Proceedings of the 3rd International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/6/20230473

Autonomous driving in the crowded area

Haoran Zheng[1,5], Yishan Yu[2], Congzheng Wu[3], Zhuolei Chen[4]

[1]The Grainger College of Engineering, University of Illinois Urbana-Champaign, Champaign, 61820, Illinois, United States
[2]Trinity College of Arts and Science, Duke University, Durham 27708, United States
[3]Mechanical Engineering, Virginia Tech, Blacksburg 24060, Virginia, United States
[4]College of Arts and Science, Boston University, Boston 02215, Massachusetts, United States

[5]2992847732@qq.com.

Abstract. With the concept of letting the vehicle perform all the driving tasks without human intervention, autonomous driving has become a popular concept in recent decades. Our project focuses on safe navigation in a crowded environment by predicting obstacles, planning routes, and avoiding collisions. Our project successfully fulfilled our goal, and due to its customizable characteristics, it can also provide a base and platform for more complicated and advanced projects.

Keywords: autonomous driving, multiple obstacles, randomized environment, customizable.

1. Introduction

Society witnessed a prosperous development and rapid advance in electronic and computer technology in the last few decades, and artificial intelligence has become one of the most popular topics and promising fields. One interesting aspect of artificial intelligence application is the autonomous driving vehicle. Despite being far from prevalent, autonomous driving vehicles are expected to bring numerous merits to society once fully tested and certified, including by automating the vehicles, Human errors can be significantly reduced by automating the vehicles since drunken behavior and could avoid distraction. Autonomous vehicles can significantly decrease the mistake due to driver and inattention as the reason for an accident [1].

Developing a functional autonomous vehicle requires enormous time and effort, but the vehicle can be deconstructed to a combination of several platforms. The platform that our group focused on is the Open Software Platform, and the prediction, planning, and control section to be specific. An algorithm needs to ensure optimal real time planning of multiple vehicles (moving in either direction along a road), in the presence of a complex obstacle network [2]. For this project, our goal is to build a highly randomized environment with multiple obstacles for the autonomous vehicle to traverse.

Our project utilizes MATLAB as the coding language, and it incorporates different parts including prediction, safe planning, and safe control parts. These parts will work cooperatively to achieve our goal of safe navigation.

DOI: 10.54254/2755-2721/6/20230473

2. Method description

2.1. Concepts & environment set up

Even though autonomous vehicle manufacturers claim that their vehicles are equipped with specific automation levels allowing these vehicles to be safely driven on a particular road category [3], in a crowded environment, multiple obstacles, either moving or stationary, will present, greatly posing a threat to our vehicle. Therefore, our objective is to simulate an autonomous vehicle that can plan its routes, detect obstacles, and safely traverse through crowded environments while moving towards the goal.

For the convenience of visualizations and debugging, we decided to start setting up the proper testing environment first. The relative background information of testing environment is shown in the article *Testing Autonomous Vehicle Software in the Virtual Prototyping Environment* [4]. The fundamental three factors of the environment in our project are the initial position, movement behavior, and the number of obstacles. Throughout our research, we developed two versions of environmental design.

The first version is relatively limited since the environment is arbitrarily set by us: the number of obstacles, their initial positions, and movement behaviors are fixed unless we manually edit the entire code. Due to its limitation, this version is only used for early testing and visualizations.

In the second version, our environment is much more randomized and customizable. All of those three fundamental factors are randomly generated by the computer given a preset interval, so each time we run the simulation, the environments will be different and unique, which allows the user to simulate many more scenarios. In addition to those three main factors, minor factors like the size of the obstacle and goal position are also randomized. All those preset intervals can also be easily accessed and customized by the user, so this version is also more user-friendly (see below).

```
dt=0.1;
x0=[0;0];

numberOfObstacle=randi (8); % Number of obstacles in the scenario, can be changed by the user

obstacleInitialPositions=randi(20, 2, numberOfobstacle); % Initial positions of obstacles, the interval can be changed
obstacleInitialPositions=obstacleInitialPositions-10; % Offset to achieve negative initial position
obstacleMovementBehavior=randi(10, 2, numberOfObstacle);% Movement behavior of obstacles, the interval can be changed
obstacleMovementBehavior=obstacleMovementBehavior-5; % Offset to achieve negative movement direction

goal=randi(16,2,1); % Goal position, can be changed
goal=goal-8;
%goal=[3;4]; % If the user want to set a well-defined goal instead of random generation
kmax=100;
thres = 0.1;
dmin = 4;
ulim =[-5,5;-6,6]; % The mobility limit of our car, and the larger difference means higher maneuverability.
```

2.2. Vehicle modules overview & visualization

Our simulated vehicle is implemented with the double-integrator dynamics model, so our control space is linear acceleration. The details of the double-integrator dynamics model are described in the article *Consensus Algorithms for Double-integrator Dynamics* [5]. This dynamics model is able to simulate the

DOI: 10.54254/2755-2721/6/20230473

acceleration process that real-life vehicles have, but the limitation is that our vehicle will have nearly infinite angular acceleration, which is different from real-life counterparts.

There are three crucial modules for our vehicle to safely navigate through the obstacles: prediction module, safe planning module, and safe control module. The first task for our vehicle is observing the movement behavior of all obstacles and predicting their future positions in each time step with the constant-velocity prediction method. This section is done by the prediction module. After the prediction results come out, those data will be passed to the safe planning module, where our vehicle can plan a route according to the position of the goal and data from the prediction module. When the planning section is finished, the planned route will be manifested as a trajectory and sent to the safe control module. The safe control module will then pursue different intermediate points on the trajectory in each time step.

For visualization, our autonomous vehicle is presented by a red circle with a bold contour line. All obstacles are manifested by circles with different colors and thinner contour lines. Both the planned and actual trajectory of our vehicle and both the predicted and actual trajectory of the obstacles will be plotted, and the goal position will be manifested by a green dot.

2.3. Prediction module

Behavior prediction function of an autonomous vehicle predicts the future states of the nearby vehicles based on the current and past observations of the surrounding environment [6]. This module will only be used once per run to predict all obstacles' future position on each time step with the constant-velocity prediction, which is relatively simple compared to regression models, but since in our scenario the obstacles also move with the constant velocity, this simple model is sufficient. As a result, our autonomous vehicle would not require online adaptive prediction either. The most challenging part of this module is to store all predicted results on each time step for a random number of obstacles. To achieve this, we decided to separate the X position and Y position and store them in two different vectors respectively. The number of obstacles is not constant, so the size of these vectors is consequently not constant. Each roll in these two vectors represents the X/Y positions of a specific obstacle throughout the whole time interval, while each column represents all obstacles' respective X/Y positions on a specific time step. The whole process to build the vectors requires two-layered loops: the outer loop iterates on the time step, and the inner loop iterates on the number of obstacles. After finishing the two vectors, they will be passed to the safe planning module. The simulation will also plot the predicted trajectory.

2.4. Safe planning module

This module will be used only once per run to plan a route with optimization methods. The details and examples of optimization methods are concluded in the article *Simulation-Optimization Methods in Vehicle Routing Problems: A Literature Review and an Example* [7]. It will generate a reference path according to the position of the goal, and in most cases, the reference path is a straight line between the goal and our vehicle's initial position. Then it will modify the reference path by using the data provided by the prediction module and restructure these vectors to fit the sstack and lstack operations (see below). After a series of complicated reform operations regarding each obstacle's position at each time step, the reference path will be modified to the planned route that avoids obstacles (see Figure 1). However, the route is not necessarily safe for our vehicle due to the potential prediction error and the mobility limitation of the vehicle. To ensure the safety of the vehicle, we still needed real-time safe control. Therefore, the planned trajectory will be plotted and sent to the safe control module.

```
for k = 1:10
% The constraint
Lstack = []; Sstack = [];
for i=1: nstep
    for j=1: numberOfObstacle
```

DOI: 10.54254/2755-2721/6/20230473

```
    lstack = []; sstack = [];
    L = refpath((i- 1)*dim+1:i*dim)- obspathAllVector((i- 1)*dim+1:i*dim,j);
    L =-L'./norm(L);
    S = L*obspathAllVector((i-1)*dim+1:i*dim,j);
    1stack =[lstack;L];
    sstack =[sstack;S-margin];

    Lstack=[Lstack;zeros(1,(i-1)*dim) lstack zeros (1, (nstep-i)*dim)];
    Sstack =[Sstack;sstack];
    end
end
%% QP
newpath = quadprog (Qref+Qabs,-Qref*refpath cost,Lstack,Sstack,[],[],[],[],[],options);

cost_profile (k+1) = fun(newpath);
iterpath = [iterpath newpath];
if norm(newpath-refpath) < 0.5
    break
else
    refpath = newpath;
end
end
```

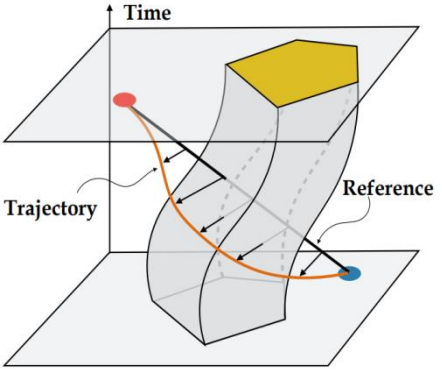

Figure 1. Optimization mechanism showcase.

2.5. Safe control module

This module will be called many times depending on how many time steps the user simulates. It determines the acceleration magnitude of our vehicle at each time step. By providing proper magnitudes for acceleration in X and Y directions, it can also determine the acceleration direction and thus indirectly modify the speed and positions of our vehicle. When the vehicle is in the safe condition, it should move directly toward the goal to achieve maximum efficiency, which is known as nominal control. However, in our scenario, goals are set to be the intermediate points on the trajectory planned by the safe planning module rather than the destination, and as time goes by, the goal will also be changing. Eventually, this route will lead the vehicle to the destination.

Even with the route from the safe planning module, sometimes nominal control alone is not sufficient to ensure the safety of our vehicle, so we needed the safe control to add another layer of protection. We utilized the energy cost function that penalized two values: the Euclidean distance between our vehicle and the obstacle, and the projection of the vehicle's velocity on the line that links the vehicle and that

DOI: 10.54254/2755-2721/6/20230473

obstacle. Those two values are calculated by one helper function each. If the Φ value of the energy cost function reaches above 0, the vehicle will consider itself in danger and modify its control in a way to minimize the Φ value. Since there are multiple obstacles to avoid, we decided to let our vehicle focus only on the most dangerous (with the highest Φ value) obstacle per time step. Because the Φ value will be updated each time, our vehicle will constantly change its avoidance priority and thus be unlikely to crash into other obstacles. To minimize the Φ value, we designed a method to run and test all possible combinations of acceleration in the X direction and Y direction and calculate their respective max Φ value among all obstacles at the current time step per combination (see below). Those max Φ values will then be compared with each other, and the combination that produces the smallest max Φ value will be used as the safe control for the current time step. When the Φ value returns to the negative values, the vehicle will switch back to its nominal control for maximum efficiency. Additionally, we also set a switch that the user can turn off the safe control and let the vehicle move with only nominal control, and this option would let users observe how the safe planning alone performs in safe navigation (see Figure 2). To achieve this function, the user can simply comment out the relative codes.

```
for u1=ulim(1,1):1:ulim(1,2) % 1 dim X轴上的control
for u2=ulim(2,1):1:ulim (2,2)% 2 dim Y轴上的 control
    u_tmp= [ul, u2]; % An arbitrary control constitute of ul and u2
    u_tmp=u_tmp+ur;
    % Evaluate the max phi at t+1
    xtp1=double_integrator (x, u_tmp, dt); % The next position and status of our car
    obslist_pred_tpl=obslist; % Get the predicted obstacle position at t+1(4,2),
    phis= [];
    for j= 1:numberOfObstacle % For each obstacle
        obs_tmp=obslist_pred_tp1(j,:); %j obs t+1 position
        phi_tmp=dmin*2-d (xtp1, obs_tmp) ^parameter1+parameter2*dotd (xtp1, obs_tmp);
        phis= [phis phi_tmp];
    end
    % phis use the u_tmp, for all obstcales, phi_i
    max_phis= [max_phis, max(phis)];
    length=length+1;
    controls= [controls; u_tmp];
end
end
```

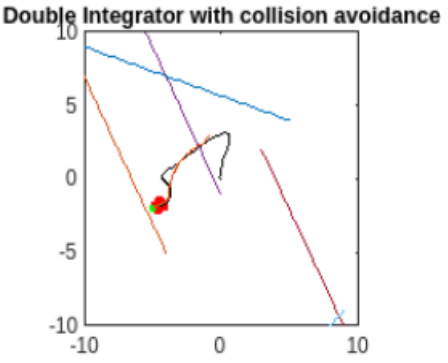

Figure 2. The car reaches its goal, the black line manifests the actual route with safe control, red curve manifest originally planned route without safe control.

DOI: 10.54254/2755-2721/6/20230473

3. Experimental result & future plan

The results fulfilled our goals: with both safe planning and safe control, the vehicle can always successfully avoid all obstacles while reaching the goal; with only nominal control, the route planned will not be safe for our vehicle for some cases where the obstacles move very fast toward our vehicle.

Nonetheless, our project has some limitations, and we found out many aspects can be improved. One important section deleted from our original plan was the machine learning part. The parameters of distance and velocity in our energy cost function are manually chosen by us, and it is not optimized for maximizing the survivability of our vehicle. We decided to reinclude this part into our project in the future. We plan to implement the CMA-ES (Covariance matrix adaptation evolution strategy) as the algorithm of machine learning. We would prepare a large specimen pool for the two parameters and randomly choose 10 specimens to run the simulation. Then, we would record the number of collisions for each specimen. The 3 specimens with the least collision cases will be chosen to be the parents of the next iteration. The specimens we choose in the next iteration will be centered around those three specimens according to Gaussian distribution. After several iterations, the number of collisions from the new specimens will reach a minimum and converge. Then we will stop the iteration and choose those values as the proper parameter for our energy cost function. For implementation, we expect to significantly modify the structure of our code while preserving most existent contents intact.

Other minor potential improvements include a more complicated prediction model and the implementation of online adaptation. Replanning can also help increase the efficiency of our vehicle.

4. Conclusion

Our project successfully builds up a crowded environment with multiple moving obstacles. The environments are highly randomized and can be easily customized by the user. With the implementation of the prediction, safe planning, and safe control modules, our autonomous vehicle can safely traverse through those obstacles and reach the goal. It is worth noting that the dynamics of the vehicle and obstacles are relatively simple, and further improvements are needed for simulating more complex scenarios. Since our project is not very complicated and advanced, we would like to allow others to use our code as the base to conduct their research and build more advanced projects.

References

[1] Bachute, Mrinal R., and Javed M. Subhedar. Autonomous driving architectures: insights of machine learning and deep learning algorithms. *Machine Learning with Applications* 6 (2021): 100164.

[2] Kala, Rahul, and Kevin Warwick. Multi-level planning for semi-autonomous vehicles in traffic scenarios based on separation maximization. *Journal of Intelligent & Robotic Systems* 72.3 (2013): 559-590.

[3] So, Jaehyun, et al. Analysis on autonomous vehicle detection performance according to various road geometry settings. *Journal of Intelligent Transportation Systems* (2022): 1-12.

[4] Kim, Baekgyu, et al. Testing autonomous vehicle software in the virtual prototyping environment. *IEEE Embedded Systems Letters* 9.1 (2016): 5-8.

[5] Ren, Wei, and Randal W. Beard. Consensus algorithms for double-integrator dynamics. *Distributed Consensus in Multi-vehicle Cooperative Control: Theory and Applications* (2008): 77-104.

[6] Mozaffari, Sajjad, et al. Deep learning-based vehicle behavior prediction for autonomous driving applications: A review. *IEEE Transactions on Intelligent Transportation Systems* 23.1 (2020): 33-47.

[7] Juan, Angel A., et al. Simulation-optimization methods in vehicle routing problems: a literature review and an example. *International Conference on Modeling and Simulation in Engineering, Economics and Management*. Springer, Berlin, Heidelberg, 2013.