Review Article

A ROS-Gazebo Interface for the Katana Robotic Arm Manipulation

Rawan A. AlRashid Agha[1,a*], **Zhwan Hani Mahdi**[1,b], **Muhammed N. Sefer**[2,c], **Ibrahim Hamarash**[1,d]

[1] Department of Computer Science and Engineering, University of Kurdistan-Hewler, Erbil, Iraq
[2] Department of Computer Engineering Techniques, Northern Technical University, Mosul, Iraq
E-mail: [a]rawan.arsn@ukh.edu.krd, [b]zhwan.hanimahdi@ukh.edu.krd, [c]mohammed.sefer@ntu.edu.iq, [d]ibrahim.hamad@ukh.edu.krd

Access this article online		
Received on: April 6, 2020	**Accepted on:** February 4, 2021	**Published on:** June 30, 2021
DOI: 10.25079/ukhjse.v5n1y2021.pp26-37	**E-ISSN:** 2520-7792	

Copyright © 2021 Rawan et al. This is an open access article with Creative Commons Attribution Non-Commercial No Derivatives License 4.0 (CC BY-NC-ND 4.0)

Abstract
Nowadays, simulators are being used more and more during the development of robotic systems due to the efficiency of the development and testing processes of such applications. Undoubtedly, these simulators save time, resources and costs, as well as enable ease of demonstrations of the system. Specifically, tools like the open source Robotic Operating System (ROS) and Gazebo have gained popularity in building models of robotic systems. ROS is extensively used in robotics due to the pros of hardware abstraction and code reuse. The Gazebo platform is used for visualisation because of its high compatibility with ROS. In this paper, ROS and Gazebo have been integrated to build an interface for the visualisation of the Katana Arm manipulator.

Keywords: ROS, Gazebo, Katana Arm, Robotic Systems, Katana Native Interface (KNI).

1. Introduction

In robotics, simulators have played a significant role in evaluating new subjects, techniques, and algorithms quickly and effectively. Simulators for robots are used to build embedded applications for any robot without having the need to depend on the actual device, which in return saves time and expenditure. Sometimes, the applications can even be transmitted onto the actual robot directly due to the functionality of the simulators (Quigley et al., 2009). Using simulators can ease up the procedure of development and validation of robotic systems, hence permitting the verification of the component integration and behaviour evaluation to take place under various circumstances (Echeverria et al., 2011). There are different simulators in the industry, some of them perform better in special types of robots, others are general and used for different robots and in different situations. Robot Operating System (ROS) (Echeverria et al., 2011) is a collection of library packages and tools to help building robot applications and establishing communication with real robots. ROS is an open-source software and its active society have assisted in building various innovative projects (Pietrzik & Chandrasekaran, 2019). Gazebo (Uslu et. al., 2017) is a three-dimensional simulation environment which is part of the Player Project (Festo MENA, n.d.). Gazebo is created to provide an accurate dynamic environment a robot may encounter as well as the ability to generate the sensor inputs for various functionalities as enabling navigation and avoiding obstacles (García & Molina, 2020). PR2, Care-O-bot and TurtleBot are well-known robots that had been simulated via ROS and Gazebo platforms successfully (Cousins, 2010). Gazebo simulator depending on ROS is considered an effective tool to simulate robots (Uslu et. al, 2017). However, the Gazebo/ROS integration is not straightforward and has its difficulties. Every robotic system has its own requirements which needs to be incorporated in the simulation. This paper is aimed to simulate a ROS-Gazebo interface manipulator for the Katana Arm.

The Gazebo simulator was the official tool for DARPA Robotics Challenge, which is considered as one of the milestones in the robotics simulation because immense use of tools like Gazebo were made; from simulating contacts and movements of the entire body to designing complicated manipulations (Ivaldi & Ugurlu, 2018). Moreover, open source models are one of the main approaches to increase the interaction amongst the robotic research groups and to offer a common platform to test numerous algorithms on a single environment. OpenHRP (Pirjanian et al., 2002), the recent Gazebo and ROS projects are two of the latest projects accomplishing this approach. Table 1 shows and compares Gazebo's status among some of the primary available robotic simulators with their performance attributes.

Table 1. Characteristics of Different Robotics Simulators (Noori et al, 2017)

Characteristic	V-REP	Gazebo	MORSE	Webots	USARSim	STDR/Stage	Unity
Main Program Language	C++	C++	Python	C++	C++	C++	C++
Operating System	MAC, Linux	MAC, Linux	BSD, MAC, Linux	MAC, Linux	Linux	Linux	Linux
Simulation Type	3D	3D	3D	3D	3D	2D	3D
Physics Engine	ODE, Bullet, Vortex, Newton	ODE, Bullet, Dart	Bullet	ODE	Unreal	OpenGL	Unity 3D
3D rendering Engine	Internal, External	OGRE	Blender Game	OGRE	Karma	-	OGRE
Portability	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Support	****	*****	****	****	***	****	**
ROS Compatibility	****	*****	****	***	**	****	*

2. Modeling of the Katana Robotic Arm

In this paper the virtual Katana Arm was utilized, see Figure 1. The Katana robotic arm was produced by Neuronics, a Zurich based company, and it is available for industrial and/or research purposes (Lu et al., 2012). The Katana robotic arm is a programmable, fully integrated electronic robot that has the capability to operate in a four to six degrees of freedom (DOF) with high accuracy.

Figure 1. Katana Robotic Arm

In such robotic arms, each joint location defines a variable condition in which the angular movement is altered. This angular motion, which is essentially described as the analysis of how a robot moves in its workspace, is controlled by kinematics (Holden, 2012). This is directly related to the expected motion, position and direction of the end effector, which is a gripper. A manipulator contains multiple links, joined together by joints where each link has a coordinate frame. "Homogeneous transformations describe the position and orientation between coordinate frames (links)" (Holden, 2012).

This robotic arm is quite versatile and its simulation package is available for users to simulate in the Gazebo simulation environment (Früh, 2003). The Katana Native Interface (KNI) is an open source software library for controlling the

Katana robot (Günther, 2017). The KNI is showcasing an execution of robotic kinematics and calculations of the path for a simultaneous control of the axes and the cross paths in space with the gripper.

Denavit-Hartenberg or D-H convention is one of the mostly used convention for the selection of frames of reference in robotics (Hassan, 2013). It is a systematic method for identifying the relationship between adjacent links and offers a mathematical description for all the manipulators based on the robot's geometry (Abbas, 2013). In other words, it simply describes the position and orientation of a current link with respect to the preceding one. In D-H, each homogenous transformation Ti is expressed as the product of four basic transformations (Hassan, 2013).

$$T_i = Rot.(z,\theta_i).Trans(z,d_i).Trans(x,a_i).Rot(x,\alpha_i)$$

Since the virtual Katana 450 Arm has 5 DOF (Günther, 2013), its schematic diagram is demonstrated in Figure 2 with having five rotational joints and a gripper and the robot is at home position where all joint angles are equal to zero.

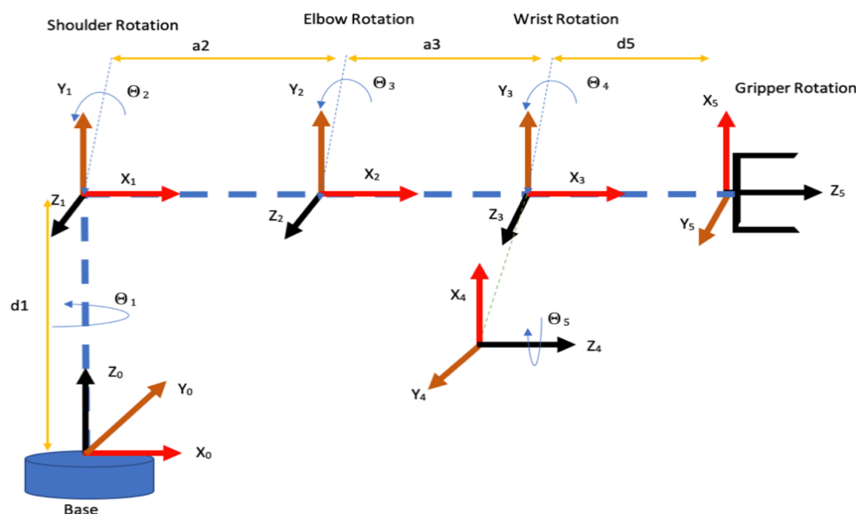

Figure 2. Schematic of a 5 DOF Katana Arm at home position

- Joint 1 is the base and the axis of motion of this joint is Z_0. In the X_0Y_0 plane, this joint makes a rotational Θ_1 angular motion around the Z_0 axis.
- Joint 2 which can be also referred to as the shoulder rotation whose axis is perpendicular to the axis of Joint 1. In the X_0Y_0 plane, this joint enables a rotational Θ_2 angular motion around the Z_1 axis.
- Joint 3 also known as the elbow rotation and Z_2 is its axis which is parallel to Z_1 axis of joint 2. Provides Θ_3 angular motion in X_2Y_2 plane.
- Joint 4 known as the wrist rotation and Z_3 is its axis which is also parallel to the Z_1 axis of Joint 2. Provides Θ_4 angular motion in X_3Y_3 plane.
- Joint 5 is referred to the gripper rotation, whose Z_4 axis is vertical to the Z_3 axis and it gives Θ_5 angular motion in the X_4Y_4 plane (Hassan, 2013).

In order to perform object manipulation and interaction, the packages used for the Katana Arm in this paper are; "Katana_arm_navigation" and "Katana_driver" stack. Basically, the robotic arm can work in the simulation via these two packages where the "katana_driver" stack runs the main driver for the Katana Arm (Schmid, 2013). The "Katana_driver" includes drivers for ROS hardware, Gazebo plugins as well as some other functionalities for the Neuronics Katana family of robot arms (Günther, 2017).

Particularly, it offers:

- Joint Trajectory and Follow Joint Trajectory execution on the physical arm
- Simulation of the Katana Arm in Gazebo
- URDF descriptions
- Simple teleoperation (Günther, 2017).

Then, via the help of the "katana" node the user is able to control and drive every single motor of the Katana Robotic Arm. In addition, kinematics are given by the node "katana arm navigation"; both forward and inverse kinematics for the arm. This node relies on a "actionlib" stack, which is a bundle that provides the required resources for producing long-running servers that can be preempted in order to send requests to the server. It can also have a client interface. Thus, to this node, the user may send goal commands. In other words, if the target is to move to the whereabouts of the gripper, it is then sent via a service call to the "katana-arm-navitation" node, where each motor's movement is

measured to know how each needs to move in order to achieve the goal. The required motor movements are therefore sent to the "katana" node, which causes the Katana Arm to shift to the desired position (Schmid, 2013).

3. Simulation

ROS is the main framework used in this paper with using Gazebo simulator to complement ROS as it provides an accurate and efficient simulation of robots in its environment with high-quality visual graphics (Mittal, 2018). Simulators attempt to mimic the actual world to an extent, that's why the use of such platforms saves time, cost, and experimentations with little damage as well as using hardware parts that are not instantly available for use. The block diagram in Figure 2 demonstrates the architecture of the RPS-Gazebo interaction. One of the important things about ROS is that; it is based on nodes, topics, services and messages (Qian et al., 2014).

3.1. Ros

ROS was developed by the Stanford Artificial Intelligence Laboratory in 2007 (Linner et al., 2011). As mentioned before, ROS is an assemblage of software frameworks that contain the necessary tools for implementing robotic applications. ROS offers standard operating system services like low-level device control, hardware abstraction, employment of generally used functionalities, package management and message transmission amongst processes (Chikurtev et al., 2018). ROS permits the simulation of creating dynamic environments with different robots of multiple degrees of freedom as well as creating different types of sensors (Linner et al., 2011). Figure 3 illustrates the architecture of a set of ROS-based processes where they occur in nodes as they are the basic computational entity. Even though a quick reaction and low delay are very crucial in robot control, ROS is not a real-time operating system (RTOS), however an integration of ROS with a real-time code is possible (Chikurtev et al., 2018).

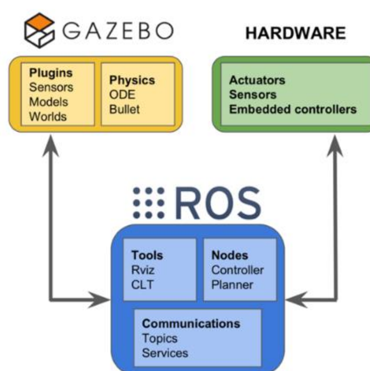

Figure 3. Architecture of ROS-Gazebo interaction (Mittal, 2018)

The philosophy behind ROS is code reuse where developers create a portion of software that can work on other robots by doing minimal code changes (Qian et al., 2014). The core of ROS is the ROS Master as it specifies name registration to all the nodes in the ROS system, see Figure 4. The master plays an important part in allowing each ROS nodes to find one another. It also enables the exchanging of messages and invoking services. All the main functionality of ROS is broken down to some chunks; which are named nodes; and communicate with one another via messages. These messages are basically data structures containing typed fields. A node sends a message by publishing to a designated topic where the topic has a name that is used to identify the content of the message. Therefore, the relevant topic must be subscribed to by any node that has an interest in a particular kind of data. For one topic, there may be several concurrent publishers and subscribers, as well as an individual node for publishing or subscribing to several subjects. In addition, via the services that are composed of a set of two message structures, request and reply can be done. By submitting a request, a client uses the service and waits for the response, in Figure 5 the ROS messaging is illustrated. Hence, ROS client libraries commonly present such a communication as if it was a remote call (Qian et al., 2014).

Moreover, in ROS the first step is building a 3D model of the environment which can be done by any of the modelling tools. The 3D model can be exported in the stereolithography (stl) or collada meshes format so it can be incorporated into ROS (Linner et al., 2011). After that, the created 3D model of each component is combined in ROS and this is done via the package of the robot model that has the parser Unified Robot Description Format (URDF) file. The URDF describes the different meshes/links by the concepts of joints and links as seen in Figure 6 two links that are connected through a joint are called a parent and a child link. Furthermore, how each component moves, is dependent on the type of joint which can be a fixed joint, prismatic, revolute, etc. This URDF file should have a full description of the robots so it can successfully be imported into ROS. The description comprises the name of the robot or model, its mesh (3D) structure, collision and extra information on the visuals of the components like the colour and texture (Linner et al., 2011).

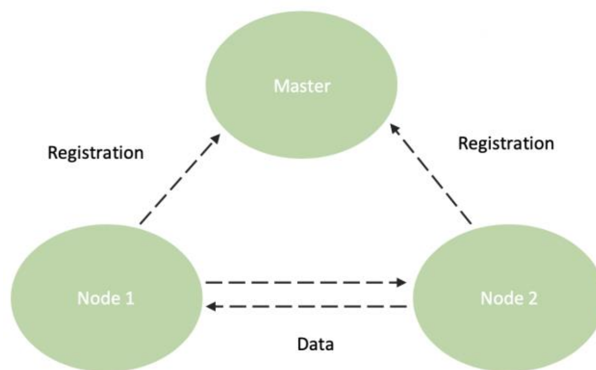

Figure 4. ROS communication (Chikurtev et al., 2018)

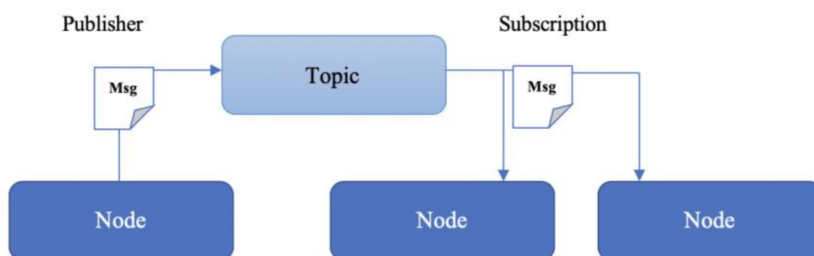

Figure 5. ROS Messaging (Yamashina, Ohkawa, Ootsu, & Yokota 2015)

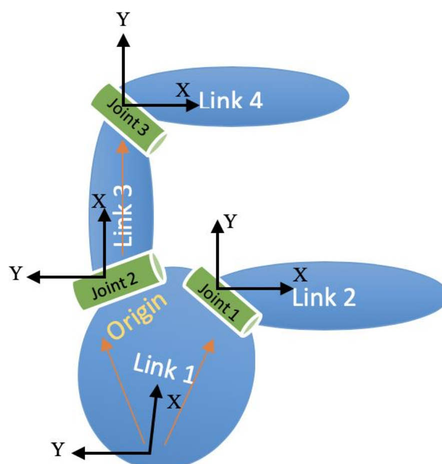

Figure 6. Robotics structure using concept of links and joints (Linner et al., 2011)

Figure 7 is an illustration how code is organised in ROS. Finally, the Gazebo simulator comes with ROS and also it can be downloaded alone. Gazebo is a real-time physics engine and has a rich set of sensors and plugins.

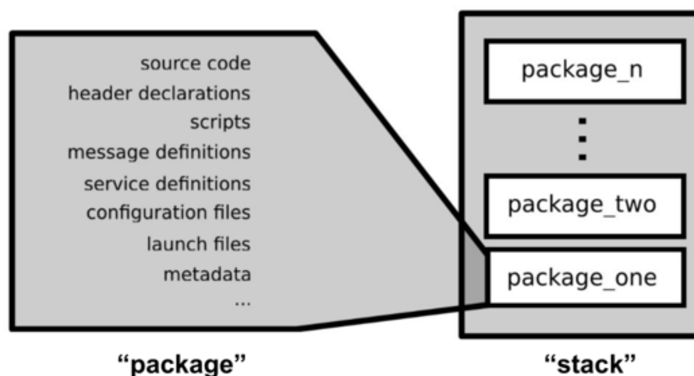

Figure 7. How code is organised in ROS (Mittal, 2018)

3.2. Gazebo Simulator

The Gazebo is a 3D robotic and sensor simulator for indoor and outdoor environments. Accurate sensor feedback and reasonable physical interactions between objects are provided by Gazebo. It is easy to build controlled environments where participants can interact in a real way with manipulators and that is why many researchers have used Gazebo to design experiments to be conducted in such a simulated setup solely because of its realistic feeling. Furthermore, Gazebo simulator can split into physics simulation libraries, user interface, communication, sensor generation and rendering (Qian et al., 2014). Gazebo's architecture is graphically described in Figure 8. The world exemplifies all the models and environmental aspects like gravity and lighting. Each model includes at least one body and a number of joints and sensors. While the third-party libraries interact with Gazebo in the lowest level, hence avoiding any model from being dependent on any certain tool that could change later. Through having a shared memory interface, the receiving and returning of client commands and data are accomplished. Moreover, a model can have multiple interfaces for different functions like controlling of the joints and the transmission of images (Koenig & Howard, 2004).

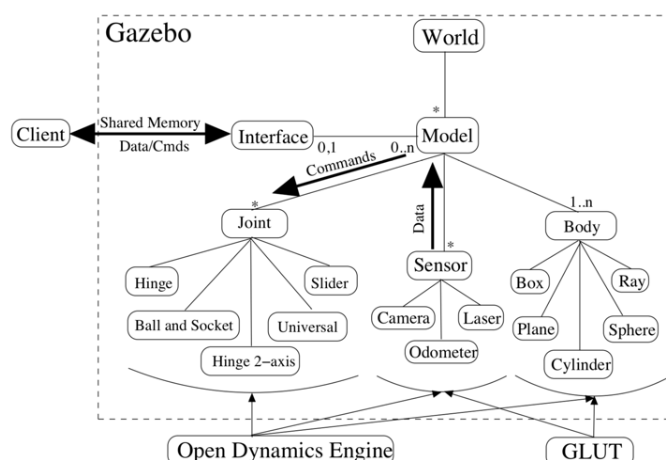

Figure 8. General Structure of Gazebo components (Koenig & Howard, 2004)

In addition, when the client sends control data, the coordinates of the objects to the server, then the server performs real-time control of the virtual robot. By putting the client and server on different machines; the realisation of a distributed simulation can be established. Installing ROS plugin for Gazebo assists in implementing an immediate communication portal to ROS, allowing the virtual and the actual robots to be controlled by the same program. Thus, this offers an efficient simulation instrument for developing and testing actual robotic systems (Takaya et al., 2016).

3.3. Robot and Environment Modelling

In ROS, to represent the robot and the models the URDF is required. To describe all the needed elements like joints, sensors, and links to be uniformed in ROS, the URDF file is used which is basically an XML or Xacro file format. The URDF can only indicate the kinematic and dynamic specifications of one robot alone. So, to make the URDF work in the Gazebo simulator correctly, extra specific simulation tags about the robots' pose, inertia, collision, friction and other properties are needed to be inserted. It contains tags such *<robot>*; this tag has the name of the robot that will be shown in all the ROS subsystems. Other tags like *<link>* has the name of the link and illustrates a visual of an object connected to another via a joint which also would need a *<joint>* tag. As mentioned before two links are split into parent and child link and since each has a fixed coordinate frame, the transformation details from parent link to child must be given in a joint block (Qian et al., 2014). Inside the link tag a *<visual>* tag is used to specify the information on the geometry of the object being used (Linner et al., 2011). Additionally, the *<inertial>* tag is necessary to be indicated for gravity. As well as, the *<collision>* tag that identifies the parameters of collision, as ROS generates an invisible layer on the visual element which is a shell that has the capability of colliding to other objects (Linner et al., 2011). Having these added properties allow the URDF file to be compatible with the built-in simulation description format (SDF) of Gazebo's model format file. Thus, the SDF enables a full description of the simulation of both the world and the robot.

Furthermore, the conversion process of URDF to SDF is simply done by the aforementioned Gazebo plugins into the URDF file. See Figure 9 for the result of the conversion of URDF file to SDF format. As the plugins can be enclosed into ROS messages and the service calls the sensor outputs and/or motor inputs, where the Gazebo plugins generate a topic between ROS and Gazebo. As explained before, the control process interaction in ROS is done by executing a publish/subscription to that specific topic. Lastly, numerous plugins are offered in Gazebo such as ROS interface for simulating cameras as well as a plugin for synchronization of multiple camera shutters to issue the images jointly—

commonly stereo cameras. Other plugins are like inertial measurement unit (IMU), planar move, differential drive, amongst many more (Takaya et al., 2016).

```
<gazebo>
  <plugin name="differential_drive_controller" \
                    filename="libdiffdrive_plugin.so">
    ... plugin parameters ...
  </plugin>
</gazebo>
```

(a) The URDF file

```
<model name="P3DX_robot_model">
  <plugin name="differential_drive_controller" \
                   filename="libdiffdrive_plugin.so">
    ... plugin parameters ...
  </plugin>
</model>
```

(b) The SDF file

Figure 9. Using Gazebo plugins (Takaya et al, 2016)

4. Results and Analysis

4.1. Installing ROS and Gazebo

Firstly, ROS needs to be installed for this simulation the ROS kinetic kame was installed on Ubuntu v.16.04 as ROS kinetic (ROS Wiki, 2019) only supports some versions of Ubuntu. The rosdep will make sure all the package dependencies are installed for any source that is needed for compilation and is necessary for running some core ROS components (Jong, 2018). Usually the rosdep is used to install rospackages with all of the dependencies. To make our own packages a catkin workspace needs to be set up.

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
source devel/setup.bash
```

Catkin and catkin_make both must be installed automatically with the ROS installation. Catkin is the build system of ROS and it combines CMake with python, so a build system is created that should make cross compiling easier (Jong, 2018). The first time, catkin_make command is run in the workspace, a CMakeLists.txt will be created in the 'src' folder. As well as, a 'build' and a 'devel' folders (Qian et al., 2014). ROS installation is complete and it consists of many packages and modules that interact with one another through messages as explained before.

Therefore; a Gazebo plugin must be added to the URDF file so it can parse the transmission tags and load the appropriate hardware interfaces and controller manager (Qian et al., 2014). A piece of code as shown below; would enable the gazebo ROS control plugin to permit the start of a list of controller manager services, that can be used to stop, start, list or switch controllers.

```
<gazebo>
<plugin name="gazebo_ros_control" filename=" libgazebo_ros_control.so">
<robotNamespace>/MYROBOT</robotNamespace>
<robotSimType>gazebo_ros_control/DefaultRobotHWSim</ robotSimType>
</plugin>
</gazebo>
```

4.2. Katana Simulation Model

After the installation of ROS and Gazebo simulator, the Katana Arm is cloned. This specific Katana simulation model has 5 DOF as mentioned previously. The Katana native interface package is C++ library for controlling application development, which provides an interface to control a robot using C++ and C# (Kanajar, 2011). Figure 10 shows the Katana Arm in the Gazebo simulator.

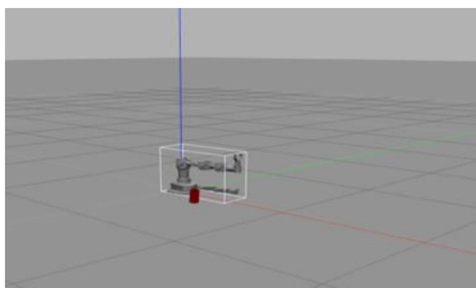

Figure 10.The Katana Arm in the Gazebo simulator

4.3. Adding Laser Module

The simulation of the Katana Arm with Gazebo comes with many models, Gazebo gives the ability to visualise models. For making the arm move, a force needs to be applied onto the joints. The Katana Arm model is defined in an URFD file or a XACRO file. Furthermore, a piece of code is added onto the URDF file to add a laser module to find the specified range between the base of the robot and the object and it can be seen with blue lines as shown in Figure 11.

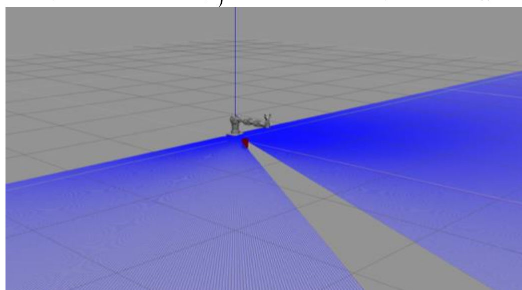

Figure 11. The Katana Arm with the laser module

For visualising the module and setting up the interface of ROS some dependencies are needed for Python and C++. The communication is done through the modules by using messages. The laser messages contain a header that's a timestamp, which defines the time of taken samples. As well as, a range message to define the minimum and maximum ranges so that the message range will be read and will ignore any data that is not within the range. Another message in the laser is the start of scanning the angle and the last one in radians, as well as the message for the time between the two scans. While for the joint, the messages contain the name of the joint and its position, the velocity of the joint and the force applied on the joint. The system needs two call back routines one for joint one for laser for doing so the function call-back needs to be subscribed with ROS and this is done by using a node handle. This subscription contains a name of the call-back function, the receiver where the ROS messages to be sent and the size of the queue.

4.4. Calculating Positions

There are some methods to interpret sensor data to positions and the positions to the joint angles. For this simulation, a simple approach is used for easy understanding and implementation.

A. Sensor to Cartesian

The objects that are around the Katana Arm can be found through a set of distance measurements as each of these objects is denoted by several measurements. So that an object around the arm is found whenever any of those measurements return a value that's between the minimum and maximum value. Assuming that when there are multiple successive measurements, they are of the same object. Which would mean that at least a single sample is measured outside those boundaries between them, hence assuring that the objects would not overlap. An illustration is depicted below in Figure 12:

Figure 12. Illustration of the arm base with the object detection (Jong, 2018)

The middle point of the objects is set as an aim point as these objects are round. Therefore, calculation of the distance to the middle point of the object can be done by taking the average between the rightmost perceived distance and the leftmost. Furthermore, to obtain the x coordinate, the multiplication of the sine of the sample angle to the distance needs to be taken. Also, to obtain the y coordinate the multiplication is done with the cosine angle and finally for obtaining the z coordinate, any number that's between zero and the height of the object should be satisfactory. Figures 13 and 14 depict the reference and body frame for the Katana Arm in the cartesian coordinate. The three lines of code

can be seen below for the aforementioned calculations. Moreover, this process can be recurring till the objects are detected.

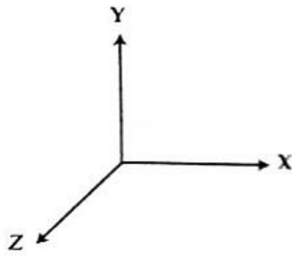

Figure 13. Reference Frame

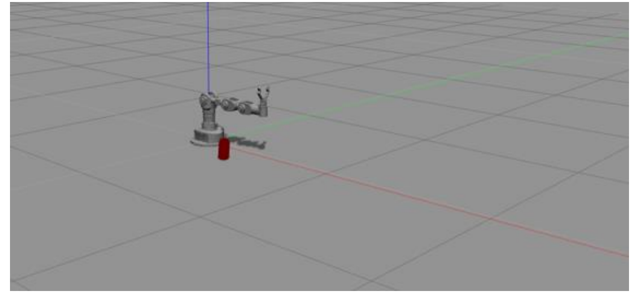

Figure 14. Body Frame

```
pos.x = sin(angle) * distance;
pos.y = cos(angle) * distance;
pos.z = 2;
```

B. Cartesian to Joint Positions

In order to get the dimensions, an easy way is to divide the arm up into triangles as seen in the Figure 15.

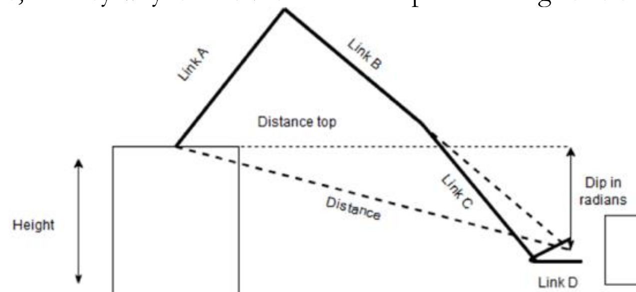

Figure 13. An illustration of the triangle system of the Arm (Jong, 2018)

The dimensions are known by solving for the dimensions of the arm that are needed to be defined for each link. Moreover, the link BC is the imaginary dashed line that is directed to the link B to the point of the gripper. In order to calculate the distance to the object, the top view distance is computed to the object, then the actual distance from the top of the base of the Katana Arm to the grabbing point of the object is computed.

```
//distance joint to the target:
float distance_top = sqrt(goal[0]*goal[0] + goal[1]*goal[1]) + 30;

float distance = sqrt(distance_top*distance_top + (height - goal[2])*(height - goal[2]));
```

In addition, the arm will set the height of the base to zero and that's due to the triangle organisation used previously. Hence, this will require a modification which is done by taking the arccosine of the distance top divided by the distance. This is needed if the point of the end effector requires to go higher. As it is what the distance should dip to from the base of the arm. So, the conventional way would say the higher the points then the dip would be negative.

Next, the calculation for the arm's rotation in alignment with the object is needed. This step needs to be done in four quadrants where the computation of the angle is dependent on the quadrant it is in.

```
//calculate the angle in which the first joint should turn if
(goal[1] > 0)

pos[0] = atan(goal[0]/goal[1]); else

if (goal[0] < 0)
pos[0] = atan(goal[0]/goal[1]) - PI;

else
pos[0] = atan(goal[0]/goal[1]) + PI;
```

The first lifting joint which is responsible on the dip and the alpha in the triangle. While the second lifting joint should make an angle that is pi minus the angle in the triangle that's being made. For the third position, because the end effector has an angle of 90 degrees to the rest of the arm, this will be corrected with taking the inverse tangent of the division of link D to link C. The final rotation joint is not used; therefore, the expected position needs to be ensured and set to zero. Once all the joint calculations are done, the trajectory message can be sent.

4.5. Path Planning

After that, the path needs to be planned because if it moves in straight line to the subsequent object it can knock it over. That's why, steps need to be put in between. However, to plot a smooth trajectory would be a bit hard with the current purposes. What's needed is to pick points that would stay away from any obstruction. Therefore, once an object is detected, it would approach it from a distance over a straight line that's between the arm base and the detected object. Moreover, the object will be lifted higher than the top of the stack so when it is placed the object on the stack, it will go back again in a straight line to the arm base and then go over to the next object.

4.6. The Grasping of the Gripper

The components that were necessary to develop the Katana Arm to localize and pick up an object with the end effector. However, the physics is not very optimal for grabbing an object as it would try to do so and it would either push over the object or pick it up for a moment and then drop. Nevertheless, there are ways to make this work which will be discussed in Section five. Figures 16, 17 and 18 are demonstrating the gripper grasping a can and lifting it up onto the other can but it can be seen that the can falls over once the gripper lets go of the can.

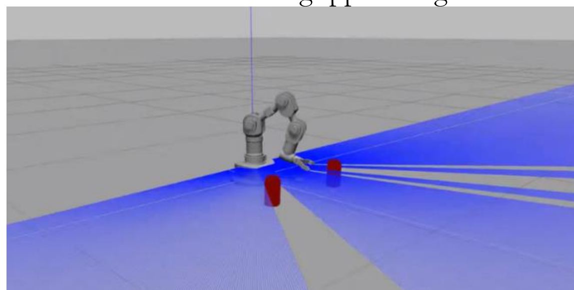

Figure 16. A demonstration of the gripper approaching the can with the laser module on

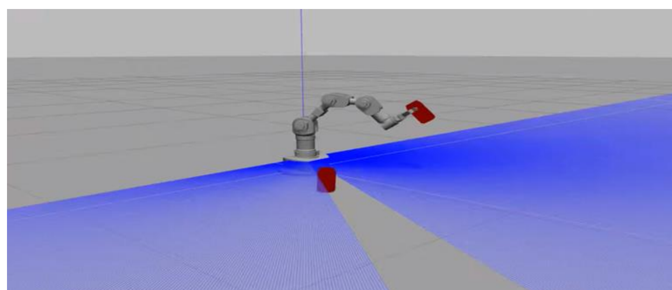

Figure 17. The gripper holding the can

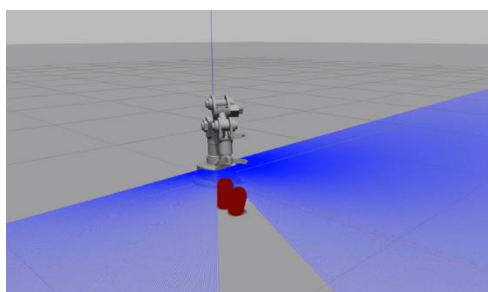

Figure 18. The gripper letting go of the can and dropping it

The development of robotic systems is a complicated procedure, the designing alone has many complexities and more tools are required to make such a process less tedious. In the actual world, a real-time response to the incoming sensor data is occurring. If actuators are being used then a thorough understanding about the domain is very crucial to understand how they are operating and how they are behaving. Therefore, in the physical world working on experiments for such robots is quite challenging. Because the equipment in real is very expensive and there is the risk of the materials and/or the environment of its operation to get damaged. Thus, designing a prototype in a simulator is a reasonable first

phase. Simulating in powerful simulators will most certainly offer a fine behavioural view of the robot. Nevertheless, good simulations could be close to the details of the real world. However, if the simulation deploys an ideal system, oscillations could be shown in the results due to the critical values of the components. The simulation of the Katana Arm, can show for example a failed friction stack can be observed as well as a weird wobble. Which demonstrates that a simulator is just a tool and needs to be dealt with a considerable amount of thought. That's why, having a thorough physics view of the actual world is very vital for software development of robotic systems. Furthermore, in the simulation of the Katana Arm, the grasping of the object is not occurring as required due to the physics engine not being optimized very well as mentioned before. However, extending the models with a suitable friction model that could aid in not letting the object slip from the grip and this can be tricky.

5. Conclusion

To conclude, robotic systems can be efficiently and easily designed through powerful simulators. Simulators aid in developing robotic systems without having to build the robot physically or damage the actual equipment. There are many possible methods to build robot models in virtual environments. Some of the aforementioned simulators are limited to 2D while others have a 3D environment. From these simulators ROS and Gazebo simulator have gained recognition in the robotics research world. ROS is heavily used owing to its hardware abstraction and code reuse and the Gazebo simulator because of its compatibility with ROS. The aim of this paper was to review the simulation of the Katana Arm in the ROS-Gazebo platform. The Katana Arm in this work was simulated in so it can move its arm and the gripper to grasp a can and a laser module was added for finding the ranges as well as visualising the model. As mentioned before, the URDF file was used to define the required components as the joints, sensors, and links in ROS and to make this file to work in the Gazebo simulator properly, additional simulation specifications were needed like inertia, pose, and collision.

The first stage of developing any robotic system should be designing a prototype in a simulator to ensure the behavioural aspect of the robot. However, simulations will not reveal all the details of the real world and that's the reason why the simulation of the Katana Arm in ROS/Gazebo conducted in this work can be enhanced to behave properly. As sometimes the grasping part does not function effectively due to the controller part of the system which has not been studied. As future work, controllers can be added to check the unpredictability in the end effector.

References

Abbas, T. (2013). Forward Kinematics Modeling of 5 DOF Stationary Articulated Robots, *Engineering and Technology Journal*, 31(3), 500-512.

Chikurtev, D., Rangelov, I., Chivarov, N., Markov, E. & Yovchev, K., (2018). Control of Robotic Arm Manipulator Using ROS. Bulgarian Academy of Sciences - *Problems of Engineering Cybernetics and Robotics*, 69, 52-61.

Cousins, S. (2010). ROS on the PR2 [ROS Topics]. *IEEE Robotics & Automation Magazine*, 17(3), 23-25. doi: 10.1109/mra.2010.938502

Echeverria, G., Lassabe, N., Degroote, A., & Lemaignan, S. (2011). *Modular open robots simulation engine: MORSE*. 2011 IEEE International Conference on Robotics and Automation. doi: 10.1109/icra.2011.5980252

Festo MENA. Retrieved 10 January 2019, from https://www.festo.com/cms/en-jo_jo/index.htm

Fruh, H. (2003). Handling Robots Equipped with Highly Redundant Sensors [Ebook] 340–341. Zurich: Schweizerische Chemische Gesellschaft.

García, J., & Molina, J. (2020). Simulation in real conditions of navigation and obstacle avoidance with PX4/Gazebo platform. *Personal and Ubiquitous Computing.* doi: 10.1007/s00779-019-01356-4

Günther, M., (2013). *Arm Navigation for the Neuronics Katana family of robot arms*. Lecture notes. Universität Osnabrück. Delivered 11 May 2013.

Günther, M. (2017). uos/katana_driver. Retrieved 7 January 2019, from https://github.com/uos/katana_driver

Haas, J. (2014). *A History of the Unity Game Engine*. BSc Thesis. Worcester Polytechnic Institute.

Hassan, A., (2013). *Path planning of robot manipulator using Bezier technique*. Master's Thesis. University of Technology.

Holden, T., (2012). *Development of an Intelligent Robotic Manipulator*. MSc. University of Central Lancashire.

Ivaldi, S., & Ugurlu, B. (2018). Chapter 35: *Free Simulation Software and Library*. Retrieved January 12, 2019 from https://hal.archives-ouvertes.fr/hal-01614032

Kanajar, P. (2011). *Neptune: Mobile Manipulator with Advanced Human Robot Interaction.* Master's Thesis. The University of Texas At Arlington.

kinetic/Installation/Ubuntu - ROS Wiki. (2019). Retrieved March 20, 2019, from http://wiki.ros.org/kinetic/Installation/Ubuntu

Koenig, N. & Howard, A., (2004). *Design and use paradigms for gazebo, an open-source multi-robot simulator.* IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), Sendai, Japan, 28 September-2 October.

Linner, T., Shrikathiresan, A., Vetrenko, M., Ellmann, B. & Bock, T., (2011). *Modeling and Operating Robotic Environments* Using Gazebo/ROS. 28th International Symposium on Automation and Robotics in Construction (ISARC 2011), Seoul, Korea, 29 June-2 July.

Lu, Z., Chauhan, A., Silva, F. & Lopes, L., 2012. *A brief survey of commercial robotic arms for research on manipulation.* 2012 IEEE Symposium on Robotics and Applications *(ISRA)*, Kuala Lumpur, Malaysia, 3-5 June.

Mittal, M., (2018). *Introduction to Robot Simulation* (Gazebo). Lecture notes, Autonomous Navigation AE640A, Eidgenössische Technische Hochschule Zürich-ETH Zurich University, delivered 10 January 2018.

Noori, F., Portugal, D., Rocha, R., & Couceiro, M. (2017). *On 3D simulators for multi-robot systems in ROS: MORSE or Gazebo?.* 2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR). doi: 10.1109/ssrr.2017.8088134

Pietrzik, S., & Chandrasekaran, B. (2019). Setting up and Using ROS-Kinetic and Gazebo for Educational Robotic Projects and Learning. *Journal of Physics: Conference Series*, 1207. doi: 10.1088/1742-6596/1207/1/012019

Pirjanian, P., Leger, C., Mumm, E., Kennedy, B., Garrett, M., & Aghazarian, H. et al. (2002). *Distributed control for a modular, reconfigurable cliff robot.* Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292). doi: 10.1109/robot.2002.1014381

Qian, W., Xia, Z., Xiong, J., Gan, Y., Guo, Y., Weng, S., Deng, H., Hu, Y. and Zhang, J., (2014). *Manipulation task simulation using ROS and Gazebo*, 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014).

Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R.C., & Ng, A.Y. (2009). *ROS: an open-source Robot Operating System.* ICRA 2009.

Schmid, S., (2013). *Creation of a robot-user interface for persons with reduced motor capabilities*. BSc.

Takaya, K., Asai, T., Kroumov, V. and Smarandache, F. (2016). *Simulation Environment for Mobile Robots Testing Using ROS and Gazebo.* In: 20th International Conference on System Theory, Control and Computing (ICSTCC). Sinaia, Romania: IEEE Control Systems Society.

Uslu, E., Çakmak, F., Altuntaş, N., Marangoz, S., Amasyalı, M., & Yavuz, S. (2017). *An architecture for multi-robot localization and mapping in the Gazebo/Robot Operating System simulation environment*. *SIMULATION*, 93(9), 771-780. doi: 10.1177/0037549717710098

Yamashina, K., Ohkawa, T., Ootsu, K., & Yokota, T. (2015). *Proposal of ROS-compliant FPGA Component for Low-Power Robotic Systems*. ArXiv, abs/1508.07123.

Zou, D. (2014). *ROS+Gazebo Quadrotor Simulator*. Presentation, Lab of Navigation and Location-based Service.