Universidad de Alcalá
Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Trabajo Fin de Máster

UGV Navigation in ROS using LIDAR 3D

Autor: Alberto Lázaro Enguita

Tutor/es: Luis Miguel Bergasa Pascual

2016

UNIVERSIDAD DE ALCALÁ
Escuela Politécnica Superior

MÁSTER UNIVERSITARIO EN INGENIERIA INDUSTRIAL

Trabajo Fin de Máster

UGV Navigation in ROS using LIDAR 3D

Autor: Alberto Lázaro Enguita

Tutor/es: Luis Miguel Bergasa Pascual

FECHA: Julio 2016

‘You don’t get what you wish for, you get what you work for’

Unknown

AGRADECIMIENTOS

'I never learned from a man who agreed with me.'

Robert A. Heinlein

En primer lugar quiero dar las gracias a Luis, mi tutor, por la confianza depositada en mí para realizar este TFM, así como por la ayuda que me ha ofrecido durante la realización de mismo. Igualmente, deseo agradecer a los profesores del grupo toda la ayuda prestada a lo largo de la realización de este trabajo.

Como no podía ser de otra forma solo tengo palabras de agradecimiento para mis compañeros del laboratorio, pues pese a no poder compartir tantos momentos con ellos como durante la realización de mi TFG siempre me han tratado genial, me han ayudado en lo que han podido y han contribuido a amenizar los días del congreso, que siempre es de agradecer.

También agradecer a mis compañeros de máster y biblioteca los buenos ratos y la ayuda prestada a lo largo de estos dos años tanto a los que ya conocía tras compartir cuatro años de grado, como a los que he conocido a lo largo del master. Sin duda alguna, sois lo mejor que me llevo de haber llevado a cabo este máster.

Tampoco quiero olvidarme de mis compañeros de trabajo, con los cuales he compartido muchos momentos y siempre han sido una ayuda más para evadirme, aunque solo fuese por unos momentos, del trabajo y los estudios.

Como no podía ser de otra forma agradezco a mis amigos todo su apoyo y ayuda, pues siempre me han animado a seguir hacia delante y han conseguido que me evadiera de los problemas diarios, aunque fuese solo por unos momentos.

Por último, agradezco a mi familia el apoyo recibido en todo momento, en especial a mis padres y mi hermano los cuales han contribuido a mi formación y han sido siempre una gran ayuda, pues sin ellos estoy seguro no habría sido posible llegar hasta aquí, gracias.

ABSTRACT

This works addresses to give a step forward the achievement of robust Unmanned Ground Vehicles (UGVs), which can drive in urban environments. More specifically, it focuses in the management of a four wheeled vehicle in ROS using mainly the inputs provided by a LIDAR 3D.

Simulations were carried out in ad-hoc scenarios designed and run using GAZEBO. Visual information provided by sensors is processed through PCL library. Thanks to this processing the needed parameters to manage the UGV are obtained and its guidance can be carried out though a PID controller.

Keywords: UGV, ROS, PCL, GAZEBO, LIDAR 3D.

RESUMEN

El foco de este trabajo consiste en avanzar un paso hacia la consecución de vehículos terrestres no tripulados robustos, que puedan circular en zonas urbanas. Más concretamente se centra en el manejo de un vehículo de cuatro ruedas en ROS usando, sobre todo, las entradas proporcionadas por un LIDAR 3D.

Las simulaciones se llevaron a cabo en escenarios ad-hoc diseñados y ejecutados usando GAZEBO. La información visual de los sensores es procesada mediante la librería PCL. Gracias a este procesamiento se obtienen los parámetros para conducir el UGV y su guiado puede ser llevado a cabo mediante un controlador PID.

Palabras clave: UGV, ROS, PCL, GAZEBO, LIDAR 3D.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. Introduction

1.1 Transport media

Since time immemorial, transportation is one of the most important activities for humans. From its origin, humans move, walk and displace. That makes them long to reach further and further to satisfy their purposes.

Since its origin, transportation has suffered a huge development. Nowadays we can distinguish among different modes of transport as it can be seen in Table 1:

- Air: aviation is able to quickly transport people and limited amounts of freight over longer distances reaching speeds close to 950 kilometers per hour. Helicopters can be for short distances or inaccessible places.
- Rail: intercity trains are long-haul services connecting cities. Modern high-speed rail is capable of speeds up to 350 kilometers per hour but it requires special build tracks.
- Road: it includes buses, trucks, automobiles, motorcycles, bicycles and pedestrians. Automobiles provide high flexibility with low capacity, but require high energy and area use.
- Water: although slow, modern sea transport is a highly efficient method of transporting large quantities of goods.
- Pipeline: send goods through a pipe. Most commonly liquids and gases are sent, but pneumatic tubes can also send solid capsules using compressed air.

Characteristics	Air	Rail	Road	Water	Pipeline
Commercial speed	- Very high. - It needs other modes, which reduces commercial speed.	- Medium speeds - Influenced by the exploitation characteristics	-Just overcomed by Air. -Door to door.	- Low speeds.	- Low.
Network amplitude	- Quite low. - It needs other modes.	- Medium - It sometimes needs other modes	- Most complete network. - Access almost everywhere.	- Incompleted. - Reduced to a number of ports and terminals.	- Small.
Independence	- It depends a lot of environmental conditions.	- It does not depend a lot of environmental conditions	-Medium. Better than rail and pipeline but worse than Air.	- Dependence of environmental conditions.	- It does not depend on environmental conditions.
Capacity	- Restricted by size and weight.	- Second in terms of capacity	-Medium. - Size and Weight limitation.	- Highest capacity mode. - Advantage with big and heavy freight.	- Low, gases and liquids. - Big sizes.
Frequency	- Quite high.	- Medium	- High. - It can adapt to the rest of modes	- Low.	- Reduced.
Cost	- The highest. - Lower fixed costs.	- Medium - Transport door to door problem	-High than water, rail and pipeline. Less fixed costs.	- Similar to pipeline. - Lower than other modes.	- The lowest.
Best for	- High Price/Size ratio. - Long distances.	- Raw materials - Industrial products.	- Almost all. - Weight/Size limitation.	- Big and heavy freight. - Raw materials.	- Gases and liquids. - Solids in suspension.

Table 1: Modes of transport comparative.

1.2 Unmanned vehicles

All the modes of transportation have developed a lot since they were invented. The last tendencies among their development are focusing on their automation. As a consequence of this automation different concepts have appeared:

- Unmanned Ground Vehicles (UGV), which can be seen in Figure 1
- Unmanned Aerial Vehicle (UAV) which is shown in Figure 2.
- Unmanned Underwater Vehicle (UUV), which is depicted in Figure 3.

Focusing on the line that this work describes, UGVs will be analyzed hereafter.

Figure 1: Unmanned Ground Vehicle (UGV).

Figure 2: Unmanned Aerial Vehicle (UAV).

Figure 3: Unmanned Underwater Vehicle.

1.3 Unmanned ground vehicles

Unmanned Ground Vehicles (UGVs) are becoming more and more used every single day. They can carry out lots of different tasks with no human supervision, achieving lower costs or working in areas where it would be dangerous for humans to work in. Those are the reasons why they are commonly used in industrial environments.

Industrial applications have a particularity, which makes easier the management of these vehicles, the environment is controlled. According to the guided systems the unmanned vehicles use, they can be classified as shown in Figure 4.

In other applications where the environment is uncontrolled the sensors and manage systems should be more complex to ensure success in the vehicle autonomous management. This is our approach, because our goal is the UGV navigation in urban scenarios.

In this work, an UGV is managed through ROS using mainly a LIDAR 3D sensor. This management is carried out in GAZEBO, a simulator where the needed models have been built.

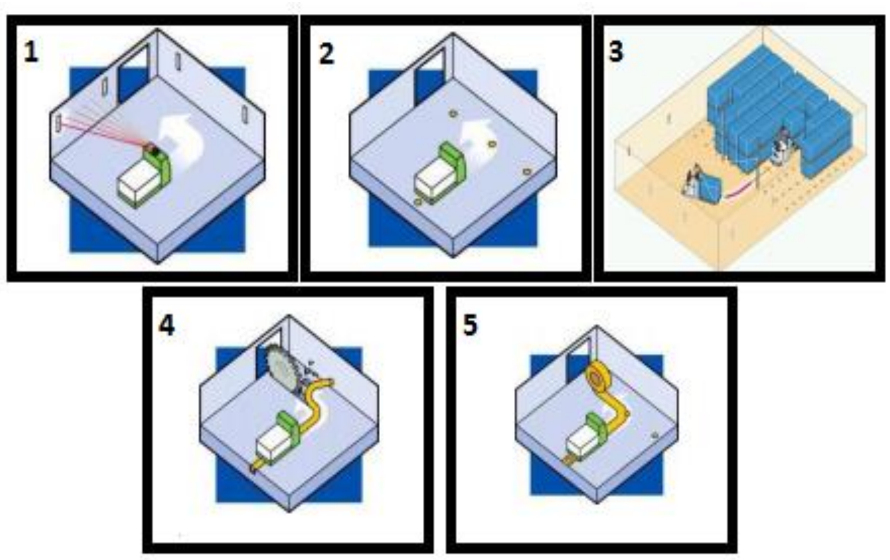


Figure 4: (1) Laser UGV (2) Magnetic UGV (3) Dual UGV (4) Rail guided UGV (5) Optical band UGV.

The reason why GAZEBO world has been build is to create a virtual reconstruction of real scenarios, where testing the perception and vehicle management systems. Virtual simulations are usually used in the first implementation steps of autonomous vehicles. It allows us to test the developed systems as many times as needed in a safety way with reduced costs.

The sensor used to be mounted in the vehicle is the VLP-16 (Velodyne Lidar Puck). It creates a 360º 3D images using 16 laser/detector pairs mounted in a compact housing. The housing rapidly spins to scan the surrounding environment. The lasers fire thousands of times per second, providing a high resolution 3D point cloud in real time.

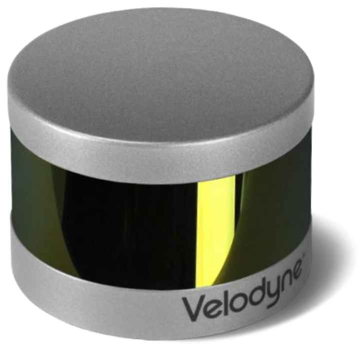


Figure 5: VLP – 16.

2 State of the art

Autonomous ground vehicle navigation in uncontrolled environments is a quite active research area in the world. The success facing this challenge is closer and closer thanks to the amount of different technologies which are continuously being improved. Autonomous navigation has multiple different purposes such as goods transportation, people transportation, etc. A vital component for unmanned vehicles is its perception system. It gets information about the surrounding environment, which will be processed to act in consequence.

Perception systems can be formed by one or more sensors, which are able to capture environment data. There are lots of different ways of acquiring this data. This allows to design a perception system which fits the purpose we are looking for.

It is possible to build a system with different technology sensors [1] or a system with several sensors which uses the same technology [2].

2.1 Perception systems

2.1.1 Flight time systems

These systems carry out a measure of the distance between them and an object through the measure of the time that a beam needs to reach that object and be detected by the receiver [3].

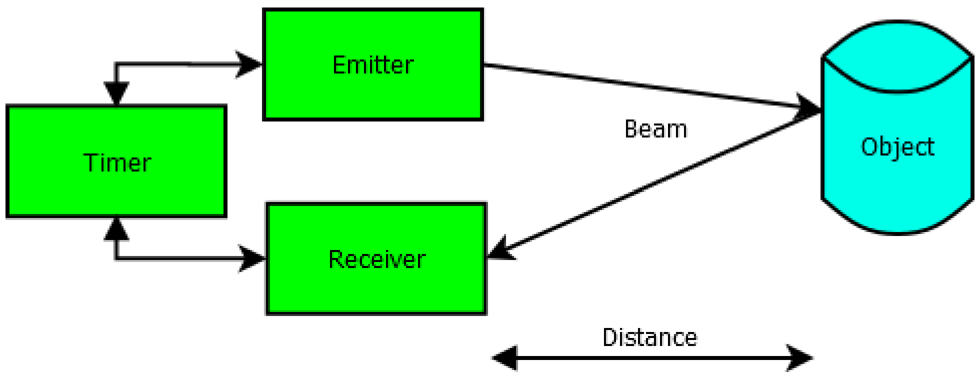


Figure 6: Time flight system scheme.

These systems can use a pulsed laser or a laser beam, depending on what it is used the estimation will be different.

2.1.2 Phase different systems

This method allows the best precision. The phases of the sent and received signals are studied to estimate the distance. The signal received would have travelled a distance that is equal to a certain integer number of its wavelengths plus a gap.

It is really important not to lose the tracing of the phase due to it is essential to carry out a precise distance estimation. These systems are quicker than flight time based systems [4].

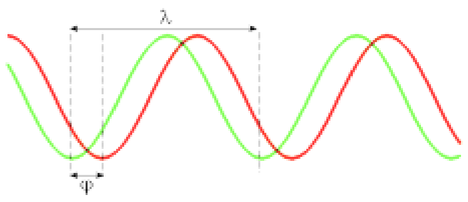


Figure 7: Gap between signals example

2.2 Perception system technologies

2.2.1 Continuous wave RADAR

The main task of these sensors is to detect the objects and measure their relative positions. To do it these systems requires a modulation in the sent signal that can be in amplitude or frequency.

This system is time flight, it estimates the distances using the time that it takes the signal to reach the object and go back. This technique has been used in different unmanned ground vehicles applications such as [5] and [6].

2.2.2 Ultrasounds

These sensors are based in the time flight concept. Distance is computed by measuring the wave´s return time thanks to the sound speed.

They are not specially used for navigation in uncontrolled scenarios because they have a big dependency on environmental conditions such as temperature and humidity. Despite the fact that they are not the ideal sensors for these sort of applications they have been used in [7].

2.2.3 Stereo cameras

The main idea of these systems design is inspired in the human vision. It consists of two cameras fixed in an axis with a known distance between them [8]. Each camera obtains an image; these two captured images allow estimating depth distances using complex calculations which have high computational costs.

To make this depth estimation possible it is necessary to fulfill some requirements:

- Both images shall have the same brightness.
- Both images shall have the same light intensity.
- Both images shall have the same contrast.
- Both images shall be similar with an underhanded part.

There is a trade-off between overlap and opening, with higher overlap the precision is better while opening is less and vice versa.

This technology is used in robots such as Curiosity [9] and in other projects: [10] and [11].

2.2.4 3D infrared cameras

Short scope infrared cameras obtain depth information about the environment allowing reconstructing a 3D image of the scenario where it is. These cameras are based on time flight technology.

There are different techniques used to achieve the distance measure but the most used is to emit an infrared pattern which will be received back by the sensors. 3D infrared cameras are quite used to detect humans, locate walls and recognize objects [12].

Their advantages are their price and their low power consumption while as disadvantage they present a bad performance under solar light conditions.

2.2.5 LIDAR

LIDAR stands for LIght Detection And Ranging. These sensors are quite used in autonomous navigation thanks to their brilliant performance. They offer high precision, reliability and high speed data acquisition [1] [2].

The Advantages of a LIDAR System are [13]:

- Measures accuracy, due to they are invariant to light intensity.
- Device alignment is not critic to obtain a good performance.
- Easy angular opening configuration
- Quick data processing.
- Low false positive rates

These advantages make them a good choice for lots of applications [14]. These devices have an emitter and a receiver which uses the flight time to calculate the distance to the object.

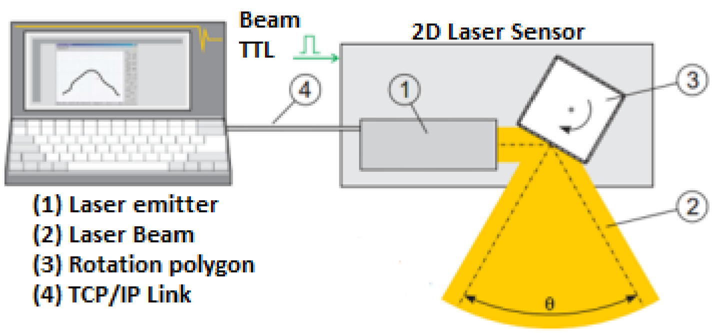


Figure 8: 2D LIDAR Working.

2D LIDAR has a laser diode which emits a ray through a rotating mirror, achieving a set of points in a bidimensional plane to be captured by the receiver. The explained process can be seen in Figure 8.

3D LIDAR works in a similar way to 2D ones and are based on them. 3D LIDARs can be obtained heaping several 2D with different angles or adding to a 2D sensor a rotation system which help the sensor to get data of different heights.

2.3 Vehicles

Unmanned vehicles can be classified in 3 different categories depending on the environment they are used:

- UGVs (Unmanned Ground Vehicles) [15]
- UAVs (Unmanned Aerial Vehicles) [16]
- UUVs (Unmanned Underwater Vehicles) [17]

Additionally to sensors a navigation system is necessary to complete the intelligence of Unmanned Vehicles. Their target is to plan trajectories avoiding obstacles or hurdles found in the vehicle´s path [18].

To carry out this work different platforms and libraries have been used to develop the computational systems that carries out the necessary tasks. These platforms or libraries are:

- ROS [19] [20] [21].
- GAZEBO (DRCSim) [22] [23].
- PCL [24] [25].

Different works can be found where these platforms have been used for similar purposes such as [26], but there is no work that joins the power or both platforms with 3D LIDAR sensors, GPS, cameras and IMUs.

3 Hypothesis and methodology

After showing the problem statement of this work and performing a background research about the related work, it's time to formulate the hypothesis that will be developed in this thesis. Attached to it, specific objectives from the initial statement and the methodology to reach them can be set.

Hypothesis formulation

After the study of the state of the art, it has been found that among the different devices which can be used to capture the environment, LIDAR sensors are the best ones in terms of performance. Among them 3D LIDAR sensors are the ones which best fit our requirements to obtain the most complete information about the environment. Among them Velodyne LIDAR Puck 16 (VLP-16) has been use as the ideal one due to the performance it offers. To compute the huge amount of points that VLP-16 captures Point Cloud Library will be use because it offers us lots of functions which ease the creation of complex algorithms through their use. .

The hypothesis of this work is to compute the three dimensional information provided by the sensor to estimate the target point that should be reached by the vehicle to ensure its integrity. GPS will be used to reference any point of the environment regard a fixed origin, if necessary.

To show the systems functionality, a simulation environment must be created which include the necessary sensors for the described purpose. The developed systems should be modular, to ease its future use in a real environment. To carry out the main objective different programs must be created using GAZEBO and ROS. GAZEBO is needed to simulate the environment and ROS to execute the algorithms that contains the smart functions.

Method for testing the hypothesis: specific goals

In order to achieve a conclusion and to validate the hypothesis some objectives are needed to overcome. These objectives and the methodology to achieve them are the following:

- Analyze real simulation environments which allow building a work scene with different elements. Among these elements it should provide the possibility to include sensors, vehicles and roads.
- Analyze robot development environments which ease the integration of the necessary programs to choose the one that best fit our requirements.
- Build a work scene in the real simulation environment, which will be used to carry out all the simulations and test our system. This work scene should have roads with curbs.
- Create the physical model of the necessary sensors to capture the three dimensional information which is necessary to guide the system that will be created.
- Create the virtual model of the necessary sensors to capture the three dimensional information which are necessary to guide the system that will be created.

- Create a vehicle which can be added to the environment and allow to represent it in the robot development environment. This vehicle should provide an interface to move it as a real one, so gas pedal, brake pedal and steering among others must be implemented.
- Include the vehicle and the sensors models to the world.
- Establish communication between the modular parts of the project in order to make that information captured from the work scene is sent to the processing part of the system where all the computations will be process in order to guide the vehicle.
- Develop a node which uses the vehicle interface to drive it using a keyboard, a joystick or a hand wheel. These nodes will allow a human to move the car as it is wanted.
- Process the input data to estimate the parameters of the line which describes the curbs. After that these lines parameter will be used to estimate the target point which must be reached by the vehicle.
- Provide a proposal to implement the system that guide the vehicle to the target point.
- Carry out the necessary simulations to get results and redact conclusions about the created system and its performance.

4 System overview

The system, which has been done in this work, contains elements in different platforms. These platforms are ROS and GAZEBO.

The reason why these two platforms are needed is that all the tests have been done using simulations, so in order to ease its future use in real applications the simulation environment and the processing part must be separated into different modules.

Simulations are quite useful, they allow us to carry out tests that otherwise would not be possible. For instance, by the moment that this work was done we did not have any VLP-16 sensor. Simulations advantages are not just a matter of material needs, they also allow us to carry out tests in a safety way due to there is no possible human damage.

In Figure 9 the architecture of the system is presented:

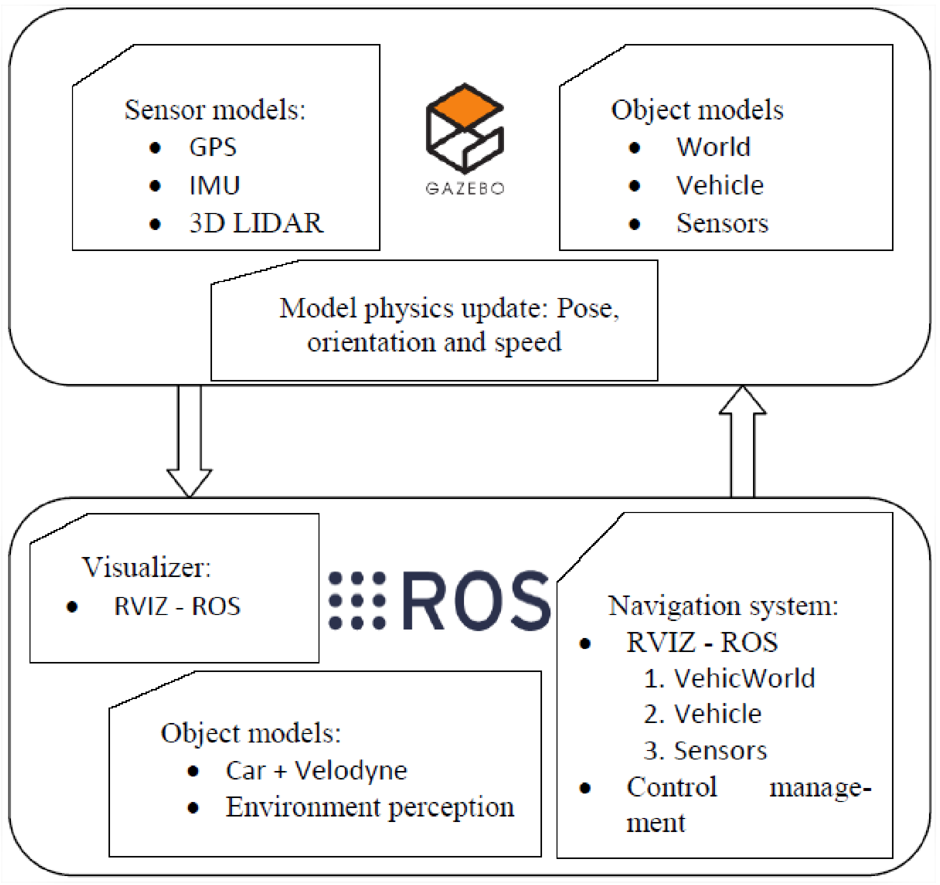


Figure 9: Proposed system architecture.

Due to the reasons presented above the system can be divided in two different parts:

- GAZEBO: substitutes the real world to enable the possibility to carry out simulations. The elements that we will use in the real world should be modeled and introduced in the created world.
- ROS: where the processing algorithms are implemented.

One of the most important points is the communication between ROS and GAZEBO. It will be implemented though ROS communication system which is based in publishers and subscribers.

5 Software analysis

Ubuntu (Linux)

The OS which has been chosen in this project is one of the Linux distributions. The reasons why Linux OS is used are:

- Freedom: Most Linux distributions are free so you do not have to pay for using it.
- Stability: Linux systems rarely crash because their memory protection.
- Multitask, multiplatform, multiuser and multiprocessor.
- Use of static and dynamic linked library.

Figure 10: Ubuntu Logo.

The used Linux distribution information is:

- Distribution: Ubuntu, which logo can be seen in Figure 10.
- Description: Ubuntu 14.04.4 LTS.
- Release: 14.04.
- Codename: trusty.

GAZEBO

GAZEBO is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environment. While similar to game engines, GAZEBO offers physics simulation at a much higher degree of fidelity, a suite of sensors and interfaces for both users and programs. Its logo is depicted in Figure 11.

Figure 11: Gazebo Logo.

Typical uses of GAZEBO include:

- Testing robotics algorithms, shown in Figure 12.
- Designing robots, depicted in Figure 12.
- Performing regression testing with realistic scenarios displayed in Figure 12.

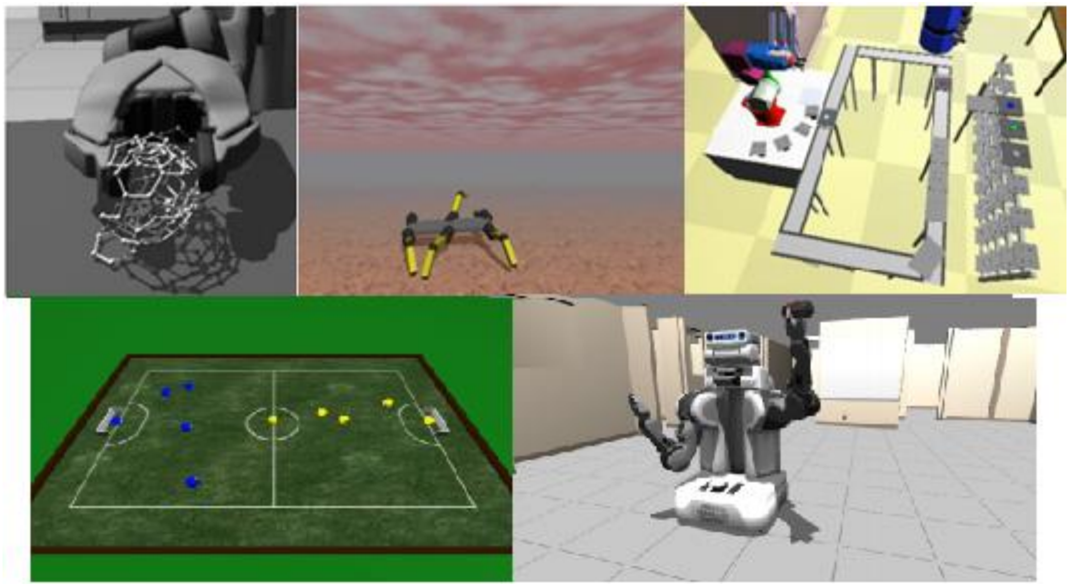


Figure 12: Examples of GAZEBO typical uses.

A few key features of GAZEBO include:

- Multiple physics engines
- A rich library of robot models and environments
- A wide variety of sensors
- Convenient programmatic and graphical
- The opportunity to create your own robot or sensor if they are not available yet.

GAZEBO is preferably used in Linux, more concretely in Ubuntu. To run it the computer shall fulfill a set of requirements which involves hardware and software requirements. These requirements are shown in Figure 13. Hardware requirements are depicted in orange color, while software requirements are represented in blue.

Simulation elements

GAZEBO includes lots of elements, which take part in simulations. These elements are:

- World: collection of models, lights, plugins and global properties. Some examples are shown in Figure 14.
  - Simple: scenarios which are normally used for objet manipulation and perception.
  - Indoor: environments typically used for path planning and mobile manipulation. Reconstructions of real indoors spaces can be done.

- Outdoor: usually employed with aerial, legged or outdoor mobile robots. Normally the working spaces in this sort of environments are bigger.

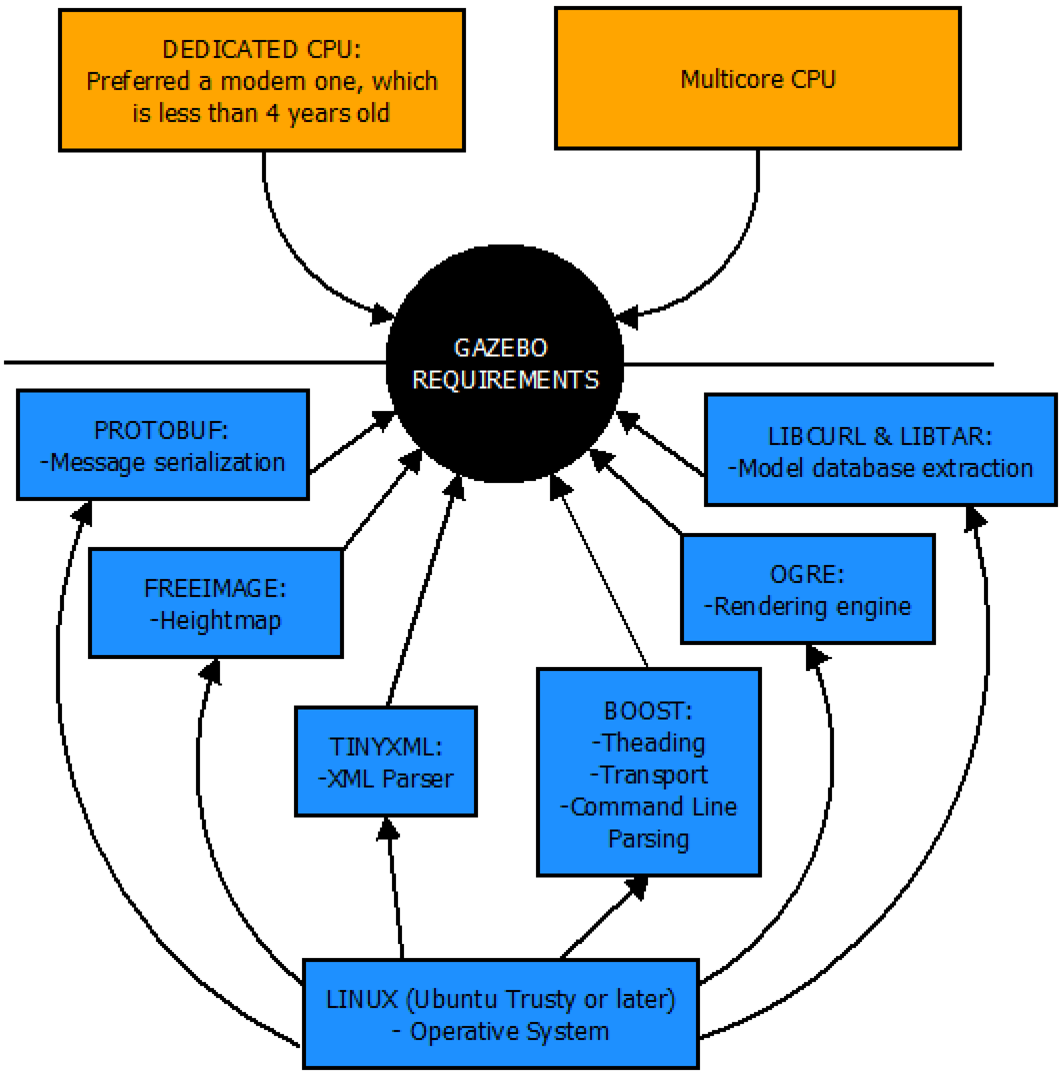


Figure 13: GAZEBO Requirements.

- Models: collection of links, joints, sensors and plugins.
  - Robot Models: simple platforms (shapes and simple skeletons), realistic physical properties (surface friction, mass and inertia) and full sensor suite (laser, cameras, kinect, contact) can be part of a robot model.
  - Non-Robot Models: elements that form the simulation environment such as walls, grounds, furniture, tools, etc.
  - To build efficient robot models the following indications shall be fulfilled:
    - Contacts shall be limited to 3
    - Kinematic trees works better than loops
    - The number of joints shall be reduced as much as possible.
    - Primitives are more efficient than trimeshes.
    - Collision mesh size shall be limited to 5K of triangles
    - Visual mesh size shall be limited to 5K of triangles
    - Limit image/depth sensor resolution or rate.

Figure 14: Types of worlds.

- Links: collection of collision and visual objects.
    - Inertial: mass, moment of inertia.
    - Collision: used by collision engine to generate contact joints for the physics engine.
    - Visual: used by render engine to generate images for GUI and camera or depth sensors.
- Collision Objects: geometry that defines a colliding surface. Some examples are shown in Figure 15.
    - Simple shapes: sphere, cylinder, box, plane.
    - Complex shapes: height maps, meshes.

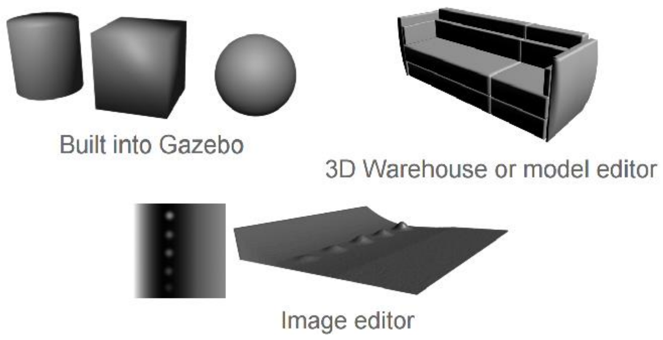


Figure 15: Collision Objects.

- Visual Objects: geometry that defines visual representation.
- Joints: constraints between links.
    - Prismatic: 1 DOF translational.
    - Revolute: 1 DOF rotational.
    - Revolute 2: Two revolute joints in series.
    - Ball: 3 DOF rotational
    - Universal: 2 DOF rotational
    - Screw: 1 DOF translation, 1 DOF rotational
- Sensors: collect, process, and output data.
    - Ray: produces range data.
    - Camera: render to off-screen buffer.

- Kinect: depth camera.
- Laser: CPU and GPU based ray casting.
- Contact: produces collision data, which are generated by collision engine.
- RFID: information generated from model positions.
- Force torque: specific to joints at the moment.

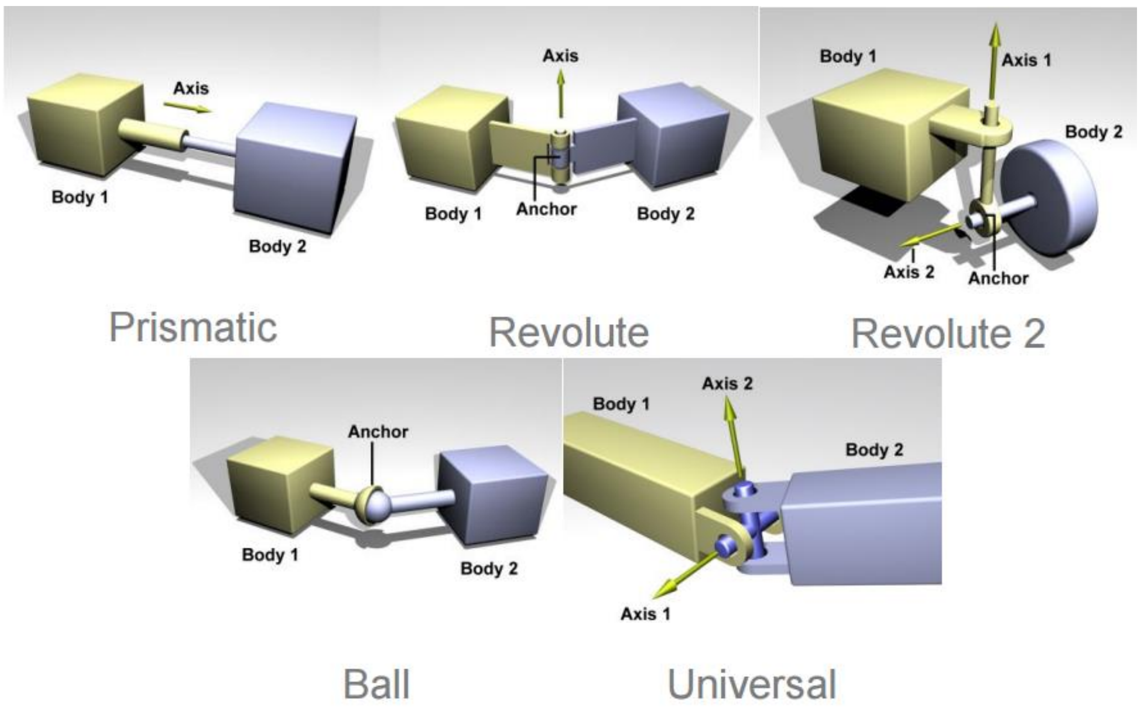


Figure 16: Types of joints.

- Lights
  - Point: omni-directional light source (a light bulb).
  - Spot: directional cone light (a spot light).
  - Directional: parallel directional light (sun).
- Plugins: code attached to a World, Model, Sensor, or the simulator itself. System plugins control the load and initialization process. World plugins manage the entire models and physics engine. Model plugins cope with joint and links supervision. Sensor plugins guide the data generation and processing.

Building models

GAZEBO offers a big amount of different models which can be imported and used in our simulations. Even more they can be founded in different databases available in Internet. It also allows building our own models. To do it the different steps should be followed (see Figure 17):

- Step 1, Collect meshes: custom meshes can be done, exported from Solid Works or obtained from online repositories. To achieve proper meshes their complexity should be reduced and they should be scaled. The mesh should be moved to the center of coordinates (0,0,0), which will ease their placement in GAZEBO Worlds.

- Step 2, Make an SDF file: starting from the static models links, joints sensors and plugins, one by one should be added, testing afterthat the model works as required.
- Step 3, Include the Model in a World: to do it the following syntax should be used:

```
<include>
<url>model://my_model</url>
</include>
```

When possible the number of joints should be reduced to make our models more efficient.

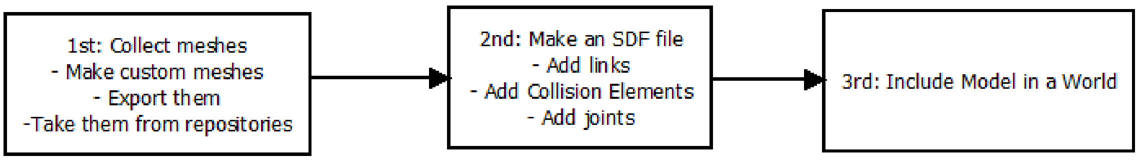


Figure 17: Steps to build a model in GAZEBO.

Architecture

GAZEBO architecture is based in the separation of physics and visualization. This separation is implemented by a client-server system, see Figure 18. The server, which is named as “gzsever” simulates the physics rendering and sensors. The client, called “gzclient” provides a graphical interface to visualize and interact with the simulation. Communication between these two executable programs is carried out through GAZEBO communication library.

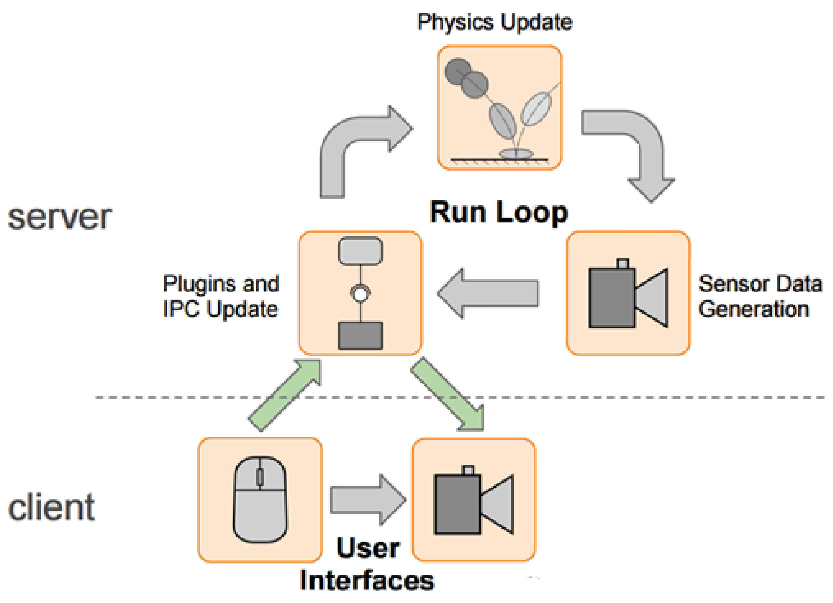


Figure 18: GAZEBO Architecture.

The communication library uses the open source "Google Protobuf" for the message serialization and boost::ASIO for the transport mechanism. It makes possible the publish/subscribe communication paradigm. This mechanism allows introspection of a running simulation, and provides a proper mechanism to control aspects of GAZEBO.

For instance, when a simulated world publishes the pose updates of a body, sensor generation and GUI will use these messages to update the outputs according to the received messages.

GAZEBO master is a topic name server which provides namelookup, and topic management. Multiple physics simulations, sensor generators and GUIs can be handled by a single master.

System components

GAZEBO system components are enumerated hereafter, see Figure 19:

- Physics Library: loads and runs the dynamics engine.
- Sensor Library: generates sensor data.
- Rendering Library: draws the world for the GUI and Sensor Library.
- Transport and Messages Library: implements socket-based connections for message passing
- Math and Common Libraries: internal math functions, and shared utilities.
- GUI and Command line tools: Include executables to visualize and manipulate simulations.

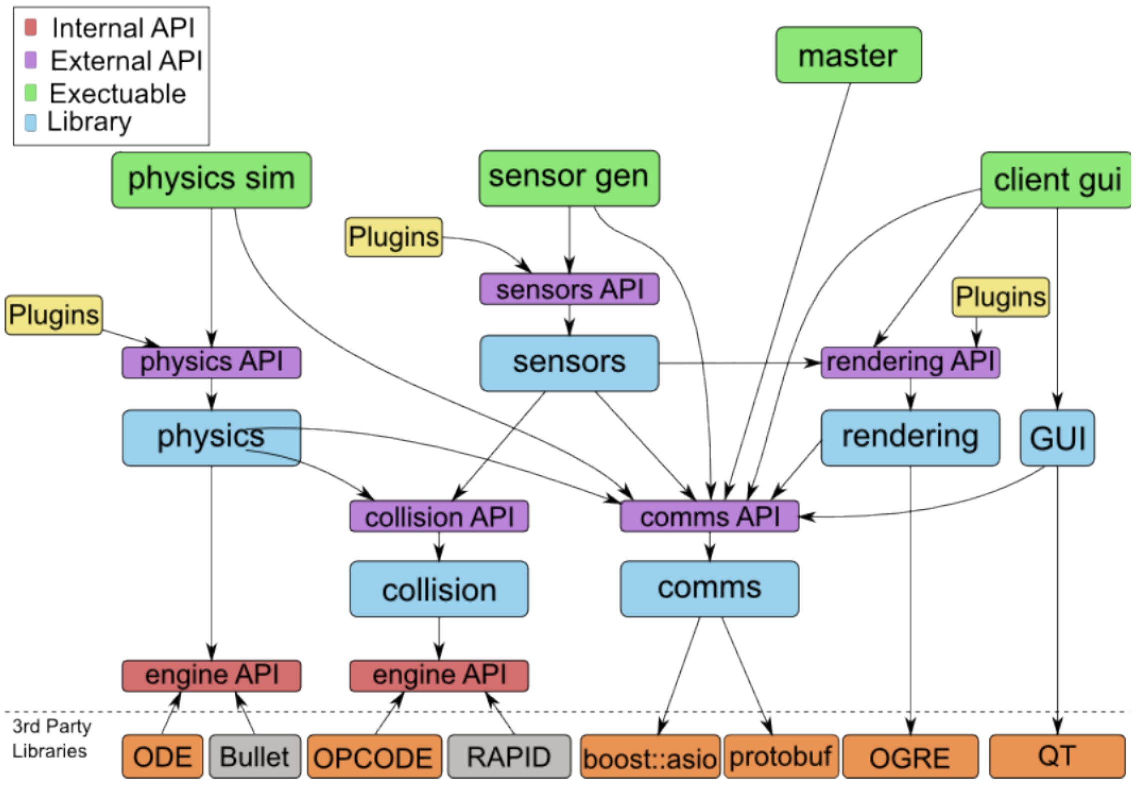


Figure 19: GAZEBO Dependency Graph.

Robot operating system

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

Figure 20: ROS Logo.

Creating robust and general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly among several tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.[4]

As a result, ROS was built from the ground up to encourage collaborative robotics software development. For instance, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was specifically designed for groups like these to collaborate and build upon each other's work, as is described throughout this site [4].

The core of ROS is licensed under the standard three-clause BSD license. BSD licenses are a family of permissive free software licenses which imposes minimal restrictions on the redistribution of covered software. It is a very permissive open license that allows be re-used in commercial and closed source products.

While ROS core parts are licensed under the BSD license, others are usually used in the community packages, such as the Apache 2.0 license, the GPL license, the MIT license, and even proprietary licenses. Each package is required to specify a license, so that it is easy to quickly identify if a package will meet your licensing needs.

ROS basic elements

Node: a node is a process that performs computation. Nodes are combined together into a graph and communicate with the others using streaming topics. The purpose of the node architecture is that they operate at a fine-grained scale. For instance, a vehicle control system would be used by several nodes: one which controls a laser range-finder, another one which manages the steering and gas and another one which is in charge of the location.

Topic: topics are named busses over which nodes exchange messages. They are intended for unidirectional, streaming communication. They should be used for continuous data flow. They allow publishing and subscribing data at any time.

Services: services should be used for remote procedure calls that terminate quickly for querying the state of a node or doing a quick calculation.

Actions: actions pools characteristics from nodes and services which make them perfect to be used for any discrete behavior that moves a robot or that runs for a longer time but provides feedback during execution.

Message: a message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, Boolean, etc.) are supported, as they are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structs).

Communication infrastructure

A communication system is often one of the first needs to arise when implementing a new robot application. ROS's built-in and well-tested messaging system saves its time by managing the details of communication between distributed nodes via topics. The communication is one of the most powerful features in ROS, it offers a message passing interface that provides inter-process communication and is commonly referred as middleware.

Every topic has its own unique name and it can just carry one type of message. There are several types of messages included in ROS which can be used depending on the information that is needed to be transmitted. One example of message, which is used in this project, is "sensor_msgs/PointCloud2ConstPtr".

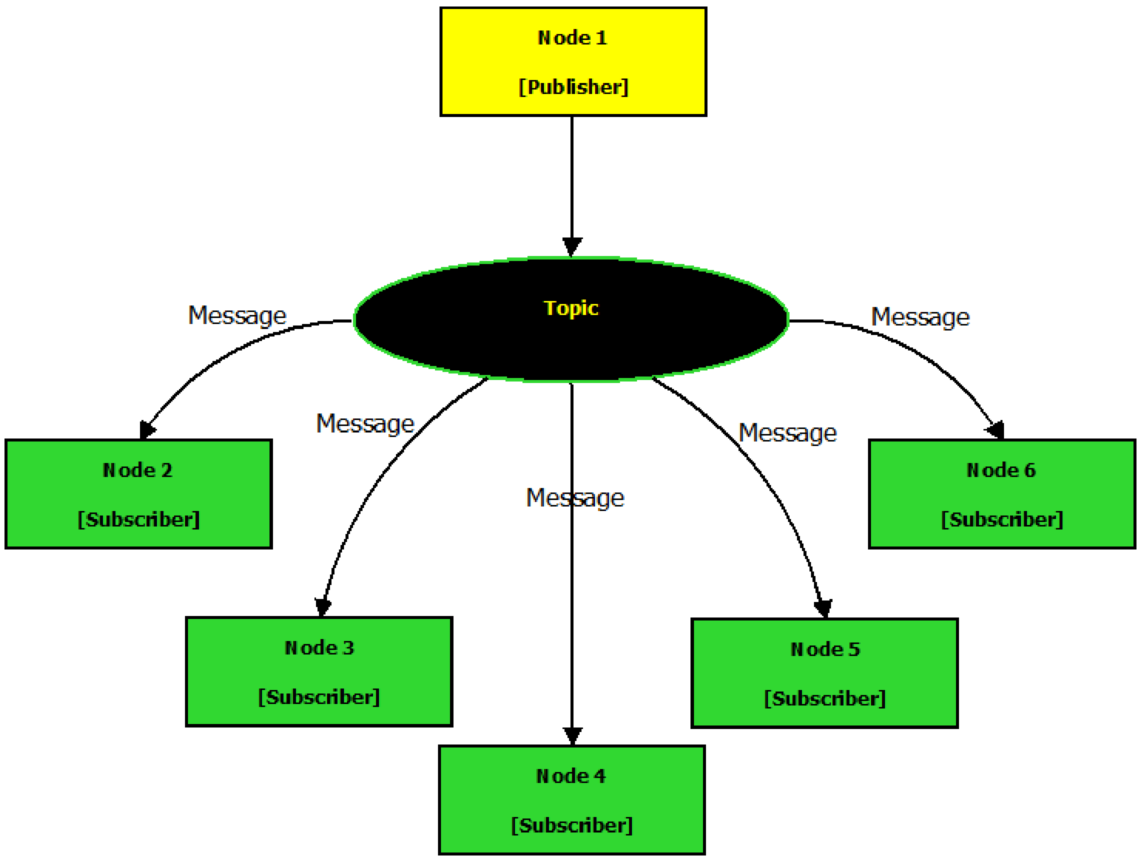


Figure 21: ROS topic communication scheme.

To establish the communication among different modules the following requirements shall be taken into account:

- The node which contains the information to be transmitted shall publish it in a topic.
- All the different nodes which need the information shall subscribe to the topic where information is being published.

The structure shown in Figure 21 is the desirable one. As it can be seen more than one node can subscribe to a topic at the same time. Several nodes can publish in the same topic simultaneously but that is not recommended.

Code 1 shows how to create a node that publishes a point cloud named “msg”. This point cloud can be filled up with the amount of point that we need to transmit.

```
#include <ros/ros.h>
#include <pcl_ros/point_cloud.h>
#include <pcl/point_types.h>

typedef pcl::PointCloud<pcl::PointXYZ> PointCloud;

int main(int argc, char** argv)
      {
      ros::init (argc, argv, "pub_pcl");
      ros::NodeHandle nh;
      ros::Publisher pub = nh.advertise<PointCloud> ("points2", 1);

      PointCloud::Ptr msg (new PointCloud);
      msg->header.frame_id = "some_tf_frame";
      msg->height = msg->width = 1;
      msg->points.push_back (pcl::PointXYZ(1.0, 2.0, 3.0));

      ros::Rate loop_rate(4);
      while (nh.ok())
            {
            msg->header.stamp = ros::Time::now().toNSec();
            pub.publish (msg);
            ros::spinOnce ();
            loop_rate.sleep ();
            }
      }
```

Code 1: Publishing point clouds

Code 1 shows an example of a subscriber node, which reads the point cloud that is being published in topic “points2”. “Callback” function can be modified as required to process the received point cloud.

```
#include <ros/ros.h>
#include <pcl_ros/point_cloud.h>
#include <pcl/point_types.h>
#include <boost/foreach.hpp>

typedef pcl::PointCloud<pcl::PointXYZ> PointCloud;

void callback(const PointCloud::ConstPtr& msg)
      {
      printf ("Cloud: width = %d, height = %d\n", msg->width, msg-
>height);
    BOOST_FOREACH (const pcl::PointXYZ& pt, msg->points)
    printf ("\t(%f, %f, %f)\n", pt.x, pt.y, pt.z);
```

```
        }

int main(int argc, char** argv)
        {
        ros::init(argc, argv, "sub_pcl");
        ros::NodeHandle nh;
        ros::Subscriber sub = nh.subscribe<PointCloud>("points2", 1,
callback);
        ros::spin();
        }
```

Code 2: Subscribing to point clouds

Message recording and playback

Publish/subscribe system is anonymous and asynchronous. Data can be easily captured and replayed without any changes to code thanks to this anonymity. Assuming we have a task, which reads data from a sensor, and you are developing another task that processes the data produced by the first task. ROS makes it easy to capture the data published by the first task to a file, and then republish that data from the file at a later time. The message-passing abstraction allows the task, which is being developed, to be agnostic with respect to the source of the data. This is a powerful design pattern that can significantly reduce the development effort and promote flexibility and modularity in your system.

The rosbag command-line provides the following functionality using the proper commands, which are shown hereafter.

- record: record a bag file with the contents of specified topics.
- info: summarize the contents of a bag file.
- play: play back the contents of one or more bag files.
- check: determine whether a bag is playable in the current system, or if it can be migrated.
- fix: repair the messages in a bag file so that it can be played in the current system.
- filter: convert a bag file using Python expressions.
- compress: compress one or more bag files.
- decompress: decompress one or more bag files.
- reindex: reindex one or more broken bag files.

Robot specific features

Command line tools

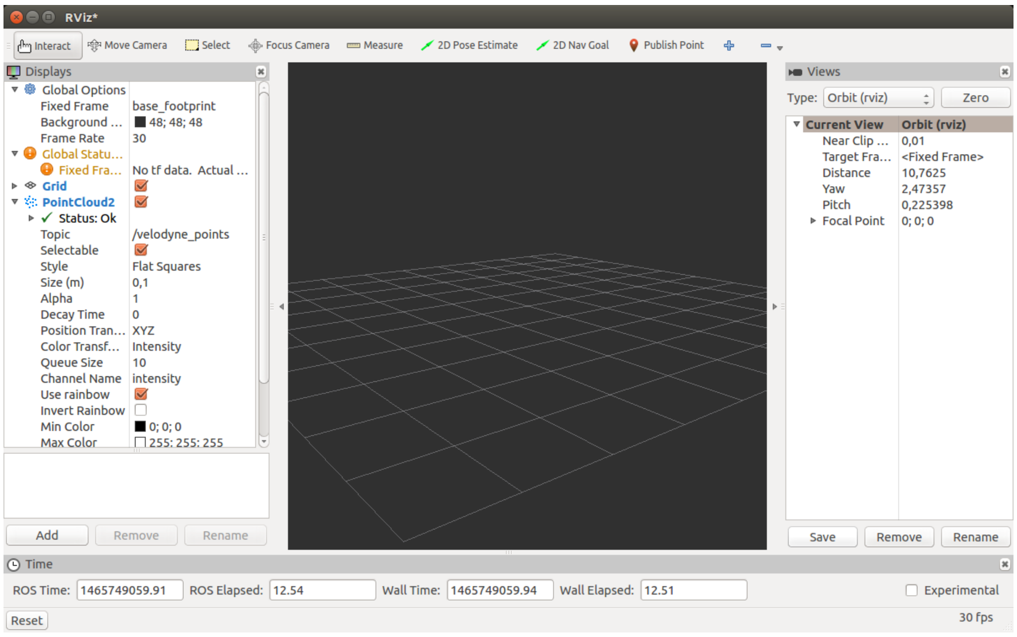
ROS can be used 100% without a GUI. All core functionality and introspection tools are accessible via one of our more than 45 command line tools. There are commands for launching groups of nodes; introspecting topics, services, and actions; recording and playing back data; and a host of other situations. If you prefer to use graphical tools, rviz and rqt provide similar (and extended) functionality.

Tools

One of the strongest features of ROS is the powerful development toolset. These tools support introspecting, debugging, plotting, and visualizing the state of the system being developed. The underlying publish/subscribe mechanism allows you to spontaneously introspect the data flowing through the system, making it easy to comprehend and debug issues as they occur. The ROS tools take advantage of this introspection capability through an extensive collection of graphical and command line utilities that simplify development and debugging.

Rviz

RVIZ can visualize many of the common message types provided in ROS, such as laser scans, three-dimensional point clouds, and camera images. It also uses information from the tf library to show all of the sensor data in a common coordinate frame of your choice, together with a three-dimensional rendering of your robot. Visualizing all of your data in the same application not only looks impressive, but also allows seeing quickly what your robot sees, and identify problems such as sensor misalignments or robot model inaccuracies.



Global Options	
Fixed Frame	base_footprint
Background ...	■ 48; 48; 48
Frame Rate	30
Global Statu...	
Fixed Fra...	No tf data. Actual ...
Grid	✓
PointCloud2	✓
Status: Ok	
Topic	/velodyne_points
Selectable	✓
Style	Flat Squares
Size (m)	0,1
Alpha	1
Decay Time	0
Position Tran...	XYZ
Color Transf...	Intensity
Queue Size	10
Channel Name	intensity
Use rainbow	✓
Invert Rainbow	☐
Min Color	■ 0; 0; 0
Max Color	☐ 255; 255; 255

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip ...	0,01
Target Fra...	<Fixed Frame>
Distance	10,7625
Yaw	2,47357
Pitch	0,225398
Focal Point	0; 0; 0

ROS Time: 1465749059.91 ROS Elapsed: 12.54 Wall Time: 1465749059.94 Wall Elapsed: 12.51

Figure 22: Rviz

Rqt

ROS provides RQT, a Qt-based framework for developing graphical interfaces for your robot. You can create custom interfaces by composing and configuring the extensive library of built-in RQT plugins into tabbed, split-screen, and other layouts. You can also introduce new interface components by writing your own RQT plugins.

The rqt_graphplugin provides introspection and visualization of a live ROS system, showing nodes and the connections between them, and allowing you to easily debug and understand your running system and how it is structured.

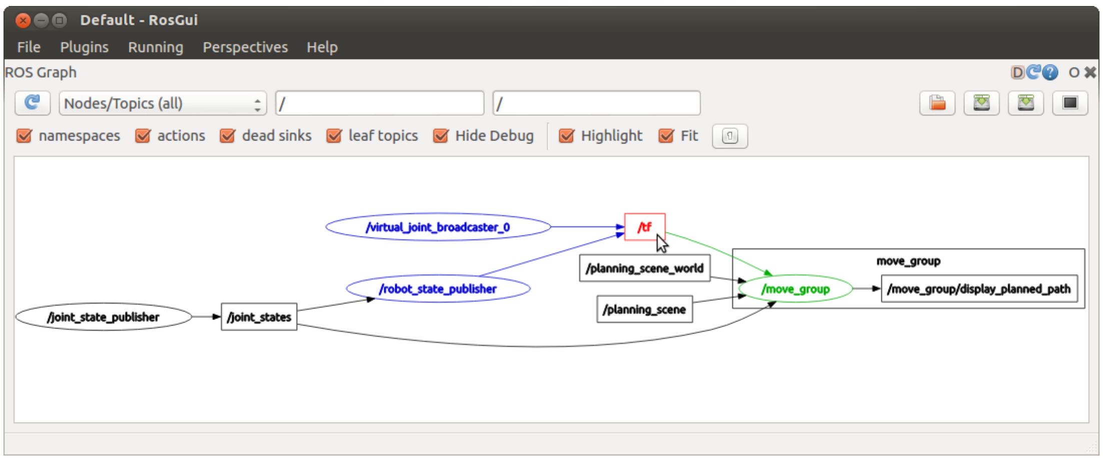


Figure 23: rqt_graph

Robot specific features

ROS provides common robot-specific libraries and tools that will get your robot up and running quickly. Here are just a few of the robot-specific capabilities that ROS provides:

- Standard Message Definitions for Robots
- Robot Geometry Library
- Robot Description Language
- Preemptable Remote Procedure Calls
- Diagnostics
- Pose Estimation
- Localization
- Mapping
- Navigation

Standard robot messages

There are messages definitions for geometric concepts like poses, transforms, and vectors; for sensors like cameras, IMUs and lasers; and for navigation data like odometry, paths, and maps; among many others. By using these standard messages in your application, your code will interoperate seamlessly with the rest of the ROS ecosystem, from development tools to capabilities libraries.

Robot geometry library

Designed with efficiency in mind, the tf library has been used to manage coordinate transform data for robots with more than one hundred degrees of freedom and updates rates of hundreds of Hertz. The tf library allows defining both static transforms, such as a camera that is fixed to a mobile base, and dynamic transforms, such as a joint in a robot arm. You can transform sensor data between any pair of coordinate frames in the system. The tf library handles the fact that the producers and consumers of this information may be distributed across the network, and the fact that the information is updated at varying rates.

Robot description language

ROS provides a set of tools for describing and modeling your robot so that it can be understood by the rest of your ROS system, including tf, robot_state_publisher, and Rviz. The format for describing your robot in ROS is URDF (Unified Robot Description Format), which consists of an XML document in which you describe the physical properties of your robot, from the lengths of limbs and sizes of wheels to the locations of sensors and the visual appearance of each part of the robot.

Preemptable remote procedure calls

While topics (anonymous publish/subscribe) and services (remote procedure calls) cover most of the communication use cases in robotics, sometimes you need to initiate a goal-seeking behavior, monitor its progress, be able to preempt it along the way, and receive notification when it is complete. ROS provides actions for this purpose.

Diagnostics

ROS provides a standard way to produce, collect, and aggregate diagnostics about your robot so that, at a glance, you can quickly see the state of your robot and determine how to address issues as they arise.

Pose estimation, localization and navigation.

There are ROS packages that solve basic robotics problems like pose estimation, localization in a map, building a map, and even mobile navigation.

Integration with other libraries

It can work joinly with other libraries thanks to the interfaces it provides. These interfaces will normally ease the communications among them. The logos of these libraries are shown in Figure 24.

Figure 24: Libraries integrated in ROS.

GAZEBO is a 3D indoor and outdoor multi-robot simulator, which includes dynamic and kinematic physics, and a pluggable physics engine. Integration between ROS and GAZEBO is provided by a set of GAZEBO plugins that support many existing robots and sensors. Because the plugins present the same message interface as the rest of the ROS ecosystem, you can write ROS nodes that are compatible with simulation, logged data, and hardware. You can develop your application in simulation and then deploy to the physical robot with little or no changes in your code.

OpenCV is the premier computer vision library, used in academia and in products around the world. OpenCV provides many common computer vision algorithms and utilities that you can use and build upon. ROS provides tight integration with OpenCV, allowing users to easily feed data published by cameras of various types into OpenCV algorithms, such as segmentation and tracking. ROS builds on OpenCV to provide libraries such as image_pipeline, which can be used for camera calibration, monocular image processing, stereo image processing, and depth image processing. If your robot has cameras connected through USB, Firewire, or Ethernet, ROS and OpenCV will make your life easier.

PCL, the Point Cloud Library, is a perception library focused on the manipulation and processing of three-dimensional data and depth images. PCL provides many point cloud algorithms, including filtering, feature detection, registration, kd-trees, octrees, sample consensus, and more. If you are working with a three-dimensional sensor like the Microsoft Kinect or a scanning laser, then PCL and ROS will help you collect, transform, process, visualize, and act upon that rich 3D data.

MoveIt! is a motion planning library that offers efficient, well-tested implementations of state of the art planning algorithms that have been used on a wide variety of robots, from simple wheeled platforms to walking humanoids. Whether you want to apply an existing algorithm or develop your own approach, MoveIt! has what you need for motion planning. Through its integration with ROS, MoveIt! can be used with any ROS-supported robot. Planning data can be visualized with rviz and rqt plugins, and plans can be executed via the ROS control system.

Point cloud library

The Point Cloud Library (PCL) is a large scale open project for 2D image and 3D point cloud processing. The PCL framework contains numerous state of the art algorithms for feature estimation, surface reconstruction, registration, model fitting and segmentation.

The development of the Point Cloud Library started in March 2010 at Willow Garage. PCL is multi-platform and has been successfully compiled and deployed on Linux, MacOS, Windows and Android/iOS.

PCL has been designed as a platform formed with several code libraries which can be compiled alone. This characteristic makes PCL more suitable to be used on devices which computational capability is reduced.

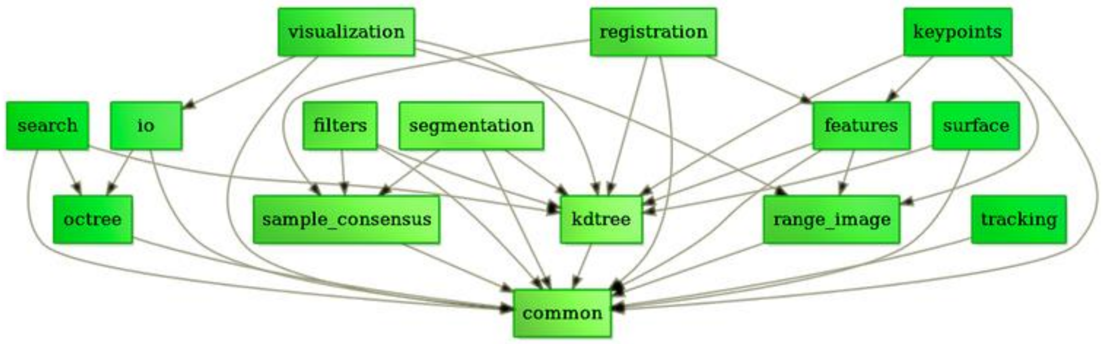


Figure 25: PCL functional parts.

Because PCL is split into a list of code libraries, the list of dependencies differs based on what you need to compile. The difference between mandatory and optional dependencies is that a mandatory dependency is required in order to a particular PCL library compiles a function, while an optional dependency disables certain functionality within a PCL library but compiles the rest of the library that does not require the dependency.

Library	Minimum version
Boost	1.47 (OpenNI)
Eigen	3.0
FLANN	1.7.1
VTK	5.6

Table 2: Mandatory dependencies

Library	Minimum version
Qhull	2011.1
OpenNI	1.3
CUDA	4.0

Table 3: Optional dependencies

Figure 26: Boost, Eigen, FLANN and VTK logos.

Figure 27: Qhull, OpenNI and CUDA logos.

PCL is released under the terms of the BSD license and is open source software. It is free for commercial and research use.

Point types

The smallest and basic type of data that it can be found in Point cloud library are three dimensional points. They are usually grouped in point clouds which contain a set of them but they can also be found as singular points.

What we all know today as PCL, was born as a library developed within ROS. The consensus then was that a Point Cloud is a complicated n-D structure that needs to be able to represent different types of information. That is the reason why there is a big amount of Point types available in PCL.

The most used point types are introduced hereafter:

PointXY – float x,y;

It is a simple two dimensional point structure:

```
struct
{
  float x;
  float y;
 };
```

PointXYZ – Members: float x, y, z :

This is one of the most used data types, as it represents 3D xyz information only. The 3 floats are padded with an additional float for SSE alignment. The user can either access *points[i].data[0]* or *points[i].x* for accessing to the x coordinate.

```
union
{
  float data[4];
  struct
  {
    float x;
    float y;
    float z;
  };
 };
```

PointXYZI – Members: float x, y, z, intensity

Simple XYZ + intensity point type. In an ideal world, these 4 components would create a single structure, SSE-aligned. However, because the majority of point operations will either set the last component of the *data[4]* array (from the xyz union) to 0 or 1 (for transformations), we cannot make *intensity* a member of the same structure, as its contents will be overwritten. For example, a dot product between two points will set their 4th component to 0, otherwise the dot product doesn't make sense, etc.

Therefore for SSE-alignment, we pad intensity with 3 extra floats. This solution is inefficient in terms of storage, but good in terms of memory alignment.

```
union
{
  float data[4];
  struct
  {
    float x;
    float y;
    float z;
  };
};
union
{
  struct
```

```
      {
        float intensity;
      };
      float data_c[4];
   };
```

PointXYZRGB - float x, y, z, rgb;

Similar to PointXYZI, except rgb represents the RGBA information packed into a float.

```
    union
    {
      float data[4];
      struct
      {
        float x;
        float y;
        float z;
      };
    };
    union
    {
      struct
      {
        float rgb;
      };
      float data_c[4];
     };
```

Modules

As shown before, the Point Cloud Library is composed of different modules, which supply lots of developed functions. We should use these functions if the performance we are looking for is implemented in any of these modules.

These are the modules which form the Point Cloud Library:

- Module Common.
- Module Features
- Module Filters
- Module Geometry
- Module IO
- Module Kdtree
- Module Keypoints
- Module Octree
- Module Outofcore
- Module Recognition
- Module Registration

- Module Sample Consensus
- Module Search
- Module Segmentation
- Module Surface
- Module Visualization

Hereafter, the ones, which are used more often, are briefly described:

Module common

The pcl_common library contains the common data structures and methods used by the majority of PCL libraries. The core data structures include the PointCloud class and a multitude of point types that are used to represent points, surface normals, RGB color values, feature descriptors, etc. It also contains numerous functions for computing distances/norms, means and covariances, angular conversions, geometric transformations, and more.

Module feature

The pcl_features library contains data structures and mechanisms for 3D feature estimation from point cloud data. 3D features are representations at a certain 3D point or position in space, which describe geometrical patterns based on the information available around the point. The data space selected around the query point is usually referred as the k-neighborhood.

In Figure 28 it is shown a simple example of a selected query point, and its selected k-neighborhood.

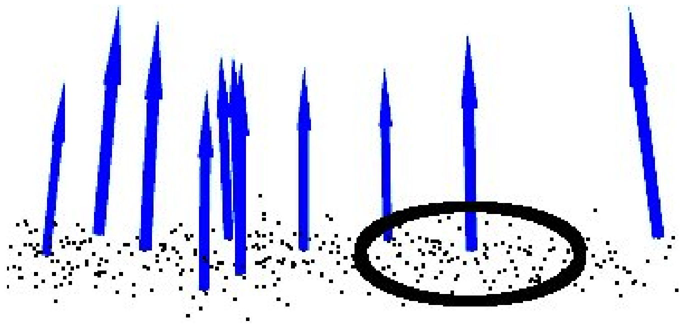


Figure 28: Example - Module features

Module IO

The pcl_io library contains classes and functions for reading and writing point cloud data (PCD) files, as well as capturing point clouds from a variety of sensing devices. This module provides an interface to store in the disk different point clouds through an efficient way.

Module filters

The pcl_filters library contains outlier and noise removal mechanisms for 3D point cloud data filtering applications.

An example of noise removal is presented in Figure 29 due to measurement errors, certain datasets present a large number of shadow points. This complicates the estimation of local point cloud 3D features. Some of these outliers can be filtered by performing a statistical analysis on each point's neighborhood, and trimming those which do not meet a certain criteria. The sparse outlier removal implementation in PCL is based on the computation of the distribution of point to neighbors distances in the input dataset. For each point, the mean distance from it to all its neighbors is computed. By assuming that the resulted distribution is Gaussian with a mean and a standard deviation, all points whose mean distances are outside an interval defined by the global distances mean and standard deviation can be considered as outliers and trimmed from the dataset.

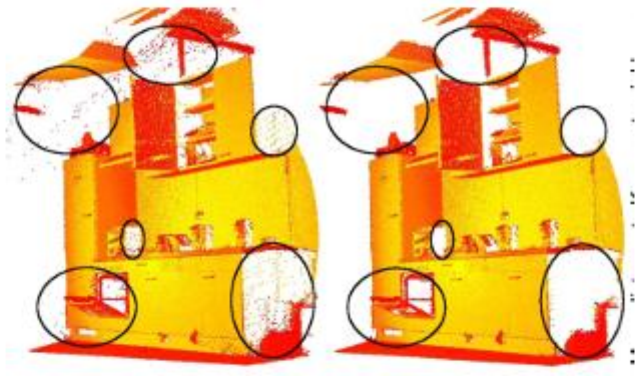


Figure 29: Example - Module filters

Module kd-tree

The pcl_kdtree library provides the kd-tree data-structure, using FLANN, that allows for fast nearest neighbor searches.

A Kd-tree (*k*-dimensional tree) is a space-partitioning data structure that stores a set of k-dimensional points in a tree structure that enables efficient range searches and nearest neighbor searches. Nearest neighbor searches are a core operation when working with point cloud data and can be used to find correspondences between groups of points or feature descriptors or to define the local neighborhood around a point or points.

Module segmentation

The pcl_segmentation library contains algorithms for segmenting a point cloud into distinct clusters. These algorithms are best suited to process a point cloud that is composed of a number of spatially isolated regions. In such cases, clustering is often used to break the cloud down into its constituent parts, which can then be independently processed.

Module visualization

The pcl_visualization library was built with the purpose of being able to ease the visualization. The results of algorithms operating on 3D point cloud data. Similar to OpenCV's highgui routines for displaying 2D images and for drawing basic 2D shapes on screen, the library offers:

- methods for rendering and setting visual properties (colors, point sizes, opacity, etc) for any n-D point cloud datasets in pcl::PointCloud<T> format;
- methods for drawing basic 3D shapes on screen (e.g., cylinders, spheres,lines, polygons, etc) either from sets of points or from parametric equations;
- a histogram visualization module (PCLHistogramVisualizer) for 2D plots;
- a multitude of Geometry and Color handler for pcl::PointCloud<T> datasets;
- a pcl::RangeImage visualization module.

Some examples are shown in Figure 30 and Figure 31.

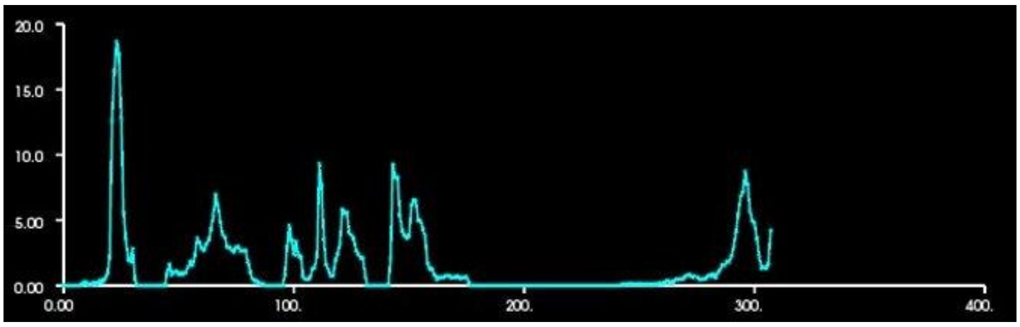


Figure 30: Example - Module visualization (Histogram)

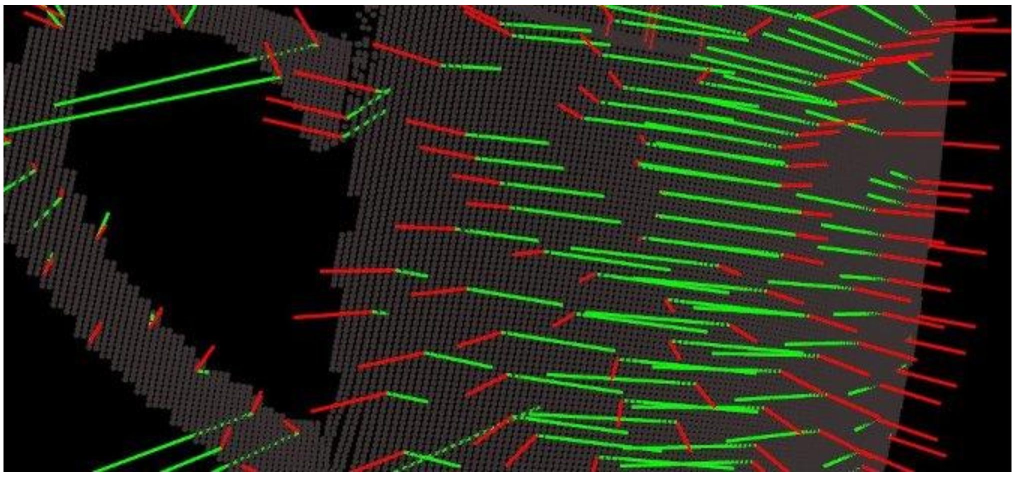


Figure 31: Example - Module visualization (Normal Point Cloud)

6 Construction of a simulation environment in GAZEBO

Gazebo is the platform which has been used to build the environment where all the simulations were done. It offers the advantages of being an independent module regard the one that carries out the estimations, so once simulation results are good the system will not need any change to be exported to real world tests.

Among the different GAZEBO versions that are available to be installed the DRCSim version has been used. It is a version which was released for the DARPA Robot Challenge. It provides several elements which can be directly used or employed to get new ones. Some of these elements are shown hereafter:

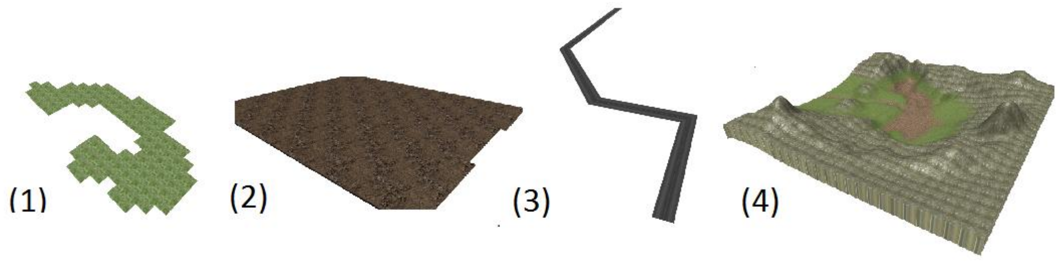


Figure 32: DRCSim Surfaces

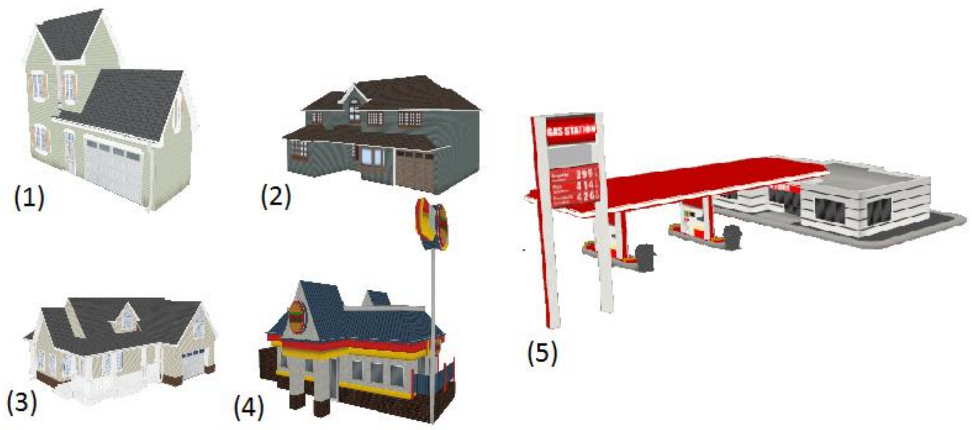


Figure 33: Buildings

In Figure 32 some of the surfaces which can be used are depicted. Number one is grass, two is mud three is road and four is a mountain landscape. Figure 33 shows some of the buildings which are included to be used.

Additionally to the elements presented before we can also include primitive shapes such as spheres, cylinders and parallelepipeds. Their textures can be changed by some of the textures that GAZEBO has, but in this work all the primitive shapes have been employed in gray color (as they are set by default).

Once the elements are placed in the scene, 6 parameters can be configured for each element. These parameters are:

- Pose.x : coordinate x where the object is placed in meters.
- Pose. y: coordinate y where the object is placed in meters.
- Pose z : coordinate z where the object is placed in meters.
- Pose. roll: object roll angle in radians.
- Pose.pitch: object pitch angle in radians.
- Pose.yaw: object yaw angle in radians.

Using some of the elements shown before and modifying the necessary parameters a world was created. This world, which can be seen in Figure 34, contains a mountain landscape with grass as floor and a road. Next to the road different houses where place to make the scene more realistic.

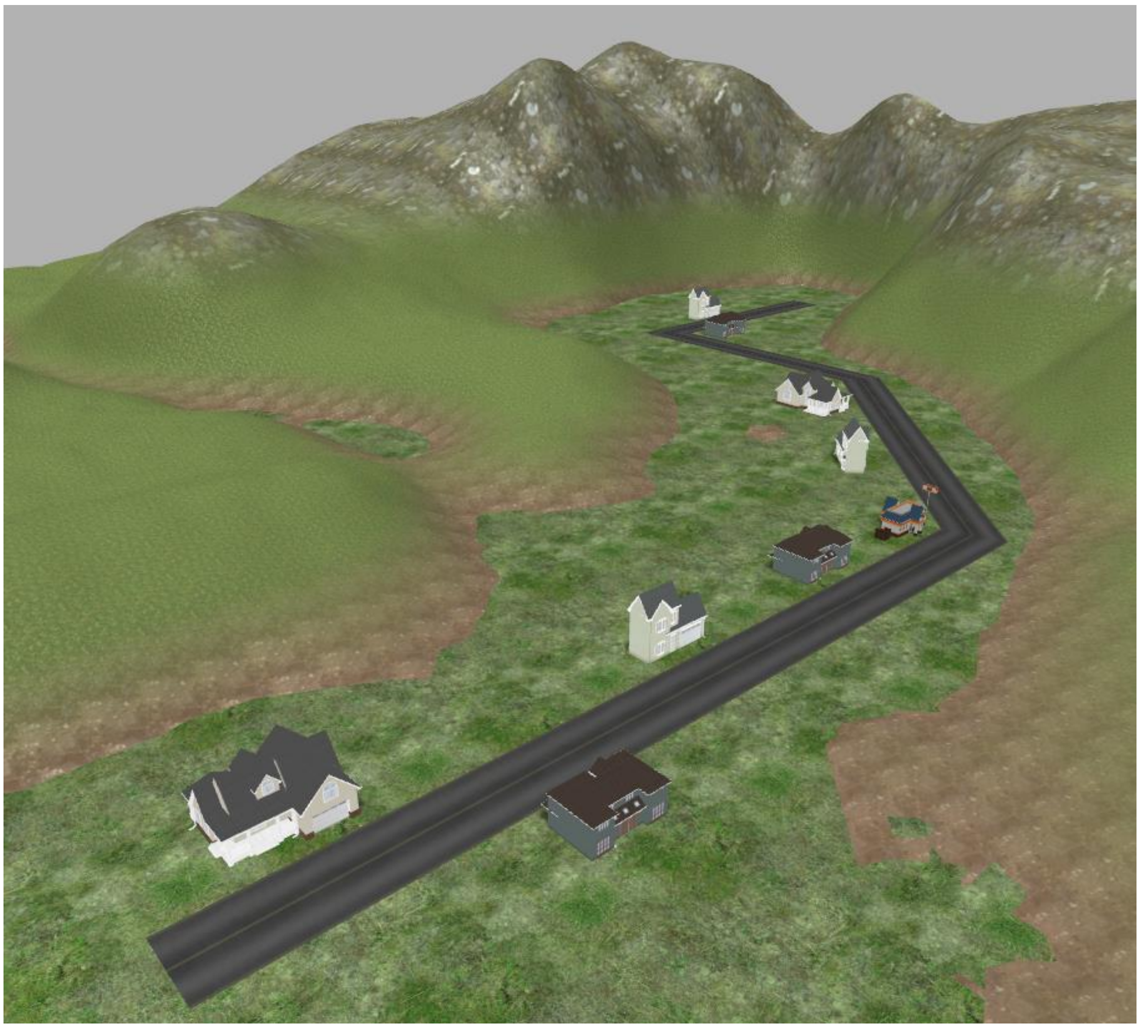


Figure 34: GAZEBO world.

Graphic interface just let you modified that parameters but those are not the only parameters that can be modified when using primitive shapes. One of the most important is their size.

Primitive shapes have two different dimensions for each coordinate, the collision one and the visual one. The first one indicates how big the object is in terms of space occupancy and the second one in terms of appearance.

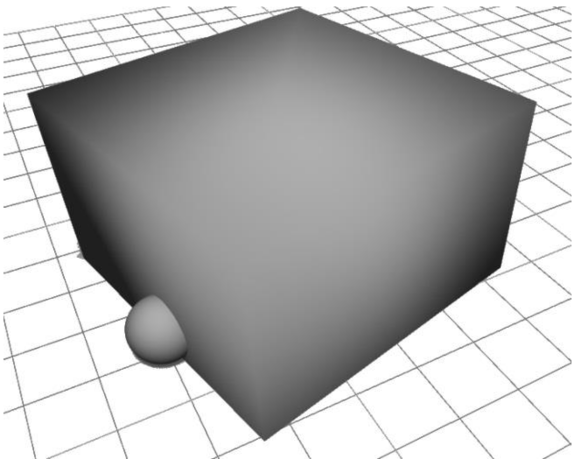


Figure 35: Primitive shapes dimension I.

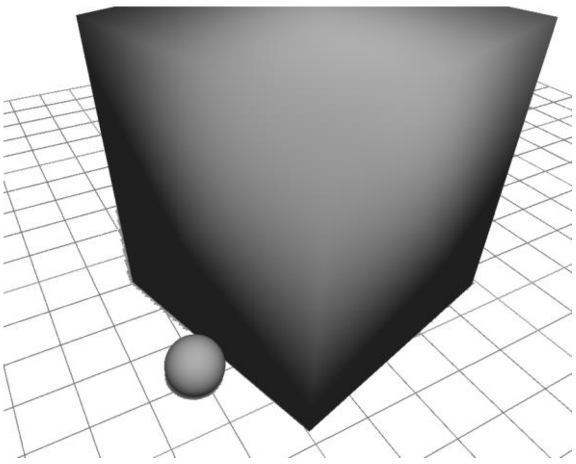


Figure 36: Primitive shapes dimension II.

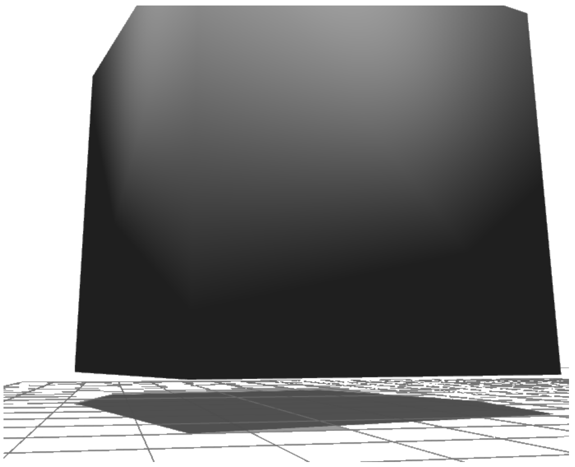


Figure 37: Primitive shapes dimension III.

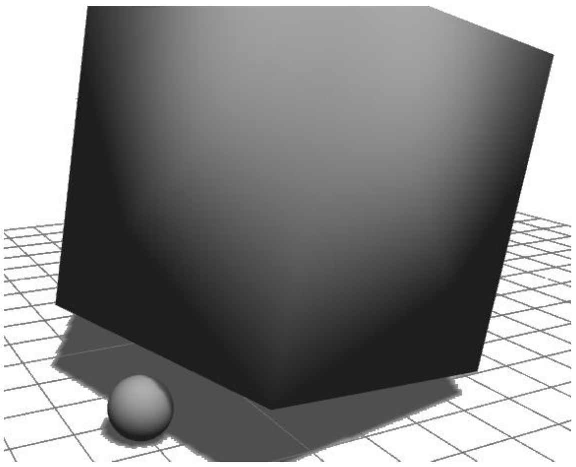


Figure 38: Primitive shapes dimension IV.

In Figure 35 collision dimensions of the square has been set to (1, 1, 1) and the visual ones (5, 5, 5). As it can be seen, it is possible to place the sphere inside the square.

Nevertheless, when collision and visual dimensions are fixed to (5, 5, 5) and we try to put the sphere inside the box it goes out pushing the square in the other direction. This fact can be observed in Figure 36.

In Figure 37 and Figure 38 the effects of setting bigger values for collision than for visual dimensions are shown. It seems that the figure is flying over the ground and if we try to place a sphere in the same point than before it pushes the square up.

Taking advantage of the feature described before, curbs have been added along both sides of the road. These curbs have been added in ten sections, one on the right side, another one on the left of each section and two more at the start and at the end. The length of these curbs depends on the length of each section. Figure 39 shows a picture of the new simulation world once the described curbs have been added. Curbs are added to ease the road border detection.

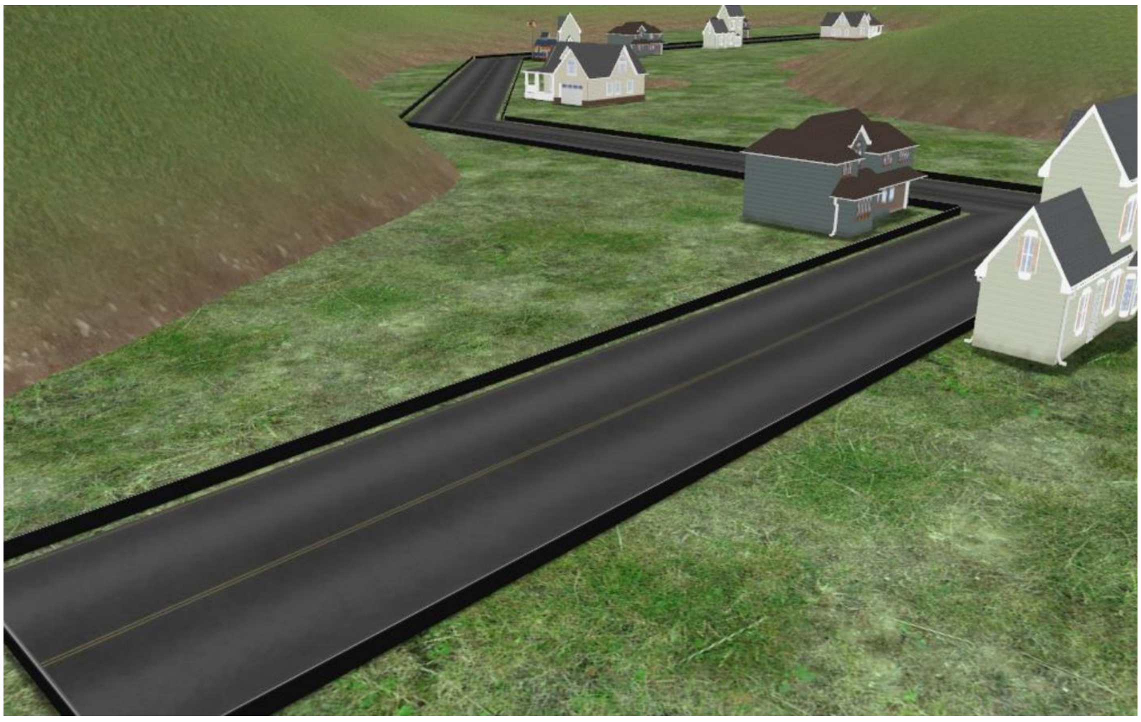


Figure 39: GAZEBO world with curbs

While, as said before, each curb has a different length, their height and width is common for all of them. They are 10 centimeters tall and 40 centimeters wide.

The only difference between the worlds shown in Figure 34 and Figure 39 are the curbs. Both images can seem to be different but that is due to they were captured from opposite sides of the scene.

One of the main problems that GAZEBO has is its high computational costs. It can be checked that the scene moves really fluent when managing a few models, but as the amount of objects placed in the scene grows up it becomes less and less fluent.

At the beginning the simulations could be done properly with a real time factor which was always above the half of second. Once the computational costs of the algorithms which were below the scene went up it was necessary to reduce as much as possible all the elements placed in the scene that only had an aesthetics purpose. With the aim of reducing as much as possible computational costs the mountain landscape and the grass surface were deleted from the scene shown in Figure 39.

The result of the scene simplification explained before, is shown in Figure 40.

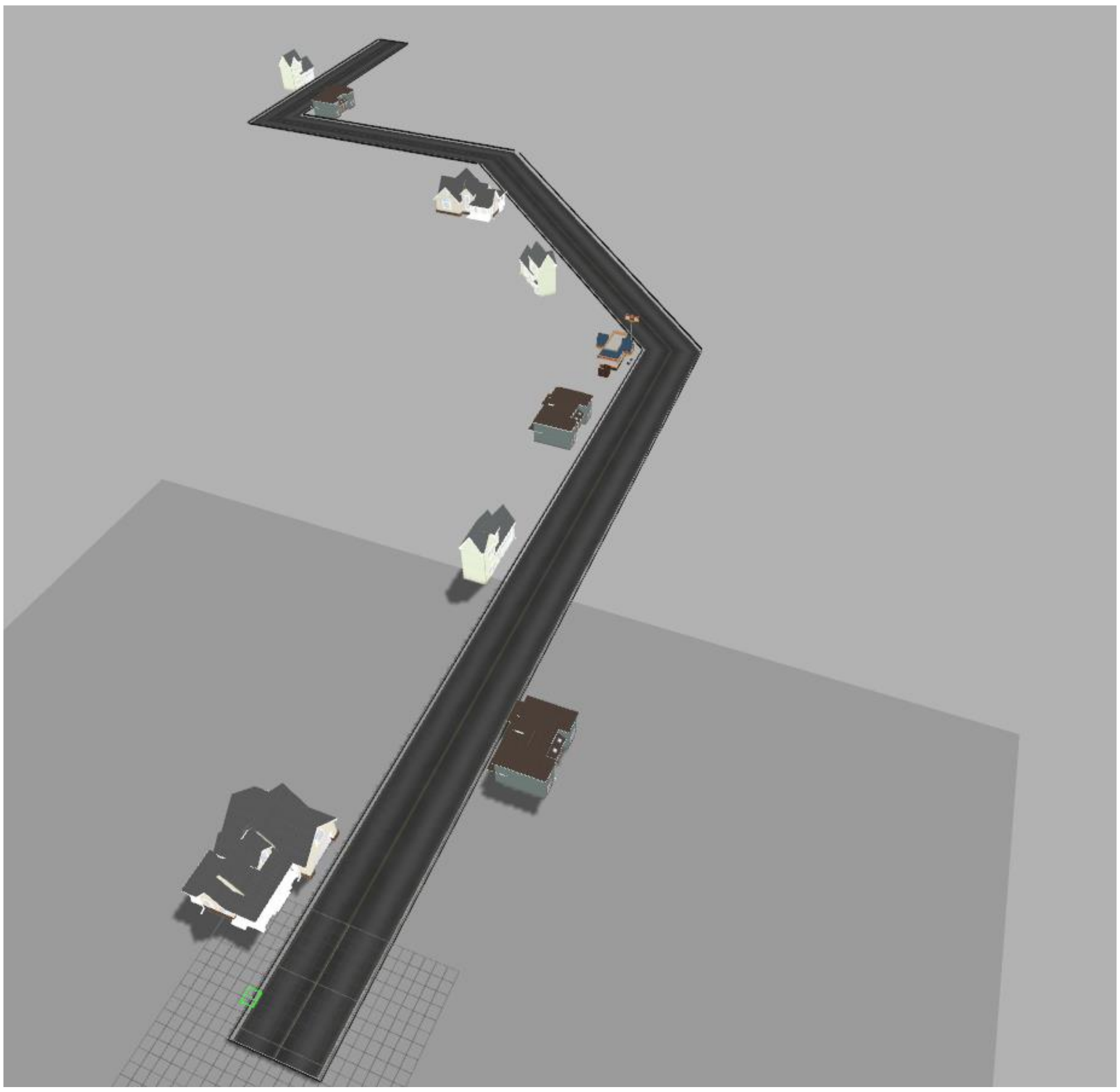


Figure 40: GAZEBO simulation world.

7 GAZEBO MODELS

As said in previous chapters GAZEBO world simulates the real world so it is necessary to build all the sensors and vehicles which are used for the simulation. All the models that should be created has a common point, they need to be accessible from GAZEBO (our real environment) and ROS (where all the necessary estimations, to get the purpose we pursue, will be carried out).

The designed architecture for both elements is a little different. To perform the function we are looking for velodyne models which get information from GAZEBO world and make them accessible for ROS.

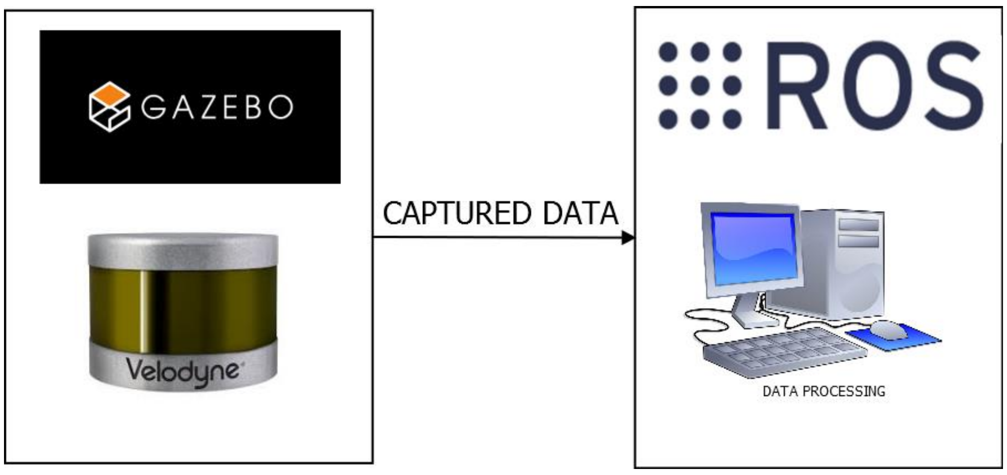


Figure 41: VLP-16 model design.

Vehicle model should establish a bidirectional communication between ROS and GAZEBO. ROS should provide the information which will indicate the model what to do. GAZEBO should supply a feedback sending the data which describes the state of the model in the simulation scene.

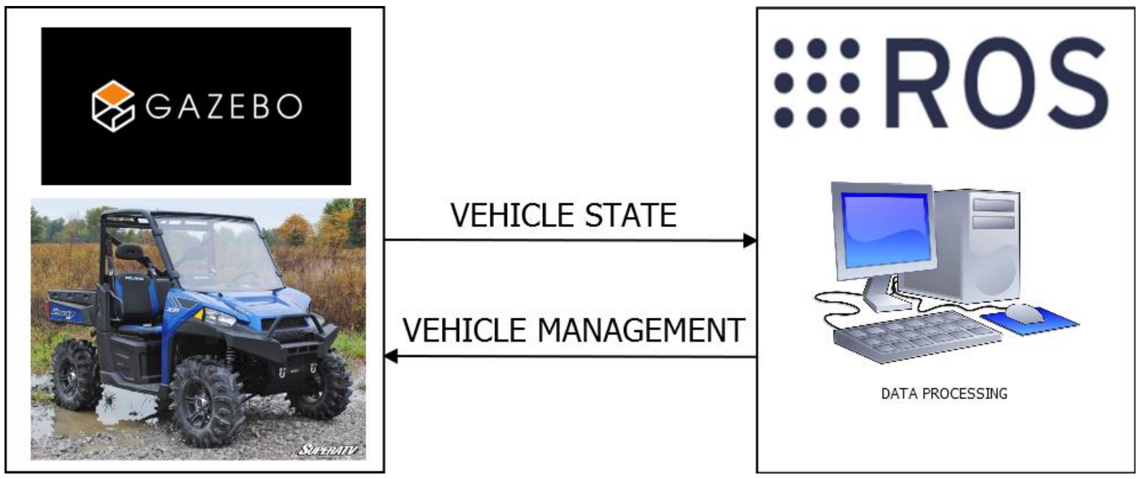


Figure 42: Vehicle model design.

Additionally other sensor has been created to ease the tests. This sensor would not be used in the real application but they were added to the world to ease the test phase. In the last part of this chapter and in future chapters we will explain the need of this sensor.

Velodyne model

The velodyne sensor, which has been used, is the VLP-16. It creates360º 3D images by using 16 laser/detector pairs mounted in a compact housing. The housing rapidly spins to scan the surrounding environment. The lasers fires thousands of times per second, providing a rich 3D point cloud in real time.

The VLP-16 measures the reflectivity of an object with 256-bit resolution independent of laser power and distance over a range from 1 m to 100m. Commercially available reflectivity standards and retro-reflectors are used for the absolute calibration of the reflectivity, which is stored in a calibration table within the FPGA of the VLP-16

Diffuse reflectors reports values from 0-100 for reflectivities from 0% to 100%. Retro-reflectors report values from 101 to 255 with 255 being the reported reflectivity for an ideal retro-reflector and 101-254 being the reported reflectivity for partially obstructed or imperfect retro-reflectors.

VLP-16 specifications

Hereafter, some of the most important specifications are described. These specifications have been taken into account for the creation of the VLP-16 model as similar as the real one.

Sensor specifications

- VLP-16 has 16 channels with the orientations shown in Table 4.

Laser ID	Vertical Angle
0	$-15\,°$
1	$1\,°$
2	$-13\,°$
3	$-3\,°$
4	$-11\,°$
5	$5\,°$
6	$-9\,°$
7	$7\,°$
8	$-7\,°$
9	$9\,°$
10	$-5\,°$
11	$11\,°$
12	$-3\,°$
13	$13\,°$
14	$-1\,°$
15	$15\,°$

Table 4: Laser channels orientation and ID.

- The range measurement of the sensors is from 1meter to 100meters.
- Measurements have an accuracy of ±3 centimeters.
- The vertical field of view is equal to 30 degrees.
- It provides a 2 degree vertical resolution.
- The horizontal field of view is equal to 360 degrees.
- It supplies a horizontal resolution between 0.1 and 0.4 degrees.
- It rotates with a rate between 5 and 20 Hz.

Also physical characteristics were taken into account to build the physical model. The real VLP-16 can be seen in Figure 43 and the created model is shown in Figure 44. Both have the same dimensions but their appearance have small differences.

Figure 43: Real VLP-16.

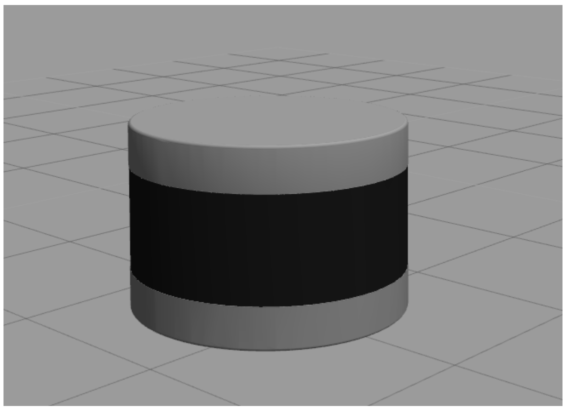


Figure 44: VLP-16 model.

VLP-16 uses a class 1 laser, which is safe for the human's eyes. These laser beams are not visible for us so we cannot see which points it is pointing at. In GAZEBO it is possible to make these beams visible as shown in Figure 45. The area painted in blue is the representation of the different laser beams.

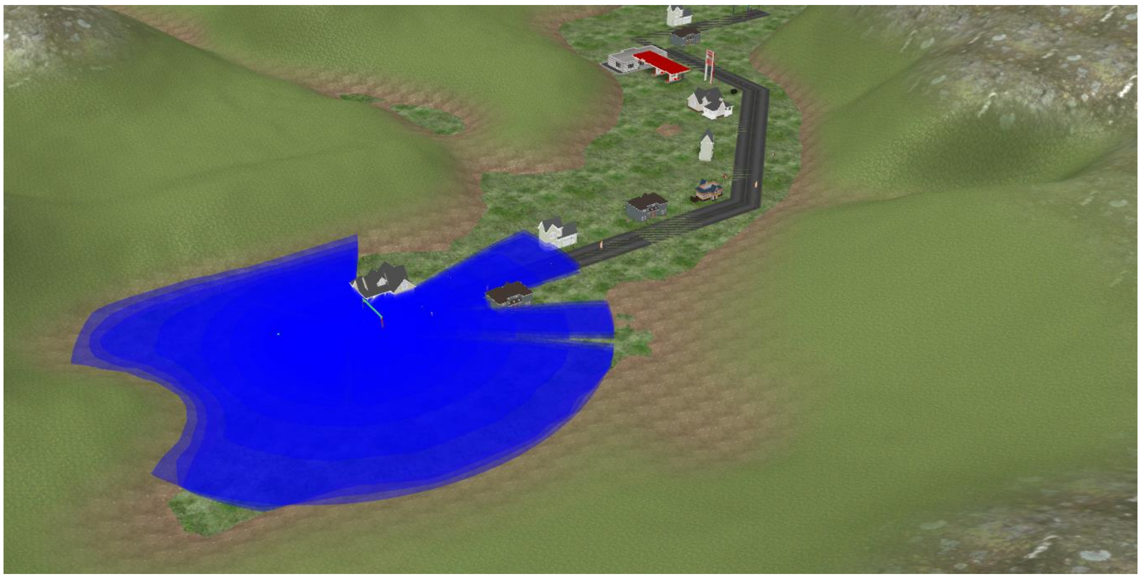


Figure 45: Laser beams of VLP-16 model I.

In Figure 46 it is depicted an image which shows the same environment as the presented before but from another point of view. This time it is easier to distinguish the different beams emitted by the velodyne sensor.

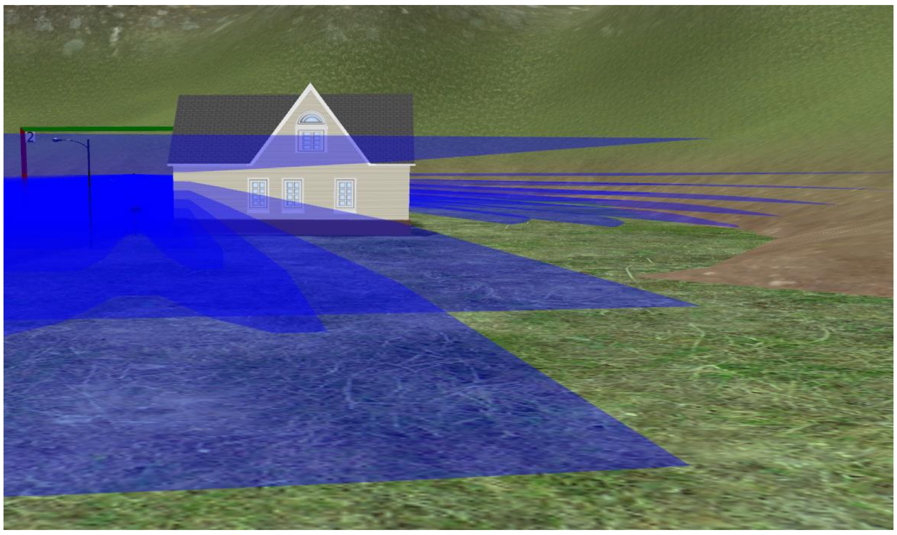


Figure 46: Laser beams of VLP-16 model I.

Unfortunately the laser just returns the distance among the sensor and the different objects that are in the environment. To build the 3D point cloud from these distances several computations must be carried out.

The first step that should be done to estimate the coordinates is to know the angles $\omega \; and \; \alpha$ which are represented in Figure 47. After that, the coordinates can be easily obtained by applying the next equations:

$$Coordinate\ X = Distance \cdot \cos(\omega) \cdot \sin(\alpha)$$

$$Coordinate\ Y = Distance \cdot \cos(\omega) \cdot \cos(\alpha)$$

$$Coordinate\ Z = Distance \cdot \sin(\omega)$$

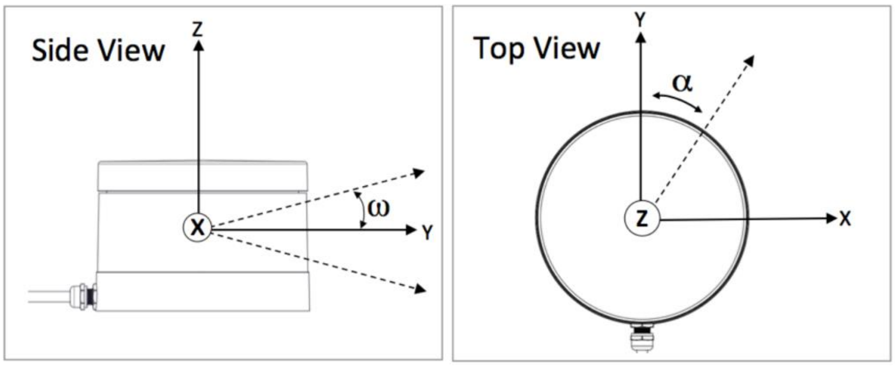


Figure 47: VLP-16 angles.

Once the coordinates are calculated, all the different points can be stored in a point cloud which should be publish using the ROS communication system based on topics. This way it will be accessible from the node which implements the point cloud processing. The type of point cloud used to publish all the three dimensional information captured by the VLP-16 is:

$$sensor\_msgs :: PointCloud2ConstPtr\&\ msg$$

Rviz allows subscribing to topics to check the proper working of the plugging that simulates the VLP-16. In Figure 48 it is depicted the world at the time the point cloud shown in Figure 49 is captured.

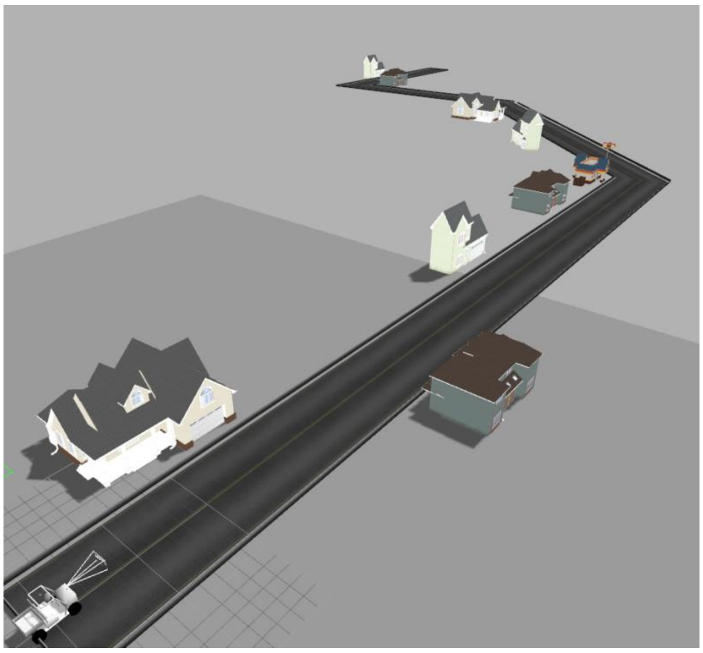


Figure 48: World at time X

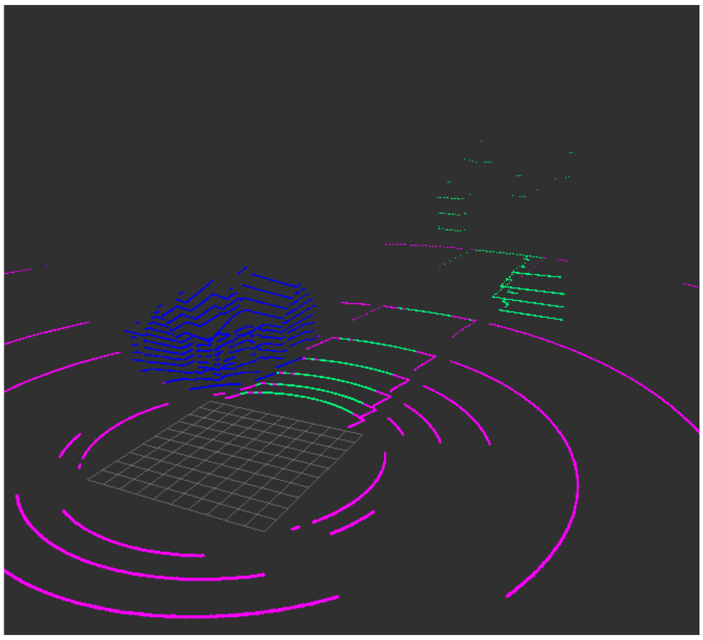


Figure 49: Point cloud captured at time X

Vehicle model

DRCSim includes the model of different vehicles, such as Polaris Ranger EV and Polaris Ranger XP900. The first one was chosen to be used in this work due to its dimensions.

Despite the fact that this vehicle was already ready to be used in GAZEBO it was necessary to incorporate several sensors to it. This addition was made necessary to convert the vehicle from SDF to URDF format, which is the format that ROS uses.

In Figure 50 the Polaris Ranger EV in SDF format is shown. Next to it, in Figure 51 it is depicted the URDF format of the same vehicle once the different sensors have been added .

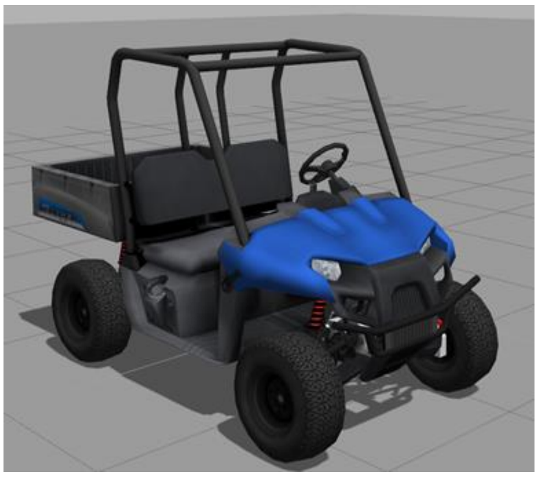


Figure 50: Polaris EV (SDF)

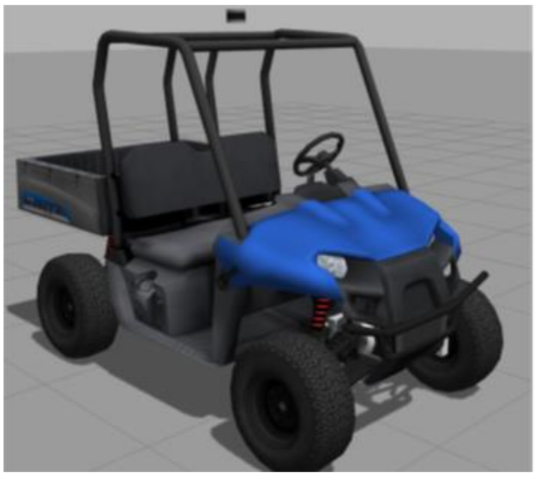


Figure 51: Polaris with sensors (URDF)

The URDF vehicle incorporates a GPS simulator which publishes continuously the position where the Polaris Ranger EV is. It also enables lots of services to act upon the different actuators.

These services are implemented as publishers and subscribers. Hereafter, a brief explanation of their use is provided:

- /Polaris_ranger_ev/brake_pedal/cmd: allows the actuation over the brake pedal.
- /Polaris_ranger_ev/direction/cmd: allows the actuation over the gearbox.
- /Polaris_ranger_ev/gas_pedal/cmd: allows the actuation over the accelerator.
- /Polaris_ranger_ev/hand_brake/cmd: allows the actuation over the hand brake.
- /Polaris_ranger_ev/hand_wheel/cmd: allows the actuation over the steering wheel.
- /Polaris_ranger_ev/key /cmd: allow to switch on and off the vehicle´s engine.
- /Polaris_ranger_ev/brake_pedal/state: informs about the brake pedal state.
- /Polaris_ranger_ev/direction/ state: informs about the gearbox state.
- /Polaris_ranger_ev/gas_pedal/ state: informs about the accelerator state.
- /Polaris_ranger_ev/hand_brake/ state: informs about the hand brake state.
- /Polaris_ranger_ev/hand_wheel/ state: informs about the hand wheel state
- /Polaris_ranger_ev/key / state: informs about the engine state.

The variables presented before have constraints, so if anyone tries to set a value out of range, it will be set to the closer valid value instead. It is important to know the limits of these ranges to program the management of this vehicle to achieve the behavior we were looking for. The possible values for all the variables are:

- brake_pedal: it can take values between 0 and 1.
- direction: it is a discrete variable. -1, 0 and 1 are the only valid values.
- gas_pedal: it can have values between 0 and 1.
- hand_brake: it can take values between 0 and 1.
- hand_wheel: it can have values between -3.14 and 3.14.
- key: it is a discrete variable. 0 and 1 are the only valid values.

On board vehicle camera

To ease the management of the vehicle and provide a first person view, a camera has been added to the vehicle. Thanks to this camera awe can have a simultaneous view of the vehicle in first and third person as shown in Figure 52 and Figure 53.

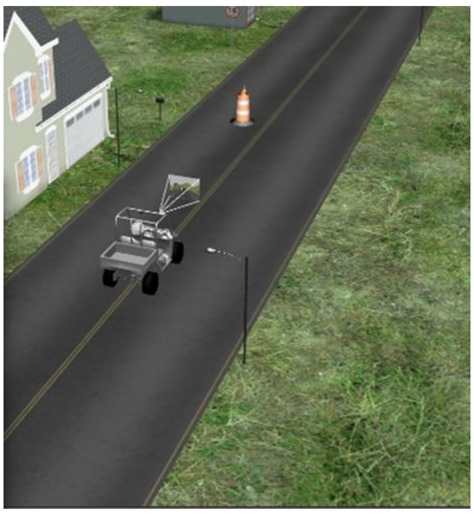


Figure 52: Vehicle third person view.

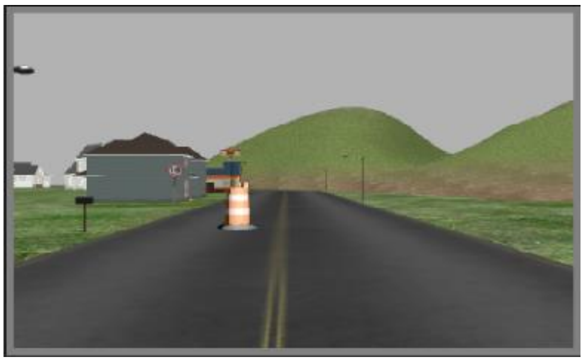


Figure 53: Vehicle first person view

8 Vehicle management

Once all the necessary elements to simulate the real world are created and joined together in the same world it’s time to work on their interfaces. Low level interfaces were already created but they are not optimized to ease the interaction, so it may take a long time to work with them.

In this chapter it is described the work done to implement the nodes that have been created with the aim of easing the vehicle management in GAZEBO. They allow managing the vehicle employing an interface which is more intuitive for humans instead of typing command in the terminal.

Two different systems have been developed to manage the vehicle. One of them has an open loop architecture due to the vehicle´s actuation depends on the inputs provided and the previous state of the vehicle.

The architecture of the other system is based on a close loop system, the vehicle´s actuation is independent of the previous state. It acts depending on the provided inputs, ignoring the rest of the factors.

Vehicle management with keyboard

A computer keyboard is the element that has been chosen as interface for the open loop vehicle control system. Through different key stokes the vehicles behavior can be modified.

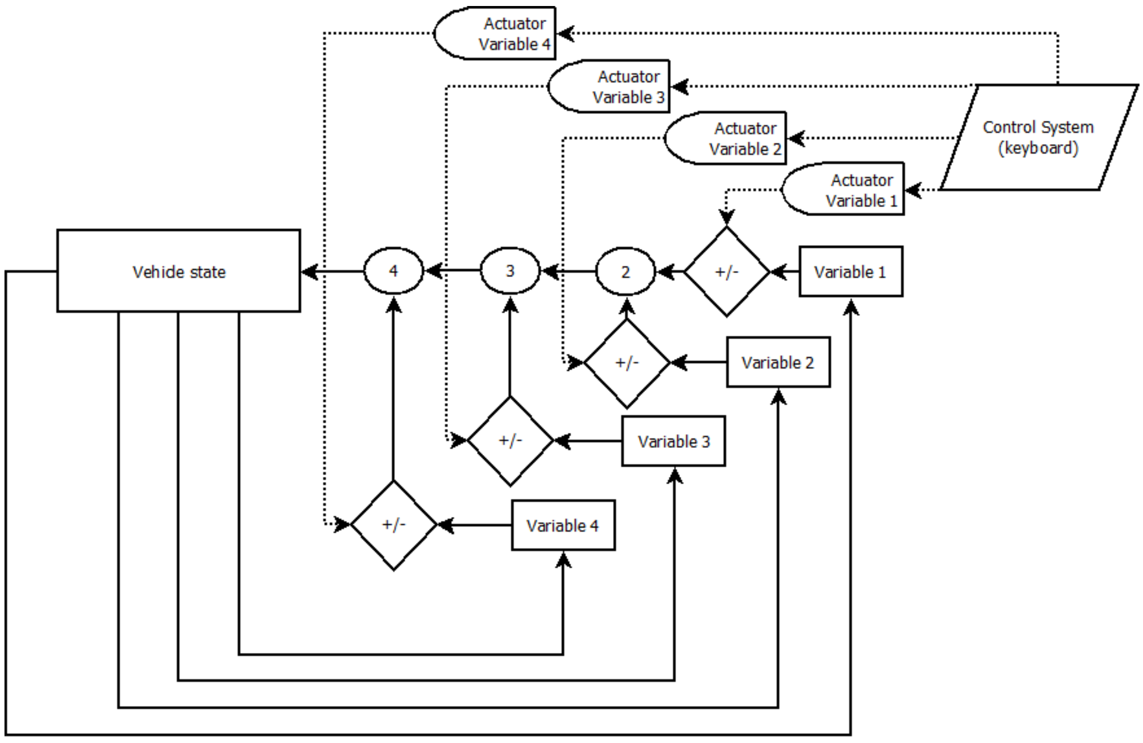


Figure 54: Vehicle management with keyboard diagram.

In Figure 54 the working diagram of the system is shown. Hereafter the keys which can modify the different actuators and the effect they produce are shown:

- d key: changes direction variable from -1 to 1 or from 1 to -1.
- b key: changes hand_brake variable from 1 to 0.
- Up arrow: increases gas_pedal variable 0.02.
- Down arrow: decreases gas_pedal variable 0.02.
- Left arrow: increases hand_wheel variable 0.1.
- Right arrow: decreases hand_wheel variable 0.1.

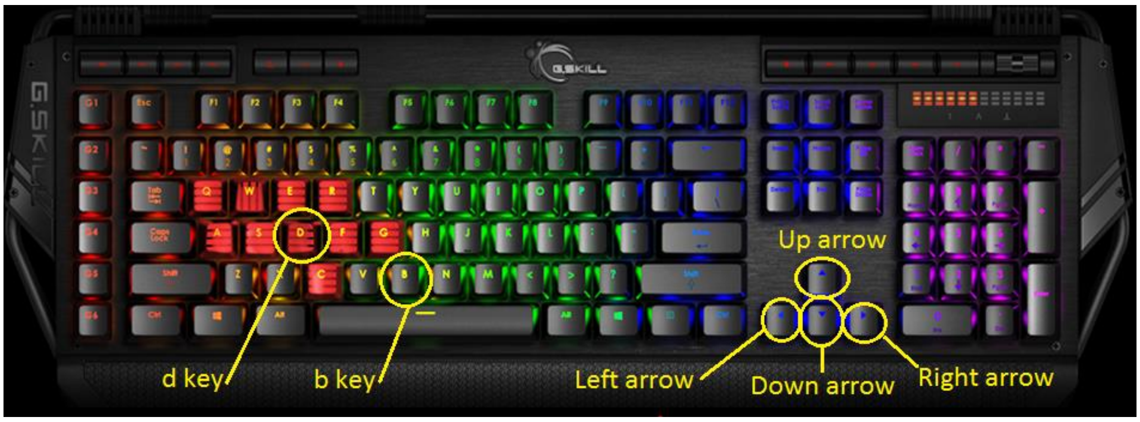
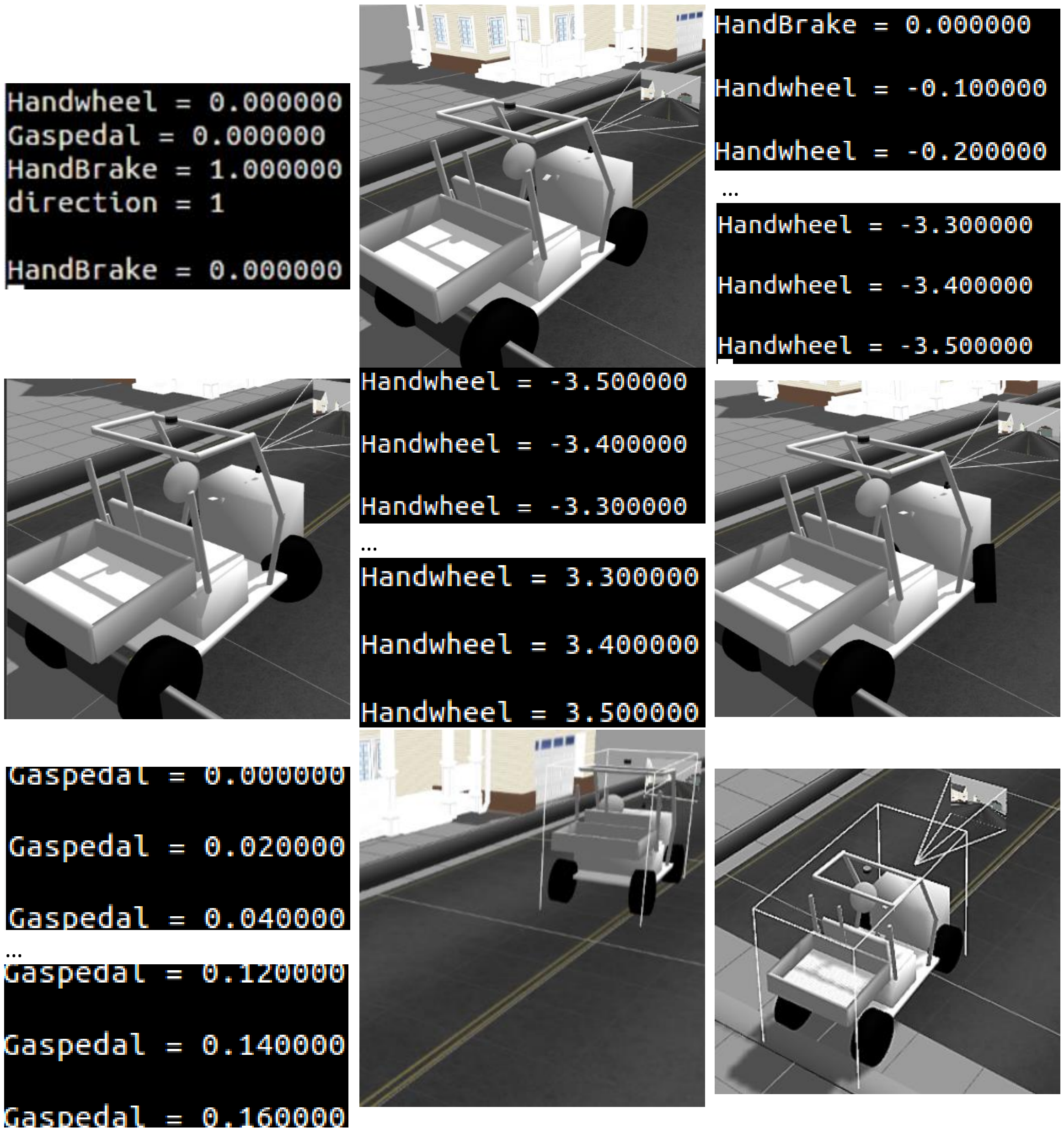


Figure 55: Keys to control the vehicle

Henceforward an example of how the systems works is shown through different images and terminal captures. Once the program starts the value of the main control variables are printed on the terminal.

After that b key is pressed to disable the hand brake as it can be seen in the first subfigure of Figure 56. In the next one, the vehicle is shown at that time. Then the right arrow is pressed till hand_wheel value is equal to -3.5 turning the wheels to the right. Once the wheels are pointing to the right side of the vehicle left arrow is pressed until the hand_wheel variable is equal to 3.5, at that point the wheels are turned to the left.

Finally, the gas pedal and the gear are tested. To speed up the vehicle the up arrow is pressed, as it can be seen in Figure 56 it moves forward its front side until d key is pressed. Then it goes back until the down arrow is pressed the necessary number of times to make the speed equal to zero.



```
Handwheel = 0.000000
Gaspedal = 0.000000
HandBrake = 1.000000
direction = 1

HandBrake = 0.000000
```

```
HandBrake = 0.000000

Handwheel = -0.100000

Handwheel = -0.200000
…
Handwheel = -3.300000

Handwheel = -3.400000

Handwheel = -3.500000
```

```
Handwheel = -3.500000

Handwheel = -3.400000

Handwheel = -3.300000
…
Handwheel = 3.300000

Handwheel = 3.400000

Handwheel = 3.500000
```

```
Gaspedal = 0.000000

Gaspedal = 0.020000

Gaspedal = 0.040000
…
Gaspedal = 0.120000

Gaspedal = 0.140000

Gaspedal = 0.160000
```

Figure 56: Keyboard management working example

Vehicle management with steering wheel

This time a steering wheel and a pair of pedals are the interface between the human and the vehicle. They provide a more intuitive interface than the one before due to its higher likeness to real cars.

This system is implemented as a close loop architecture system. The only factors which takes into account to behave in a way or in another one is just the steering wheel and the pedals. The Figure 57 shows the diagram of this system. Regard the control system which was presented in the previous version, this one offer a better performance in terms of agility and control due to it allows quick movements while keeping a high degree of accuracy.

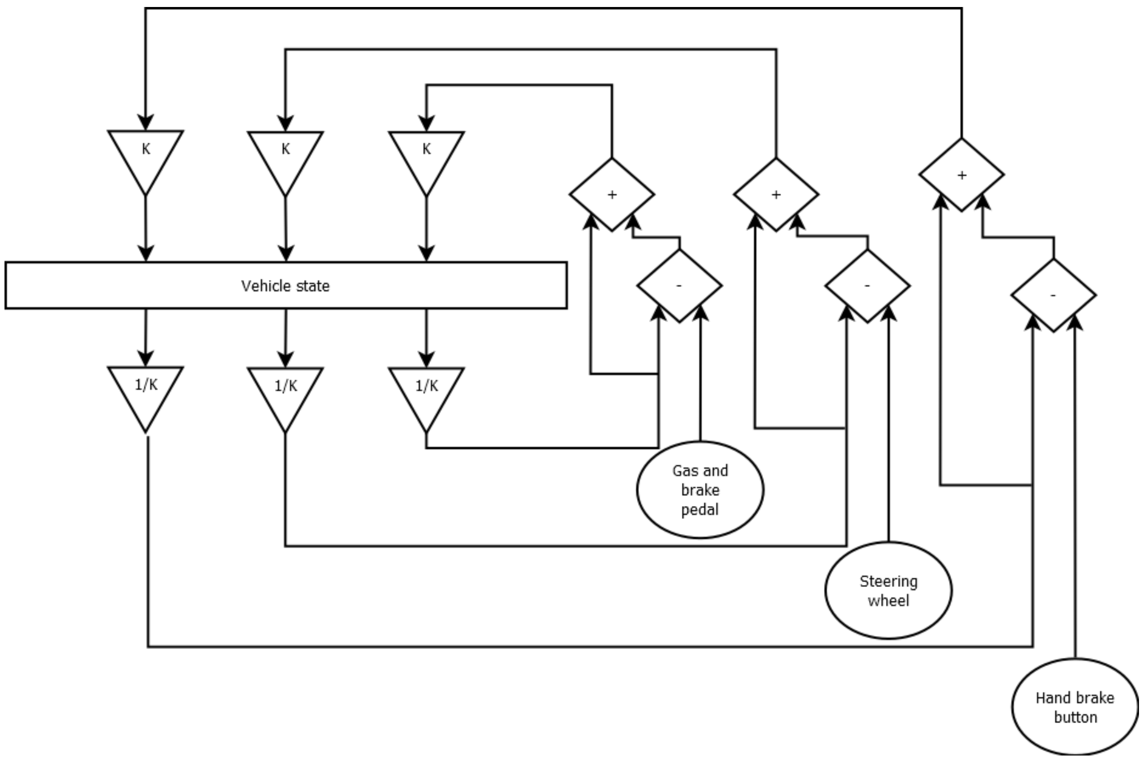


Figure 57: Vehicle management with steering wheel.

In Figure 58 the necessary instructions, which should be taking into account, to manage the car using the steering wheel system are shown.

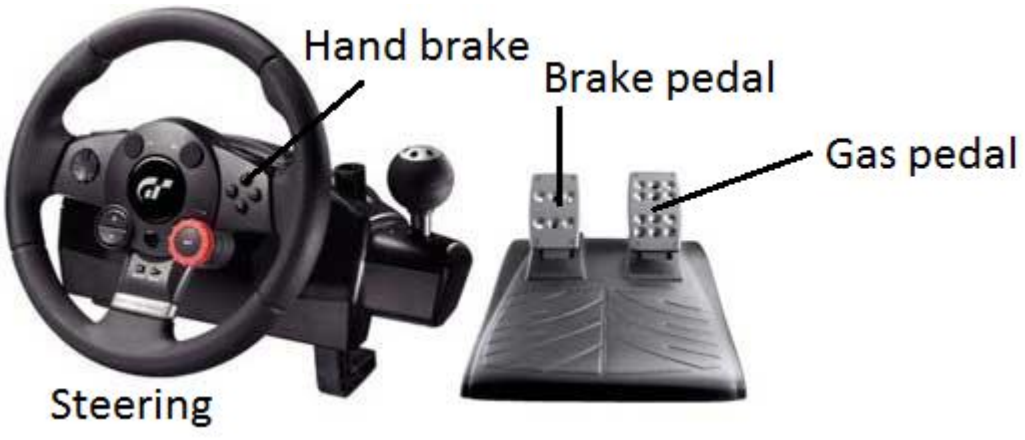


Figure 58: Steering management controls

9 Vehicle guidance

In this chapter the smart part of the system will be described. It processes the inputs captures by the different sensors to return the point that should be the target for the vehicle to ensure its advance and integrity and the direction consign.

It has been implemented in a ROS node which constantly communicates with GAZEBO through ROS communication via topics to receive the data that the different sensors have captured and send the calculated consigns for the system: the target point and direction.

Getting the PCL

Firstly a subscriber has been added to the node. It has been set as a non-buffer subscriber, so it will attend to the messages only when the callback function has finished its process. This node subscribes to the topic where the point cloud captured by the VLP-16 model is being published.

After getting the point cloud it should be converted to a point cloud library format, to ease its process taking advantage of the thousands of functions which are already implemented in that library.

PCL preprocessing

Once the point cloud has been converted to a pointxyz format, a pass-through filter is employed to define the region of interest where further algorithms will carry out the necessary estimations.

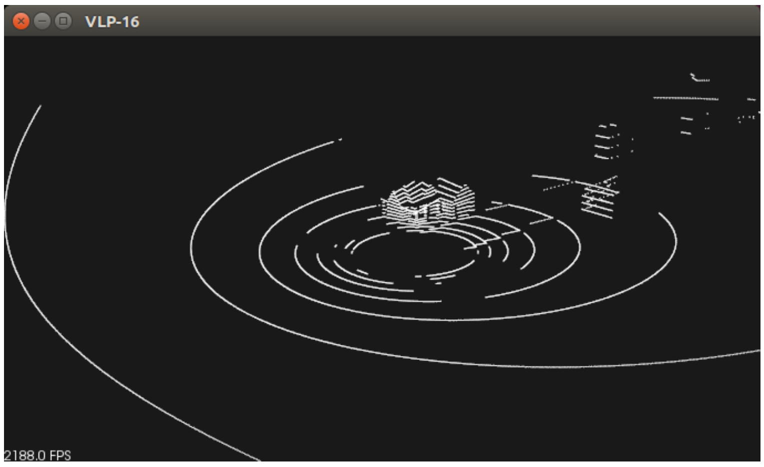


Figure 59: VLP-16 world capture.

In Figure 59 the point cloud captured by the velodyne is depicted, after that all the points which have an X coordinate which value is bigger than 20 or smaller than -10 are removed from the cloud. Vehicles speeds are normally higher when they are going forward, that is the reason why it was considered that this choice is better than a symmetric one.

The result of this elimination can be observed in Figure 60. Next steps consist on carrying out the same steps that were done before, but this time with Y and Z coordinates.

The thresholds, which were set for the Y coordinate, were -8 and 8. These values should be wide enough to ensure that the curbs are not removed if the vehicle is not in the middle of the road.

The point cloud which results from filtering the original cloud through an X and a Y pass-through filter is shown in Figure 61.

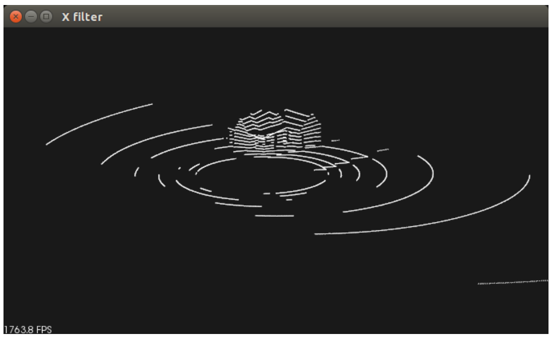


Figure 60: VLP-16 world capture filtered in X.

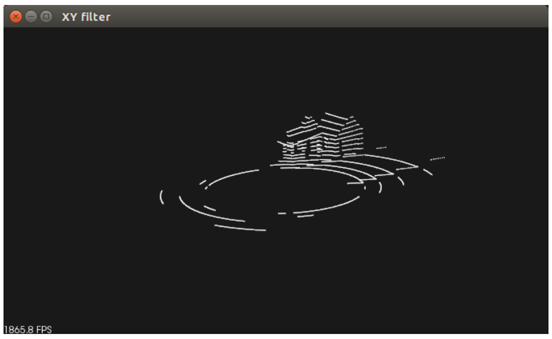


Figure 61: VLP-16 world capture filtered in Y.

Once more the same process is done, but this time focusing on Z coordinate. The point cloud which is obtained this time is depicted in Figure 62.

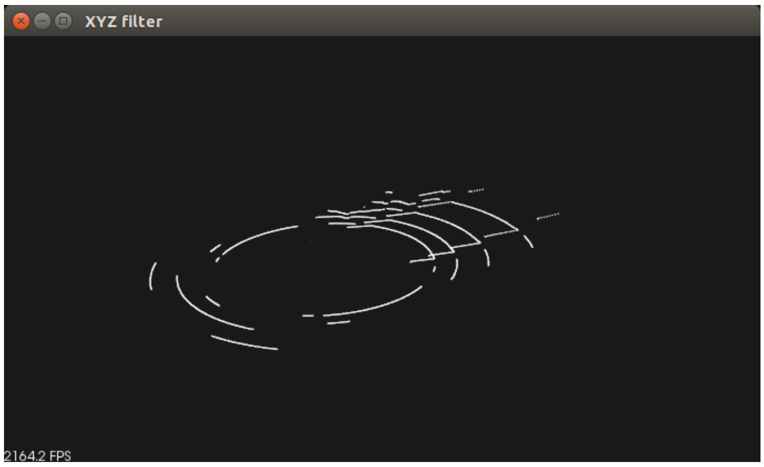


Figure 62: VLP-16 world capture filtered in Z.

PCL downsampling.

After applying the filtering position the resultant point cloud is processed by a Voxelgrid filter. A Voxelgrid filter, which is a Uniform Sampling algorithm, is used to downsample the point cloud. The aim of using a downsampling filter is to get a descriptive point cloud which has a significant lower size than the input one because it will ease the computational requirements of latter steps. The filter builds a 3D Voxelgrid over the point cloud and estimates the centroid of all the points that belong to the same cell, substituting them by the calculated centroid.

The only parameter to be configured in the filter is the size of each cell of the grid. On the one hand this value must be big because it will return a smaller point cloud but, on the other hand the pointcloud must be big enough to avoid harming later algorithms efficacy.

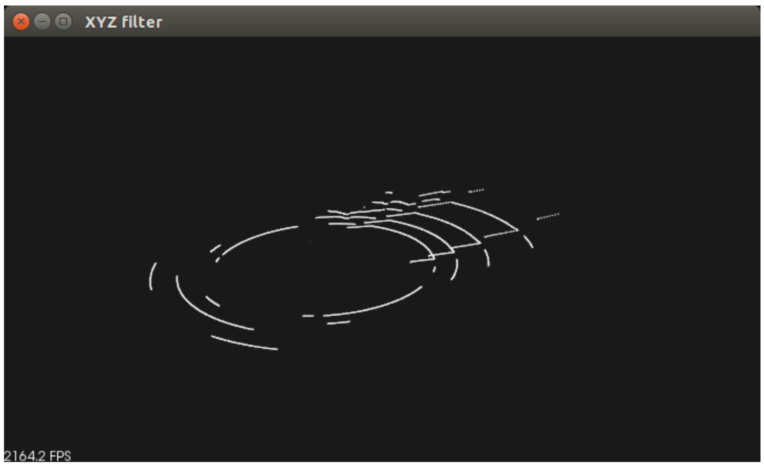


Figure 63: Voxelgrid input point cloud.

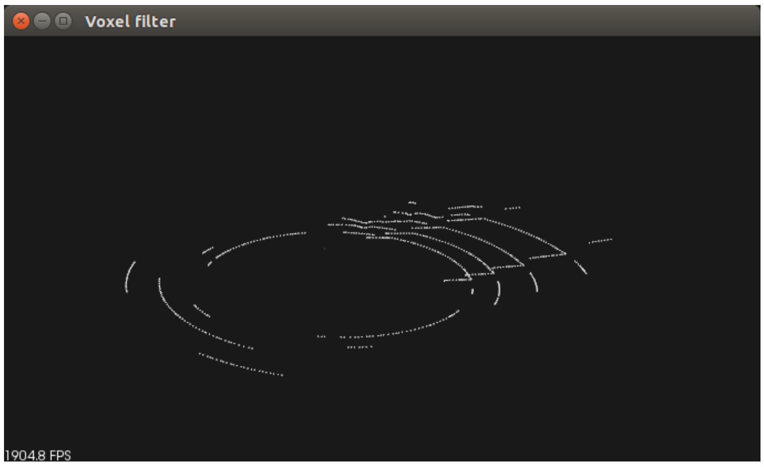


Figure 64: Voxelgrid output point cloud.

The cloud which was used as input for this algorithm is shown in Figure 63. The output one, which is obtained after applying the Voxelgrid filter, is depicted in Figure 64.

Curbs detection

After the point cloud preprocessing and downsampling, it is time to disguise all the pointcloud points which form the curbs.

To manage a vehicle the most basic thing that must be done is to keep it in the road, that is the reason way these curbs are so important for us.

The curbs detection has been carried out using a sample consensus algorithm (SAC). Among the several models that this algorithm offers, the linear model is the one which has been chosen.

Sample consensus algorithm works as it is explained in the following steps:

- A point cloud and a model is provided to sample consensus algorithm.
- The algorithm estimates the model which includes the biggest amount of points of the point cloud.
- The algorithm classifies the whole cloud in inliers, which are the points that fits the model, and outliers, which are the rest of the points.

Providing to this algorithm the point cloud obtained from the downsample we will be able to extract the points which belongs to one of the curbs. Once done that, those points must be removed before using the sample consensus algorithm again to estimate the second curb.

The advantage that this algorithm offers is that it helps to extract the curbs in two separated curbs and it also returns the estimated line model.

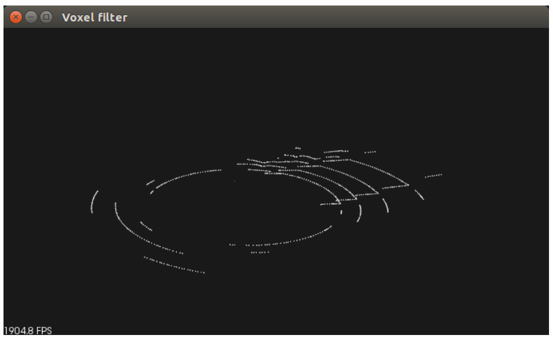


Figure 65: SAC input cloud.

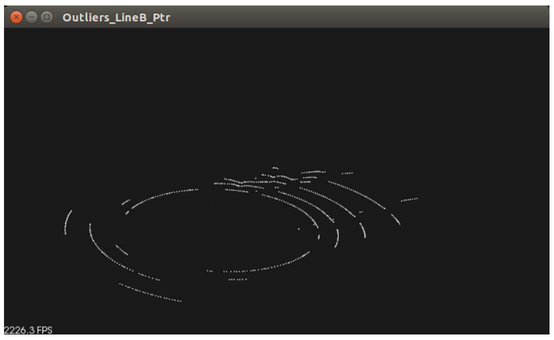


Figure 66: Outliers after the first SAC execution.

In Figure 65 and Figure 66 the two point clouds that were provided to the sample consensus algorithm are depicted. The first one is obtained from the downsample and the second one it formed by the outliers of the first line model estimation.

Once both curbs are extracted from the original cloud, they can be combined in a point cloud as it can be seen in Figure 67.

Also the line models are stored, these models jointly with the point clouds are the ones that will be useful to estimate the parameter that are needed to guide the vehicle.

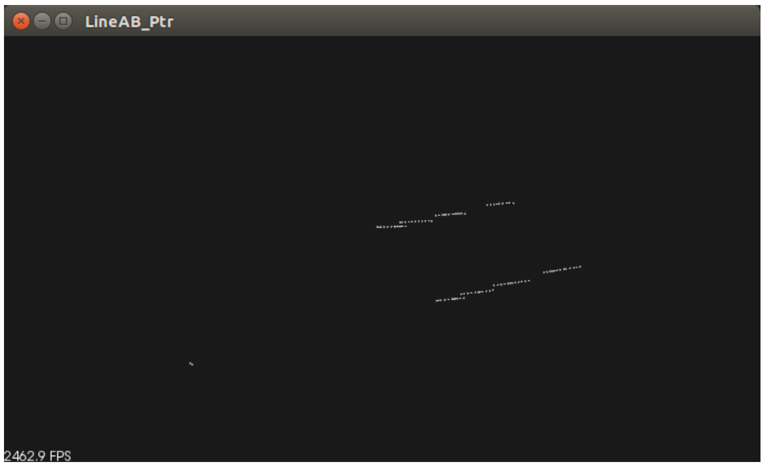


Figure 67: Curbs detection

Target estimations.

Using the data calculated in the previous subsection the necessary consigns to guide the vehicle are computed. To guide a vehicle two parameters should be calculated. The first on is the target point, which will be used for the vehicle to reach it. The second parameter necessary to guide the vehicle is a direction vector which will represent the track that should be followed by the vehicle.

The target point is calculated through geometrical calculations to get the middle point in the road. The vector is estimated to obtain one which is parallel to the curbs.

The aim of calculating these parameters is to manage the vehicle in an autonomous way. To do it a PID controller will be used, but before the target point should be used to estimate the position error and the vector will be used to compute the direction error. By means of reducing these errors to cero using the PID controller the vehicle will be automatically managed.

10 Results

In this chapter the results of the different tests carried out are presented. The aim of this test is to provide an objective measure to evaluate the performance of the developed system.

To carry out the purpose of this chapter several tests were done. These tests can be divided in unitary tests and system tests. The first ones check how partial modules work while the second ones supervise the working of the whole system.

Voxelgrid performance

In this section the effect of using different sizes of VoxelGrid filters will be analyzed. In Table 5 the filter is analyzed depending on the grid size chosen. In the second and third column the initial and the final number of points respectively are shown. Their quotient is represented through the percentage of points that has the filtered cloud regard the original one.

As it can be noted, to carry out all these tests the same cloud has been employed as input to the Voxelgrid filter.

Voxelgrid size	Initial n of points	Final n of points	Time costs	Percentage
0.05, 0.05, 0.05	3313	2452	33 ms	74 %
0.1, 0.1, 0.1	3313	1534	30 ms	46.3 %
0.15, 0.15, 0.15	3313	991	26 ms	29.9 %
0.2, 0.2, 0.2	3313	752	25 ms	22.6 %
0.25, 0.25, 0.25	3313	662	23 ms	19.9 %
0.3, 0.3, 0.3	3313	514	18 ms	15.5 %
0.35, 0.35, 0.35	3313	438	16 ms	13.2 %

Table 5: Voxelgrid results.

It can be noted that there are no big different in terms of time costs depending on the size we chose. The reason why there is no big different among the tests done is that the input cloud has a number of points which is not really big.

Curbs estimation performance

Using the sample consensus algorithm, two inputs must be fixed. The first one is the type of model which it is going to be searched and the second one is the tolerance of the points to consider them as inliers. This tolerance is the allowed distance among the model and all the points considered as inliers.

As it can be seen in Table 6 the bigger the tolerance is the bigger the resultant cloud after applying the sample consensus algorithm is. Input cloud should have the same number of points that curb A plus curb B plus outliers of curb B.

Tolerance	Input cloud	Curb A	Outliers curb A	Curb B	Outliers curb B
0.05	608	12	596	14	582
0.1	608	20	588	17	571
0.15	608	25	583	33	540
0.2	608	32	576	39	537
0.25	608	35	573	54	519

Table 6: Sample consensus results

System performance

To check the performance of the development system the position and direction error has been calculated.

Position error is just calculated in the most important axis, the one which is perpendicular to the direction of movement. This approximation can be done because the road is not quite wide, so the deviation of the vehicles direction cannot be big without going out the road. This error is computed as the position returned by the algorithm minus the target ground truth.

Direction error is expressed as a deviation angle taking into account the two axes which are parallel to the ground. This deviation has been estimated using the vector provided by the algorithm and the vehicle´s one at the moment that the point cloud was captured.

Test number	Position error (meters)
1	-0.12
2	1.26
3	0.5685
4	0.5075
5	-0.083
6	0.367
7	0.253
8	1.094
9	0.846
10	0.642

Table 7: Position error.

Test	Direction error (degrees)
1	15.03
2	-1.7
3	0.159
4	0.039
5	10.72
6	0.563
7	0.543
8	2.296
9	1.85
10	0.234

Table 8: Direction error.

As it can be seen in the obtained results the performance is quite good, but the only problem is that sometimes we get an isolated value with a quite bad performance. This happens due to the simulation environment. It sometimes provokes some error in the velodyne model which ends in a non-complete environment capture. If the point cloud is not complete the algorithm will not give an optimum result.

The results presented in Table 7 and Table 8 were captured in certain points. The position where the vehicle was, is shown in Figure 68:

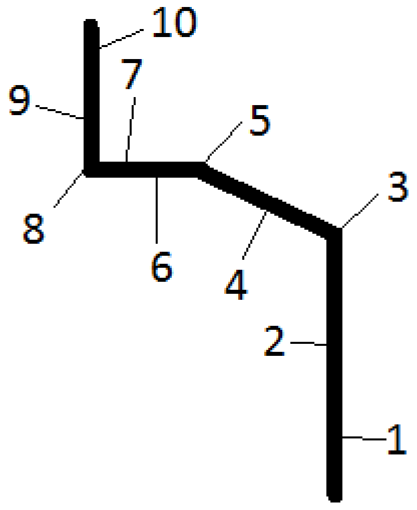


Figure 68: Test vehicle position

11 Conclusions and future work

Conclusions

As it is shown in this Master thesis the objectives that were set at the beginning of this work have been amply achieved.

First of all a whole simulation world was created jointly with all the necessary models to carry out simulations. After that a couple of nodes were implemented to allow a human manage the vehicle which had been added to the simulation environment. Finally a vehicle guidance system was programmed to compute the point and direction that a vehicle should reach to avoid going out the road.

Jointly with this work different complementary tasks have been done such as writing three papers which were accepted and presented in a Robotics conference. The content of one of these papers is a part of the work which is shown in this book.

Future work

There are two possible development lines which can be followed to go ahead with the target of getting closer and closer to the autonomous unmanned vehicle.

The first one consists on going forward using all the developed elements. It will consists on implementing a PID controller, a fuzzy logic controller or any other controller to guide the vehicle using the information provided by the vehicle guidance module, which has been created in this work, and the services provided to manage the vehicle actuators.

The second one consists on looking for any other simulation environment not as time consuming and complex as GAZEBO. GAZEBO needs lots of resources to execute simulations and for big environments as the needed in UGVs applications the simulation can became tough.

12 Specifications

In this chapter the necessary conditions, which should be satisfied to carry out the project, are described:

Hardware specifications

One of the most important tools which have been used are personal computers. At the beginning they were used to carry out searches and download information, later for programming and simulating and finally to carry out tests.

Computational project costs are quite height. In fact during the last development part of the project a more powerful computer were used. The reasons by computational costs are that high are:

- PCL works with a huge amount of information due to the big amount of points that the VLP-16 is able to capture. Different filters can be used to down sample these point clouds but sometimes it is not recommended if it is wanted to hold the application performance level.
- ROS represents lots of points jointly with different robot axes, which makes the computer to consume more resources.
- GAZEBO simulator offers quite good simulation results but it need a high amount of resources to make it work fluently.

The most important specifications of the computers used are described hereafter:

Computer 1

- Processor: Intel® Core ™ i7-3610QM CPU @ 2.30GHz (8CPUs) 2.3GHz.
- Ram Memory: 8192MBs.
- Video Card: NVIDIA® GeForce® 610Mwith 2GB DDR3 VRAM.

Computer 2

- Processor: : Intel® Core ™ i7 CPU 860 @ 2.80GHz (x8CPUs)
- Ram Memory: 3.9 GBs
- Video Card: ATI Radeon HD 6610.

Along the lifecycle project time two different computers were used, the first one was used until we were force to change it due to the computational costs of the developed applications were to height.

Software specifications

All the mandatory software to develop the project does not require any purchase. As it was said before, all these software have licenses that allow you to use it without paying without quite relevant conditions. The necessary software is:

- Linux Operating System: it is recommended to use 14.04 LTS version, but others can be used too.
- Point Cloud Library (PCL): it is integrated in ROS full version so it is not necessary to install it.
- Robot Operating System (ROS): the full version of this software must be installed.
- GAZEBO: DRCSim version must be installed to ease the implementation costs.

13 Budget

In this chapter a breakdown of the estimated costs for this project is attached. The different paragraphs in which we have separated the budget are hardware, software and manpower.

Hardware costs

Concept	Price	Number of units	Total per concept
Laptop Intel Core i7	575	1	575 €
PC Intel Core i7	600	1	600 €
24" Screen	140	1	140 €
PC peripherals	40	1	40 €
Total Hardware			1.355 €

Table 9: Hardware costs

Software costs

In the software breakdown is shown that most of the software requirements are free. The only program which is not free is Office 2010 Professional. Its price is higher but the student version is the one which has been chosen because its price is more economical.

Concept	Price	Number of units	Total per concept
O.S Ubuntu 14.04 LTS	0 €	2	0 €
Point Cloud Library	0 €	2	0 €
Robot Operating System	0 €	2	0 €
GAZEBO	0 €	2	0 €
Office 2010 Professional	70 €	1	70 €
Gcc and G++ compiler	0 €	2	0 €
Kile	0 €	1	0 €
Total Software			70 €

Table 10: Software costs

Manpower costs

Despite the fact that all the work has been done by the same person, the work has been divided in different work profiles. "Development" concept is the time inverted in research and design. "Installation" refers to the PCs setting up time, "Testing" to final proofs which are necessary to check that the system works and obtain performance results. "Documentation" is the time inverted in documenting. Finally "Training" is the time inverted in courses, workshops, conferences, web based trainings, etc.

Concept	Price	Number of hours	Total per concept
Development	40 €	650	26.000 €
Installation	25 €	32	800 €
Testing	30 €	95	2.850 €
Documentation	20 €	150	3.000 €
Training	25 €	185	4.625 €
Total Manpower			37.275 €

Table 11: Manpower costs

Total costs

Concept	Total per concept
Hardware costs	1.355 €
Software costs	70 €
Manpower costs	37.275 €
Total costs	38.700 €

Table 12: Total costs

Contract budget

The contract budget is the total costs plus an added 13% (general costs) and 6% (Industrial benefits).

Concept	Total per concept
Material budget execution	38.700 €
General Costs	5.031 €
Industrial benefits	2.322 €
Total costs	46. 053 €

Table 13: Contract Budget

Total budget amount

Finally, here is the estimation of the final budget for this project. To the contract budget which has been calculated before, it should be added the IVA percentage (21%).

Concept	Total per concept
Contract budget	46. 053 €
IVA costs (21%)	9.671,13 €
Total costs	55.724.13 €

The final cost of the project is FIFTY FIVE THOUSAND SEVEN HUNDRED AND TWENTY FOUR euros and THIRTEEN cents, 55.724.13 €.

Madrid, 15 July 2016

Signed: Alberto Lázaro Enguita

14 User Manual

Installation guide

The most important elements which are necessary to install in a computer to execute the created programs are:

- Linux operating system: Ubuntu 14.04.
- The Robot Operating System(ROS) Indigo version.
- GAZEBO DRCSim version.

Linux Operating System (14.04 version)

To run this application the first thing which is needed is a Linux version. The distribution that has been used to develop the programs is Ubuntu 14.04, so that is the reason why it is the best to run them. Proper working can be guaranteed with Ubuntu 14.04 due to some tests have been carried out with it.

Ubuntu14.04 can be installed following the steps explained in the website: http://howtoubuntu.org/how-to-install-ubuntu-14-04-trusty-tahr.

ROS indigo installation

Configure Ubuntu repositories

The first step that should be done is to configure Ubuntu repositories to allow “restricted”, “universe” and “multiverse”. To do it the following lines must be typed in the terminal:

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu trusty universe"
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu trusty main universe
restricted multiverse"
sudo add-apt-repository "deb http://archive.canonical.com/ubuntu trusty partner"
```

Setup sources lists

Setup the computer to accept software from packages.ros.org. ROS Indigo only supports Saucy (13.10) and Trusty (14.04) for Debian packages typing the next command:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
```

Set up the keys

Set up the keys by executing on the terminal:

```
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net --recv-key 0xB01FA116
```

Installation

Type the next sentence to make sure your Debian package index is up to date:

sudo apt-get update

If you are using Ubuntu Trusty 14.04.2 and experience dependency issues during the ROS installation, you may have to install some additional system dependencies (DO NOT USE THIS IF YOU ARE USING 14.04, IT WILL DESTROY YOUR X SERVER).

sudo apt-get install xserver-xorg-dev-lts-utopic mesa-common-dev-lts-utopic libxatracker-dev-lts-utopic libopenvg1-mesa-dev-lts-utopic libgles2-mesa-dev-lts-utopic libgles1-mesa-dev-lts-utopic libgl1-mesa-dev-lts-utopic libgbm-dev-lts-utopic libegl1-mesa-dev-lts-utopic

Alternatively try installing this to fix dependency issues:

sudo apt-get install ros-indigo-desktop-full

Initialize rosdep

Before using ROS, it will be necessary to initialize rosdep. Rosdep enables to install system dependencies for the source which is wanted to be complied in an easy way. To do it, these commands should be typed:

sudo rosdep init

rosdep update

Environment setup

It's convenient if the ROS environment variables are automatically added to your bash session every time a new shell is launched. Use the next commands to add them:

echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc

source ~/.bashrc

Getting rosintall

Rosinstall is a frequently used command-line tool in ROS that is distributed separately. It enables you to easily download many source trees for ROS packages with one command. To install this tool on Ubuntu, run:

sudo apt-get install python-rosinstall

GAZEBO DRCSIM installation

First time setup

If this is the first time installing the simulator in the computer, these are the steps that must be followed:

To configure Ubuntu repositories to allow “restricted”, “universe” and “multiverse”. You just have to type the following sentences:

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu trusty universe"

sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu trusty main universe
restricted multiverse"

sudo add-apt-repository "deb http://archive.canonical.com/ubuntu trusty partner"
```

Then set up the computer to accept software from packages .osrfoundation.org and packages.ros.org (the DRC Simulator uses some parts of ROS). To do it, write the next commands can be used:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" >
/etc/apt/sources.list.d/ros-latest.list'

sudo sh -c 'echo "deb http://packages.osrfoundation.org/drc/ubuntu trusty main" >
/etc/apt/sources.list.d/drc-latest.list'
```

The next step that should be done is to retrieve and install the keys for the ROS and DRC repositories:

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

wget http://packages.osrfoundation.org/drc.key -O - | sudo apt-key add -
```

After that install the DRC simulator:

```
sudo apt-get update
```

Finally, install drcsim package:

```
sudo apt-get install drcsim
```

GAZEBO update to DRCSIM version

If any other version has previously been installed on the PC, it can be upgraded with the following commands:

```
sudo apt-get update

sudo apt-get install drcsim
```

User guide

ROS configuration

Once the installations described above have been carried out the following steps must be done:

Open a terminal and type:

```
gedit .bashrc
```

After that add at the end of the file the following lines, where “{user}” must be replaced by the name of the user which has been chosen in Ubuntu installation:

```
source /opt/ros/indigo/setup.bash

source /opt/ros/indigo/setup.bash
source /usr/share/drcsim/setup.sh
source /home/{user}
```

Save the file and close it. Then back in the terminal write the next commands:

```
mkdir catkin_ws
catkin_make
cd ~/catkin_ws/
catkin_make install
```

Before using the next command introduce all the packages in /catkin_es/src directory.

```
catkin_make
```

ROS program execution

Once the steps shown before have been done its time to run the .launch and .run programs.

To execute a .launch program the following command shown hereafter should be used, where the elements written between brackets are replaced by what they mean.

```
roslaunch {package name} {.launch file}
```

If the program is a .run the sentence shown hereafter should be used instead:

```
rosrun {package name} {.run file}
```

15 Bibliography

[1] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., ... & Johnston, D. (2009). Junior: the stanford entry in the urban challenge. InThe DARPA Urban Challenge (pp. 91-123). Springer Berlin Heidelberg.

[2] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., ... & Lau, K. (2006). Stanley: The robot that won the DARPA Grand Challenge.Journal of field Robotics, 23(9), 661-692.

[3] Portland State University. Light Detection and Ranging (LiDAR): http://web.pdx.edu/~jduh/courses/geog493f12/Week04.pdf

[4] Losada, C., Mazo, M., Palazuelos, S., & Jiménez, D. (2012). Segmentación de movimiento a partir de la información de cámaras de tiempo de vuelo.

[5] Cory, P., Everett, H. R., & Heath-Pastore, T. A. (1999, January). Radar-based intruder detection for a robotic security system. In Photonics East (ISAM, VVDC, IEMB) (pp. 62-72). International Society for Optics and Photonics.

[6] Vargas González, D. (2014). Detección de movimiento mediante técnicas radar CW-FM en banda W.

[7] Elvira, S., De Castro, A., & Garrido, J. (2013). ALO: An ultrasound system for localization and orientation based on angles. Microelectronics Journal, 44(10), 959-967.

[8] Ensenso and IDS GmbH. Obtaining Depth Information from Stereo Images: http://www.sanxo.eu/content/whitepaper/IDS_Whitepaper_3D_Stereo_Vision.pdf

[9] NASA Mission pages Seventeen Cameras on Curiosity: http://www.nasa.gov/mission_pages/msl/multimedia/malin-4.html#.V3jqkfmLS01

[10] Trincado Alonso, J. L., Torres Salcedo, B., & Pérez Alonso, A. (2011). Sistema de Visión Estereoscópica para Navegación Autónoma de vehículos no tripulados.

[11] Martín Carabias, D., Requero García, R., & Rodríguez Salor, J. A. (2010). Sistema de visión estereoscópica para navegación autónoma de vehículos no tripulados.

[12] Lázaro A., Moriano J., Bersaga L.M., Barea R., Lopez E. (2016) 3D Bbject recognition and pose estimation using VFH descriptors. Robocity16 Open Coference on Futur Trends in Robotics (pp. 113-120)

[13] Stoyanov, T., Mojtahedzadeh, R., Andreasson, H., & Lilienthal, A. J. (2013). Comparative evaluation of range sensor accuracy for indoor mobile robotics and automated logistics applications. Robotics and Autonomous Systems,61(10), 1094-1105.

[14] Yin, H., Yang, X., & He, C. (2016). Spherical Coordinates Based Methods of Ground Extraction and Objects Segmentation Using 3-D LiDAR Sensor. IEEE Intelligent Transportation Systems Magazine, 8(1), 61-68.

[15] Lane, G. R., Lescoe, P., & Cooper, S. (1991, March). Unmanned ground vehicle control technology. In Telesystems Conference, 1991. Proceedings. Vol. 1., NTC'91., National (p. 329). IEEE.

[16] Faessler, M., Fontana, F., Forster, C., Mueggler, E., Pizzoli, M., & Scaramuzza, D. (2015). Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle. Journal of Field Robotics, 1.

[17] Eren, F., Pe'eri, S., Rzhanov, Y., Thein, M. W., & Celikkol, B. (2016). Optical Detector Array Design for Navigational Feedback Between Unmanned Underwater Vehicles (UUVs). IEEE Journal of Oceanic Engineering, 41(1), 18-26.

[18] Aeschimann, R., & Borges, P. V. K. (2015, May). Ground or obstacles? Detecting clear paths in vehicle navigation. In 2015 IEEE International Conference on Robotics and Automation (ICRA) (pp. 3927-3934). IEEE.

[19] Agüero, C. E., Koenig, N., Chen, I., Boyer, H., Peters, S., Hsu, J., ... & Krotkov, E. (2015). Inside the virtual robotics challenge: Simulating real-time robotic disaster response. IEEE Transactions on Automation Science and Engineering, 12(2), 494-506.

[20] Koenig, N., & Howard, A. (2004, September). Design and use paradigms for gazebo, an open-source multi-robot simulator. In Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on (Vol. 3, pp. 2149-2154). IEEE.

[21] “Gazebo official website,” http://gazebosim.org/.

[22] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

[23] “ROS official website, “ http://www.ros.org/.

[24] Rusu, R. B., & Cousins, S. (2011, May). 3d is here: Point cloud library (pcl). InRobotics and Automation (ICRA), 2011 IEEE International Conference on (pp. 1-4). IEEE.

[25] PCL official website , http://pointclouds.org/

[26] Gupta, N., Vijay, R., Korupolu, P. V., Bansal, J., & Kapuria, A. (2015, July). Architecture of autonomous vehicle simulation and control framework. In*Proceedings of the 2015 Conference on Advances In Robotics* (p. 65). ACM.

Appendix: Additional activities

In order to achieve the required knowledge for the subject “Introducción al TFM” some additional activities were performed. In the draft, we proposed some of them which are shown hereafter:

- Approach of the thesis using the scientific method. To do it, documentation about the scientific methodology was studied to suggest a hypothesis. All the work done to design and validate it is shown in this book.
- Development of a complete state of the art related with the technologies and techniques included in the thesis. The state of the art is included in the thesis and bibliographic sources are shown in the references chapter.
- Writing of several scientific papers, which will be sent to international conference in the field of robotics. These papers will be added as part of the book.
- Implementation of the designed algorithms in a real environment simulations that allows validating the obtained results using an experimental methodology.
- Attendance to conferences and lectures that are interesting for the development of the project. Also courses, which topics are related with the TFM development or the subjects studied along the master.

Approach of the thesis using the scientific method.

Scientific method was studied. All the information was collected through internet. A brief summary of the most relevant information is detailed hereafter:

Scientific method comprehends all the practices and techniques which are used and ratified by the scientific community. They have the aim of validating the previously raised hypothesis. Francis Bacon defined the scientific methods as follows:

1) Observation: it consists of applying the senses to study an object or a phenomenon as it is presented in the reality.
2) Induction: the process of generalizing the facts that have been observed in singular cases.
3) Hypothesis: elaborate a draft explanation of the observed facts and their consequences.
4) Test the hypothesis by experimenting.
5) Demonstrate or refuse the hypothesis.
6) Thesis or scientific theory

In Figure 69 a more general description of the scientific method is depicted. Science does not try to be absolute nor dogmatic. All the ideas, hypothesis, theories and scientific knowledge should be reviewed studied and modified, if necessary. Experiments should be carried out an it is really important to analyze the results in a proper way to avoid cognitive prejudice

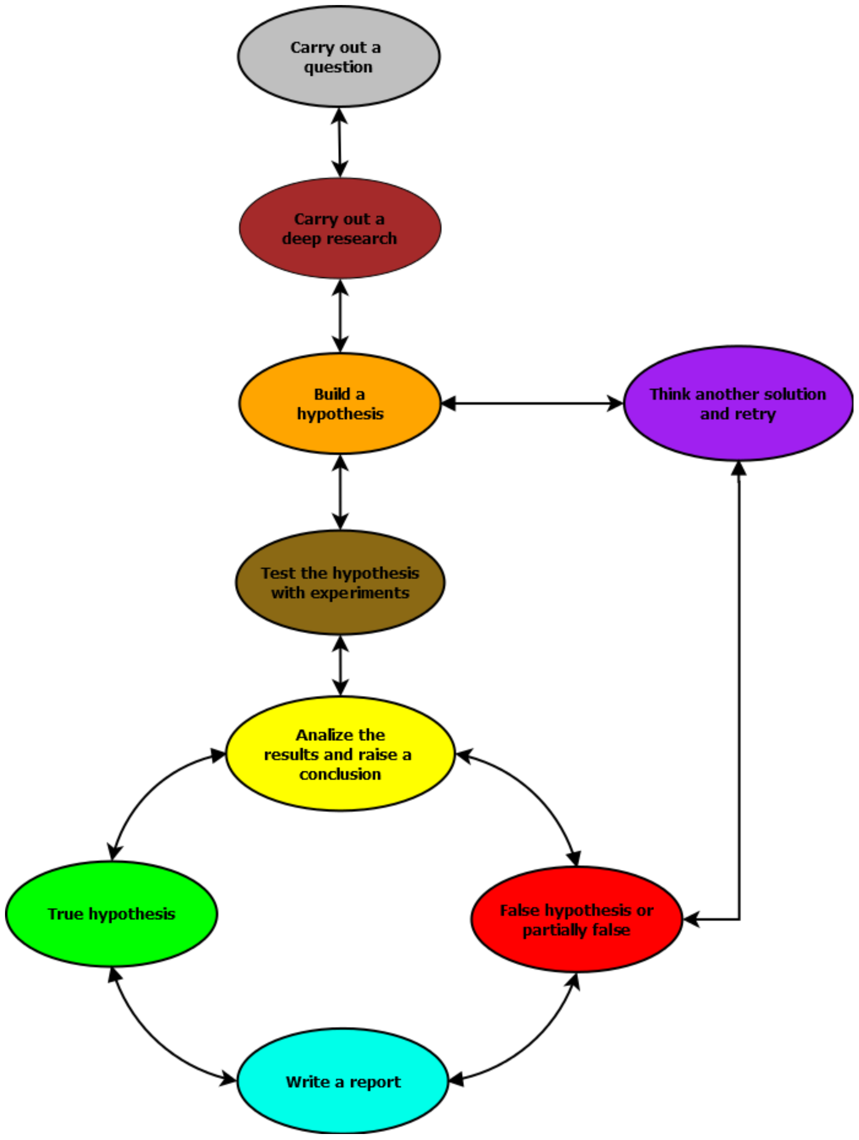

Carry out a question
Carry out a deep research
Build a hypothesis
Think another solution and retry
Test the hypothesis with experiments
Analize the results and raise a conclusion
True hypothesis
False hypothesis or partially false
Write a report

Figure 69: Simplified scheme of scientific method

Development of a complete state of the art.

A deep research was down to find the state of the art technologies and works which are related with the field were this project can be framed. Also the development of the different kinds of unmanned vehicles was studied to locate the project in the transport media field. All the references that were used to carry out the state of the art chapter can be found in bibliography chapter.

Some of the found references were used to learn about the different tools that were employed to carry out the development of the project. These tools provide a big functionality but their use is quite complex.

Writing of several scientific papers.

Along the project period three different papers were prepared and accepted in the RoboCity 16 Open Conference on future trends in robotics for oral exposition.

Conference link: http://www.robocity2030.org/events/event/evento-esp-2-2/

One of the papers accepted for this conference contains the description of the same work that is covered along this book. The other two present some of the work that was carried out in my TFG (3D Object Recognition and Pose Estimation using VFH Descriptors).

The data of the publications is shown hereafter:

Authors: A. LÁZARO, L.M. BERGASA, R. BAREA and E. LOPEZ

Tittle: UGV NAVIGATION IN ROS USING LIDAR 3D

Publication: RoboCity16 - Open Conference on future trends in robotics

Date: May 16

Authors: A. LÁZARO, J. MORIANO, L.M. BERGASA, R. BAREA and E. LOPEZ

Tittle: 3D OBJECT RECOGNITION AND POSE ESTIMATION USING VFH DESCRIPTORS

Publication: RoboCity16 - Open Conference on future trends in robotics

Date: May 16

Authors: A. LÁZARO, J. MORIANO, L.M. BERGASA, R. BAREA and E. LOPEZ

Tittle: GRABBING OBJECTS THROUGH A ROBOTIC ARM AND HAND IN A SAFETY WAY

Publication: RoboCity16 - Open Conference on future trends in robotics

Date: May 16

In the following pages the three accepted papers can be seen. Also the certificate that ratifies that it was defended in the conference.

CHAPTER 14

3D OBJECT RECOGNITION AND POSE ESTIMATION USING VFH DESCRIPTORS

A. LÁZARO, J. MORIANO, L.M. BERGASA, R. BAREA and E. LOPEZ

Robesafe group, Universidad de Alcalá, alberto2991lazaro@gmail.com

This paper presents the perception system involved in an application for grabbing objects in an industrial environment through a robotic arm by using 3D pointcloud images taken from a RGBD camera. This system recognizes and estimates the pose of the objects placed on the working area. It uses VFH descriptors and Kd-tree classifiers for the recognition and geometrical models and a RANSAC estimator for the pose estimation using this information. Some experimental results in a real environment validate our proposal.

1 Introduction

This work is focused on object recognition and pose estimation in industrial environments using a RGBD camera. This vision system is able to detect objects, recognize them and estimate their position offering the possibility of detecting important features which can be used to automate processes. This target can be achieved using RGB cameras but they have some drawbacks faced with depth cameras which will be explained hereafter. The stages that have to be completed by the vision system to provide the necessary data are segmentation, recognition, pose estimation and scene reconstruction. Segmentation identifies the interested objects to be grabbed. Once these objects are segmented they pass to the recognition algorithm. Shape estimator is used to recognize and obtain the position and orientation of the objects. Taking advantage of ROS communication facilities, the obtained data are published to be reconstructed thanks to RVIZ as it can be seen in Fig. 1.

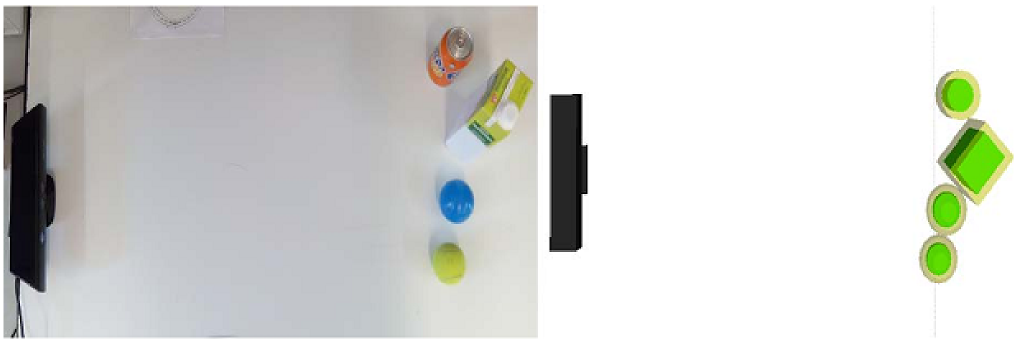


Fig. 1. Captured environment and virtual reconstruction.

2 State of the art

Regarding to the topic of automatic objects classification, several projects have been carried out. Hereafter we show some of the most important. In (Bdiwi, 2012) a more complex classification process is developed, in this case the system is able to difference among books depending on their shapes and using alphanumeric codes. They use a CCD camera, it uses a rectangle to compare the different shapes. Another example for the object classification is included in (Li, 2014) using a depth sensor. In this work, a multilevel part-based object model was proposed using latent support vector machine as a core learning machine for training.

Our proposal, as difference with the above works, uses a depth camera for obtaining 3D Point Cloud images (Cruz,2012), (Rusu,2011) and then to recognize the shape of the different objects. It provides several advantages, the main ones are the use of an RGB-D camera and a better 3D reconstruction of the scene.

3 Object recognition method

The vision system must detect the objects placed in the working scene. So 3D pointcloud is clustered in foreground and background classes using depth. After that, foreground class is downsampled using Uniform Sampling algorithm named VoxelGrid filter before being clustered in objects.

The aim of clustering the foreground data is to divide an unorganized PCL into smaller parts, which contain the points that belongs to a same object (Muja, 2009). This clustering has been done relying on spatial decomposition techniques that find boundaries and subdivisions. The algorithm used is based on Euclidean distances.

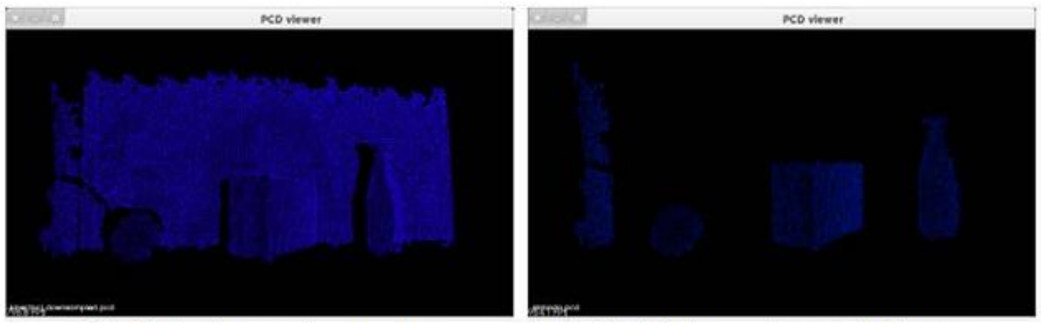


Fig. 2. Background extraction example: in the left image the original capture.

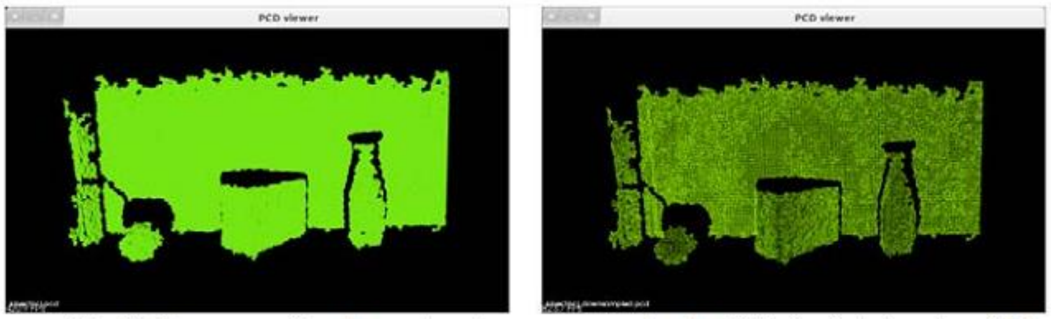


Fig. 3. Downsampling the captured scene example: the PCL depicted on the right.

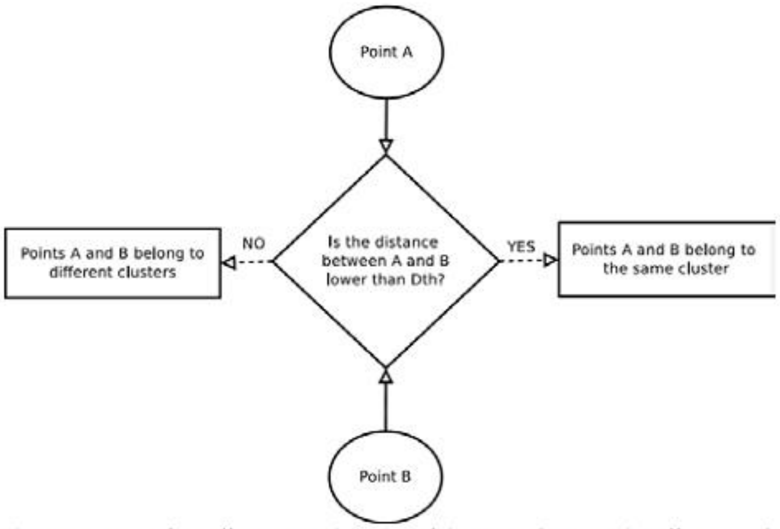

Point A
Is the distance between A and B lower than Dth?
NO
Points A and B belong to different clusters
YES
Points A and B belong to the same cluster
Point B

Fig. 4. Cluster extraction diagram: the algorithm works out the distance between two points, depending on this value the evaluated points are considered as part of the same cluster or as part of two different clusters according to the diagram.

Once the scene is clustered, is time to tackle the object recognition of the different objects placed in the scene (Rusu, 2010; Rusu, 2009). To carry out this recognition VFH descriptors, which is based on the FPFH descriptor, are used (Rusu, 2009; Rusu, 2008). This recognition consists in two processes:

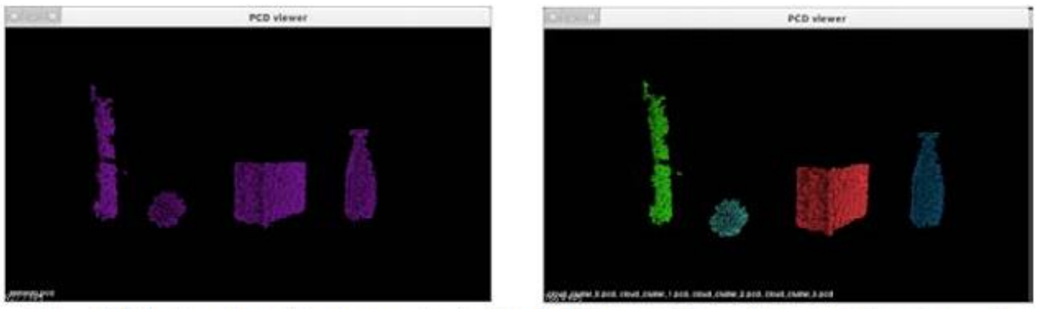


Fig. 5. Cluster extraction example: in the right image the points considered as part of the same cluster are represented in the same color.

3.1 Training process

The first step is to build a dataset that contains some captures (36 in our case) taken from different points of view for each object that is wanted to be recognized by the system and to calculate their VFH descriptor. This dataset is used to train the object recognition system. The training process builds a space-partitioning data structure that stores a set of K-dimensional points in a tree structure that enables efficient range searches and nearest neighbor searches called Kd-tree (Rusu, 2008), (Salti, 2011).

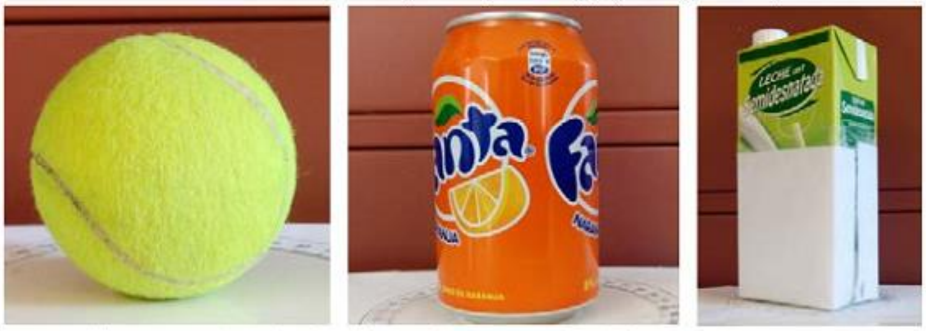


Fig. 6. Objects employed to build the dataset. Each object represents a sample of the three classes needed in this work: ball can and milk (from left to right).

3.2 Testing process

This process compares VFH descriptors for each of the segmented objects extracted from the captured scene and the ones that are saved in the training dataset through K-Nearest neighbors algorithm(k-NN) (Muja, 2011). The result of this comparison is a set of index which values are inversely proportional to the likeness between the analyzed cluster and each element of the dataset (Bariya, 2010). These index are provided to a voting method to determine the class among the trained classes which belongs to the unknown object if the analyzed object belongs to one of the trained classes. If so, the voting system classifies it.

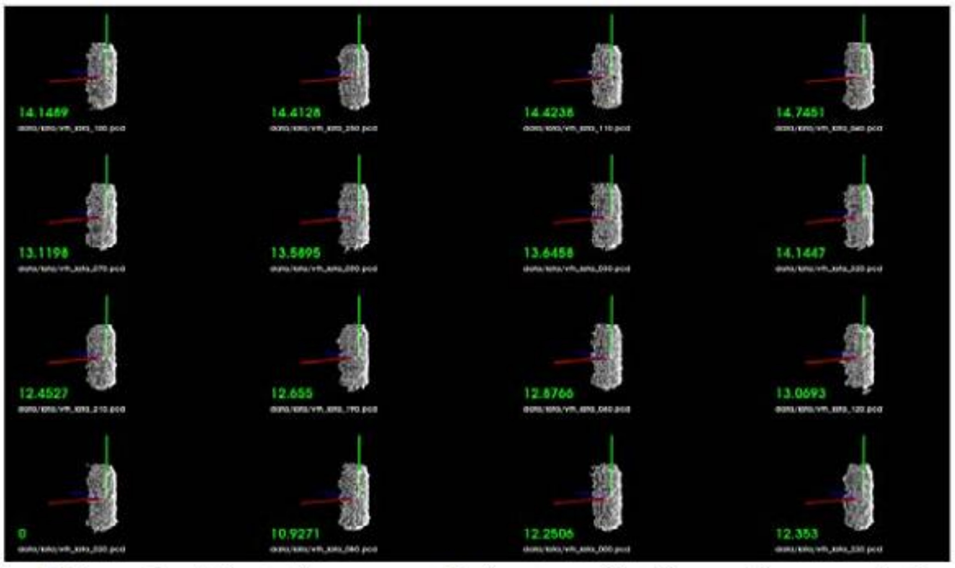


Fig. 7. Example of the testing process: the input used in this working example is one of the PCLs of the dataset.

4 Pose estimation method

Once each of the elements placed in the working scene are recognized, the coefficients that determine their position and orientation are recovered by using RANdom SAmple Consensus (RANSAC). RANSAC algorithm is usually employed to carry out in objects recognition approaches when the set of objects to be distinguished have different shapes (Schnabel, 2007), (Papazov, 2010), but in this work it has been used just for estimating the pose of the objects and their dimensions.

Three models (sphere, cylinder, plane) jointly with their RANSAC estimators are used to estimate the parameters that defines the pose and the dimensions of the different objects (balls, cans, milk cartons) (Aldoma, 2012), (Mishra,2011). To estimate the dimensions of the carton box once the different planes have been estimated geometric constraints are applied.

5 Perception system experiments

The following tests have been carried out to measure the power of the designed vision system evaluating different aspects such as time costs or percentage success. Hereafter we present the different tests carried out to evaluate our vision system performance regarding the processing time an object recognition results. The first experiment tests the performance of our objects recognition system for 5 different objects of each class not used in the training process. As it can be seen in Table 1 the correct recognition ratio is close to 100%.

Table 1. Object recognition results: index shown are the returned by the system.

Object/Candidate	Ball	Can	Milk carton
Ball	100%	0%	0%
Can	1%	95%	4%
Milk carton	1%	5%	94%

The object recognition system runs in a computer with Intel Core i7 CPU 860 @ 2.8GHz x8} processor. Table 2 evaluates the different processing time depending on the recognized elements

Table 2. Object recognition processing times.

Object	No. balls	No. cans	No. Cartons	Time costs
Test 1	1	0	0	190ms
Test 2	2	0	0	275ms
Test 3	3	0	0	410ms
Test 4	0	1	0	345ms
Test 5	0	2	0	650ms
Test 6	0	0	1	890ms
Test 7	1	1	0	515ms
Test 8	1	1	1	1345ms

Fig. 7 shows the results extracted from all tests to determine the dimension, position and orientation errors through the mean and the deviation for the different primitive shapes measured in meters.

6 Conclusions and future work

This paper has addressed the object recognition, pose estimation and dimensions appraisal. Due to the real time constraints geometric restrictions, VFH descriptors and a Kd-tree classifier have been applied over the 3D point cloud images. Object pose estimations are precise enough for a robot to interact with them. The vision system algorithm could provide a robotic system the information needed to move or avoid objects. Although we have presented a real time solution for the addressed problem we will go on working on reducing the processing time in order to make our system able to recognize objects faster than a human.

Acknowledgements

This work was supported in part by the MINECO SmartElderlyCar (TRA2015-70501-C2-1-R) project and by the RoboCity2030 III-CM project (S2013/MIT-2748) funded by Programas de I+D en la Comunidad de Madrid and cofunded by Structured Funds of the UE.

References

Aldoma, A., Marton, Z.-C., Tombari, F., Wohlkinger, W., Potthast, C., Zeisl, B., Rusu, R.B., Gedikli, S., Vincze, M., 2012. Tutorial: Point Cloud Library: Three-Dimensional Object Recognition and 6 DOF Pose Estimation. IEEE Robot. Automat. Mag.

Bariya, P., & Nishino, K. (2010, June). Scale-hierarchical 3d object recognition in cluttered scenes. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on (pp. 1657-1664). IEEE.

Bdiwi, M., & Suchý, J. 2012. Robot control system with integrated vision/force feedback for automated sorting system. In Tech. for Practical Robot Applications (TePRA), 2012 IEEE Int. Conf. on (pp. 31-36). IEEE.

Cruz, L., Lucio, D., & Velho, L. 2012. Kinect and rgbd images: Challenges and applications. In Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on (pp. 36-49). IEEE.

Li, K., & Meng, M. 2014. Robotic object manipulation with multilevel part-based model in RGB-D data. In Robotics and Automation (ICRA), 2014 IEEE International Conference on (pp. 3151-3156). IEEE.

Mishra, A. K., & Aloimonos, Y. (2012). Visual segmentation of “simple” objects for robots. Robotics: Science and Systems VII, 1-8.

Muja, M., & Lowe, D. G. 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. VISAPP (1),2, 331-340.

Muja, M., Rusu, R. B., Bradski, G., & Lowe, D. G. (2011, May). Rein-a fast, robust, scalable recognition infrastructure. In Robotics and Automation (ICRA), 2011 IEEE Intern.l Conference on (pp. 2939-2946). IEEE.

Papazov, C., & Burschka, D. (2010). An efficient RANSAC for 3D object recognition in noisy and occluded scenes. In Computer Vision–ACCV 2010 (pp. 135-148). Springer Berlin Heidelberg.

Rusu, R. B., Blodow, N., & Beetz, M. 2009. Fast point feature histograms (FPFH) for 3D registration. In Robotics and Automation, 2009. ICRA'09. IEEE International Conference on (pp. 3212-3217). IEEE.

Rusu, R. B., Bradski, G., Thibaux, R., & Hsu, J. 2010. Fast 3d recognition and pose using the viewpoint feature histogram. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ Intern. Conf. on (pp. 2155-2162). IEEE.

Rusu, R. B., Holzbach, A., Beetz, M., & Bradski, G. 2009. Detecting and segmenting objects for mobile manipulation. In Computer Vision Workshops (ICCV WS), 2009 IEEE 12th Int. Conf. on (pp. 47-54). IEEE.

Rusu, R. B., Marton, Z. C., Blodow, N., & Beetz, M. 2008. Learning informative point classes for the acquisition of object model maps. InControl, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on (pp. 643-650). IEEE.

Rusu, R. B., Marton, Z. C., Blodow, N., & Beetz, M. 2008. Persistent point feature histograms for 3D point clouds. In Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany (pp. 119-128).

Rusu, Radu Bogdan and Cousins, Steve. 2011. 3D is here: Point Cloud Library (PCL). ICRA.

Salti, S., Tombari, F., & Stefano, L. D. 2011. A performance evaluation of 3d keypoint detectors. In 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 Intern. Conf. on (pp. 236-243). IEEE.

Schnabel, R., Wahl, R., & Klein, R. (2007, June). Efficient RANSAC for point-cloud shape detection. In Computer graphics forum (Vol. 26, No. 2, pp. 214-226). Blackwell Publishing Ltd.

CHAPTER 37

UGV NAVIGATION IN ROS USING LIDAR 3D

A. LÁZARO[1], R. BAREA[2], L.M. BERGASA[3] and E. LOPEZ[4]

[1]Robesafe group, Universidad de Alcalá, [2]Robesafe group, Universidad de Alcalá, [3]Robesafe group, Universidad de Alcalá, [4]Robesafe group, Universidad de Alcalá, alberto2991lazaro@gmail.com.

This paper addresses to give a step more forward the achievement of robust Unmanned Ground Vehicles (UGVs), which can drive in urban environments. More specifically, it focuses in the management of a four wheeled vehicle in ROS using mainly the inputs provided by a LIDAR 3D. Simulations are carried out in ad-hoc scenarios designed and run using GAZEBO. The vehicle and the LIDAR sensor physical models have been added to the simulation environment in order to get a higher degree of reality. Navigation tests will be provided in the final version.

1 Introduction

Unmanned Ground Vehicles (UGVs) are becoming more and more used every single day. They can carry out lots of different tasks with no human supervision, achieving lower costs or working in areas where it would be dangerous for humans to work in. Those are the reasons why they are commonly used in industrial environments.

Industrial applications have a particularity, which makes easier the management of these vehicles, the environment is controlled. According to the guided systems the unmanned vehicles use, they can be classified as shown in Fig. 1.

In other applications where the environment is uncontrolled the sensors and manage systems should be more complex to ensure success in the vehicle autonomous management. This is our approach, because our goal is the UGV navigation in urban scenarios.

In this work, an UGV is managed through ROS using a LIDAR 3D sensor. This management is carried out in GAZEBO, a simulator where the needed models have been built.

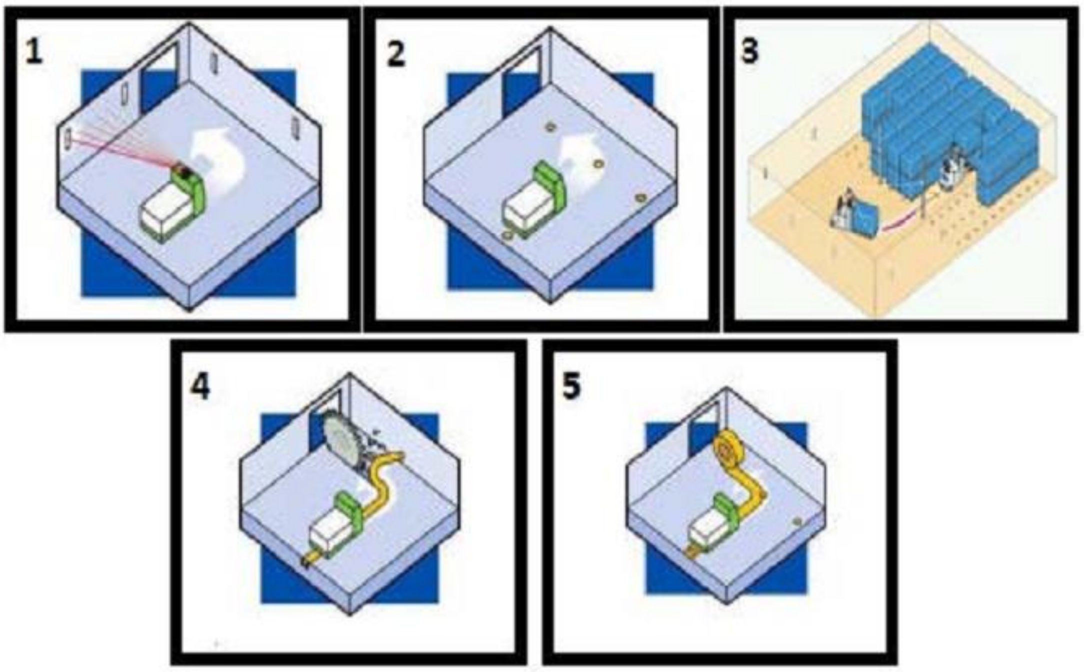


Fig. 1: (1) Laser UGV (2) Magnetic UGV (3) Dual UGV (4) Rail guided UGV (5) Optical band UGV.

The reason why GAZEBO world has been build is to create a virtual reconstruction of real scenarios, where testing the perception and vehicle management systems. Virtual simulations are usually used in the first implementation steps of autonomous vehicles. It allows us to test the developed systems as many times as needed in a safety way with reduced costs.

The sensor used to be mounted in the vehicle is the VLP-16 (Velodyne Lidar Puck). It creates a 360º 3D images using 16 laser/detector pairs mounted in a compact housing. The housing rapidly spins to scan the surrounding environment. The lasers fire thousands of times per second, providing a rich, 3D point cloud in real time.
VLP-16 includes the following characteristics:

- Horizontal Field of View (FOV) of 360°
- Rotational speed of 5-20 rotations per second (adjustable)
- Vertical Field of View (FOV) of 30°
- Returns of up to 100 meters (useful range depends on application)

2 State of the art

Nowadays, an interesting research line is the development of unmanned robotic systems. This target has multiple different purposes such as goods transportation, people transportation, etc.

These robots can be classified in 3 different categories: UGVs (Unmanned Ground Vehicles) (Lane, 1991), UAVs (Unmanned Aerial Vehicles) (Faessler, 2015) and UUVs (Unmanned Underwater Vehicles) (Eren, 2016).

Perception systems make possible unmanned vehicles. They let the vehicle know how the environment around it is and if there is any object around them. These perception systems can consist on several sensors such as cameras (Ahmed, 2007) or laser (Liu, 2015).

There are different types of 2D and 3D laser sensors. Comparison shows the advantages that 3D laser sensors provide (Nieuwenhuisen, 2015).

Navigation systems are also necessary to complete the intelligence of Unmanned Vehicles. Their target is to plan trajectories avoiding obstacles or hurdles found in the vehicle´s way (Aeschimann, 2105) (Quigley, 2009).

In this work several platforms has been used to develop the system which manages the autonomous vehicle such as ROS (Quigley, 2009 and GAZEBO (Roberts, 2003).

Different works can be found where these platforms have been used for similar purposes such as (Gupta, 2015) but there is no work that joins the power or both platforms with LIDAR 3D sensors, GPS, cameras and IMUs.

3 Work description

The steps which have been tackled to get the final goal are the following ones:

- Build a proper GAZEBO world to run the simulations.
- Build the UGV model with the sensor attached to it to make them move in solidarity.
- Prepare a ROS node which depicts the information obtained through RVIZ.
- Develop a management system which drives the car using the available information.

A general scheme of our system architecture is shown in Fig. 2.

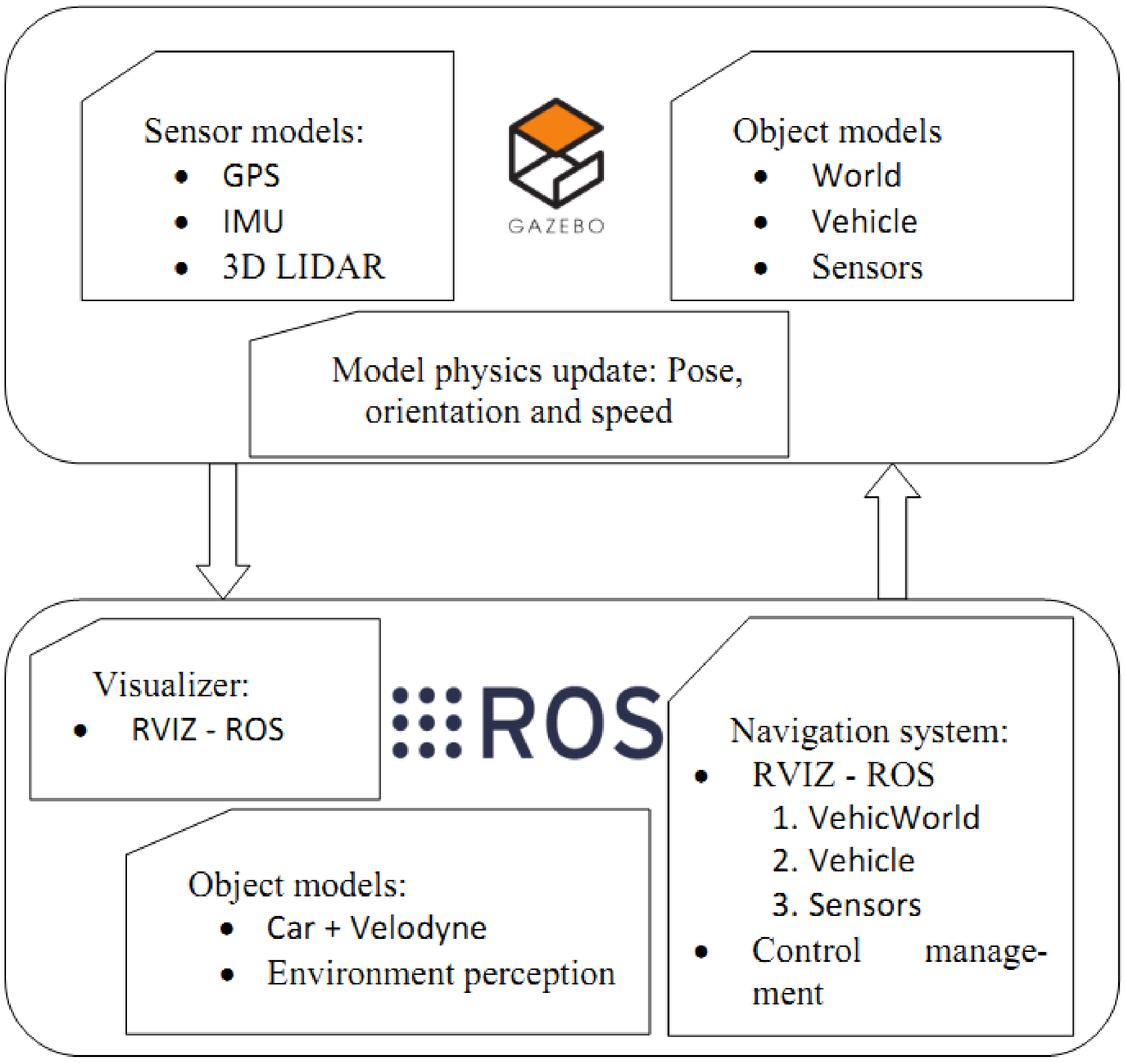


Fig. 2: Proposed system architecture.

3.1 Simulation world

The world is the environment where the UGV will be managed. To build it a grass plane has been used as base of the world. Different models from the DRCSim GAZEBO version have been added to the base. The models used are the ones which are shown hereafter:

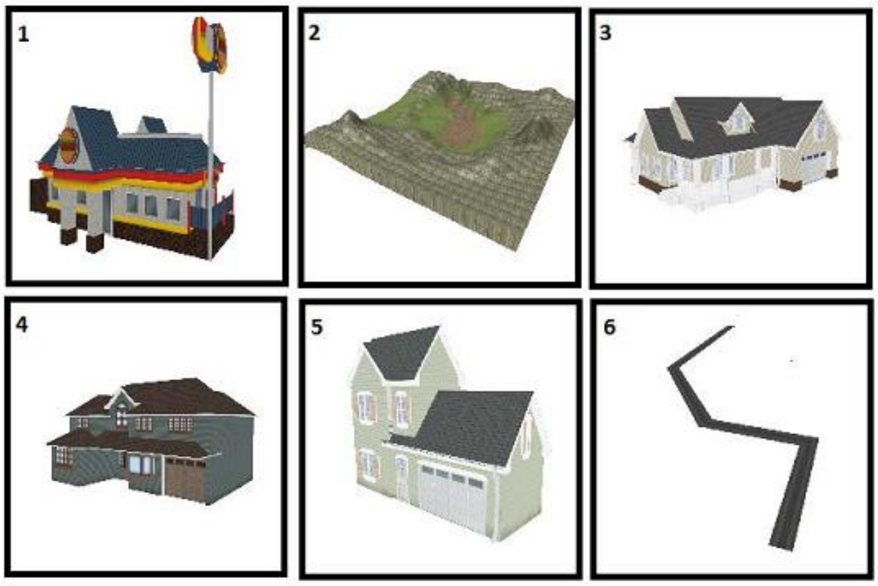


Fig. 3: (1) Fast food restaurant (2) Mountains landscape (3) House 1 (4) House 2 (5) House 3 (6) Road.

Furthermore, curbs have been added along both sides of the road. These curbs have been added in ten sections, one on the right, another one on the left of each section and two more at the start and at the end. The length of these curbs depends on the length of each section. Fig. 4 shows a picture of our simulation world.

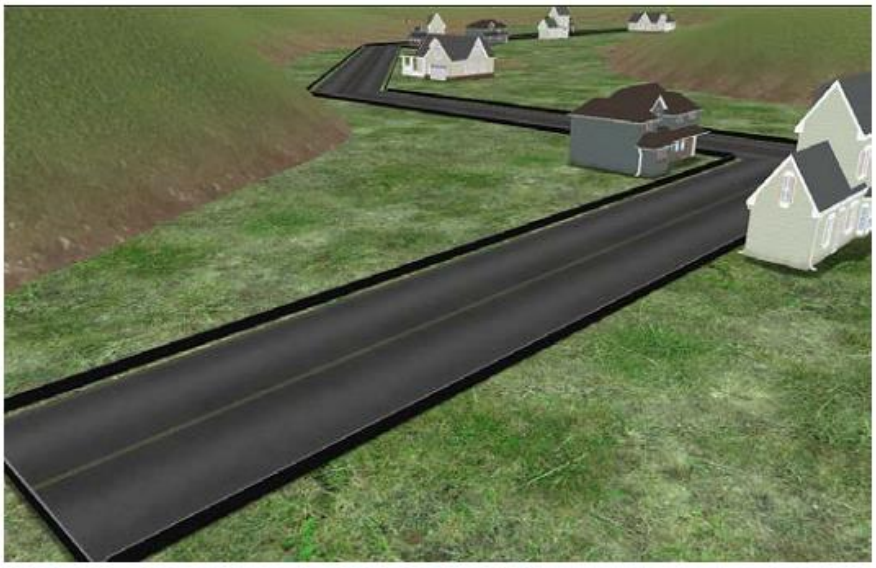


Fig. 4: Simulation world.

3.2 UGV model

The Vehicle used is the DRC Vehicle, which can be found in the DRCSim GAZEBO version. Its model has been modified to add to it the VLP-16 (Velodyne Lidar Puck). After adding the model different constraints have been defined to make the VLP-16 move in solidarity with the Vehicle.

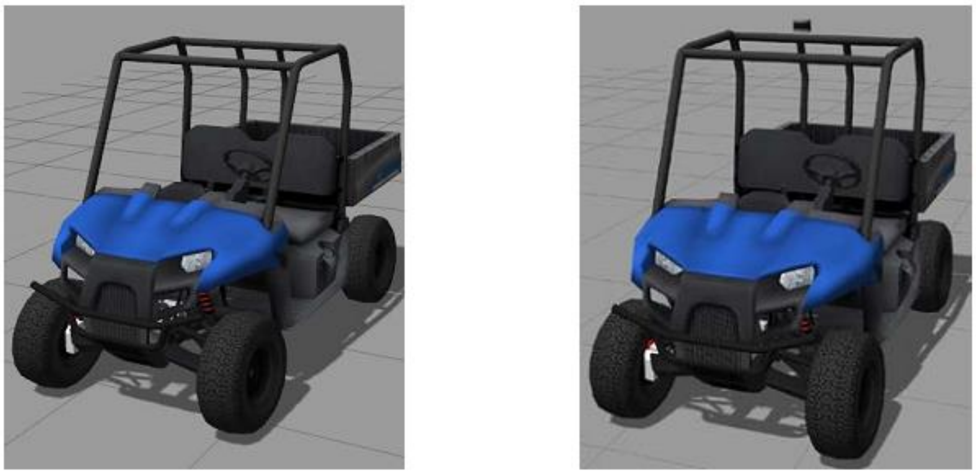


Fig. 5. (1) DRC Vehicle. (2) DRC Vehicle + VLP-16.

3.3 RVIZ sensors representation.

In Fig. 6 the real world simulation and the VLP-16 are depicted. Here it can be seen that the useful range is quite big while obtaining high precision.

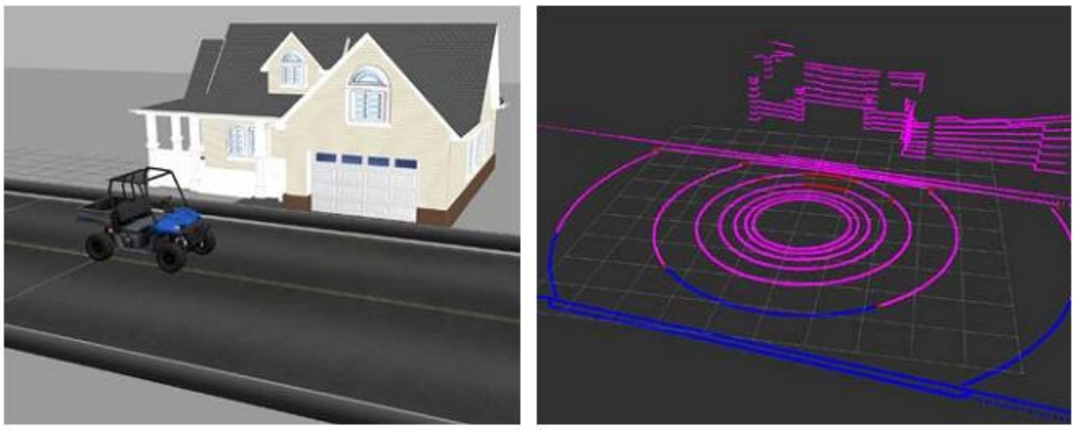


Fig. 6. (1) Real world simulation. (2) VLP-16 detection depicted RVIZ.

3.4 Navigation system.

A ROS node has been implemented to manage the UGV. It receives information of the environment provided by the sensors and information that shows the state of the DRC Vehicle such as position and speed.

All this information is processed and used to act in the control systems of the UGV managing the hand wheel, the gas pedal, the brake pedal, etc. Control architecture will be included in the trial version of the paper.

4 Experimental results

Experimental results consist on navigation text on the designed world in an autonomous way avoiding static and moving obstacles. We are working on it and they will be included in the final version.

5 Conclusions

The work presented in this paper addresses to achieve the management of an UGV in an urban scenes using ROS. It provides as novelty the jointly use of ROS, GAZEBO and a Velodyne sensor. Currently we are working in the control system, which is the reason why control architecture, experimental results and conclusions will be completed in the final version of the paper.

Acknowledgements

This work was supported in part by the MINECO SmartElderlyCar (TRA2015-70501-C2-1-R) project and by the RoboCity2030 III-CM project (S2013/MIT-2748) funded by Programas de I+D en la Comunidad de Madrid and cofunded by Structured Funds of the UE.

References

Gupta, N. Vijay R. Korupolu P.V.N. Bansal J. and Kapuria A. (2015). Architecture of autonomous vehicle simulation and control framework.. AIR.

Lane G. R. Lescoe P. Cooper S. 1991. Unmanned Ground Vehicle control technology. Telesystems Conference, 1991. Proceedings. Vol.1., NTC '91., National.

Aeschimann R. and Borges P. V. K. 2015. Ground or obstacles? Detecting clear paths in vehicle navigation. 2015 IEEE International Conference on Robotics and Automation (ICRA).

Ahmed J. Shah M. Miller A. Harper D. and Jafri M.N. 2007. A Vision-Based System for a UGV to Handle a Road Intersection", AAAI Twenty-Second Conference on Artificial Intelligence.

Nieuwenhuisen M. Droeschel D. Beul M. and Behnke S. 2014. Obstacle detection and navigation planning for autonomous micro aerial vehicles. Unmanned Aircraft Systems (ICUAS), 2014 International Conference on.

Quigley, M. Conley, K. Gerkey B.P. Faust J. Foote T. Leibs J. Wheeler R. and Ng A.Y. 2009. ROS: an open-source Robot Operating System. In ICRA Workshop on Open Source Software .

Eren F. Pe'eri S. Rzhanov Y. Thein W. M. and Celeikkol B. 2016. Optical Detector Array Design for Navigational Feedback Between Unmanned Underwater Vehicles (UUVs). IEEE Journal of Oceanic Engineering.

Faessler, M., Fontana, F., Forster, C., Mueggler, E., Pizzoli, M. and Scaramuzza, D. 2015, Autonomous, Vision-based Flight and Live Dense 3D Mapping with a Quadrotor Micro Aerial Vehicle. J. Field Robotics. doi: 10.1002/rob.21581

Liu S. Atia M.M. Karamat T.B. and Noureldin A. 2015. A LiDAR-Aided Indoor Navigation System for UGVs. Journal of Navigation.

Roberts D. Wolff R. Otto O. and Steed A. 2003. Constructing a Gazebo: Supporting Teamwork in a Tightly Coupled, Distributed Task in Virtual Reality. Presence.

CHAPTER 44

GRABBING OBJECTS THROUGH A ROBOTIC ARM AND HAND IN A SAFETY WAY

A. LÁZARO, J. MORIANO, L.M. BERGASA, R. BAREA and E. LOPEZ

Robesafe group, Universidad de Alcalá, alberto2991lazaro@gmail.com.

This paper presents a system for grabbing objects in an industrial environment through a robotic arm and a hand grip in a safety way including three dimensional obstacle avoidance. The environment information is provided by a vision system, based on a RGBD camera, through the Robotic Operating System (ROS) (Quigley, 2009). A comparison between the different solvers is also carried out for a typical industrial scenario and experimental results with an IRB120 robotic arm are also shown.

1 Introduction

Industrial robotics systems tend to autonomously be able to complete more complex tasks with the minimal human intervention. In last years, taking advantage of the newest object recognition techniques in computer vision, it is possible to automatically classify the detected objects in different groups.

In a working space with several different objects, once the vision system has classified every object in the group it belongs, it is possible to physically classify and grab all of them by means of a robotic arm and a robotic hand. In the proposed work, the trajectory calculation for grabbing and placing the target object, comparing different solvers, is faced. These trajectories are calculated avoiding collisions with the rest of detected objects in a three-dimensional way.

The employed vision system using a RGBD camera is explained in (Lázaro, 2016) and it includes object classification, pose and dimension estimation. The communication between the vision and robotic modules has been implemented taking advantage of ROS communication facilities.

The main goal of this paper is, by reconstructing a scene with the provided information, in which the objects, the robot arm and its environment are included, be able to interact with the different objects placed in the working scene in a safety way, avoiding damages in the robot and in the manipulated objects. Different planner strategies have been analyzed and compared to select the one that better fits with the required needs.

2 State of the art

Regarding to the topic of automatic objects classification employing robotic arm, several projects have been carried out. Hereafter we show some of the most important.

In (Szabo, 2012), (Bdiwi, 2012) and (Li, 2014) object classification and sorting is faced depending on the object color or shape. They also employ industrial robots for grabbing the different objects and placing them. These works are mainly focus in computer vision algorithm for detection and classification of the different objects.

Our proposal, as difference with the above works, employs different techniques for robotic arm planning and they are compared to choose the one that best fits our requirements. The selected planning techniques in ROS are employed to develop the robot manage system and to carried out comparison results between the different OMPL (Open Motion Planning Library).

Furthermore, the trajectories are calculated at the computer employing a robotic CAD model. This way the system is easily portable for a different robotic arm. Using OMPL planners and incorporating the detected objects to the scene, the path is calculated as a non-collision path.

3 Set up and robot Control

All the communication processes have been implemented by means of ROS, employing different nodes and topics. The vision system is set up as a unique ROS node while the robotic system is separated in several ROS nodes. The communication is established through a common topic where the vision system is publishing continuously.

To carry out the trajectories planning, an URDF model which includes the IRB 120 robotic arm shape and its configuration parameters, was developed using the robot CAD models as it is depicted in Fig. 1. The robot parameters, that are set up in the URDF file, define the axes and the joints movement limits.

Once the model is completed it is introduced into the "MoveIt!" software, where the robot environment is added with the provided information (object class, position, orientation and dimensions), so trajectories plan can be faced.

Due to several objects can be touched by the robot when some of them is grabbed, in the grabbing maneuvers the rest of the objects will be defined as "collision objects", which cannot collide with any part of the robot. A safety area is added around the collision object in order to increase the security of the maneuver.

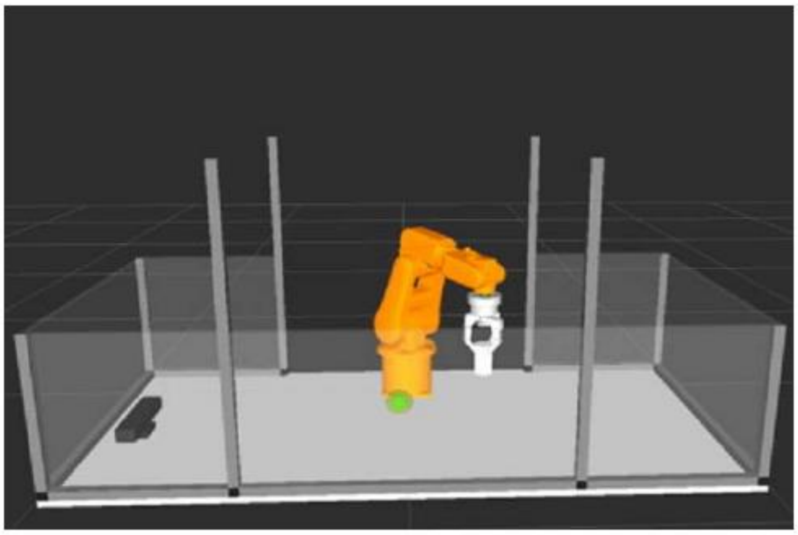


Fig. 1. Robot Model: the robot model and its working environment are shown. There are two different objects placed in the working scene: a can (cylinder) and a ball (sphere), which is depicted with a safety area around it

4 Robot Control

Once all the objects placed in the working scene are detected and the virtual environment is created, the robot recognizes the different objects according to the explained methods. The robot classifies the objects from the biggest to the smallest. Different grab processes have been developed to optimize the hand-grip taking into account the shape the objects have.
The robot uses the information of the vision system to estimate the height where the robotic hand should close.

After grabbing an object, it is placed in the assigned position according to the class it belongs as it can be seen in Fig. 2. The locations where the objects are deposited are filtered by the system, so once they are placed they disappear in the computer reconstruction.

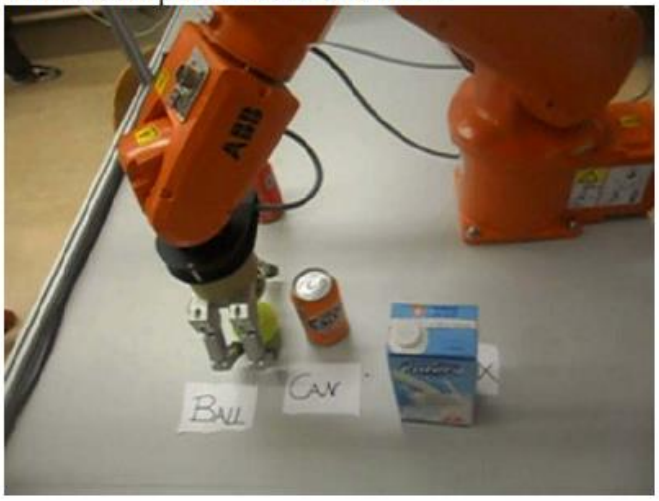


Fig. 2. Object sorting: Different objects are shown once the robot has sorted them

4.1 Collision avoidance

Due to the fact that the number of objects placed in the working scene is usually bigger than one, a planner which is able to find free-collision-paths is required. Different motion based planners (PRM planners and Tree based planners) (Hsu, 1999), (Sanchez, 2003), (Suçan, 2009) and (Muja, 2009), which are available in OMPL are compared in our scenario.

The maximum distance among nodes is the most important parameter. The dimensions of the objects and the whole distance trajectory should be taken into account to choose it properly. This distance should reach a compromise between processing time and safety. After several tests, the maximum distance that returns better results in terms of performance is 5 cm, which is the reason why this distance is the one that has been established to estimate the trajectories.

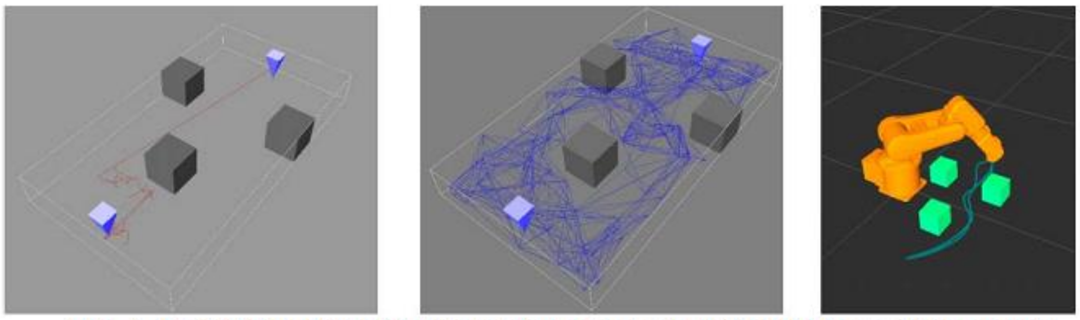


Fig. 3. In this figure the differences between probabilistic road-map and tree-based planning strategies are shown using KPIECE (left image) and PRM planner (center image). They are employed on an environment close to the real cases they will be faced. These

Simulations have been carried out thanks to OMPL application. In the right image the KPIECE planner is tested in the simulated environment and different paths were planned meeting the constraints and avoiding the three collision objects.

All the planners were tested solving the same case several times in which the robot needs to avoid three box obstacles (10x10x10 cm). The robot is forced to find a free three dimensional collision trajectory with a constrain of 10 cm in height. Average values of the obtained results are shown in Table 1 to choose the planner that has more interesting features. This analysis reveals that KPIECE and BKPIECE planners are the ones that stay closer to the desired requirements. Time costs are shown for the different tested planners in the next table:

Table 1. Planners times with 3 obstacles.

Planner	Average time (s)
BKPIECE	22.5
KPIECE	25.71
LBKPIECE	90
EST	90
PRM	45
PRM Star	180
RRT	60
RRT Connect	90
RRT Star	-
TRRT	-
SBL	36

In 3D environments, tree-based planners are really interesting because they can find a path without collisions in a small time, reducing the calculations. In the Fig. 3, the two planner categories (three-based and probabilistic) are compared applying them to the same problem.

KPIECE tree-based planner was selected as the better option because it is one of the fastest planners for our scenario and the paths are smooth enough to get our goals. Due to this planner carries out a discretization, it takes longer, compared with random ones, to define the better path.

4.2 Application Test

To test our whole application different objects such as balls, cans and boxes are used. The objects are detected by the vision system which provides to the robot the position where all of them are located so it can reach their position. According to the objects class they are grabbed and deposited by the robot in different places. The robotic system receives the path through an ROS–ABB socket.

Firstly, the robot moves to the initial position, which is high enough to allow the vision system captures properly the working scene. Afterwards the working scene simulation is updated according to the information it receives.

The robot goes to the location of the biggest object in the scene and grabs it setting the finger angles to the established position, then robot deposits the grabbed object in a predefined location according to the class it belongs. Finally, the robot goes back to the initial position out of the Kinect vision field and the process starts again in the same way. The working scene stops updating while the robot is in the Kinect vision field.

It should be delighted that through the feedback of robotic hand and arm states it is possible to get in real time a simulated view of the whole process thanks to RViz.

In Fig. 4 an example of the application is shown in simulation and with the real arm (IRB120 of ABB) and hand (BH8-262 of BarrettHand). In this figure it can be seen how the box is not included in the simulated scene due to the fact that it has been previously placed by the robot in its proper location employing a position filter the box is no longer included in the scene. Therefore, there are just two objects in the working scene: a can and a ball. The can is the target object for the robot in the shown case because it is the biggest one. Therefore, the ball is considered as collision object.

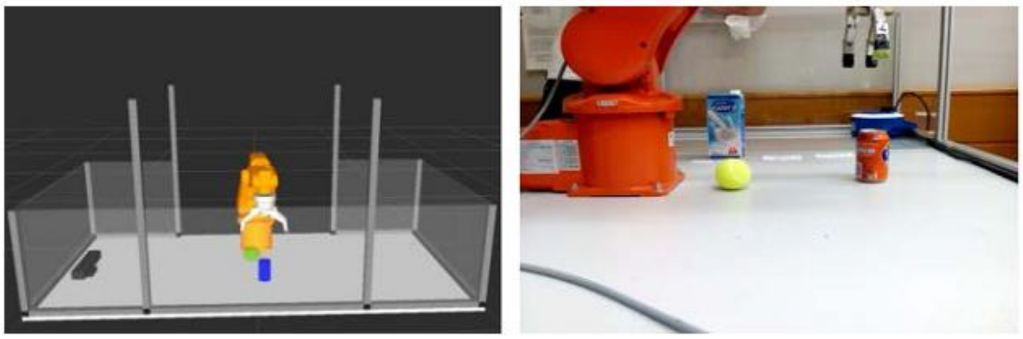


Fig. 4. Application example: The left figure shows the simulated working scene while the right one depicts the real working scene from the Kinect's point of view. Both images were taken at the same time.

5 Conclusions and future work

This paper has addressed the trajectory planning for object grabbing in a 3D free collision way, both in simulation and in a real industrial environment. To achieve the robot movement ensuring there is no collision an optimum movement planner, which is able to avoid hurdles, is used. Although it has been presented a real time solution for the addressed problem an enhancement will be done on working on reducing the processing time in order to make our system able to recognize and move objects faster than a human.

Acknowledgements

This work was supported in part by the MINECO SmartElderlyCar (TRA2015-70501-C2-1-R) project and by the RoboCity2030 III-CM project (S2013/MIT-2748) funded by Programas de I+D en la Comunidad de Madrid and cofunded by Structured Funds of the UE.

References

Bdiwi, M. 2012. Robot Control System with Integrated Vision / Force Feedback for Automated Sorting System. , pp.31–36.

Cruz, L. 2012. Kinect and RGBD images: Challenges and applications. Proceedings: 25th SIBGRAPI - Conference on Graphics, Patterns and Images Tutorials, SIBGRAPI-T 2012, pp.36–49.

Hsu, D. 1999. Path Planning in Expansive Configuration Spaces. International Journal of Computational Geometry & Applications, 09(04n05), pp.495–512.

Lázaro, A. 2016. 3D Object recognition and pose estimation using VFH descriptors. Open Conf. on Future Trends in Robotics (Robocity,'16').

Li, K. 2014. Robotic object manipulation with multilevel part-based model in RGB-D data. Proceedings - IEEE International Conference on Robotics and Automation, pp.3151–3156.

Muja, M. 2009. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. International Conference on Computer Vision Theory and Applications (VISAPP '09), pp.1–10.

Quigley, M. 2009. ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

Rusu, R.B. 2011. 3D is here: point cloud library. IEEE International Conference on Robotics and Automation, pp.1 – 4.

Sanchez. 2003. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. Robotics Research, pp.403–417.

Şucan, I. A. 2009. Kinodynamic motion planning by interior-exterior cell exploration. In Algorithmic Foundation of Robotics VIII (pp. 449-464). Springer Berlin Heidelberg.

Szabo, R. 2012. Automated colored object sorting application for robotic arms. 2012 10th International Symposium on Electronics and Telecommunications, ISETC 2012 - Conference Proceedings, pp.95–98.

UNIÓN EUROPEA
Fondos Estructurales
Invertimos en su futuro

Comunidad de Madrid

RoboCity2030.org

Leganés, 30 de junio de 2016

D. Eduardo Silles McLaney, Gestor de RoboCity2030,

HACE CONSTAR:

Que D. ALBERTO LÁZARO ENGUITA ha defendido en el congreso ROBOCITY16 - OPEN CONFERENCE ON FUTURE TRENDS IN ROBOTICS celebrado los días 26 y 27 de mayo de 2016 en la Universidad Politécnica de Madrid y organizado por el programa de la Comunidad de Madrid (cofinanciado por los Fondos Estructurales de la UE) RoboCity2030-III-M (S2013/MIT-2748) los siguientes artículos:

3D OBJECT RECOGNITION AND POSE ESTIMATION USING VFH DESCRIPTORS.
Autores: A. LÁZARO, J. MORIANO, L.M. BERGASA, R. BAREA y E. LOPEZ

UGV NAVIGATION IN ROS USING LIDAR 3D
Autores: A. LÁZARO, R. BAREA, L.M. BERGASA y E. LOPEZ

GRABBING OBJECTS THROUGH A ROBOTIC ARM AND HAND IN A SAFETY WAY
Autores: A. LÁZARO, J. MORIANO, L.M. BERGASA, R. BAREA y E. LOPEZ

[illegible]

Eduardo Silles McLaney
Gestor RoboCity2030

Real environment simulations

Real environment simulations and the obtained results are explained and shown along this book.

Attendance to conferences, lectures and courses.

Some conferences, lectures and courses that dealt with related fields of this work or other topics related with the master were attended. In the following lines the attended conferences lectures and courses will be enumerated:

Courses

- ROS seminar introduction
- Office work, displacements and visits.
- Export Control and Customs – Global Awareness
- Open Source Software Basics
- InfoSec – you make it
- Property Intellectual Rights Basics
- Innovation Workshop

Lectures

- Mobile Robotics Master Subject (No writing proof provided)

Conferences

- RoboCity 16 – Open Conference on Future Trends in Robotics

Universidad de Alcalá

DIRECCIÓN

ESCUELA POLITÉCNICA SUPERIOR
Campus Universitario
28871 Alcalá de Henares (Madrid)
Tlfno. 91 885 65 05 Fax 91 885 68 35
direccion.eps@uah.es

D. ANTONIO JOSÉ DE VICENTE RODRÍGUEZ, SECRETARIO DE LA ESCUELA POLITÉCNICA SUPERIOR DE LA UNIVERSIDAD DE ALCALÁ,

CERTIFICA:

Que **ALBERTO LÁZARO ENGUITA** con D.N.I. 03204184, de acuerdo con la documentación existente en esta Escuela, ha asistido al Seminario de Introducción a ROS celebrado el 8 y 9 de febrero de 2016.

Y para que sirva donde convenga a petición del interesado, firmo la presente certificación en Alcalá de Henares, a 16 de febrero de 2016.

[illegible]

UNIVERSIDAD DE ALCALÁ · DIRECCIÓN · ESCUELA POLITÉCNICA SUPERIOR ·

Antonio J. de Vicente Rodríguez
Secretario de la EPS

SIEMENS

www.siemens.com

Certificado (ID:3244281)

Alberto Lazaro Enguita

Trabajos en Oficinas, Desplazamientos/Visitas

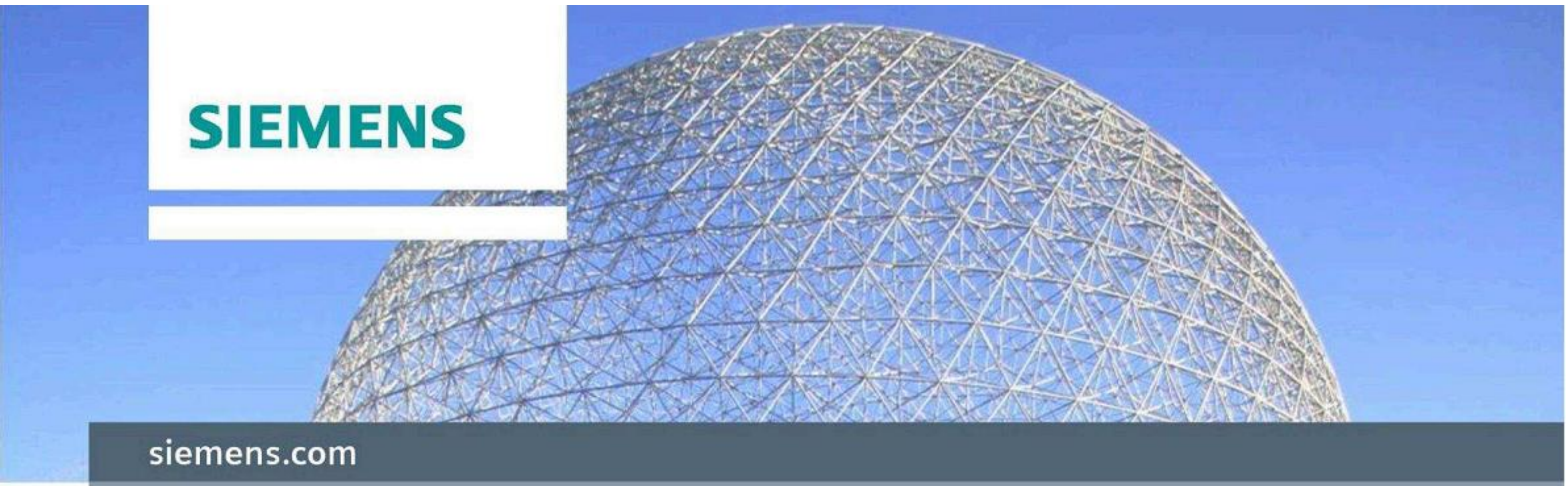
Ha superado con éxito la formación On-line de "Trabajos en Oficinas y Accidente en los Desplazamientos/Visitas". Duración 1 h

* Cuyo contenido ha sido:
- Riesgos asociados al trabajo en oficinas. Medidas preventivas
- Carga mental. Definición. Recomendaciones
- Medidas de emergencia. Introducción. Normas generales para la evacuación
- Seguridad en los desplazamientos. Riesgos y medidas preventivas: Plan de movilidad de Siemens
- Riesgos durante visitas a centros de clientes. Riesgos y medidas preventivas.
- Señalización de seguridad.

Tres Cantos (Madrid), 10/12/2015

Documento automáticamente generado y válido sin necesidad de firma.

José Bolaños Pulido
Servicio de Prevención Mancomunado



Certificate

Export Control and Customs - Global Awareness (Web Based Training)

Alberto Lazaro

(alberto.lazaro_enguita@siemens.com / Z003KSKE)

successfully completed the (approx.) 20 minutes online training Export Control and Customs - Global Awareness on Jan 4, 2016.

Training Content:
This training provides the participant with an overview of the area of Export Control and Customs (ECC). The web-based training will heighten the participants awareness of ECC topics relevant in the daily business in order to protect the companys and employees interest.

Furthermore important day-to-day situations are addressed like cross-border business trips, technology transfer and the handling of sending goods abroad and receiving of customs goods.

The participant will get to know possible risks, resulting of non-conformance with ECC regulations, including an introduction to the legal context.

Additionally the employee knows, where to find help in connection with the ECC regulations within the company.

This is an automatically generated document and therefore valid without signature.
Germany, Munich, Jan 4, 2016,

Heidi Anette Zötzl
Head of GS ECC
Export Control and Customs

SIEMENS

Confirmation of participation

Web Based Training Open Source Software Basics (WBTOSS)

Siemens Global Learning Campus and CT CSG SWI OSS herewith confirm that

Alberto Lazaro
MO MM R&D ES HW / Z003KSKE

successfully attended the Web-based Training

Open Source Software Basics

on Feb 8, 2016.

Training content:

- importance of Open Source Software (OSS),
- benefits and risks of using OSS,
- legal and license aspects of OSS usage,
- processes and Siemens support with OSS application.

This is an automatic generated document and valid without signature.
Munich, Germany, Feb 8, 2016

Kai-Holger Liebert	Oliver Fendt
Head of Global Learning Campus	Head Open Source Software
HR LE Global Learning Campus	CT CSG SWI OSS

SIEMENS

siemens.com

Certificate

InfoSec - you make it! (Web Based Training)

Alberto Lazaro

(alberto.lazaro_enguita@siemens.com / Z003KSKE)

has successfully completed the Security Online Training on Jun 21, 2016.

Training Content:

This training program focuses on important basic aspects of Information Security. Four typical forms of attack are shown and the participant gets to know the indicators for identifying these potential risks and learns how to ward off the attacker. The topics are as follows:

- Fraudulent e-mails: Why are fraudulent e-mails a threat to the information security of the company? How to identify those e-mails? How to handle them correctly?
- Fake phone calls: What are fake phone calls? Which data should be provided via phone? How to react when receiving an unusual/unexpected call?
- Social Media/Social Engineering: How can information in social networks be used by social engineers to gain the trust of an employee and then lead to access of critical information? What should one keep in mind when using social media?
- "There are many ways": How to contribute to the security of Siemens by sticking to the InfoSec basics, (e.g. the right classification of information and the respective handling of it; maintaining a secure working environment) and also help to ensure that no unauthorized person gets access to critical company information?

This is an automatically generated document and therefore valid without signature.
Germany, Munich, Jun 21, 2016

Ralf P. Thomas
Member of the Managing Board of Siemens AG
Responsible for IT

SIEMENS

Don **Víctor Ayuso García, Head de MO MM R&D ES** por el presente certifica que:

Alberto Lázaro Enguita

Ha participado en la sesión "Introducción a los Derechos de Propiedad Intelectual" impartida por Ismäel Palaci el 15 de junio de 2.016 con duración de una hora.

En Madrid a 22 de Junio de 2016

[illegible]

SIEMENS

Don **Víctor Ayuso García, Head de MO MM R&D ES** por el presente certifica que:

Alberto Lázaro Enguita

Ha participado en el Workshop de Innovación impartido por Fernando Fernández y Antonio Planells y celebrado en nuestra oficina de Tres Cantos el 29 de junio de 2.016 con una duración de 8 horas.

En Madrid a 30 de Junio de 2016

[illegible]

SIEMENS RAIL AUTOMATION, S.A.U.
Avda. Ronda de Europa, 5
28760 Tres Cantos
(Madrid)
C.I.F.: A-28512598

UNIÓN EUROPEA
Fondos Estructurales
Invertimos en su futuro

Comunidad
de Madrid

RoboCity2030.org

Leganés, 30 de junio de 2016

D. Eduardo Silles McLaney, Gestor de RoboCity2030,

HACE CONSTAR:

Que D. ALBERTO LÁZARO ENGUITA ha asistido al congreso ROBOCITY16 - OPEN CONFERENCE ON FUTURE TRENDS IN ROBOTICS celebrado los días 26 y 27 de mayo de 2016 en la Universidad Politécnica de Madrid y organizado por el programa de la Comunidad de Madrid (cofinanciado por los Fondos Estructurales de la UE) RoboCity2030-III-M (S2013/MIT-2748).

[illegible]

Eduardo Silles McLaney
Gestor RoboCity2030