



**OTC 24484**

## **A Robotics Framework for Planning the Offshore Robotizing Using Virtual Reality Techniques**

I. H. Santos, M. Galassi, Petrobras; P. From, Norwegian University of Life Sciences; L. Hsu, F. Lizarralde, R. Costa, G. Freitas, G. Motta-Ribeiro, T. Almeida-Antonio, F. Coutinho, Universidade Federal do Rio de Janeiro; A. Raposo, F. Carvalho, D. Medeiros, PUC-Rio de Janeiro

Copyright 2013, Offshore Technology Conference

This paper was prepared for presentation at the Offshore Technology Conference Brazil held in Rio de Janeiro, Brazil, 29–31 October 2013.

This paper was selected for presentation by an OTC program committee following review of information contained in an abstract submitted by the author(s). Contents of the paper have not been reviewed by the Offshore Technology Conference and are subject to correction by the author(s). The material does not necessarily reflect any position of the Offshore Technology Conference, its officers, or members. Electronic reproduction, distribution, or storage of any part of this paper without the written consent of the Offshore Technology Conference is prohibited. Permission to reproduce in print is restricted to an abstract of not more than 300 words; illustrations may not be copied. The abstract must contain conspicuous acknowledgment of OTC copyright.

### **Abstract**

The Oil & Gas industry has seen increasing costs of finding and extracting hydrocarbons, especially in remote locations, ultra-deep water reservoirs (400 m or deeper) or in hostile environments. Those new exploration frontiers have been increasing the production complexity and logistic costs. In such conditions, oil exploration feasibility depends on new technologies to optimize production efficiency.

One possible solution to this challenge is to increase the degree of automation in production units. New design concepts also consider the use of robotic devices in such scenarios. In this paper we present a robotics framework, SimUEP-Robotics (Robotics Simulator for Stationary Production Units - Unidades Estacionárias de Produção or UEPs, in Portuguese), aimed to enable planning the offshore platform robotizing using virtual reality techniques.

The SimUEP-Robotics is based on ROS (Robot Operating System), a middleware for exchanging messages between different devices and processes that cooperate to accomplish a robotics task. SimUEP-Robotics is designed concerning the offshore requirements and is a flexible framework that allows the inclusion of new robots and devices in a virtual operation scenario. This capability enables the robotization and automation of offshore facilities that gradually evolve, starting from a complete virtual scenario towards a complete robotic system operating on a real platform, progressively including real devices.

SimUEP-Robotics has an integrated Virtual Reality Engine (VR-Engine) specially tailored to provide realistic visualization of large offshore scene models in an immersive environment. The monitoring and management of remote operations of Stationary Production Units (SPU) is an activity that can also benefit by the usage of virtual reality scenarios due to the potential to reduce the complexity and difficulty in visualizing and validating simulations of operations performed by robots on a real SPU. The framework supports simultaneous simulation of multiple robots equipped with sensors and actuators like cameras, laser range finders and robotic manipulators. SimUEP-Robotics has also some specialized visualization tools like trajectory visualizer, ghostview robot animation, point-to-point measurement and a scenario editor that allows the user customize the target scenario accordingly. Through the use of those visualization tools it is possible, for example, to better understand the quality of the planned robot trajectory and propose new algorithms that can be further evaluated in the virtual environment. In conclusion, we argue that the validation process in an immersive virtual environment reduces risks and costs of real operation tests scenarios.

SimUEP-Robotics has also an integrated Robotics-Simulator which is responsible for taking care of task planning and execution based on the information of the virtual scenario provided by the VR-Engine. To illustrate the effectiveness of the framework, different robotics applications were developed. One is an underwater application that calculates the whole dynamics of an operated ROV to simulate and test complex ROV operations in deep waters, like the connection of a flowline to a Christmas tree. The other one represents a topside offshore platform scenario where different virtual robots, derived from real mechanisms like Motoman DIA10, Puma 560, Seekur and others, operates. Results obtained on a pick and place task demonstrate the benefits of the proposed robotics framework for offshore applications.

## Introduction

The vast majority of new Oil & Gas fields present high production complexity and logistic costs, in addition to several other challenges. Several companies explore oil in Alaska and Arctic regions where extreme temperature conditions restrict human intervention. In Brazil, exploration in the pre-salt layer represents a mark in the oil industry, breaking production records in deep and ultra deep waters. The pre-salt platforms operate far away from the coasts at distances that can exceed 300~km, increasing significantly transportation costs, logistics infrastructure and response time, especially in emergency situations. Under such conditions, oil exploration feasibility depends on new technologies to optimize production efficiency.

Robotics and automation can greatly improve the operational quality and productivity of offshore Oil & Gas installations (From 2010). Robots can work non-stop without the need for intervals and pauses, are less prone to errors, and also more reliable than human workers. This can increase the operation of the platform by reducing down-time and production stops, which are main concerns for the platform operators. Introducing robotic and autonomous solutions will also decrease costs related to commissioning and logistics, particularly by avoiding flying human operators to distant fields. In addition, unmanned platforms imply in reduced construction areas by removing the hotel services, restaurants, and other living facilities (S. O. Johnsen 2005) (S. O. Johnsen 2007).

Another highlighted point is the improvement of Health, Safety and Environment (HSE) conditions, as robots can replace humans in tasks performed in unhealthy, hazardous, or confined areas (Moghaddam 2012). Substituting human workers with robots in hazardous environments will reduce the risk of accidents, and also reduce the exposure of the workers to toxic gases and other damaging substances. Robots are believed to alleviate humans from dangerous tasks such as drilling and PIG operations, and time-consuming and repetitive tasks such as inspection, surveillance, and maintenance of the offshore platform. This will greatly improve the overall HSE as the operators can monitor and remotely control the operations from a safe location.

In the specific Brazilian case, the Oil & Gas industry is growing at a high pace, mainly due to recent discoveries of big oil reserves in the pre-salt layer off the Brazilian coast. These oil reservoirs are located farther away from the shore (300 km or more) and at depths of 5000 to 7000 m. These factors, especially the large distances, motivate the development of offshore production systems with a high degree of automation based on advanced robotics systems, eventually leading to unmanned platforms.

The next generation of offshore platforms will present high degree of automation. New design concepts consider the use of robotic systems to accomplish autonomous or teleoperated tasks on production plants, even though there are several technological challenges that should be overcome to have feasible robotics technology available offshore. In this paper, we propose an offshore robotics framework as a first step to enable the planning of offshore platform robotizing, using auxiliary virtual reality tools and techniques aiming to improve the understanding of the planned task throughout its execution in an immersive environment.

The proposed robotics framework is formulated concerning the offshore requirements. It is a flexible framework that allows rapid prototyping development of new applications through the addition of customized robots and devices in the target scenario. The proposed framework also simplifies the development and validation of the control algorithms by integrating virtual and real robots, sensors and different scenarios. This capability enables the robotization and automation of offshore facilities that gradually evolve, starting from a complete virtual scenario towards a complete robotic system operating on a real platform, progressively including real devices.

Virtual Reality (VR) technologies can improve the understanding of any engineering project. Particularly, VR environments and new techniques for 3D visualization and interaction are being used in engineering applications for recreating environments with a high degree of realism for design review, HSE training, etc. In the present work those techniques are used for visualizing the offshore platform robotizing activities.

Immersive visualization environments for robotics simulation have the potential to reduce the complexity and difficulty in visualizing and validating simulations of operations performed by robots on a real SPU. The visualization tools manage data and present visual information that enables professionals to observe and acquire a better understanding of problems, such as robot motion planning. Without visualization, it is hard to understand the quality of the solution found for the movement of the robot based only on simulation results. Validation reduces risks and costs of testing operations in real scenarios.

Besides the security and productivity gain due to a high degree of automation, the addition of teleoperated robots enables the execution of a non-programmed task. It is also possible to share workers with specific knowledge and skills across multiple platforms without the need for transportation. In that way, several different production fields may be attended by the same personal on a single day.

## Related Work

Different robotic frameworks and simulation platforms have been developed both for research purposes and industrial applications. However some important functionalities present in our proposed framework are absent in the existing ones. Before starting the development of SimUEP-Robotics we made a thoroughly comparison among the most advanced tools available in the market at that time (2011), such as Webots (Cyberbotics 1996), Microsoft Robotics Developer Studio

(Microsoft Corp. 2010), V-REP (Coppelia Robotics 2010) and ROS (Quigley 2009); and other industrial application tools such as RobotStudio (ABB 2001) and ROS industrial (ROS-Industrial™ Consortium 2012).

Webots is a fast prototyping and simulation software for mobile robotics that has been under development since 1996 by Cyberbotics. Complex robotic scenarios can be designed, with different robots that can be equipped with simulated sensors and actuators. The advantages are that the robot controller can be programmed with other development environments and that it is available on any operation system (OS). The main disadvantage is that it requires a paid license to have full capabilities and an annual renewal license for software update. The ability to build a customized scenario is a feature also presented in SimUEP-Robotics. In our solution the robot controller can also be programmed by external Robotics Simulators as long as they are compatible with ROS (Figure 1).

Robotics Developer Studio (RDS) is a Windows-based environment for robot control and simulation developed since 2006 by Microsoft. RDS is based on Concurrency and Coordination Runtime (CCR) and Decentralized Software Services (DSS) concepts. The first one is a .NET-based concurrent library implementation that manages asynchronous parallel tasks. The second is a lightweight state-oriented service model which allows multiple services to run on a single node achieving complex behaviors. The current version, RDS 4, is free. However, it only works with Windows. In our solution we also use ROS and its ecosystem's components.

V-REP is a virtual robot experimentation platform that was first public released in March 2010 by Coppelia Robotics. V-REP has an integrated development environment and is based on a distributed control architecture. In this way, objects can be controlled via an embedded script, a plugin, a ROS node, a remote API client or a custom solution. V-REP is open-source, but requires payment to have all functionalities. We also perceived the V-REP integration with ROS as a promised solution that we decided to pursue.

RobotStudio is a simulation and offline programming software that was developed by ABB Robotics. RobotStudio is built on top of the ABB VirtualController, which is an exact copy of the software that runs on the robots from ABB. In this way tasks like training can be performed without disturbing production. The program requires a paid license to have all functionalities and is restricted to ABB models.

The Robot Operating System (ROS) is a multiplatform meta OS which establishes a communication layer and some services and applications provided through a package framework. It is a software framework for robot software development, providing operating system-like functionality on a heterogeneous computer cluster. Willow Garage maintains the core system with an open-source free software license. An extensive worldwide community efficiently expands it. As an OS it provides hardware abstraction, low-level device control, user transparent inter process connection and communication based on the message passing paradigm. The process nodes are identified by its corresponding IP address, so network nodes integration is automatically implemented. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to a “robot framework”.

For robotics simulation, ROS ecosystem integrates the Gazebo project, which is a three dimensional multi robot simulator providing sensor data and physical interaction between objects. The Gazebo integration with ROS inspired very much our solution indeed. In fact, we decided to create the SimUEP-Robotics due to the lack of powerful VR tools and techniques in the ROS ecosystem.

When considering industrial applications, ROS-Industrial is a BSD-licensed software library which extends the capabilities of the ROS software to industrial applications. It is a recent initiative from Southwest Research Institute (SwRI), being supported by the ROS-Industrial™ Consortium (RIC).

## **SimUEP-Robotics Architecture**

In robotics applications, the classic programming approach of writing a specific code to each project is being modified. The new programming paradigm is the use of multiple reusable subtask programs to accomplish current and future tasks. This not only decreases the rework of researchers but also reduce the project development time.

In this context, a flexible architecture that permits rapid prototyping of different scenarios and applications is a valuable tool for the industry. Additionally, integrating virtual and real robots, environments, sensors and objects improves the development capacities.

As mentioned earlier ROS framework establishes a communication layer between different process and devices, providing broadcast topics with periodic messages and on demand services. Furthermore, ROS focuses on the message content, letting the sender and receiver be transparent to each other. In virtual and real environment integration scenarios, this transparency permits to change between resources without code reimplementations. To increase the portability of applications, standard messages to common required resources such as laser range finders, robot's joint state and odometry, are available,

Figure 1 shows the SIMUEP-Robotics architecture, which is component-based having among other components, the VR-Engine, which is responsible for the visualization of the simulation of actual operations performed with robots in offshore scenarios. Other components of the framework provide resources for manipulating different 3D input devices (spaceball, flystick, wiimote, tablets, etc), 3D interaction techniques, audio and video streaming, collaboration and a communication interface bridge to the ROS system, named ROS-Bridge.

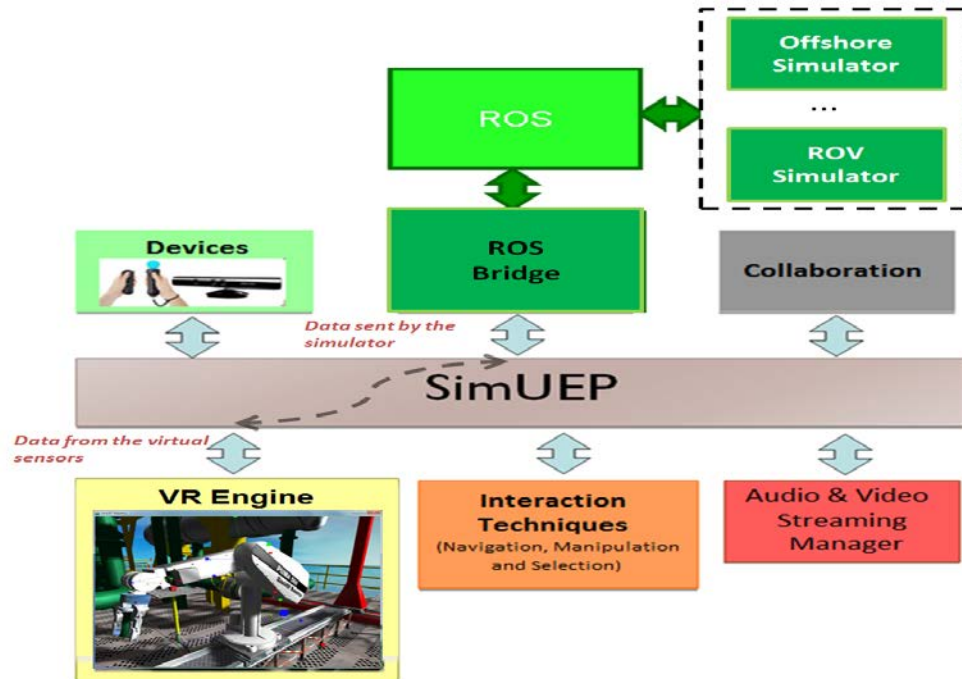


Figure 1. Diagram representing the SimUEP's component based architecture

In the following sections, we present further information about ROS Bridge and the Robotics-Simulator Modules currently available: Topside Offshore Simulator and Underwater ROV Simulator.

### ROS Bridge

The ROS Bridge is responsible for exchanging data between ROS compatible robotics simulators and the virtual robots created by the VR-Engine according to the target virtual scenario where the robotic task is being executed. The main goal of development of ROS Bridge is to make the data from the VR-Engine available to ROS compatible simulators. Data communication between the VR-Engine and ROS applications are controlled by ROS Nodes, which are processing units capable of establishing communication with each other through topics or services:

- ✓ **Topics:** structures that store messages in an asynchronous communication model (public/subscribe). The nodes publish and receive information in topics that are identified by names.
- ✓ **Services:** asynchronous communication with client/server architecture (one provider, multiple users).

The integration of ROS in SimUEP-Robotics allows the creation of a communication layer between the visualization of a virtual scene and the simulation of real robot's tasks. The exchanged messages are similar to those used to control real robots.

The SIMUEP component based approach enables the creation of new "bridges" to other robot frameworks, like Rock (ROCK 2012) that is in our near future development plans.

### Robotics-Simulator Modules

At present, the SimUEP-Robotics has two robotics simulator modules implemented allowing us to instantiate two different applications: one for topside offshore operations and another for sub-sea engineering, proving the effectiveness and value of the proposed framework. For the topside, the robot simulation strategy depends on the used virtual device and the task that should be planned. As a proof-of-concept we have developed a pick-and-place task in a typical offshore scenario (Figure 14). In the second application, the ROV simulator module implements the dynamic motion equations for an underwater ROV vehicle from Liu Hsu *et. al.* (2000). It is also clear that the SimUEP-Robotics can be extended with other ROS compatible robotic simulators addressing other operation problems.

The robotics-simulator modules implementation integrate seamlessly some open-source community maintained ROS packages according to its needs. This integration is driven by the definition of the robots on the Unified Robot Description File (URDF) associate3d to its virtual robot used in the target scenario. The URDF is a XML file format that provides information about robots and associated sensors (*e.g.*, laser, camera), trough joints and links definition at different levels such as kinematic, dynamic and visual. A simple URDF example is shown in Table 2 with a robot containing a revolute and a prismatic joint. Sensors can be incorporated using the Gazebo interface, which will be described later.

**Table 1. Relations of Sensor & Actuators with Ros Messages.**

Sensor	Data Description	ROS Message Type
Proximity	Name of the link being reached	std_msgs/String
Collision	Array of 3D intersection points	sensor_msgs/PointCloud2
LaserScan	Array of distances captured	sensor_msgs/LaserScan
Image	2D Image	sensor_msgs/Image
Joint	Set of joints values	sensor_msgs/JointState

Actuator	Data Description	ROS Message Type
Pose	3D position and orientation	geometry_msgs/PoseStamped
Joint	Set of joints values	sensor_msgs/JointState

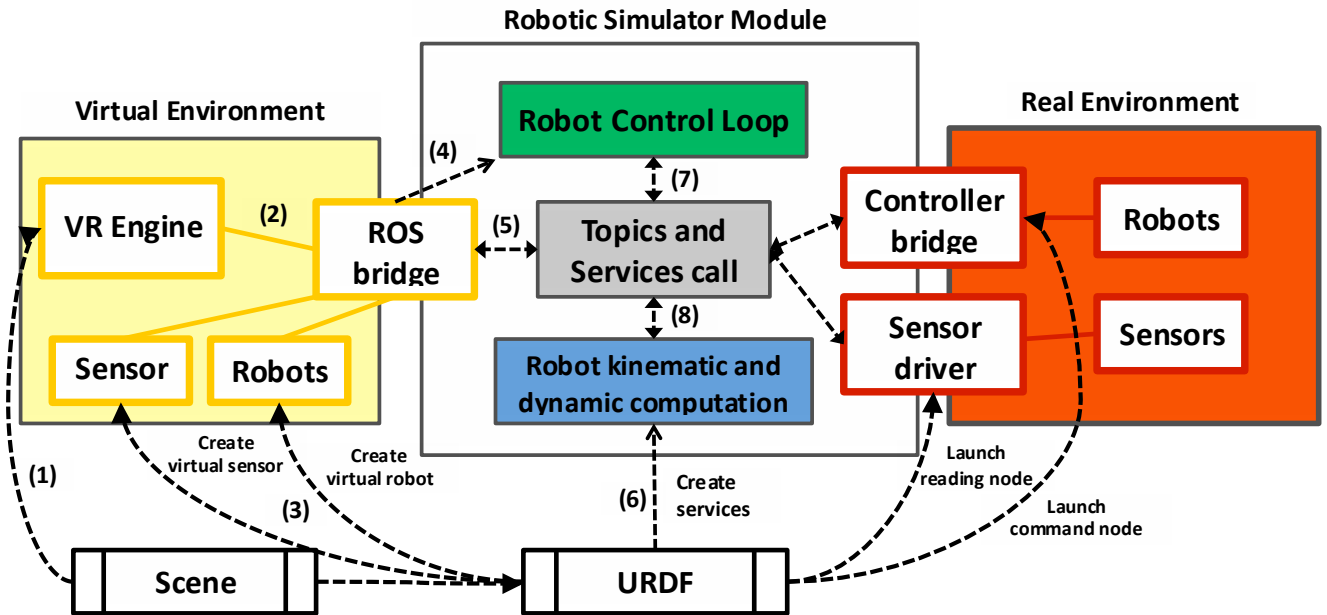
Upon reading the robot URDF files in the scene the Robotics-Simulator configures the necessary standard messages to interact with sensors and robots (see Table 1). In addition, it uses different ROS integrated libraries and packages to perform tasks like: navigate with mobile robots (navigation stack), transform data through different coordinate systems (TF package), build 3D maps (OctoMap) and process images (OpenCV).

After selecting one of the SimUEP-Robotics available applications (instantiated by selecting an specific Robotics-Simulator module according to the addressed task), the VR-Engine reads the scene description defined by the user and the related URDF files to create the virtual environment together with the visual representations of the robots (

Figure 2, steps 1 to 3). The ROS-Bridge is responsible for creating a ROS’s root node and exchanging information through its associated ROS topics and services. Some of those created topics are the sensor’s camera and laser information for each virtual sensor instantiated in the virtual scenario according to its URDF definition. After that, the VR-Engine starts the corresponding simulation module that communicates with robots controllers and read sensor data (

Figure 2, steps 4 to 7). Besides that, it also starts services to calculate robot's dynamics and kinematic under request of other nodes (see

Figure 2, step 8). The robot's behavior in virtual environment (yellow side) is the same in the real environment (red side). In fact, we have conducted some experiments in the university’s lab confirming this assumption using the same Robot Simulator Module.



**Figure 2. Diagram representing the Robotics Modules interaction with SimUEP’s ROS Bridge**

In the following section, we describe the SimUEP-Robotics Virtual Environment main features and VR tools available to Robotics-Simulator modules.

## Virtual Environment

In any instantiated SimUEP-Robotics application, the virtual environment is loaded from a scene description file. This file contains a scene description of the target scenario. A scenario is basically composed by an observer, environment objects, lights and robots.

SimUEP-Robotics uses the game engine Unity3D as its graphic engine component. Unity3D is a tool for developing games created by Unity Technologies (Unity3D s.d.). This engine was chosen due to resources and graphic quality it provides, enabling the creation of realistic environments.

### 3D Scene Description

The scene description is a XML file that describes the target scenario. It has a hierarchical structure describing all items in the environment. The first item of this file is *displaySettings*, which defines the properties of virtual camera (near and far planes) and output settings to enable visualization modes for immersive systems, e.g., CAVEs, PowerWalls and other projection systems. Global settings of the physics engine, such as update frequency and gravity vector are specified in item *physicsSettings*. The user visualizes the scene by specifying the *observer* item, which has properties such as position, orientation, height and speed. All the lights are defined inside the item *lights*. They can be of type point, spot and directional. Common attributes are position, orientation, color, intensity and type of shadow. The scene objects are specified by *envObjects* items. For each *envObject* the following attributes are required: position, orientation, bounding box, mesh, and some physical properties, such as mass, inertia matrix and material information (friction, bounciness, etc).

```

<robots prefix="SimUEP">
  <services>
    <draw3DLines name="drawPathService"/>
    <enableSensor name="enableSensorService"/>
    <cameraInfo name="setCameraInfoService"/>
  </services>
  <robot name="Puma560">
    <urdf fileName="Data/Urdfs/Puma_560.urdf">
    <transform position="0.0 0.0 -0.363" rotation="0 0 0" scale="1 1 1">
    <actuators>
      <joint>
        <topicName name="jointState"/>
      </joint>
    </actuators>
    <sensors>
      <joint>
        <topicName name="jointStateReal"/>
      </joint>
      <collision>
        <topicName name="collision">
        <precision mode="high" defaultValue="1.5"/>
      </collision>
      <proximity>
        <topicName name="proximity"/>
        <range value="0.10">
      </proximity>
    </sensors>
  </robot>
</robots>

```

Figure 3. Example of a simplified robot URDF file.

Each robot in the scenario file is defined by a URDF, a position and a set of actuators and sensors (Figure 3). Some of these sensors are also defined in the URDF, for example, cameras and lasers. For each sensor and actuator, it is necessary to define a topic name to identify the associated ROS message.

### Virtual Robotic Systems

We have implemented six different ROS compatible robots, namely, Motoman DIA10, Motoman SDA10, Grant Crane, Seekur and a generic ROV (see Figure 4). Any robot can be instantiated in any application and positioned (position, attitude and scale) defined in the 3D scene file of the target scenario.

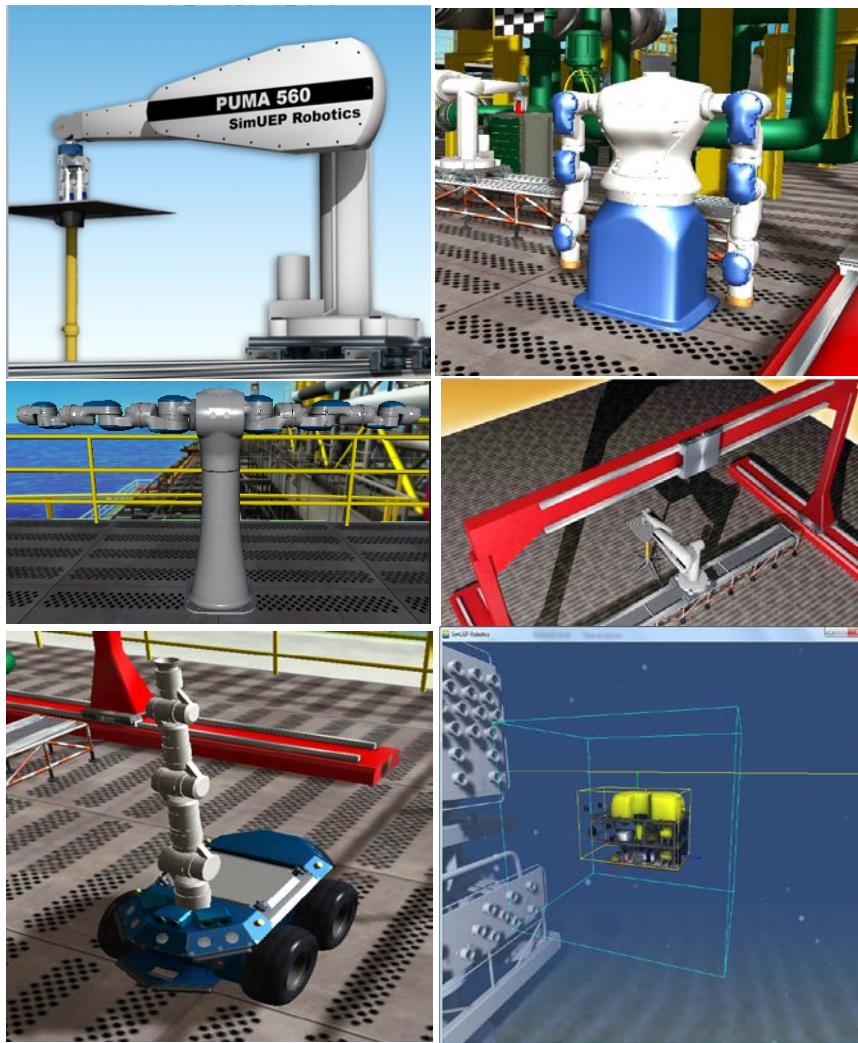


Figure 4. Six Robots available in SimUEP-Robotics (Puma560, Motoman DIA10, MotoMan SDA10, Grant Crane, SeekUR and ROV)

**Actuators**

The actuators are used to modify some positional aspects of the robots. There are two actuators available: the first one is based on the use of joint values; and the second on the absolute position of the whole robot. Table 1, previously presented, shows the relationship between ROS messages and sensors and actuators available in the SimUEP-Robotics.

**Sensors**

Sensors are used for a robot to perceive the environment and, through them, being able to react to its surroundings. Each robot may be composed of one or more types of sensors. In SimUEP Robotics, four different types of sensors are available: lasers, cameras, proximity sensors, and collision sensors.

<pre> &lt;!-- Camera --&gt;   &lt;gazebo reference="linkCamera1"&gt;     &lt;sensor:camera name="linkCamera1"&gt;       &lt;imageSize&gt;512 512&lt;/imageSize&gt;       &lt;imageFormat&gt;R8G8B8&lt;/imageFormat&gt;       &lt;hfov&gt;0.7853982&lt;/hfov&gt;       &lt;nearClip&gt;0.01&lt;/nearClip&gt;       &lt;farClip&gt;4.5&lt;/farClip&gt;       &lt;updateRate&gt;0.25&lt;/updateRate&gt;     &lt;/sensor:camera&gt;   &lt;/gazebo&gt; </pre>	<pre> &lt;!-- Laser --&gt;   &lt;gazebo reference="linkLaser1"&gt;     &lt;sensor:ray name="linkLaser1"&gt;       &lt;rayCount&gt;512&lt;/rayCount&gt;       &lt;minAngle&gt;-0.6981317&lt;/minAngle&gt;       &lt;maxAngle&gt;0.6981317&lt;/maxAngle&gt;       &lt;minRange&gt;0.1&lt;/minRange&gt;       &lt;maxRange&gt;3.0&lt;/maxRange&gt;       &lt;resRange&gt;0.03490658&lt;/resRange&gt;       &lt;updateRate&gt;0.2&lt;/updateRate&gt;     &lt;/sensor:ray&gt;   &lt;/gazebo&gt; </pre>
--	---

Figure 5. (a) Camera and (b) Laser sensors description in URDF file.

Camera sensors are able to generate images from the virtual environment. A robot can have multiple camera sensors that are normally positioned along its links. This positioning allows the visualization of different views of the same robot at the same time. Each camera is described (within the URDF file) following the description format SDF (from Gazebo Project) and has attributes such as image size, lens aperture, pixel format, among others (Figure 5a). The Virtual Environment camera sensor component is responsible for capturing data from the camera and sending them to the Robotics-Simulator Module through the associated ROS topic defined in the robot URDF file. A feature that generates a visual representation of these cameras is available (Figure 6a).

Robots can also use sensors like lasers. These are used to recognize the environment through a one-dimensional scan. As a result of this scan a vector containing values corresponding to the distance between the laser and the obstacle of the corresponding scenario is obtained. Figure 6b shows the visual representation of two lasers in SimUEP, as well as images that represent the distance vector captured by each one. The laser sensor is also defined in the robot URDF file. Its parameters contain information such as the initial position of the sensor, resolution and maximum and minimum angles, topic name of the message, as well as parameters that control their display in virtual scene (Figure 5b).

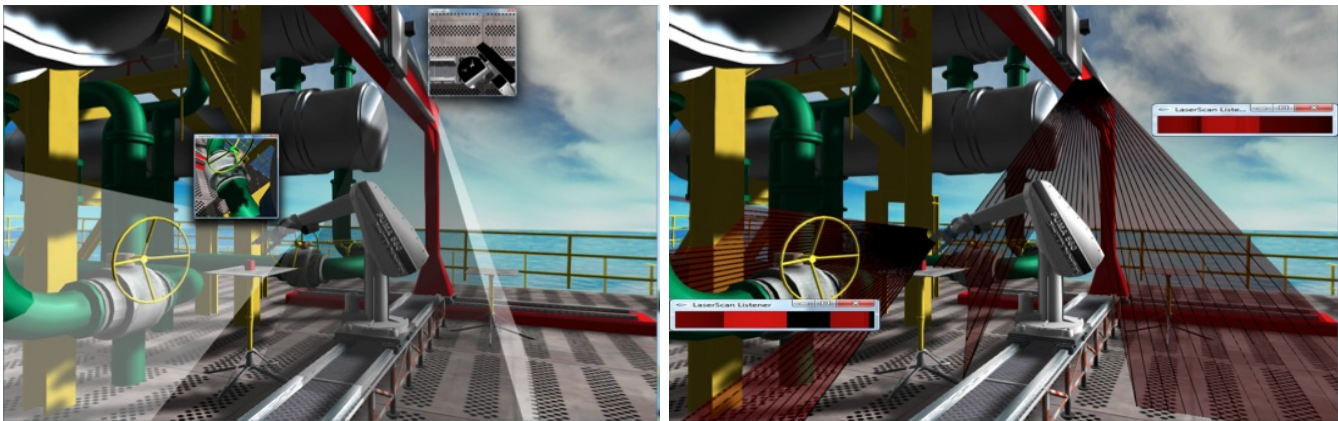


Figure 6. (a) Camera and (b) Laserscan visualization in SimUEP-Robotics.

Collision sensors are sensors that allow the robot to identify when an object is close to some of its physical components (links). This allows the robot to be able to avoid obstacles, and enable the measurement of these obstacles to a certain part of the robot. This sensor is implemented by using bounding boxes. When any object penetrates a bounding box, all interception points are informed (Figure 7) using the message *sensor\_msgs/PointCloud2*. The description of this sensor is present in the scene description file and it has a parameter for the precision.



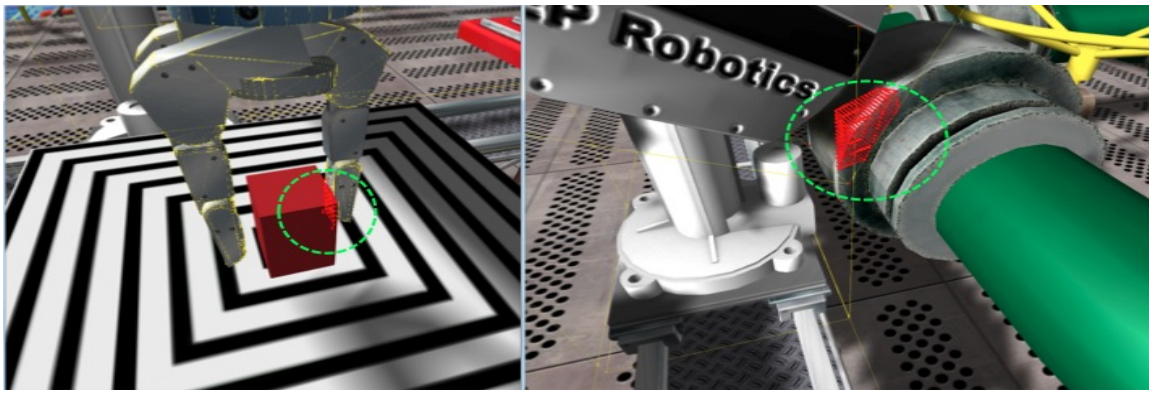


Figure 7. Collision Sensor, the red dots are the intersections points detected in the link's bounding box.

Proximity sensors are similar to collision sensors; however, they only inform the name of the link hit. Basically, when something enters a bounding of this sensor, a message is sent to ROS. Figure 8 shows a robot's proximity sensor (green box).

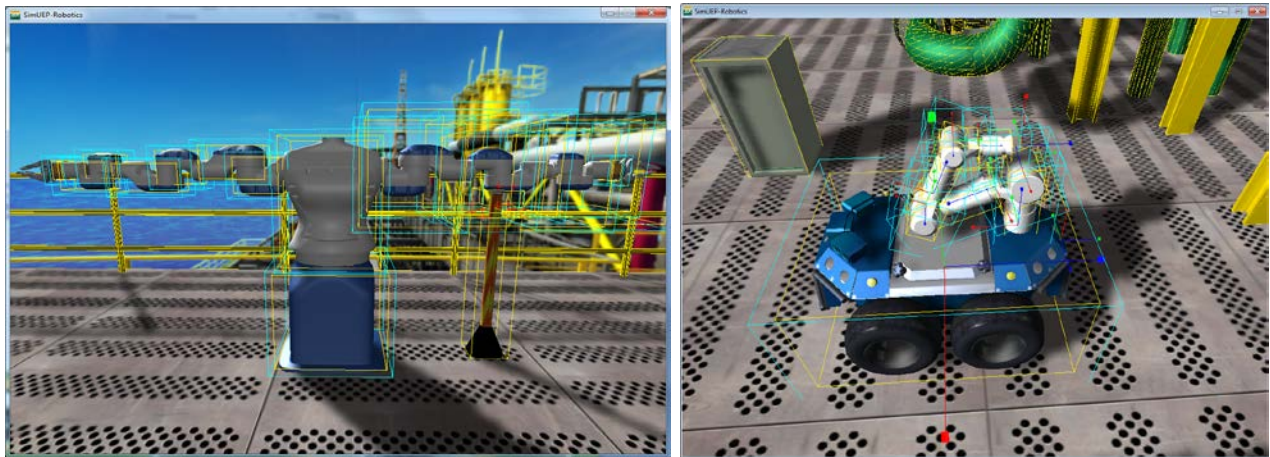


Figure 8. Proximity Sensors visualization using green boxes.

**Visualization Tools**

SimUEP has a tool that allows the creation of measurements between points (Figure 9 **Error! Reference source not found.**). The creation of a measurement is done by selecting two points, which are connected visually by a line segment. Each measurement has a label which displays the distance between the points. Moreover, the measurements have a dynamic behavior. Thus, if the position of the objects (comprising measurement) changes the measurement is automatically recalculated.

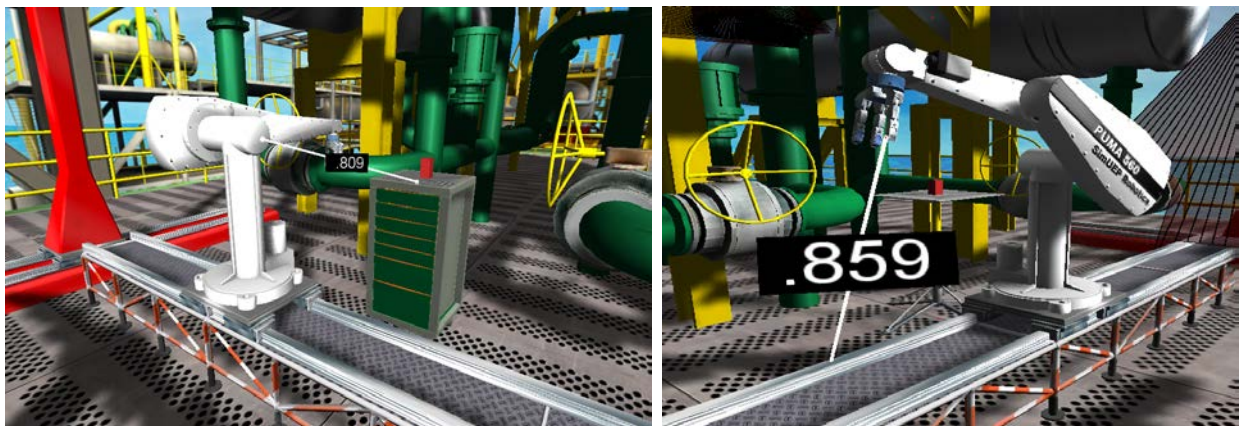


Figure 9. Measurement Tool.

Another important aspect is the visualization support to evaluate the simulation results as well as to validate different offshore scenarios composed of robots, valves and other equipment defined in the scene description file. In this scope, the following visualization data can be mentioned: Bounding Boxes and Axis Joints (Figure 10).

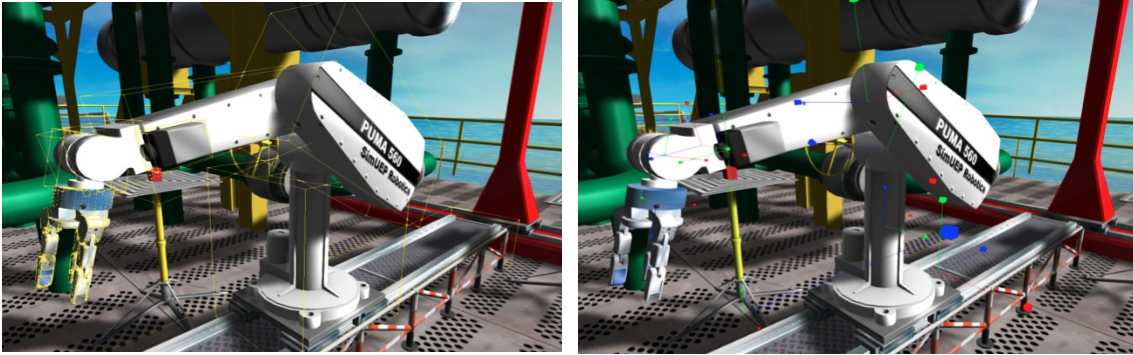


Figure 10. Bounding Box and Axis features.

The SimUEP-Robotics has two extra features, Ghostview and 3D Point Trajectory, which allow the visualization of the entire motion path performed by robots. In Ghostview the viewer captures successive snapshots of the robot along the trajectory movement. Thus, it may be used to visualize the robot at different points in the simulation. The later feature, Point Trajectory, allows visualization of the trajectory of a point, such as a joint or a link. Figure 11 presents examples of visualization of trajectories through functionalities Ghostview and 3D Point Trajectory, respectively.

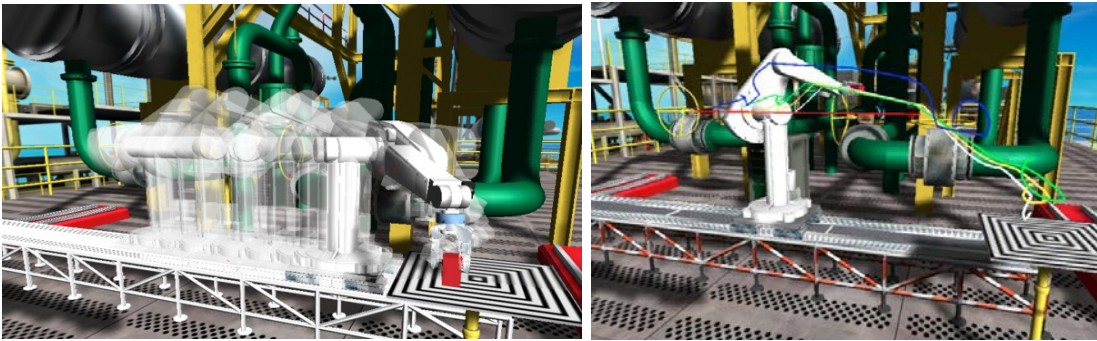


Figure 11. Robot Trajectory visualization modes: SnapShot3D and Lines3D.

The SimUEP-Robotics can be used in desktop environments as well as in immersive environments such as CAVE (Figure 13), PowerWalls and other projection systems. One can navigate in the environment by means of specific controls such as 3D trackers.

In the next section we describe the TopSide Robotics Simulator main characteristics and algorithms used for trajectory planning with avoidance obstacle.

### Topside Robotics Simulator Module Cognition and Control

The robot cognition is based on image and point cloud information using ROS packages and integrated external libraries. Some examples are the OpenCV (Bradski 2000) library for image processing, TF package to coordinate system transform tracking and the 3D mapping library OctoMap (Hornung 2013).

#### Image Processing

OpenCV is used for camera calibration, distance and size measurement and featuring recognition. The distance measurement make use of a disparity image (*stereo\_msgs/DisparityImage*) provided by OpenCV tools fed with images (*sensor\_msgs/Image*) of two cameras disposed side by side with known relative configuration, cameras calibration and extrinsic information (*sensor\_msgs/CameraInfo*). This distance can feed the image space to Cartesian space conversion of any camera with known position, also others than the two already mentioned, using the frame coordinate transformations provided by the TF package. To measure object size other image processing are performed to recognize features like valves, square pattern surfaces and colored objects.

The feature recognition uses combinations of OpenCV functions taking advantage of pre-known information of the scenario. Images could be converted between different color models, filtered, blurred and mixed. Objects can be selected by color, area, shape and surface pattern. A specific case of use on offshore robotics is a valve identification procedure. For this consider a yellow circle valve commonly present in Brazilian platforms. Recognizing this valve consist in a yellow identification on a hue, value and saturation, HSV, color model (*cvtColor*, code=*CV\_RGB2HSV*), find contours (*findContours*) and then fit a circle (*fitEllipse*). Figure 15 shows steps of this image processing.

#### Point Cloud and Mapping

Information provided as point cloud (*sensor\_msgs/PointCloud2*) is used to produce two different types of maps. The first one is used for obstacle avoidance, i.e., the filtered point cloud to take out the robot and out-of-workspace points. The second is an Octomap, a probabilistic map that extend to 3D the concept of 2D occupancy grid representing the environment as occupied, free and on an unknown state voxels (a 3D pixel) (Hornung 2013), se Figure 16 for an example.

The map in Figure 16 is a representation of the scene in Figure 14, produced by a laser range finder attached to the robot end-effector scanning the environment. The sensor readings (*sensor\_msgs/LaserScan*) are continuously grouped on a buffer and transmitted by a service as point clouds when requested. These clouds are transmitted periodically by another ROS node and then used by the mapping algorithm. All the data is transformed to an inertial coordinate system using TF and the kinematic chain provided in the URDF file.

### Trajectory Planning with Obstacle Avoidance

The idea of using robots in dynamic and partially unknown environments is only possible by ensuring that it does not collide with anything or anyone. In this way, accidents can be prevented guaranteeing the safety of the operators and the completeness of the task.

One way to ensure that the robot will not harm the environment around it is by making it able to accomplish tasks by moving the end-effector through collision-free trajectories while the joints avoid obstacles without breaking their mechanical limits. This is in many cases only be possible if the robot has more degrees of freedom (DoF) than the task requires, so it can use the remaining DoF for obstacle avoidance. When the robot has extra DoF, they are called redundant robots.

This obstacle avoidance is achieved by obtaining joint velocities  $u$  from a control law that ensures desired Cartesian linear and angular velocities to the end-effector with additional terms that could satisfy possible joints restrictions of movement. This leads to the following control law:

$$u = J_w^\dagger \begin{bmatrix} K_p E_p + \dot{x}_d + V_r \\ K_o E_o + \omega_d \end{bmatrix} + \alpha (I - J_w^\dagger J) W^{-1} \nabla H(\theta), \quad (1)$$

where  $J_w^\dagger$  is the weighted pseudoinverse of the Jacobian  $J$  which satisfies  $J J_w^\dagger = I$  (Bjerkeng, et al. 2011),  $K_p$  and  $K_o$  are the position and orientation gains,  $E_p$  and  $E_o$  are the position and orientation errors (Siciliano, et al. 2009),  $\dot{x}_d$  is the derivative of the desired position,  $V_r$  is a repulsive vector pointing in the opposite direction of the obstacle (Flacco, et al. 2012),  $\omega_d$  is the desired angular velocity,  $\alpha$  is a negative constant,  $W^{-1}$  is a diagonal positive definite weighing matrix (Bjerkeng, et al. 2011) and  $\nabla H(\theta)$  is the subtask to be performed.

The first term makes the end-effector move from a starting position to the goal, while avoiding obstacles and the second term uses the null space of the Jacobian to satisfy an addition subtask without modifying the position and orientation of the end-effector. The chosen subtask, in this case, uses repulsive artificial potential fields ( $U_{r,i}(\theta)$ ) around the obstacles, repelling the robot from them, such as (Siciliano, et al. 2009):

$$U_{r,i}(\theta) = \begin{cases} \frac{k_{r,i}}{\gamma} \left( \frac{1}{\eta_i(\theta)} - \frac{1}{\eta_{0,i}} \right)^\gamma, & \eta_i(\theta) \leq \eta_{0,i}, \\ 0, & \eta_i(\theta) \geq \eta_{0,i}, \end{cases} \quad (2)$$

where  $k_{r,i} > 0$ ,  $\eta_i(\theta)$  is the minimum distance between a point of interest ( $r$ ) on the robot and one obstacle ( $i$ ),  $\eta_{0,i}$  is the range of influence of the obstacle (the maximum distance in which the robot senses the obstacle and is a scaling factor (a typical choice is 2). To simplify, we can choose the joints to be the points of interest of the robot.

The repulsive potential is zero outside and positive inside the range of influence and tends to infinity as the distance approaches zero. To reduce the computational effort, the obstacles are considered directly as point clouds obtained from sensors. This way each point can be avoided separately. The subtask is then formed by the contribution of all points near each of the joints of the robot, given by:

$$\nabla H(\theta) = \sum_{r,i}^{n,o} \nabla U_{r,i}(\theta), \quad (3)$$

where  $n$  is the number of points of interest on the robot,  $o$  is the number of obstacles and  $\nabla U_{r,i}(\theta)$  is the gradient of the repulsive potential (Siciliano, et al. 2009) that can be defined as:

$$\nabla U_{r,i}(\theta) = \frac{k_{r,i}}{\eta_i^2(\theta)} \left( \frac{1}{\eta_i(\theta)} - \frac{1}{\eta_{0,i}} \right) \nabla \eta_i(\theta), \quad (4)$$

using  $\gamma = 2$ . Note that by using this approach the task is prioritized, so collisions can in principle happen. But after some simulation and practical tests this wasn't a problem and the robot could safely complete all his tasks. Some results are presented in the next section.

### Case Study 1: Pick-and-place task in a virtual offshore scenario

In order to illustrate the utilization of the proposed framework, we used a pick-and-place task in an offshore scenario. For this class of tasks the robot is supposed to grasp an object and move it from one location to another. The robot should then follow a collision free path, preserving the transported object and itself. To accomplish this, the task is performed in four steps:

1. Environment mapping;
2. Object grasping;
3. Path planning;
4. Path following.

These subdivisions of the pick-and-place task are good examples of the different capabilities that are necessary for a robotics system to operate in cluttered environments like offshore platforms. Mapping is used for robot localization and motion planning. Grasping is not only important for the load transport task, but in almost any operation demanding interaction between the robot and the process, like valve manipulation and pig launch/recovery. A collision free path planning is needed for every task demanding robot displacement in the presence of obstacles.

In the simulated example, the robot must follow consecutive desired positions to transport a red rectangular prism between two tables placed at the platform section showed in Figure 14. The four steps that compose the task are represented in Figure 17.

The mapping is performed using laser ranger finders like the one showed in Figure 17a. The resulted map is shown in Figure 16. Figure 17b displays the sensed contact between the gripper fingers and the object. This information is transmitted as a point cloud, simulating a proximity sensor, and used to ensure that the gripper reaches the object during closure.

The planned paths for each robot joint are represented by the colored lines on Figure 17c; these are calculated according to the previous section employing the map generated during the first step. Finally, Figure 17d shows the robot following the planned path with Snapshot3D visualization.

Results showed hereafter are obtained using properly tuned controllers and range of influence equals to 0.30 m. For comparison purposes, the same task is performed with and without the proposed collision avoidance control strategy. Figure 18 shows the distance from each link to its corresponding closest obstacle and Figure 19 presents the position errors of the end effector.

Using the proposed control strategy, the links are generally farther from the obstacles than without collision avoidance. As the controller attempts to avoid collision between obstacles and all robot links simultaneously, some movements can make a part of the robot approach an obstacle. However, except for the initial position, each link has a distance to the closest obstacle greater than approximately 0.10 m. Without avoidance, the minimum distance from the robot to an obstacle reaches approximately 0.065 m.

In Figure 19, one can notice that obstacle avoidance doesn't intervene with the task execution, as the error converges to zero in both graphs. The peaks in the position error happen when the point denoting the desired position changes. While these points are the same in both tests, the path that the robot follows in each one is different, as expected. With obstacle avoidance, a faster convergence occurs.

### Case Study 2: ROV navigation

The ROV simulator module implements the dynamics equation for an underwater ROV vehicle. Thus, SimUEP-Robotics can be used to animate the ROV motion in 3D. The main module functionalities are: 6 DoF ROV model simulation, xml-based ROV parameters configuration, static or dynamic thrusters model. The general dynamic model allows the simulation of practically any commercial ROV actuated by thrusters with speed control. Multiple ROVs can be simulated concurrently. An almost real time simulation allows the use of this module for operator training as an important application. This module was developed with a modular concept, which makes maintenance and future extension very simple. In the model implementation, the function used to describe hydrodynamic coefficients, thrusters characteristics, etc., are represented using lookup tables, since in general these parameters are obtained from experimental test in wind tunnel. The solution of the Euler-Lagrange differential equations, which governs the ROV motion, is calculated using integration methods such as Euler or Runge-Kutta. Other integration algorithm could be easily added without rewriting the ROV model.

The ROV model which is now embedded in the SimUEP-robotics opens possibilities to test different type of localization systems for dynamic positioning, advanced control strategies and autonomous navigation algorithm (Figure 12).

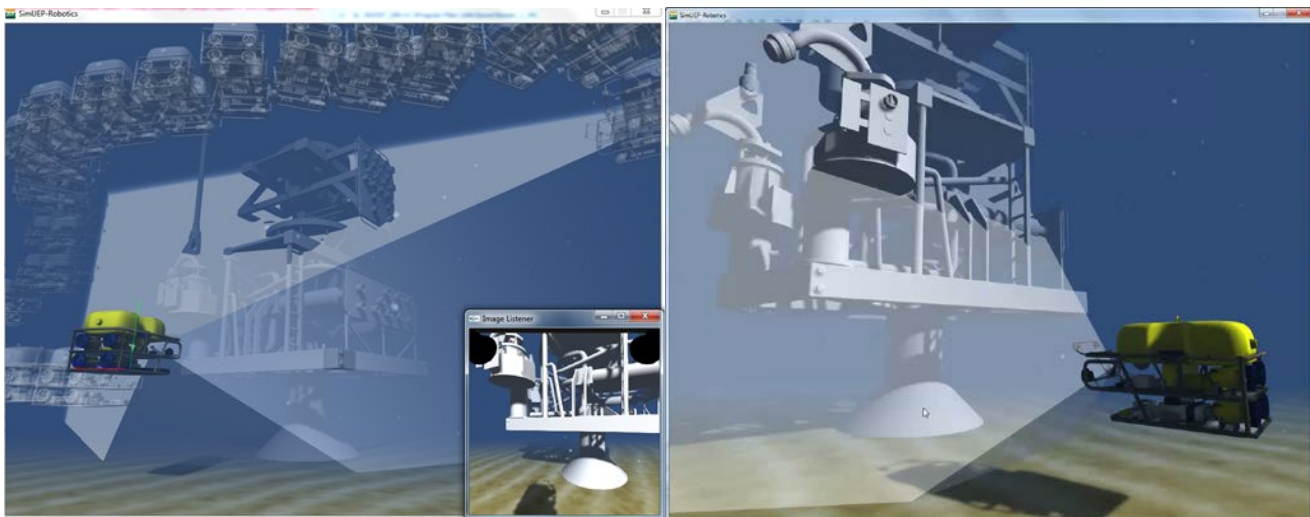


Figure 12. ROV navigation in a virtual Christmas tree scenario.

## Conclusion

Hostile environments are seen as one of the greatest challenges for oil & gas companies developing new fields. The use of robots in these situations is foreseen as a great opportunity for the robotics industry. Thus a lot of effort and investments are being pushed on the development of specialized robots for those environments. This development can be extremely improved if proper tools and methodologies are used.

Tools such as simulators and immersive visualization is a very promising development strategy, providing an environment for testing and visual debugging. The ability to compare different simulations with the support of different visualization tools can help in the interpretation of virtual simulated tasks allowing the development of new robots and, eventually, more appropriate offshore scenarios and tools to accomplish the proposed tasks. Furthermore, development methodologies that focus on the reuse of resources are extremely important to give flexibility in the process of developing new products.

The work presented here is based on the above mentioned development strategies by creating flexible development environment to support the modeling and visualization of the simulation of actual operations performed with robots in Stationary Production Units. ROS as communication middleware provides a powerful framework to create and reuse different robot algorithms. In the same way, the use of SimUEP-Robotics component-based platform resulted in the development of a flexible and useful framework, which can be considered as a rich visual debugger to support the development process and to analyze the results of the simulations. Such component approach gave the possibility to develop different offshore robot applications paving the way towards platform robotizing furnishing the robotization and automation of offshore facilities.

## Acknowledgments

Thanks to Petrobras for supporting this project. The authors would also like to thank the support provided by the research funding organizations CNPQ and FAPERJ.

## References

- ABB. *RobotStudio*. 2001. <http://new.abb.com/products/robotics/robotstudio>.
- Bjerkeng, M., A. A. Transeth, K. Y. Pettersen, E. Kyrkjebø, and S. A. Fjerdingen. "Active Camera Control with obstacle avoidance for remote operations with industrial manipulators: Implementation and experimental results." *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. 2011. 247-254.
- Bradski, G. "The OpenCV Library." *Dr. Dobbs's Journal of Software Tools*. 2000.
- Coppelia Robotics. *Virtual Robot Experimentation Platform (V-REP)*. 2010. <http://www.coppeliarobotics.com/>.
- Cyberbotics. *Webots 7*. 1996. <http://www.cyberbotics.com/>.
- Flacco, F., T. Kroger, A. De Luca, and O. Khatib. "A depth space approach to human-robot collision avoidance." *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. 2012. 338-345.
- From, P.J. *Off-Shore Robotics - Robust and Optimal Solutions for Autonomous Operation*. PhD Thesis: Norwegian University of Science and Technology, 2010.
- Hornung, A., Wurm, K. M., Bennewitz, M. Stachniss, C. & Burgard, W. "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees." *Autonomous Robots*, 2013.
- Johnsen, S. O., Ask, R. & Roisli, R. "Reducing Risk in Oil and Gas Production Operations." *IFIP International Federation for Information Processing* (Springer New York) 253 (2007): 83-95.
- Johnsen, S. O., Askildsen, A. & Hunnes, K. "Challenges in Remote Control and Remote Cooperation of Offshore Oil and Gas Installations in the North Sea." *Proceedings of European Safety and Reliability Conference*. Tri City, Poland, 2005.
- Liu Hsu, Costa, R.R., Lizarralde, F. & Da Cunha, J.P.V.S. "Dynamic positioning of remotely operated underwater vehicles." *Robotics & Automation Magazine, IEEE*, Sep de 2000: 21-31.
- Microsoft Corp. *Microsoft Robot Developer Studio (RDS)*. 2010. <http://www.microsoft.com/robotics/>.
- Moghaddam, A. F. , Lange, M., Mirmotahari, O. & Hovin, M. "Novel Mobile Climbing Robot Agent for Offshore Platforms." *World Academy of Science, Engineering and Technology*, n. 68 (2012): 29-35.
- Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. "ROS: an open-source Robot Operating System." *ICRA Workshop on Open Source Software*. 2009.
- ROCK. *Rock: The Robot Construction Kit*. 2012. <http://rock-robotics.org>.
- ROS-Industrial™ Consortium. *ROS-Industrial*. 2012. <http://www.rosindustrial.org/>.
- Siciliano, B., L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modeling, Planning and Control*. Springer-Verlag London Ltd., 2009.
- Technologies, Unity3D. s.d. <http://unity3d.com/>.
- Unity3D. s.d. <http://unity3d.com/>.

Appendix

Table 2. Example of URDF.

```

<robot name="robot">
  <link name="baseLink">
    <visual>
      <geometry> <sphere radius="0.2"/> </geometry>
      <material name="blue"> <color rgba="0 0 0.8 1"/> </material>
    </visual>
  </link>
  <link name="link1">
    <visual>
      <origin rpy="0 0 0" xyz="0 0 0.45"/>
      <geometry> <cylinder length="0.6" radius="0.15"/> </geometry>
    </visual>
    <inertial>
      <mass value="10"/>
      <inertia ixx="1" ixy="0" ixz="0" iyy="1" iyz="0" izz="1"/>
    </inertial>
  </link>
  <link name="link2">
    <visual>
      <origin rpy="0 -1.5708 0" xyz="0.15 0 0"/>
      <geometry> <cylinder length="0.3" radius="0.10"/> </geometry>
      <material name="blue"/>
    </visual>
  </link>
  <joint name="joint1" type="revolute">
    <parent link="baseLink"/>
    <child link="link1"/>
    <limit effort=" 1000.0" lower="-1.5708" upper="1.5708" velocity="0.5"/>
  </joint>
  <joint name="joint2" type="prismatic">
    <parent link="link1"/>
    <child link="link2"/>
    <origin rpy="0 -1.5708 0" xyz="0 0 0.45"/>
    <limit effort=" 1000.0" lower="0" upper="0.3" velocity="0.5"/>
  </joint>
</robot>

```

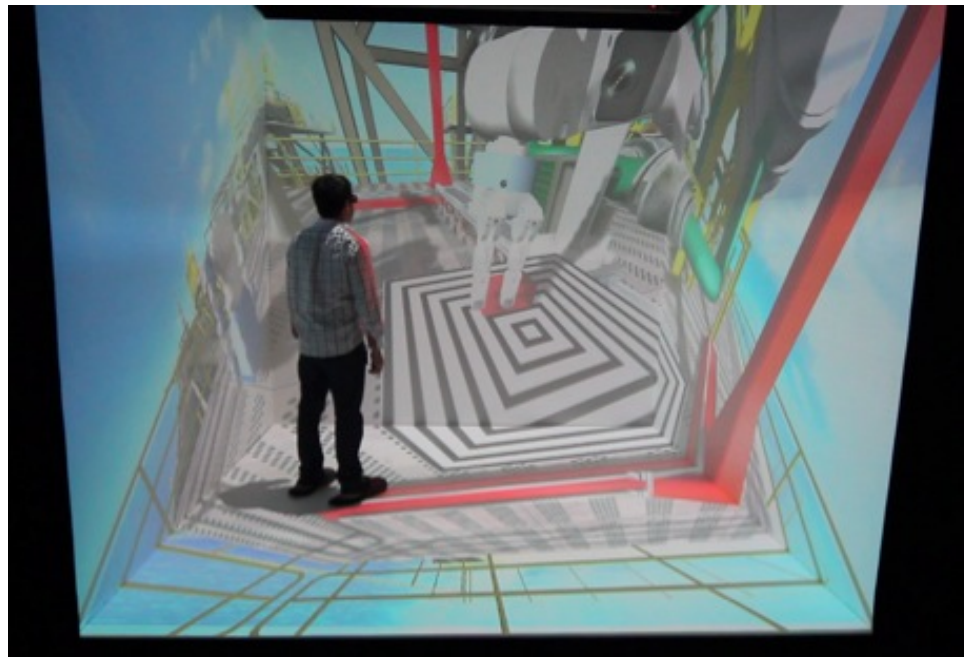
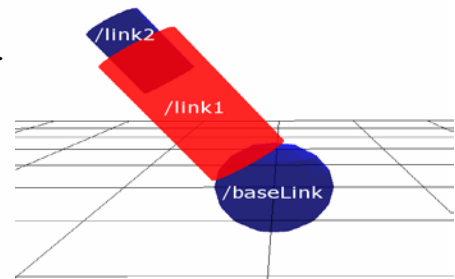
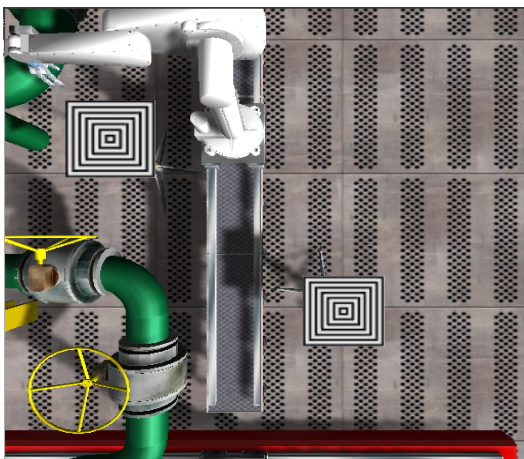


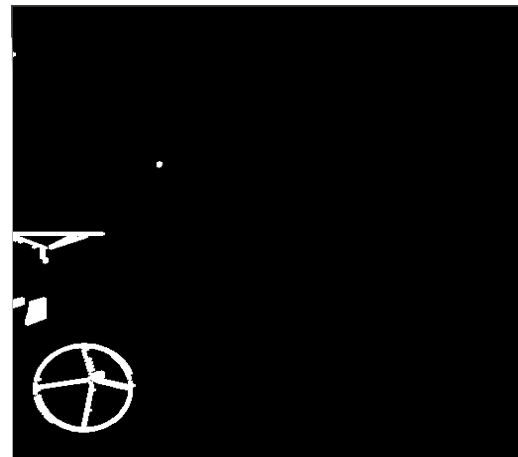
Figure 13. Virtual reality visualized on a CAVE immersive environment.



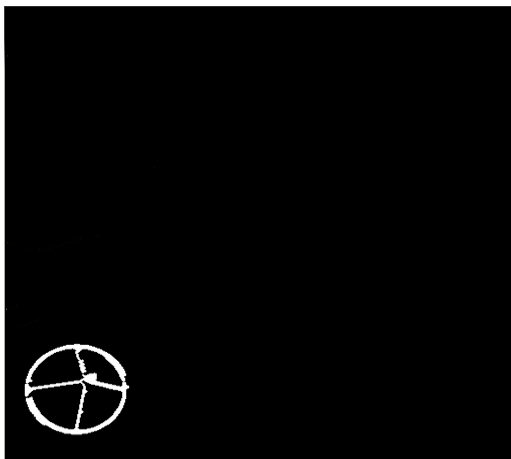
Figure 14. Virtual scene representation of a platform section.



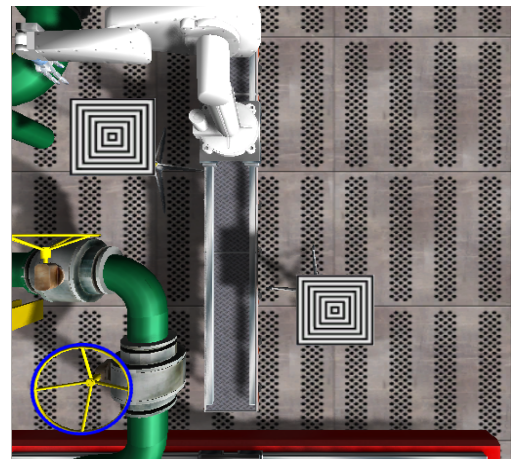
(a)



(b)



(c)



(d)

Figure 15. Valve recognition image processing. Original simulated camera image (a), yellow objects identification (b), circle fitting filter (c) and original figure with valve marked (d).



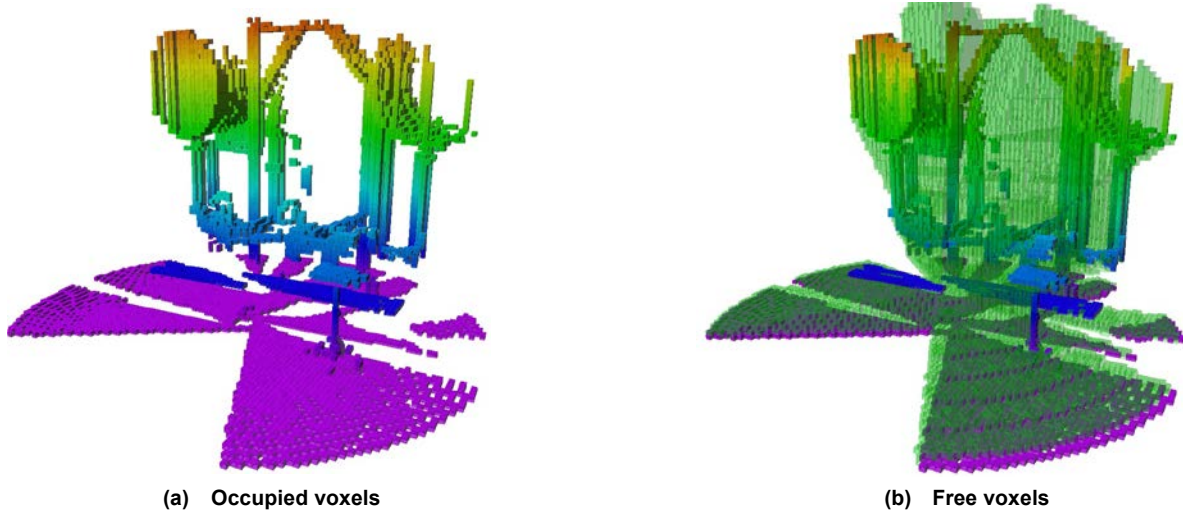
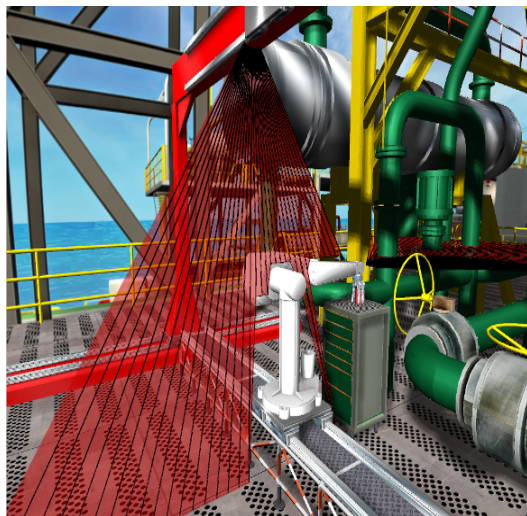
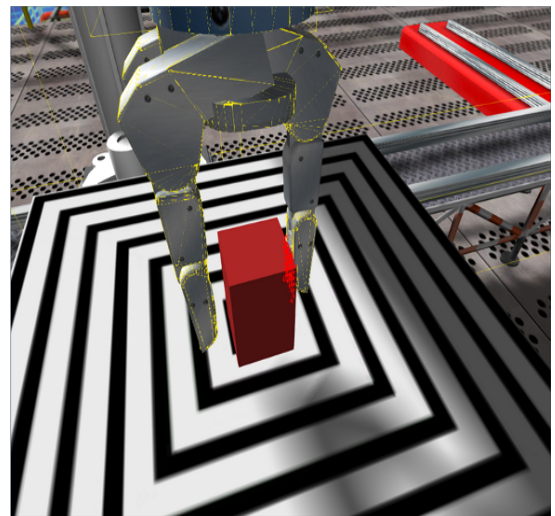


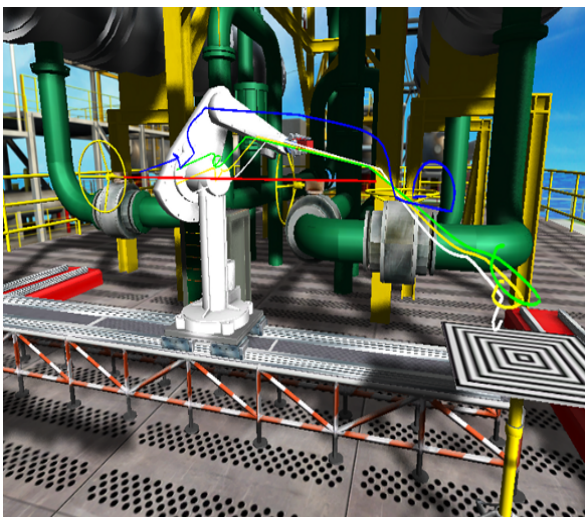
Figure 16. Visualization of occupied and free OctoMap voxels.



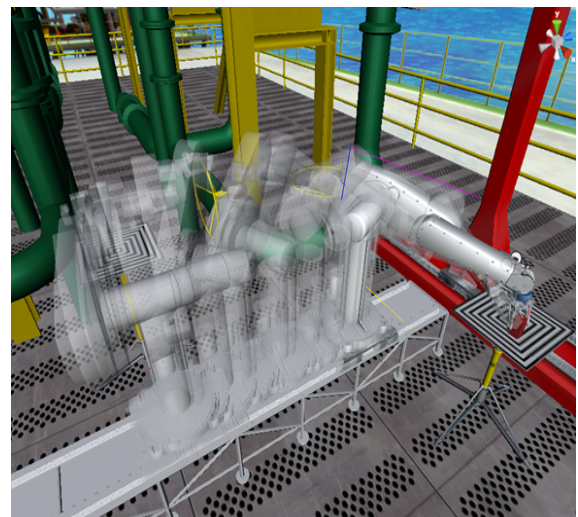
(a) Environment mapping



(b) Objects grasping



(c) Path Planning



(d) Path following

Figure 17. Representation of the four steps that compose the pick-and-place task.

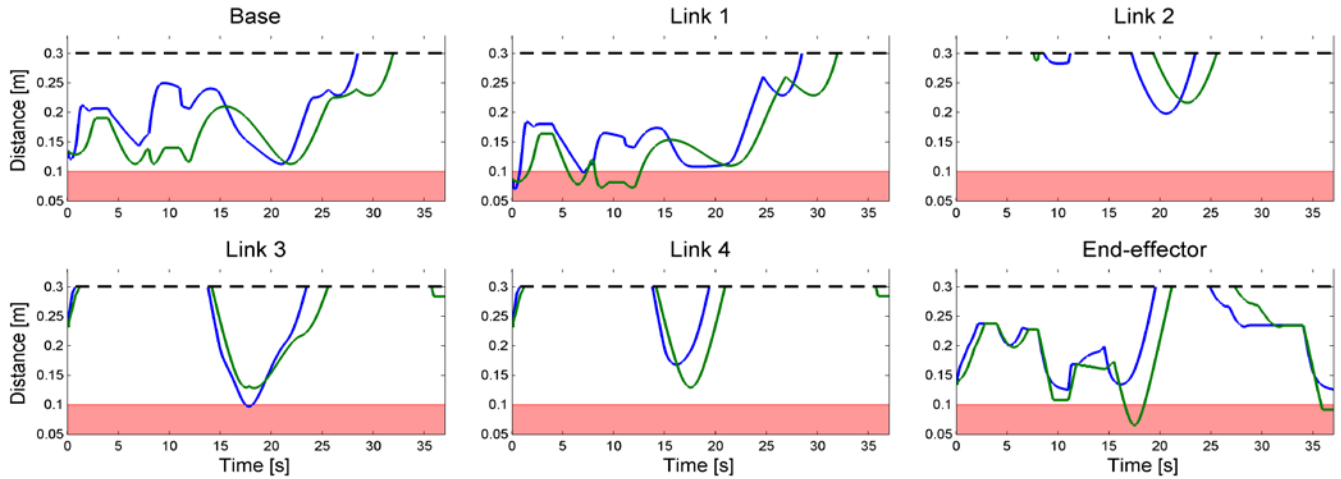


Figure 18. Distance from each link to its corresponding closest obstacle without collision avoidance (green line) and with obstacle avoidance (blue line).

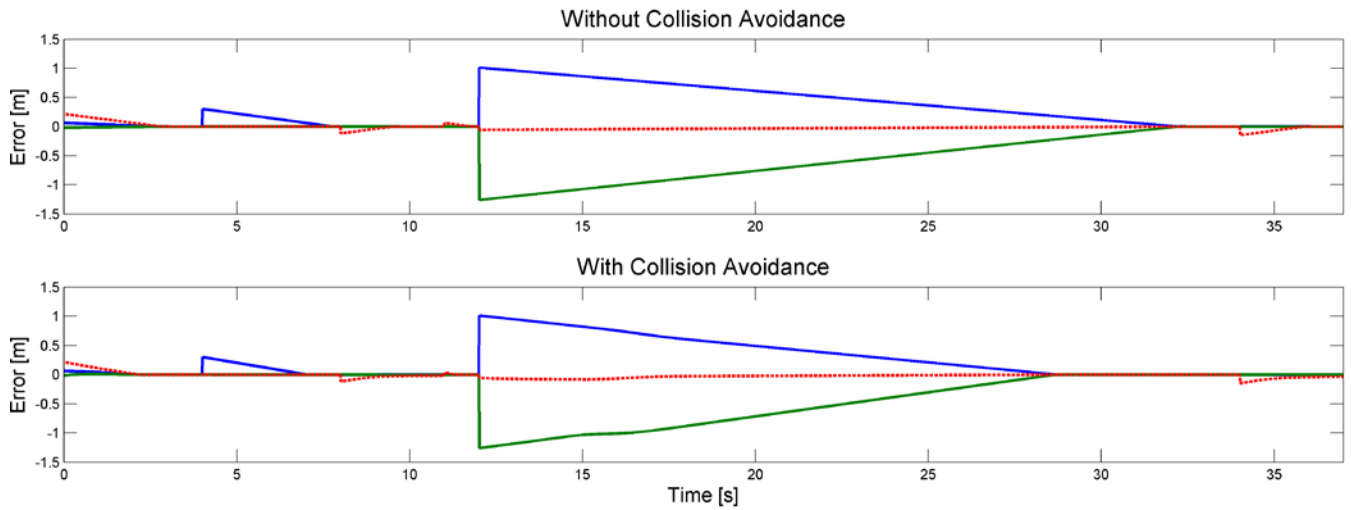


Figure 19. Position errors with and without collision avoidance. The blue line, the green line and the red line refers to the first, second and third dimensions of the respective errors.