Int. J. Nonlinear Anal. Appl.
Volume 12, Special Issue, Winter and Spring 2021, 1753-1760
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2021.5882

Design and implementation of the 6-DoF robotic manipulator using robot operating system

B. Vinod[a,], B. Bindu[a], G. N. Koushik Karan[b], V. E. Jayanth Akash[b], S. Dinesh Kumar[b]

[a]*Faculty at Department of Robotics and Automation, PSG College of Technology, India*
[b]*Department of Robotics and Automation Engineering, PSG College of Engineering, India*

(Communicated by Madjid Eshaghi Gordji)

Abstract

Material handling robots are replacing human workers in most of the manufacturing shop floors. Robot operating system is an open-source framework that enables visualization and implements various complex robots and their functions. A 6-DoF robotic manipulator with a gripper is designed to perform the pick and place operations. The aim is to integrate the designed robot with the robot operating system. The integrated system is then visualized and controlled using a gazebo and RViz to perform pick and place operations.

Keywords: Robotic manipulator, Pick and place, Robot operating systems, Motion planning

1. Introduction

A. Robotic manipulator

A robotic manipulator consists of one fixed end and one free end to perform the specific assigned task. Robots that are specific to industry perform various tasks such as picking and placing which are similar to the operations of a human arm. Pick and place robots used in modern manufacturing scenarios handle repetitive tasks while freeing up human workers to focus on more complex work [12]. Typically mounted on a fixed stand, pick and place robots are positioned to access different areas to perform the task.

Email addresses: bvinod172@gmail.com (B. Vinod), bindu_psg@gmail.com (B. Bindu), koushikkaran6@gmail.com (G. N. Koushik Karan), akashjayanth14@gmail.com (V. E. Jayanth Akash), dinesh.s@gmail.com (S. Dinesh Kumar)

Received: August 2021 *Accepted:* November 2021

B. Robot operating system

The Robot Operating System which is commonly called ROS is an open-source framework that provides easy integration of components into the robot. ROS is a meta operating system that usually works upon some other operating system that generally is Ubuntu Linux. ROS provides services like an operating system that include low-level device control, hardware abstraction, implementation of commonly-used functionality, package management, and message-passing between processes. ROS is written in C++, Python, and Lisp. ROS is designed as a loosely coupled system in which a process is called a node. ROS provides a variety of tools to visualize and record data, create scripts automating complex configuration and setup processes, and easily navigate the ROS package structures [10]. The addition of these tools greatly increases the capabilities of systems using ROS by simplifying and providing solutions to several common robotics development problems. Important concepts of ROS are nodes, topics, master, service, messages, parameter server.

Gazebo is an open-source 3D robotics simulator. With a gazebo, it is possible to create a 3D scenario with robots, obstacles, and many other objects. By using a gazebo, it is possible to test the robot in difficult or dangerous scenarios without any harm to the robot. The gazebo provides realistic rendering of environments such as high-quality lighting, shadows, and textures. Gazebo is the actual real-world simulator that will allow the developer to set up an environment and simulate the robot. In gazebo, it is possible to create a world with the help of building a model editor inbuilt in the gazebo and use it in the project.

RViz is a 3D visualization software tool in Robot operating system for robot applications. With the help of the RViz tool developer can visualize what the robot is doing and seeing. This is the view of how the robot sees the environment. With the help of RViz developers can debug or reconstruct or reframe the method for the desired result. RViz allows the user to view the simulated robot model, sensor information from the sensors of the robot [6]. If an actual robot is communicating with a workstation that is running RViz, it will display the robot's current configuration on the virtual model. ROS topics will be displayed as live representations based on the sensor data published by any cameras, IR sensors, and laser scanners that are part of the robot's system. This can be useful to develop and debug.

MoveIt! is the state-of-the-art software for the manipulation of robots. The basic function of the MoveIt! system is to provide the trajectories for the robotic manipulator and to put the end effector in a particular place. It incorporates the latest technologies in manipulation, motion planning, kinematics, control, 3D perception, and navigation. It provides a platform for the development of advanced robotics applications, evaluating new robot designs, and building integrated robotics products. MoveIt! is open-source software for manipulation and has been used on over 65 different robots which are widely used. MoveIt! allows its users to import their robots and configure the kinematics of the robots [3].

2. Methodology

The design and implementation of 6-DoF robotic manipulator using ROS have various steps as represented in figure 1.

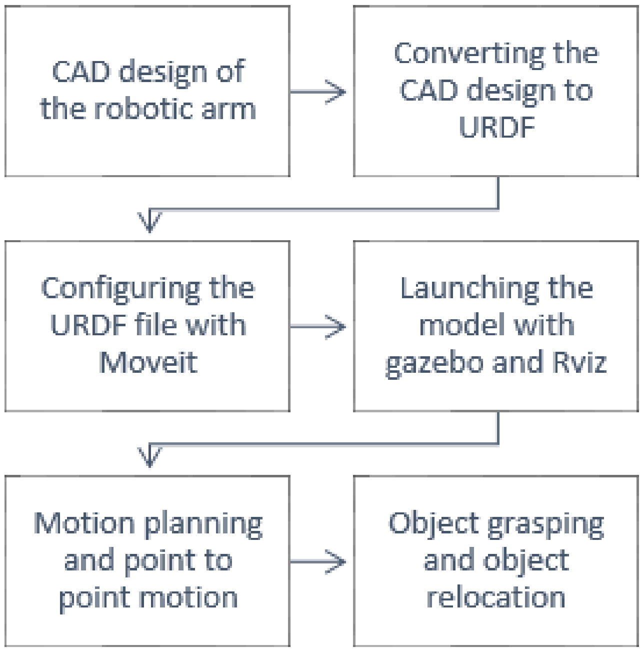

CAD design of the robotic arm
Converting the CAD design to URDF
Configuring the URDF file with Moveit
Launching the model with gazebo and Rviz
Motion planning and point to point motion
Object grasping and object relocation

Figure 1: Process flowchart

The first step involves the CAD design of the robotic manipulator with a gripper in SolidWorks software. The CAD model is converted then into a URDF model. The URDF file is then configured using MoveIt! to implement motion planning and other complex functions. Once the package for MoveIt! is created the robot model is further launched using gazebo and RViz. Motion planning is then implemented and then point-to-point motion using the pose data is executed using move_group. In gazebo, objects are created and gazebo plugins are added to pick the objects. Further, a python file is executed which picks the objects from the desired pose to the destination.

A. CAD design of the robotic arm

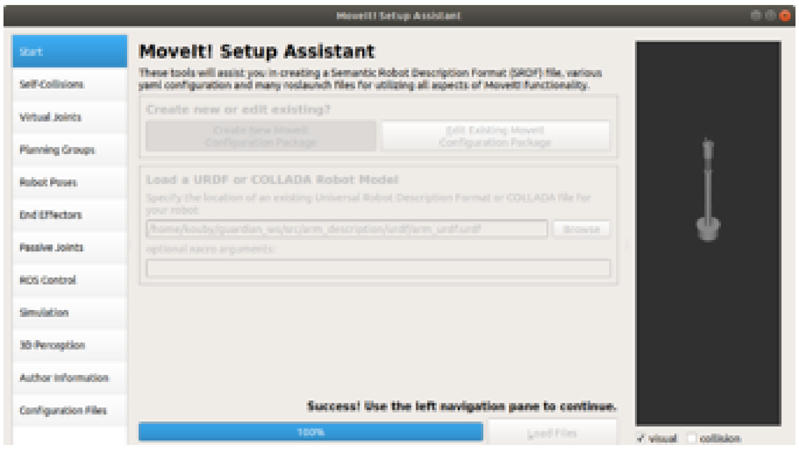
The used model is designed using SolidWorks. Each link of the robot has been designed individually and assembled. The collision model is also generated in SolidWorks. The collision model is a lightweight model of the robot without textures. This is to reduce the computational requirement for simulating and controlling the robot [1].

B. Converting the CAD design to URDF

The model which has been designed in SolidWorks needs to be converted to the format of URDF. This is done using an add-in called SW2URDF. After installing the add-in, export as URDF will be available in the File menu of SolidWorks. There are three stages involved in the conversion [5]. The first stage is to link the parts of the model and to select the reference geometry and the joint type. A tree structure of the model will be generated. The second stage of the export process is to define the joint properties like the coordinate reference for the joints can be configured and the joint limits can be defined in this stage. The third stage is the link properties stage where the inertial matrix is to be checked for availability. If not, the process needs to be repeated. The colour of the links can also be changed at this stage. The conversion process is completed and the model will be exported as a URDF file.

C. Configuring the URDF file with MoveIt!

A workspace is created and a folder named src is made. The converted URDF file is put into the src folder and the package is built. The MoveIt! setup assistant as shown in figure 2 is launched and the URDF file is loaded into it. The preview of the model will appear on the screen.



MoveIt! Setup Assistant

These tools will assist you in creating a Semantic Robot Description Format (SRDF) file, various yaml configuration and many roslaunch files for utilizing all aspects of MoveIt! functionality.

Create new or edit existing?

Load a URDF or COLLADA Robot Model

Specify the location of an existing Universal Robot Description Format or COLLADA file for your robot

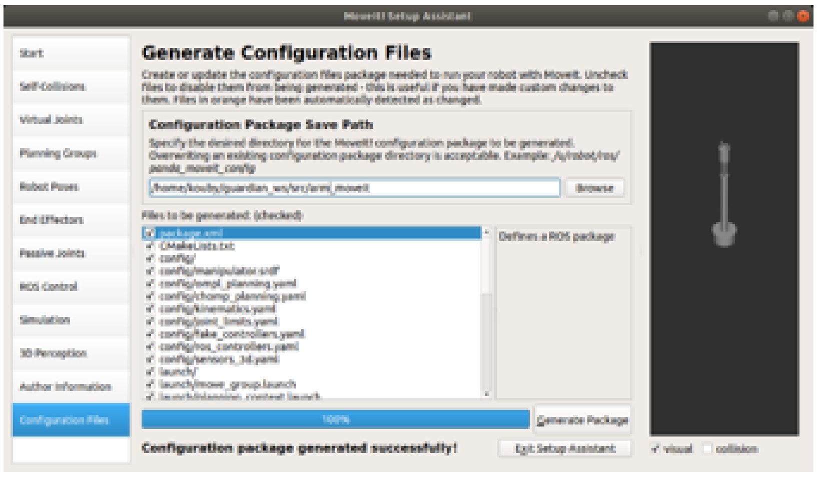
/home/kouby/guardian_ws/src/arm_description/urdf/arm_urdf.urdf

optional xacro arguments:

Success! Use the left navigation pane to continue.

Figure 2: MoveIt! setup assistant

In the MoveIt! setup, collision matrix is generated with low sampling density. In the planning groups section, a group name is defined and the kinematic solver used is 'kdl_kinematics_plugin/KDL KinematicsPlugin' and then the joints are added. The defined joints are revolute as specified in the SolidWorks export process. The links and joints are defined for the robotic manipulator, a planning group has been defined for the arm and then another planning group is to be defined for the gripper, similarly, the links and joints are defined in this group. In the robot poses section, a predefined set of values for each joint can be set. These values are used during the motion planning. The robot poses can be distinguishably set for the arm and the gripper. Link 6 which is the end-effector is set as the parent link for the gripper. The passive joints section is left empty. Under the ROS control section, the controller type is set as 'position_contollers/JointTrajectoryController' and similarly, a controller is set for the gripper.



Generate Configuration Files

Create or update the configuration files package needed to run your robot with MoveIt. Uncheck files to disable them from being generated - this is useful if you have made custom changes to them. Files in orange have been automatically detected as changed.

Configuration Package Save Path

Specify the desired directory for the MoveIt! configuration package to be generated. Overwriting an existing configuration package directory is acceptable. Example: /u/robot/ros/panda_moveit_config

/home/kouby/guardian_ws/src/arm_moveit

Files to be generated: (checked)

```
✓ package.xml
✓ CMakeLists.txt
✓ config/
✓ config/manipulator.srdf
✓ config/ompl_planning.yaml
✓ config/chomp_planning.yaml
✓ config/kinematics.yaml
✓ config/joint_limits.yaml
✓ config/fake_controllers.yaml
✓ config/ros_controllers.yaml
✓ config/sensors_3d.yaml
✓ launch/
✓ launch/move_group.launch
✓ launch/planning_context.launch
```

Defines a ROS package

Configuration package generated successfully!

Figure 3: MoveIt! package generation

The definition of the planning groups and the ROS controllers is completed. In the simulation section, the necessary URDF file is generated and that data is transferred to the main URDF file.

The final step under MoveIt! is the generation of the package. The required file name is provided to the configuration package save path which generates the package necessary for motion planning as shown in figure 3.

D. Launching the model with gazebo and RViz

The package generated by the MoveIt! setup contains two folders the config and the launch. The launch files contain the necessary files to open RViz and Gazebo [9]. To launch the required files we need to add the joint_state_controller in the ros_controller.launch file. The motion planning is performed by launching the gazebo, RViz, and the move_group planner. The gazebo launch file is to spawn the model developed into a simulation world as shown in figure 4 and the first visual output of the robot is seen [7]. This is as per the designed CAD model.

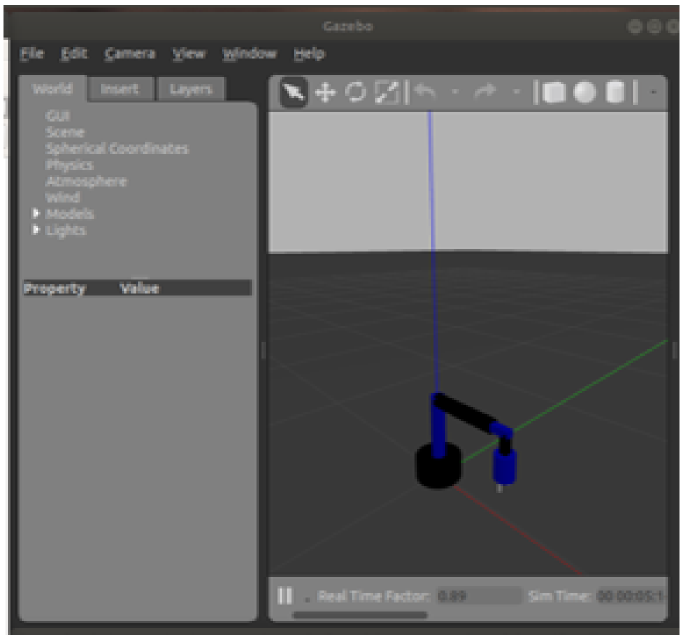
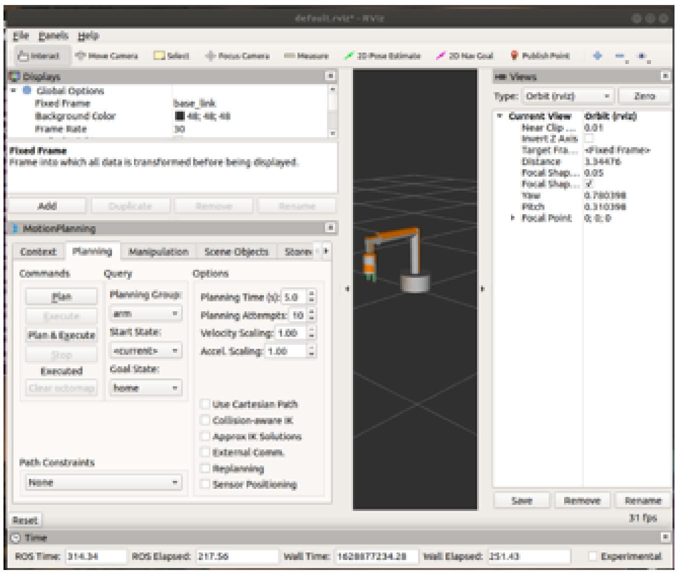


Figure 4: Robot launched in gazebo

The colour combination of the model can be defined in SolidWorks or can be customized in the URDF file. Further objects can be added and surroundings can be changed if required. To operate with RViz move_group planner file is launched. The RViz launch file is launched, the fixed frame in RViz is changed to base_link and the robot model is added. The added object in gazebo can also be added here.

E. Motion planning and point to point motion

The move_group planner launch file is responsible for the motion planning. In RViz, motion planning is added and loaded. Now, as provided in the MoveIt! setup the planning group contains two sections the arm and the gripper which will enable motions to be planned for the arm and the gripper. The pre-defined planning is done with the robot poses defined in the MoveIt! setup. Once, the initial state and the final state have been defined, the path can be planned. Similarly, manual planning is possible through the joints tab. The planned path is visualized in RViz. The planning is monitored and the feedback is obtained in the move_group planner terminal. Then the execute command is implemented in RViz as shown in figure 5 and the planned path is executed in gazebo.

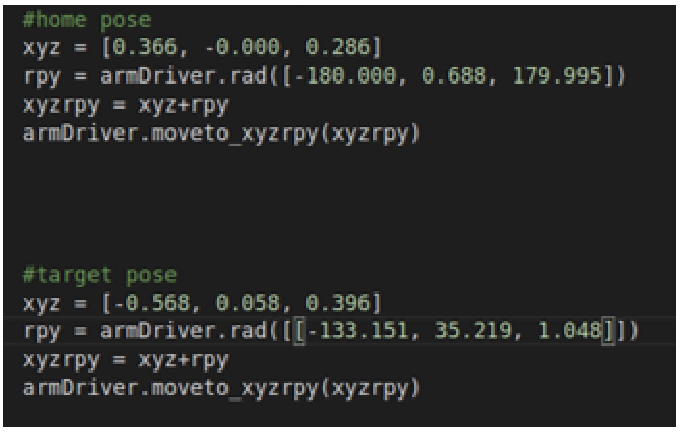


```
Global Options
  Fixed Frame         base_link
  Background Color    48; 48; 48
  Frame Rate          30
Fixed Frame
Frame into which all data is transformed before being displayed.

Query
  Planning Group: arm
  Start State: <current>
  Goal State: home
Options
  Planning Time (s): 5.0
  Planning Attempts: 10
  Velocity Scaling: 1.00
  Accel. Scaling: 1.00
  Use Cartesian Path
  Collision-aware IK
  Approx IK Solutions
  External Comm.
  Replanning
  Sensor Positioning
Path Constraints
  None

Type: Orbit (rviz)
Current View    Orbit (rviz)
  Near Clip ...   0.01
  Invert Z Axis
  Target Fra...   <Fixed Frame>
  Distance        3.34476
  Focal Shap...   0.05
  Focal Shap...   ✓
  Yaw             0.780398
  Pitch           0.310398
  Focal Point     0; 0; 0

ROS Time: 314.34   ROS Elapsed: 217.56   Wall Time: 1628877234.28   Wall Elapsed: 251.43   Experimental
```

Figure 5: Motion planning

The planning group is then changed to the gripper which contains two states, open and close. These are used to grab and release the object. This commences the motion planning.

To combine one or more planning motions a code is run interfacing all these components. In the MoveIt! config file a new folder is created called scripts which contains the necessary coding to move the end-effector from one point to other. The desired positions and orientations are planned and executed.



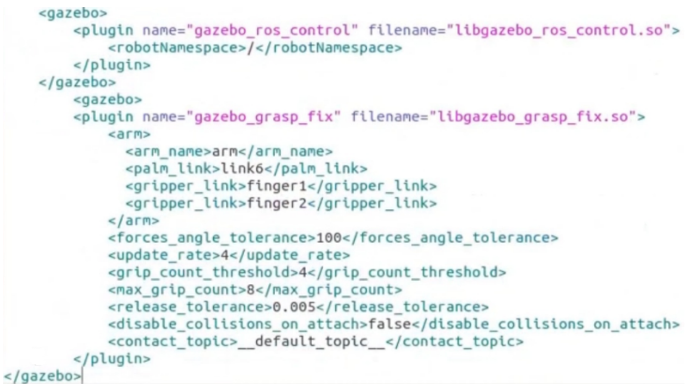
```
#home pose
xyz = [0.366, -0.000, 0.286]
rpy = armDriver.rad([-180.000, 0.688, 179.995])
xyzrpy = xyz+rpy
armDriver.moveto_xyzrpy(xyzrpy)



#target pose
xyz = [-0.568, 0.058, 0.396]
rpy = armDriver.rad([[-133.151, 35.219, 1.048]])
xyzrpy = xyz+rpy
armDriver.moveto_xyzrpy(xyzrpy)
```

Figure 6: Code for point-to-point movement

The position and orientation include translation and rotation in RPY angle as shown in figure 6. The code contains several functions to implement point-to-point movement and the main function is the inverse kinematic function which gives the position of each joint. The current position and orientation and the final position and orientation of the end-effector are provided to the code by the corresponding topic. The points in which the end-effector needs to move are also given to the code which is planned. The inverse kinematic function generates the joint angles for all the joints and is then executed [11]. The same process is carried out for different poses and the poses are updated in the main code. The code is then executed to move from one point to another.

F. Object grasping and object relocation

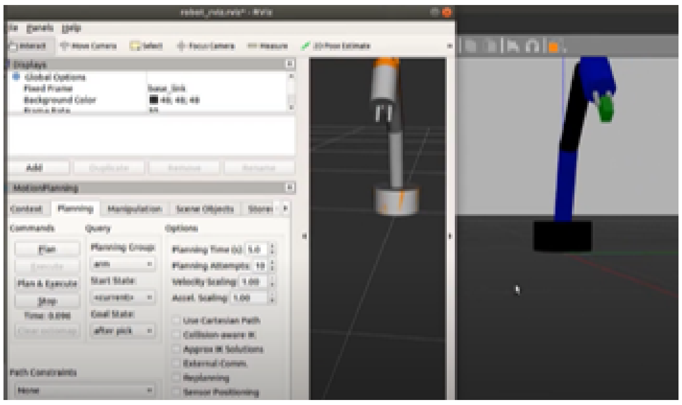
The gazebo is first launched and the desired object is created, then this world is saved. The RViz file is launched, the robot model and the motion planning are added, then saved. A launch file is created which includes the newly made files of gazebo, RViz, and move_group planner. Similarly, various objects can be designed based on the requirement. The MoveIt! configuration can be edited to add various robot poses for the motion planner. The predefined poses include 'pick pose', 'after pick pose', 'place pose', 'after place pose', and 'home pose'. The arm reaches the object and the gripper grasps it. When attempted to lift the object, it slips from the gripper.



```
    <gazebo>
        <plugin name="gazebo_ros_control" filename="libgazebo_ros_control.so">
            <robotNamespace>/</robotNamespace>
        </plugin>
    </gazebo>
        <gazebo>
        <plugin name="gazebo_grasp_fix" filename="libgazebo_grasp_fix.so">
            <arm>
              <arm_name>arm</arm_name>
              <palm_link>link6</palm_link>
              <gripper_link>finger1</gripper_link>
              <gripper_link>finger2</gripper_link>
            </arm>
            <forces_angle_tolerance>100</forces_angle_tolerance>
            <update_rate>4</update_rate>
            <grip_count_threshold>4</grip_count_threshold>
            <max_grip_count>8</max_grip_count>
            <release_tolerance>0.005</release_tolerance>
            <disable_collisions_on_attach>false</disable_collisions_on_attach>
            <contact_topic>__default_topic__</contact_topic>
        </plugin>
</gazebo>
```

Figure 7: Gazebo grasp plugin

This is because the URDF file does not contain the gazebo grasp plugin and other parameters as shown in figure 7. These are added to the URDF file and the pick and place operation is resumed [8]. The predefined poses are then planned and executed. The addition of gazebo grasp plugin ensures that the robotic arm grasps and relocates the object as shown in figure 8.



Planning Group: arm
Start State: <current>
Goal State: after pick
Planning Time (s): 5.0
Planning Attempts: 10
Velocity Scaling: 1.00
Accel. Scaling: 1.00
Path Constraints: None

Figure 8: Gazebo grasp plugin

3. Conclusion

The CAD model of a 6-DoF robotic manipulator was successfully designed and implemented on Robot Operating System using MoveIt!. Also, the pick and place operation using motion planning was implemented.

4. Future scope

The implemented 6-DoF robotic manipulator can be integrated with a vision system that can be used for vision augmented robot feeding [2] and can be used for vision-based sorting applications. The major industrial applications are fast packaging and fast assembly [4].

References

[1] N. Bratovanov, *Robot modeling, motion simulation, and off-line programming based on solidWorks API*, Third IEEE Int. Conf. Robotic Comput. 2019.

[2] A. Candeias, T. Rhodes, M. Marques and M. Veloso, *Vision augmented robot feeding*, Proc. European Conf. Computer Vision (ECCV) Workshops (2018) pp. 50–65.

[3] S. Chitta, *MoveIt!: An Introduction*, Robot Operat. Syst. (2016) 3–27.

[4] A. Farooqi, N.B. Yusoff and L.S. Chung, *Automatic pick-and-place packaging system with vacuum lifter*, IOP Conf. Ser. Materials Sci. Engin. 697 (1) (2019) pp. 012008.

[5] S. Hernandez-Mendez, C. Maldonado-Mendez, A. Marin-Hernandez, H.V. Rios-Figueroa, H. Vazquez-Leal and E.R. Palacios-Hernandez, *Design and implementation of a robotic arm using ROS and MoveIt!*, IEEE Int. Autumn Meeting on Power, Electronics and Computing (ROPEC) 2017.

[6] H.R. Kam, S.H. Lee, T. Park and C.H. Kim, *RViz: a toolkit for real domain data visualization*, Telecommun. Syst. 60(2) (2015) 337–345.

[7] N. Koenig and A. Howard, *Design and use paradigms for gazebo, an open-source multi-robot simulator*, IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS) (IEEE Cat. No.04CH37566), 2004.

[8] E. Mingo Hoffman, S. Traversaro, A. Rocchi, M. Ferrati, A. Settimi, F. Romano and N.G. Tsagarakis, *Yarp Based Plugins for Gazebo Simulator*, Lecture Notes in Computer Science, 2014.

[9] S. Putz, T. Wiemann and J. Hertzberg, *Tools for visualizing, annotating, and storing triangle meshes in ROS and RViz*, European Conf. Mobile Robots 2019.

[10] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler and A.Y. Ng, *ROS: an open-source robot operating system*, ICRA Workshop on Open Source Software 3 (2) (2009).

[11] A.D. Souza, S. Vijayakumar and S. Schaal, *Learning inverse kinematics*, IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the Next Millennium (Cat. No.01CH37180). 2001.

[12] Y. Zhang, B.K. Chen, X. Liu and Yu Sun, *Autonomous robotic pick-and-place of microobjects*, IEEE Trans. Robot. 26(1) (2010) 200–207.