Beijing University of Chemical Technology 777 RMUS Technical Report

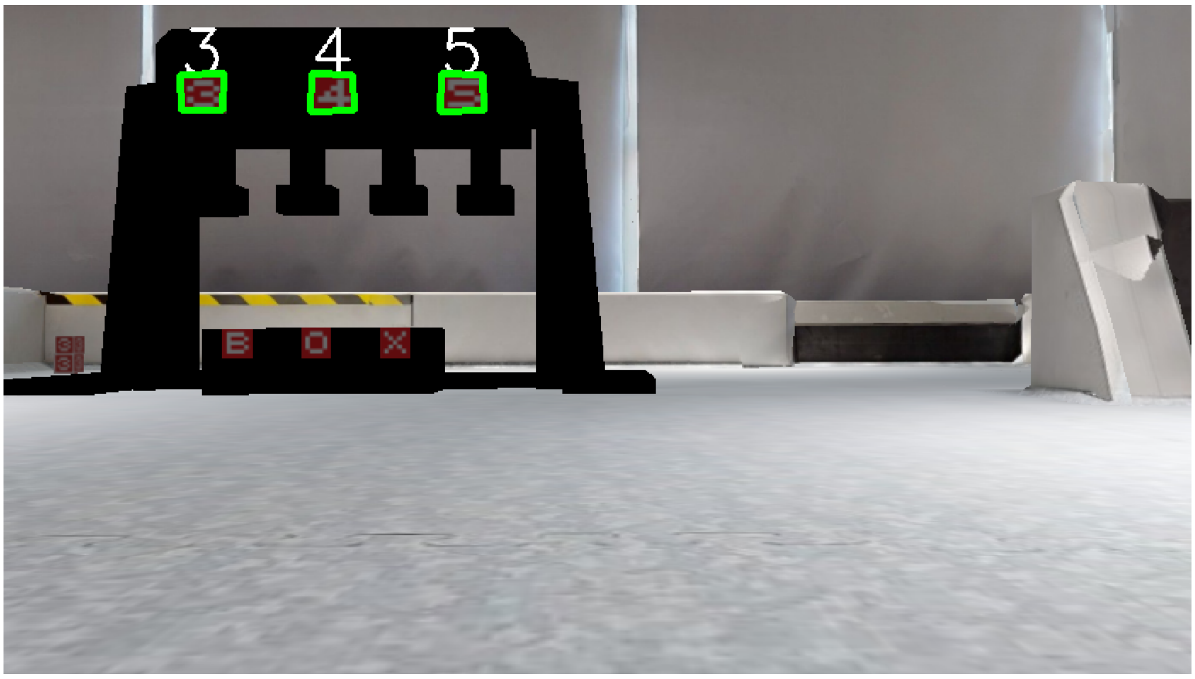
Sorry for writing this report in English although we're a Chinese team. Our Ubuntu system had some problems with Chinese typing. Switching system to Windows would make it inconvenient to get some pictures and screenshots. So in order to save more time, we decided to write this report in English.

1. Simulator Test Stage

1.1 Auto-Detection

1.1.1 Detect ID

In this part we modified the official **ep_detect** code to make it able to detect ID 4 and ID 5.

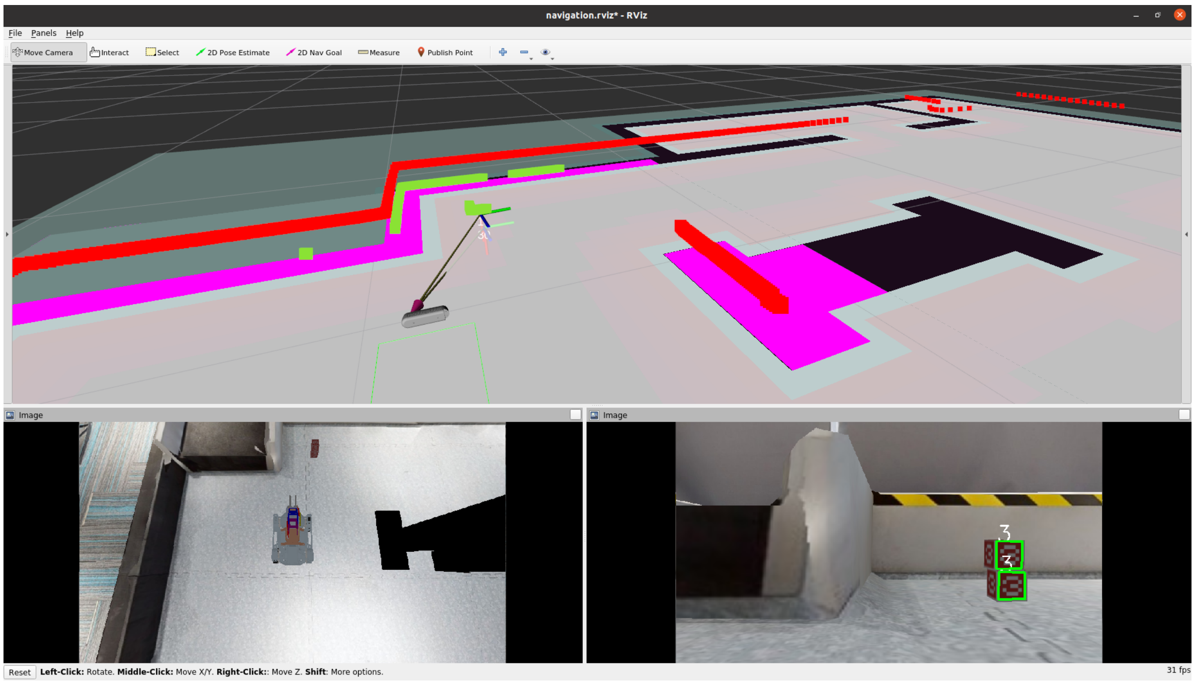


1.1.2 Wrap Every Box's Information in TF Tree

Also, we extracted the transform and rotation matrix from the official code in solvePnP function. Then we customized out own message format (similar to [darknet_ros]) and wrapped the transform and rotation matrix into this message so that we can get the ID, position and orientation of every box in TF tree.

Then, we applied a simple homogeneous transform matrix operation. Although we've got the pose data of every box, it is actually the pose data of the box's surface. Sometimes when we only detected a side of a box, it would regard the side of the box as the position to grasp. To solve this problem, we transformed the pose data of every surface marker with a homogeneous transform matrix to get the central point of the box.

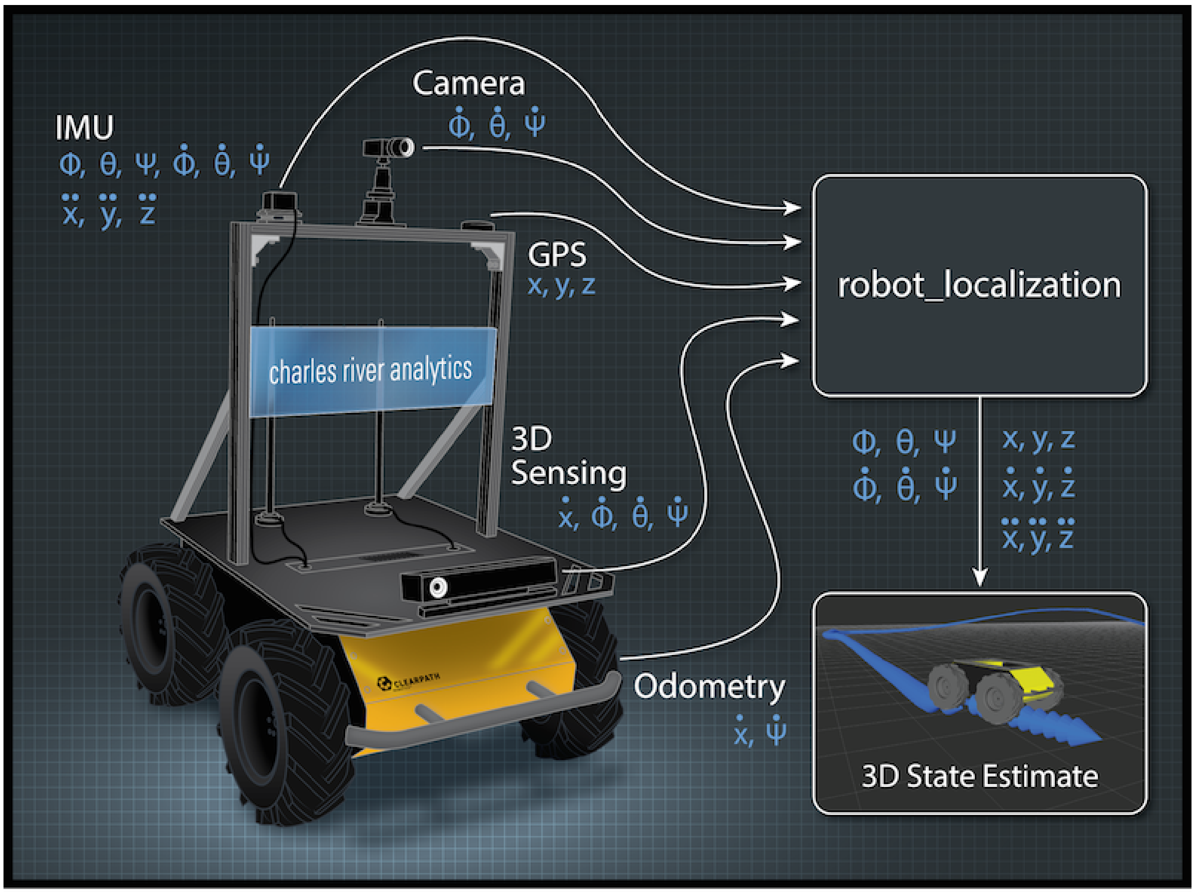
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0.0225 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(homogeneous transform matrix)}$$



1.2 Positioning

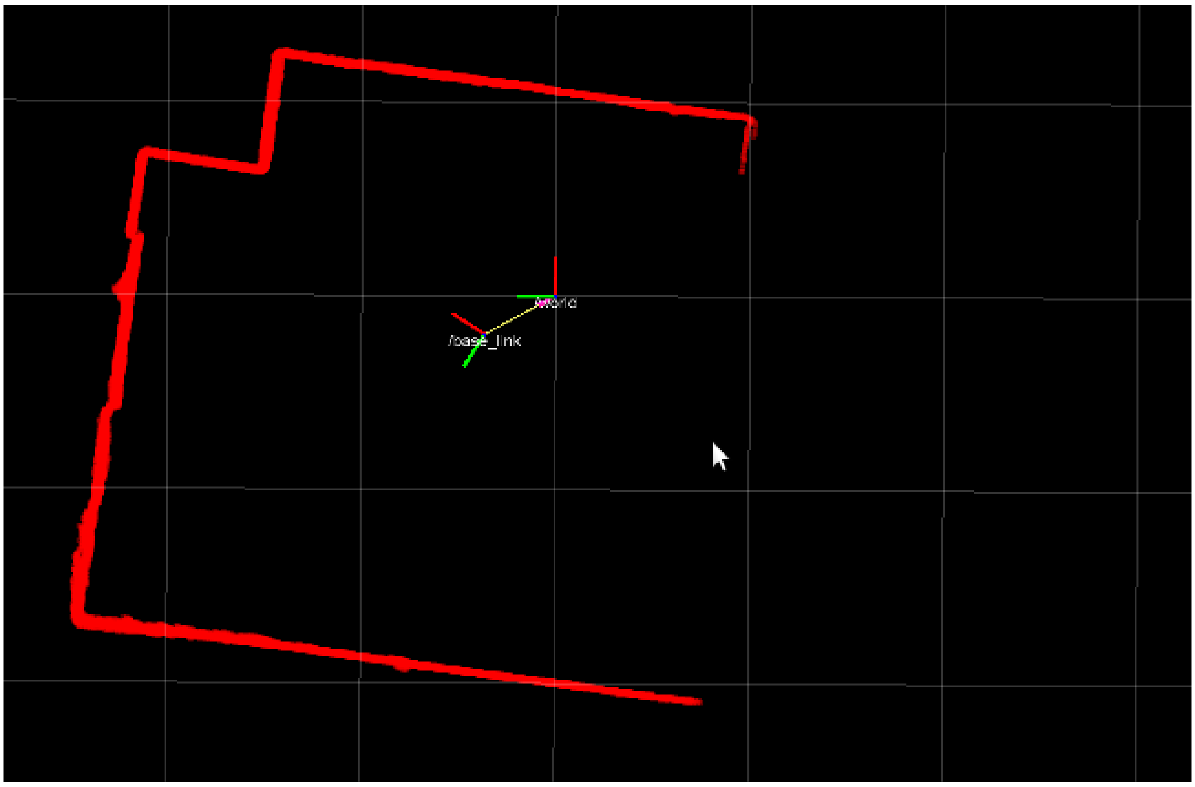
1.2.1 odom --> base_link

We fused the sensor data by using **robot_localization** package. It is a collection of state estimation nodes, each of which is an implementation of a nonlinear state estimator for robots moving in 3D space. The method we applied here is Extented Kalman Filter (EKF). The main reason why we use this package instead of **robot_pose_ekf** is that **robot_localization** can fuse arbitrary number of sensors while **robot_pose_ekf** can only fuse limited sensor data.


IMU
$\Phi, \theta, \psi, \dot{\Phi}, \dot{\theta}, \dot{\psi}$
$\ddot{x}, \ddot{y}, \ddot{z}$
Camera
$\dot{\Phi}, \dot{\theta}, \dot{\psi}$
GPS
x, y, z
3D Sensing
$\dot{x}, \dot{\Phi}, \dot{\theta}, \dot{\psi}$
Odometry
$\dot{x}, \dot{\psi}$
charles river analytics
CLEARPATH
robot_localization
Φ, θ, ψ
$\dot{\Phi}, \dot{\theta}, \dot{\psi}$
x, y, z
$\dot{x}, \dot{y}, \dot{z}$
$\ddot{x}, \ddot{y}, \ddot{z}$
3D State Estimate

The first input source is wheeled odometry, which is directly provided by the official code in **/ep_odom** topic. We take the linear velocity in x and y axis, and angular velocity in z axis as input data.

The second input source is laser odometry. We used a [laser_scan_matcher] to get this input. In this laser odometry, each laser scan is compared to a certain previous scan called the key frame so that we can get the transformation between the two scans. The keyframe scan is updated after the robot moves a certain distance. We take the linear velocity in x and y axis, and angular velocity in z axis as input data.



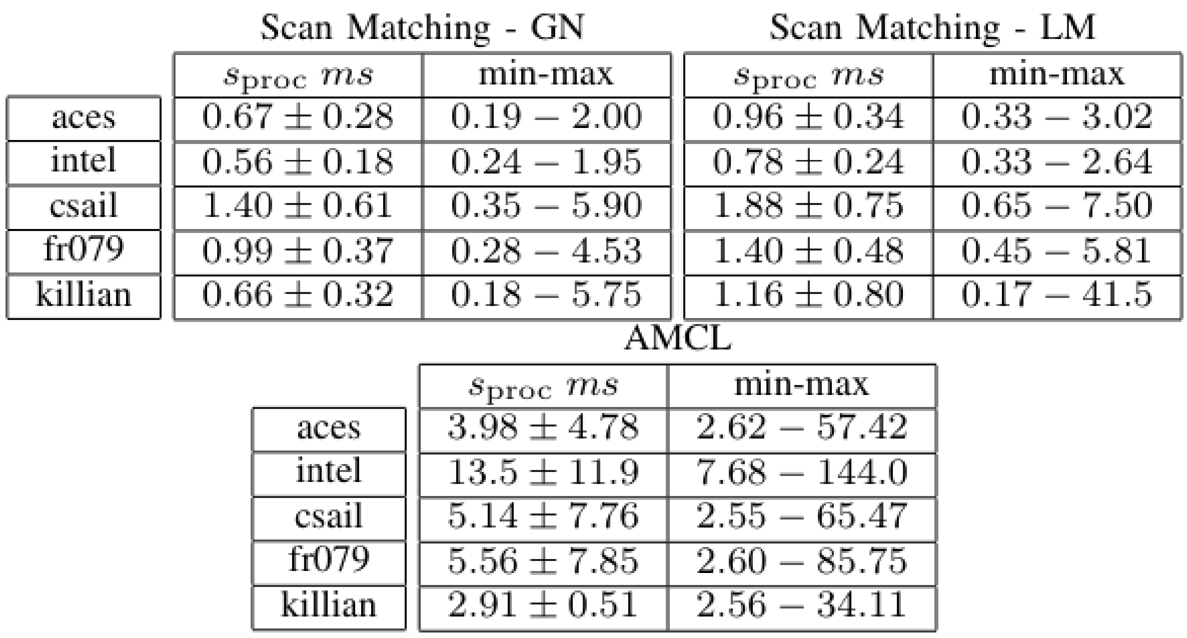
The third input source is still laser odometry, but the laser is from depth camera instead of rplidar by using depthaimage_to_laserscan. We take the linear velocity in x and y axis, and angular velocity in z axis as input data.

The last input source is IMU, which is directly povided by the official code in **/imu/data_raw** topic. We take the angular velocity in z axis and orientation in z axia as input data.

1.2.2 map --> odom

Here we used a localization package called iris_lama_ros instead of amcl, rtab-map or cartographer. The main feature is efficiency, low computational effort and low memory usage whenever possible. It provides a fast scan matching approach to mobile robot localization supported by a continuous likelihood field. It can be used to provide accurate localization for robots equipped with a laser and a not so good odometry.

MEAN EXECUTION TIMES AND RESPECTIVE STANDARD DEVIATION, AND MIN-MAX VALUES FOR ALL DATASETS, GROUPED BY ALGORITHM.



	Scan Matching - GN		Scan Matching - LM	
	s_{proc} ms	min-max	s_{proc} ms	min-max
aces	0.67 ± 0.28	$0.19 - 2.00$	0.96 ± 0.34	$0.33 - 3.02$
intel	0.56 ± 0.18	$0.24 - 1.95$	0.78 ± 0.24	$0.33 - 2.64$
csail	1.40 ± 0.61	$0.35 - 5.90$	1.88 ± 0.75	$0.65 - 7.50$
fr079	0.99 ± 0.37	$0.28 - 4.53$	1.40 ± 0.48	$0.45 - 5.81$
killian	0.66 ± 0.32	$0.18 - 5.75$	1.16 ± 0.80	$0.17 - 41.5$

	AMCL	
	s_{proc} ms	min-max
aces	3.98 ± 4.78	$2.62 - 57.42$
intel	13.5 ± 11.9	$7.68 - 144.0$
csail	5.14 ± 7.76	$2.55 - 65.47$
fr079	5.56 ± 7.85	$2.60 - 85.75$
killian	2.91 ± 0.51	$2.56 - 34.11$

As we tested in real robots before, amcl and rtab-map requires much computation. We need to save computation power for teb_local_planner and other potential applications of some algorithms. And about Cartographer, it is based on sub-map matching, which would make the positioning result shaking in the case of stopping at the goal point. As a result, move_base couldn't recognize whether the robot has arrived at the goal point in time. Thus, we found iris_lama_ros the most efficient package to apply.

1.3 Motion planning

1.3.1 Global Planner

Global path planning needs to grasp all the environmental information, and carry out path planning according to all the information of the environmental map, which belongs to static planning. A* (A-Star) algorithm is one of the most effective direct search methods for solving the shortest path in a static road network. One, is also an efficient algorithm for solving many search problems. The closer the distance estimate in the algorithm is to the actual value, the faster the final search will be.

1.3.2 Local Planner

We applied Timed Elastic Band algorithm. In ROS, it has a wrapped package called **teb_local_planner**.

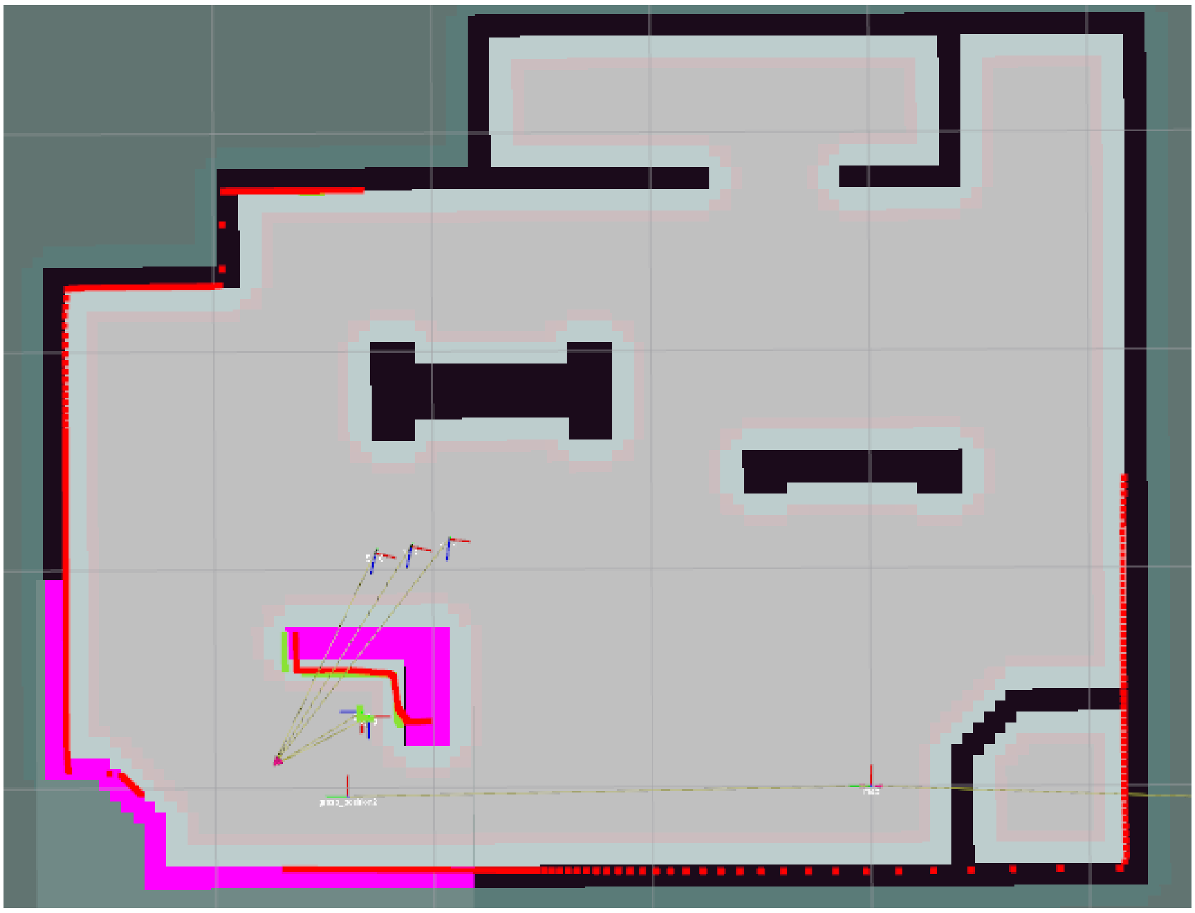
This package implements an online optimal local trajectory planner for navigation and control of mobile robots as a plugin for the ROS navigation package. The initial trajectory generated by a global planner is optimized during runtime w.r.t. minimizing the trajectory execution time (time-optimal objective), separation from obstacles and compliance with kinodynamic constraints such as satisfying maximum velocities and accelerations.

The optimal trajectory is efficiently obtained by solving a sparse scalarized multi-objective optimization problem. The user can provide weights to the optimization problem in order to specify the behavior in case of conflicting objectives.

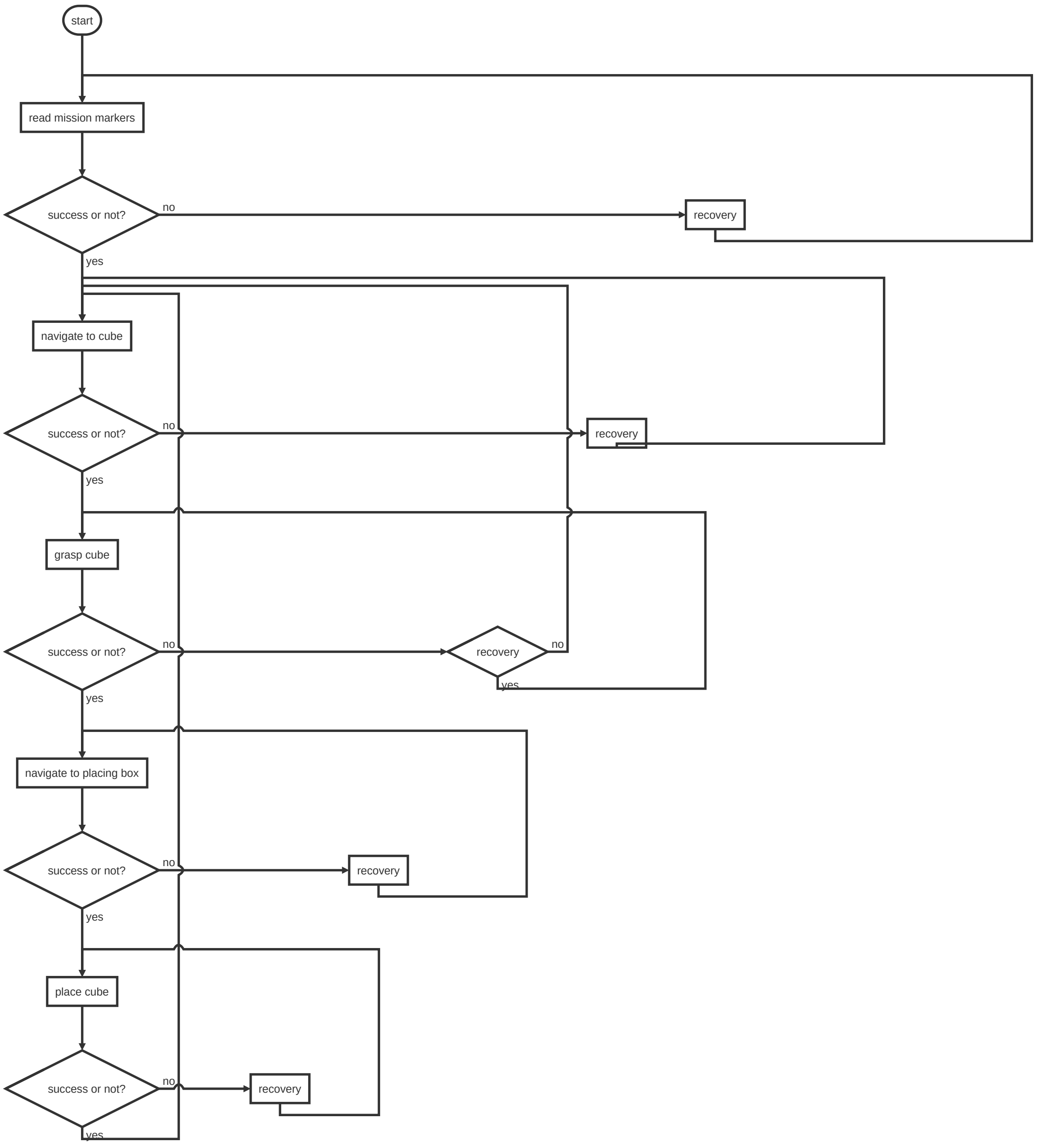
1.3.3 Obstacle Avoidance

We only have one lidar on robot, and it cannot sense the back the the robot. When planning a path, without the sensing of every angle of the robot, the robot cannot avoiding collisions. To solve this problem, we imported the whole map for the static layer of costmap.

Also, the laserscan data is used for dynamic obstacle avoidance. It is used for obstacle layer. However the lidar cannot reach every box on the ground. To solve this problem, we took the depth image of D435 for scanning boxes on the ground with the package **depthimage_to_laserscan**. As you can see in the picture below, the green scan data.



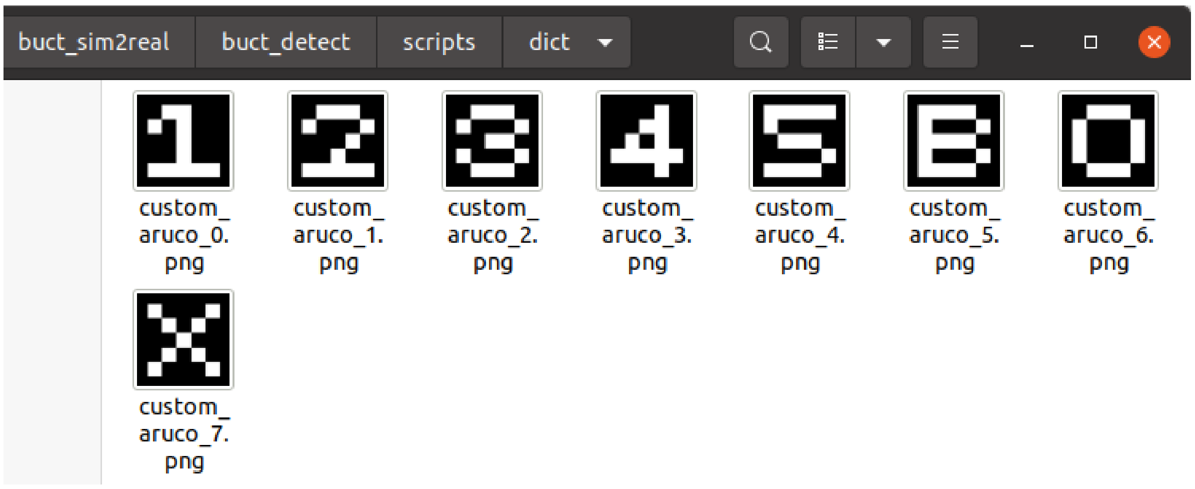
1.4 Decision making


start
read mission markers
success or not?
no
recovery
yes
navigate to cube
success or not?
no
recovery
yes
grasp cube
success or not?
no
recovery
no
yes
yes
navigate to placing box
success or not?
no
recovery
yes
place cube
success or not?
no
recovery
yes

2. Sim2Real Test Stage

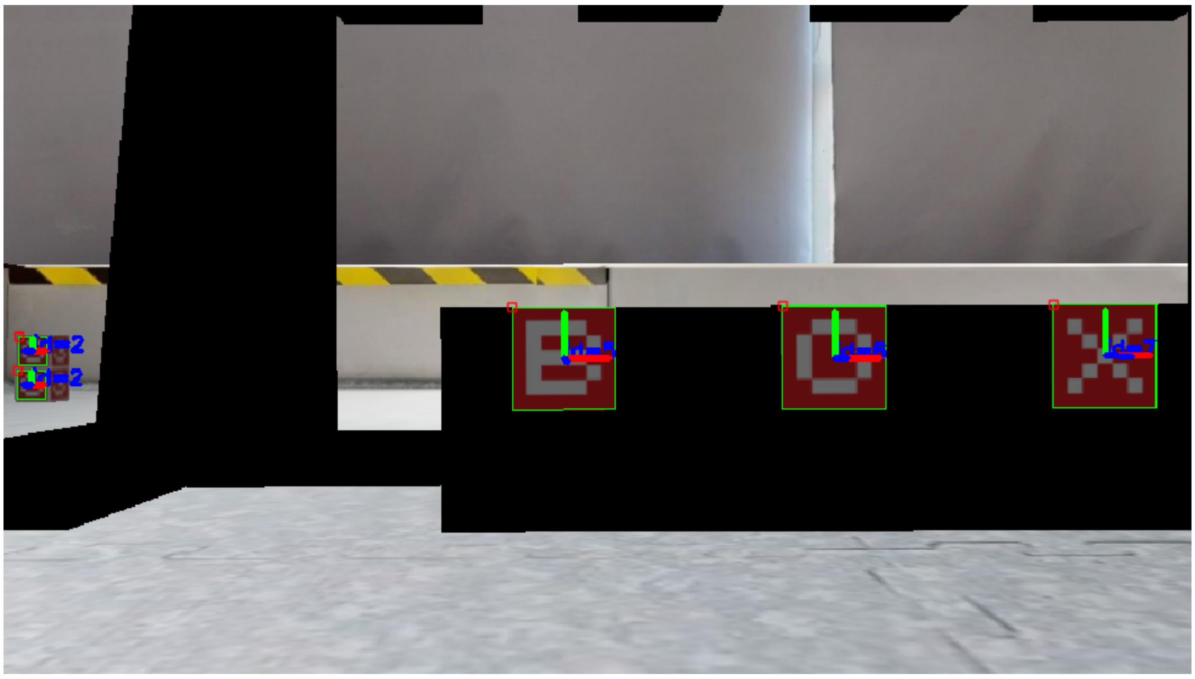
2.1 Auto-Detection

In the Sim2Real part, we replaced the official **ep_detect** with our own approach. We customized an ArUco dictionary of all the 8 figures.



So, we applied ArUco detection wrapped in OpenCV to detect all the markers. However, the markers in this challenge is red, so, before we convey the images to ArUco detection functions in OpenCV, we turned the red into black and other part into white so that marker detection can be more accurate.

After that, we can get accurate poses of every marker, more accurate than the official **ep_detect**.



2.2 Positioning

The Positioning algorithm in Sim2Real Test Stage is the same as what it was in Simulator Test Stage.

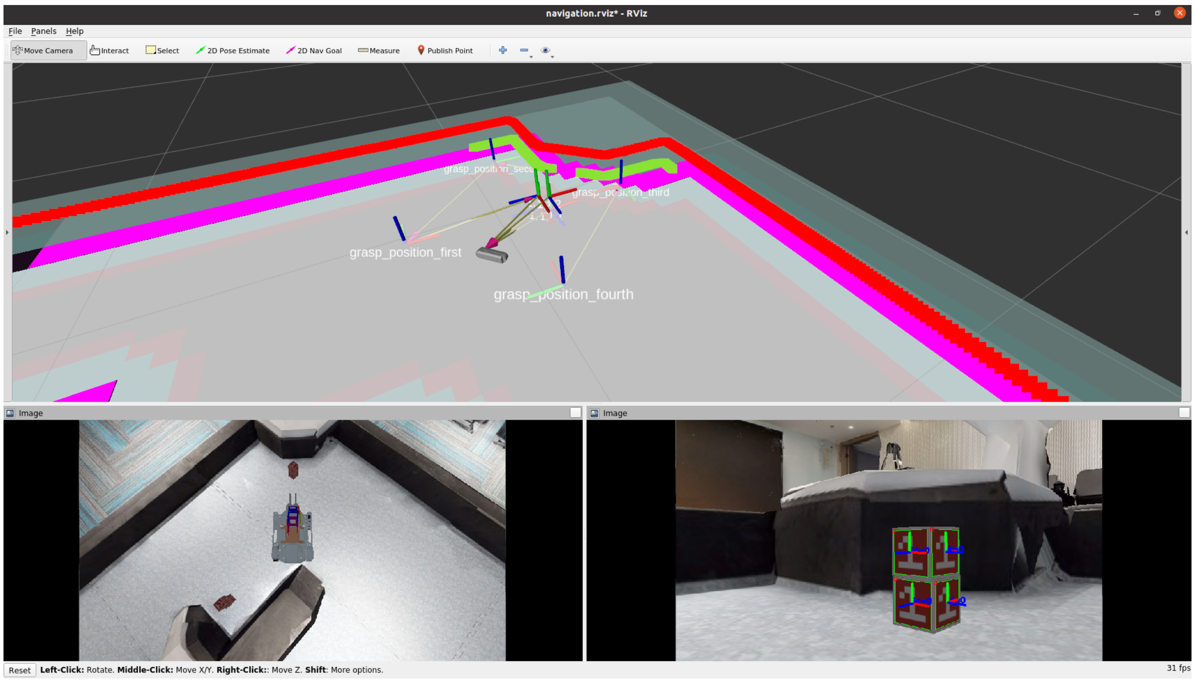
2.3 Motion planning

The Motion planning algorithm in Sim2Real Test Stage is the same as what it was in Simulator Test Stage.

2.4 Decision making

The basic decision making algorithm in Sim2Real Test Stage is the same as what it was in Simulator Test Stage.

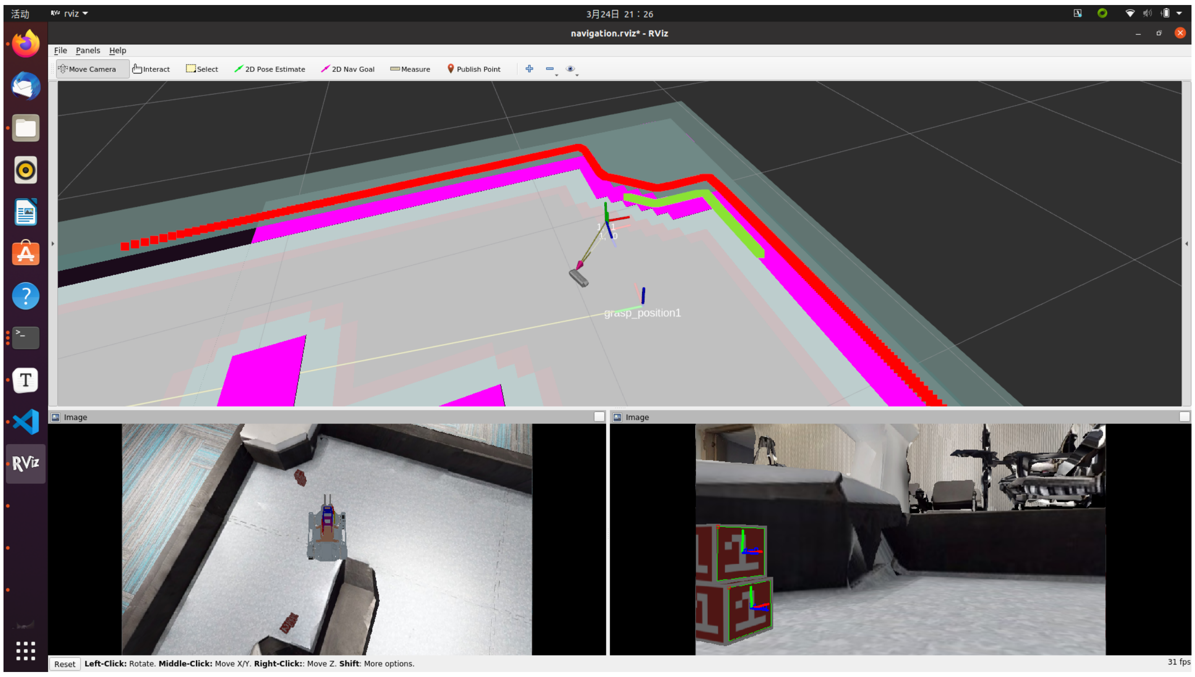
However, in reality, every box is positioned in a random angle towards the robot. To solve this problem, we developed a best grasp position searching algorithm. It is based on the accurate poses of every marker we got in Auto-Detection part. With this information, we can get four potential poses that the robot can face the box straightly. You can see the four potential grasping poses in the image below.


navigation.rviz* - RViz
File Panels Help
Move Camera Interact Select 2D Pose Estimate 2D Nav Goal Measure Publish Point
grasp_position_first
grasp_position_third
grasp_position_fourth
Image
Image
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.
31 fps

Then, we designed some conditions of the best grasping pose:

- the xy position must be in the map frame
- the orientation must be appropriate to make sure the robot won't collide on the wall when grasping

Then we select the nearest position as the best graping position and then navigate there to begin grasping process.


活动 rviz
3月24日 21：26
navigation.rviz* - RViz
File Panels Help
Move Camera Interact Select 2D Pose Estimate 2D Nav Goal Measure Publish Point
grasp_position1
Image
Image
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Move Z. Shift: More options.
31 fps