Master's Thesis of Engineering

Remote Lab: A Remote Autonomous Driving Application Testing Platform

리모트 랩: 원격 자율 주행 애플리케이션을 위한 테스트 플랫폼

August 2021

Graduate School of Computer Science and Engineering
Seoul National University
Computer Science

Rut Diane Cuebas

Remote Lab: A Remote Autonomous Driving Application Testing Platform

이창건

Submitting a master's thesis of
Computer Science

August 2021

Graduate School of Computer Science and Engineering
Seoul National University
Computer Science

Rut Diane Cuebas

Confirming the master's thesis written by
Rut Diane Cuebas
August 2021

Chair	하순회
Vice Chair	이창건
Examiner	김지홍

Abstract

Real-world autonomous vehicle research is often inaccessible to researchers outside of those related to automotive industry giants (i.e. Hyundai, Tesla) or automotive hardware manufacturers (i.e. NVIDIA, Bosch) due to the high baseline costs of creating even the simplest autonomous vehicle platform.

Remote Lab's primary function is to provide access to a remote experimentation platform to the general public, with a focus on users who may only have a beginner or intermediate level of knowledge in the field of autonomous driving. This work presents Remote Lab's architecture, its various features, and an overview of issues relating to the development of Remote Lab. Additionally, an in-depth analysis of the primary localization algorithm utilized on the platform, Normal Distributions Transform (NDT) matching, discusses the algorithm's weaknesses with respect to consistent and robust localization. ①

Keyword: Autonomous Driving, Remote Testing Platform, Self-Driving Car, Normal Distributions Transform
Student Number: 2019-28748

① The author of this thesis is a Global Korea Scholarship scholar sponsored by the Korean Government.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

1.1. OSCAR Platform and Remote Lab

OSCAR (Open-Source Self-Driving CAR) is an end-to-end autonomous driving application development platform with tools that aim to streamline the application development process. Currently, there exists no commercial tools that provide a full-stack solution for self-driving vehicle development.

The OSCAR platform consists of four major components: a DAG (Directed Acyclic Graph) system design tool, a design time validation tool, a code generation tool, and a remote experiment tool.

The focus of this work is OSCAR's remote experiment tool named Remote Lab. Remote Lab is a small-scale remote laboratory designed to mimic a real-life driving environment. Users can upload and run their autonomous driving applications onto the OSCAR Scaled Car and monitor the car's behavior through Remote Lab's web-accessible user interface.

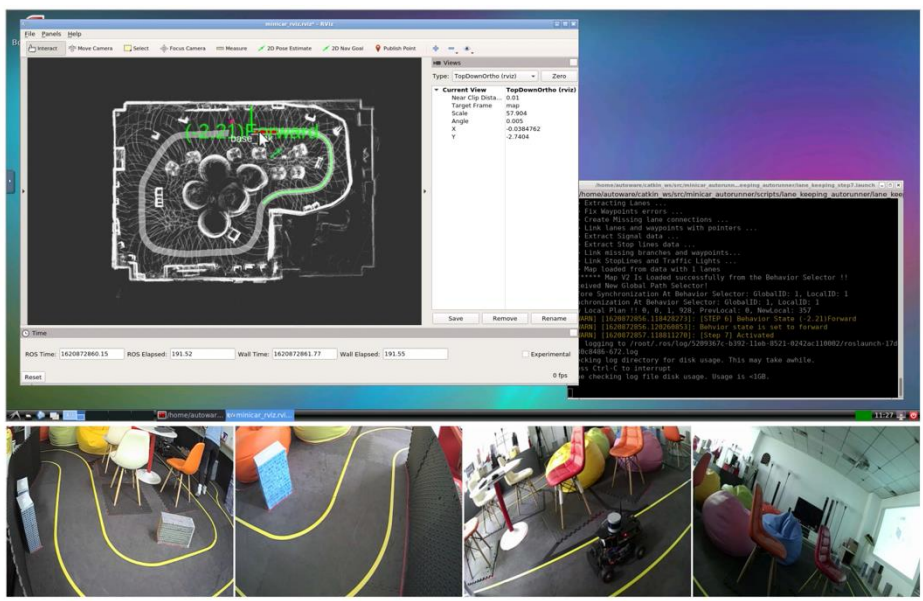

Figure 1: Remote Lab's web-accessible monitoring interface

Each user has access to a pre-configured default environment running on the OSCAR scaled car. Advanced users can load their preexisting applications into this environment and observe the car's behavior in real-time. Inexperienced users can utilize a one-click launcher that automatically executes all necessary commands to run a lane keeping application. These inexperienced users can easily alter application settings and observe changes in the car's behavior through the four available cameras.

1.2. Remote Lab's Motivation & Goals

The major motivation behind the creation of Remote Lab is to improve the accessibility of autonomous driving application development tools by creating a platform that can be used with only an internet connection and basic autonomous driving application knowledge.

One of the major obstacles autonomous driving developers face is the cost of real-world testing. Full-scale testing involves a baseline cost of the vehicle, sensors, actuators, and processing units. Testing using scaled down vehicles lower this baseline cost significantly, but the cost can still run upwards of $3000 for a one-tenth scaled RC car [1]. When considering individual, novice, or adolescent developers this cost remains a barrier to entry into the field. Remote Lab's motivation is to reduce this cost to zero for its users.

Another aspect of development that limits its accessibility is the difficulty of use of development tools for young students or beginner-level developers. Assuming cost is not a deterrent, developers still require a wide range of expertise across autonomous driving's numerous related fields. From hardware integration onto the vehicle, to the installation of middleware, frameworks, and dependencies, beginners will face difficulties before even starting application development. Remote Lab offers a pre-configured environment where users can begin experimentation in minutes, with no necessary installations or coding.

The final motivation behind the creation of Remote Lab is the need

for an intermediary testing stage between simulation and small-scale automotive testing. Simulators are not able to fully imitate a real-world environment, but many developers do not have the resources to create their own small-scale automotive testing environments. Hence, a halfway point between the two fills a need not currently addressed in the autonomous driving field. Users do not take on the burden of creating a small-scale environment but can test their applications using real-world input.

To summarize, the goals of Remote Lab include: to increase accessibility to autonomous driving development tools and real-world testing, to introduce students and beginners to autonomous driving application development tools, and to create an intermediary between simulators and small-scale automotive testing.

1.3. Organization

The organization of this paper is as follows. Chapter 2 gives an overview of Remote Lab's architecture and details all available features. Chapter 3 discusses Remote Lab's development challenges, including an in-depth analysis of the primary localization algorithm, Normal Distributions Transform (NDT) matching, where the algorithm's weaknesses are covered. Chapter 4 concludes this paper and discusses any future work.

Chapter 2. Remote Lab Features

2.1. Architecture Overview

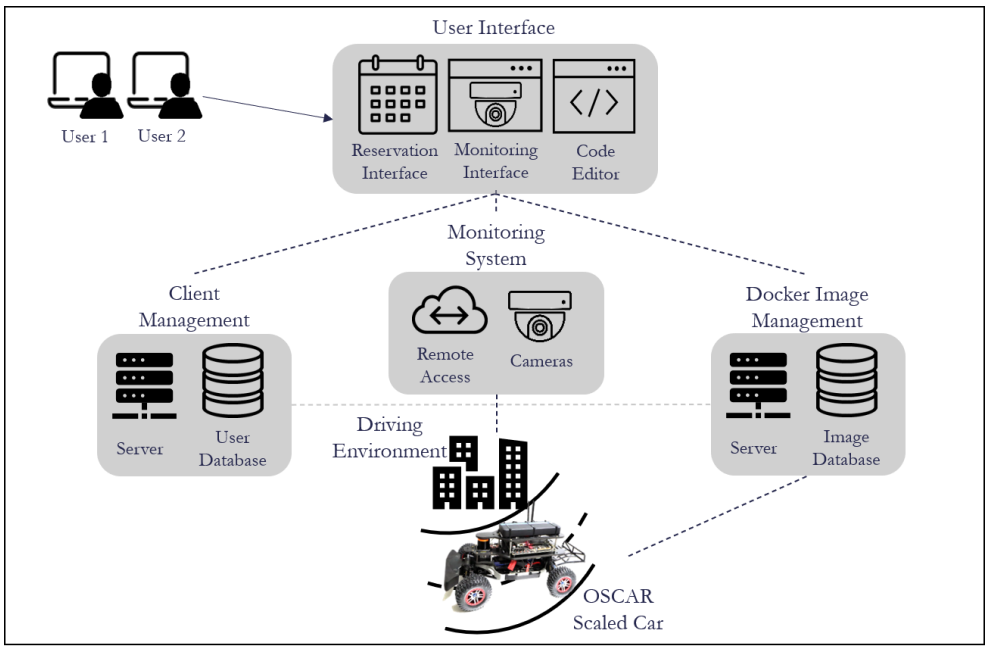

Figure 2: Remote Lab Architecture

Remote Lab consists of several major components: the OSCAR Scaled Car, the Driving Environment, Client Management, Docker Image Management, the User Interface, and the Monitoring System.

The OSCAR Scaled Car is a one-tenth scale RC car outfitted with a full sensor suite and appropriate software stack to provide a fully functional autonomous driving platform.

The Driving Environment is a physical enclosure that houses a one-tenth scale track intended to resemble the real-world. It includes numerous scaled buildings of various sizes that are placed alongside the track to provide landmarks for localization and object avoidance. This environment also includes several types of obstacle objects, ranging from traffic signs to traffic cones and barriers.

Client Management handles all user data using a server to service requests between the User Interface and the User Database, as well as requests between the User Database and the Docker Image

Database.

Docker Image Management consists of a Docker Image Database and server which handle the storage and dispatch of the users' docker images onto the OSCAR Scaled Car's processing unit. Each Docker image is a fully isolated environment that is capable of running as a Docker container on the car. From within the container the user can launch any preloaded applications, collect sensor data or create their own applications.

The User Interface is the user's point of access to Remote Lab. From there a user can execute any and all necessary actions for interacting with Remote Lab's numerous features. It consists of three major components: the Reservation Interface, the Monitoring Interface, and the Code Editor. The User Interface is entirely accessible through the OSCAR homepage and functions well with a standard internet connection.

Lastly, the Monitoring System is the system by which the Driving Environment can be viewed remotely, and the OSCAR Scaled Car's processing unit can be accessed remotely. The Monitoring System is visualized by the User Interface's Monitoring Interface.

2.2. Remote Lab User Flow

A typical user flow of a Remote Lab user is as follows. The user will begin by logging in their username and password credentials. They will next navigate to the 'Reservations' tab of their homepage and select their desired reservation date and time. After requesting a reservation, the user must wait for approval from a Remote Lab administrator. After approval is granted, the user is notified and their reservation request status changes from 'pending' to 'approved'.

Any time before their reservation novice developers and younger users are recommended to utilize the Code Editor provided to alter configuration values for the preloaded lane keeping application. Users can interact with a game-like interface where they are able to drag and drop various interlocking blocks to create a configuration file and change parameter values. After clicking the 'Generate Code' button

a user can see the code they have altered. When satisfied with the generated code the user can click ‘Push to Registry’ to push a newly created Docker Image that contains their altered parameters to their own personal Docker registry. A user can confirm the successful addition of their new image on their homepage.

Before reservation time the user can then select which Docker image they want to load onto the OSCAR Scaled car’s processing unit for the duration of their reservation. The ‘Select Docker Image’ tab on their homepage will list the available default image, and the image previously altered by the Code Editor.

At reservation time the user will go to the ‘Remote Connection’ tab of their homepage and be redirected to the Monitoring Interface. After loading the CCTV cameras, the Remote Connection Interface will prompt the user for their reservation password.

After inputting the correct reservation password, the running container of the previously selected Docker image will be visualized. This isolated environment is running atop the OSCAR Scaled Car’s processing unit and has all sensors and actuators connected and configured.

Inexperienced users can utilize the one-click launcher located on the container’s desktop to automatically execute all necessary commands to run a lane keeping application.

After clicking the launcher, a terminal will appear, followed by a visualization tool that will display a map of the Driving Environment. After all modules are loaded the car’s location will appear on the map as a red and green marker. A light gray line indicating the track’s path will also appear. The user then clicks the ‘2D Nav Goal’ button and selects the desired goal point. Path planning modules are launched automatically, along with a control module with the parameters specified by the user while utilizing the Code Editor.

The user can observe changes in the car’s movement through the four live camera feeds located at below the Remote Connection Interface.

At the reservation’s finish time, all changes made by the user are committed to the running container and a new Docker image is created

with these changes. The new image is pushed to the user's personal Docker registry and can be accessed for future reservations.

2.3. Reservation Request

As seen in Figure 4, Remote Lab provides a Reservation Request feature where users can view a calendar of Remote Lab's upcoming reservations, select an unclaimed interval of time, and send the Remote Lab administrators a request to reserve access to the OSCAR Scaled Car and the Driving Environment.

The calendar provides a clear and concise visual indicator of what times are available for the user to reserve. Reservations are displayed as blocks of time, with the username of the person who has reserved it. Before receiving approval from administrators, the reservation will be displayed as a gray block and the reservation status will read as 'Pending'. Upon approval the block will change to green and the status becomes 'Approved'. Users are alerted of a reservation approval or denial by notification banners on their homepage, as seen in Figure 3. Users are also able to cancel their reservation at any time.

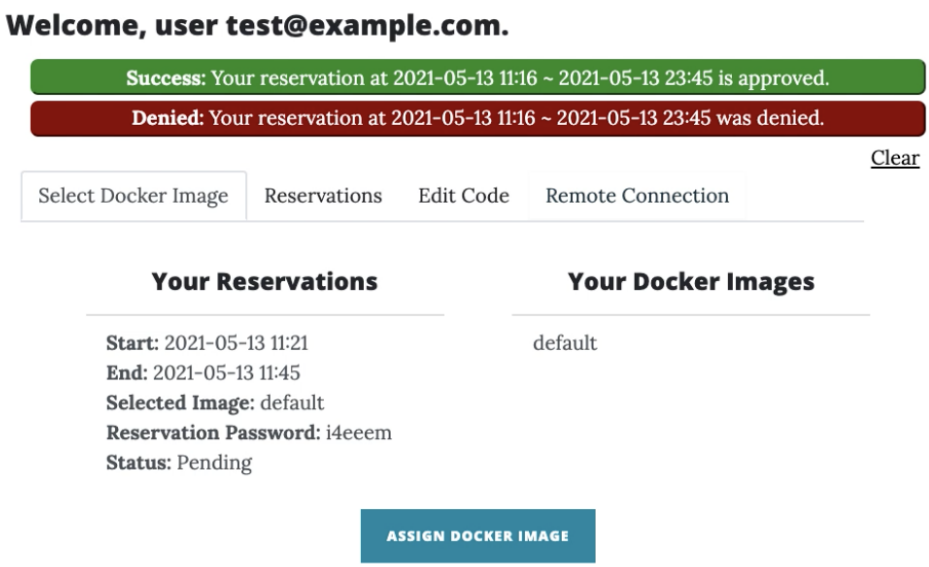

Figure 3: 'Select Docker Image' tab of the user's home page and approval/denial notifications

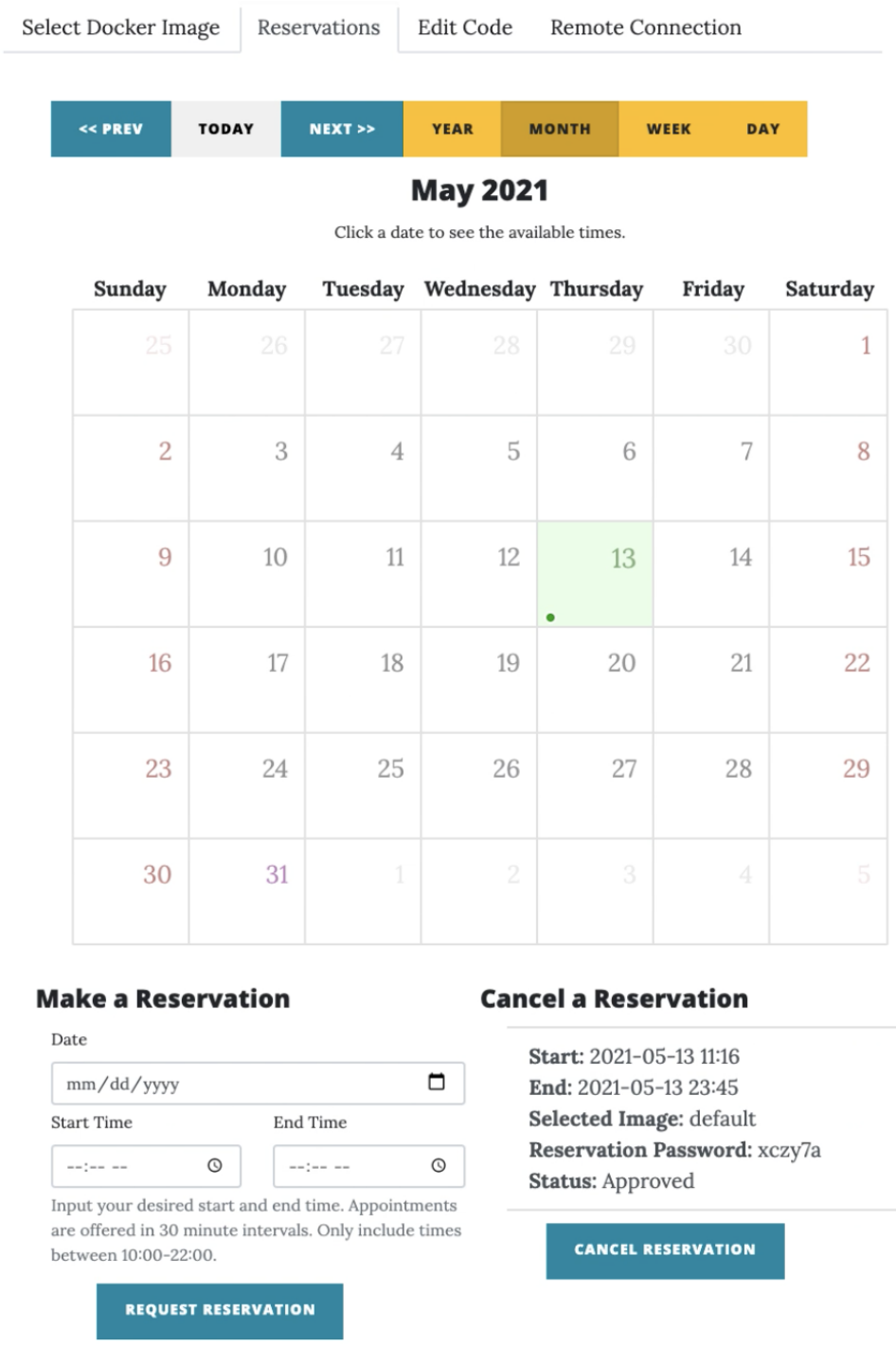

Figure 4: Reservation Interface Calendar and Request form

As seen in Figure 3, every reservation has the following fields: Start, End, Selected Image, Reservation Password, and Status. Start and End are the reservation's starting and ending times. Selected Image is the name of the Docker image that is scheduled to be loaded onto the OSCAR Scaled Car processing unit at the start of the reservation. Reservation password is the password needed to access the Remote Connection Interface. And lastly, the Status field indicates whether or not the reservation has been approved or is still pending.

In Figure 3, the interface where the user can change the Docker image is displayed. Under 'Your Docker Images' a list of all Docker images previously pushed onto the user's Docker registry are shown. To the left, 'Your Reservations' list all of the user's current reservations. To change the 'Selected Image' field of the reservation, the user can click their desired reservation, and Docker image from the lists, and click 'Assign Docker Image'. This alteration can be done up to five minutes before the reservation time. It allows users to load their own self-defined environments onto the OSCAR Scaled Car, the default pre-configured environment we have created, or any altered environments created using the Code Editor or from previous Remote Lab reservations.

2.4. Code Editor

The Code Editor provides a block-based visual programming interface that removes the complexities of code creation and enables users with little to no coding experience to generate error-free code. Users can access premade blocks, each with their own function, that apply changes to the default image.

The Code Editor consists of three components: the Toolbox, the Workspace, and the Code Textbox.

As shown in Figure 6, the Toolbox holds all available blocks, each which alter the provided default Docker image in a different way. Blocks are dragged from the Toolbox to the Workspace in order to be activated. In the workspace, block fields can be edited, blocks can be

rearranged, and blocks can be deleted.

The Code Textbox, located below the Toolbox and Workspace in Figure 5 displays the code generated by the active blocks in the Workspace. Here the user can see how changes in their workspace affect the generated code.

At the time of this paper there currently exists two types of blocks: the configuration file block, and parameter blocks. The configuration file block pictured in Figure 6 represents the default Docker image's control module launch file (see Section 2.6.). Within the configuration file block, any number of parameter blocks can be inserted. Parameter blocks allow the user to specify a value for any parameter included in the configuration file. For example, the max velocity of the car when running the default Docker image's lane keeping module can be defined using the parameter block labeled 'Max Velocity' and inputting a new value.

When the user is satisfied with the code generated in the Code Textbox, they can choose to create a new Docker image with this code. The new Docker image will appear under the 'Your Docker Images' list shown in Figure 3.

The Code Editor utilizes Google's JavaScript library for building visual programming editors named Blockly [2]. At its base the Blockly library provides an editor that can be injected into any existing web page. Developers then define their own blocks, implementing the appropriate functionalities. The Blockly library includes an additional editor, called the Block Factory, where developers can design the block's visual features, shape, input type, and more [3]. A generator stub can then be created once the developer is satisfied with the block's appearance and interaction with other blocks. The generator stub can be added into the developer's code and the developer can define what code the block generates.

The Blockly interface is perfectly suited for young or inexperienced developers, allowing them to create fully-functional code with minimal expertise. Moreover, Blockly's Block Factory and extensive library allow for an easy expansion of the Code Editor's current toolbox, either by Remote Lab developers or by users.

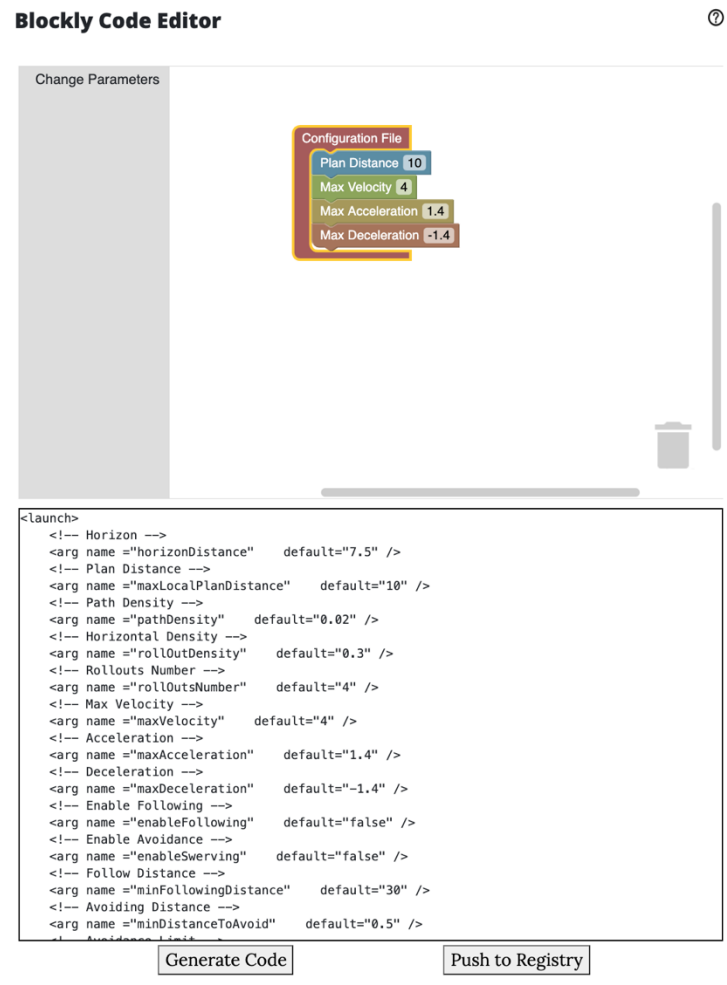

Figure 5: Code Editor Interface including the collapsed Toolbox, the workspace, and the Code Textbox

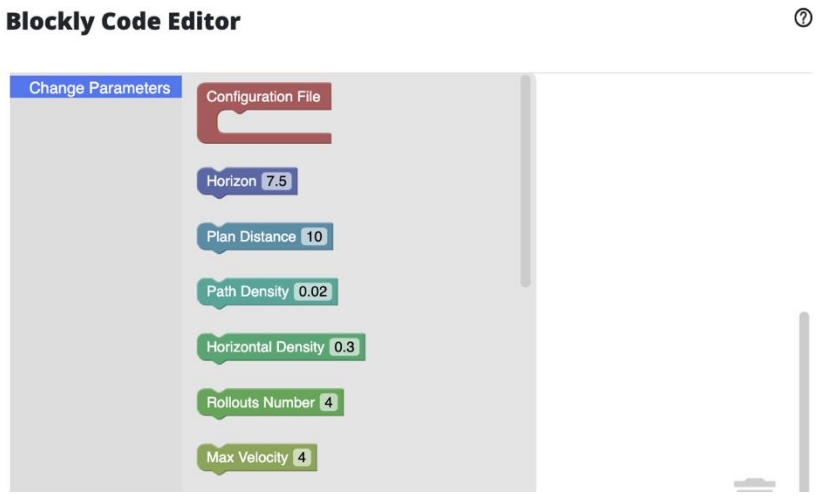

Figure 6: Code Editor Toolbox expanded

2.5. Monitoring Interface & System

The Monitoring Interface is the webpage where the user can access the OSCAR Scaled Car's processing unit while it is running the user's selected Docker image and where the user can view a live video feed of the Driving Environment.

As shown in Figure 1, The Monitoring Interface consists of two components: the Remote Access Interface, and the CCTV Camera Feed.

The Remote Access Interface is where the user can remotely control the OSCAR Scaled Platform by running user applications, editing files, executing experiments or any other actions permitted from within the running container. The Remote Access Interface is accessed by navigating to the 'Remote Access' tab of the user's home page. At reservation time, the user can click 'Launch Remote Connection' and will be redirected to the Monitoring Interface.

The Remote Access Interface utilizes an application built on top of an HTML VNC (Virtual Network Computing) client JavaScript library called noVNC [4]. The noVNC library allows for a VNC server (a server which facilitates graphical desktop-sharing) running within a system to enable desktop-sharing to a user from within a web browser. At the start of each reservation a new password protected VNC server is launched within the user's selected Docker image and becomes accessible to the user on the Monitoring Interface.

The CCTV Camera Feed pictured in Figure 1 consists of four live camera feeds that provide various viewpoints of the Driving Environment in order for the user to view the behavior of the OSCAR Scaled Car. The four cameras are IP cameras that connect to Remote Lab's local network and have addresses that are forwarded to publicly accessible IP addresses. These addresses are then displayed on the Monitoring Interface's webpage.

2.6. Default Docker Image

A docker container is a package of executable software that

includes all necessary system tools, libraries, dependencies, and code to run an isolated and consistent computing environment regardless of the host computer's system.

Prior to the start time of a reservation, the user's selected Docker image gets deployed onto the OSCAR Scaled Car's processing unit. Then, when the reservation is completed the running container is stopped and any changes made during the course of the reservation are pushed as a new image to the user's registry. Each user's image is completely isolated from any other containers and allows the users to make an unlimited amount of alterations with no impact to other users, the car's underlying processing unit, or the user's previous images.

To further promote ease-of-access Remote Lab also provides a default Docker image that includes all necessary installations needed to run a lane keeping module using the OSCAR Scaled Car's built-in LiDAR sensor.

As shown in Figure 7 the default Docker image's base is the NVIDIA Linux4Tegra (L4T) package, which facilitates the execution of Linux on Jetson devices [5], including the OSCAR Scaled Car's processing unit, the NVIDIA Jetson TX2. This base also includes CUDA, TensorRT, and VisionWorks support, commonly used GPU, deep learning, and computer vision toolkits [5].

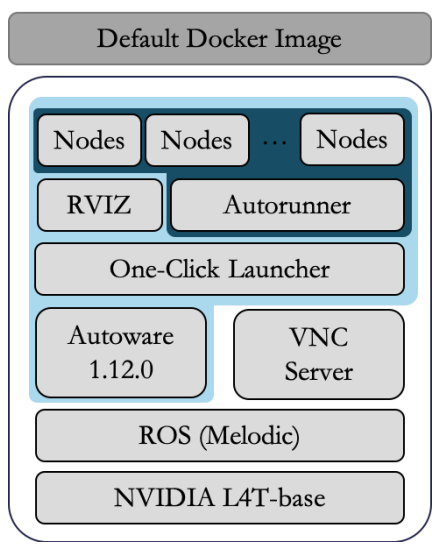

Figure 7: Default Docker image software stack

Above the NVIDIA L4T base is ROS (Robot Operating System), a framework for writing robot software that acts as a middleware between the system's operating system and applications. It provides both operating system services like hardware abstraction and low-level device control, but also additionally provides libraries which assist in the development of robotic applications [6].

Above the ROS layer is the Docker image's application layer. This layer includes the VNC server mentioned in Section 2.5. and the Docker image's Autoware installation. Autoware is an open- source software for autonomous driving technology which includes numerous self-driving modules that cover the principle facets of autonomous driving: sensing, perception, decision, planning, and actuation [7]. The previously mentioned lane keeping module utilizes these provided modules.

The One-Click Launcher pictured above Autoware in Figure 7 automates the launching of all modules and nodes necessary to execute the lane keeping module. This includes launching RVIZ, a 3D visualization tool used to visualize the Driving Environment's point cloud map, the OSCAR Scaled car's current location, and more. It also includes launching an Autorunner, an automated package launcher used to execute all nodes and packages necessary for an Autoware module.

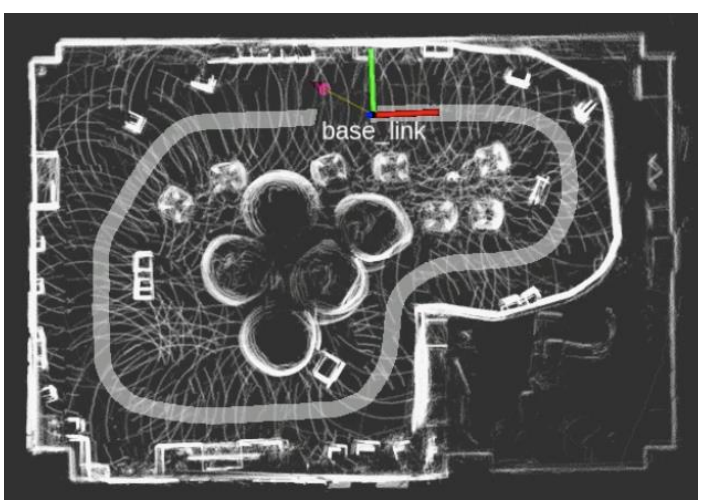

Figure 8: Point cloud map and vector map visualized by RVIZ

The lane keeping module allows a car to determine where it is located with respect to a point cloud map and to simultaneously stay within a specified path or ‘lane’ (defined by a vector map) until the car has reached a user-defined goal point. The car utilizes a continuous stream of LiDAR-generated point cloud data in order to localize itself.

The Autorunner launches the necessary lane keeping nodes as follows. First the LiDAR configuration node is launched, maps are loaded, the TF (transform tree) between the map and car is established, and the voxel grid filter is launched. The TF allows the module to keep track of the car’s location with respect to the map. The voxel grid filter works by reducing the raw point cloud data by taking the spatial average of the points. The filter aids in the reduction of noise included in the raw LiDAR data.

Next the NDT (Normal Distribution Transform) matching node is launched, which uses the NDT algorithm in order to process LiDAR data and match the points to the point cloud map. The algorithm will be further discussed in Section 3.1.1.

Then the global planner node is launched generates a global path from the car’s current location to the user-defined goal point [7]. Concurrently, another node is launched that relays the NDT matching node’s data and publishes it as the car’s current position.

The next collection of nodes handle the local planning, including the car’s local trajectory generator, trajectory evaluator, and subsequent behavior selector. Additionally, parameters like velocity and acceleration are defined by these nodes.

The last two nodes are the pure pursuit node and the waypoint follower node. The pure pursuit node uses a tracking algorithm that calculates the curvature necessary to move the vehicle properly along the path. The way point follower node simply follows the waypoints indicated by the vector map.

The Autorunner launches these nodes in the appropriate order, with the appropriate configuration values. After the user sets a goal point, the control module can then be launched which will move the OSCAR Scaled car one lap around the Driving Environment’s track.

2.7. OSCAR Scaled Car

The OSCAR Scaled Car is a one-tenth scale RC car outfitted with a full sensor suite needed to execute autonomous driving applications. It follows the specifications proposed in [8] with one modification - an improved LiDAR sensor.

The OSCAR Scaled Car' s sensor suite includes a LiDAR, IMU (Inertial Measurement Unit), and a high-resolution stereo camera. As shown in Table 1 the Velodyne VLP-16 LiDAR has a 360-degree detection angle, and up to 100-meter measurement range, making it suitable for LiDAR-assisted object detection and localization. The Sparkfun 9DoF Razor IMU features an accelerometer, gyroscope, and magnetometer, which are used to sense linear acceleration, angular

Category	Component	Features
Car chassis	Traxxas Slash 4x4 Platinum	• 1/10 scale • Low center of gravity • Direct four-wheel drive
ESC	Velineon VXL-3s	• Brushless
Motor	Velineon 3500	
Steering servo	Traxxas 2075	• Digital • High-torque
Speed sensor	Traxxas RPM Sensor	• Easy installation
Microcontroller	Teensy 3.2	• 32-bit ARM Cortex-M4 74MHz processor • 64kB RAM, 256kB flash memory • 5V digital input tolerance • 12-bit analog output, 16-bit PWM
IMU	Sparkfun 9DoF Razor IMU M0	• 32-bit ARM Cortex-M0+ processor • MPU-9250 9DoF sensor • Small size, reprogrammable, multipurpose
LiDAR	Velodyne VLP-16	• 360-degree detection angle • 0.1 to 2-degree angular resolution • 100m measurement range • Up to 0.3 million points/second
Camera	Stereolabs ZED	• 2K stereo camera with dual 4MP RBG sensors • 110-degree field of view • f/2.0 aperture • depth perception up to 20m at 100fps • 6-DoF positional tracking • 3D SLAM capable
Computer	NVIDIA Jetson TX2	• Integrated 256-core Pascal GPU • Dual-core NVIDIA Denver2 + quad-core ARM Cortex-A57 • 8GB 128-bit LPDDR4

Table 1: OSCAR Scaled Car Hardware Components [8]

rotation velocity, and magnetic field vectors [8]. Similar to LiDAR data, IMU data is used alone or in combination with other sensors (i.e. GPS or LiDAR) for SLAM or other autonomous driving applications. The Stereolabs ZED camera includes a 110-degree field of view and 2K resolution with dual 4MP RBG sensors that can be utilized as input for any number of computer vision and SLAM algorithms.

The OSCAR Scaled Car's processing unit is the NVIDIA Jetson TX2. It has an integrated 256-core Pascal GPU, 8GB of memory, Dual-Core NVIDIA Denver 2 64-Bit CPU and Quad-Core ARM Cortex-A57 MPCore. The manufacturer claims it is the fastest, most power-efficient embedded AI computing device capable of computing 1.33 TFLOPs (one trillion floating point operations per second), making it an ideal processing unit for a small-scale autonomous vehicle platform [9].

Figure 9: OSCAR Scaled Car

Chapter 3. Remote Lab Development Challenges

3.1. NDT Matching Localization Inconsistency

As previously discussed in Section 2.6, the localization method used for the lane keeping module is NDT matching. When using this algorithm for localization, its behavior did not display a consistent pattern. Due to the unpredictable rate of successful localization the lane keeping module often resulted in the OSCAR Scaled car not following the intended path. The default docker image is a key feature that improves the ease of use of the Remote Lab platform for inexperienced users. Therefore, the consistent execution of the lane keeping module is necessary in order to provide the full intended experience of Remote Lab. In the following experiments, the algorithm' s sensitivities and limitations are discussed with the intention of ultimately improving the rate of successful localization and overall execution of the lane keeping module.

3.1.1 Normal Distributions Transform (NDT)

Normal Distributions Transform (NDT) transforms the discrete set of scan points into a piecewise continuous and differentiable probability density that consists of a set of normal distributions. When using NDT the real world is represented as a 3D plane that is then subdivided regularly into cells with a constant size [10]. Then each cell containing a minimum of three scan points is assigned a normal distribution, which models the probability of whether or not a certain scan point matches a map point within the cell. The algorithm attempts to maximize the likelihood that the input scan matches the map points by optimizing the rotation and translation of the given pose estimate [11].

Proposed benefits of using the NDT method for scan matching over more common methods like ICP (Iterative Closest Point) are that

is it less computationally expensive since computing the set of normal distributions is a one-time computation, not involving continuous iteration. Additionally, compared to ICP, NDT can more robustly handle poor initial position approximations, measurement errors, and slight changes between the reference map and input scan points [10, 11]

However, there exists a number of drawbacks to the NDT algorithm. Most notably the algorithm's performance is dependent on the suitability of the grid size to the scan environment [12]. And NDT is also commonly used with filtering algorithms to reduce the interference of noise on matching accuracy.

3.1.2. NDT Matching Localization Experiments

All experiments were conducted using the OSCAR Scaled Car, which include an NVIDIA Jetson TX2 as its processing unit, and a Velodyne VLP-16 as its LiDAR sensor. As shown in Figure 10, experiments were conducted in a 9.6m by 6.6m room, altered features include one-tenth scale road barriers of height 10cm, one-tenth scale buildings of height 40cm and 50x50x80cm chairs. Part of the track is surrounded by a 64cm barrier to reduce interference from miscellaneous items.

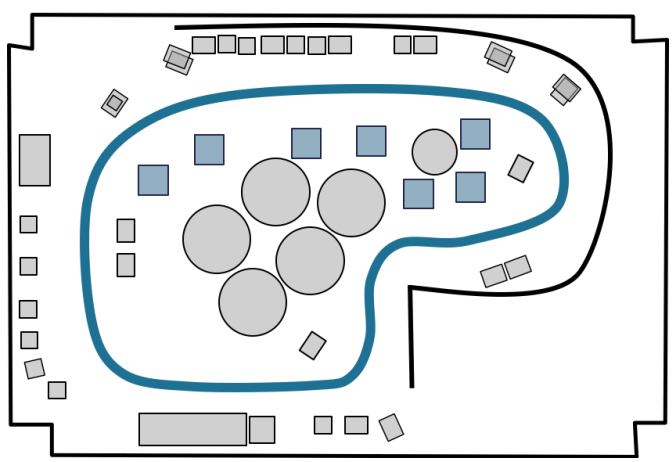

Figure 10: Driving Environment track. Buildings are gray squares, 50x50x80cm chairs are blue squares, 100cm diameter chairs are large circles, the track is pictured as the blue line, and the 64cm barrier is represented by the black curved line.

All experiments were conducted for 200 iterations. Each iteration the NDT matching node executed for 10 seconds before being killed and relaunched.

Three major experiments were conducted with respect to their impact on NDT matching performance: the effects of initial position inaccuracy, the effects of map inaccuracy, and the effects of map resolution.

To determine the effects of an initial position inaccuracy on the NDT matching algorithm's performance various translated and rotated positions were given to the algorithm. The translated positions ranged from a 10cm to 250cm offset from the car's true position. The rotated positions were 15°, 45°, 90°, and 180°.

As seen in Figure 11, results of the experiment show that as the magnitude of the translation error increases from 10cm to 50cm a decrease in performance from 35.5% to 20.5% is shown. However, as the translation error increased an improvement in the rate of success can be seen. At a 100cm translation error the success rate increases 7.5% to 28.0% and all larger errors average a success rate of 47.6%. This indicates that after a certain threshold (between 100cm and 150cm of displacement) the rate of successful localization is comparable to the baseline rate of 49.5%. Therefore, when a relatively small (10cm to 100cm) translation error exists, a decrease of performance can be observed. However, when the error is significantly large enough the NDT matching algorithm reverts to its baseline behavior.

In contrast, a rotation error shows a minimal decrease in performance, with 15° and 45° of rotation error showing up to a 1% decrease. Both 90° and 180° rotation errors show a slightly larger decrease with a 41.5% and 36.5% success rate, respectively.

The effects of map inaccuracy on the NDT matching algorithm were determined through the translation, rotation, addition, and removal of features from the Driving Environment. Chairs indicated in Figure 10 as blue squares and buildings indicated as gray squares were used as the altered features. The translation/rotation experiments translated 1, 2, and 3 chairs 5cm and 15cm and rotated the same

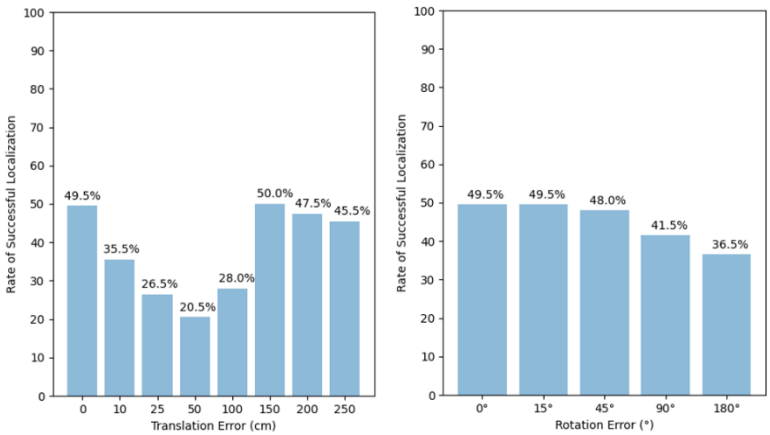

Figure 11: Initial position inaccuracy experiment results

number of chairs 45° and 90° . For the removal/addition experiment 1,3, and 5 chairs were removed from the environment and 1,2, and 3 chairs were added. The same experiment was also conducted with 40cm buildings.

When examining the overall effect of rotation and translation of existing features a slight decrease in performance can be observed. When translating the position of one, two, and three features the rate of success averages to 38%, which is a modest 11.5% decrease from the baseline success rate. A 45° rotation also shows a similar but slightly higher rate of success, averaging a success rate of 41.6%, only 7.9% less than the baseline rate. Conversely, when the chairs were rotated 90° the success rate drops to 25.5%, 14.0%, and 12.5% for one, two, and three rotated chairs, respectively.

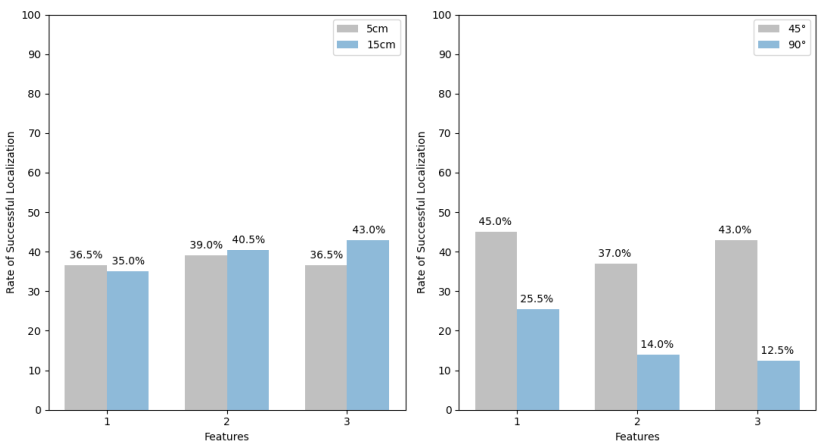

Figure 12: Feature translation (left) feature rotation (right)

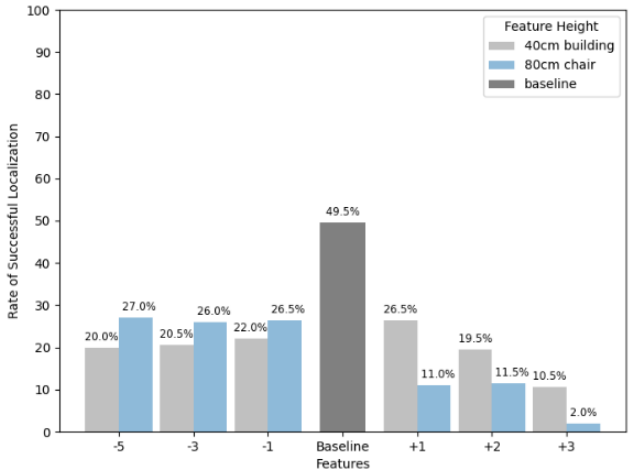

Figure 13: Feature removal and addition experiment results

As seen in Figure 13, when features were removed from the Driving Environment, regardless of the number of removed features the success rate was measured to be an average of 26.5% for removed chairs and 20.8% for removed buildings. Conversely, the addition of features shows a greater impact on performance with an average success rate of 8.2% for chairs and 18.8% for buildings. The results clearly show the impact of additional features impair performance more than the removal of features. Moreover, as more features are added the performance shows a downward trend and does not plateau.

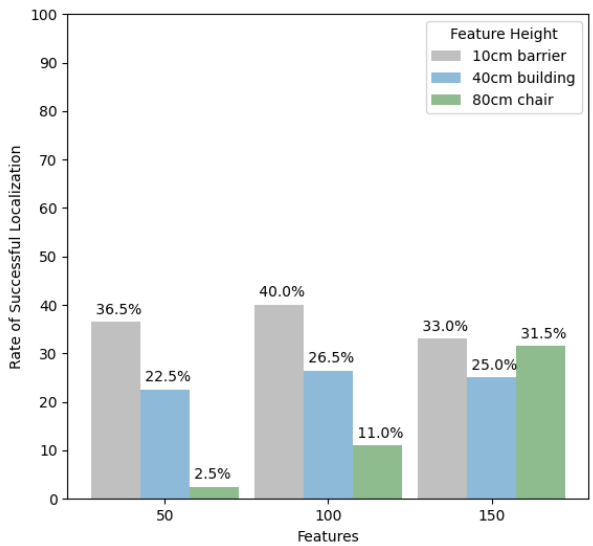

Figure 14: Feature addition of various sizes at increasing distances

To examine the impact of the addition of features further, features of 10cm, 40cm, and 80cm height were added to the Driving Environment at distances of 50cm, 100cm, and 150cm. Unsurprisingly, as the 80cm chair was placed closer to the car the performance decreased substantially from 31.5% at 150cm to 2.5% at 50cm. However, the additions of both the building and the barrier had stable success rates, regardless of the feature's distance to the car. This shows that under a certain size (approx. 40cm height) the interference caused by an additional feature is not relative to its distance from the car.

In order to determine the effects of map resolution on the algorithm's performance every n^{th} point of the baseline map was removed and the threshold where a performance decrease occurred was observed. The value of n ranged from 100 to 2.

As shown in Figure 15, when the value of n was greater than 50, performance was improved, with a maximum success rate of 58% when n is equal to 75. When n was less than 50 the success rate began to be negatively impacted with the lowest success rate being 15% when n equals 2.

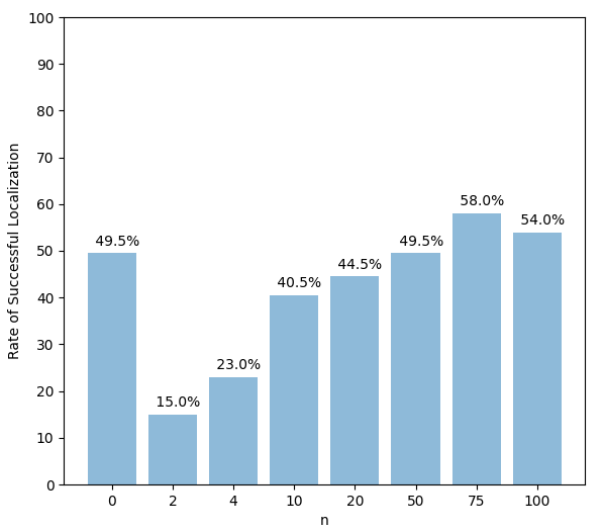

Figure 15: Map resolution experiment results

3.1.4. Experiment Conclusion

The data presented in the experiments above support a number of conclusions.

Firstly, initial position errors impact NDT matching performance when they are between 0 to 50cm within the ground truth of the car's position. When examined further there appears to be no discernible difference between the path of the localization points across the experiments where the initial position was translated 0 to 50cm and when it was translated an amount greater than 100cm. The data does not show a difference in the percentage of slightly shifted unsuccessful localizations (± 30cm), or of localizations outside the reference map boundary (± 500cm) when comparing small initial position inaccuracies and larger inaccuracies. A further investigation of the reasons behind this aspect of the algorithm's weakness is a suitable topic for further research.

Secondly, the results shown in Figure 13 indicate that the NDT algorithm is less tolerant of added noise (features) than feature removal. A claim also supported by the map resolution experiment results in Figure 15. As shown in Table 2, data shows that failed localization was often characterized by only a slight translation (± 30cm displacement) of the car's true position. This occurs because the algorithm incorrectly identifies the added feature as a feature that is already present on the map, skewing all the other points around it. In other words, the algorithm believes that an added chair is actually a chair marked on the map next to it and transform the input set of points accordingly to match the map. The algorithm is unable to reliably match a scan of a feature to an existing feature on the reference scan when a similar feature is placed near it. This is true even for the case when all other features remain exactly placed as they are in the reference map. As stated in [11], a characteristic that adds to NDT's robustness is that it is tolerant of small changes in the feature space. However, as shown in this experiment, this alleged robustness can lead to a decrease in localization accuracy when an added feature is mistaken as a feature on the reference map and a

translated location is accepted as truth. For real-world autonomous driving applications this can greatly deteriorate the algorithm's reliability and safety. Within the scope of Remote Lab, it implies that a map will have more robust performance when any possible features are included, even if they will be removed at some point.

Table 2: Successful localization within ±1cm and ±30cm of accuracy for removed and added features (chairs)

Features	Successful Localization (%) ± 1cm	Successful Localization (%) ± 30cm
-5	27.0	28 (+1.0)
-3	26.0	27 (+ 1.0)
-1	26.5	27 (+ 0.5)
+1	11.0	62.5 (+ 51.5)
+2	11.5	63.0 (+ 51.5)
+3	2.0	75.5 (+ 73.5)

Lastly, when the effects of map resolution were studied, it suggests that more points on the map does not always lead to more accurate localization. As shown in Figure 15, n equal to 75 and 100 both displayed higher rates of success than the baseline map which had no points removed. This suggests that the modest (1-2%) removal of points can improve localization by reducing noise and the computational burden of matching more points.

In conclusion, the same features of the NDT matching algorithm that lend to its robustness can impact the algorithm's accuracy. One of the algorithm's most significant weaknesses is that initial position inaccuracies are inevitable in real-life situations, and a small inaccuracy can potentially impact the algorithm's performance by more than half. Additionally, if the removal of points from the map improves its performance it is unclear how the algorithm's baseline performance of only 49.5% can be further improved. With a baseline success rate of less than 50% it cannot be said that NDT is sufficiently reliable as a localization algorithm.

Chapter 4. Conclusion and Future Work

4.1. Conclusion

Remote Lab’ s primary goal is to provide access to a remote experimentation platform to the general public, especially to users who may only have a beginner or intermediate level of knowledge in the field of autonomous driving and programming. The Remote Lab platform enables users with limited knowledge to interact with, observe, alter, and produce self-driving applications. Along with being easy to use, Remote lab also provides an accessible platform for users who do not have the financial capability to create their own autonomous vehicle platforms. As the platform progresses and improves, more features that enhance accessibility and education will be added.

This work also investigates the NDT matching localization algorithm and concludes a major weakness of the algorithm is that small inaccuracies in the initial position impact performance by more than half. In addition, the baseline success rate is 49.5%, making the performance unreliable even when the correct initial position is given, and no features are altered.

4.2. Future Work

Future work for Remote Lab includes a redesign of the Driving Environment. The track will be replaced, obstacles will be changed, and more CCTV cameras will be added to improve the user visibility of the OSCAR Scaled car’ s movements. Furthermore, an expansion of the code editor’ s functionality and the addition of more driving modules to the default Docker image are planned for the future.

Bibliography

[1] F1TENTH Foundation, "Building the F1TENTH Car," F1TENTH Foundation, 2021. [Online]. Available: https://f1tenth.org/build.html. [Accessed 2 June 2021].

[2] Google Developers, "Blockly," Google Developers, [Online]. Available: https://developers.google.com/blockly. [Accessed 2 June 2021].

[3] Google Developers, "Blockly Developer Tools," Google Developers, [Online]. Available: https://developers.google.com/blockly/guides/create-custom-blocks/blockly-developer-tools. [Accessed 2 June 2021].

[4] J. Martin, "noVNC," [Online]. Available: https://novnc.com/info.html. [Accessed 2 June 2021].

[5] NVIDIA, "NVIDIA L4T Base," NVIDIA, [Online]. Available: https://ngc.nvidia.com/catalog/containers/nvidia:l4t-base. [Accessed 02 June 2021].

[6] Open Robotics, "About ROS," Open Robotics, [Online]. Available: https://www.ros.org/about-ros/. [Accessed 2 June 2021].

[7] The Autoware Foundation, "Project Autoware.AI," The Autoware Foundation, 2020. [Online]. Available: https://www.autoware.ai/. [Accessed 2 June 2021].

[8] A. Kazakova, Y. Cho and C.-G. Lee, "OSCAR: An Open-Source, Self-Driving CAR Testbed," *한국정보과학회 학술발표논문집,* pp. 720-722, 2018.

[9] NVIDIA, "Jetson Modules," NVIDIA, [Online]. Available: https://developer.nvidia.com/embedded/jetson-modules. [Accessed 2 June 2021].

[10] P. Biber and S. Wolfgang, "The normal distributions transform: A new approach to laser scan matching," *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems,* vol. 3, pp. 2743-2748, 2003.

[11] M. Magnusson, "The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection," Doctoral dissertation, Örebro universitet, 2009.

[12] M. Attia and Y. Slama, "Normal Distribution Transform with Point Projection for 3D Point Cloud Registration," in *5th International Conference on Control & Signal Processing (CSP-2017)*, Kairouan, Tunisia, 2017.

[13] The Autoware Foundation, "OpenPlanner - Global Planner," The Autoware Foundation, 2018. [Online]. Available: https://autoware.readthedocs.io/en/feature-documentation_rtd/DevelopersGuide/PackagesAPI/mission/op_global_planner.html. [Accessed 2 June 2021].

Abstract

실제 자율주행차 연구는 가장 단순한 자율주행차 플랫폼조차도 높은 기준제조단가로 인해 자동차 산업 대기업(예: 현대, 테슬라)이나 자동차 하드웨어 제조업체(예: NVIDIA, Bosch)와 관련 연구자들이 아니고선 종종 접근할 수 없습니다.

원격랩의 주요 기능은 자율주행 분야에서 초급 또는 중간 수준의 지식만 가질 수 있는 사용자에게 초점을 맞춰 원격 실험 플랫폼에 일반 대중이 접근할 수 있도록 하는 것입니다. 본 연구에서는 Remote Lab의 아키텍처, 다양한 기능 및 Remote Lab 개발과 관련된 문제에 대한 개요를 제공합니다.

추가로 플랫폼에 활용되는 기본 로컬라이제이션 알고리듬인 NDT (Normal Distributions Transform) 매칭에 대한 심층 분석은 일관되고 강력한 로컬라이제이션과 관련된 알고리듬의 약점을 논의합니다. ②

주요어: 자율주행, 원격 실험 플랫폼, Normal Distributions Transform
학번: 2019-28748

② 본 논문작성자는 한국정부초청장학금(Global Korea Scholarship)을 지원받은 장학생임