi}_a$ remains 1 when J_c is larger than 14. Thus, with the safety as the first priority, the planner that performs the best in tests is the Bayesian Learning planner with feature $\vec{\phi}_a$ and J_c around 15, which gives the highest average speed while keeping the success rate as 1. This result is contradictory to our initial belief that points closer to the end should have more weights on predicting collision possibility. Similar reasons as explained in the case

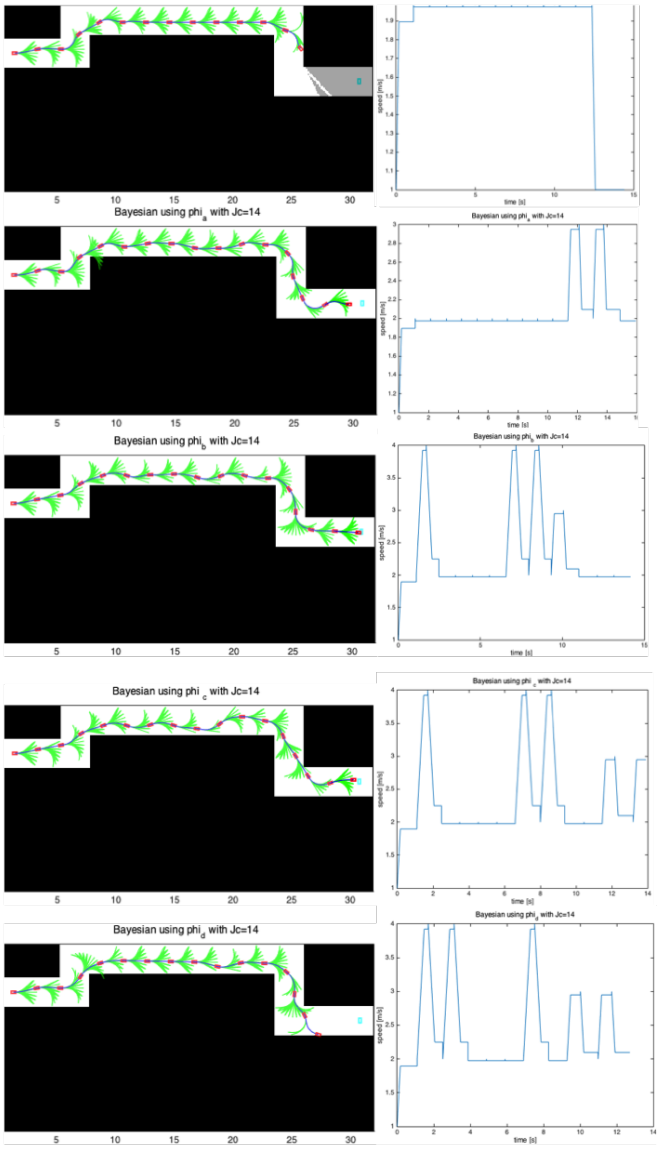

Fig. 3. A simulation case at $J_c = 14$ using Bayesian Learning planners and baseline planner. The blue lines show the "optimal" trajectories that the vehicle follows. The green lines show the possible trajectories at each configuration. The speed profile is shown on the right. The small peak in the profile reflects the discretization of the vehicle configuration.

with $J_c = 14$ could lead to this result. Since both the unknown area and obstacle often appear at the end of the trajectory, values of ϕ_2 and ϕ_3 are both small if we only sampling points close to the end of the trajectory. However, values of ϕ_2 and ϕ_3 on the other part of the trajectory could be different. If there is obstacle ahead at the end of the trajectory, it often means the vehicle make a turn in the straight hallway, in which case only the ϕ_2 and ϕ_3 at the points closer to the end would be small. On the contrary, if there is unknown area ahead at the end of the trajectory,in most of the case, the vehicle is entering a real turn in the map, in which case values of ϕ_2 and ϕ_3 are small for all points along the trajectory. Thus, using sample points locating around the whole trajectory behaves better on distinguishing whether small values of ϕ_2 and ϕ_3 are caused by obstacle ahead or unknown area ahead.

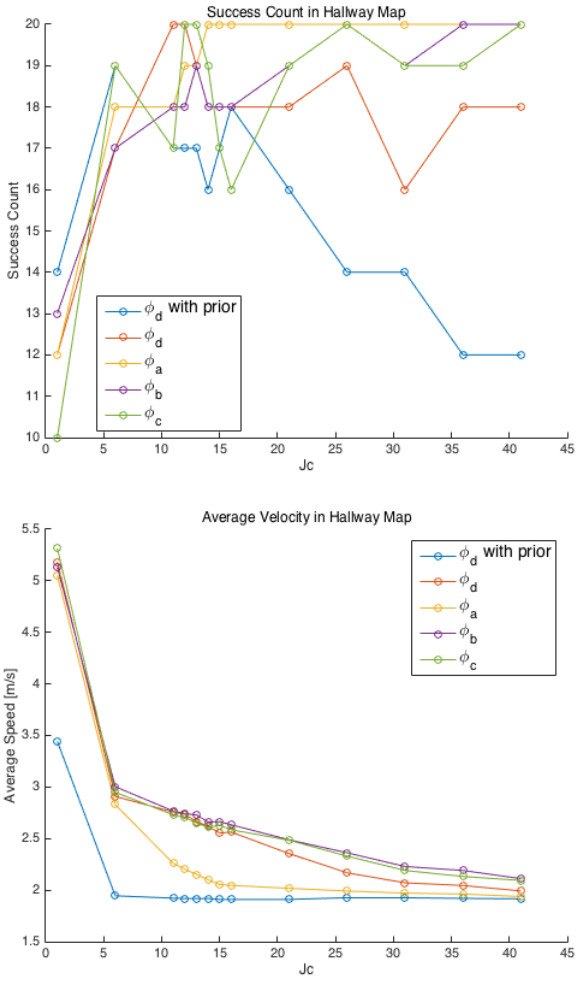

Fig. 4. Success count and average speed with respect to Jc in 20 randomly generated hallway maps

VII. Conclusion

In conclusion, we implemented the Bayesian Learning planner from Charles's paper[1] and our baseline planner for vehicle navigation in unknown maps. We modified the planner with different feature extraction strategies and found that the features extracted from sample points uniformly spanning the trajectory ($\vec{\phi}_a$) gives the best result in predicting the collision possibility. A possible reason is given in section VI saying that sampling around the whole trajectory dose better on distinguish the situation of obstacle ahead and unknown area ahead.

For future work, we think there are several aspects worth more research. First, we think more features and feature learning techniques could be used to further improve the feature extracting strategy. For example, instead of using average value of straight free path ahead and mean range to obstacle in 60^o cone ahead over different points along trajectory, we could treat the two values for each sample point as independent features and use feature selection techniques to choose the best sample points. Second, collision probability could also be learned through global approximation from the training data or through

reinforcement learning, thus saving much computational effort for real time application. The last idea is to study the effect of different training maps , such as forest and maze, on the performance and behavior of the planner during the test.

APPENDIX A
LIST OF SYMBOLS

Symbol	Meaning
x, y	vehicle position
Ψ	vehicle heading angle
k	curvature or steering angle
v	vehicle forward speed
q_t	vehicle configuration$\{x, y, \Psi, k, v\}(x, y)$
m	local map known to the vehicle.
M	the global true map
s_t	vehicle state $\{q_t, m\}$
A_t	set of possible actions the vehicle could take at time t
a_t	vehicle action at time t
$Ja(a_t)$	time consumed by taking action a_t
J_c	collision penalty

APPENDIX B
PRE-CALCULATION OF POSSIBLE ACTIONS

Given a vehicle configuration q, there are infinite number of actions could be taken. It is impossible to explore every possible actions during training and testing. Thus, we discretize the vehicle configuration and pre-compute several possible actions the vehicle could take for each configuration. We discrete the vehicle's speed with a resolution of 1m/s and curvature of resolution $0.1m^{-1}$, and for each configuration, we sampled 840 terminal states with different position, heading, and speed. First, we sampled 24 positions on a 2m radius circle with angle $\theta = \{-150^o : 30^o : -90^o, -80^o : 10^o : 80^o, 90^o : 30^o : 180^o\}$ with respect to the vehicle position. For each θ, 5 different speeds are uniformly sampled in the range of possible speeds could reach by the vehicle. Finally, 7 heading angles are uniformly sampled from $\theta - 90^o$ to $\theta + 90^0$. For each terminal state, we used Newton's method to solve for the action parameters.

ACKNOWLEDGMENT

We thank Professor Marco Pavone and Dr. Lucas Jason from Stanford University for the idea and advice for this research. We also want to specially thank Dr. Charles Richter from MIT Robust Robotics group for his generous help on this research.

REFERENCES

[1] W. V.-B. Charles Richter and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments," *International Symposium on Robotics Researc*, 2015.

[2] D. F. W. B. S. Thrun, D. Fox, and W. Burgard, "The dynamic window approach to collision avoidance," *IEEE Transactions on Robotics and Automation*, vol. 4, p. 1, 1997.

[3] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 704–710, IEEE, 2007.

[4] D. Althoff, J. J. Kuffner, D. Wollherr, and M. Buss, "Safety assessment of robot trajectories for navigation in uncertain and dynamic environments," *Autonomous Robots*, vol. 32, no. 3, pp. 285–302, 2012.

[5] H. Bai, D. Hsu, M. J. Kochenderfer, and W. S. Lee, "Unmanned aircraft collision avoidance using continuous-state pomdps," *Robotics: Science and Systems VII*, vol. 1, 2012.

[6] A. K. Thomas M. Howard, Colin J. Green, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments.," *Journal of Field Robotics*, 2008.