OASD: An Open Approach to Self-Driving Vehicle

Sudip Dhakal, Deyuan Qu, Dominic Carrillo, Qing Yang, Song Fu
Computer Science and Engineering
University of North Texas
Denton, USA
{SudipDhakal, DeyuanQu, DominicCarrillo}@my.unt.edu
{Qing.Yang, Song.Fu}@unt.edu

***Abstract*—Building autonomous vehicles is an arduous task and requires a lot of technical understanding and cognizance. Grasping the essentials of such technologies and finding the perfect combination of hardware and software implementation is key to building autonomous vehicles. As we move towards autonomous vehicles, How do we find a solution to building an autonomous vehicle? To answer this question, we break it down into different hardware and software requirements. We present a hands-on solution for building autonomous vehicles by using an open-source platform Autoware which is based on ROS (Robot Operating System). Our initial implementation consists of detailing the overall system requirements, sensors calibration, point cloud map generation, waypoint generation and following, and finally vehicle control. We have also conducted the simulation test of localization and path following based on the point cloud map and waypoint generated. We believe our implementation provides a general understanding for building autonomous vehicles in the academic as well as in research field.**

***Index Terms*—Autoware, ROS, Autonomous vehicle, Calibration, Point Cloud Map, Waypoint, Simulation**

I. Introduction

Autonomous vehicle is an emerging field and is quickly becoming the next essence of mobility framework. From social life to industrial, autonomous vehicles are destined to be used extensively in the near future. Because they provide easy maneuverability, we can think of diverse applications that can benefit from the system. In terms of implementation, we can find very few approaches that provide an uncomplicated and basic overview of the entire autonomous vehicle system. Despite the overwhelming popularity of autonomous vehicles and well-studied architecture and approaches, not much is known about the details of the development of autonomous vehicles form ground level. In particular, there are very few papers on development of autonomous vehicles, concerning real problems of autonomous vehicles. These is largely due to the fact that most of research focuses on higher level of autonomous vehicle system and very few are concerned with the hands-on accumulation of hardware and software systems required for the development of autonomous vehicles. Hence, our goal is to provide a solution to building autonomous vehicles by breaking down the very basic software and hardware components, how they are used in the system and how they are in-cooperated with each other to build an autonomous platform. We define this solution from both a hardware and software perspective. The main contribution from this project are as follows:

• hands on manual for building autonomous vehicle system
• detailed instruction on usage of Autoware for sensor calibration, localization, waypoints generation and simulation
• SSC node for enabling control functionality on the autonomous vehicle

Our implementation of AVS (Autonomous Vehicle System) consists of high-end hardware components namely, AStuff-Spectra, LiDAR, Camera, PACMod (Platform Actuation and Control Module), and the vehicle. Similarly, software components used in our AVS are an open source autonomous platform called Autoware, Robot Operation System (ROS) among others. Usually different functional components of autonomous vehicles fall into sensing, computing, and actuation. Examples of sensing devices or sensors include cameras, LiDAR, radars and so on. For this purpose we are using Velodyne VLP-16 LiDAR, Flir BFLY-PGE-31S4M/C Camera. Our vehicle has PACMod integrated into the system, which allows to add "by-wire" control functionality for controlling vehicles and AStuff-Spectra as a computational platform. Similarly, Autoware is the open source software program used in our implementation. This platform has been extensively used in research and industry with automotive manufacturers considering it as a baseline for prototyping automated vehicles.The main reason for selecting these components as part of our implementation is their ease of use and availability. The open source community working for research and development of autonomous vehicles provide enough resource for all these components. Especially for Velodyne VLP-16 Lidar, Flir BFLY-PGE-31S4M/C Camera we can find abundant resources and forums dedicated to answering questions and queries from the research community. This provides ease for debugging and implementation of our platform.

II. System Overview

A. Vehicle and Sensors

"Mario" is our vehicle with various sensors and modules for enabling autonomous driving . This vehicular platform is a customizable vehicle with drive-by-wire functionality using a module referred to as Platform Actuation and Control Module (PACMod) [1][2]. Various features of the PACMod include:-

• CAN communication interface for user control and feedback

978-1-6654-4594-8/21/$31.00 ©2021 IEEE
DOI 10.1109/MetroCAD51599.2021.00017

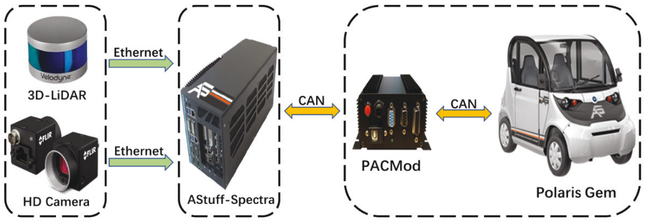


Fig. 1. Autonomous vehicle system model

• ability to physically control the vehicle via control of the actuators
• ability to send and receive CAN messages
• ability to update vehicle firmware via USB

For sensing the environment we are using Velodyne VLP-16 LiDAR. LiDAR scanners illuminate a target with pulsed lasers and based on the time taken to get the reflected pulses from the target/object, they provide an approximation of the distance to the object. We are using lidar to generate PCD (Point Cloud Data) which can be used to generate digital 3D representations of the different scanned objects as well as the surrounding.

Autoware supports different versions of Point Grey, Baumer, Allied Vision, and Generic UVC web cameras. Grasshopper 3 Point Grey and Ladybug 5 cameras are used to detect objects. The Ladybug 5 camera is Omni-directional, covering a 360-degree view, whereas the Grasshopper 3 is unidirectional running at a high rate. The first one can be used for detecting moving objects, and the latter one is used for recognizing traffic lights. Multiple Cameras are supported but they should be configured separately for individual purposes [3]. Flir BFLY-PGE-31S4M/C Point Grey camera is used as another sensing device in our system. We can use cameras for object detection as well as to recognize traffic lights.

B. Spectra

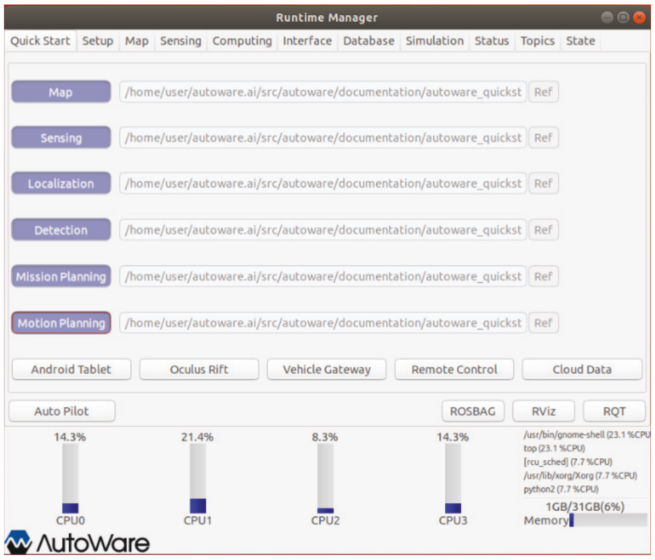
AStuff-Spectra is the world's first industrial-grade GPU computer supporting high-end graphics cards [4]. Because we are dealing with complex algorithms for autonomous driving, we need a high end computing device or computer that supports intensive computing and data parallel algorithms. This helps to increase the execution speed of those algorithms significantly.

C. ROS

Robot Operating System (ROS) is a set of computer operating system architecture designed for robot software development. It is an open source meta-level operating system (post-operating system) that provides operating system services, including hardware abstract description, step-by-step driver management, execution of shared functions, message transfer between programs, and program release package management. It also provides some tools and libraries for acquiring, building, writing and executing multi-computer fusion programs [5]. In this paper, we are using ROS 1 Melodic version.

III. Autoware

Autoware is an open source platform or software for autonomous driving technology, which provides algorithms and alternatives for perception, location mapping, detection and planning required for autonomous driving navigation, as well as an easy-to-use graphical user interface (GUI) package [6].In this paper, we are using Autoware.Ai master version. Autoware provides runtime manager as shown in Fig.2 which is the GUI that makes it easy for simulation and operation of autonomous driving vehicle. We can launch ROS nodes using runtime manager. Including the Autoware.ai version some of the Autoware related projects currently used are as follows:
• Autoware.ai: This is the first version of the Autoware and is developed on ROS 1. We can find a large group of researchers and open source community contributing to this project.
• Autoware.io: This is an extended version of Autoware also known as the interface project for Autoware. This is designed to be extended with proprietary vehicle software and other third-party libraries. This also allows us to work with variant source of device drivers for sensors as well as by-wire controllers for vehicle for example PACMod.
• Autoware.auto: This is a redesigned architecture of Autoware.ai based on ROS 2. This version of Autoware follows the accepted software engineering practices and industrial safety standards such as ISO 26262[7][8]. The modules and APIs used in this version of Autoware are similar but upgraded version of those used in previous version and they provide better support and overall performance to reproduce behaviors live and on development machines.



```
/home/user/autoware.ai/src/autoware/documentation/autoware_quickst
/home/user/autoware.ai/src/autoware/documentation/autoware_quickst
/home/user/autoware.ai/src/autoware/documentation/autoware_quickst
/home/user/autoware.ai/src/autoware/documentation/autoware_quickst
/home/user/autoware.ai/src/autoware/documentation/autoware_quickst
/home/user/autoware.ai/src/autoware/documentation/autoware_quickst

14.3%   21.4%   8.3%   14.3%
/usr/bin/gnome-shell (23.1 %CPU
top (23.1 %CPU)
[rcu_sched] (7.7 %CPU)
/usr/lib/xorg/Xorg (7.7 %CPU)
python2 (7.7 %CPU)
1GB/31GB(6%)
```

Fig. 2. Autoware Runtime Manager

A. Sensing

Autoware supports camera, LiDAR, radar, and GPS/IMUs primary sensors. These sensors helps to perceive the surrounding environment as well as driving scenarios. LiDAR

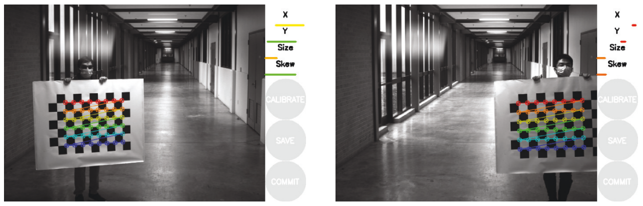


Fig. 3. X-axis Field of View

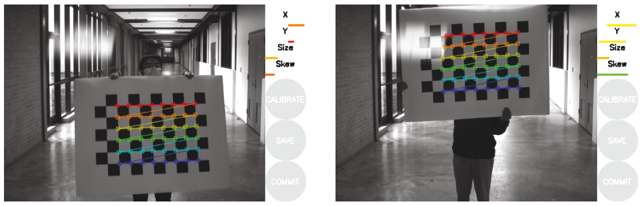


Fig. 4. Y-axis Field of View

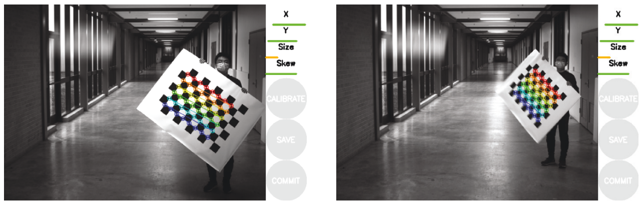


Fig. 5. Size Field of View

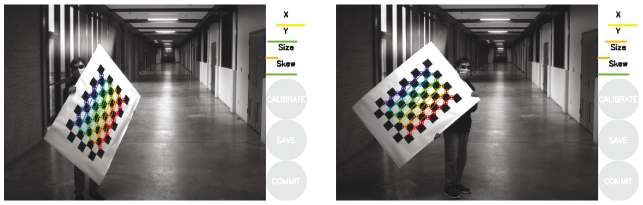


Fig. 6. Skew Field of View

scanners measure the distance to objects by illuminating a target with pulsed lasers and measuring the time of the reflected pulses [6]. Point cloud data obtained from LiDAR can be used to create a 3D map or representation of the surrounding environment. LiDAR sensor fused together with camera sensor can be used to create refined and more accurate 3D representation that can help not only to create a 3D map but also to recognize traffic lights, pedestrians, cycles and other features of the scanned objects.

B. Localization and Mapping

Localization is one of the most basic and important problems in autonomous driving. The reliability of the autonomous driving depends hugely upon localization. In autoware the localization problem is solved by using the Normal Distributions Transform (NDT) algorithm. To be precise, we use the 3D version of NDT to perform scan matching over 3D point-cloud data and 3D map data.3 As a result, localization can perform at the order of centimeters, leveraging a high-quality 3D Lidar sensor and a high-precision 3D map. We chose the NDT algorithms because they can be used in 3D forms and their computation cost does not suffer from map size (the number of points).Localization is also a key technique to build a 3D map. Because 3D Lidar sensors produce 3D point-cloud data in real time, if our autonomous vehicle is localized correctly, a 3D map is created and updated by registering the 3D point-cloud data at every scan. This is often referred to as simultaneous localization and mapping [9].

C. Detection and Tracking

It is of the upmost priority for autonomous vehicles to detect the surrounding objects such as pedestrians, traffic signals, lights, motorcycles, and other vehicles. The reliability and accuracy of autonomous system increases significantly as the number of detected objects increases. Various approaches supported by autoware for object detection include SSD, YOLOv2, YOLOv3 algorithms.

Since autonomous driving environment is volatile and unpredictable with frequent changing behaviour shown by the surrounding objects, tracking the trajectories of those objects is very important for decision making. These trajectories are input to the mission planning and path following modules and helps to determine the appropriate direction of the moving autonomous vehicle.

D. Mission Planning

Based on the input from the detection and tracking module of autoware and also the 3D map obtained through localization/mapping, path trajectories are determined through mission planning module of autoware. This modules also include navigation from the vehicle current position to the the destination.The 3D map obtained through localization and mapping module contains various features that can be used for mission planning. On the top of the route generated by the map navigation system, mission planner directs the autonomous vehicle to follow the centre lines of the lanes. Lane changes, turning and other important features of autonomous driving are facilitated by mission planning module.

E. Motion Planning and Path Following

Given the global trajectories, the motion planning module of autoware helps to generate local feasible trajectories. For motion planning search algorithms such as A*, hybrid A* are used in order to find the minimum cost path to the destination. For trajectory generation autoware uses trajectory generation algorithms such as lattice-based methods. Another notable method implemented by autoware is waypoints generation/follower for trajectory based path following. In this methods, waypoints are the trajectory points of the vehicle, a series of position information formed by matching the vehicle LiDAR data and the point cloud map. At first, the waypoints are generated using the 3D map obtained through mapping and saved in a file. This file contains the waypoints which can be used for path following.

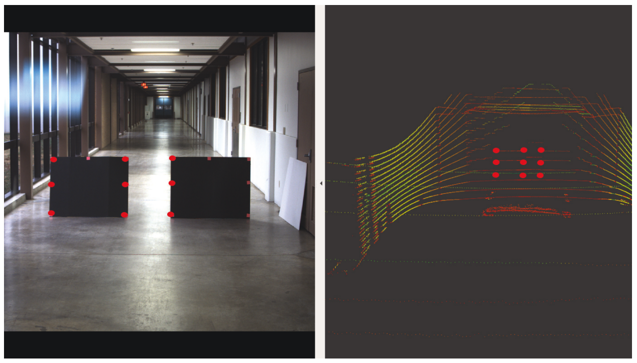


Fig. 7. Camera-LiDAR extrinsic

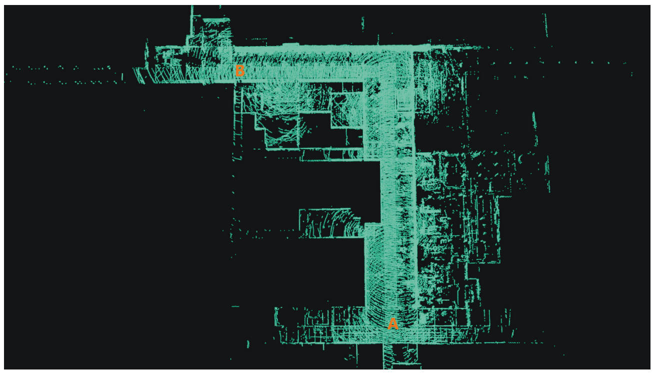


Fig. 8. University of North Texas DP building Point Cloud Data

IV. Sensor Calibration

Autonomous Vehicle Systems (AVS) generally relies on multiple sensors to perceive the environment. LiDAR sensor helps to perceive the surroundings for object detection, localization and mapping. Since, AVS are equipped with multiple sensors, leveraging the power of all the sensors and fusing them together can result in a more accurate and precise measurement as well as object detection. LiDARs are fused with cameras to improve the overall accuracy of 3D object detection. However, the fusion should be carried out carefully such that the perceived transformations from LiDAR and camera have been assigned to a shared coordinate frame so that we can fuse the sensor data. Hence, calibration of all the sensors equipped in the system is required. Camera calibration is done to find out the true parameters of the camera. These parameters also known as camera intrinsic parameters include focal length, principal point, lens distortion, projections and so on. Camera calibration helps to tune in these parameters. LiDAR Calibration, with respect to camera, is to estimate the LiDAR extrinsic parameters. Extrinsic parameters represents the geometric relation between camera and the LiDAR. Camera-Lidar extrinsic calibration is done to tune in the points in the 3D-LiDAR points plane and pixels in the image plane. Since we are using Velodyne-16 LiDAR and Flir camera, our calibration include both the camera intrinsic and extrinsic calibration.

A. Camera Intrinsic

For obtaining the camera intrinsic, we use the autoware camera calibration script provided by autoware. Like typical calibration methods, we are also using the chessboard (8x6). This is a simple method used for camera calibration and is very robust to implement.The method is robust over a range of illumination and a complicated background [10]. They have also proven in the past to yield accurate calibration results. The chessboard provides features that can be computed as intersection of lines.

This allows linear features to be detected accurately. Similarly, one advantage of using chessboard pattern over a traditional dotted pattern is that since we have line features, it is easier to infer distortion coefficient values, which is not possible in case of dotted pattern. Hence, using the Autoware camera calibration script along with the required parameters for size of chessboard and also the camera topic which is camera/image raw in our case we will obtain a camera stream. In order to get a good calibration result we will need to provide multiple field of view of the chessboard to the camera frame as shown in Fig. 3, 4, 5 and 6. We have 4 different field of views namely X, Y, size and skew such that:

• checkerboard on the camera's left, right, top and bottom of field of view
• X bar - left/right in field of view
• Y bar - top/bottom in field of view
• Size bar - toward/away and tilt from the camera
• checkerboard filling the whole field of view
• checkerboard tilted to the left, right, top and bottom (Skew)

As for the X- axis view we hold the chessboard in left and right direction as shown in Fig. 3. For Y-axis, we place the chessboard in top and bottom direction as shown in Fig. 4. For size, we place the chessboard towards and away from the camera sensor as shown in Fig. 5. Similary for skew, we place the chessboard in tilted manner as shown in Fig. 6.

B. Camera Extrensics

For obtaining the camera extrinsics, we use the Autoware camera LiDAR calibrator node. Executing this node provides a camera and LiDAR stream in two seperate windows. In autoware camera-LiDAR extrinsics are obtained by clicking on the corresponding points in the image and the point cloud. For camera instrinsic, the output file is obtained once all the field of views required are executed by the script.

However for camera-LiDAR extrinsics, we have to click on 9 different corresponding points in the image and the points cloud respectively. We used 9 sticky notes to pin 9 different points in the camera stream and corresponding points were identified carefully in the LiDAR stream and matched properly to finally obtain the camera-LiDAR extrinsic file i.e camera-LiDAR calibration. Our experimental setup for performing camera-LiDAR calibration shown in Fig. 7.

V. Localization Based on Point Could Map

Localization is the core module of autonomous driving system. The traditional method can use GPS/IMU for localization, but this method is not suitable for use in urban areas because the GPS signal will be weak, and the price will be relatively high if high-precision GPS sensors are used. Another method is to use conventional simultaneous localization and mapping (SLAM), SLAM determines the position of the current vehicle position by using the observed environmental features, but it is difficult to apply to the field of autonomous driving. Autonomous vehicles are fast and the road environment is complicated. In long distances driving, as the distance increase the deviation of SLAM localization will gradually increase.

Generally, for autonomous vehicles, it is best to achieve centimeter-level positioning accuracy under current road conditions. In actual localization, use the current LiDAR sensor scanning and the pre-built high-precision map to match the point cloud to determine the specific location of our autonomous vehicle on the map. Registration and positioning based on high-precision point cloud images and LiDAR usually account for a large proportion in the overall fusion localization due to their high accuracy and reliability, and are relatively reliable data sources in autonomous driving localization systems. Therefore, creating a LiDAR-based point cloud map is the first step in building a high-precision map [11][12].

A. *Record Rosbag for Point Cloud Data (PCD)*

We recorded the rosbag around our lab inside the University of North Texas (UNT) Discovery Park (DP) building. As you can see the Fig. 8, first we drove the vehicle from point A to point B slowly, and used the runtime manger provided by Autoware to record the rosbag file while driving. Since we only use LiDAR sensors, so just select */points raw* topic in the rosbag record interface and it will only save point cloud data based on LiDAR sensor.

B. *Normal Distribution Transform (NDT) Algorithm*

In order to find the current position from the point could map, usually we compare the point cloud scanned by LiDAR with the point cloud of the map. However, one of the problems is that the point cloud scanned by LiDAR may be slightly different from the point cloud from the map. The deviation here may come from measurement errors, or the change in scenarios (for example, pedestrians, vehicles). NDT algorithm is used to solve these subtle deviation problems. It does not compare the gap between the two-point clouds, but converts the reference point cloud (ie, high-precision map) into the normal distribution of multi-dimensional variables. If the transformation parameters can make the two scanning data match very well, then the probability density of the transformation point in the reference frame will be very large [13] [14].

Autoware provided two applications for NDT algorithm: NDT Mapping and NDT Matching. Usually, we use NDT Mapping for point cloud map generation, for localization we select NDT Matching. Both are based on NDT algorithm, their mainly difference are updating *target map*.

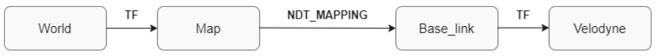


Fig. 9. Mapping Coordinate System

1) Normal Distribution Transform (NDT) Mapping: Before generating the actual point cloud map we have to understand the relationship between Autoware's mapping coordinate systems. There are four basic coordinate systems namely world coordinate system, map coordinate system, base link coordinate system and Velodyne coordinate system. Fig. 9 shows

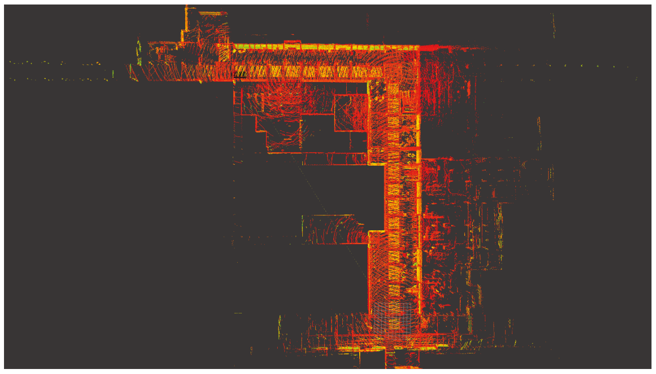


Fig. 10. Point Cloud Map Generation by using NDT Mapping

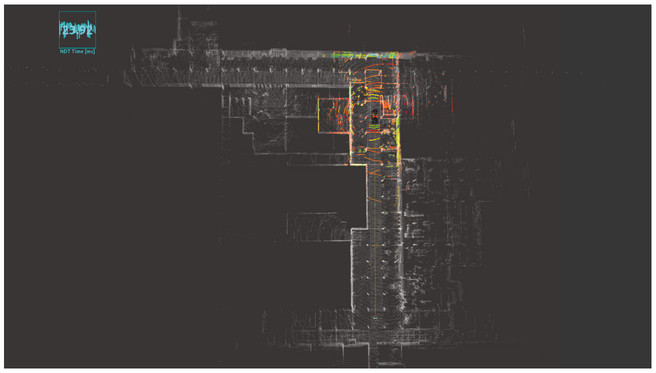


Fig. 11. Localization Simulation by using NDT Matching

the conversion relationship between them. For coordinate conversion from world to map and conversion from base link to Velodyne we can use the ROS TF from the runtime manager. As for the coordinate conversion from map to base link as shown in the Fig. 9, it requries a scan-to-map algorithm also known as Normal Distribution Transform (NDT) algorithm which is also provided by the Autoware runtime manager [15].

Hence, once the point cloud data collection is completed, we can make point cloud maps offline. The basic idea is to input the point cloud data firstly, that means to load the previously recorded rosbag file. Then as mentioned earlier Autoware uses the Normal Distribution Transform (NDT) Mapping to generate the point cloud map. The result of the point cloud map is shown in Fig. 10.

2) Normal Distribution Transform (NDT) Matching: NDT is the core localization algorithm of Autoware autonomous driving open-source project, which is also widely used in the field of real autonomous driving. After loading the point cloud map, NDT Matching will continuously receive the point cloud scanned in real-time and update the target asynchronously map, and then use the NDT algorithm to continuously matching to obtain the current position.

The objective function optimized by the NDT algorithm is mainly the similarity of the probability distribution of the input point cloud and the target point cloud. The computational complexity of this registration algorithm is positively correlated with two elements: Enter the point density of the point cloud and Bias of initial position estimation [16]. In this project we used voxel grid filter to down-sampling the density of the input point cloud, and we used the default initial position which provided by Autoware.

C. Simulation for Localization

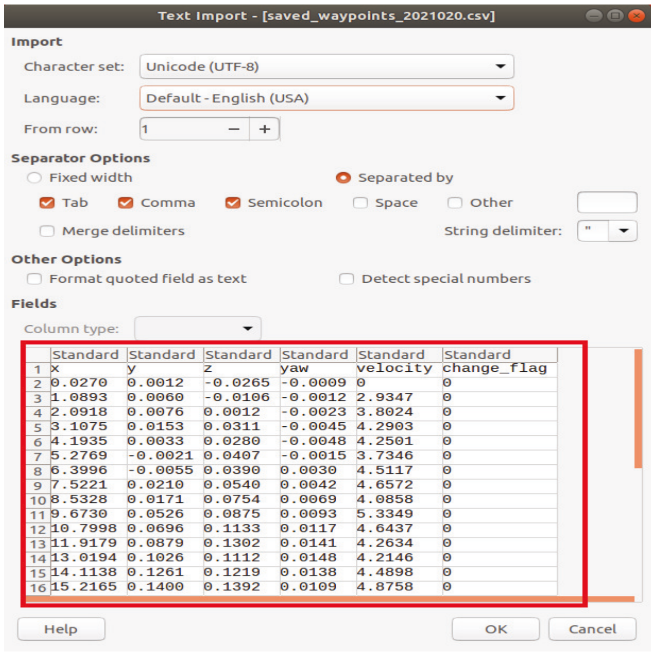
After constructing the point cloud map, we can perform the simulation test of localization based on the point cloud map. First, upload the generated PCD map file, and then select NDT matching in the runtime manager. Select voxel grid filter, vehicle pose node, and default initial position from runtime manager as well.

The simulation results shown in Fig. 11. In RViz, we can see whether the LiDAR point cloud matches the existing point cloud map after loading the NDT algorithm.

VI. Waypoints Generation and Planning

In this paper, waypoints are the trajectory of the vehicle, a series of position information formed by matching the vehicle LiDAR data and the point cloud map. So first we have to use the runtime manager to load the generated PCD map file. Then we also need to set the filter, select voxel grid filter in the runtime manager, this is a down-sampling method. For a given point cloud map, this method defines a 3D voxel grid.There are multiple points inside each voxel. Usually, the centroid or centre of all these points is used as approximation. This helps to reduce the number of points in points cloud map and makes the representation of the underlying surface more accurate and fast.

Then we have to choose NDT matching algorithm, because we need to convert the map coordinate system to the base link car coordinate system. Finally start the vehicle pose connect node and select the waypoint saver. Waypoints will be automatically generated after done of rosbag file playing [17]. Each waypoint has a specific value for velocity, x, y, z, yaw and change flag information. An example of saved waypoints file is shown in Fig. 12. In the given example x, y, z are values in the local East, North, Up (ENU) coordinate system. Yaw value stands for direction/heading in radians of a car along the lane. Velocity is target velocity for vehicle,


Text Import - [saved_waypoints_2021020.csv]

Import

Character set: Unicode (UTF-8)

Language: Default - English (USA)

From row: 1

Separator Options

Fixed width [✓ Separated by]

[✓ Tab] [✓ Comma] [✓ Semicolon] Space Other

Merge delimiters String delimiter: "

Other Options

Format quoted field as text Detect special numbers

Fields

Column type:

	Standard	Standard	Standard	Standard	Standard	Standard
1	x	y	z	yaw	velocity	change_flag
2	0.0270	0.0012	-0.0265	-0.0009	0	0
3	1.0893	0.0060	-0.0106	-0.0012	2.9347	0
4	2.0918	0.0076	0.0012	-0.0023	3.8024	0
5	3.1075	0.0153	0.0311	-0.0045	4.2903	0
6	4.1935	0.0033	0.0280	-0.0048	4.2501	0
7	5.2769	-0.0021	0.0407	-0.0015	3.7346	0
8	6.3996	-0.0055	0.0390	0.0030	4.5117	0
9	7.5221	0.0210	0.0540	0.0042	4.6572	0
10	8.5328	0.0171	0.0754	0.0069	4.0858	0
11	9.6730	0.0526	0.0875	0.0093	5.3349	0
12	10.7998	0.0696	0.1133	0.0117	4.6437	0
13	11.9179	0.0879	0.1302	0.0141	4.2634	0
14	13.0194	0.1026	0.1112	0.0148	4.2146	0
15	14.1138	0.1261	0.1219	0.0138	4.4898	0
16	15.2165	0.1400	0.1392	0.0109	4.8758	0

Fig. 12. Waypoint File Generated by Autoware

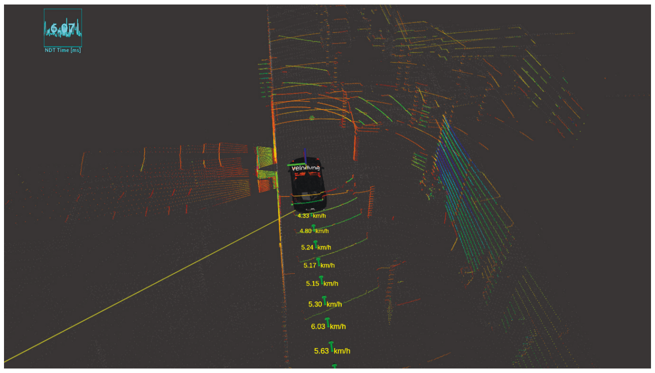


Fig. 13. Visualization for Waypoint Data

and change flag is basically stored as 0 (straight ahead), but if you want to change the lane, you can modify by yourself (1 turn right, 2 turn left) [18]. The Autoware runtime manager provides waypoint loader node through which we can visualize these waypoints in RVIz. We also select other options such as velocity-set, vel pose connect, pure parsuit, twist filter among other for meticulous visualization. Once the waypoint file is loaded and all the aforementioned nodes are selected we can run the motion planning simulation for visualization in RViz shown in Fig. 13. The waypoints data are clearly visible as the vehicle moves along the given path.

VII. Vehicle Control

A. Speed and Steering Control (SSC) to PACMod

There is no direct communication between Autoware and Polaries Gem vehicle, although the software of PACMod is also based on ROS. It can be seen from the Fig. 14 that the control signal sent by Autoware Speed and Steering Controller interface (SSC interface) node belongs to the high level command, but the control command required by PACMod node belongs to the low level command.

Based on the Fig. 14, We develop a Speed and Steering Control (SSC) node (written by C++) which acts as an interface for latitudinal and longitudinal control of vehicles [19]. This node helps to convert the high level control commands from Autoware SSC interfce node such as desired speed, desired curvature into low level control commands such as throttle, break and steering wheel control messages as required by the lower level drive-by-wire control system. We need to make this conversion because the PACMod only understands the low level commands and currently there are no ways to manually do the conversion. This is one of the main contributions from this paper.

B. PACMod to Vehicle

PACMod is a general-purpose electronic controller, which has been integrated into the vehicle wiring. It is an intermediate component that communicates original equipment manufacturer (OEM) sensors with external software. The onboard computer uses CAN to send commands and read sensor data from PACMod. This allows users to control and feedback through the CAN communication interface [20]. In addition, AutonomouStuff provided open source ROS package for developing [21]. Our final overview of Autoware Architecture and our Implementation is shown in Fig. 15.

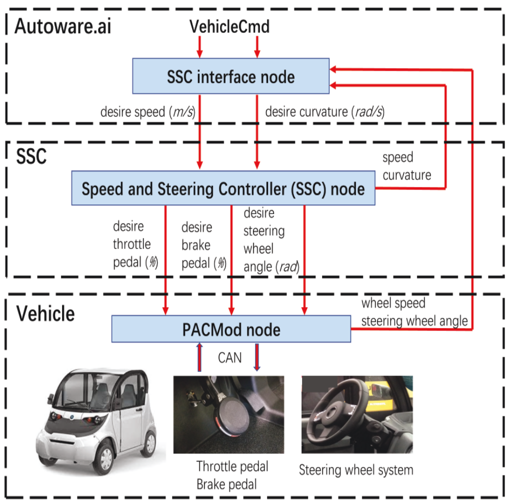


Fig. 14. Overview of SSC Structure

VIII. Future Work

The project presented in this paper is a foundation for building autonomous vehicle. The sections that we have presented here fulfils the initial requirement only. Improvement to each section including addition of other principal features of AVS are planned for future work. Object detection is a key

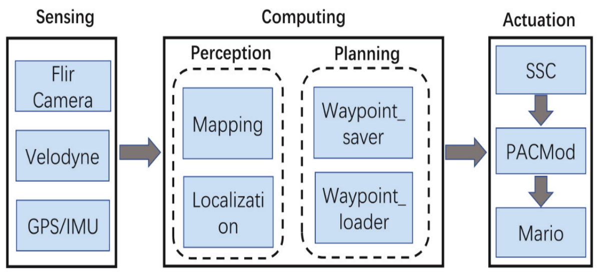


Fig. 15. Overview of Autoware Architecture and our Implementation

component and we plan to implement it along with obstacle detection and avoidance. In order to get more accuracy position in localization, we will add GPS/IMU sensor. In our future work we also plan to integrate the control functionality of the vehicle with sensing and computing. Currently our vehicle control system i.e. SSC node is a standalone node. We plan to blend SSC node with all the other nodes and functionalities required for AVS.

IX. Conclusion

This project expands the usefulness of the current solution to building autonomous vehicles. We have presented Autoware and Astuff-Spectra on board. This paper will provide a hands on solution to building autonomous vehicles. One step further from the available Autoware support we are adding a control functionality into the platform by building a SSC node, which will provide an interface for lateral and longitudinal control of the vehicle. We believe our implementation will provide a comprehensive and sturdy foundation for building AVS.

Acknowledgement

The work has been supported in part by the National Science Foundation grants CNS-1852134, OAC-2017564, and ECCS-2010332.

References

[1] URL: https://autonomoustuff.com/product/astuff-nev/.
[2] URL: https://autonomoustuff.com/product/pacmod/.
[3] Vysyaraju Manikanta Raju, V. Gupta, Shailesh Lomate, “Performance of Open Autonomous Vehicle Platforms: Autoware and Apollo” in IEEE 5th International Conference for Convergence in Technology (I2CT),2019.
[4] URL: https://autonomoustuff.com/product/astuff-spectra/.
[5] URL: https://en.wikipedia.org/wiki/Robot_Operating_System
[6] URL: https://www.autoware.ai/
[7] International Standardization Organization, “ISO 26262-10:2018 road vehicles – functional safety – part 10: Guidelines on ISO 26262,” International Standardization Organization, Tech. Rep. ISO 26262- 10:2018, 2018.
[8] A. Carballo, D. Wong, Y. Ninomiya, S. Kato and K. Takeda, ”Training Engineers in Autonomous Driving Technologies using Autoware,” 2019 IEEE Intelligent Transportation Systems Conference (ITSC), Auckland, New Zealand, 2019, pp. 3347-3354, doi: 10.1109/ITSC.2019.8917152.
[9] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, “An open approach to autonomous vehicles,” IEEE Micro, vol. 35, no. 6, pp. 60–69, 2015.

[10] Yu, Chunsheng Peng, Qingjin. (2006). Robust recognition of checkerboard pattern for camera calibration. Optical Engineering - OPT ENG. 45. 10.1117/1.2352738.
[11] URL: https://adamshan.blog.csdn.net/article/details/79230612
[12] URL: https://adamshan.blog.csdn.net/article/details/106739856
[13] URL: https://zhuanlan.zhihu.com/p/77623762
[14] P. Biber and W. Straßer, ”The Normal Distributions Transform: A New Approach to Laser Scan Matching,” in IEEE/RJS Intern. Conf. on Intelligent Robots and Systems, 2003.
[15] URL: https://www.cnblogs.com/hgl0417/p/11130747.html
[16] URL: https://adamshan.blog.csdn.net/article/details/106739856
[17] URL: https://www.cnblogs.com/hgl0417/p/11144203.html
[18] Tun, Wai Nwe, Sangho Kim, Jae-Woo Lee, and Hatem Darweesh. ”Open-Source Tool of Vector Map for Path Planning in Autoware Autonomous Driving Software.” In 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), pp. 1-3. IEEE, 2019.
[19] URL: https://github.com/Autoware-AI/autoware.ai/pull/1945
[20] Autonomoustuff, “PACMod 2.0 Startup Guide”, Version 2.2.0, 2018.
[21] URL: https://github.com/astuff