Assignment 1: Mobile Robot ROS Gazebo
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

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.

1.1. Installation

Due to having an Apple Macbook with Windows already partitioned on the hybrid, my personal workspace has been setup using a virtual machine.

Tri-boot is possible using a GRUD push loader with uEFI boot manager for both MBR, and GUID - However due to limitations on hardware and internal memory (SSD space) a virtual machine was necessary for this build - please excuse some continuity graphics issues, and screen shots etc.

There are a few installation methods for various supported operating, while OSX support is shown as "Experimental" this option did not work with the setup unfortunately.

Figure 2: Installation choices

2. METHOD

A detailed description of what was done, how, and why; using roscore, the teleop node, installing plugins and running gazebo and rivz etc.

2.1. Block Diagram of running model

Using rosrun the run-sapce can initialise a Gazebo window. After loading up Gazebo a model can be spawned, in this case the iRobot vacuum cleaner is used. The model has been pre-edited with a plugin ”differential controller” to enable control via a node ”teleop node” - with the teleop node python script it is then possible to control the device. After activating a movement ”rviz” is then used to capture the odometry data and produce a graphical representation of the movement with both position and direction.

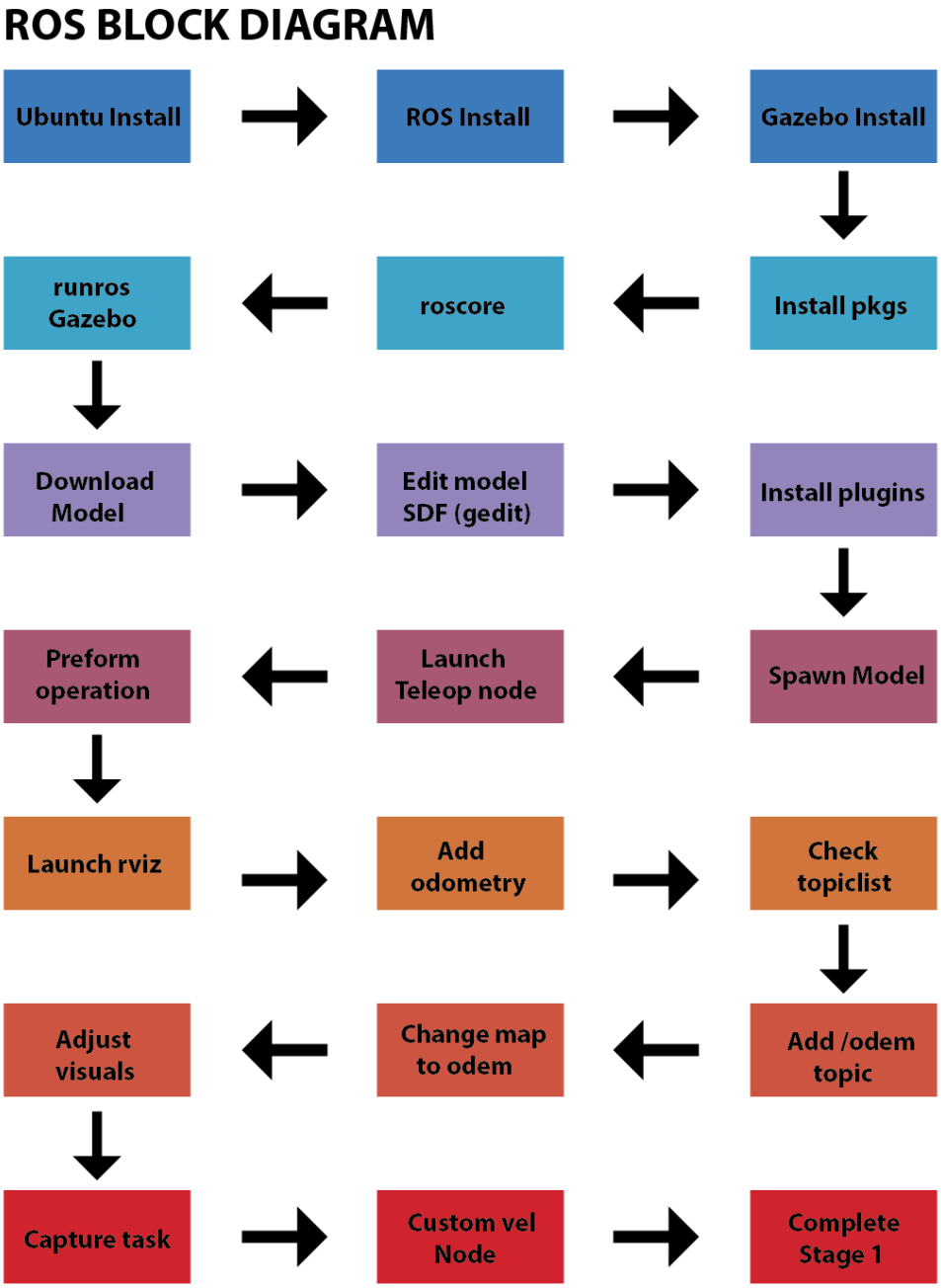


Figure 3: A model iRobot Create spawned in Gazebo using teleop node for movements process diagram

Below is a figure of the ROS and Gazebo setup showing the communication between nodes. Please note the reference from the ROS tutorials in the active diagram.

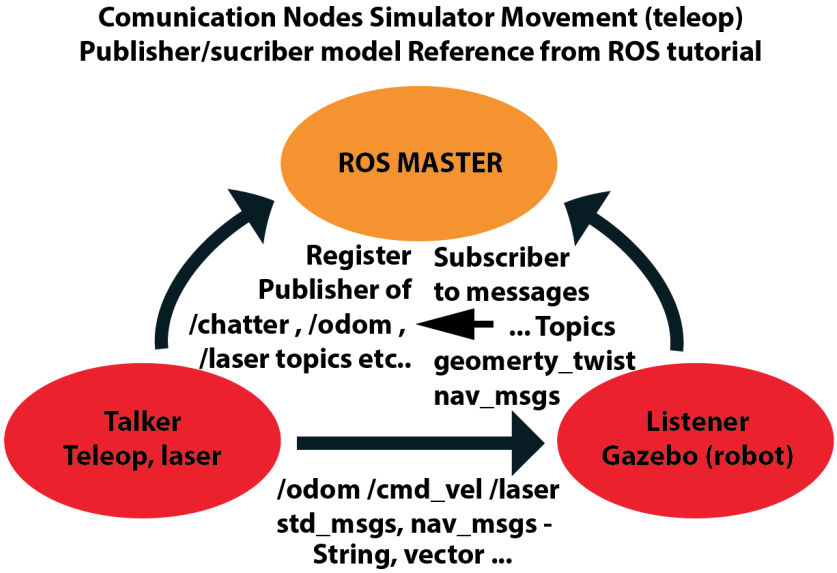


Figure 4: ROS and Gazebo setup showing the communication between node

Using the rqt graph command we can see that the active nodes, for this example teleop being the publisher, and gazebo the subscriber, are passed the cmd velocity; to move the differential drive for a given specific velocity and direction to describe which way for the robot to move.

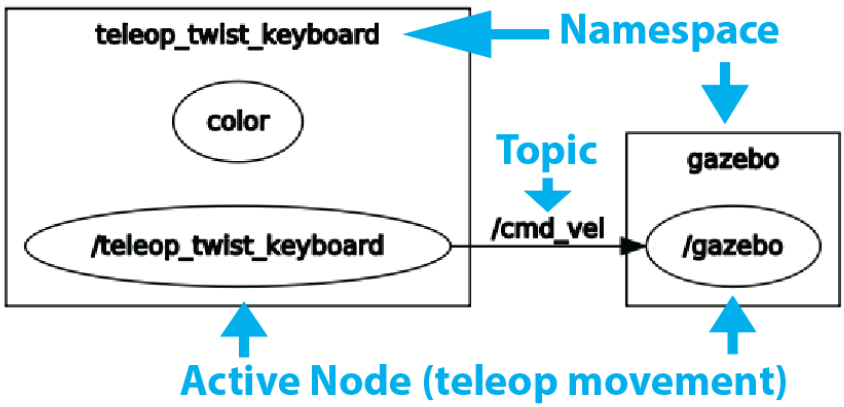
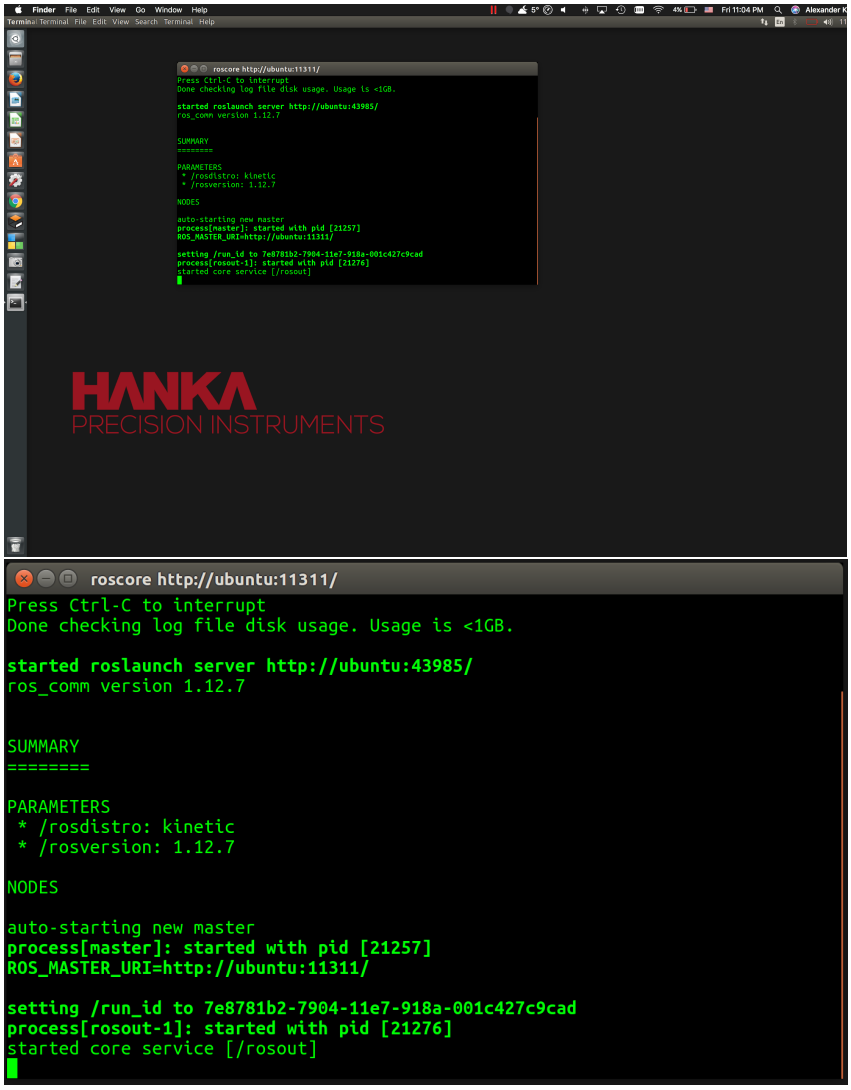


Figure 5: rqt graph showing active nodes

2.2. ROSCORE

Roscore is the fundamental processing command on which ROS can activate the simulation environment; from here gazebo can spawn models which can be edited with plugins and controlled via nodes as explained later.



Above is a demonstration of ROSCORE running in a terminal on Ubuntu operating system. This is currently in coherent mode but for better performance in later tasks the virtual runtime is switched to windowed mode.

2.3. ROS Nodes (Plugins)

Using some of the default plugins provided from the ROS package site, in this case the ”Differential Drive” we can assign the left and right joints (wheels) to the controller. The controller in turn will take the response value passed from the Teleop Node and preform the movement in the joint.

Differential Drive

Description model plugin that provides a basic controller for differential drive robots in Gazebo. You need a well defined differential drive robot to use this plugin.

```
<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>${update_rate}</updateRate>
    <leftJoint>base_link_right_wheel_joint</leftJoint>
    <rightJoint>base_link_left_wheel_joint</rightJoint>
    <wheelSeparation>0.5380</wheelSeparation>
    <wheelDiameter>0.2410</wheelDiameter>
    <torque>20</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>base_footprint</robotBaseFrame>
  </plugin>
</gazebo>
```

model-1_4.sdf (~/.gazebo/models/create) - gedit

Open

```
        <friction>
          <ode>
            <mu>10</mu>
            <mu2>10</mu2>
            <fdir1>0 0 0</fdir1>
            <slip1>0</slip1>
            <slip2>0</slip2>
          </ode>
        </friction>
      </surface>
    </collision>
    <visual name="visual">
      <pose>0 0 0 0 1.570700 1.570700</pose>
      <geometry>
        <cylinder>
          <radius>0.033000</radius>
          <length>0.023000</length>
        </cylinder>
      </geometry>
    </visual>
  </link>

  <joint name="left_wheel" type="revolute">
    <parent>base</parent>
    <child>left_wheel</child>
    <axis>
      <xyz>0 1 0</xyz>
    </axis>
  </joint>

  <joint name="right_wheel" type="revolute">
    <parent>base</parent>
    <child>right_wheel</child>
    <axis>
      <xyz>0 1 0</xyz>
    </axis>
  </joint>

  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <alwaysOn>true</alwaysOn>
    <updateRate>${update_rate}</updateRate>
    <leftJoint>right_wheel</leftJoint>
    <rightJoint>left_wheel</rightJoint>
    <wheelSeparation>0.5380</wheelSeparation>
    <wheelDiameter>0.2410</wheelDiameter>
    <torque>20</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>base_footprint</robotBaseFrame>
  </plugin>

  </model>
</sdf>
```

Note: The highlighted text is copied directly without the ”gazebo” tags; also once the plugin is added to the model sdf file it is then edited with gedit to match the joint name values e.g. left wheel and so on.

Again we can do this for the laser plugin and add the plugin to the create model 1.4

```
<!-- hokuyo -->
<gazebo reference="hokuyo_link">
  <sensor type="gpu_ray" name="head_hokuyo_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>false</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>30.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo laser
             achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
             stddev of 0.01m will put 99.7% of samples within 0.03m of the true
             reading. -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_gpu_laser.so">
      <topicName>/rrbot/laser/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

Figure 8: laser plugin node setup

Here is the model with the laser scanner attached

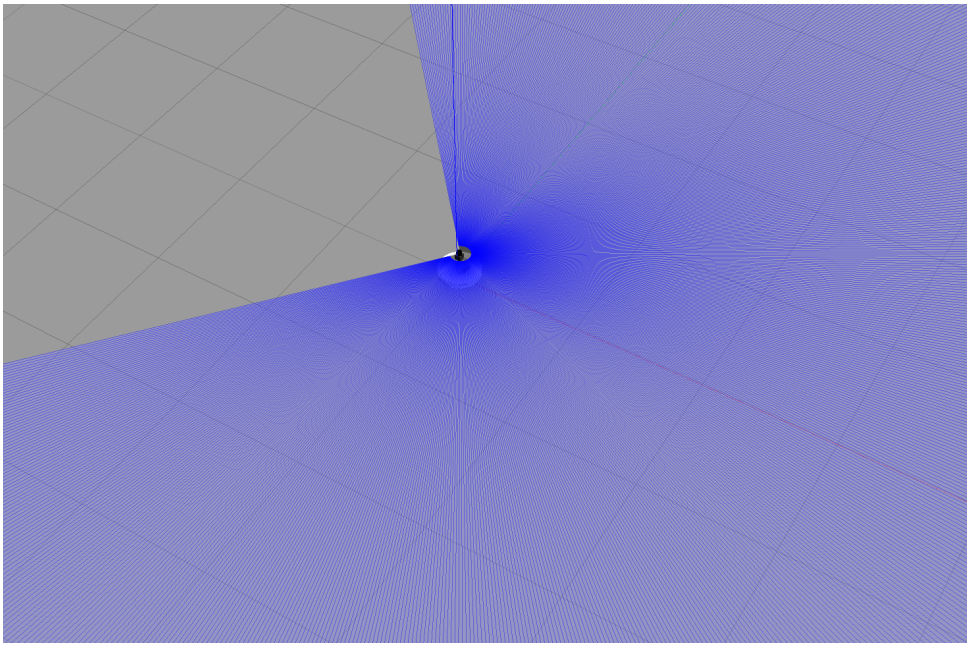


Figure 9: laser plugin node mesh graphic

After the plugin is assigned to the sdk file the subscriber nodes can be setup. By creating a package and compiling it, the executables can then be run and useful output given from various broadcasted topics.

Below is a demonstration of the odometry topic, and a node being used to give the user a notice when the 5 meter mark has bene reached in along the x and y axis.

```
#include "ros/ros.h"
#include "nav_msgs/Odometry.h"

int boundary_x = false;
int boundary_y = false;

void chatterCallback(const nav_msgs::Odometry::ConstPtr& msg)
{
  if(msg->pose.pose.position.x < 5 && boundary_x){
    ROS_INFO("in range of x axis");
    boundary_x = true;
  }
  if(msg->pose.pose.position.x < 5 && boundary_y){
    ROS_INFO("in range of y axis");
    boundary_y = true;
  }
  if(msg->pose.pose.position.x > 5 && boundary_x){
    ROS_INFO("5m mark has been exceeded along the x axis");
    boundary_x = false;
  }
  if(msg->pose.pose.position.x > 5 && boundary_y){
    ROS_INFO("5m mark has been exceeded along the x axis");
    boundary_y = false;
  }
}


int main(int argc, char **argv)
{
  ros::init(argc, argv, "listener");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("odom", 1000, chatterCallback);
  ros::spin();
  return 0;

}
```

Figure 10: Subscriber odom node setup

Here is the cpp code for the subscriber to the cmd velocity topic.

```
#include "ros/ros.h"
#include "geometry_msgs/Twist.h"

void listen2(const geometry_msgs::Twist::ConstPtr& vel)
{
  ROS_INFO("LINEAR : x: [%f] y: [%f] z: [%f]", vel->linear.x, vel->linear.y, vel->linear.z);
  ROS_INFO("ANGULAR: x: [%f] y: [%f] z: [%f]", vel->angular.x, vel->angular.y, vel->angular.z);
}


int main(int argc, char **argv)
{
  ros::init(argc, argv, "odom_listener");
  ros::NodeHandle n;
  ros::Subscriber sub = n.subscribe("cmd_vel", 1000, listen2);
  ros::spin();
  return 0;
}
```

Figure 11: Subscriber cmd vel node setup

Then the new cpp files are added to the cmake file to be compiled as executables the next time catkin make command is called.

listener.cpp | listener2.cpp | CMakeLists.txt

```
## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_beginner_tutorials.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)

include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(talker src/talker.cpp)
target_link_libraries(talker ${catkin_LIBRARIES})
add_dependencies(talker beginner_tutorials_generate_messages_cpp)

add_executable(listener src/listener.cpp)
target_link_libraries(listener ${catkin_LIBRARIES})
add_dependencies(listener beginner_tutorials_generate_messages_cpp)

add_executable(listener2 src/listener2.cpp)
target_link_libraries(listener2 ${catkin_LIBRARIES})
add_dependencies(listener2 beginner_tutorials_generate_messages_cpp)
```

Figure 12: nodes executables setup

After completing and running the new node when the subscriber receives information from the broadcaster; it is then displayed on the console like the following.

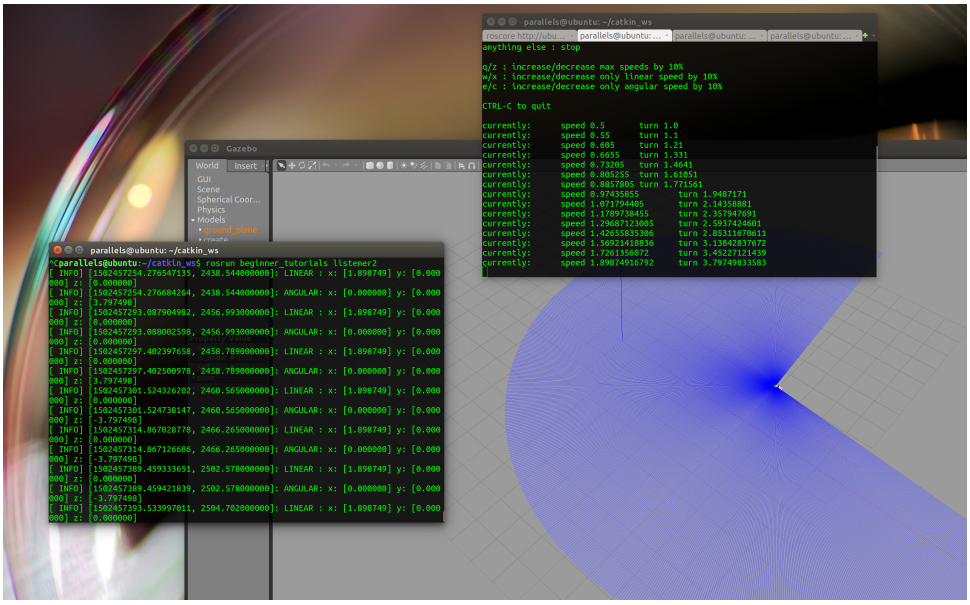
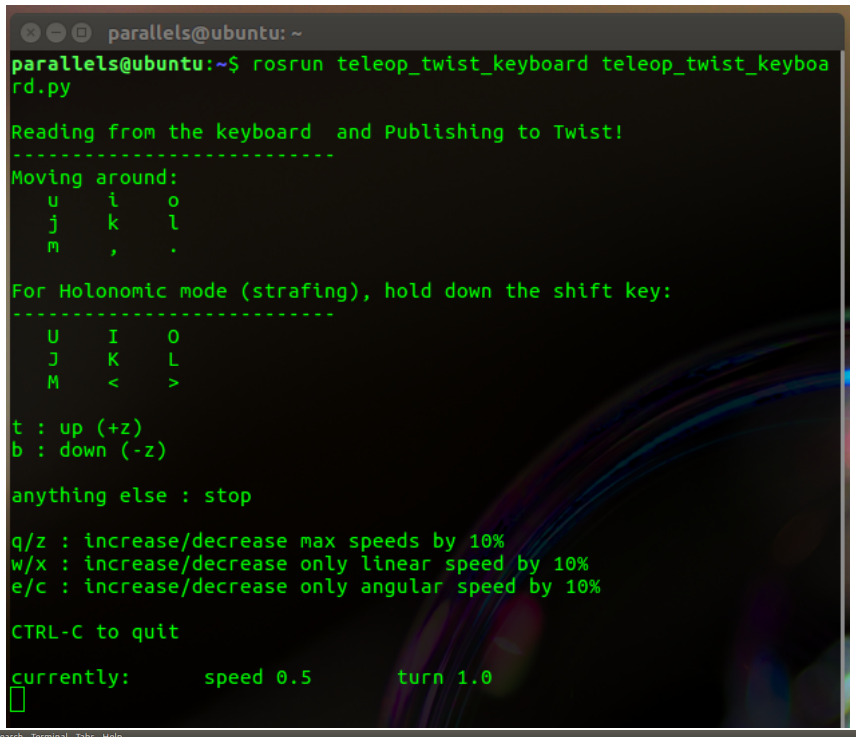
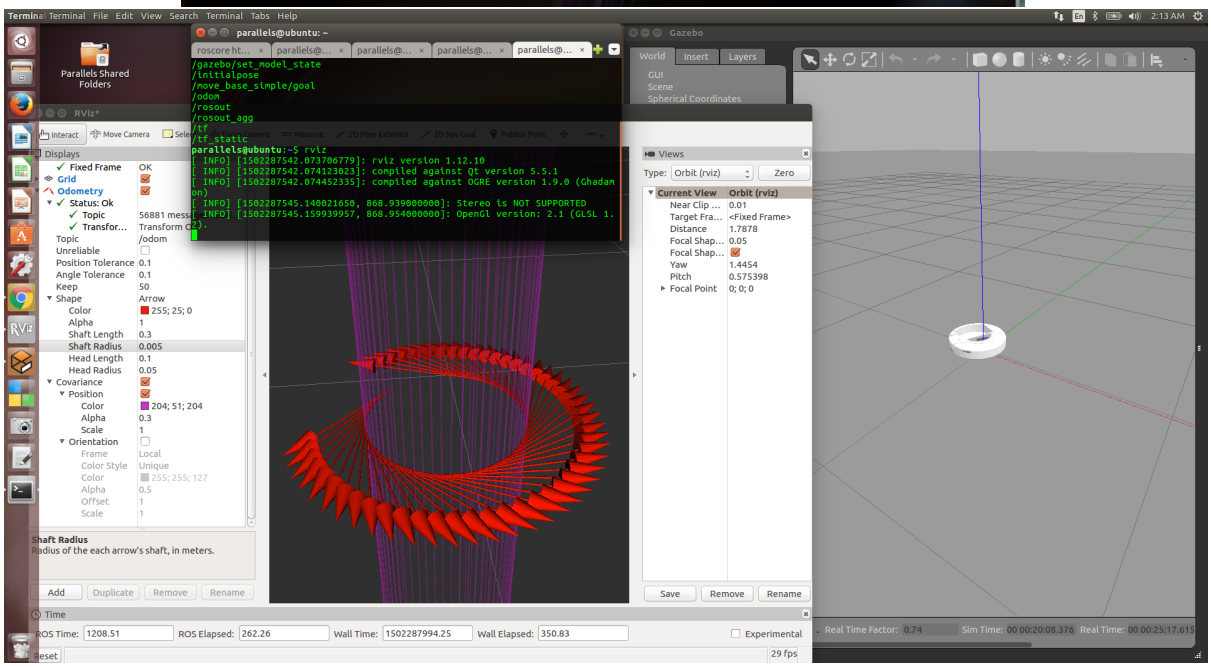


Figure 13: A demo showing the cmd vel node being executed, output from the moving model

2.4. Teleop Node Robot Control with Rviz

Below is a demonstration of the Teleop Node used to control the left and right wheels with the plugins mentioned earlier.





Shown above is the rviz graphical representation of the odometry, the purple vertical lines represent the position while the red arrows show the directional vector. Rviz is great for graphical demos of what the robot interacts with in the model environment, e.g a scanner attached to find a obstacles and other objects.

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.

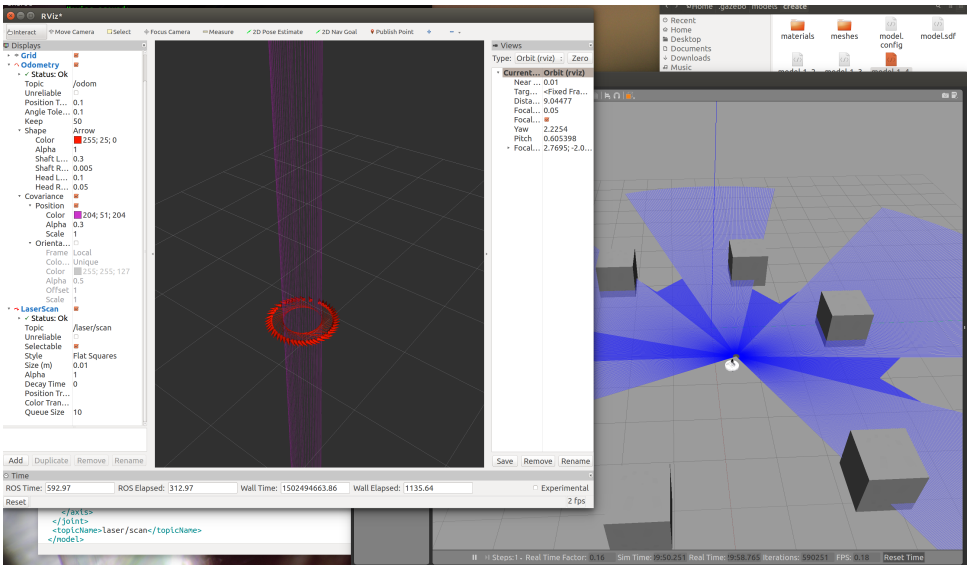


Figure 15: A demo showing the cmd vel node being executed, output from the moving model

3. OUTCOMES

Through the above steps ROS has been successfully installed, the Gazebo runtime is used with the catkin workspace to spawn a model with edited plugins that are able to be controlled via the teleop node.

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

I have come to a further understanding learned the fundamentals of ROS with the use of such simulation tools like Gazebo. I am aware of other simulation tools available but due to the community support this is a great place to start. The use of plugins with models to be controlled by the teleop python script is used in calibration with the rviz graphical output to help visualise the robots (models) sight.

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.