ECE5532 FINAL PROJECT: IGVC COURSE

Professor: Wing-Yue Geoffrey Louie

Date: April 20, 2020

Grudzien, Benjamin

bgrudzien@oakland.edu

Tkac, Sam

samtkac@oakland.edu

Abstract

The ROS Navigation Stack was used to autonomously navigate a differential drive robot through the IGVC course. To achieve this the “Move Base” parameters of the ROS Navigation stack were adjusted. The ROS Navigation stack utilizes simulated Lidar sensor data which allow the robot to autonomously drive. Also, a concise algorithm was implemented which fed the differential drive robot goal waypoint locations for it to drive to. The robot was able to successfully complete the basic IGVC courses and the advanced level IGVC courses.

Table of Contents

Instructions: CTRL + Click heading to jump to that section of the report.

Introduction

At a high level, this report documents the steps taken and the required components necessary for completion the IGVC course. This report covers the kinematics for a differential drive robot, how the ROS Navigation Stack was utilized to autonomously avoid obstacles, it covers how an algorithm was implemented to give the robot goal waypoints to drive to, and it provides supplemental figures and descriptions of how the overall system was implemented in ROS.

Differential Drive Kinematics

The robot kinematics are based on the differential drive kinematics of the Roundbot. It is important to note that the ROS Navigation Stack move base parameters had a default "Roundbot Local Planner." This parameter implemented the kinematics. Therefore, the kinematics did not need to be manually implemented. For reference, the Roundbot is shown and labeled in Figure 1.

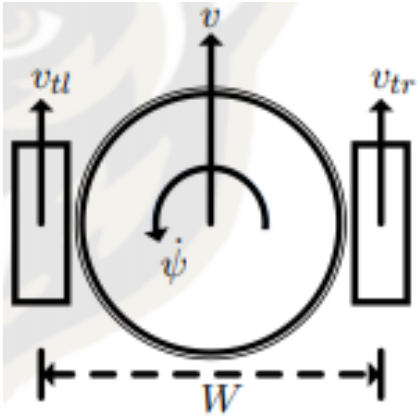

Figure 1: The Roundbot

The Roundbot movement commands are published by the ROS navigation stack. They are determined by speed command in the "linear x" direction and in the "angular z" direction. The Roundbot drives each wheel independently. The forward and inverse kinematics of the Roundbot are shown in Figure 2.

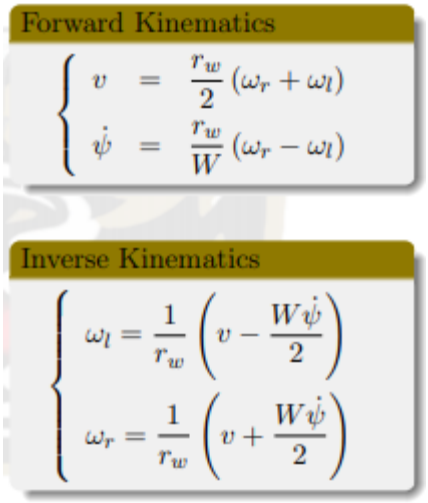

Figure 2: Roundbot kinematics

ROS Navigation Stack Setup

The ROS Navigation Stack enables the robot to autonomously avoid obstacles in its way to its goal using move base parameters. Move base parameters are implemented in the launch file. These parameters are instantiated using YAML files. The YAML files develop the global and local costmaps for the robot as well as a global and local planner. The global planner receives the desired waypoint and the global costmap. The local planner receives the desired path from the global planner and the local costmap. The global costmap refers to the boundaries of the entire course before the robot traverses it. The local costmap updates in real time based on the Lidar sensors obstacle detection. A flow chart of this interaction is shown in Figure 3. Figure 4 shows our launch file.

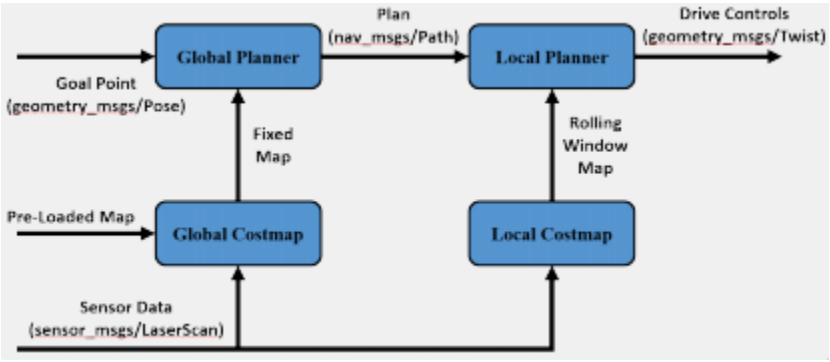

Figure 3: The ROS Navigation Stack

```
<?xml version="1.0"?>
<launch>

  <include file="$(find igvc_flatland)/launch/basic_course_world.launch" >
    <arg name="initial_pose_x" value="-25.0" />
    <arg name="initial_pose_y" value="-1.0" />
    <arg name="initial_pose_a" value="1.5707" />
    <arg name="run_rviz" value="true" />
  </include>

  <node pkg="move_base" type="move_base" name="move_base">
    <rosparam ns="global_costmap" file="$(find igvc_flatland)/yaml/global_costmap_params.yaml" />
    <rosparam ns="global_costmap" file="$(find igvc_flatland)/yaml/global_costmap_mapping_params.yaml" />
    <rosparam ns="local_costmap" file="$(find igvc_flatland)/yaml/local_costmap_params.yaml" />
    <rosparam ns="NavfnROS" file="$(find igvc_flatland)/yaml/global_planner_params.yaml" />
    <rosparam ns="TrajectoryPlannerROS" file="$(find igvc_flatland)/yaml/local_planner_params.yaml" />
    <rosparam file="$(find igvc_flatland)/yaml/move_base_params.yaml" />
  </node>


  <node pkg="tf" type="static_transform_publisher" name="world_to_map" args="0 0 0 0 0 0 world map 30"/>
  <node name="map_server" pkg="map_server" type="map_server" args="$(find igvc_flatland)/maps/basic_course_lines.yaml"/>
  <node pkg="igvc_flatland" type="final_project_basic_north" name="final_project_basic_north"/>
  <node pkg="rviz" type="rviz" name="rviz_igvc_config" args="-d $(find igvc_flatland)/rviz/rviz_igvc_config.rviz" />

</launch>
```

Figure 4: The launch file for basic course north

This launch file first generates the desired course in RViz and instantiates the starting position of the robot. Then the move base parameters are initialized. The move base parameters refer to YAML files in the project package. The YAML file parameters determine the size of the costmaps and designated cost relative to certain situations. Specifically, the costmap size is dependent on the robot radius and inflation parameters. They are shown in Figure 5 and Figure 6.

```
#robot_radius: 0.5
footprint: [[-0.457, 0.304], [-0.457, -0.304], [0.457, -0.304], [0.457, 0.304], [-0.457, 0.304]]
```

Figure 5: YAML screenshot of robot dimensions

```
inflation:
  inflation_radius: 1.0
  cost_scaling_factor: 2.0
```

Figure 6: YAML screenshot of inflation parameters

The robot wants to take the lowest cost path to its navigation goal. The robot dimensions determine the highest cost areas of the costmap which is designated with a blue color. The inflation parameter adds additional cost to the costmap, so the robot doesn't get too close to obstacles. This is designated in red. The cost scaling parameter is an exponential decay factor that adjusts the cost of the inflation radius. It's important to note that we are utilizing Dijkstra's path planning

algorithm for the navigation path. The costmaps feed that algorithm data which constantly optimizes and updates the robot's path.

Finally, the launch file localizes the frames by transforming the world frame to the map frame. It runs the map server node to generate the global costmaps, runs the waypoint algorithm, and launches a specific RViz configuration.

Algorithm

The Move Base parameters within the navigation stack provided the inputs to move the robot. The functionality of this could be tested within RViz. A navigation goal could be selected, and the robot could be monitored moving through the course. All of this needed to be automated so it would run within the launch file.

The algorithm utilizes the actionlib package to publish a move base goal to the robot. This works by instantiating a move base goal using the action client and then monitoring the action client's status relative to that move base goal. The move base goals are then updated using a case statement to set the next goal once the previous goal has been hit. A screenshot of this is shown in Figure 7.

The strategy behind the case statements was to trigger a new location based off of the action library state results. Once the nav goal is sent, the state is monitored to know when the location has been reached. Every time there was a state change to "SUCCEEDED" the "current waypoint parameter was incremented by 1, triggering the next case and navigation goal input.

There were initial problems when setting a navigation goal. The robot would move to the desired location but when the second navigation goal was input, the position did not align with the desired second waypoint. Moreover, it was not always consistent in the overall location on the map. This issue was corrected when the goal target frame ID was changed from "base" to "map". This ensured the nav goal inputs were related to the global map and not to the moving robot. Testing showed that the navigation goal location inputs were accurate and repeatable. Many of the nav goals targeted the waypoint centers but there were instances where locations were moved within the waypoint or added to better position the robot before it received its next navigation goal. Once this was completed the entire system could be run with the course launch file to test the input parameters as the robot traversed the course.

```
typedef actionlib::SimpleActionClient<move_base_msgs::MoveBaseAction> MoveBaseClient;

//we'll send a goal to the robot to move 1 meter forward
goal.target_pose.header.frame_id = "map";
goal.target_pose.header.stamp = ros::Time::now();

goal.target_pose.pose.position.x = 21.92;
goal.target_pose.pose.position.y = 15.65;
goal.target_pose.pose.orientation.w = 1.0;

ROS_INFO("Sending goal");
ac.sendGoal(goal);

ac.waitForResult();
ROS_INFO("Waiting for result");

if(ac.getState() == actionlib::SimpleClientGoalState::SUCCEEDED){
    ++current_waypoint;
}
else
     ROS_INFO("The base failed to move towards goal");
```

Figure 7: Screenshot of the algorithm for the "Basic Course North"

Implementation

This section analyzes how the ROS Navigation Stack and algorithm were implemented to navigate the Roundbot through the IGVC course. A screenshot of the global costmaps are shown in Figure 8. The local costmaps are shown in Figure 9. The green line in the figures shows the robots plan to the goal based on the costmaps.

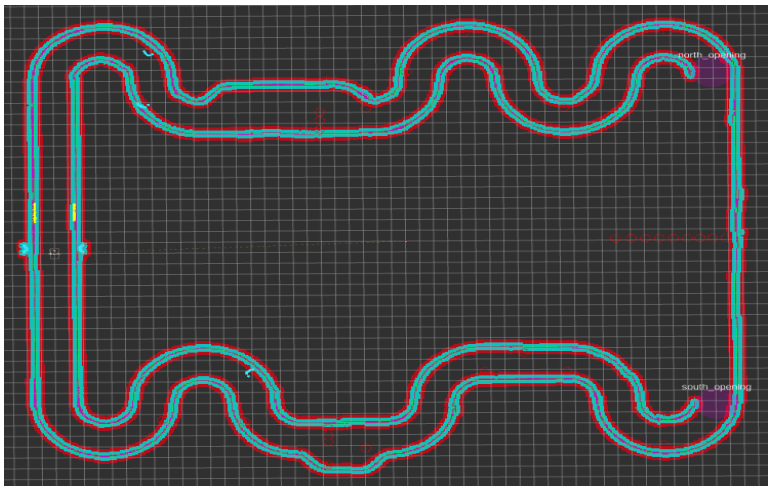

Figure 8: Global costmaps for the Advanced Course North

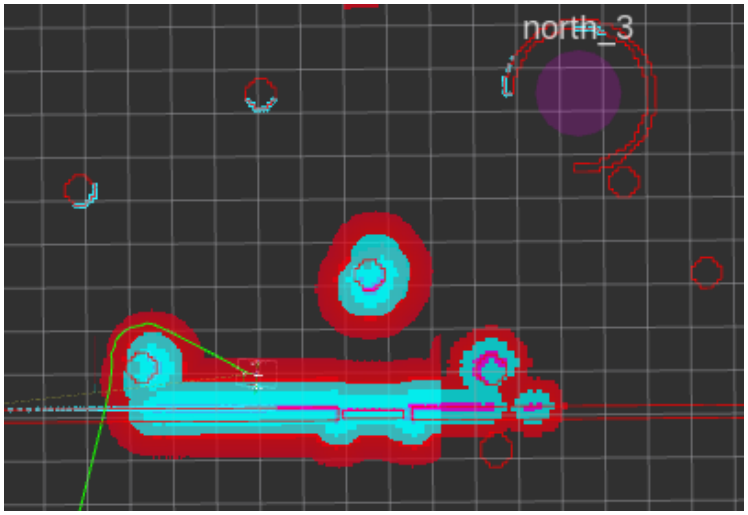

Figure 9: Local costmaps

The transformation tree is shown in Figure 10. This shows how the world frame is localized to the map frame of the simulation. It also shows how the base frame of the robot relates to its wheels and the sensors. There are two lidar sensors on this robot, one to detect obstacles and one that detects the lanes of the course. Figure 11 shows the rqt graph which demonstrates how the topics communicate with each other and with the algorithm.

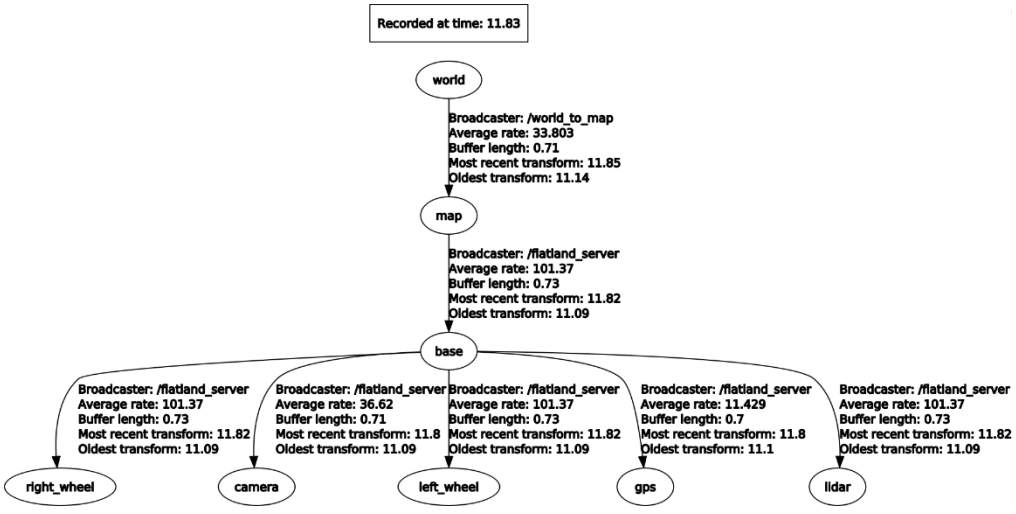

Figure 10: The transformation tree

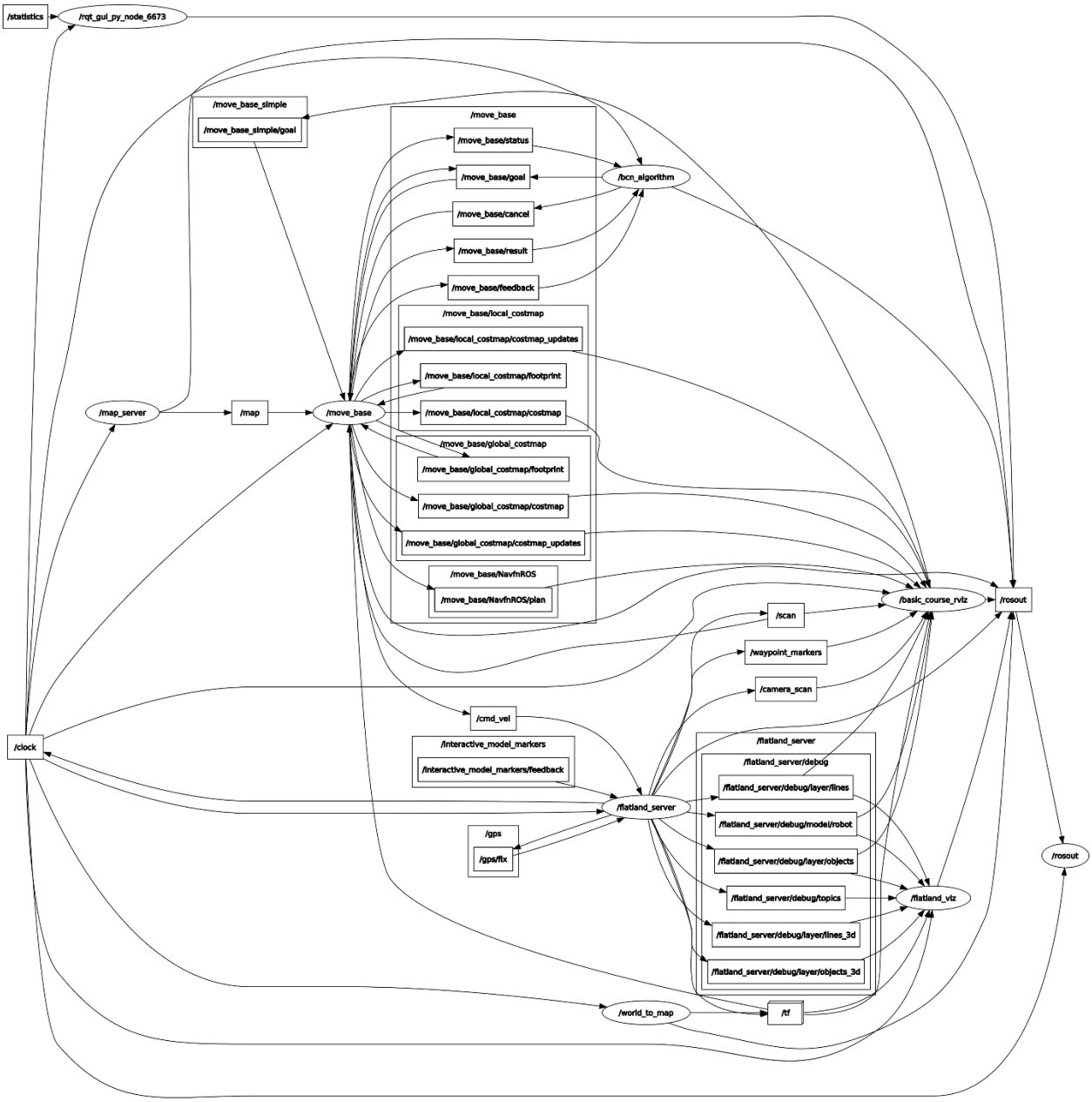

Figure 11: The rqt graph for advance course north

Results

This system's utilization of the ROS Navigation Stack and the algorithm were able to complete the advanced and basic IGVC courses. The time to beat was 6min for the basic course and 10min for the advanced. We bet the basic course in 5min and the advanced course in 8min.

In conclusion, we were successfully able to complete the IGVC basic and advanced course using ROS. It was not perfect however, as the ROS Navigation Stack parameters do not successfully complete the course 100% of the time by itself. Additional waypoints were added to guide the robot out of tricky situations. This is the primary area of improvement relative to this project. Overall, since the robot was able to complete the advanced course successfully, as shown in the demonstration video, this project was a success.

Honorable Mention

Initially, we thought there was a 3D gazebo model of the IGVC course already made that works with this simulation. Turns out that wasn't the case. But, before that was figured out, R3D3 was developed. R3D3 is variant of the robot R2D2 from Star Wars. This robot was developed by disconnecting the mesh files from the Mantis robot URDF files and then creating the frame completely manually (inspiration took from ROS wiki).

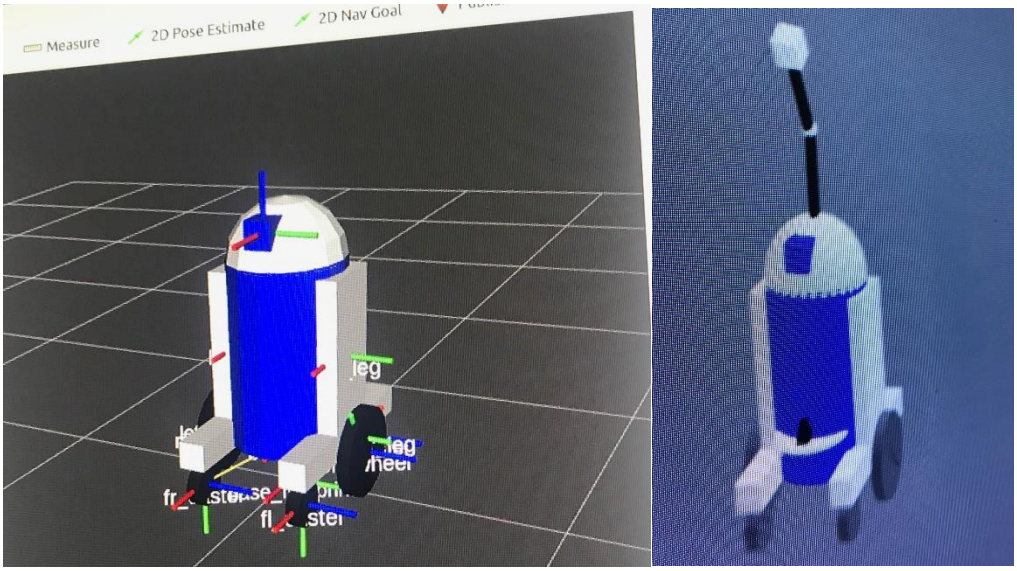

Figure 12: R3D3 model: RViz on left, Gazebo version on right

References

Link to algorithm code: http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals

Link to ROS wiki on Move Base: http://wiki.ros.org/move_base