Assignment 2: ROS SLAM for a Gazebo Simulated Robot Simulation Modelling and Optimisation 282 758

Marc Alexander Sferrazza
12164165 *†

*This work was not supported by any organization

†Faculty of Mechatronics Engineering, Massey University, Albany, Auckland, New Zealand Progress of project: https://github.com/alex1v1a/Simulation-Modelling-and-Optimisation/

Contents

List of Figures

1. INTRODUCTION

ROS (Robot Operating System) robotics development platform is a powerful tool for acquiring and running simulation techniques. The OS along with a simulator can help so quickly test algorithms, regression, train AI and help to design robots to a more effective standard of testing before going to a build stage.

While it may not be the only or best development platform, it has support for many different simulation tools and is a great, easy environment to learn in. ROS is a free linux tool and is supported by the community.

ROS supports many different simulators and for purpose of this courses' learning development, Kinetic Kame will be used with Gazebo simulation population tool. Gazebo is used as it is also a powerful and free tool supported by the community.

Gazebo is a great tool for simulating populations of complex systems in any environment with its high quality graphics and physics engine. Not only is it convenient to import and preform operations on different selected robots, it also has a great graphical interface.

Please refer to https://github.com/alex1v1a/Simulation-Modelling-and-Optimisation/tree/master/Khalid/A2 for full details and code for this assignment

1.1. SLAM and Gmapping

The goal of SLAM (Simultaneous Localisation And Mapping) for mobile robots is to provide a means for a robot to detect and manoeuvre around objects autonomously locally without means of external navigation sources such as GPS etc and the ability to identity the differences between a wall, an object that may have moved, and temporary obstructions such as peoples feet walking by.

and can be found in more detail on from SLAM for dumies

$https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf$

The navigation stack uses the setup of the create model with the laser to locally navigate to set waypoints. Without means of GPS and only local

The diagram below shows this configuration, which was used from tutorial and edited to the tailor needs of the create model.

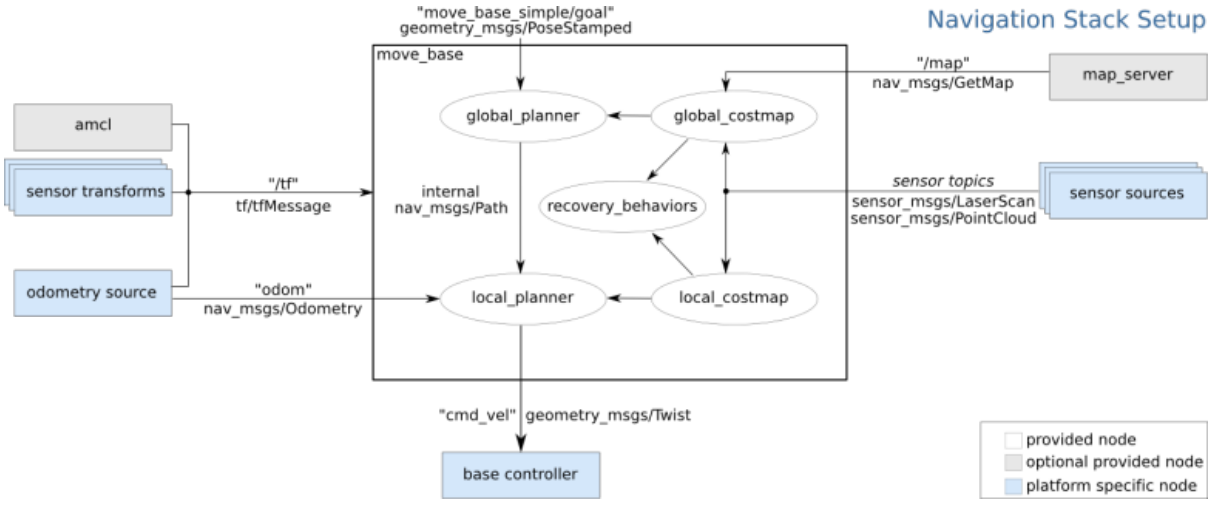


Figure 2: RobotSetup tutorial navigation stack setup

2. METHOD

A detailed description of what was done, how, and why; using the mapping server, the SLAM nodes, installing plugins and running the pose array waypoint to find in rivz etc.

Using an appropriate sensor model (laser scanner or RGB-D camera) mounted on the create robot in Gazebo, an optimised system will preform SLAM in a robust manner. If an obstacle or object is added to the environment of the robot, it can update the map and navigate from point to point in an autonomous mode.

2.1. Laser Scanner SDF

From the previous assignment the Hokuyo URG laser scanner will be used as the sensor for the navigation stack to orient and position itself in the map accordingly, in turn this helps the robot find the waypoints in the most optimal path. Below is the code snippet for the base laser for the topic to Rviz etc positioned on the create robot.

```
 <!-- Hokuyo Laser -->
<link name="hokuyo_link">
  <pose>0.05 0 0.11 0 0 0</pose>
  <inertial>
    <inertia>
      <ixx>1e-6</ixx>
      <ixy>0</ixy>
      <ixz>0</ixz>
      <iyy>1e-6</iyy>
      <iyz>0</iyz>
      <izz>1e-6</izz>
    </inertia>
    <mass>1e-5</mass>
  </inertial>
  <collision name="collision">
    <pose>0 0 0.5 0 0 0</pose>
    <geometry>
      <box>
        <size>0.1 0.1 0.1</size>
      </box>
    </geometry>
  </collision>
  <visual name="visual">
    <geometry>
      <mesh>
        <uri>model://hokuyo/meshes/hokuyo.dae</uri>
      </mesh>
    </geometry>
  </visual>
  <sensor name="laser" type="ray">
    <pose>0.01 0 0.0175 0 -0 0</pose>
    <ray>
      <scan>
        <horizontal>
          <samples>640</samples>
          <resolution>1</resolution>
          <min_angle>-2.26889</min_angle>
          <max_angle>2.268899</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.08</min>
        <max>10</max>
        <resolution>0.01</resolution>
      </range>
    </ray>
    <plugin name="laser" filename="libRayPlugin.so" />
    <always_on>1</always_on>
    <update_rate>30</update_rate>
    <visualize>true</visualize>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
    <topicName>/scan</topicName>
    <frameName>base_laser</frameName>
    </plugin>
  </sensor>
</link>

 <joint name="hokuyo_joint" type="revolute">
  <parent>base</parent>
  <child>hokuyo_link</child>
  <axis>
    <xyz>0 1 0</xyz>
    <limit>
      <upper>0</upper>
      <lower>0</lower>
    </limit>
  </axis>
```

Figure 3: The model iRobot create sdf 1.4 version is used, the Hokuyo URG laser scanner is used

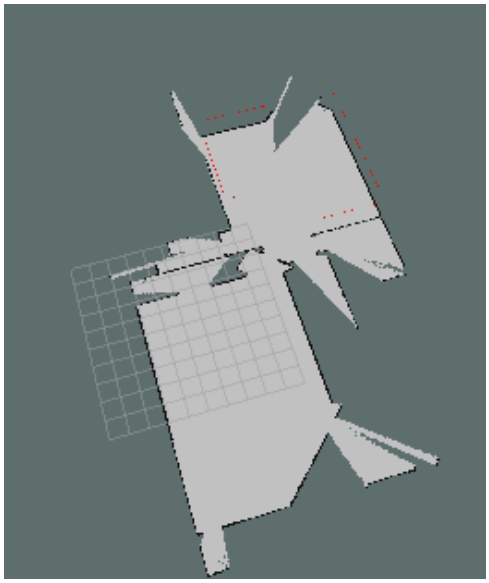
2.2. Pre-Requisites

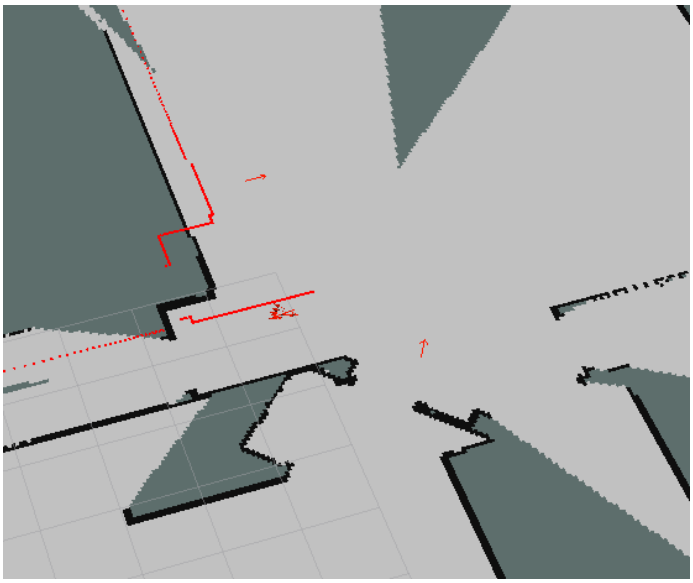
In order for the 2D map to be of use, the navigation stack must be able to find the local variables and adjust accordingly. There are several pre-requisites for this to engage which include transform configurations, sensor sources(laser scanner RGB-D cam etc), odometry source, the base controller(move base), and the map server. For a more detailed explanation for the installation of these packages please refer to assignment one, and the appendix links used tutorials.

The navigation stack setup envelopes several steps of creating a package of the base, transform, odom, sensor configurations; and using the pre-configured launch file and common configurations tweaked for local and global costmap, base planner, the navigation stack is then setup and ready to be launched. Please see RobotSetup ROS tutorial link in the references for more details.

Before starting the navigation stack however, waypoints will need to be setup to act as goals to send to the navigation stack. Follow waypoints package is installed, with this package the navigation stack now has access to a new node of follow waypoint, that can passes the move base action and will subscribe and publish the Pose waypoints topics to then be set as goals for the navigation stack.

Rviz is used to then send the waypoint positions to the node and once all is set the path ready parameter is initialised. Below is an example of the waypoints in use with the navigation stack. The waypoints package used was made by Daniel Snider and is available on GitHub via the package summary link from the follow waypoints tutorial.

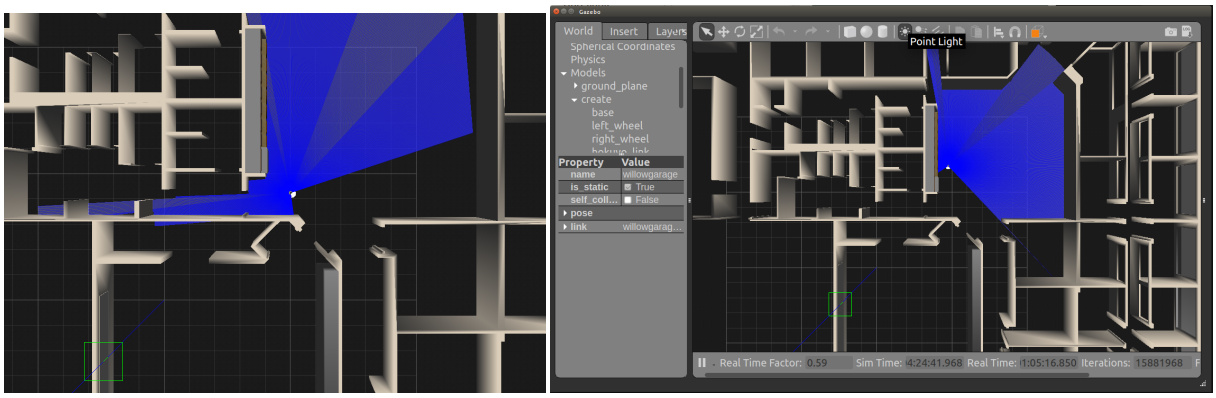




Above shows using the 2D map saved to the pgm file and updated the the navigation stack from the map server the robot can show the path and position to each waypoint, Rviz showing the mapping of the robot. Now the map is successfully registering and passing to the stack the waypoints can be added. The laser sensor is shown with the red lines along the walls, with the odom as red arrows on the model and waypoints shown in pink as individual smaller arrows.

2.3. Navigation and Waypoints (Plugins)

Before we begin using SLAM and Gmapping with waypoints there are several packages that need to be created and installed; these include navigations, slam gmapping, map server, follow waypoints, and other transform and base packages for odomerty and sensory etc. Once the plugins are setup as described above and are in place the robot is then ready to navigate to goal points positioned form the pose array. Please see the gmapping and RobotSetup tutorial links for the ROS website for step by step setup.



Shown above is the Gazebo graphical representation of the robot moving using SLAM to navigate using scanner attached to find a obstacles and other objects and follow the waypoints set to avoid these with the navigation stack.

Once linked the robot can begin to find itself within a local area to avoid obstacles using the navigation stack. Shown below is the demo of the first manual waypoint set and the beginning of the mapping etc.

Again Shown the rviz using the laser node with obstruction blocks and the subscribed laser topic, the representation is given by flat squares in red.

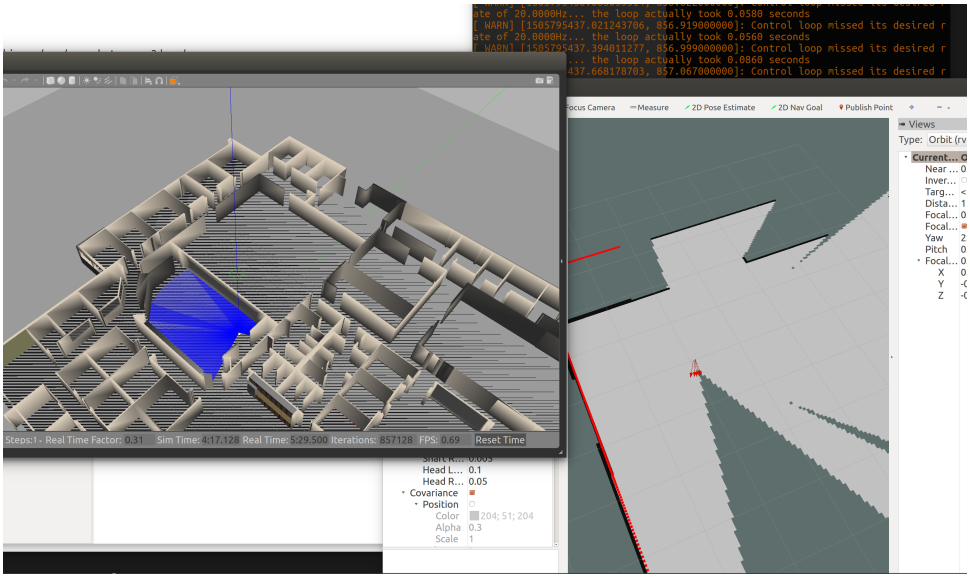
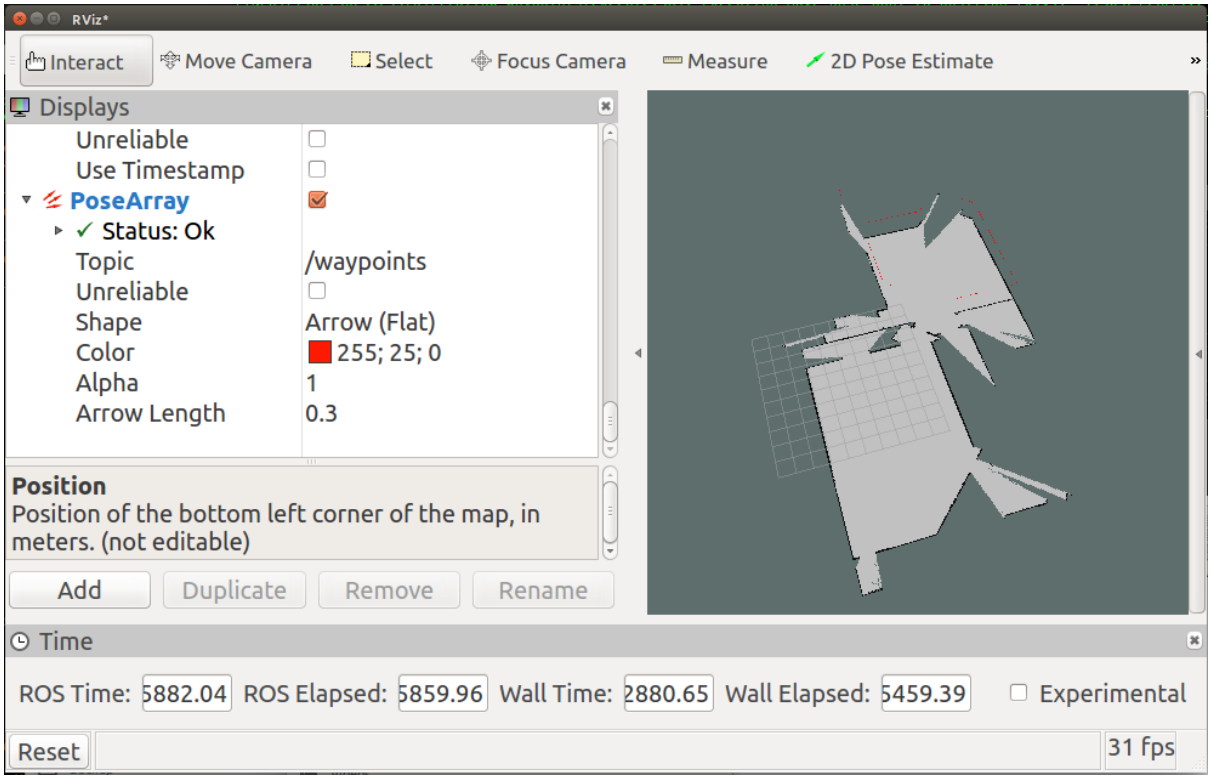
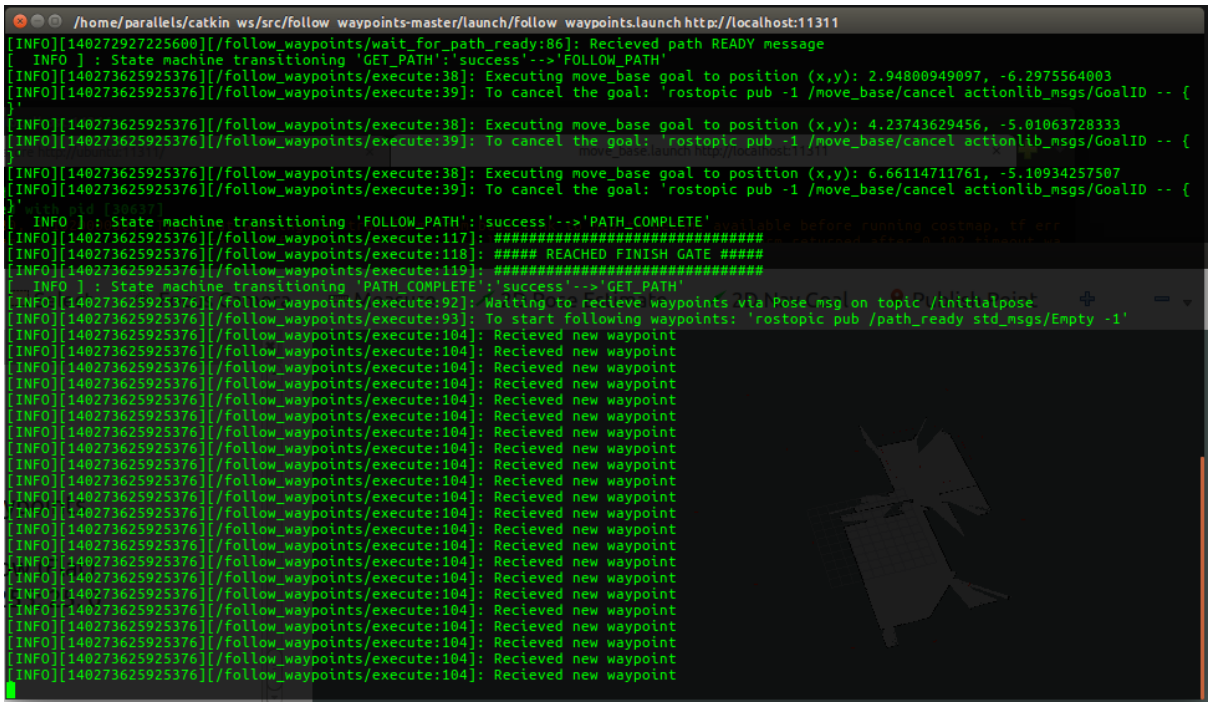


Figure 6: Rviz passed waypoints to the navigation stack

The points are provided by the user in Rviz and is passed to the navigation stack using a modified version of files from the tutorials on ROS website, please see the appendix for links and references.





After the waypoints are sent via the Pose Array the state machine will then transition and issue the Follow Path state; once the path is complete the process will then broadcast get path to say it is ready to recieve the next waypoint instruction, once they have been loaded and set the ready status can be run again to restart this process.

2.4. Pose Array Robot Control with Rviz

When building these packages it is important to look for areas which may require certain tuning to suit the given environment. The consideration for thresholding and inflation radius as well as min and max x speeds and such others are checked to ensure optimal performance when traveling between goals.

While running the path there was an issue discovered with the optimal frequency which due to hardware limitations had to be manually edited to better fit the speed of the computers processing capabilities. The issue which occurred is shown below. The problem was overcome by setting the launch code frequency manually to 5hz.

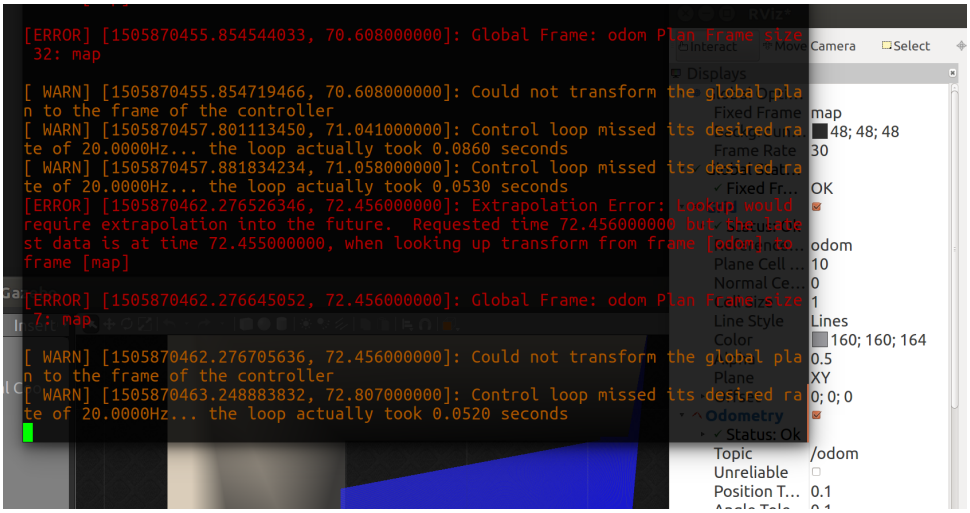


Figure 8: Issue discovered with the optimal running frequency

After fixing all the issues and smoothing things out with trial and error tuning to find the bugs, the robot could now move freely from waypoint to waypoint as shown below

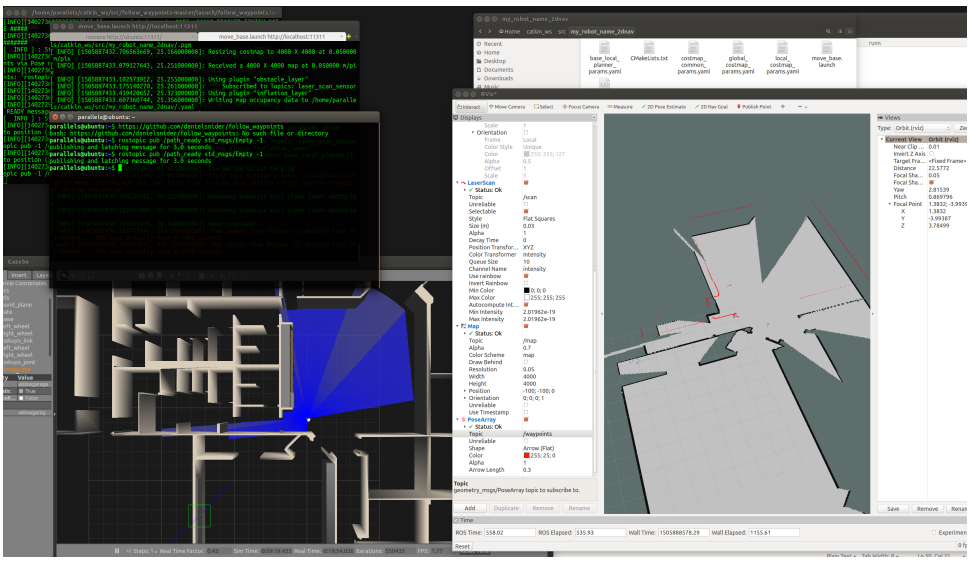


Figure 9: Issue discovered with the optimal running frequency

2.5. Custom Bash Scripts

For ease of use some bash scripts have been made to speed up the process. The scripts step through (first script ”run”) launching roscore, opening gazebo, importing models, creating transforms, running rviz, displaying the topic list for easy debugging, launching GMAP, launch the waypoints, and after a period of time for the points to be set to start the path.

```
echo "Importing model..."
rosrun gazebo_ros spawn_model -file /home/parallels/.gazebo/models/create/model-1_4.sdf -sdf -model create &
pid=$!
sleep 10s

echo "Importing model..."
rosrun gazebo_ros spawn_model -file /home/parallels/.gazebo/models/willowgarage/model-1_4.sdf -sdf -model willowgarage &
pid=$!
sleep 10s

echo "Creating Transforms.."
rosrun tf static_transform_publisher 0 0 0.5 0 0 0 base_link base_laser 100 &
pid=$!
sleep 1s
rosrun tf static_transform_publisher 0 0 1 0 0 0 base_footprint base_link 100 &
pid=$!
sleep 1s
#rosrun tf static_transform_publisher 0 0 0 0 0 0 map odom 100 &
#pid=$!
#sleep 1s

echo "Running Rviz..."
rviz &
pid=$!
sleep 5s

echo "Topic List..."
rostopic list &
pid=$!
sleep 5s

echo "Launching GMAP"
rosrun gmapping slam_gmapping &
pid=$!
sleep 5s

echo "Follow Waypoints"
roslaunch follow_waypoints follow_waypoints.launch
pid=$!
sleep 85s

#echo "Running Teleop..."
#rosrun teleop_twist_keyboard teleop_twist_keyboard.py
#pid=$!
#sleep 5s

echo "Finding Waypoints..."
rostopic pub /path_ready std_msgs/Empty -1
pid=$!
sleep 5s
```

Figure 10: ”run” first bash script

While this first script is running a second script (runn) is called that starts the map server and begins updating the move base.

```
#!/din/bash

echo "Saving map file..."
rosrun map_server map_saver -f /home/parallels/catkin_ws/src/my_robot_name_2dnav/ &
pid=$!
sleep 5s

echo "Move Base"
cd /home/parallels/catkin_ws/src/my_robot_name_2dnav/
roslaunch move_base.launch
pid=$!
sleep 5s

#trap "echo...; kill -2 $pid; exit" SIGINT SIGTERM

sleep 24h
```

Figure 11: ”runn” seccond bash script

Once both these steps are processed the robot then begins to start the path from waypoint to waypoint and saves a pgm map as it goes detecting walls and other surrounding objects.

3. OUTCOMES

Due to the size of the launch files and packages the entire workspace has been included with all relevant files and uploaded to GitHub. A Readme file has also been created to describe the contents and how to run the bash scripts for ease of use. Please note, few lines may need to be edited in the bash for the user's account, as I have used parallels virtual machine on Mac the account Home user will need to be updated accordingly etc.

Below is a full working demo shot of the create robot being used with the above mentioned navigation tasks, SLAM, the navigation stack, gampping etc

Please refer to https://github.com/alex1v1a/Simulation-Modelling-and-Optimisation/tree/master/Khalid/A2 for full details and code for this assignment

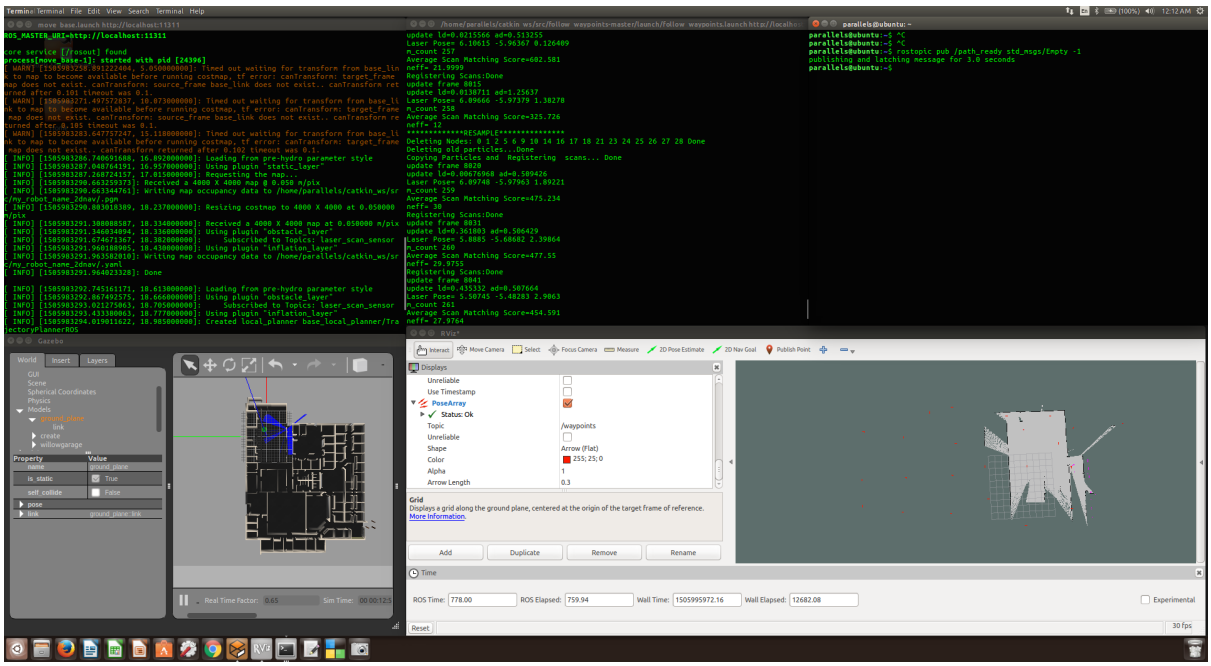


Figure 12: A demo showing the full working model following waypoint goals

The output is then displayed on the rviz graphical demonstration. A custom node is implemented to subscribe to the velocity topic and display the variable speed with a mentioned warning.

4. CONCLUSIONS

For any progress related to the report please see the public Github repo for alex1v1a or use the link in the cover page to be automatically redirected to this project. The repo provides all relative project information and files relevant to the assignment under ROS, Assignment 2

I have come to a further understanding learned more of the fundamentals of ROS using SLAM with gmapping to travel along guided waypoints using the navigation stack with rviz user defined waypoints (posearray).

This is a powerful tool and from the basic level of understanding is an extremely vital aspect of the mechatronics processing, it is a great skill to learn.

References

[1] ROS, "Ros tutorial slam map building with turtlebot." http://wiki.ros.org/turtlebot_navigation/Tutorials/Build%20a%20map%20with%20SLAM. [Online; accessed 18-Aug-2017].

[2] ROS, "Ros tutorial how to use gmapping to build a map." http://answers.ros.org/question/117396/how-to-use-gmapping-to-build-a-map/. [Online; accessed 18-Aug-2017].

[3] ROS, "Ros tutorial how to build a map using logged data." http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData. [Online; accessed 18-Aug-2017].

[4] ROS, "Ros tutorial how to build a map using logged data 2." http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData#download. [Online; accessed 18-Aug-2017].

[5] ROS, "Ros tutorial setup and configuration of the navigation stack on a robot." http://wiki.ros.org/navigation/Tutorials/RobotSetup. [Online; accessed 18-Aug-2017].

[6] ROS, "Ros tutorial what does tf do why should i use tf." http://wiki.ros.org/tf. [Online; accessed 18-Aug-2017].

[7] ROS, "Ros tutorial sending goals to the navigation stack." http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals. [Online; accessed 18-Aug-2017].

[8] ROS, "Ros tutorial follow waypoints." http://wiki.ros.org/follow_waypoints. [Online; accessed 18-Aug-2017].

[9] ROS, "Github repo for follow waypoints used." $https://github.com/danielsnider/follow_waypoints$. [Online; accessed 18-Aug-2017].