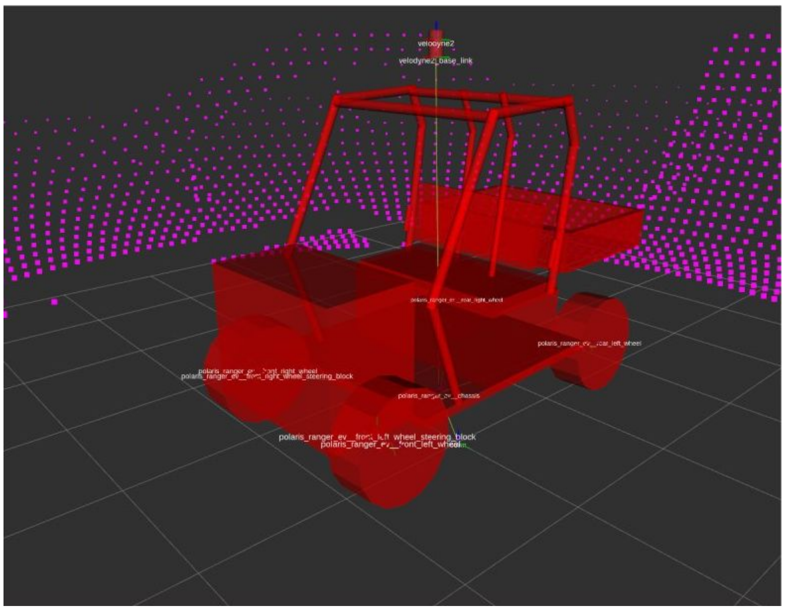
Aalto University, School of Electrical Engineering
Automation and Electrical Engineering (AEE) Master's Programme
ELEC-E8004 Project work course
Year 2019

# Final Report

# Project 12
# Building a Simulation Platform for Polaris e-ATV in ROS using Gazebo



Date: 31.5.2019

Lukas Wachter
Onur Sari
Valtteri Karhu
Panu Pellonpää

# Information page

Students
Lukas Wachter
Onur Sari
Valtteri Karhu
Panu Pellonpää

Project manager
Lukas Wachter

Official Instructor
Tabish Badar

Other advisors
Mika Vainio
Arto Visala

Starting date
10.1.2019

Completion date
31.5.2019

Approval
The Instructor has accepted the final version of this document
Date: 31.5.2019

## Abstract

The objective of this project was to design a Universal Robot Description File (URDF) for the Polaris Ranger Electric All Terrain Vehicle (e-ATV). The URDF file is equipped with all the necessary sensors for generating topics to test control algorithms, navigation routines and SLAM procedures on the real-life counterpart. The project used the open-source platform Robot Operating System (ROS) along with the RViz and Gazebo programs to design the platform for testing the URDF functionality.

The URDF file was designed by using an existing Simulation Description Format (SDF) file as the foundation. SDF is a more advanced file format for describing robots. However it does not function in ROS. Gazebo uses the SDF format to better integrate robots into the world, while ROS is only concerned with the robot itself. The resulting URDF file was compared step-by-step with manufacturer datasheets of the Polaris Ranger EV and verified in the ROS environment. Major adjustments were necessary to make the file viable, due to backwards incompatibility between the SDF and URDF formats.

The robot described by the URDF file was simulated in Gazebo. It simulates the 3D rigid body dynamics of robots along with the static objects in the environment. The 3D environment designed in this project was developed for the purpose of testing the functionality of the URDF file. This includes the motion capabilities of the robot and its behaviour on inclines. Additionally, the sensors' data from the Gazebo environment is visualized in RViz. For example, the 3D point cloud generated by the Light Detection and Ranging (LiDAR) equipment illustrates how the robot sees the virtual environment in Gazebo. Therefore, the designed environment also served as a useful tool to test the performance of a simulated sensor.

# Table of Contents

## 1. Introduction

The robot used in this project was the Polaris Ranger e-ATV with LiDAR, IMU and GPS sensors (Fig. 1). Robots such as the Polaris Ranger can be described with the URDF file format, which is a document written in eXtensible Markup Language (XML) that describes all the necessary properties of a robot. This includes the geometry, joints and controllers required to achieve control and translation in a software environment. URDF files work in the ROS environment and the functionality of URDF files can be simulated by using Gazebo. Gazebo is a set of ROS packages used to simulate 3D rigid body motions. It allows the importing of 3D models, generating LiDAR sources and populating the world with obstacles, which makes it a versatile environment for simulation.

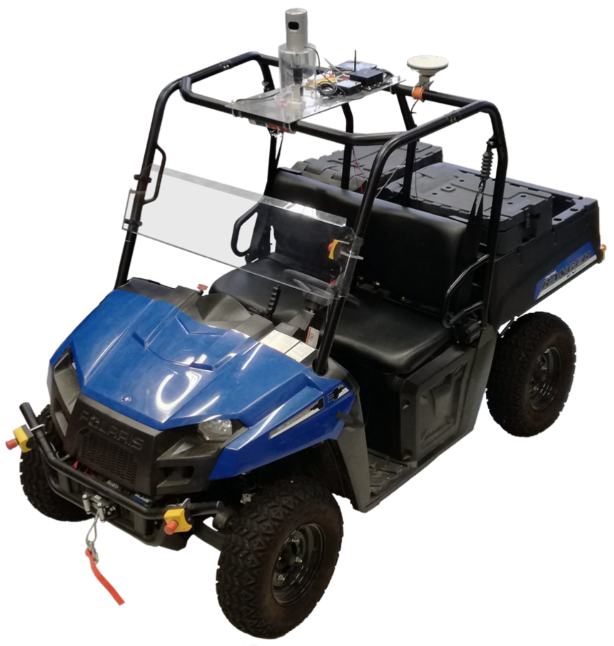


*Figure 1. The real Polaris Ranger with the sensors visible on top of the roll cage.*

Developing a virtual model for a robot helps with testing and experimentation, such that the risk to the real robot can be mitigated by catching some possible errors and design flaws before any experimentation in the real world is conducted. It provides an opportunity to prototype without any material cost. So, that the proper sensors and controllers can be chosen based on experiments in the simulator before installing them on the actual robot.

## 2. Objective

At the end of this project, a URDF of the Polaris Rover EV should exist and work in the Gazebo. It is supposed to contain all the sensors of the actual test vehicle, which allows the user to conveniently test and debug new software before pushing it onto the real test vehicle. It also allows testing the rover in different conditions by changing its surroundings.

The simulation is for research purposes and streamlines the development of the autonomous driving capabilities of the Polaris vehicle. This project is a continuation of the previous research done on developing an autonomous vehicle. Thus, by the conclusion of this project there should be a meaningful contribution to the development of a simulation platform for the Polaris. The success of this project can be measured by the usefulness of the simulation platform in future research and development.

## 3. Project plan

In our project plan, we have defined the work breakdown structure as seen in Figure 2 below.

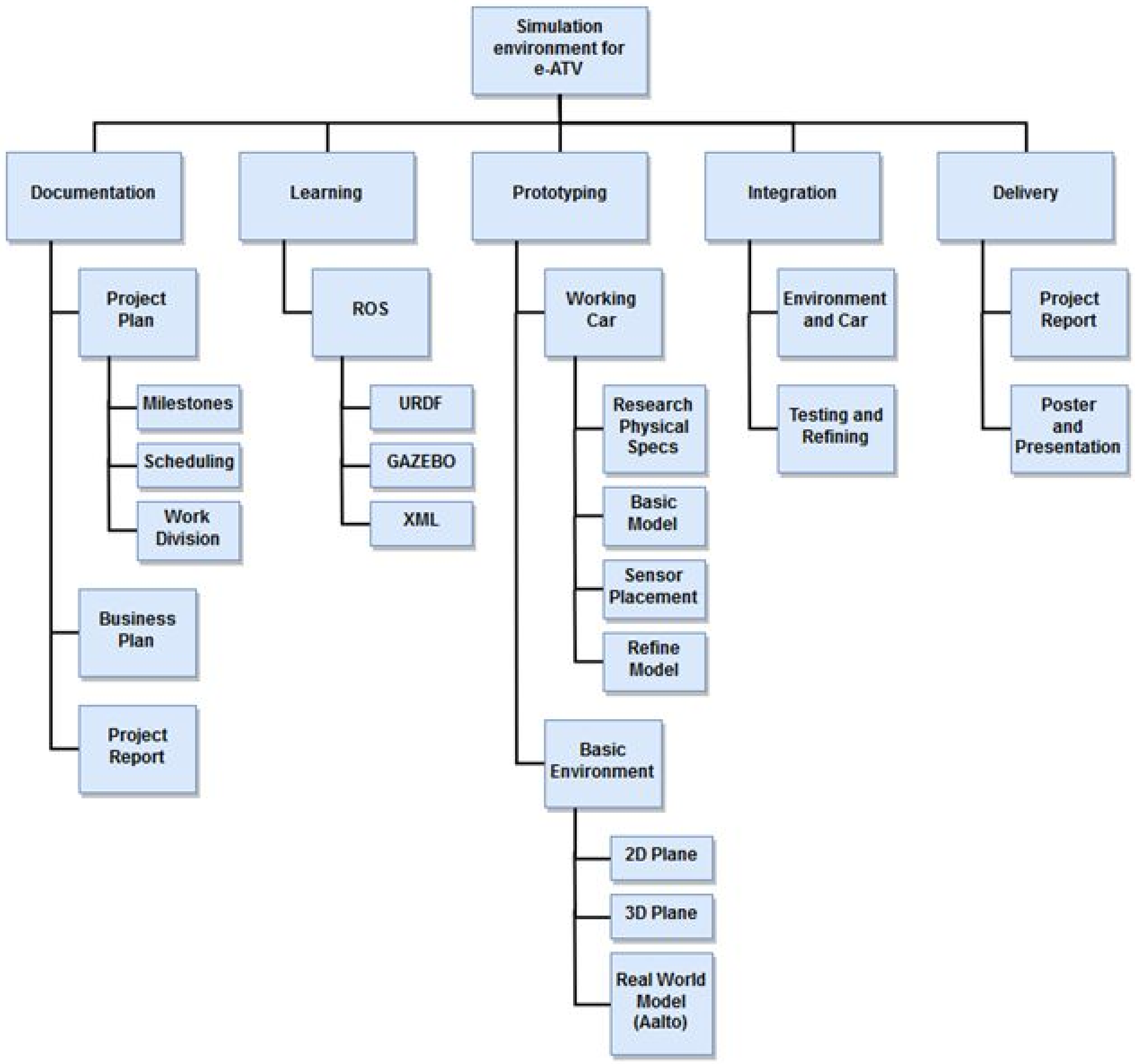
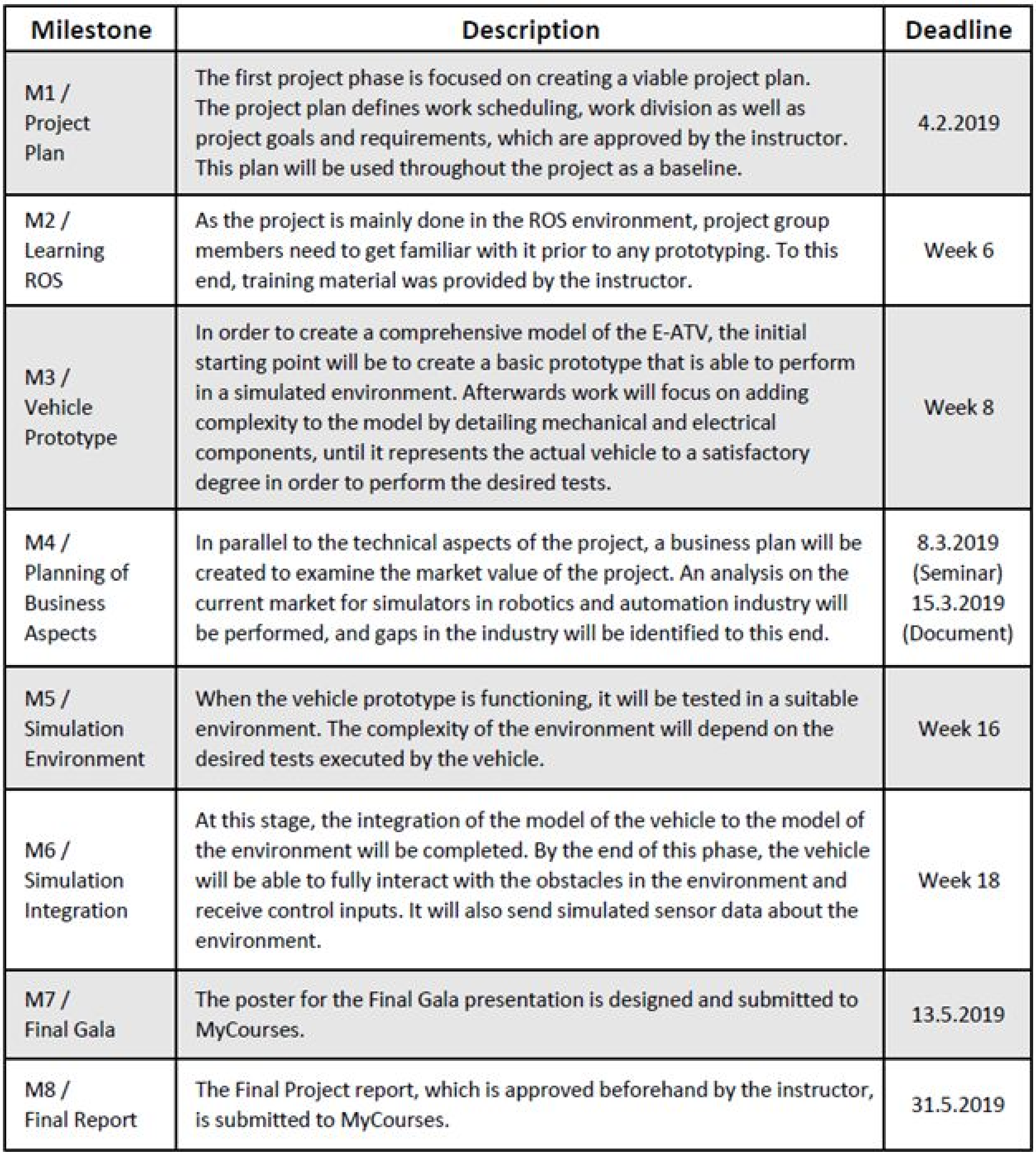


*Figure 2: Work Breakdown Structure as defined in the project plan*

The tasks defined in this structure were aimed to be completed as indicated in the milestones Table 1 below. These milestones were defined such that they would agree with the overall program of the course, and allow for reasonable time for each task they indicate to be implemented successfully.



| Milestone | Description | Deadline |
|---|---|---|
| M1 / Project Plan | The first project phase is focused on creating a viable project plan. The project plan defines work scheduling, work division as well as project goals and requirements, which are approved by the instructor. This plan will be used throughout the project as a baseline. | 4.2.2019 |
| M2 / Learning ROS | As the project is mainly done in the ROS environment, project group members need to get familiar with it prior to any prototyping. To this end, training material was provided by the instructor. | Week 6 |
| M3 / Vehicle Prototype | In order to create a comprehensive model of the E-ATV, the initial starting point will be to create a basic prototype that is able to perform in a simulated environment. Afterwards work will focus on adding complexity to the model by detailing mechanical and electrical components, until it represents the actual vehicle to a satisfactory degree in order to perform the desired tests. | Week 8 |
| M4 / Planning of Business Aspects | In parallel to the technical aspects of the project, a business plan will be created to examine the market value of the project. An analysis on the current market for simulators in robotics and automation industry will be performed, and gaps in the industry will be identified to this end. | 8.3.2019 (Seminar) 15.3.2019 (Document) |
| M5 / Simulation Environment | When the vehicle prototype is functioning, it will be tested in a suitable environment. The complexity of the environment will depend on the desired tests executed by the vehicle. | Week 16 |
| M6 / Simulation Integration | At this stage, the integration of the model of the vehicle to the model of the environment will be completed. By the end of this phase, the vehicle will be able to fully interact with the obstacles in the environment and receive control inputs. It will also send simulated sensor data about the environment. | Week 18 |
| M7 / Final Gala | The poster for the Final Gala presentation is designed and submitted to MyCourses. | 13.5.2019 |
| M8 / Final Report | The Final Project report, which is approved beforehand by the instructor, is submitted to MyCourses. | 31.5.2019 |

*Table 1: Milestones as defined in the project plan*

As seen, the overall plan begins with a planning stage, followed by a training stage for group members to get familiar with the technical aspects. In the following weeks, the technical work is carried out parallel to the business aspects of the project, and the overall project is concluded with a presentation in final gala and the submission of the final report. These tasks are executed in parallel. In other words, necessary work is carried out to meet more than one milestone at the same time. Figure 3 below illustrates this by visualizing work packages with dependencies.

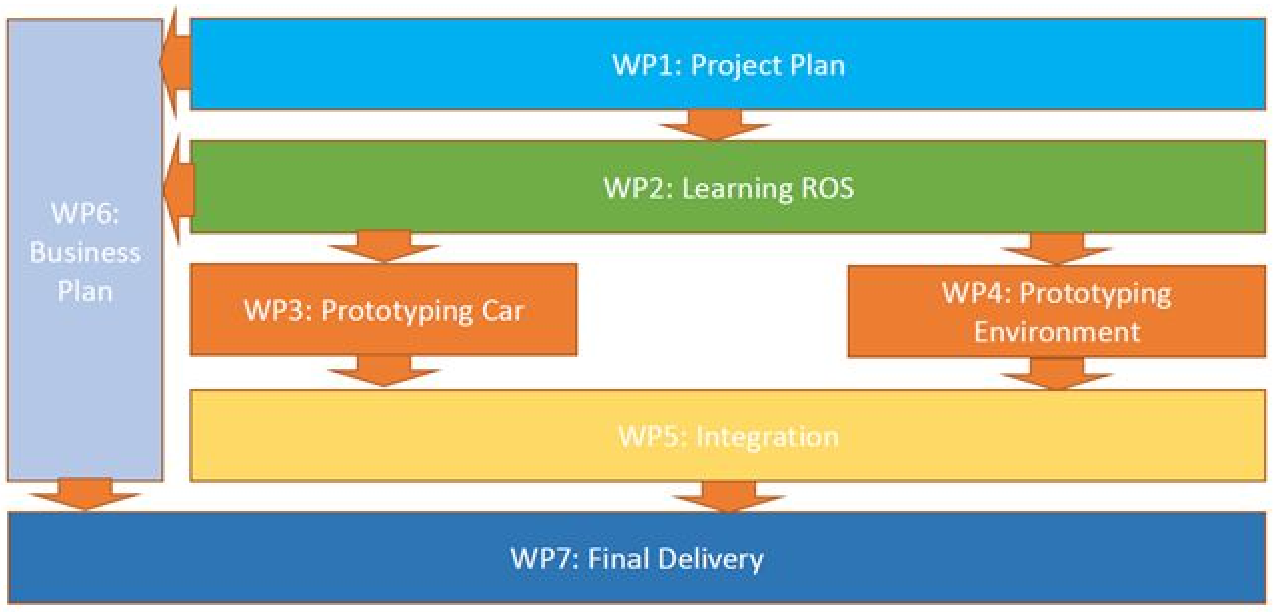


*Figure 3: Work package visualization with dependencies as defined in project plan*

# 4. Framework

This chapter describes the main software framework of the project. The main framework used was ROS (Robot Operating System). Gazebo was used as a physics simulator to simulate the behaviour of the vehicle and RViz was used to actually visualize the sensor data that is produced by the robot. As a versioning tool, the Aalto GitLab was used to keep track of software changes and allow working on different parts of the project at the same time.

## *4.1. Robot Operating System (ROS)*

The Robot Operating System (ROS) is an open source framework for robots. It was developed in 2007 at the Stanford Artificial Intelligence Laboratory [1]. Its main components are hardware abstraction and communication between different modules. There is a multitude of packages for many different robots and applications available that help the user in the testing of robots. Its main components are the so called nodes, which communicate via the subscription to topics and messages. In the ROS framework, nodes communicate with one another through topics. A node can be a subscriber to multiple topics, and it can advertise numerous topics. The topics are special messages whose definitions are placed inside *.msg files.

### 4.1.1. ROS Nodes

ROS nodes are basically single separated processes [2]. Each one of these processes is doing a separate task for the robot and a robot control system is usually consisting of multiple of these nodes. This reduces code complexity and also allows for easy extension and expansion of the robot's functionality. Nodes can communicate with each other via messages that get sent to topics. Every node needs to be started individually. They then register to the master, which manages all the nodes currently running. The master always runs in the background. In this project, we are making extensive use of ROS nodes to achieve the needed functionality. Controllers defined to be able to move the car, the means of transferring information between simulators and logging necessary information to files are all achieved by using these nodes, which are all explained in more detail in their relative chapters.

#### 4.1.2. ROS Topics

ROS topics are a way of decoupling the communication of nodes with each other [3]. A topic functions as a midpoint for the communication. Nodes can send messages to a topic, which can then be requested by other topics. For a node to send a message to a topic it needs to 'publish' to that topic, and for another node to get the information from that topic, it needs to 'subscribe' to that topic. The type of the message needs to match with the type of the topic. This is a convenient way of separating communication by type and allows different nodes to easily access the information of the robot, depending on their needs. As the primary way of communication between nodes, these topics are used in our project mostly to transfer the vehicle's odometry and joint state information, such as '/odom' topic publishing the odometry data of the vehicle as well as the sensor data.

#### 4.1.3. Launch files

Launch files are files in ROS that simplify the startup of a robot or simulation by allowing to group several nodes together and launch them with only one file, instead of launching each node individually [4]. It also allows to easily change some parameters of the simulation by setting parameters for some names and settings. In the launch file defined for our project, the necessary visualisation tools, controllers and parameters are defined and launched together to enhance usability.

### *4.2. Gazebo*

Gazebo is a free robot simulation tool [5]. It includes a detailed physics simulation and a graphical user interface that allows for easy interaction and visualization of the interactions happening in the simulation tool. The reason why it is used in this project is that there is a ROS API plugin that allows Gazebo to run as a ROS node which can interact with other nodes in the same way as described above. It allows the user to manipulate the simulation environment from the ROS framework and change certain parameters.

### *4.3. RViz*

ROS visualization (RViz) is a visualization tool designed to visualize data from the ROS framework. It is a 3D visualizer that displays a multitude of information about the robot or simulation. It also runs as a ROS node that can subscribe to topics and display the contents of these messages. This is mainly used to display sensor data, like point cloud data from a LiDAR or infrared distance measurements that are published to this topic by the sensor.

# 5. Development of the Simulation

This chapter describes the development process of the simulation in more detail. This includes the first working prototype and its refinement and the adding of sensors and actuators. For more details on the dependencies and how to run the simulation, see the appendix.

## *5.1. General Structure*

The first step in setting up the project was the general ROS software. The ROS version used was ROS Kinetic on Ubuntu 16.04 LTS. A catkin workspace was created for this project specifically. Catkin is a build system for ROS that is based on CMake. It builds all packages in ROS. The setup of the workspace was detailed in the ROS courses of the ETH Zürich [6]. A new package was created in the workspace called 'polaris_description' that includes all the files created in this project.

## *5.2. SDF to URDF*

In a Gazebo model database, an exact replication of the Polaris Ranger EV was found. The description of this model was in the SDF file format that Gazebo uses. This file format is not supported by ROS and is not usable for visualization of the robot model in RViz. ROS uses URDF files as a standard to describe the kinematics and dynamics of the robot. This includes dimensions, masses and degrees of freedom for each joint and link in an xml-style format. The file basically defines the robot and all its moving parts and their relation with each other. It uses links to define the actual parts of the robot and joints, which connect two link to each other. The type of link defines the motion the two links can do with relation to each other and what degrees of freedom there are.

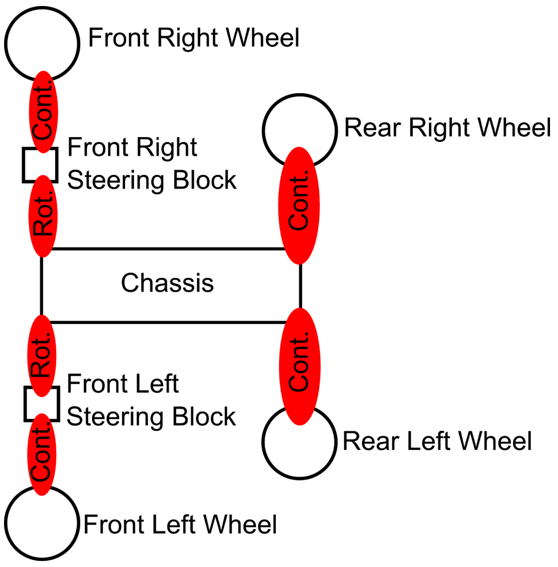


*Figure 4: General structure of the URDF. White boxes are the links and in red are the joints connecting them*

This SDF was converted to URDF using a converter [7]. The converter was not working perfectly and almost all aspects of the created URDF file had to be checked and corrected. The collision boxes were used as the visual model since the mesh supplied with the SDF was not convertible. The joints were also transferred incorrectly. Since the robot in this case is a car, the only joints necessary are continuously revolving joints for the wheels and two rotating joints for the steering. After the conversion, the URDF initially had all the joints defined as rigid, which then had to be manually redefined to be able to continuously rotate. As an example, the result of the initial conversion has the front right wheel joint, which is supposed to be a 'continuous' joint that is capable of continuous rotation defined as

```
<joint name="polaris_ranger_ev__front_right_wheel_joint" type="fixed">
```

This has been rectified by changing the joint type as follows:

```
<joint name="polaris_ranger_ev__front_right_wheel_joint" type="continuous">
```

Another problem was the rear axle, where the rear axle was a child link of rear-left wheel and the rear-right wheel the child of the rear axle. Both wheels were also defined as children of the chassis, which led to a collision, since each link can only have one parent in a URDF. Removing the rear axle and referencing both rear wheels to the chassis solved this issue. This is illustrated in Figure 5 below, where an arrow pointing from a link to another indicates that the second link is a child of the first.

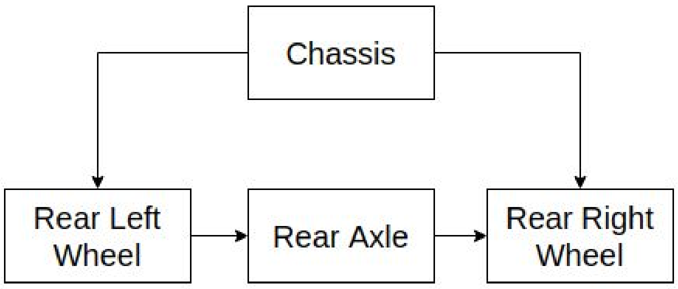


*Figure 5: First faulty link relation, in which the rear right wheel is the child of the chassis twice*

The SDF already had most of the physical values of the car defined. The dimensions and masses were cross checked using the Polaris Ranger EV manual [8]. A sketch of the general structure of the URDF can be seen in Figure 4. This shows the links of the robot, which are the four wheels, the chassis and the steering blocks and the joints that connect them.

For a first picture of the URDF in Gazebo, see Figure 6. In this picture the steering joints are still moving towards each other since they are freely rotating without any resistance and the wheels are at a slight angle, like in the real vehicle.

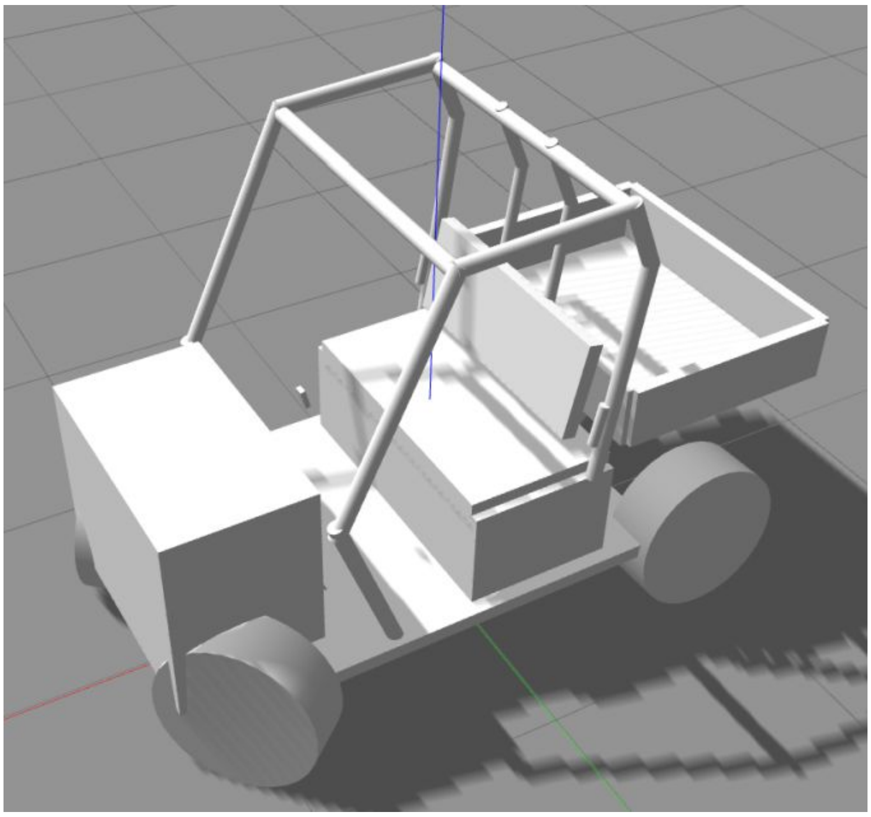


*Figure 6: First URDF spawned in Gazebo*

## *5.3. Publishing the data from Gazebo*

### 5.3.1. Odometry data

Not only it should be possible to view the model in Gazebo in its physics simulation, but the model should also be visible in RViz. RViz can only subscribe to topics, since it is a normal ROS node. Gazebo includes a built-in feature that publishes the state of the model as an odometry message, once the "get_model_state" service has been started. This functionality was used in the creation of a node that starts this service and subscribes to the model state which is published by Gazebo. This contains information about how much the robot moves in relation to the origin of the world (the point the robot spawns in). The node also acts as a publisher to convey the information it has received from Gazebo. It is now possible to visualize the position of the vehicle in relation to this odometry frame, which allows RViz to visualize the movement of the car, since it can now get this information from the /odom topic.

### 5.3.2. Joint data and actuators

In addition to the position of the robot, it should also be possible to know the current position every joint is in. The physics simulation simulates the interaction of the robot with the environment. This can be seen visually in the Gazebo, but it is not possible to get this data outside the simulation out of the box. To tell a piece of software about the current state of the robot, we need sensors. In Gazebo, controllers also act as sensors and can publish the orientation of a joint. Controllers can be defined in the URDF for each joint by defining a transmission element. The type of the element depends on the controller. For the steering joints, one position controller was defined for each steering block. This is an example of a transmission definition for the steering position controller for the left side:

```
  <joint name="polaris_ranger_ev__front_left_steering_joint" type="revolute">
          <parent link="polaris_ranger_ev__chassis"/>
          <child link="polaris_ranger_ev__front_left_wheel_steering_block"/>
          <origin rpy="1.57079  0     0" xyz="1.03  0.5   0.32"/>
          <axis xyz="0  1  0"/>
          <limit effort="1000.0" lower="-0.8727" upper="0.8727" velocity="1.0"/>
  </joint>

  <transmission name="polaris_ranger_ev__front_left_steering_trans">
          <type>transmission_interface/SimpleTransmission</type>
          <actuator name="polaris_ranger_ev__front_left_steering_motor">
          <mechanicalReduction>1</mechanicalReduction>
          </actuator>
          <joint name="polaris_ranger_ev__front_left_steering_joint">
 <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
          </joint>
 </transmission>
```

This is the definition of the joint on the top, which is connects the steering block to the chassis. The effort limit is the maximum torque that can be transferred through this joint. On the bottom, we can see the definition of the transmission element, which in this case is a position interface, so the position of the steering block joint is the thing that gets manipulated. To define a controller for this transmission, we need to define a configuration *.yaml file in this form:

```
type: position_controllers/JointPositionController
joint: polaris_ranger_ev__front_left_steering_joint
```

We define the type of the controller and the joint it acts on. For the PID values, a separate *.yaml file was created, that includes all the PID values the controller

```
gazebo_ros_control/pid_gains:
  polaris_ranger_ev__front_left_steering_joint: {p: 1000, i: 100, d: 10}
```

The two front wheels are connected to the steering blocks via position controllers. The rear wheels drive the car and are connected via velocity controllers. The transmissions now publish their actual joint state and we can read those positions using RViz, allowing the model to be visualized as it is in the simulated world. The front wheels spin freely and the joints cannot produce any torque since their effort value in the URDF is set to 0. They publish their orientation, so it is possible to get the orientation of the wheels from outside the simulation. This allows it to be displayed in RViz. To get 4-wheel-drive, it is possible to add velocity controllers to those joints as well, using the same controllers as for the rear wheels.

Each controller is launched as a separate node. The PID controllers were tuned during runtime using rqt_reconfigure, which can be launched using:

```
$ rosrun rqt_reconfigure rqt_reconfigure
```

and allows tuning of the parameters while the simulation is running. The P- I- and D- values for the controllers were tuned in a way to minimize oscillations and to react reasonably fast to changes. This was quantified by publishing a value to the controller and estimating the time it takes the vehicle to reach the desired velocity for different PID values. Figure 7 shows the behaviour of the vehicle for different values of the PID when a desired velocity was published to the wheels. There were several different configurations for the PID controllers, which can be seen in table 2.

| **Config no.** | **Controller** | **P** | **I** | **D** |
|---|---|---|---|---|
| **1** | Steering | 100 | 0.1 | 10 |
| | Velocity | 100 | 0.1 | 0.1 |
| **2** | Steering | 10 | 0.1 | 10 |
| | Velocity | 10 | 0.1 | 0.1 |
| **3** | Steering | 1000 | 0.1 | 10 |
| | Velocity | 1000 | 0.1 | 0.1 |
| **4** | Steering | 1000 | 1 | 10 |
| | Velocity | 1000 | 1 | 0.1 |
| **5** | Steering | 1000 | 10 | 10 |
| | Velocity | 1000 | 10 | 0.1 |
| **6** | Steering | 1000 | 100 | 10 |
| | Velocity | 1000 | 100 | 0.1 |

*Table 2: Different PID configurations to test the controllers in the simulation*

The D-values were mostly kept constant since those turned out to be good as is and especially for the velocity controllers high D-values were causing issues with very unstable behaviour, even when no velocity message was published. For each configuration a simulation run was recorded and the velocity and position plotted. Those plots can be seen in figure 7 for the velocity and in figure 8 for the position of the vehicle.

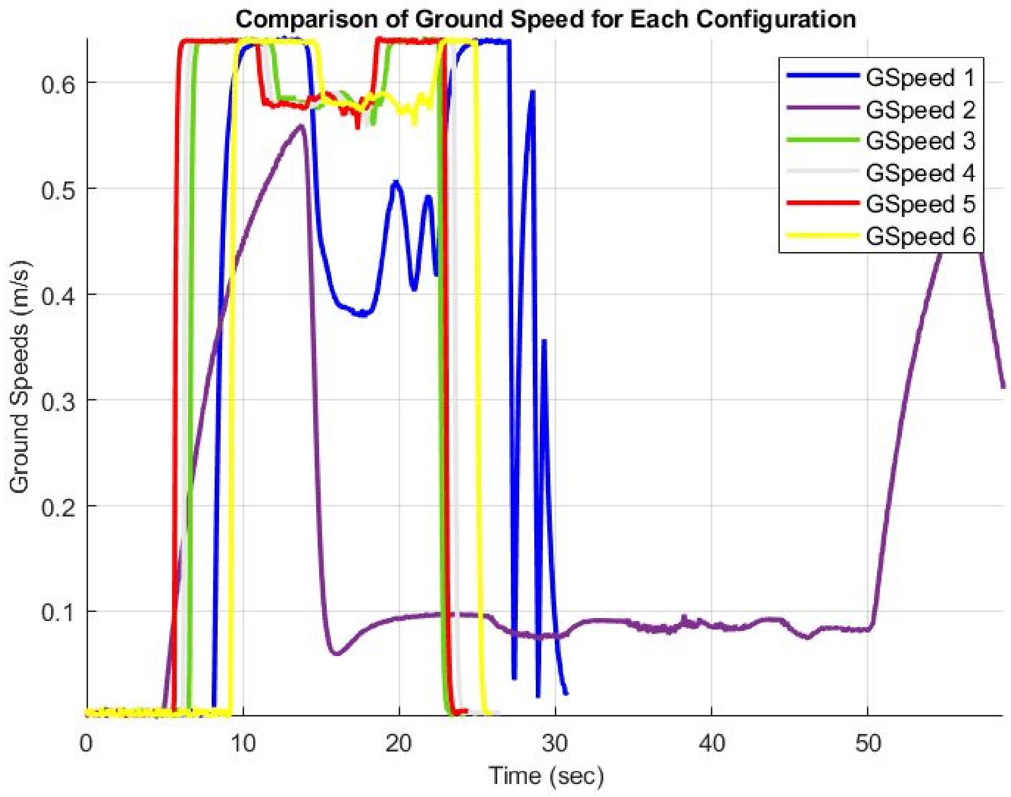


*Figure 7: Different measured PID configurations for the PID controllers.*

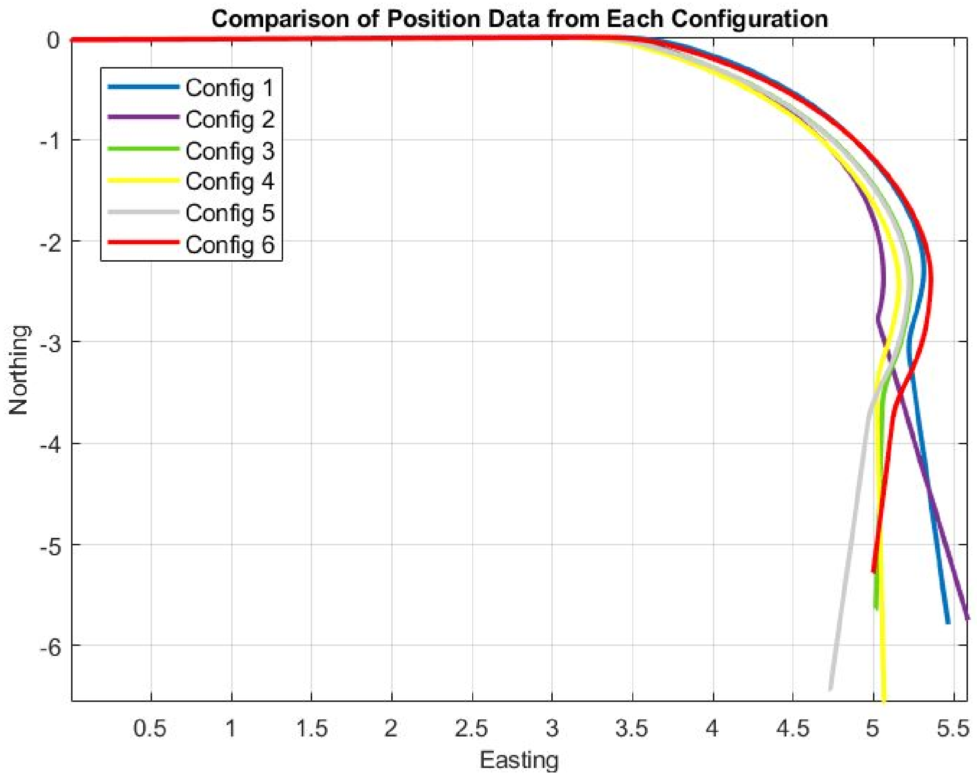


*Figure 8: Eastern and northward position for the PID tests. The vehicle was first driven in a straight line and then put into a turn with constant wheel angle of 0.7 radians*

To create these simulation runs, the simulation was started and a constant angular velocity of $\omega = 2$ radians per second was published to the velocity controllers without steering input. This should result in a ground speed of $v = 0.64$ m/s with a wheel radius of $r = 0.32$ m due to the relation $v = r\omega$.

Then at approximately the same point for each simulation, a steering input of 0.7 radians was published to both front wheel steering blocks. An approximate 90° turn was driven before the car was straightened out. Since, all of this was manual input, there were slight deviations between the runs and the timings. From the plots, we can see that configurations 4 to 6 are very similar in behaviour. They follow the reference reasonably well and don't do any abrupt movements or become unstable. The proportional gain seems to have the biggest impact on the behaviour on the vehicle. For now, the PID gains were set to follow configuration 5, which appears to be the smoothest among the six configurations.

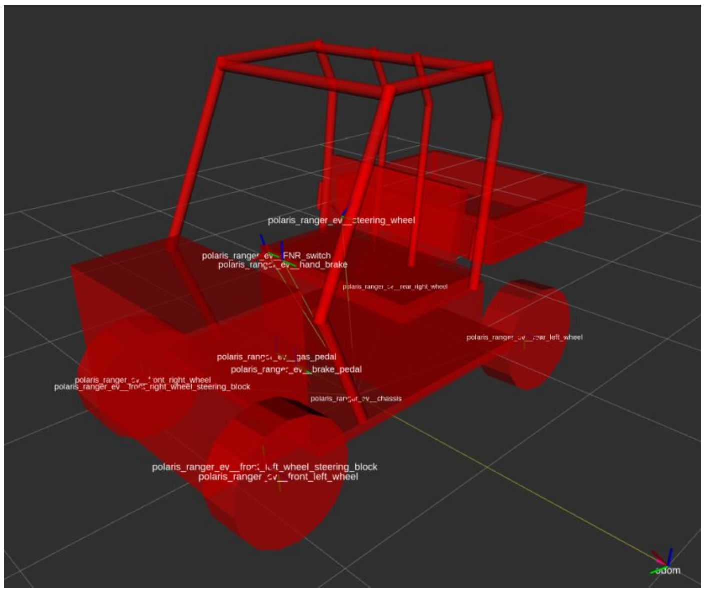

polaris_ranger_ev__steering_wheel
polaris_ranger_ev__FNR_switch
polaris_ranger_ev__hand_brake
polaris_ranger_ev__rear_right_wheel
polaris_ranger_ev__rear_left_wheel
polaris_ranger_ev__gas_pedal
polaris_ranger_ev__brake_pedal
polaris_ranger_ev__front_right_wheel
polaris_ranger_ev__front_right_wheel_steering_block
polaris_ranger_ev__chassis
polaris_ranger_ev__front_left_wheel_steering_block
polaris_ranger_ev__front_left_wheel
odom

*Figure 9: URDF as seen in RViz*

To allow for easier testing of the controllers, a publisher was written that takes input from the keyboard and publishes this data to the rear wheel velocity controllers and the front wheel steering position controllers. This can be launched in a separate terminal using

```
$ roslaunch polaris_description teleop.launch
```

The position controllers take values in radians, which allows for setting of the steering angle of each wheel separately. This has the advantage, that things like Ackermann steering can easily be implemented in software by modifying the publisher. It is possible to only specify one steering radius and let the publisher calculate the values for each steering angle. Figure 10 shows the problem. The inner and outer wheel go around different radii. If both wheels are at the same angle, there will be slippage and efficiency loss during the corner. To account for that, the Ackermann geometry angles both wheels at different angles to ensure efficient cornering.

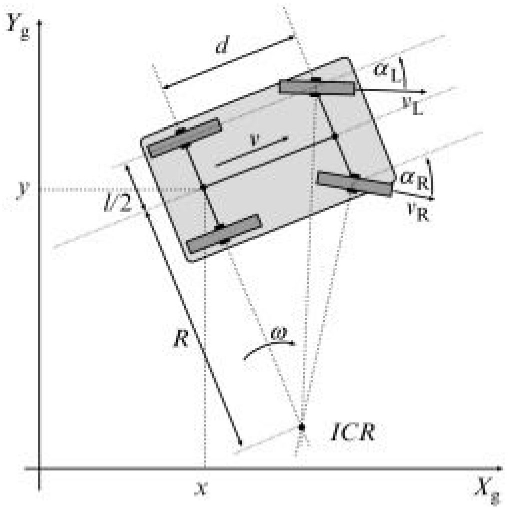


*Figure 10: Schematic for Ackermann steering [9]*

A steering node could now receive a desired turn radius *R* (see figure 10). From that it can calculate the steering angle for the right and the left wheel from [9]:

$$\alpha_L = \frac{\pi}{2} - arctan\frac{R+l/2}{d}$$
$$\alpha_R = \frac{\pi}{2} - arctan\frac{R-l/2}{d}$$

The same is true for the rear wheels. They don't have a differential in the rear axle, but both wheels have separate controllers, which allow for modification of the velocities depending on the steering angle.

The vehicle can now be driven around in Gazebo and the position and joint state can be seen in RViz. The view in RViz can be seen in Figure 9. The finished URDF can be found in appendix 4.

### 5.3.3. Sensors

To enable the vehicle to sense not only its joint states but also its environment. Two sensors were implemented to the URDF. Those are defined as Gazebo plugins and as such don't need to be launched separately. Their position is defined in the URDF and as soon as the URDF is launched in Gazebo, those sensors also get launched. The inertial measurement unit (IMU) publishes acceleration data to the /imu topic and the LiDAR publishes its point cloud data to a topic called /velodyne2. Environment and interactions of robot with the environment. Both sensors are referenced to the chassis to define their position (see also figure 12). This now allows for the vehicle to sense its environment. In addition, an additional GPS (in the form of a *hector* Gazebo plugin) was implemented in the urdf, which also publishes its data. However, this plugin was not sufficiently tested, but it publishes its data to the relevant topics and can be refined further in future. It is now possible to listen to the sensor data of the vehicle during runtime of the simulation using

```
$ rostopic list
$ rostopic echo /topicname
```

An example of each sensor message can be found in the appendix. To make data evaluation easier, an additional subscriber was written that logs the IMU data to a file. There is also one for the point cloud data of the LiDAR, due to the huge number of points generated each timestep, this is not really feasible.

A graph, created with rqt_graph, shows the topics and nodes that are active during the simulation. It can be seen in Figure 11.

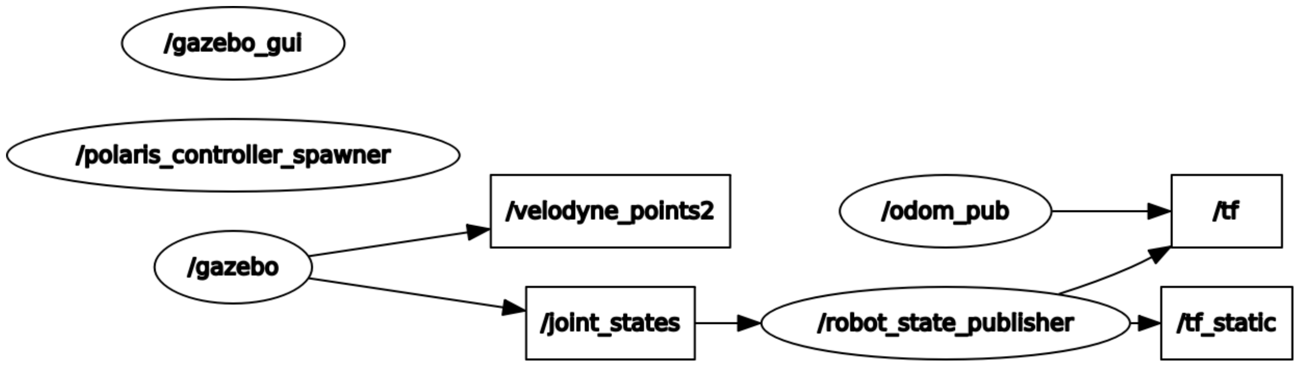


*Figure 11: rqt_graph, showing the nodes (circles) and the topics (rectangles) they subscribe and publish to. RViz is not shown in this graph.*

In addition to that, appendix 2 shows the whole TF-tree using rqt_tf_tree, showing all references of links with each other. Figure 12 shows the final dimensions as used in the URDF for sensor position and position of links.

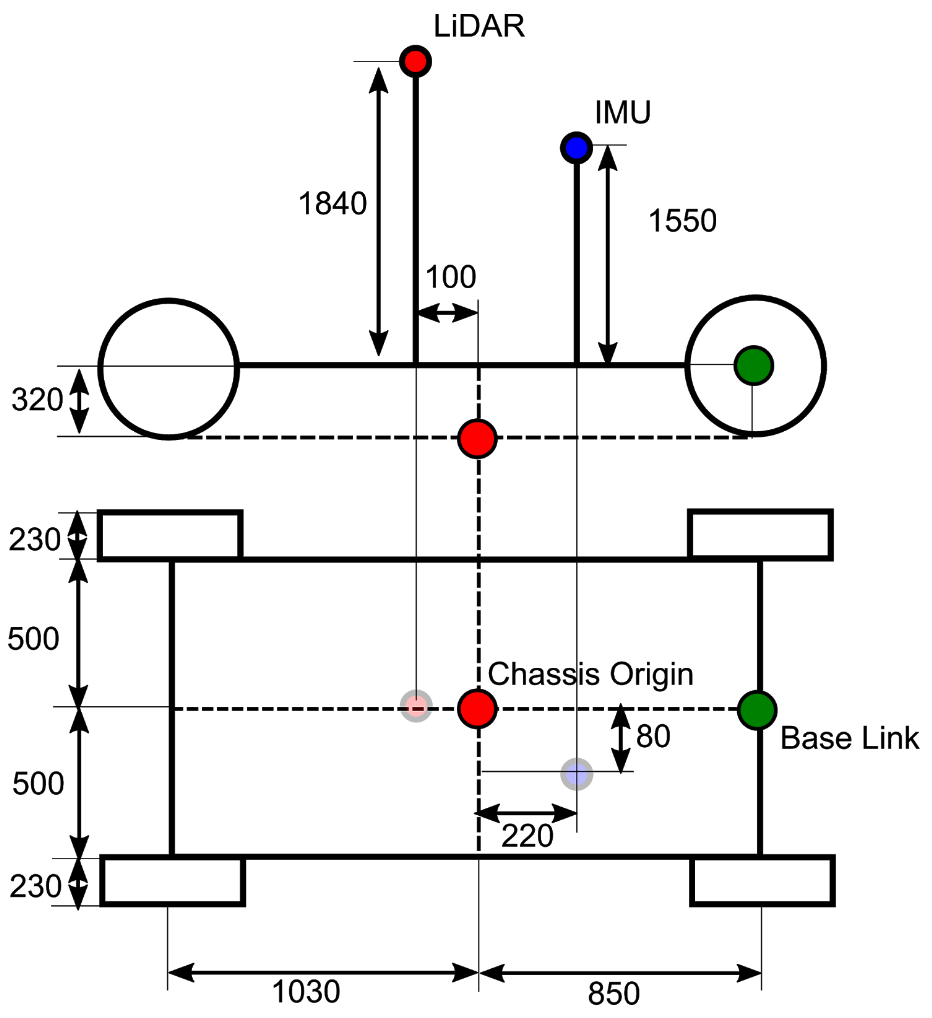


*Figure 12: Dimensions of the Polaris Ranger (in mm) as used in the URDF (not to scale). The Origin is referenced to the base-link, which sits in the center of the rear axle*

## *5.4. Environment and its interactions with the robot*

The environment was modeled with *Wings 3D* software, after a google maps bird's-eye view image (Fig. 10). The 3D model was then UV-mapped onto a 2D plane. The outlines of every edge were drawn into a texture which were directly exported from the 3D modeling program. UV-mapping is the process of dividing a model into projections of its surface, so that a texture, amap or other 2D object can be applied to it. UV stands for the two new coordinates u and v, and the projection is from p(x,y,z) → p'(u,v). Everything was textured on top of the outline of the environment model with GIMP by using public domain images as a base. The finished environment mesh was then exported as a COLLADA file (which is identified by *.dae) with Blender. The textures and the mesh were placed in the Gazebo models folder, from where the model can be spawned into any new or old world file (Fig. 11).

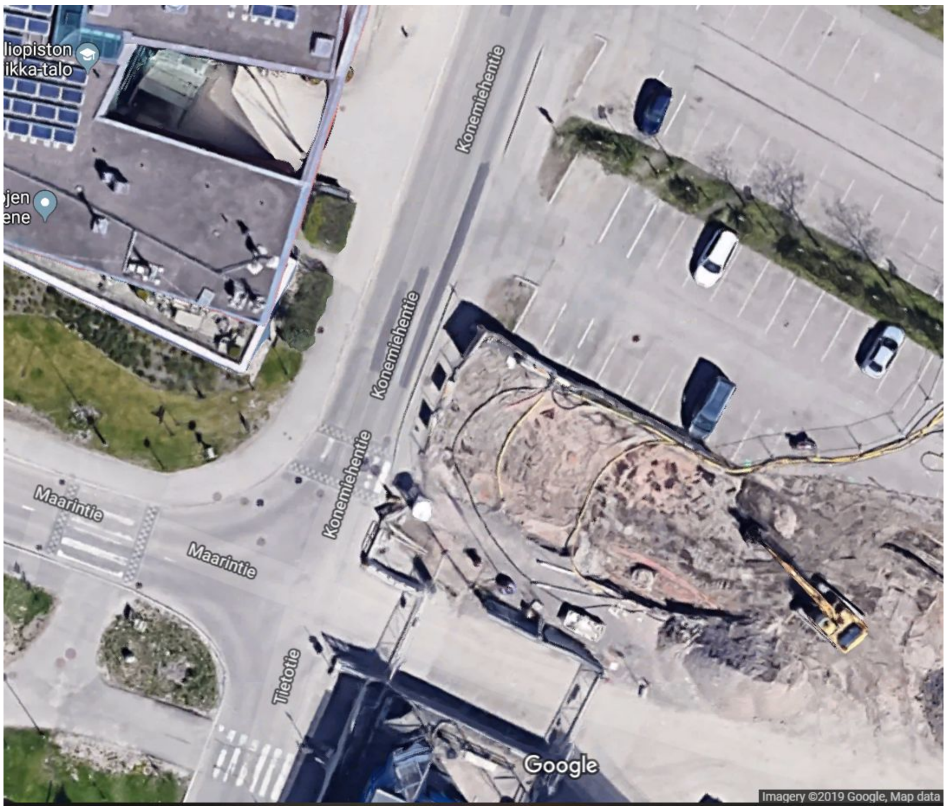


*Figure 13. Recently updated Google Maps view of the Tietotekniikka-talo parking lot.*

The environment comes populated with basic obstacles like the two ramps and the street cones. Additional props can be added directly with the Gazebo program, where the resulting environment can be exported into a world file to be used later. These props exist to test the point cloud generation of the LiDAR in RViz, as well as the motion capabilities of the robot.

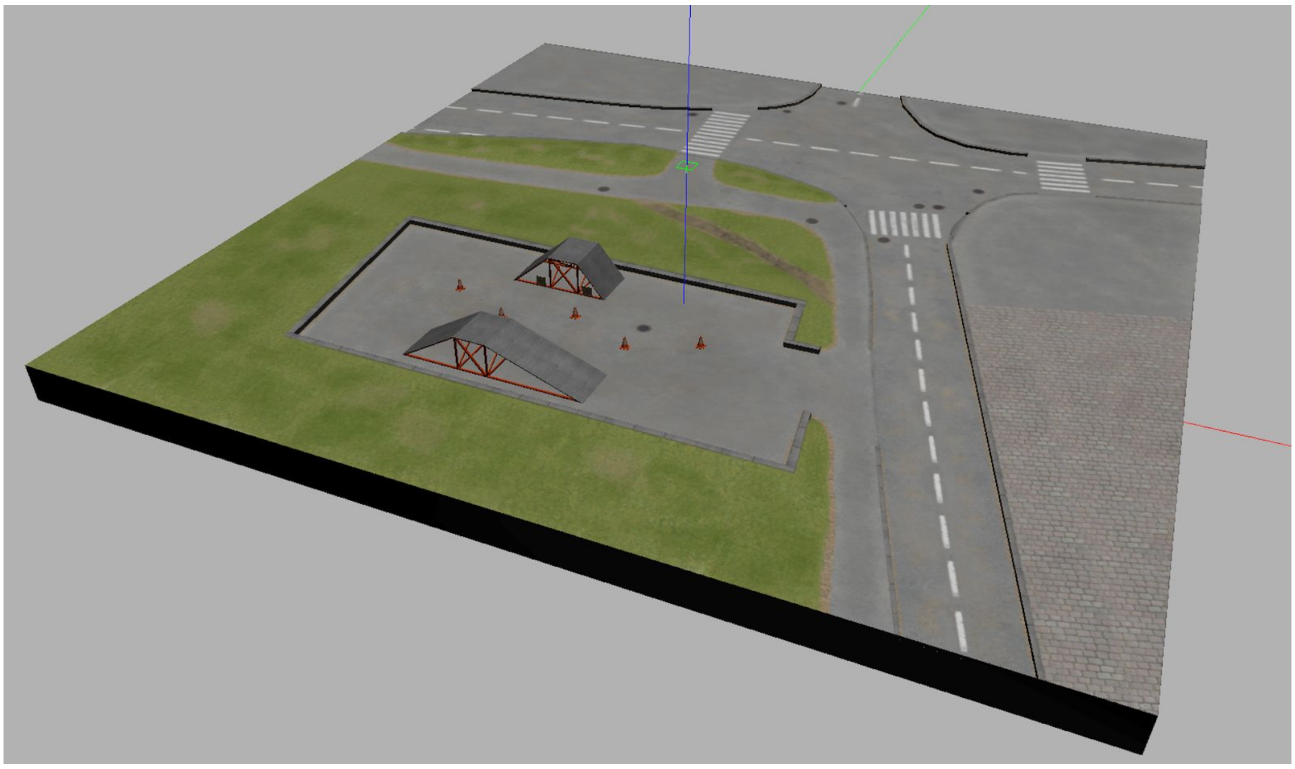


*Figure 14. The 3D environment as seen in Gazebo.*

The robot can now interact with the environment via its sensors. The physical interaction is simulated by defining the masses and inertial properties in the URDF. Collisions get resolved with the collision boxes, defined in the URDF, which are very close to the real dimensions of the car. For simplicity's sake, the geometry was slightly simplified though, using only cuboids and cylinders as basic geometric shapes. The contact patches to the ground are the tires. To accurately simulate their interaction with the ground, friction was defined in the URDF. This is a Gazebo property since it is necessary for the physics simulation. The default value for all links is a friction constant of 1. For the wheels, this was changed to 0.7, since this is the approximate value [10] for the contact between rubber tires and an asphalt road.

# 6. Results of the Project

The final result of the project is working as intended. We reached all the set goals for this project and made a reliable simulation platform for the rover. The simulation platform can be extended and possibly used in a further research. Screenshots of the final simulation in working order can be seen in Figure 15.

## *6.1. Simulation Environment*

The physics based environment is working as intended and visualized in Gazebo. The vehicle is modeled with its physical properties and kinematics. Single parameters like friction, masses or inertia can be easily modified in the URDF file. The interaction with the environment is simulated with sufficient detail.

The environment was based on real life satellite data and is used to test the simulation. It includes different elevations and obstacles to test the sensors and the behaviour of the model along with some ramps for friction testing.

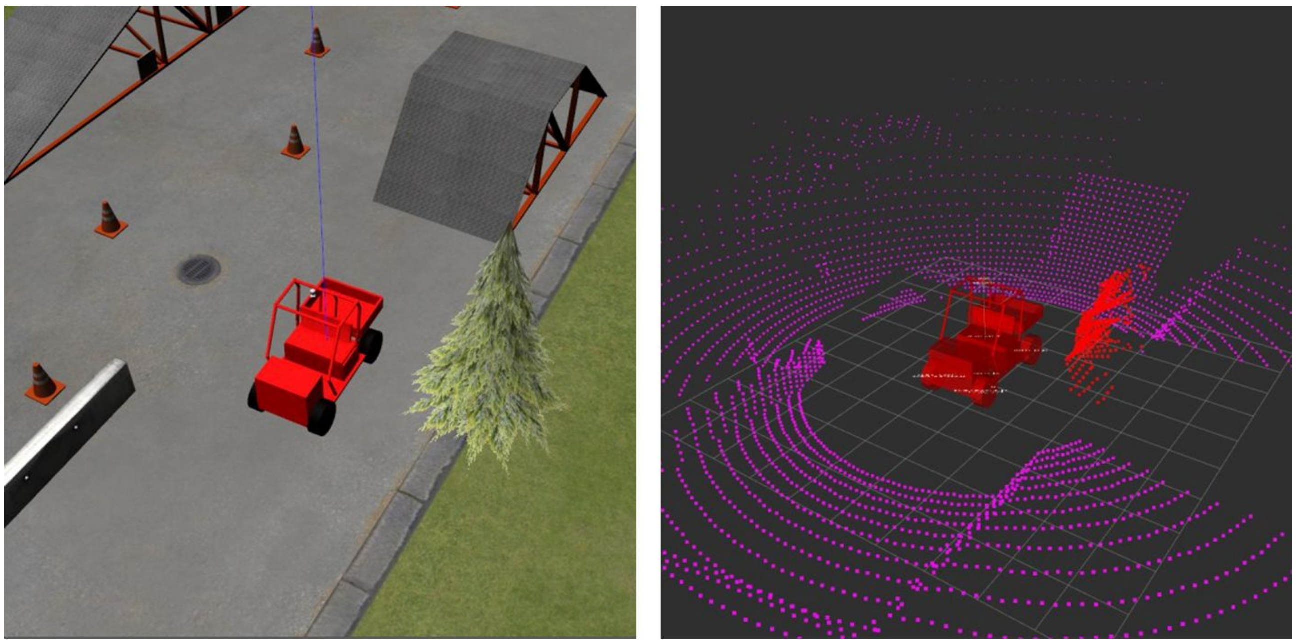


*Figure 15. Gazebo (left) and RViz (right) view of the simulation*

## *6.2. Sensor Data*

Currently LiDAR, GPS and IMU sensors are included in the final result. LiDAR data is visualised in RViz view to easily follow what the rover sees. This point cloud data can be used to build maps for the rover to follow, when automatic driving is implemented. The IMU gives acceleration data for the vehicle to detect movement and orientation. For details about the GPS data, see appendix 3.

The finished simulation can be now used to test SLAM procedures or autonomous driving routines like path planning and obstacle avoidance. It will save time in the development phase.

## *6.3. Limitations of the Simulation and possible next steps*

The software works and is simulating the vehicle to a detailed level. Current limitations are the way the controllers are implemented. A differential for the rear axle is not defined in the URDF and currently only one value for angular velocity is published to both wheels, so during curves one wheel is always slightly slipping. This can be fixed by adapting the velocity publisher to account for that, since both wheels can be commanded different velocities since they have separate controllers. The same is true for the steering. Currently the same angle gets published to both wheels. Since both steering blocks have separate controllers, it is no problem to implement Ackermann steering on the software (in curves, both wheels have different turn radii). However, this was outside the scope of the project work.

Moreover, even though the real vehicle has 4-wheel drive, the simulation currently drives the rear wheels. The front wheels already have controllers implemented though, they simply need to be changed for velocity controllers equivalent to the rear wheels. The YAML files need to be adapted accordingly.

Friction is defined with a constant for each contact point in Gazebo, so for each wheel a friction value of 0.7 is set as constant for the simulation. Since the friction coefficient depends on both surfaces, the tire material and the surface the car rolls on, it might change in reality when the surface gets wet or changes in other ways. In the simulation, the friction value is constant set as 0.7 for contact between asphalt and tires. Dynamic feedback and allocation might be possible, in the timeframe of the project this was not the main focus.

# 7. Reflection of the Project

## *7.1. Reaching objective*

The expected output of the project was to create a simulation environment in which the Polaris Ranger is implemented to a sufficiently detailed level with all the necessary sensors, physical and visual properties and functionality included.

By the end of the project, we have fully reached the expected objectives and provided the simulation environment to the satisfaction of the instructor.

## *7.2. Timetable*

The timetable of the project was realized quite well. We have met every milestone deadline defined in the project plan, and was able to complete both technical and business aspects of the project without any delays.

As we got more technically competent during the course of the project, we have realized that we have overestimated the workload of certain packages and underestimated others. As an example, we realized quite early on that ROS already provided means of integration of the environment and the vehicle, thus, we shifted the workload of responsible people to other work packages. However, the total workload estimation and planning were on point, so there were no complications in staying true to the overall project plan.

## *7.3. Risk analysis*

In Table 3 below, the previously foreseen risks, their occurrence and corresponding reflection can be found.

Other than the mentioned risks in the project plan, our project team has not run into any significant risks throughout the project work.

### *7.4. Project Meetings*

We had regular project meetings weekly amongst group members. In those meetings, we decided on the course of action for the following week and the duties of each member. Between meetings, a Telegram channel was used for internal communication, whenever needed.

Every project meeting had an agenda, and a desired person to write the memo and share it with the team members later on. We have learned that sticking to the predefined agenda and keeping the meeting as brief and clear as possible was the best course of action.

In addition to the project meetings, the project manager has regularly reported to the instructor. Due to a temporary absence of the first instructor, communication was via two different channels for a portion of the project, e-mail with Tabish Badar and face to  face meetings with demonstrations and discussions with Mika Vainio.

| *Foreseen Risk* | *Foreseen Likelihood* | *Occurrence* | *Reflection* |
|---|---|---|---|
| Loss or corruption of data | Moderate | None | As we have used versioning software and frequent backups, we did not run into the risk of loss or corrupted data. |
| Dropout of a project member | Low | None | The original 4 members are still included in the project. |
| Model is not good enough to simulate a real environment | Low | None | Physical properties of the desired environment are included in the model. |
| Sickness or impairment of project member | Moderate | Moderate | There were times where certain team members were unavailable due to sickness or planned trips. This has no adverse effect on the project, as their workload was shifted successfully to other available time slots. |
| Miscommunication between project group and instructor | Low | None | With regular reporting to the instructor, we made sure that there was no miscommunication. |

| | | | |
|---|---|---|---|
| Miscommunication among the group members | Low | None | With regular meetings and a friendly atmosphere, it was ensured that every group member had a say in the matters, and internal communication was kept brief but clear. |
| Insufficient Planning | Low | None | As the project plan was carefully laid out and followed, we did not run into any misplanning or delay risks. |
| Scope of the project bigger than anticipated | Low | None | The scope of the project was made clear with the instructor before the commencement of the technical aspects. |

*Table 3. Foreseen risks and their occurrence*

### *7.5. Quality management*

We have followed the defined quality plan, and did not run into any quality problems throughout the implementation of the project, including the demonstration and integration phase. This was achieved thanks to upholding a clear and active communication between the project group and the instructor and making sure the work was going on in the desired quality level. In case other group members observed a chance to improve the quality of other members' works, this was immediately communicated and acted upon, which ensured a high quality output.

# 8. Discussion and Conclusions

In addition to the points discussed in Chapter 7, completing this project successfully has been very beneficial to the group members in many additional ways. The importance of having a clear project plan and following it as closely as possible has been obvious in all stages of the work. However, we have also learned adaptability when needed was one of the key points to achieve success in a project. We have learned that doing a project for the sake of itself, no matter how interesting, is never enough and business aspects should always be considered. In addition, it was helpful to be part of a project from the very beginning of it until its completion, and observe and solve many kinds of problems that arise during its implementation.

In conclusion, the project has been completed successfully as a result of hard work, careful planning, good teamwork and supportive instruction. It had begun as an idea to assist in further research and has developed to an end-product that has successfully fulfilled is desired objective.

## List of Appendices

- Dependencies and Instructions (see below)
- TF tree (see below)
- Project Plan (separate file)
- Poster (separate file)
- URDF code
- Topic messages for sensors and odometry

## References

[1] About ROS. [Online] Available at: http://www.ros.org/about-ros [Retrieved 27.5.2019]

[2] ROS Nodes. [Online] Available at: http://wiki.ros.org/Nodes [Retrieved 27.5.2019]

[3] ROS Topics. [Online] Available at: http://wiki.ros.org/Topics [Retrieved 27.5.2019]

[4] Launch Files. [Online] Available at: http://wiki.ros.org/roslaunch [Retrieved 27.5.2019]

[5] Gazebo. [Online] Available at: http://wiki.ros.org/gazebo [Retrieved 27.5.2019]

[6] How to setup developer PC for ROS. [Online] Available at: https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/ROS2019/how_to_setup_developer_pc.pdf [Retrieved 27.5.2019]

[7] SDF to URDF converter. [Online] Available at: http://answers.gazebosim.org/question/1310/sdf-to-urdf-converter/ [Retrieved 27.5.2019]

[8] Polaris Ranger 2012 EV Manual. [Online] Available at: https://ranger.polaris.com/en-us/owners-manuals/ [Retrieved 27.5.2019]

[9] Ackermann - an overview https://www.sciencedirect.com/topics/engineering/ackermann [Retrieved 30.5.2019]

[10] Friction and Friction Coefficients https://www.engineeringtoolbox.com/friction-coefficients-d_778.html [Retrieved 30.5.2019]

# Appendix

## *1. Dependencies and Instructions*

The package was developed in ROS Kinetic on Ubuntu 16.04 LTS. We used catkin tools, which depend on the ROS depositories. To set up the workspace, the document from the ETH Zürich course on ROS was used as the basis:

https://www.ethz.ch/content/dam/ethz/special-interest/mavt/robotics-n-intelligent-systems/rsl-dam/ROS2019/how_to_setup_developer_pc.pdf

Next step is to clone the polaris_description package into the /workspace/src folder. Then we need to build the package with

```
$ catkin build
```

while in the workspace. Before we can first launch the simulation, several steps need to be taken first. We need to install all the controller packages:

```
$ sudo apt-get install ros-kinetic-gazebo-ros-control
$ sudo apt-get install ros-kinetic-position-controllers
$ sudo apt-get install ros-kinetic-effort-controllers
```

Then we need to make the python scripts for some of the nodes executable:

```
$ chmod +x src/polaris_description/src/write_to_file.py
$ chmod +x src/polaris_description/src/controller.py
```

in the workspace.

We also need the velodyne simulator, which is the plugin used to simulate the LiDAR in Gazebo:

```
$ sudo apt-get install ros-kinetic-velodyne-simulator
```

And we need the hector Gazebo plugin to run the GPS sensor:

```
$ sudo apt-get install ros-kinetic-hector-gazebo-plugins
```

Then we need to move the /workspace/src/polaris_description/environment folder to /.gazebo/models for Gazebo to be able to find the environment.

Now we just need to source the setup file from the workspace folder (we need to do this every time before we launch the simulation):

```
	$ source devel/setup.bash
```

Now we can run the simulation by running:

```
	$ roslaunch polaris_description sensors.urdf.xacro
```

If Gazebo constantly crashes during the startup, switching from open source to proprietary drivers in Ubuntu seems to solve the problem. In Ubuntu 16.04 this is in the 'Software & Updates' window in the 'Additional Drivers' tab. Changing from open source drivers to proprietary drivers should fix the problem.

With the simulation running, we now see the vehicle in RViz and Gazebo. To drive around, we need to launch the keyboard controller in an extra terminal window using:

```
	$ roslaunch polaris_description teleop.launch
```

Before doing this, we need to source the setup file in this terminal as well, see above. This allows to publish information to the controllers using the keyboard. For manual publishing of values, use

```
	$ rostopic list
```

To see the list of available topics.

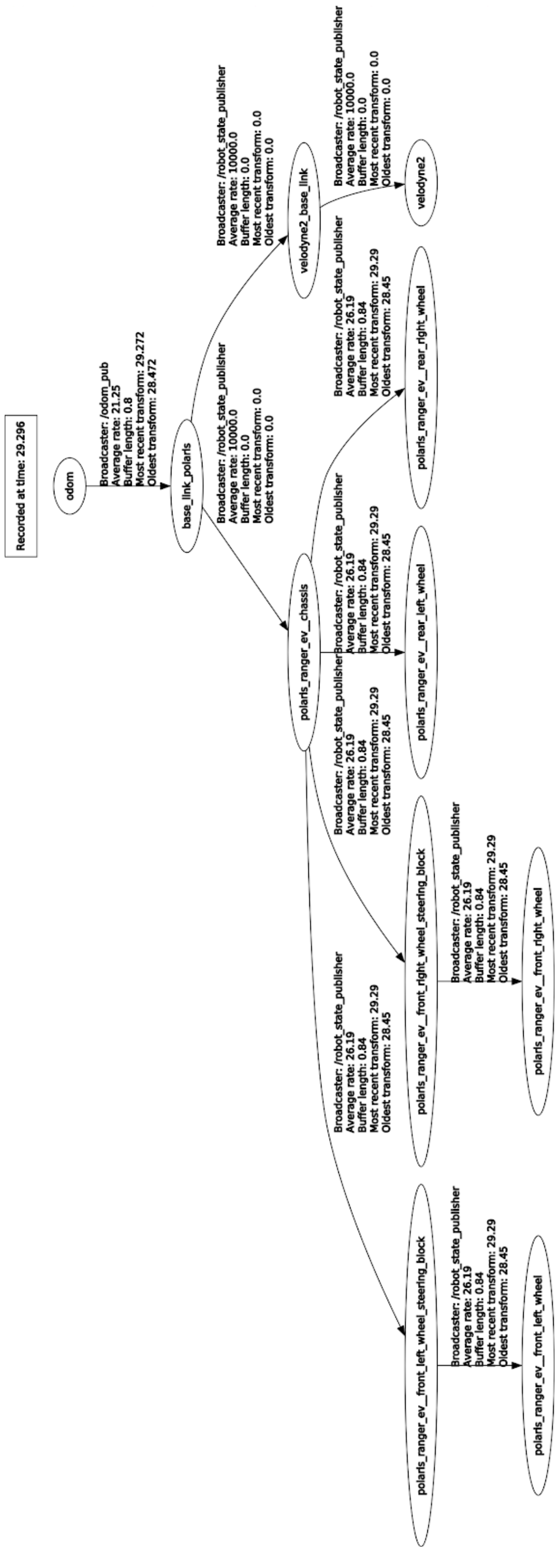
```
	$ rostopic pub /topicname
```

to publish values to a topic. Multiple presses of tab after the topic name fill in the correct form and only the numbers need to be changed. For example to publish an angle to the steering controller:

```
$ rostopic pub /polaris_steering_controller/command std_msgs/Float64MultiArray
	"layout:
	dim:
		- label: ''
		size: 2
		stride: 1
		data_offset: 0
		data: [0.7, 0.7]"
```

Where 0.7 radians is sent as an angle to both wheels. Of course it is also possible to send different angles to each wheel. The velocity controller works in the same way and shows up in the topic list (/velocity_controllers). The controller gets the angular velocity of the wheels for each wheel. So the actual velocity of the vehicle is the angular velocity times the radius of the wheels, which in this case is 32cm.

## 2. TF Tree



Recorded at time: 29.296
odom
Broadcaster: /odom_pub
Average rate: 21.25
Buffer length: 0.8
Most recent transform: 29.272
Oldest transform: 28.472
base_link_polaris
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
velodyne2_base_link
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
velodyne2
Broadcaster: /robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
polaris_ranger_ev__chassis
Broadcaster: /robot_state_publisher
Average rate: 26.19
Buffer length: 0.84
Most recent transform: 29.29
Oldest transform: 28.45
polaris_ranger_ev__rear_right_wheel
Broadcaster: /robot_state_publisher
Average rate: 26.19
Buffer length: 0.84
Most recent transform: 29.29
Oldest transform: 28.45
polaris_ranger_ev__rear_left_wheel
Broadcaster: /robot_state_publisher
Average rate: 26.19
Buffer length: 0.84
Most recent transform: 29.29
Oldest transform: 28.45
polaris_ranger_ev__front_right_wheel_steering_block
Broadcaster: /robot_state_publisher
Average rate: 26.19
Buffer length: 0.84
Most recent transform: 29.29
Oldest transform: 28.45
polaris_ranger_ev__front_right_wheel
Broadcaster: /robot_state_publisher
Average rate: 26.19
Buffer length: 0.84
Most recent transform: 29.29
Oldest transform: 28.45
polaris_ranger_ev__front_left_wheel_steering_block
Broadcaster: /robot_state_publisher
Average rate: 26.19
Buffer length: 0.84
Most recent transform: 29.29
Oldest transform: 28.45
polaris_ranger_ev__front_left_wheel

## 3. Topic messages for sensors and odometry

The  sensor message of the  IMU in /imu looks like this:

```
header:
  seq: 5
  stamp:
      secs: 87
      nsecs:  33000000
  frame_id: "imu_link"
orientation:
  x: -8.36898675613e-07
  y: 3.28385333544e-05
  z: 0.022937264608
  w: 0.999736905797
orientation_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
angular_velocity:
  x: -0.00357463032606
  y: -0.00132037139929
  z: 0.000319497282385
angular_velocity_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
linear_acceleration:
  x: -0.31303599832
  y: 0.133812598897
  z: 9.87149488962
linear_acceleration_covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

So we get the orientation of the vehicle, the angular velocity and the linear acceleration.  An example of the /odom topic looks like this:

```
header:
  seq: 168
  stamp:
      secs: 224
      nsecs: 989000000
  frame_id: "odom"
child_frame_id: "polaris_ranger_ev__chassis"
pose:
  pose:
      position:
      x: 0.0902822818141
      y: -0.195067939456
      z: 0.411199408071
      orientation:
```

```
      x: 2.80294852154e-06
      y: 3.11264879298e-05
      z: 0.060590172
      w: 0.998162727255
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
twist:
  twist:
      linear:
      x: 0.000978881359273
      y: -0.00196984112264
      z: -0.00393515569602
      angular:
      x: -0.00346967790505
      y: -0.00137775020998
      z: 0.000222907537717
  covariance: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

Where the pose is the actual position of the robot in relation to the odom frame and the twist part is the velocity relative to this frame. For the GPS we have two topics, one is the fix in /fix:

```
header:
  seq: 3695
  stamp:
      secs: 372
      nsecs: 200000000
  frame_id: "/world"
status:
  status: 0
  service: 0
latitude: 49.8999333906
longitude: 8.89992292839
altitude: 0.38463476648
position_covariance: [25.01, 0.0, 0.0, 0.0, 25.01, 0.0, 0.0, 0.0, 25.01]
position_covariance_type: 2
```

Which gives the position in global coordinates. Gazebo does not support this kind of geo-referenced coordinates, so the GPS plugin internally converts the Gazebo world coordinates to geo-referenced coordinates (WGS84, GNSS) , which is close enough as long as we don't move big distances with the car and are mostly interested about relative movement.
Then there is the /fix_velocity topic of the GPS plugin which gives the velocities as taken from the gps measurements:

```
header:
  seq: 5506
  stamp:
      secs: 553
      nsecs: 300000000
  frame_id: "/world"
vector:
  x: 0.0735152634689
  y: 0.0632961203499
  z: -0.103189235537
```

The velocities are north, west and up direction, also in GNSS.

## 4. Contents of URDF

These are the contents of the final URDF file:

```
<?xml version="1.0" ?>
<robot  xmlns:xacro="http://www.ros.org/wiki/xacro" name="polaris_ranger_ev">

  <xacro:arg name="gpu" default="false"/>
  <xacro:property name="gpu" value="$(arg gpu)" />

  <material name="Black">
      <color rgba="0.0 0.0 0.0 1.0"/>
  </material>
  <material name="Red">
      <color rgba="0.8 0.8 0.8 1.0"/>
  </material>
  <material name="White">
      <color rgba="1.0 1.0 1.0 1.0"/>
  </material>
  <material name="Blue">
      <color rgba="0.3 0.3 0.6 1.0"/>
  </material>



  <joint name="base_link_joint" type="fixed">
      <parent link="base_link_polaris"/>
      <child link="polaris_ranger_ev__chassis"/>
      <origin rpy="0  0        0" xyz="0.85  0   -0.32"/>
  </joint>

  <joint name="polaris_ranger_ev__front_left_steering_joint" type="revolute">
      <parent link="polaris_ranger_ev__chassis"/>
      <child link="polaris_ranger_ev__front_left_wheel_steering_block"/>
      <origin rpy="1.57079  0        0" xyz="1.03  0.5   0.32"/>
```

```
        <axis xyz="0  1  0"/>
        <limit effort="1000.0" lower="-0.8727" upper="0.8727" velocity="1.0"/>
    </joint>

    <transmission name="polaris_ranger_ev__front_left_steering_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <actuator name="polaris_ranger_ev__front_left_steering_motor">
        <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        <joint name="polaris_ranger_ev__front_left_steering_joint">

  <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        </joint>
  </transmission>

    <joint name="polaris_ranger_ev__front_left_wheel_joint" type="continuous">
        <parent link="polaris_ranger_ev__front_left_wheel_steering_block"/>
        <child link="polaris_ranger_ev__front_left_wheel"/>
        <origin rpy="-3.09079  0       0" xyz="0   0  -0.1"/>
        <axis xyz="0.00000000e+00   8.30000000e-04   1.00000000e+00"/>
        <limit effort="0" velocity="100.0"/>
    </joint>

    <transmission name="polaris_ranger_ev__front_left_wheel_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <actuator name="polaris_ranger_ev__front_left_wheel_motor">
        <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        <joint name="polaris_ranger_ev__front_left_wheel_joint">

  <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
        </joint>
  </transmission>

    <joint name="polaris_ranger_ev__front_right_steering_joint" type="revolute">
        <parent link="polaris_ranger_ev__chassis"/>
        <child link="polaris_ranger_ev__front_right_wheel_steering_block"/>
        <origin rpy="1.57079  0        0" xyz="1.03 -0.5   0.32"/>
        <axis xyz="0  1  0"/>
        <limit effort="10000.0" lower="-0.8727" upper="0.8727" velocity="1.0"/>
    </joint>

    <transmission name="polaris_ranger_ev__front_right_steering_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <actuator name="polaris_ranger_ev__front_right_steering_motor">
        <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        <joint name="polaris_ranger_ev__front_right_steering_joint">
```

```
<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      </joint>
  </transmission>


  <joint name="polaris_ranger_ev__front_right_wheel_joint" type="continuous">
      <parent link="polaris_ranger_ev__front_right_wheel_steering_block"/>
      <child link="polaris_ranger_ev__front_right_wheel"/>
      <origin rpy="-0.05079  0       0" xyz="0   0   0.1"/>
      <axis xyz="0.00000000e+00   8.30000000e-04  -1.00000000e+00"/>
      <limit effort="0" velocity="100.0"/>
  </joint>

  <transmission name="polaris_ranger_ev__front_right_wheel_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <actuator name="polaris_ranger_ev__front_right_wheel_motor">
      <mechanicalReduction>1</mechanicalReduction>
      </actuator>
      <joint name="polaris_ranger_ev__front_right_wheel_joint">

<hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
      </joint>
</transmission>

  <joint name="polaris_ranger_ev__rear_left_wheel_joint" type="continuous">
      <parent link="polaris_ranger_ev__chassis"/>
      <child link="polaris_ranger_ev__rear_left_wheel"/>
      <origin rpy="-1.52  0    0" xyz="-0.85  0.6   0.32"/>
      <axis xyz="0.00000000e+00   8.30000000e-04   1.00000000e+00"/>
      <limit effort="20000.0" velocity="100.0"/>
  </joint>

  <transmission name="polaris_ranger_ev__rear_left_wheel_trans">
      <type>transmission_interface/SimpleTransmission</type>
      <actuator name="polaris_ranger_ev__rear_left_wheel_motor">
      <mechanicalReduction>1</mechanicalReduction>
      </actuator>
      <joint name="polaris_ranger_ev__rear_left_wheel_joint">

<hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
      </joint>
</transmission>

  <joint name="polaris_ranger_ev__rear_right_wheel_joint" type="continuous">
      <parent link="polaris_ranger_ev__chassis"/>
      <child link="polaris_ranger_ev__rear_right_wheel"/>
      <origin rpy="1.52  0     0" xyz="-0.85      -0.6  0.32"/>
```

```
        <axis xyz="0.00000000e+00   8.30000000e-04  -1.00000000e+00"/>
        <limit effort="20000.0" velocity="100.0"/>
    </joint>

    <transmission name="polaris_ranger_ev__rear_right_wheel_trans">
        <type>transmission_interface/SimpleTransmission</type>
        <actuator name="polaris_ranger_ev__rear_right_wheel_motor">
        <mechanicalReduction>1</mechanicalReduction>
        </actuator>
        <joint name="polaris_ranger_ev__rear_right_wheel_joint">

  <hardwareInterface>hardware_interface/VelocityJointInterface</hardwareInterface>
        </joint>
  </transmission>

  <link name ="base_link_polaris"/>

    <link name="polaris_ranger_ev__chassis">
        <inertial>
        <mass value="720.0"/>
        <origin rpy="0  0  0" xyz="0.1  0   0.4"/>
        <inertia ixx="140" ixy="0.0" ixz="0.0" iyy="550" iyz="0.0" izz="550"/>
        </inertial>
        <collision>
        <origin rpy="0  0  0" xyz="0.1   0   0.37"/>
        <geometry>
        <box size="1.5 1.34 0.06"/>
        </geometry>
        </collision>
        <visual>
        <origin rpy="0  0  0" xyz="0.1   0   0.37"/>
        <geometry>
        <box size="1.5 1.34 0.06"/>
        </geometry>
        </visual>
        <collision>
        <origin rpy="0  0  0" xyz="-0.9  0   0.9"/>
        <geometry>
        <box size="0.9 1.2 0.01"/>
        </geometry>
        </collision>
        <visual>
        <origin rpy="0  0  0" xyz="-0.9  0   0.9"/>
        <geometry>
        <box size="0.9 1.2 0.01"/>
        </geometry>
        </visual>
        <collision>
```

```
<origin rpy="0  0  0" xyz="-0.45     0     1.02499"/>
<geometry>
<box size="0.05 1.2 0.25"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="-0.45     0     1.02499"/>
<geometry>
<box size="0.05 1.2 0.25"/>
</geometry>
</visual>
<collision>
<origin rpy="0  0  0" xyz="-1.35     0     1.02499"/>
<geometry>
<box size="0.05 1.2 0.25"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="-1.35     0     1.02499"/>
<geometry>
<box size="0.05 1.2 0.25"/>
</geometry>
</visual>
<collision>
<origin rpy="0  0  0" xyz="-0.9      0.6   1.02499"/>
<geometry>
<box size="0.9 0.05 0.25"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="-0.9      0.6   1.02499"/>
<geometry>
<box size="0.9 0.05 0.25"/>
</geometry>
</visual>
<collision>
<origin rpy="0  0  0" xyz="-0.9      -0.6  1.02499"/>
<geometry>
<box size="0.9 0.05 0.25"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="-0.9      -0.6  1.02499"/>
<geometry>
<box size="0.9 0.05 0.25"/>
</geometry>
</visual>
<collision>
```

```
<origin rpy="0  0  0" xyz="-0.1   0  0.56"/>
<geometry>
<box size="0.6 1.22 0.50"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="-0.1   0  0.56"/>
<geometry>
<box size="0.6 1.22 0.50"/>
</geometry>
</visual>
<collision>
<origin rpy="0  0  0" xyz="-0.1   0  0.81"/>
<geometry>
<box size="0.6 1.15 0.1"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="-0.1   0  0.81"/>
<geometry>
<box size="0.6 1.15 0.1"/>
</geometry>
</visual>
<collision>
<origin rpy="0  -0.2  0" xyz="-0.3   0     1.125"/>
<geometry>
<box size="0.06 1.0 0.4"/>
</geometry>
</collision>
<visual>
<origin rpy="0  -0.2  0" xyz="-0.3   0     1.125"/>
<geometry>
<box size="0.06 1.0 0.4"/>
</geometry>
</visual>
<collision>
<origin rpy="0  0  0" xyz="1.05  0   0.7"/>
<geometry>
<box size="0.58 1.0 0.8"/>
</geometry>
</collision>
<visual>
<origin rpy="0  0  0" xyz="1.05  0   0.7"/>
<geometry>
<box size="0.58 1.0 0.8"/>
</geometry>
</visual>
<collision>
```

```
<origin rpy="0  -0.2  0" xyz="-0.3   0.61   1.055"/>
<geometry>
<cylinder length="0.17" radius="0.02"/>
</geometry>
</collision>
<visual>
<origin rpy="0  -0.2  0" xyz="-0.3   0.61   1.055"/>
<geometry>
<cylinder length="0.17" radius="0.02"/>
</geometry>
</visual>
<collision>
<origin rpy="0  -0.2  0" xyz="-0.3   -0.61   1.055"/>
<geometry>
<cylinder length="0.17" radius="0.02"/>
</geometry>
</collision>
<visual>
<origin rpy="0  -0.2  0" xyz="-0.3   -0.61   1.055"/>
<geometry>
<cylinder length="0.17" radius="0.02"/>
</geometry>
</visual>
<collision>
<origin rpy="3.14159  1.53159  3.14159" xyz="0   0.61  1.92"/>
<geometry>
<cylinder length="0.68" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="3.14159  1.53159  3.14159" xyz="0   0.61  1.92"/>
<geometry>
<cylinder length="0.68" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="3.14159  1.53159  3.14159" xyz="0   -0.61  1.92"/>
<geometry>
<cylinder length="0.68" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="3.14159  1.53159  3.14159" xyz="0   -0.61  1.92"/>
<geometry>
<cylinder length="0.68" radius="0.03"/>
</geometry>
</visual>
<collision>
```

```
<origin rpy="1.57079  0        0" xyz="0.325  0   1.89"/>
<geometry>
<cylinder length="1.22" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="1.57079  0        0" xyz="0.325  0   1.89"/>
<geometry>
<cylinder length="1.22" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="1.57079  0        0" xyz="-0.33  0   1.92"/>
<geometry>
<cylinder length="1.22" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="1.57079  0        0" xyz="-0.33  0   1.92"/>
<geometry>
<cylinder length="1.22" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="0   -0.44  0" xyz="0.54  0.61  1.45"/>
<geometry>
<cylinder length="1.04" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0   -0.44  0" xyz="0.54  0.61  1.45"/>
<geometry>
<cylinder length="1.04" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="0   -0.44  0" xyz="0.54 -0.61  1.45"/>
<geometry>
<cylinder length="1.04" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0   -0.44  0" xyz="0.54 -0.61  1.45"/>
<geometry>
<cylinder length="1.04" radius="0.03"/>
</geometry>
</visual>
<collision>
```

```
<origin rpy="0     0.35  0" xyz="0.64 -0.61  0.7"/>
<geometry>
<cylinder length="0.72" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0     0.35  0" xyz="0.64 -0.61  0.7"/>
<geometry>
<cylinder length="0.72" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="0     0.35  0" xyz="0.64  0.61  0.7"/>
<geometry>
<cylinder length="0.72" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0     0.35  0" xyz="0.64  0.61  0.7"/>
<geometry>
<cylinder length="0.72" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="0   -0.14  0" xyz="-0.37  0.61  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0   -0.14  0" xyz="-0.37  0.61  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="0   -0.14  0" xyz="-0.37 -0.61  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0   -0.14  0" xyz="-0.37 -0.61  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.03"/>
</geometry>
</visual>
<collision>
```

```
<origin rpy="0   -0.14  0" xyz="-0.37   0.155  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.023"/>
</geometry>
</collision>
<visual>
<origin rpy="0   -0.14  0" xyz="-0.37   0.155  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.023"/>
</geometry>
</visual>
<collision>
<origin rpy="0   -0.14  0" xyz="-0.37  -0.155  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.023"/>
</geometry>
</collision>
<visual>
<origin rpy="0   -0.14  0" xyz="-0.37  -0.155  1.25"/>
<geometry>
<cylinder length="0.90" radius="0.023"/>
</geometry>
</visual>
<collision>
<origin rpy="0   0.4  0" xyz="-0.38  0.61  1.82"/>
<geometry>
<cylinder length="0.29" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0   0.4  0" xyz="-0.38  0.61  1.82"/>
<geometry>
<cylinder length="0.29" radius="0.03"/>
</geometry>
</visual>
<collision>
<origin rpy="0   0.4  0" xyz="-0.38 -0.61  1.82"/>
<geometry>
<cylinder length="0.29" radius="0.03"/>
</geometry>
</collision>
<visual>
<origin rpy="0   0.4  0" xyz="-0.38 -0.61  1.82"/>
<geometry>
<cylinder length="0.29" radius="0.03"/>
</geometry>
</visual>
<collision>
```

```
    <origin rpy="0   0.4  0" xyz="-0.38   0.155  1.82"/>
    <geometry>
    <cylinder length="0.29" radius="0.023"/>
    </geometry>
    </collision>
    <visual>
    <origin rpy="0   0.4  0" xyz="-0.38   0.155  1.82"/>
    <geometry>
    <cylinder length="0.29" radius="0.023"/>
    </geometry>
    </visual>
    <collision>
    <origin rpy="0   0.4  0" xyz="-0.38  -0.155  1.82"/>
    <geometry>
    <cylinder length="0.29" radius="0.023"/>
    </geometry>
    </collision>
    <visual>
    <origin rpy="0   0.4  0" xyz="-0.38  -0.155  1.82"/>
    <geometry>
    <cylinder length="0.29" radius="0.023"/>
    </geometry>
    </visual>
</link>

<gazebo reference="polaris_ranger_ev__chassis">
    <material>Gazebo/Red</material>
    </gazebo>

<link name="polaris_ranger_ev__rear_left_wheel">
    <inertial>
    <mass value="12"/>
    <origin rpy="0  0  0" xyz="0  0  0"/>
    <inertia ixx="0.5" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.0" izz="1.0"/>
    </inertial>
    <collision>
    <origin rpy="0  0  0" xyz="0  0  0"/><material name="Black">
    <color rgba="0.0 0.0 0.0 1.0"/>
</material>
<material name="Red">
    <color rgba="0.8 0.0 0.0 1.0"/>
</material>
<material name="White">
    <color rgba="1.0 1.0 1.0 1.0"/>
</material>
<material name="Blue">
    <color rgba="0.0 0.0 0.8 1.0"/>
</material>
```

```
    <geometry>
    <cylinder length="0.23" radius="0.32"/>
    </geometry>
    </collision>
    <visual>
    <origin rpy="0  0  0" xyz="0  0  0"/>
    <geometry>
    <cylinder length="0.23" radius="0.32"/>
    </geometry>
    </visual>
</link>

<gazebo reference="polaris_ranger_ev__rear_left_wheel">
    <mu1>0.7</mu1>
    <material>Gazebo/Black</material>
    </gazebo>

<link name="polaris_ranger_ev__rear_right_wheel">
    <inertial>
    <mass value="12"/>
    <origin rpy="0  0  0" xyz="0  0  0"/>
    <inertia ixx="0.5" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.0" izz="1.0"/>
    </inertial>
    <collision>
    <origin rpy="0  0  0" xyz="0  0  0"/>
    <geometry>
    <cylinder length="0.23" radius="0.32"/>
    </geometry>
    </collision>
    <visual>
    <origin rpy="0  0  0" xyz="0  0  0"/>
    <geometry>
    <cylinder length="0.23" radius="0.32"/>
    </geometry>
    </visual>
</link>

<gazebo reference="polaris_ranger_ev__rear_right_wheel">
    <mu1>0.7</mu1>
    <material>Gazebo/Black</material>
    </gazebo>

<link name="polaris_ranger_ev__front_right_wheel">
    <inertial>
    <mass value="12"/>
    <origin rpy="0  0  0" xyz="0  0  0"/>
    <inertia ixx="0.5" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.0" izz="1.0"/>
    </inertial>
```

```
        <collision>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.23" radius="0.32"/>
        </geometry>
        </collision>
        <visual>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.23" radius="0.32"/>
        </geometry>
        </visual>
    </link>

    <gazebo reference="polaris_ranger_ev__front_right_wheel">
        <mu1>0.7</mu1>
        <material>Gazebo/Black</material>
    </gazebo>

    <link name="polaris_ranger_ev__front_right_wheel_steering_block">
        <inertial>
        <mass value="1"/>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
        </inertial>
        <collision
  name="polaris_ranger_ev__front_right_wheel_steering_block_collision">
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.01" radius="0.05"/>
        </geometry>
        </collision>
        <visual name="polaris_ranger_ev__front_right_wheel_steering_block_vis">
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.01" radius="0.05"/>
        </geometry>
        </visual>
    </link>
    <link name="polaris_ranger_ev__front_left_wheel">
        <inertial>
        <mass value="12"/>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <inertia ixx="0.5" ixy="0.0" ixz="0.0" iyy="0.5" iyz="0.0" izz="1.0"/>
        </inertial>
        <collision>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.23" radius="0.32"/>
        </geometry>
        </collision>
        <visual>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.23" radius="0.32"/>
        </geometry>
        </visual>
    </link>

    <gazebo reference="polaris_ranger_ev__front_left_wheel">
        <mu1>0.7</mu1>
        <material>Gazebo/Black</material>
    </gazebo>

    <link name="polaris_ranger_ev__front_left_wheel_steering_block">
        <inertial>
        <mass value="1"/>
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0" izz="1.0"/>
        </inertial>
        <collision
name="polaris_ranger_ev__front_left_wheel_steering_block_collision">
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.01" radius="0.05"/>
        </geometry>
        </collision>
        <visual name="polaris_ranger_ev__front_left_wheel_steering_block_visual">
        <origin rpy="0  0  0" xyz="0  0  0"/>
        <geometry>
        <cylinder length="0.01" radius="0.05"/>
        </geometry>
        </visual>
    </link>

    <gazebo reference="base_link_polaris">
        <gravity>true</gravity>
        <sensor name="imu_sensor" type="imu">
        <always_on>true</always_on>
        <update_rate>100</update_rate>
        <visualize>true</visualize>
        <topic>imu</topic>
        <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
        <topicName>imu</topicName>
        <bodyName>imu_link</bodyName>
        <updateRateHZ>10.0</updateRateHZ>
```

```
      <gaussianNoise>0.0</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset>
      <frameName>imu_link</frameName>
      </plugin>
      <pose>0.63 0.08 1.55 0 0 0</pose>
      </sensor>
  </gazebo>

  <gazebo>
      <plugin name="novatel_gps_sim" filename="libhector_gazebo_ros_gps.so">
      <alwaysOn>1</alwaysOn>
      <updateRate>10.0</updateRate>
      <bodyName>base_link_polaris</bodyName>
      <topicName>fix</topicName>
      <velocityTopicName>fix_velocity</velocityTopicName>
      <drift>5.0 5.0 5.0</drift>
      <gaussianNoise>0.1 0.1 0.1</gaussianNoise>
      <velocityDrift>0 0 0</velocityDrift>
      <velocityGaussianNoise>0.1 0.1 0.1</velocityGaussianNoise>
      <pose>0.63 0.08 1.55 0 0 0</pose>
      </plugin>
  </gazebo>

 <xacro:include filename="$(find velodyne_description)/urdf/HDL-32E.urdf.xacro"/>
  <HDL-32E parent="base_link_polaris" name="velodyne2" topic="/velodyne_points2"
hz="10" samples="220" gpu="false">
      <origin xyz="0.95 0 1.84" rpy="0 0 0" />
  </HDL-32E>
<!--2.16-->

  <gazebo>
      <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
      <robotNamespace>/</robotNamespace>
      <legacyModeNS>true</legacyModeNS>
      </plugin>
  </gazebo>

</robot>
```