Design and Implementation of a Small-scale Autonomous Vehicle for Autonomous Parking

Tianhao Yu, Wei Lu, Yanshen Luo, Chenguang Niu, and Wencen Wu

Computer Engineering Department
San José State University
San José, CA, USA
Email: {tianhao.yu, wei.lu, yanshen.luo, chenguang.niu, wencen.wu}@sjsu.edu

2021 6th International Conference on Automation, Control and Robotics Engineering (CACRE) | 978-1-6654-3576-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/CACRE52464.2021.9501311

***Abstract*—In this paper, we introduce the design and implementation of a low-cost, small-scale autonomous vehicle equipped with an on-board computer, a camera, a Lidar, and some other accessories. We implement various autonomous driving-related modules including mapping and localization, object detection, obstacle avoidance, and path planning. In order to better test the system, we focus on the autonomous parking scenario. In this scenario, the vehicle is able to move from an appointed start point to a desired parking lot autonomously by following a path planned by the hybrid A* algorithm. The vehicle is able to detect objects and avoid obstacles on its path, and achieve autonomous parking.**

I. Introduction

With the rapid development of autonomous driving technologies, vehicles on the street are becoming more intelligent with various kinds of autonomous driving capabilities including localization, perception, prediction, planning, and motion control. Without human interaction, autonomous vehicles have the ability to understand their environment with the help of sensors, such as cameras, Lidar, and radar. However, research on real vehicles associates with high cost including vehicle cost and high-cost sensors, which may pose challenges for researchers to develop and test autonomous driving algorithms. Thus, there is a need to develop low-cost and easy-to-integrate vehicles with high processing capability as the replacement to test various autonomous driving-related algorithms in lab environments[1], [2].

In this paper, we introduce the design and implementation of a low-cost, small-scale autonomous vehicle and develop an autonomous parking strategy to test the performance of the vehicle in a lab-based parking scenario. In addition to the basic components, the 1/20 scale autonomous vehicle is equipped with a Ydlidar G4 Lidar and a ZED stereo camera for object detection and recognition, and uses a Jetson TX2 as the processing unit. We implement various autonomous driving-related modules including mapping and localization, object detection, obstacle avoidance, and path planning.

Autonomous parking enables a vehicle to drive and park without any human interaction. There exist many literature introducing various autonomous parking strategies [3], [4]. To test the functionalities of the hardware components as well as the autonomous driving-related software modules of the vehicle, we build a simple autonomous parking scenario in lab and design a simple autonomous parking strategy for the vehicle. The autonomous driving scenario consists of eight parking lots, an entrance, and an exit, with some obstacles spreading over the parking lot. Hybrid A* algorithm [5] is used for path planning and YOLOv4 [6] is used for object detection. The experimental results show good performance of the vehicle.

The rest of the paper is organized as follows. Section II introduces the hardware and software architectures of the autonomous vehicle. Section III presents the proposed autonomous parking strategy. Section IV demonstrates the experimental results, and Section V concludes the paper and introduces future directions.

II. System Design

In this section, we introduce the system design of the small-scale autonomous vehicle, including the hardware architecture and software architecture.

A. Hardware Architecture

The hardware architecture consists of a vehicle chassis, an on-onbard computer, batteries, and sensors such as a stereo camera, a Lidar, and an IMU. The vehicle is separated into three levels to hold different components. The lowest level is the vehicle chassis, which came from the Traxxas 1/20 Scale Remote Control AWD Car. After removing all electrical components that were attached to the original chassis, the remaining basement serves as the base structure of the vehicle. The battery system and the automotive power system are placed on this level. The battery system is responsible for charging the Jetson computer and other sensors through USB connections. Fig. 1 illustrates the first level components. With four standoffs supported, the second level is equipped with a Jetson TX2 computing board. This embedded computer is designed for high-performance AI computing and contains both the graphic processing unit (GPU) and the central processing unit (CPU). It is able to deal with huge amounts of data and transfer data quickly. Fig. 2 shows the Jetson TX2 module and other components on the second layer. Next to the computing board, there is a ZED stereo camera installed

at the front of the second level. This camera is able to detect 3D objects and perform long-range 3D sensing and motion tracking. The Ydlidar G4 Lidar scanner that is used for object detection, obstacle avoidance, and mapping and localization is placed on the top level. It can rotate at a very fast speed and emit vertical rays to measure the surrounding distance. All the sensors transfer data to the computing board through a USB port. Once the computing board has collected data, it will transfer the data into signals and send it to the motion control unit (MCU). The MCU is responsible for controlling the vehicle moving and braking. Fig. 3 illustrates the sensors equipped on the vehicle and Fig. 4 shows the front and back views of the vehicle.

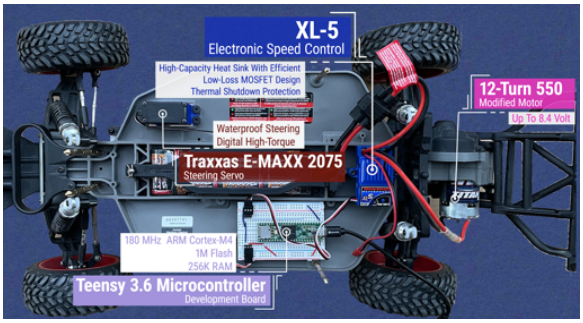


Fig. 1: First level control components.

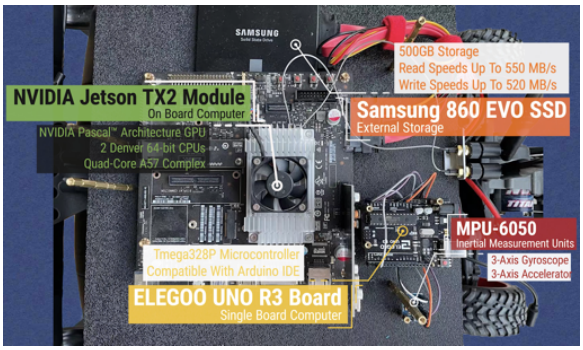


Fig. 2: Jetson TX2 module and other components on the second level.

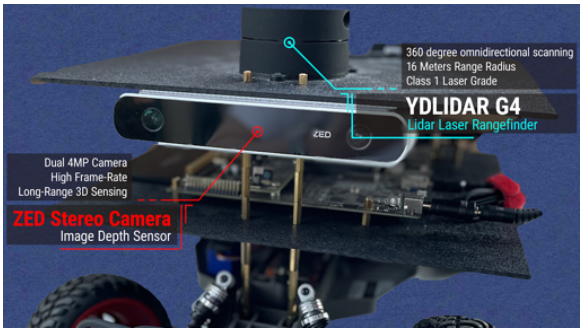


Fig. 3: ZED stereo camera and Ydlidar G4 Lidar on the vehicle.

Fig. 4: The front and back views of the vehicle.

B. *Software Architecture*

The base platform for the software is the ARM-based Ubuntu 18.04 Operating system. As illustrated in Fig. 5, the core of this architecture is Robot Operating System (ROS), which is the center to process, exchange, and coordinate the data from different autonomous driving-related modules, and issue decisions. The autonomous driving-related modules include localization, mapping, perception, prediction, planning, and control, which accept sensor inputs from the camera, Lidar, and IMU, and exchange information and data with ROS to achieve final decisions for the motion control for motors through Arduino.

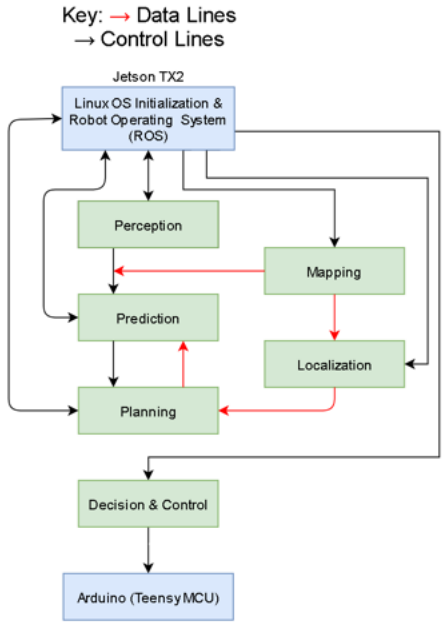


Fig. 5: Software architecture. Red lines are data lines and black lines are control lines.

III. Autonomous Parking

In this section, we introduce the autonomous driving-related modules that enable autonomous parking.

A. *Localization*

Given the hardware components, we adopt the laser scan matcher as our localization method. This method is based on

the PL-ICP [7] algorithm, which only uses Lidar data. PL-ICP utilizes the point-to-line metric to accelerate the iterative closest point method, which makes the method fast enough to deal with real-time localization. However, one disadvantage of this algorithm is that its performance will be hugely affected when taking large rotation. Fortunately, this large rotation will barely happen in our scenario (parking lot).

B. Path Planning

In our experiment, we implement the hybrid A* algorithm [5] as our planner to generate a drivable path given the starting point and endpoint coordinates in a fixed map. The hybrid A* algorithm is an advanced algorithm based on the traditional A* algorithm. Hybrid A* searches in a discretized grid by splitting the direction radius. Thus, Hybrid A* has a search space of (x, y, θ), while θ is the orientation. By introducing one more dimension (direction) than A*, hybrid A* could generate a path considering its start orientation and end orientation. We also take advantage of hybrid A*'s collision check and make sure the generated path could avoid the obstacles.The summary of the Hybrid A* algorithm is shown in Algorithm 1.

Algorithm 1: Path Planning: Hybrid State A*

Input: Starting State, Destination State, Local Map, Obstacles Info
Output: Path: a list of points
Pre-Compute a shortest-distance matrix H from each point to the destination by Dynamic Programming
while *A path to the destination is not found* **do**
  Select the state (x, y, θ) with shortest path cost from the priority queue as the current state
  if *a reed-shepp path is found* **then**
    Path is found and Stop the program
  else
    Push the neighbour states of the current state into the queue

C. Controller

For the controller side, we keep the car driving at a constant speed, thus, we only need to consider the lateral control. Considering that the results from the planner is a list of points along the planned path and the car knows where the next point it should go is, we adopt the simple open-loop controller. Based on the difference between the current point and the next point as well as the direction bias, we can easily calculate a turning angle ϕ as the driving command. Algorithm 2 illustrates the summary of the control module implementation. It is well-know that open-loop controller needs accurate localization and motion controller, and has accumulated errors. Thus, we will continue improving the controller design by using a PID controller and takes into consideration of the vehicle kinematic and dynamic models in the future.

Algorithm 2: Control module implementation

Input: The configuration file of the vehicle, The coordinate information of the parking lot map, Start point coordinate, End point coordinate
while *Vehicle status is normal* **do**
  if *The distance between vehicle current position and end point* $<$ *threshold* **then**
    STOP the vehicle ; `// Arrive at the end point`
  if *Path planner generates empty results* **then**
    STOP the vehicle ; `// No valid path`
  else
    while *Exist valid next point on the path* **do**
      if *Turning angle* $\phi == 0°$ **then**
        The vehicle go straight
      else if *Turning angle* $\phi > 0°$ *and Turning angle* $\phi <= 90°$ **then**
        The vehicle turn left
      else
        The vehicle turn right

D. Obstacle Avoidance

Obstacle avoidance is critical in autonomous driving [8]. We also enable the car to avoid obstacles. When the Lidar or the camera detects an obstacle, the position and size of the obstacle will be sent to the planning module and the obstacle will be labeled in the planner's map. Then, the planner updates the map and generates a new path that considers the obstacles. In this way, the vehicle could follow the new path to drive away from the obstacles.

With the basic principles, we set up a efficient workflow in our experiment. The Lidar will always be spinning to detect obstacles. Once it detects an obstacle on its way, it will hold on to the controller, and the camera will start to perform obstacle recognition. If the obstacle is recognized as a people, the vehicle will wait until the people moves away. But if the obstacle is recognized as a static obstacle such as a bottle or box, the controller will restart and the vehicle will follow the re-planned path generated by the planner. This work flow has an advantage of saving computational load. Opening the camera and running the perception model all the time will heavily weaken the computation unit's performance. Request once needed is a good compromise to our limited-resource situation.

E. Perception

The autonomous vehicle is equipped with a ZED camera, which is a stereo camera. Thus, we can implement object detection and depth perception using the camera. Considering the limited GPU performance on the NVIDIA Jetson TX2 on-board computer, we decide to adopt the YOLO series models as object detection module. The ZED and its SDK are natively

supported within the Darknet framework[9]. In our object detection module, we select one of YOLOv4 model, which is described in [6]. The authors made some modifications on YOLOv4 architecture. For example, it added Cross Stage Partial Network (CSPNet) to the original backbone(Darknet53) and created a new backbone named CSPDarknet53 to enhance learning capability. Besides, it adopted spatial pyramid pooling(SPP) and modified path aggregation network(PANet) as the neck in order to increase the receptive field of the network and enhance information propagation in representative pipelines respectively. Above modifications make the YOLOv4 has increased about 10 % accuracy compared with YOLOv3. Based on the object detection model, the ZED 3D camera can calculate the distance between objects and camera according to the detection results.

The summary of the autonomous parking algorithm we implement in experiments is shown in Algorithm 3.

Algorithm 3: Autonomous parking implementation

Input: The configuration file of the vehicle, The coordinate data of the parking lot map, Start point coordinate, End point coordinate

Initialization:

Start the vehicle Start the Localization, Perception, Planning, and Control modules

The planning module generates a path between start point and end point

while *Exist valid path and the vehicle status is normal* **do**

- **if** *The distance between vehicle current position and end point* $<$ *threshold* **then**
  - STOP the vehicle ; `// Arrive at the end point`
- **if** *Detects an obstacle* **then**
  - Calculate the distance to the obstacle and recognize the type of the obstacle ; `// Perception module`
  - Record the coordinate data of the obstacle on the map ; `// Using Lidar`
  - Update the next way point ; `// Path planner module`
- **if** *The next way point is NULL* **then**
  - STOP the vehicle ; `// No valid path`
- **else**
  - Move the vehicle to next way point ; `// Control module`

IV. Experiments

A. Experimental Setting

To test the performance of the autonomous vehicle and to verify the autonomous driving algorithms, we build a small parking lot as the experimental test-bed. Due to the limited resource, we use yellow tapes to make the boundary of the parking lot and each parking space. Besides, we use some cardboard and cut them to surround the parking lot boundary as the wall. Thus, the vehicle can use the localization algorithm to calculate its coordinates. Fig. 6 shows the parking lot layout with some obstacles inside. The entire parking lot is 4.28 meters long and 3 meter wide. It has an entrance, exit, and eight parking areas. Based on the size of the vehicle, we design the length and width of each parking area to be 0.7 meter.

To evaluate the performance of the planner, we develop an openCV drawing tool to generate a virtual parking lot map, which corresponds to the experimental test-bed and is calibrated accordingly. Fig. 7 shows the virtual parking lot map, where the entrance and exit are marked with "Entrance and exit". We set the lower left corner as the origin of the parking lot. Therefore, we can calculate coordinates for each line. This calibrated map is used by the path planning module to generate the path for autonomous vehicle.

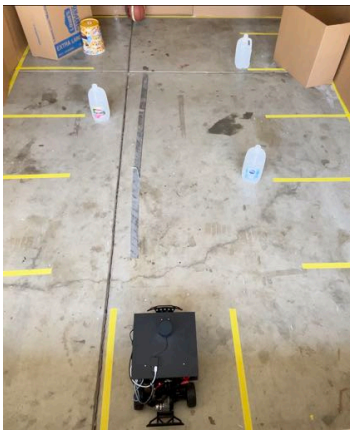


Fig. 6: Experimental parking lot.

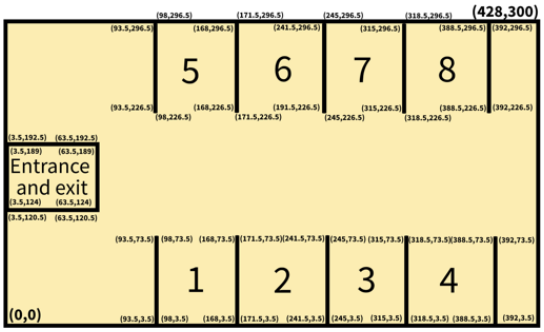


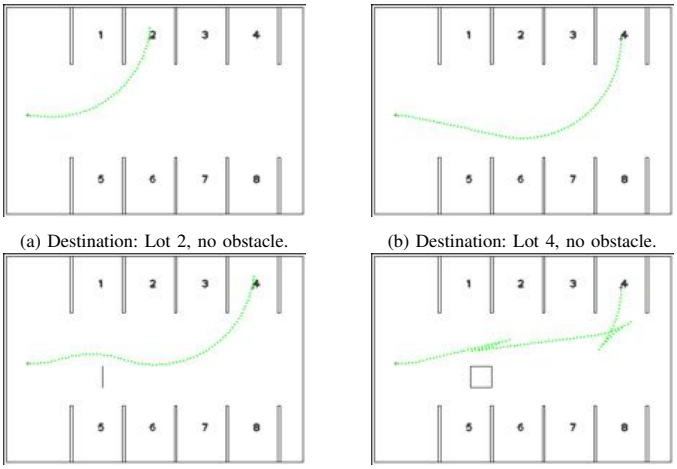
Fig. 7: Calibrated parking lot.

B. Results

We conducted many test runs to verify the proposed autonomous parking algorithm and the performance of the associated modules. In each run, we select one parking lot as the destination and let the autonomous vehicle start from the entrance. The Lidar scans the surrounding environment for obstacles, sends data back to the central processing unit. With the detected obstacles, the planning module plans the

correct route for the vehicle according to Algorithm 2. The control module commands the vehicle to bypass the obstacle and finish the remaining route according to Algorithm 1. In the meanwhile, the camera detects and recognizes objects in the environment, and the vehicle reacts based on the different types of the objects.

Figs. 8 (a)-(d) show the trajectories of the autonomous vehicle in four test runs, which we draw in the calibrated map. In Figs. 8 (a) and (b), the destinations are Lot 2 and Lot 4, respectively, without obstacles. We can observe that the vehicle moves smoothly to the destinations. In Figs. 8 (c) and (d), the destination is also Lot 4, but with obstacles of different shapes (the vertical line in (c) and the box in (d)). We can observe that the vehicle is able to move around the obstacle and reach the destination. In Fig. 8 (d), giving the size of the obstacle, the vehicle needs to adjust its motion by backing up twice in order to reach Lot 4, which demonstrates the robustness of our proposed algorithm.



(a) Destination: Lot 2, no obstacle.

(b) Destination: Lot 4, no obstacle.

(c) Destination: Lot 4, with an obstacle.

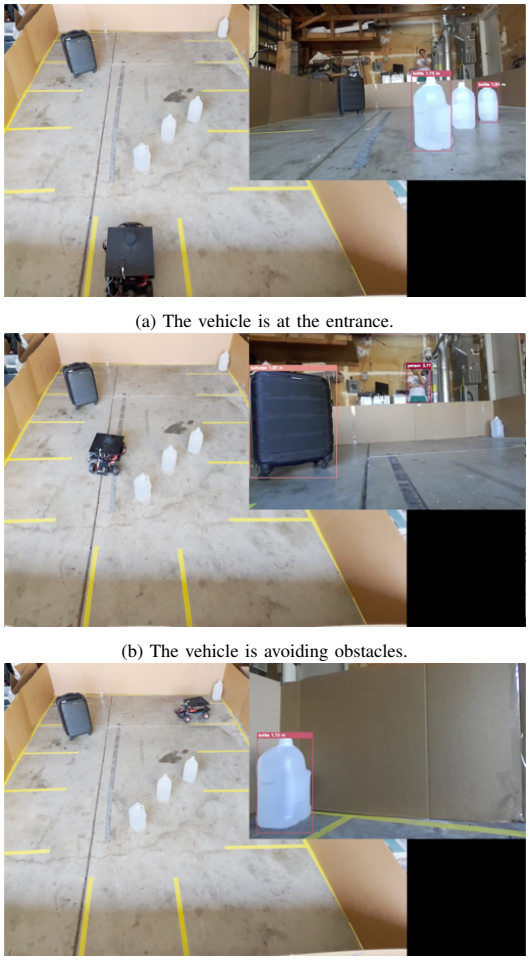
(d) Destination: Lot 4, with an obstacle.

Fig. 8: The trajectories of the autonomous vehicle in four test runs.

Figs. 9 (a)-(c) show three images of one test run with obstacles. As we can observe, the autonomous vehicle is able to detect and recognize obstacles of different types, avoid obstacles, and move smoothly to the destination (Lot 4).

V. Conclusions and Future Work

This paper introduces the design and implementation of a small-scale autonomous vehicle equipped with a Nvidia Jetson TX2 computer, a stereo camera, a Lidar, and some accessories. In the software side, various modules are loaded into the vehicle through ROS. To test the system, we focus on the autonomous parking scenario, in which the vehicle is able to move from a start point to a destination autonomously by following a planned path. The vehicle can detect objects and avoid obstacles on its path. In the future, we will work on improving the hardware architecture of the system within allowed budget, improving the autonomous parking algorithm, and testing in more complex parking scenarios.



(a) The vehicle is at the entrance.

(b) The vehicle is avoiding obstacles.

(c) The vehicle reaches the designation.

Fig. 9: One test run in the experimental test-bed. The upper right figures show the object detection results.

References

[1] R. Krauss, "Combining raspberry pi and arduino to form a low-cost, real-time autonomous vehicle platform," in *2016 American Control Conference (ACC)*. IEEE, 2016, pp. 6628–6633.

[2] W. Zong, C. Zhang, Z. Wang, J. Zhu, and Q. Chen, "Architecture design and implementation of an autonomous vehicle," *IEEE access*, vol. 6, pp. 21 956–21 970, 2018.

[3] X. Shen, X. Zhang, and F. Borrelli, "Autonomous parking of vehicle fleet in tight environments," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 3035–3040.

[4] W. Wang, Y. Song, J. Zhang, and H. Deng, "Automatic parking of vehicles: A review of literatures," *International Journal of Automotive Technology*, vol. 15, no. 6, pp. 967–978, 2014.

[5] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, 2008.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[7] A. Censi, "An icp variant using a point-to-line metric," *2008 IEEE International Conference on Robotics and Automation*, pp. 19–25, 2008.

[8] Y. Peng, D. Qu, Y. Zhong, S. Xie, J. Luo, and J. Gu, "The obstacle detection and obstacle avoidance algorithm based on 2-d lidar," in *2015 IEEE international conference on information and automation*. IEEE, 2015, pp. 1648–1653.

[9] J. Redmon, "Darknet: Open source neural networks in c," 2013.