Setup of the Yaskawa SDA10F robot for industrial applications, using ROS-Industrial

Carol Martinez, Nicolas Barrero, Wilson Hernandez, Cesar Montaño, Iván Mondragón
Pontificia Universidad Javeriana, Department of Industrial Engineering, School of Engineering
Cra. 7 No 40-69, Bogotá, Colombia
carolmartinez@javeriana.edu.co

***Abstract*—The ROS-Industrial open source project was intended to expand the applicability of industrial robots by reducing the gap that existed between researchers and manufacturers, in order to start developing state of the art applications for the industry. This is part of the scope of the PIR (Perception for Industrial Robots) project from the Pontificia Universidad Javeriana Bogotá (PUJ). PIR project looks for expanding robot's capabilities for working in dynamic industrial environments. This paper presents the starting point of this project: the setup of the Motoman SDA_10 dual-arm robot with ROS-Industrial. In this paper, the different steps and modifications carried out to setup the system are presented. Simulation experiments and real tests were conducted in order to analyse and check the behaviour of the system. Results show some of the advantages of using ROS-Industrial, and the wide range of tasks that can be explored with these kind of robots from the research and application point of views (in Latin America only a few robots with this type of configuration is available for research purposes).**

I. Introduction

In industrial processes automation systems are an essential factor in reaching high levels of productivity. In the last years, the application of robotics in industrial automation has taken an important role. With more efficient processes and demanding productivity indicators required, the implementation of automated systems, based on robotics, becomes an invaluable improvement opportunity for enterprises. Those systems would be more useful if they could be adapted to work in different tasks, and also if they could recognize changes in the environment where these activities are developed.

A huge challenge when using robotics in automated processes is configuration. Each process has specific requirements that must be arranged in the robot previously. Industrial robots have been limited to work in the same kind of tasks, such as handling and dispensing objects because they are cost-effective for repetitive and high volume tasks, but are not cost-effective for high-mix/low-volume production [1]. A disadvantage of developing applications based on robotics in industrial environments, is the complexity of writing the software, since the proficiency required is beyond the skills of only one researcher. Trying to solve this situation, robotics researchers have developed different frameworks, that make it easier to manage the complexity of writing software. These frameworks also help simulating the different applications without requiring to use the real hardware [2], making it possible to test different feasible sceneries and choosing the optimal one without spending money in unnecessary tests.

An important framework for researchers is ROS (Robot Operating System) [3]. It promises to make designing robotic software easier and less expensive. This framework provides software for different robots, especially the ones most used in industry.

It is important to remark that ROS is not just an operating system, it also provides software modules for performing typical robot activities, for example object recognition. ROS makes possible to simplify the task of writing complex software, so the design of the robotic application for a company could be done in shorter time.

Currently, the biggest challenge for ROS is to become the dominant robotics platform.[4]. The software is distributed under BSD license and is free for anyone. It is used with Linux operating systems, so researchers can use ROS for commercial or non-commercial purposes. In fact, the ROS community has developed ROS-I [5], which stands for ROS-Industrial, as an open-source project that is focused in industrial robotics applications, making the programming of robots in the industry even easier, in addition to bringing support and documentation for industrial robots.

Based on ROS-Industrial, the Centro Tecnológico de Automatización Industrial (CTAI) [6] at Pontificia Universidad Javeriana Bogotá launched the PIR project: Perception for Industrial Robots. This project aims to incorporate different sensors to allow industrial robots to adapt easily to different tasks (adding avoidance capabilities, which are commonly found when working in dynamic environments).

PIR project will start developing research on the Motoman SDA10F Yaskawa dual-arm robot [7], and will focus on finding industrial applications of this robot, where the robot's adaptability to a complex environment is required for improving productivity of companies.

This paper presents the different steps and modifications carried out to setup the Motoman SDA10F robot with ROS-Industrial. Simulation experiments and real tests are conducted in order to analyze the behavior of the system highlighting the advantages of using ROS-Industrial.

The paper is organized as follows. First, an analysis of state of the art of industrial robots is presented. Then, the hardware and software architectures of the Motoman SDA10F robot is presented. Section IV presents tests and results. Finally, Section V presents the conclusions and the direction of future work.

II. State of the art

During the last few years, interest in anthropomorphic or dual-arm manipulators had increased. In the industry, it is expected that anthropomorphic robots can replace human workers in the future. These kind of robots are dual arm manipulators, whose main characteristic is the way of manipulation. It makes distinction between non-coordinated (the two arms are performing two different tasks) and coordinated manipulation (where the arms perform different parts of the same task)[8].

The use of a dual arm robot system for performing operations is currently being studied. For example, [9] presented a case study of the final assembly area of an automotive plant. The study presented the mechanical and programming aspects of using a dual arm robot in activities that are typically performed by humans. The implemented system is based on a COMAU Smart Dual arm robot platform. They have found that by using dual arm robots the cost of setting up a cell decreased, avoiding the cost of investing in expensive grippers which are required when conducting the same task with single arm robots. Additionally, the programming tasks are simplified due to the synchronized movements that already come with dual arm robots.

In [11] a computationally efficient and robust kinematic calibration algorithm for industrial robots is presented. The HP20D manipulator provided by Yaskawa (Yaskawa Motoman Robotics, Inc. manipulators are known to be fairly accurate) with the DX100 controller, were used to test the algorithm. They tested 135 points through defined trajectories. Future work is focused on improving the processing speed, because the current prototype is in MATLAB without hardware acceleration.

A method based on human-likeness to solve the manipulation planning problem for articulated robots, was presented in [12]. To test the algorithm, the Motoman SDA10D industrial robot was used. All the manipulation planning experiments were carried out for the right arm, and the MATLAB Robotics Tool box was used to carry out basic robotics calculations, such as robot forward kinematics.

Programming industrial robots is a very demanding task. Each manufacturer has its own communication protocols and programming interfaces which make the development of industrial applications a difficult task. For this reason, the Robot Operating System (ROS) is being widely used for dealing with different robots. ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [15]. ROS simplifies the task of programming robots by providing a robust framework where the designers are provided with direct control for the robot. Typically, a ROS implementation has the components shown in Fig. 1, [14]:

- Nodes: basic processes that perform computation.
- Topics: a method for exchanging messages.
- Services: provides a stricter communication model where there is an established request and response message.

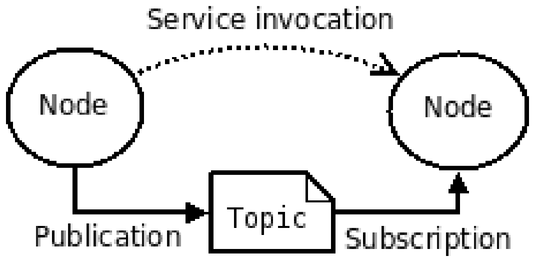

Fig. 1: Basic components for a ROS implementation [14]

Using ROS framework, in [14], a reconfigurable control system was developed. They present results with two robotic arms which require reconfiguration of the underlying control system in order to to carry out a task.

In [17] presents a simulation environment of an industrial manipulator based on ROS. The system structure consists of ROS as a middleware, visualization system, and simulation system. The robotic visualization simulates a five-joint augmented SCARA robot. The links and joint of the robot are defined as the Unified Robot Description Format (URDF) which is used by Gazebo to simulate the robot. A ROS based architecture is used in [18] to the coordination of human robot cooperative assembly tasks. The paper presents simulation results where an operator and a dual arm robot share assembly task and workspace.The assembly sequence is modeled in XML format, generated off-line. The COMAU dual arm robot is simulated and using ROS messages, the database communicates with the model retrieving, visualizing, and sending information to the robot controller through the TCP/IP communication protocol.

Yaskawa Motoman Robotics announced in 2004, his sponsorship in the Amazon Picking Challenge. Yaskawa provided robots, software (including MotoRos Driver), and technical support at the event [19]. On 2016, a team composed by collaboration between TU Delft Robotics Institute and the company Delft Robotics, won the Amazon Picking Challenge. The team built a flexible robot system based on industry standards. The system is equipped with a single arm Yaskawa Motoman robot with seven degrees of freedom, high-quality 3D cameras, and an in-house developed gripper. To control the robot, the team integrated advanced software components based on state of the art artificial intelligent techniques and robotics. The components are developed under the Robot Operating System for industry framework (ROS-Industrial) [20]. The ROS package used by the team is an open source system[21].

The collaboration between industrial robot manufactures, research institutions, and the use of open source software, allows to a rapid penetration of intelligent robot systems on modern industries. Several logistics companies (Amazon included) have announced new challenges and competitions that involve the use of industrial robots in applications like picking, packing, transportation, among others [19]. These competitions will improve the development of open robot control architectures, suitable for real time object recognition, tracking, and manipulation.

III. The SDA10F Motoman Robot

The industrial robot presented in this paper is the dual-arm robot SDA10F from Yaskawa [7] (see Fig. 2). Each arm has seven (7) degrees of freedom (DOF), and there is an additional axis of rotation in the base (torso), for a total of fifteen (15) DOF. The robot at Pontificia Universidad Javeriana (PUJ) features an additional axis (not used in this paper), which allows the robot to have linear displacement (forwards-backwards or from left to right).

Each arm of the SDA10F robot can be operated independently, or in a simultaneous way along with the torso. This robot is commonly used for material handling, machine tending, picking/packing, assembly, load and unload, and other applications, where the manipulation of parts is required.

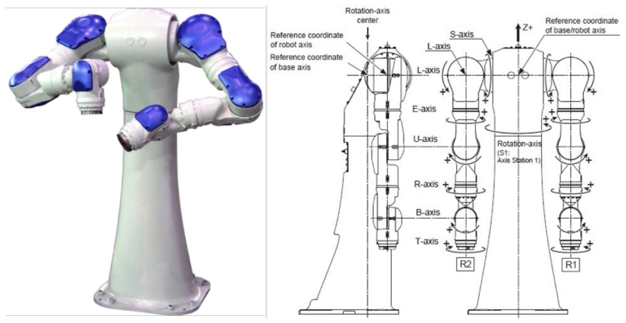
Rotation-axis center
Reference coordinate of robot axis
Reference coordinate of base axis
Reference coordinate of base/robot axis
S-axis
L-axis
E-axis
U-axis
R-axis
B-axis
T-axis
Rotation-axis (S1: Axis Station 1)
R2
R1

Fig. 2: Robot SDA10F. Image taken from [7]

Figure 2 shows the 15 axes of a typical configuration of the SDA10F robot (axes are represented by capital letters). The joints have the following ranges of motion: S 170°, L 180°, E 110° , U 135°, R 180°, B 110°, T 180°, and the torso (S1) 180°. Also the maximum speed of each joints is, S 170°/s, L 170°/s, E 170°/s, U 170°/s, R 200°/s, B 200°/s, T 200°/s and the torso 180°/s. The repeatability is $\approx$ 0.1 mm, its approximate weight is 220 kg, and the payload is 10 Kg per arm.

Figure 3 shows the work space of the robot. From the figure, it can be seen that the maximum range in the horizontal axis is 1970 mm and for the vertical one is 1440 mm. The way the robot is positioned is with an absolute encoder .

The controller of this robot is the FS100 from yaskawa [7]. This controller provides high performance in applications such as picking, packing, and parts handling tasks. This controller has the advantage of offering an open software architecture, which allows the creation of customized industrial applications using C/C++/C#/and .NET. The controller supports MotoSync, MotoPlus, MotoCom, and ROS-Industrial software environments, and allows communication protocols such as Fieldbus and Ethernet. Additionally, the controller has 16 digital inputs and 16 digital outputs, along with a safety module where it is

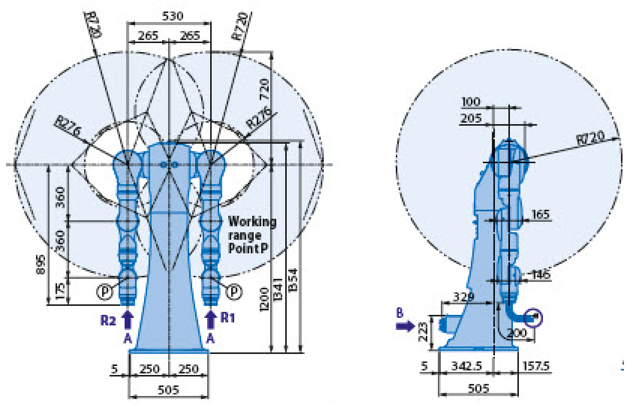

Fig. 3: Workspace of Robot SDA10F. Image taken from [7]

possible to connect extra push-buttons for an emergency stop and indicator lights.

The PUJ robot has two-different end-effectors from the company ROBOTIQ [24], one in each arm. A two-fingers gripper, as the one shown in the left side of Fig. 4, which has a load force of 5 kg, and 85 mm stroke. The grip force range is from 5 N up to 220 N. The second gripper is a three-fingers gripper, shown in the right side of Fig 4. This gripper has a wider aperture range (155 mm), the force range is from 15 N up to 60 N, and its load force is 10 kg. Both grippers (two and three fingers) are equipped with different sensors: position sensor, force, and speed.

Fig. 4: Gripper robot SDA10F. Image taken from [24]

A. *Software Architecture*

The PIR (Perception for Industrial Robots) project at PUJ aims to augment the capabilities of industrial robots by incorporating different sensors in the robot's workspace. PIR project is based on the ROS-Industrial project. ROS-Industrial (ROS-I) [5] is an open source project which is used in this paper to interact with the robot. ROS-I comes with a simulation environment where it is possible to test the behaviors of different robots. Additionally, ROS-I features perception sensors such as cameras, laser, radar, etc. The main idea of ROS-I is to be able to develop software that can run in different robot platforms.

ROS-Industrial extends ROS (Robot Operating System) [3] capabilities to the industrial area. Therefore, the development that has been reached so far in the research robotic area with ROS, can be extended to different applications, including industrial robotics in order to extend the capabilities of industrial robots. The Motoman robot SDA10F used in PIR project is one of the robots included in ROS-I project. This allows the integration of different functionalities that come with ROS and ROS-I facilitating the developing and testing of different industrial applications.

Many functionalities of ROS can be applied in ROS-I. Figure 5 shows an example of how the information flows between ROS and ROS-I. ROS can be used to define the path for the robot, for example using MoveIt! (this occurs in the PC-side). ROS-I is in charge of decomposing the path into smaller points, and then those points are passed to the robot controller. In this part, ROS-I will be in both sides, in the PC side; and robot side, where the server (based on ROS-I) is installed.

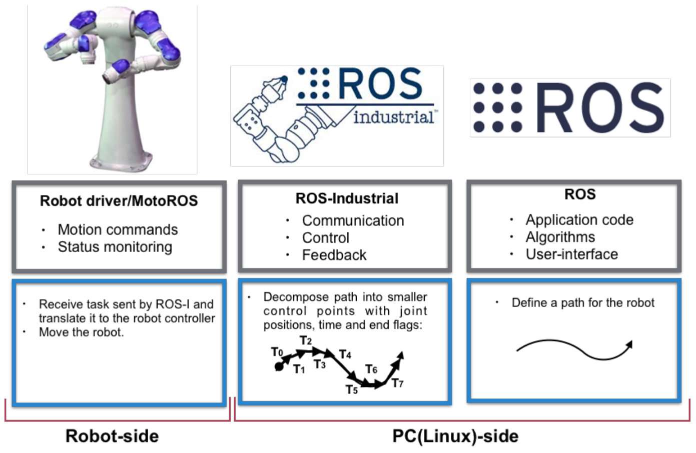

Fig. 5: ROS framework with industrial robots. Different functionalities of ROS can be used with ROS-I. ROS-I framework runs in the PC-side, as well as in the robot controller

Figure 6 shows the ROS-based architecture of the Motoman package. ROS-Industrial consists of 4 layers: some are generic to ROS-I, others depend on the robot manufacturer, and others still under development.

ROS-Industrial packages used in this paper includes:

- Simple Messages: defines the communication protocol to an industrial robot.
- Industrial Robot Client: library for communicating with an industrial robot.
- Vendor specific package (Motoman): this is a ROS package that allows communication with industrial robot controllers. It contains URDF models for different robots, and MoveIt! navigation packages with default configurations.

B. Motoman stack

Figure 6 shows the high level architecture of ROS-Industrial (ROS-I) when using the Motoman robot. ROS-I allows to bring

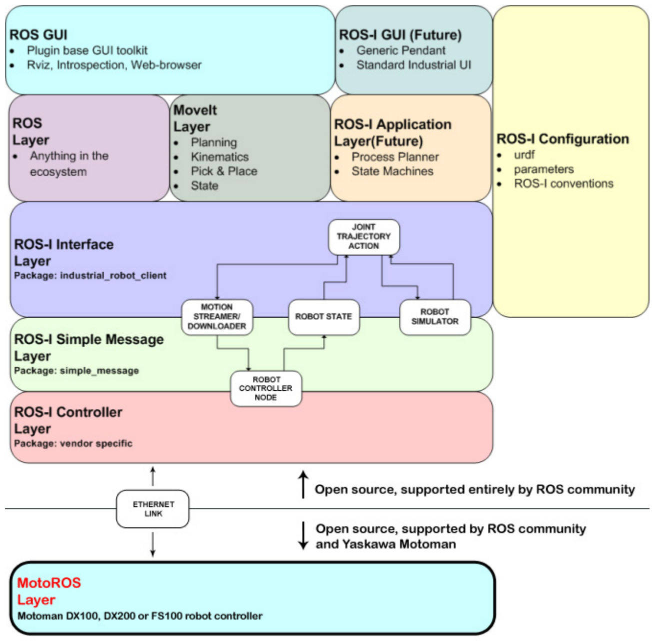

Fig. 6: Motoman package hierarchy. Image taken from [3]

well-known functionalities of ROS into the industrial robotics world, such as using MoveIt! for dealing with motion planning and kinematics, or using Rviz as a simulation tool.

The Motoman stack can be downloaded from [25]. It contains libraries, configuration files, and ROS nodes for controlling a Motoman robot using ROS-Industrial [3]. In our case, it contains the configuration files for the SDA robot with the FS100 controller, i.e. the SDA10F packages that comes with the stack.

The Motoman_driver is the communication interface between the controller and the ROS computer. It contains the ROS-I Interface Layer (see Fig. 6), the ROS-I Simple Message Layer, the ROS-I Controller Layer, and the MotoROS Layer. As shown in Fig. 6, the first three layers are located in the ROS computer, whereas the MotoROS layer is located on the robot controller. The communication is based on the simple_message package and some Motoman-specific messages. The MotoROS layer (MotoPlus folder inside the motoman_driver package) should be ordered from Motoman who will be the one in charge of configuring the robot controller for allowing ROS-Industrial integration with the robot.

On the other hand, the Motoman stack contains support packages (for the PUJ robot is the motoman_sda10f_support package). These packages contain meshes (normal and collision meshes) of the robot, URDFs (Unified Robot Description Format) which are XML formats for representing a robot model, and configuration files such as joints name definition and motion group definitions. The URDFs can be created with CAD models of the robot. It is important to mention that the FS100 robot controller does not support more than 4 groups (only groups right arm, left arm, torso1 and torso2). All launch files inside the support packages are in charge of establishing communication with the robot, and sending and receiving information to/from the robot.

Finally, the Motoman stack comes with a MoveIt! config package which will be described in more detail in the following subsection.

C. *MoveIt!*

MoveIt! (not a ROS-I package) [26] is a motion planning framework. It encapsulates software for motion planning (including collision detection and avoidance), manipulation, 3D perception, kinematics, control and navigation. It has been designed as an "easy-to-use platform for creating advanced robotics applications" [26]. The motion planning algorithms incorporated by MoveIt!, come from the Open Motion Planning Library (OMPL) [27], which contains implementation of different planners.

A MoveIt! package is created for a specific robot. The URDF robot data is required to create the package. Additionally, work-cell geometry can be included. Both URDF and work-cell geometry are used by MoveIt!, for visualization purposes and collision checking, among others. When creating a MoveIt! package (which is done following a set-up assistant GUI), it creates automatically all the required nodes, and generic config and launch files for the robot and/or sensors. In the set-up assistant GUI, it is possible to define motion groups which will be used for planning (very useful when working with dual arms robots, e.g. group right_arm, group left_arm), to define end-effectors, the kinematics solvers, and specific poses for a specific group (e.g. home position for left_arm group).

As mentioned before, the Motoman stack comes with a pre-configured MoveIt! packages for different Motoman robots (for the PUJ's robot, it is the motoman_sda10f_moveit_config). This package contains config files automatically created by the MoveIt! setup assistant. These files contain configuration information about the robot - .sdrf file- (links, predefined poses, end-effectors, motion groups, collision information, etc.), information about the controllers, the joint limits, the defined kinematic solvers for each group, and the motion planning algorithm.

There are two launch files in the MoveIt! config package that are used to test the package created and allows you to start testing motion planning in your robot in a straight forward fashion. The demo.launch lets you test if your moveIt! configuration is correct (setting up of the links, end-effectors, poses, motion groups, etc). This files does not create any connection to a real robot. It has a dummy joint-state-publisher which emulates the robot connection. When demo.launch is launched the Rviz simulator comes up with the robot in the scene. With this launch it is possible to move the end-effector of the robot manually, or creating random positions in order to plan and execute different motions with the robot.

The files generated by MoveIt! can be modified in order to provide information about a specific robot (controllers, etc) to allow communication between MoveIt! and the robot nodes.

The second launch file: the moveit_planning_execution.launch lets you test in simulation and with the real robot, the config files of your package. The first test should be offline in order to test if the MoveIt! package is well-configured. By default, that launch file lets you test everything with the industrial robot simulator. Once sure about the configuration this launch file can be used to interact directly with the real robot by passing as parameter the robot ip address and the robot controller type.

Both previously mentioned launch files open the planning environment (Rviz + Motion Planning plugin). It is possible to see the robot and if connected to the real robot, the simulated robot should have the same pose as the real one (see Fig. 10).

For planning, it is possible to do it directly in the planning environment by moving the end-effector markers to the goal state and then pressing plan and execute commands from the Motion planning plugin. On the other hand the same command can be send through a customize node using C++, which can be created to make the robot follow specific trajectories or perceive the environment and based on that move the robot towards a specific object. MoveIt! API for sending the motion request to the planners and for executing the commands in the simulated or real robot. Planning with both the GUI and by C++ let you plan with obstacle avoidance capabilities.

IV. Tests and Results

The following section presents some of the capabilities of the Motoman stack for controlling dual arm robots. In this section the PUJ's robot setup is presented. Additionally, an analysis of the strengths and drawbacks of the driver are discussed.

A. *Experimental setup*

- MotoRos setup: as can be seen in Fig 6 MotoROS should be configured in the controller (ROS Server). A pre-built binary is available in the Motoman Stack, and should be selected based on the robot controller. A guide of how to install the binary and the INFORM Code in the robot controller is found in [25]. Make sure you have loaded the .DAT, .JOB, .out files. Additionally, some controllers require modification of the ALL.PRM file from the controller. In the PUJ's case, this file was set by the support people from YASKAWA. It is important to make sure that none .out file is in the controller. We had to remove MotoSight2D.out MPLM.out. These two files were part of the camera that came with the robot. However, MotoROS requires that motoros.out has to be the only application running on the robot. Additionally, our MotoROS software had to be modified, this is due to the particular configuration of our robot (5 groups, we have and additional motor) and MotoROS allows to configure only four groups. Therefore, YASKAWA support team modified the MotoROS part to ignore our fifth group (the additional axis of the PUJ's robot, see Section III for more details), this is because the FS100 controller will never support more than 4 groups.

- Universal outputs checks: the universal outputs of the controller are used by MotoROS to indicate its state. In order to make sure the controller can receive remote commands, check the universal output signals in the pendant: OUT#0889 ROS_READY, OUT#0890 ROS_DONE, OUT#0891 INIT_DONE, OUT#0892 CONNECTION_SRV, OUT#0893 MOTION_SERVER, OUT#0894 STATE_SERVER OUT#0896 FAILURE. Output #899 should be ON, if the controller is well configured to receive motion from ROS. Output #891 confirms that the initialization was completed. Output #892, that the server threads are running properly, and Output #893 and Output #894 will only be ON when at least one client is connected, and should be OFF when the clients disconnect. if Output #896 is ON, a failure occurred and MotoROS should be reset (reboot the controller). For more details check document M2092-EDS [28].

- ROS computer setup: the computer used was a DELL Intel Core I5 2.6 GHz, 8GB RAM memory, running Ubuntu 14.04 with ROS-indigo installed with the following packages: motoman driver indigo-devel, ROS-Industrial indigo-devel, and MoveIt.

- Motoman Stack configuration: In order to be able to echoing information from the robot, as mentioned in [25], it was required to modify the file sda_10f_motion_interface.yaml. Instead of having groups IDs from 1-4, we had to modify them to 0-3. By doing this, it was possible to have communication with the real robot and to be able to read joints state information. This change is required only because of the non-standard configuration of the SDA10F robot the PUJ has.
- MoveIt! configuration: Motoman stack comes with a pre-configured MoveIt! configuration package (motoman_sda10f_moveit_config). By launching the demo.launch file of this package it is possible to check if the package is working properly. If so, you should be able to plan and execute commands, using the Motion Planning plugging from Rviz. Rviz should show the Robot, the goal positions you are defining and the motion executed in the simulated robot. In order to test the real robot, launch the motoman_sd10f_motion_streaming_interface.launch file passing the robot ip and the controller type, as arguments. The pendant of the robot must be in REMOTE mode. After launching this file, the current position of the robot should be shown in the Rviz simulated robot. The SDRF file that comes with the MoveIt! configuration package was modified using the MoveIt! setup assistant in order to define some poses for each motion groups. The five motion groups that are defined in the package are: arm_left, arm_right, torso, sda10f, and arms. Group sda10f allows to control all the joints of the robot, and the arms group allows to control both arms simultaneously. Some of the poses defined for the different groups were: "HOME_ARMS", "LEFT_FRONT", "RIGHT_ELBOW", "RIGHT_PICK", etc. See Fig. 7 that illustrates the different pre-configured poses.

B. Testing the ROS-I Motoman Stack

The MoveIt! API (Application Programming Interface) has different functions that let you define goal states, plan and execute them. In these tests we will show some of this functionalities using both the simulated and the real robot.

*1) **Pre-defined poses**:* When configuring MoveIt! with the MoveIt! setup assistant, it is possible to configure specific robot poses, for the different motion groups that has been defined. Figure 7 shows 4 of the multiple poses that were defined for the different motion groups of the Motoman robot. Figure 7 shows poses for the motion groups arm_left and arm_right: "LEFT_FRONT and RIGHT_FRONT", "LEFT_ELBOW and RIGHT_ELBOW", "RIGHT_PICK" and "LEFT_FRONT", and "RIGHT_PICK"-"LEFT_PICK" . These poses can be called from a ROS node using the MoveIt! API or by using the Rviz graphical user interface.

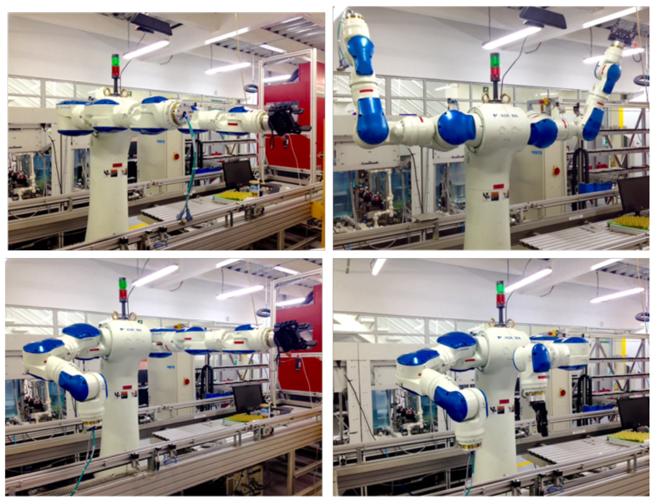
Fig. 7: Poses defined in the SDRF file from Motoman MoveIt— config file

*2) **Joint commands**:* Another way to interact with the robot instead of using predefined poses is by moving specific joints. Figure 8 shows how the position of the robot changes, when the value of specific joints are modified. The robot starts with its elbows bended, as can be seen in Fig. 8, left image. Then, the position of joints L and U (see Fig 2 for clarifying the names of the joints) is modified. Figure 8, right image shows the position of the robot, after modifying joints L and U.

C. Obstacle avoidance

One of the advantages of using ROS-Industrial is that many ROS functionalities can be used. MoveIt! is one of those. It

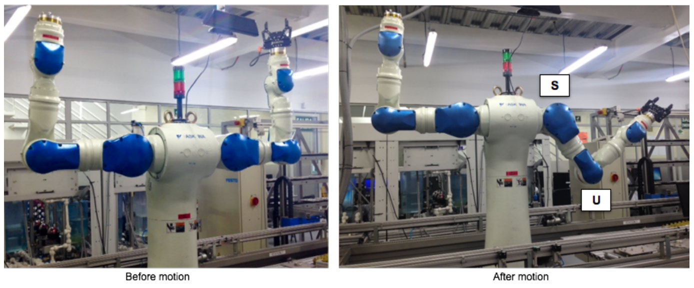

Fig. 8: Joints motion test. The MoveIt—'s API allows to send joint commands to the robot. The figure shows the robot when moving joints L and U. Left image shows the starting position, and right image the final position.

is a motion planning framework [26], and here in this test we show one of its great attributes: motion planning with obstacle avoidance capabilities. By creating the MoveIt! config package of the robot, it is straightforward to plan avoiding obstacles.

In the following tests, two ROS nodes where created: one for publishing objects in the scene, and the second one is in charge of planning and executing a specific trajectory. The trajectory was programmed using functions from the MoveIt! API. The programmed trajectory moves the robot from position 1 (extended arm, see Fig. 9, number 1) to position 2 (see Fig. 9, number 2), and from position 2 to position 3 (moving towards the right arm of the robot, see Fig. 9, number 3).

MoveIt! comes with a simulator based on Rviz that lets you plan the robot's movements without requiring the real hardware. All the motions that occur in the simulator are the ones the real hardware will follow.

Figure 9 shows the path followed by the simulated robot when moving towards the three previously mentioned positions. The first row shows the motions when no object was in the environment. The second row, shows the simulated path, followed by the robot, when a green box is placed on the conveyor. Comparing figures shown in rows 1 and 2, it is possible to see how the planning algorithm gets in charge of finding collision-free path for reaching the desired positions.

A closer look at the trajectories when the robot moves from position 2 to 3 (see Fig. 9), let us see that, if the motion planning did not have obstacle avoidance capabilities, the robot had crashed with the box. Therefore comparing both trajectories from Fig. 9 shows that indeed, MoveIt! plans different path based on the objects in the scene, without requiring to write a specific program for collision avoidance. MoveIt! has it already encapsulated in its motion planning algorithms which represents an big advantage for rapid applications development.

Figure 10 shows the robot following the trajectory defined for this tests (reaching the 3 previously explained positions). Comparing the images from the first and second row, it can be seen that the same positions reached by the simulated robot, are the ones followed by the real one.

*1) **Trajectories:*** In this test, we show how to program the robot to follow a specific trajectory. MoveIt!'s API allows you to stack different waypoints, and then execute them. These poses can be created using the Rviz GUI, by moving the simulated robot with the interactive markers and saving the different coordinates. Additionally, it is possible to generate the trajectory by specific increments of a previous pose.

For this test, the left and right arms were programmed to move following a square trajectory in the YZ plane. An initial pose was given to each arm, and then different way points were programmed by increments in the Y (left/rigth) and Z (up/down) axis of the robot. A ROS node was created to plan and execute the trajectory. Figure 11 shows the trajectory followed by the real robot, which is represented in real-time in Rviz using the simulated robot, when the real robot moves. The green line in the figure represents the trajectory of the end effector. Both arms performed movements that generated a square trajectory in the YZ plane.

Figure 12 shows representative images of the real robot when moving its right arm. The first image (from left to right) shows the arm positioning in the upper left corner of the square trajectory. In the second image, the arm is in the upper right corner of the square. The third and fourth images show the arm in the lower right corner and lower left corner of the square, respectively. A video of the test can be seen in [29].

V. Conclusions and Future Work

This paper presented the starting point of PIR (Perception for Industrial Robots) project at Pontificia Universidad Javeriana Bogotá. This project aims to augment the capabilities of industrial robots by including different sensors, that allows them to adapt dynamically to the environment. Therefore, expanding the robot's capabilities to different tasks.

In this paper, the configuration set up of the robot with ROS-Industrial was presented. Additionally, different capabilities of ROS and ROS-I ecosystems were described, providing a general view of the functionalities that can be used for programming an industrial robot.

During the tests, it was possible to experience the advantages of MoveIt!, not only for solving the planning problem, but also for providing a powerful simulation environment, that let conduct tests without requiring to be connected with the real robot.

The tests presented in this paper were conducted using the arm_left and arm_right groups of MoveIt!. Currently, the robot can be programmed, using ROS and ROS-I, for developing tasks that require the motion of the arms separately. However, simultaneous motion of the robot (arms + torso) or simultaneous motion of the arms (both arms at the same time) can not be achieved yet. This is due to the particular configuration of the robot at PUJ.

Future work in PIR project is focused on enabling the use of the other motion groups, and on incorporating cameras, that will be used to locate in real-time the position of objects; and a laser, that will provide 3D information of the objects present in the scene. By incorporating these sensors, it is expected to

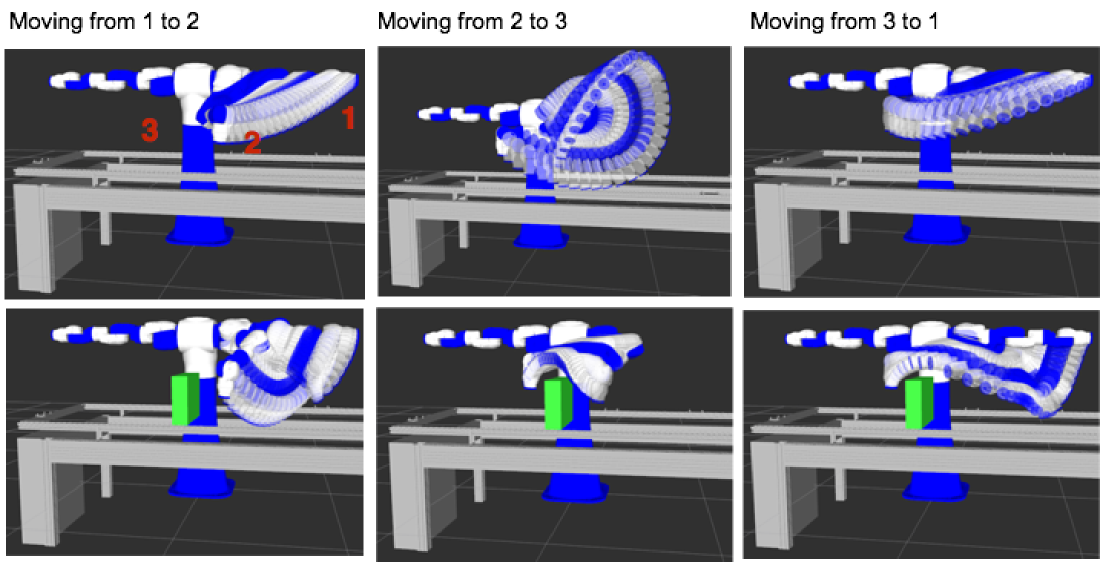

Fig. 9: Collision avoidance capabilities with MoveIt!. First row shows the motion of the robot without obstacles. Second row shows the motion of the robot when an obstacle was added to the scene. Check the different trajectories of the robot.

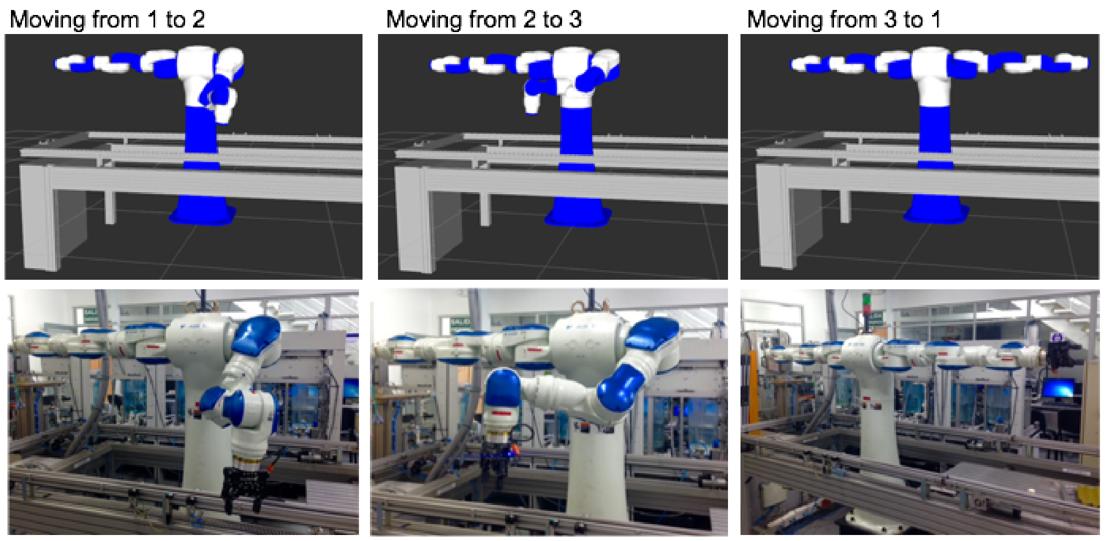

Fig. 10: Collision avoidance capabilities, real test. First row shows the motion of the simulated robot. Second row shows the motion of the real robot. Check that simulated and real robot position coincide.

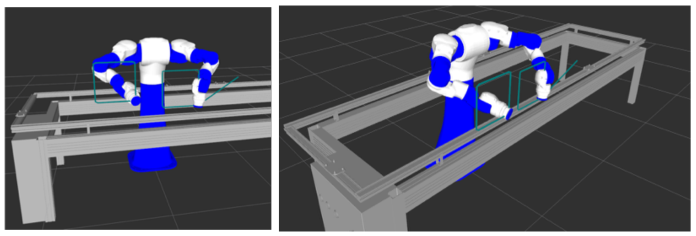

Fig. 11: Executing trajectories using MoveIt!: simulated robot. Right and left arms of the robot were commanded to follow a square trajectory. The image shows the simulation environment with the trajectory (green line) followed by the robot.

allow the robot to conduct different tasks, based on what it perceives.

ACKNOWLEDGEMENTS

We would like to thank the Pontificia Universidad Javeriana for supporting this research. We also want to thank the Centro Tecnolgico de Automatizacin Industrial (CTAI) for letting us use the installations and equipments for developing this research.

REFERENCES

[1] "The challenge." [Online]. Available: http://rosindustrial.org/the-challenge/

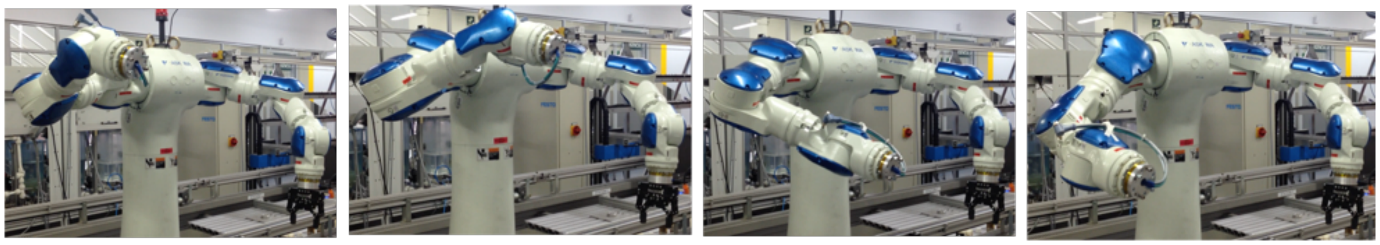

Fig. 12: Executing trajectories using MoveIt!: real robot. Right and left arms of the robot were commanded to follow a square trajectory. The image shows the real robot when reaching the four corners of a square trajectory.

[2] 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," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[3] ROS, "Robot operating system," http://www.ros.org, 2016.

[4] L. Garber, "Robot os: A new day for robot design," *Computer*, vol. 46, no. 12, pp. 16–20, Dec 2013.

[5] ROS-I, "Robot operating system-industrial," http://rosindustrial.org, 2016.

[6] Centro Tecnógico de Automatización Industrial, "Pontificia Universidad Javeriana, Bogot," http://www.javeriana.edu.co/blogs/ctai, 2016.

[7] Motoman, https://www.motoman.com/industrial-robots/sda10f, 2016.

[8] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic, "Dual arm manipulation—a survey," *Robotics and Autonomous Systems*, vol. 60, no. 10, pp. 1340 – 1353, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S092188901200108X

[9] P. Tsarouchi, S. Makris, G. Michalos, M. Stefos, K. Fourtakas, K. Kaltsoukalas, D. Kontrovrakis, and G. Chryssolouris, "Robotized assembly process using dual arm robot," *Procedia CIRP*, vol. 23, pp. 47 – 52, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827114011354

[10] F. Basile, F. Caccavale, P. Chiacchio, J. Coppola, and C. Curatella, "Task-oriented motion planning for multi-arm robotic systems," *Robotics and Computer-Integrated Manufacturing*, vol. 28, no. 5, pp. 569 – 582, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0736584512000191

[11] T. Messay, R. Ordóñez, and E. Marcil, "Computationally efficient and robust kinematic calibration methodologies and their application to industrial robots," *Robotics and Computer-Integrated Manufacturing*, vol. 37, pp. 33 – 48, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S073658451500071X

[12] Y. Gan and X. Dai, "Human-like manipulation planning for articulated manipulator," *Journal of Bionic Engineering*, vol. 9, no. 4, pp. 434 – 445, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1672652911601364

[13] Y. Motoman, "SDA10F." [Online]. Available: http://www.yaskawa.eu.com/uk/products/robotic/motoman-robots/productdetail/product/sda10f/

[14] J. M. Aitken, S. M. Veres, and M. Judge, "Adaptation of system configuration under the robot operating system," {*IFAC*} *Proceedings Volumes*, vol. 47, no. 3, pp. 4484 – 4492, 2014, 19th {IFAC} World Congress. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667016423050

[15] P. Publishing, Ed., *Learning ROS for Robotics Programming*. Packt Publishing, Sep. 2013. [Online]. Available: http://www.packtpub.com/learning-ros-for-robotics-programming/book

[16] E. Ruiz, R. Acuña, N. Certad, A. Terrones, and M. E. Cabrera, "Development of a control platform for the mobile robot roomba using ros and a kinect sensor," in *Robotics Symposium and Competition (LARS/LARC), 2013 Latin American*, Oct 2013, pp. 55–60.

[17] S. Ergur and M. Ozkan, "Trajectory planning of industrial robots for 3-d visualization a ros-based simulation framework," in *Robotics and Manufacturing Automation (ROMA), 2014 IEEE International Symposium on*, Dec 2014, pp. 206–211.

[18] P. Tsarouchi, S. Makris, G. Michalos, A.-S. Matthaiakis, X. Chatzigeorgiou, A. Athanasatos, M. Stefos, P. Aivaliotis, and G. Chryssolouris, "Ros based coordination of human robot cooperative assembly tasks-an industrial case study," *Procedia CIRP*, vol. 37, pp. 254 – 259, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212827115008987

[19] A. Robotics, "Amazon robotics challenge." [Online]. Available: https://www.amazonrobotics.com/pickingchallenge

[20] RobotValley TuDelft, "Team delft wins the amazon picking challenge 2016." [Online]. Available: http://www.robovalley.com/news/team-delft-wins-amazon-picking-challenge2/

[21] M. Bharatheesha, R. Burger, M. D. Vries, W. Ko, and J. Tan, "Motion module for the amazon picking challenge 2016 - team delft," August 2016. [Online]. Available: http://moveit.ros.org/moveit!/ros/2016/08/22/teamdelftamazon.html

[22] R. Industrial, "Motoman sda10f ros industrial." [Online]. Available: http://rosindustrial.org/news/2015/1/16/motoman-sda10f-ros-industrial

[23] Y. Motoman, "Yaskawa motoman announces new labview and ros-industrial support for motoman robots." [Online]. Available: https://www.motoman.com/media/pr/201301-labview-ros

[24] Robotiq, "Industrial grippers," http://robotiq.com, 2016.

[25] M. driver, "ROS-I driver for Mottoman robots," http://wiki.ros.org/motoman_driver, 2016.

[26] MoveIt, http://moveit.ros.org, 2016.

[27] OMPL, "The Open Motion Planning Library," http://ompl.kavrakilab.org, 2016.

[28] M. driver, "M2092-eds document. motoplus-ros especifications," http://wiki.ros.org/motoman_driver, 2016.

[29] Video, "Planning and executing trajectories with ros-i," https://drive.google.com/open?id=0B-0RPWkwjpOyTWk3VG0wUk5qekk, 2016.