Intelligent Motion Planning in Human-Robot Collaboration Environments

Zahid Iqbal, Liliana Antão, Vítor H. Pinto, Gil Gonçalves

SYSTEC, Research Center for Systems and Technologies
DIGI2, Digital and Intelligent Industry Lab
Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
Email: {zahid, lpsantao, vitorpinto, gil} @fe.up.pt

***Abstract*—A robot needs to have adequate Artificial Intelligence (AI) support for operating as a human co-worker, making its behaviour flexible and autonomous. Applications and development methodology for collaborative robots (cobots) differ substantially from the norm of traditional robot applications marked by pre-programmed, repetitive tasks with well-defined behaviours and little or no autonomy on the part of the robot. We present a modular solution for application development for cobots. Intelligent workspace monitoring, motion planning, and re-planning make for essential solution components. Our solution incorporates safety by design, imparts the robotic arm a partly autonomous behaviour and is a step in the direction of collaborative interactions with the arm. We implement perception and planning as separate modules and demonstrate how these modules integrate. The main focus of the work is the motion planning component, developed in Robot Operating System (ROS) and MoveIt. The proposed framework can receive multiple goal points, monitor safety thresholds, and account for many humans in the workspace modelled as dynamic obstacles. In particular, we show path planning with obstacle avoidance and report some performance measurement results of different built-in planning algorithms.**

***Index Terms*—collaborative robots, motion planning, sampling-based planning, MoveIt.**

I. Introduction

Several industries employed robots to improve production volumes and acquire better precision and accuracy in the overall production process. Traditional industrial applications (between the 1960s through 1990s) used robots for simple repetitive tasks with well-defined, pre-programmed behaviours. From the 2000s onwards, the development of robotic technology is driven primarily by advancements in the industrial Internet of Things (IoT) sensors [1][2], industrial wireless communication protocols [3], and software, in particular, AI techniques such as Machine Learning (ML), to make robots more autonomous. McMorris [4] and International Federation of Robotics (IFR) [5] nicely outline important milestones in technology development for robots. Typically, a sequence of steps makes up an industrial process. Along the process, human-robot interaction happens at defined points, for instance, when loading or unloading items. Such cases present obvious hazards due to the size of the robotic arm and proximity of the human [6]. Typical industrial environments confine robots to separate operation spaces isolated from human workers and bring robot operation to a halt if a human enters this space [7][8], safety being the primary concern.

A notable transition aims to position robots as co-workers for humans. While robots have proven efficient with hazardous, unpleasant or repetitive jobs, a complex process calls for creative thinking, flexibility, decision making or adaptability where the human role becomes crucial. When placed side-by-side with humans, we need to equip robots with strong AI making their behaviour nearly as skilful and flexible as that of humans. We call this approach Human-Robot Collaboration (HRC) [9]. To that end, we have cobots or collaborative robots that include some integrated safety features, such as force-limited joints and smooth surfaces to minimise impact hazards. Computer vision or similar techniques [10][11] that detect the presence of humans in the environment make for an essential component of the application developed for cobots. Most works on collaborative interactions concern safety aspects. Rybski et al. [11] and Bdiwi et al. [12] identify zones based on human position in the co-space and choose a safety operation accordingly. The work in [10] turns a standard industrial robot into a human-safe platform. It uses Light Emitting Diode (LED) markers for actors in the workspace and thus poses limitations for scalability and the presence of arbitrary persons in the co-space. Safety being a baseline, our work further explores task distributions using hierarchical task models and is promising for large industrial settings.

We present a modular solution that incorporates safety by design, imparts the robotic arm a partly autonomous behaviour, and is a step towards collaborative interactions with the robotic arm. We develop the work, in particular, for Universal Robot manipulator `UR5` [13]. Implementation is carried out within ROS [14], using the motion planning framework MoveIt [15]. We define distinctly *task planning* and *path planning*. We assume that target scenarios comprise specific task objectives that we can discretize in granular operations. A task planner automatically generates a graph (i.e., a hierarchical task model) in which nodes constitute sub-tasks or atomic operations [16]. Thus, it encapsulates the different operation sequences that

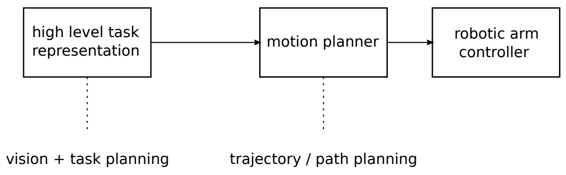


Fig. 1. Overall architecture

can fulfil a given task objective (root node of the graph). If the operation's actor is the robot, the task planner will input the specific coordinates that the path planner employs to compute a correct robot motion. For implementation, we divide the activity into two parts, namely, *perception* and *planning*. Perception provides the requisite vision information to the system about the robot workspace and the available objects, i.e., objects to manipulate, obstacles or humans. It already encapsulates the task planning referred to earlier. Path planning refers to robot motion planning that will take it to a target point while avoiding collision with workspace objects. We have decoupled the system making perception and path planning separate modules. The path planner uses the information provided by the perception module and guides the movement of the robotic arm to the target position. The work reported in this paper, in particular, encompasses motion planning. Having perception available as an independent module takes off the significant computational effort of the path planner. Conducting perception offline allows planners to process the workspace more efficiently. Fig. 1 shows a high level relationship of the main system components.

The presented work concerns industrial robotic arms or so-called manipulators. In particular, the proposed ideas apply to trajectory planning of `UR5`. The contributions of the paper are as follows:

(a) We provide an overview of the collaboration approach highlighting some aspects in which it differs from other works.
(b) We describe in detail the development steps of the motion planning component of our solution within ROS and MoveIt and provide some results that validate the implementation.

The rest of the paper is organized as follows. In the following section, we describe the tools and frameworks in which we have developed our solution. Section III presents the solution approach, i.e., how different modules integrate. In Section IV, we detail the development steps of our method. Section V presents some results from the implementation, in particular, the motion planning. Finally, in Section VI, we conclude the paper and indicate some future work directions.

II. Tool support for developing a motion planning application

In this section, we overview the main tools and framework used to develop the motion planning for the robotic arm.

A. Robot Operating System (ROS)

ROS provides a flexible platform for writing software for robots. Over the years, robot hardware, applications, and capabilities have significantly grown. A typical robotic system is complex, often combining several components that incorporate cameras, laser scanners, and odometry information. A functional robot system must include code for sensing, coordinate transforms, trajectory planning, navigation, hardware abstraction, control and more. For individual engineers, it becomes very challenging to cover each aspect of robot software development and then seamlessly integrate it into a complete system. ROS manages this complexity efficiently by providing tools, libraries, and open packages for standard robot functionality, allowing developers to reuse existing code from the available repositories and focus on their project-specific features. Such an approach lends fast and easy development of working prototypes for testing and experimentation in a simulated world, saving cost and time. Hardware interfaces allow running the program on the real robot. ROS supports C++ and Python for its API and Linux (Ubuntu) as its platform.

Within ROS, `roscore` is the system core providing a collection of nodes and programs. `roscore` contains three main functional module suits, which are `ROS Master`, `Parameter Server` and `rosout` logging node. The Master provides name registration and lookup for the rest of the nodes. The Parameter Server is a global key-value store through which nodes can share configuration information. `node` is the smallest unit of computation within ROS, i.e., an executable process. An application may distribute its functionality across several nodes. ROS maintains a peer-to-peer computation graph of ROS nodes which send and receive messages over named channels called `topics`. A package is the basic unit of ROS. It has the minimum structure to develop an application. It contains the application-specific configurations and launch files that allow running nodes from the same package and other packages. Concerning communication, there are different possibilities, i.e., topics, services, or actions. Topics follow a publish-subscribe model of asynchronous communication. Nodes publish to a topic to send a message and subscribe to it to receive a message; sender and receiver are decoupled. Service follows a request-response paradigm where communication is bi-directional and synchronized. The service client requests a service and the server responds to the request. Action is similar to service, however, it is used when the requested objective takes a long time to complete and intermediate feedback is necessary. The communication is asynchronous in this case (Fig. 2).

B. MoveIt

MoveIt is a set of software packages with specific capabilities for mobile manipulation, such as kinematics, motion planning and control, 3D perception and navigation. MoveIt runs on top of ROS and builds on the ROS messaging and build systems, and uses some of the common tools in ROS, e.g., the ROS Visualizer (RViz) and the Unified Robot

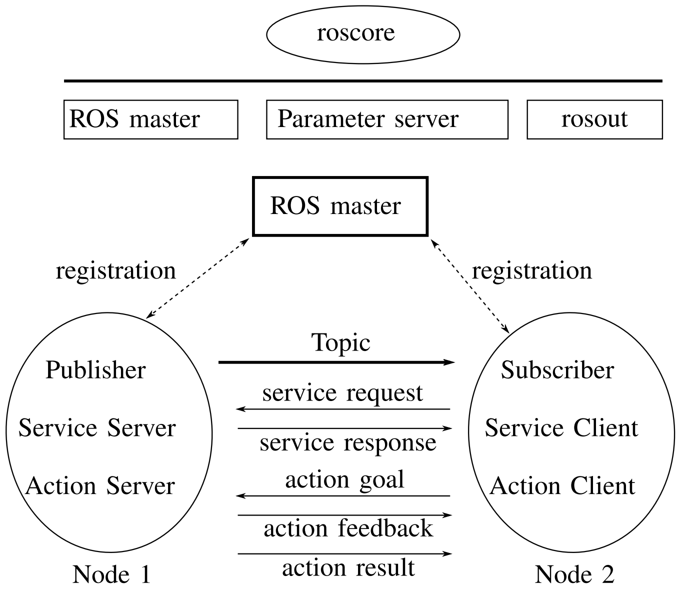


Fig. 2. ROS core communication system, and different message communication modes between nodes (adapted from [17]).

Description Form (URDF). It uses plugins for most of its functionality; motion planning (Open Motion Planning Library (OMPL) [18]), collision detection (default: Fast Collision Library (FCL) [19], and kinematics (default: OROCOS Kinematics and Dynamics Library (KDL) for forward and inverse kinematics for generic arms. A motion-planning framework such as MoveIt aims to lower the barrier of entry to robotic software. Additional to the underlying principle of code reuse and easy customisation as in ROS, MoveIt has specific design goals [20] primarily addressing the user-base without the breadth of knowledge to customise each toolchain with the right parameters or where time, effort or expertise needed to integrate different software components into the robot are considerable.

III. Proposed approach - an overview

Our solution to collaborative motion planning comprises three phases, namely *perception*, *pre-planning* and *planning* phases. The perception and pre-planning phases involve scene acquisition, voxelization of the workspace and generation of atomic operations. The planning phase is the focus of this work, which involves building collision-free paths between given start and goal positions. Fig. 3 presents the overall architecture of the motion-planning solution for collaborative environments.

The output of the perception phase is a voxel grid. A voxel represents a value on a regular grid in three-dimensional space, useful for many applications such as Dynamic Roadmaps [21] to create a mapping from an area in the workspace to states in the configuration space. For efficient path planning with obstacle avoidance and considering a collaboration scenario, an intelligent monitoring system is inevitable [22][23]. The present work uses the monitoring solution in [22]. We integrate the voxel-based grid with the motion planner. The voxel-based grid renders the entire scene of the collaborative environment as a grid composed of cubes with a given dimension, known as voxels. We can describe the granularity of the grid with the size of one side of the cube. The resolution can be set higher or lower by choosing the dimension of the unit cube in the grid, i.e., for a higher resolution, we choose a smaller size of the voxel, thus corresponding to more voxels in the grid, and vice versa. The pre-planning phase can produce voxel grids with varying degrees of information. A simple voxel grid is the occupancy voxel grid, which will set a voxel 1 or 0 depending if the corresponding workspace is free or occupied.

A. Mapping voxels to the system coordinates

A complementary framework loads the voxel grid into a 3d array within the planning solution in ROS and allows us to access the position of each voxel. Fig. 4 shows an overview of the approach. When the information is available, we segment regions of interest in the workspace, i.e., obstacles or humans. Corresponding voxels to such objects are occupied in the grid. With the distinct regions available, we can use specific algorithms to create bounding shapes around them, e.g., bounding boxes or spheres. ROS / MoveIt needs key dimensions, i.e., for a sphere it needs its radius and centre point, to render objects of interest in the simulation or real tests. Within MoveIt, we can calculate each point on the robotic arm and in the workspace in reference to the planning frame, which is the frame of the `base-link` of the arm, and its origin is at position $(0, 0, 0)$. In robotics, this frame is often referred to as the `world` frame. With this information, and knowing the dimensions of individual voxels and the hyper-cube that represents the complete voxel grid, we map the goal positions and obstacle positions from the voxel grid into the MoveIt so that we can test path planning with obstacle avoidance in real environments.

We define two structures, one representing a voxel-point vp in the grid VG. The other structure represents a point rp that we use to specify the position of any object within ROS, so-called real-point. The notion $vp_i(x)$ gives the value of x coordinate for ith voxel-point. Similarly, $rp_j(y)$ provides the value of y coordinate for a real-point j. Let sz denote the voxel size in the given VG, and let lx, ly, lz indicate the lower limits on the voxel grid. Then, we can find the value of the real coordinate points for a given voxel point, as shown in equations (2)-(4). Considering that coordinates are expressed in meters, we divide the final result by 100 in each case.

$$vp = (x, y, z) \tag{1}$$

$$rp_j(x) = \begin{cases} lx, & \text{if } vp_j(x) = 0 \\ \dfrac{lx + vp_j(x) \times sz}{100}, & \text{otherwise} \end{cases} \tag{2}$$

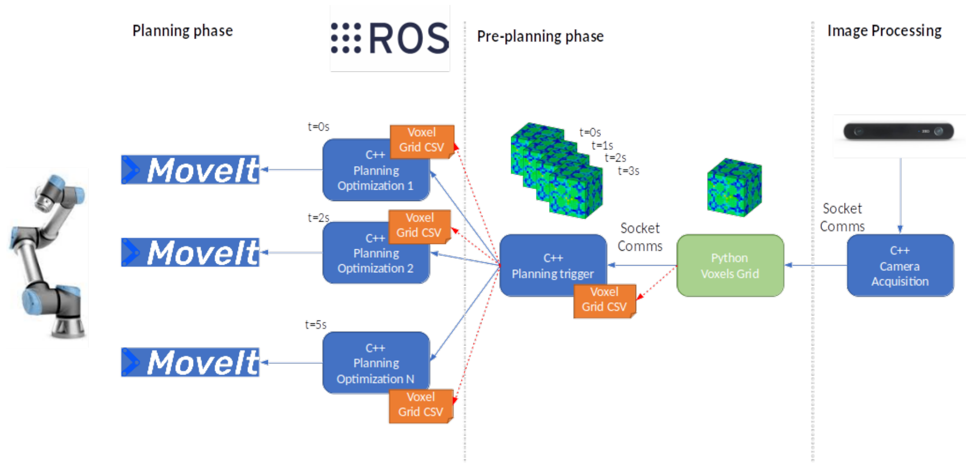


Fig. 3. Path planning solution architecture and module relationship.

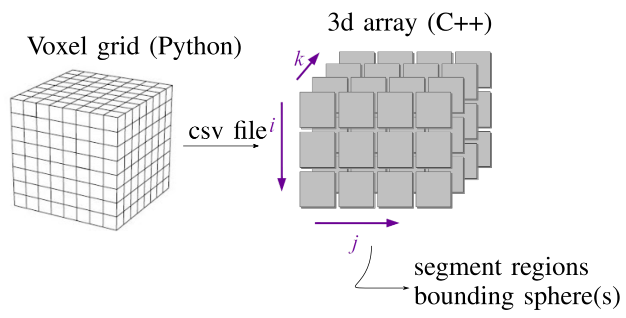


Fig. 4. An overview of the preliminary approach for integration of voxel grid within the planning solution.

$$rp_j(y) = \begin{cases} ly, & \text{if } vp_j(y) = 0 \\ \dfrac{ly + vp_j(y) \times sz}{100}, & \text{otherwise} \end{cases} \tag{3}$$

$$rp_j(z) = \begin{cases} lz, & \text{if } vp_j(z) = 0 \\ \dfrac{lz + vp_j(z) \times sz}{100}, & \text{otherwise} \end{cases} \tag{4}$$

In dynamic scenarios, the scene can change over time, and the robotic arm must be able to plan accordingly. We can input fresh voxel grids to the planning program with a certain frequency. The refresh rate of the voxel grid depends on the response time of the previous planning query. Currently, we are looking into solutions to address this issue.

IV. Development of the Motion Planning Solution

We detail the main steps required for manipulator control with ROS, including developing a model of the robot's physical structure, publishing coordinate transforms data and applying standard algorithms, such as path planning. We have used ROS industrial repositories of `UR5` [24], in particular, `ur_modern_driver` [25] to control the actual robotic arm.

A. *Modeling the workcell with URDF*

URDF is an XML format to describe the robot model in ROS. The principal components describing a robot model are the joints and links. The link component outlines the body by its physical aspects (dimensions, position of origin, colour etc.), and the joint describes the kinematic properties of the connection (axis of rotation, type of joint, connected links etc.). A joint connects two links, a parent link that precedes the joint and a child link that follows the joint. Such a topology makes a tree structure, where each link has precisely one parent, but can have multiple child nodes.

For the robot to move, we set the joint type as revolute and specify the axis around which the following child link will rotate. The visual tag in the urdf file lends a visual appearance to the respective element in the simulator. The visual representation of an element could be specified by primitive geometric shapes such as a box or a cylinder or by using carefully created meshes such as COLLAborative Design Activity (COLLADA) models. The `urdf` model of `UR5` used in this work uses meshes for defining visual components. In particular, the `urdf` model describes parts of the robot that do not change over time. Some examples include the topology, the relation between links and joints, and the complete link tree. Listing 1 shows part of the `urdf` model of the `UR5` robotic arm. The information in the `urdf` does not depend on the robot 3d position. For the ROS system to know about the robot model, a launch script loads the `urdf` file to the `Parameter Server`; `robot_description` is the name of the ROS parameter where the `urdf` file is stored on the `Parameter Server`. This format allows us to model other objects in the workspace, such as the table where the robotic arm is mounted.

```
<link name="base_link">
   <visual>
     <geometry>
       <mesh filename="package://ur_description/
            meshes/ur5/visual/base.dae"/>
     </geometry>
```

```
      <material name="LightGrey">
        <color rgba="0.7 0.7 0.7 1.0"/>
      </material>
    </visual>
    ...
  </link>
 <joint name="shoulder_pan_joint" type="revolute">
    <parent link="base_link"/>
    <child link="shoulder_link"/>
    <origin rpy="0.0 0.0 0.0" xyz="0.0 0.0
        0.089159"/>
    <axis xyz="0 0 1"/>
  <limit effort="150.0" lower="-6.28318530718" upper
      ="6.28318530718" velocity="3.15"/>
    <dynamics damping="0.0" friction="0.0"/>
  </joint>
```

Listing 1. Part of UR5 `urdf` file.

We can use a command line tool `check_urdf` to check the model validity. This tool attempts to parse the file and either prints a description of the resulting kinematic chain or an error message. Fig. 5 shows the output after running `check_urdf` on the UR5 `urdf` model.

```
robot name is: myworkcell
---------- Successfully Parsed XML ---------------
root Link: world has 1 child(ren)
    child(1):  table
        child(1):  base_link
            child(1):  base
                child(1):  tool0_controller
            child(2):  shoulder_link
                child(1):  upper_arm_link
                    child(1):  forearm_link
                        child(1):  wrist_1_link
                            child(1):  wrist_2_link
                                child(1):  wrist_3_link
                                    child(1):  ee_link
                                    child(2):  tool0
```

Fig. 5. The link tree of UR5 manipulator mounted on a flat surface 'table'.

B. Creating the MoveIt pacakge based on URDF

The MoveIt Setup Assistant (MSA) provides a graphical user interface for configuring a robot with MoveIt. It can automatically set up the task pipeline for producing an initial configuration quickly. Its main functions comprise generating a Semantic Robot Description Format (SRDF) file, creating the collision matrix of the robot and defining the planning groups. The robot model URDF (section IV-A) is a prerequisite for the MoveIt setup assistant. The configurations set by the assistant include a self-collision matrix, planning group definitions, robot poses, end effector semantics and virtual and passive joints list. The first step of the Setup Assistant (SA) is the generation of a self-collision matrix for the robot used in future planning to speed up collision checking. This collision matrix encodes pairs of links on a robot that we can safely discard from collision checking due to the kinematic infeasibility of a collision. We have modelled a flat table surface into the work cell. The collision matrix, calculated at the initial step, will now find which links are in collision accounting for the table. Hence, the generated plans at runtime will only be made in the reachable workspace, making the planning faster and correct. The user can provide information on different motion planning aspects in a step-by-step process. Virtual joints attach the robot to the world. Planning groups semantically describe parts of the manipulator, i.e., what constitutes a gripper or which joints and links comprise an arm. MoveIt plans for a group considering only the joints that belong to that group. MSA allows adding certain fixed poses of the manipulator. We can define query positions as the initial and goal configurations of the manipulator, and end-effectors can be labelled. In the last step, the MSA generates several configurations and launch files that can be used inside a ROS package.

C. Planning and algorithms integration

On the planning side, we use MoveIt to integrate state-of-the-art sample-based motion planning algorithms in our solution. Using the motion planning APIs of MoveIt, the `MoveGroupInterface`, we have implemented and tested simple motions of the robotic arm. In particular, we can specify either pose goals or joint-space goals. Pose goals define the end-effector position in 3-d cartesian coordinates, whereas a joint-space goal identifies a distinct final configuration for the joints (given by individual joint angles). For both cases, we can plan the movement of the robotic arm to the desired goal. These tests have been done within the graphical simulator `RViz` and on the UR5 robotic arm. The Kinematics solver and planner algorithm are the main components of a motion-planning solution. MoveIt has built-in support for this functionality; it uses plugins for computing kinematics and path planning, Open Motion Planning Library (OMPL) [18] for motion planning, and Kinematics and Dynamics Library (KDL) for forward and inverse kinematics. In recent tests, we have used `trac_ik` inverse kinematics solver [26] that performs better in terms of its speed and success rate of finding a solution (in a set of 10000 random tests, it was 10% more successful and solving time was about half than KDL, on average [27]). In the OMPL library, we have random sample-based planners such as Probabilistic RoadMaps (PRM), Rapidly Exploring Random Trees (RRT), RRTConnect etc. We have tested with PRM and RRTs in our work. The framework allows us to add collision objects (obstacles) to the workspace. Collision objects are geometric primitives such as a box or a cylinder that we can easily define through their key dimensions and 3d position. In this case, the planned trajectory will avoid the obstacle or report failure when it cannot reach the target configuration.

We organize different nodes into a launch file to run the system. In Listing 2, we see an excerpt of the main ROS launch file of the system.

```
<launch>
  <rosparam command="load" file="$(find
      liu_moveit_config)/config/joint_names.yaml"/>
  <arg name="sim" default="true" />
  <arg name="robot_ip" unless="$(arg sim)" />
  <include file="$(find liu_moveit_config)/launch/
      planning_context.launch" >
    <arg name="load_robot_description" value="true"
        />
  </include>
  <group if="$(arg sim)">
  <include file="$(find industrial_robot_simulator)
      /launch/robot_interface_simulator.launch" />
</group>
<group unless="$(arg sim)">
  <include file="$(find ur_modern_driver)/launch/
      ur5_bringup_compatible.launch" >
    <arg name="robot_ip" value="$(arg robot_ip)"/>
  </include>
</group>
<group if="$(arg sim)">
<node name="robot_state_publisher" pkg="
    robot_state_publisher" type="
    robot_state_publisher" />
</group>

...
...

</launch>
```

Listing 2. `moveit_planning_execution.launch` file to bring up `UR5`.

V. Evaluations

In this section, we report the results from system execution. In particular, we present an analysis of the nodes' communication graph, the performance of different planning algorithms and an experiment with path planning and obstacle avoidance.

A. ROS computation graph

The program can be tested in `RViz` as well as on the robotic arm by simply setting `sim:=true` or `false` in the following command which launches the main script from Listing 2.

```
roslaunch liu_moveit_config
moveit_planning_execution.launch
sim:=false  robot_ip:=192.168.0.103
```

Besides Listing 2, we run other nodes, i.e., for publishing goal positions, or the coordinate points of the bounding boxes for dynamic obstacles such as humans. Fig. 6 shows the ROS graph of our application. The developed motion planning node `CMotionPlanner` receives a target pose on topic `sp1_pose`, whereas, it receives coordinates for the human bounding box on topic `chatter`. The primary node provided by MoveIt is the `move_group` node which serves as an integrator pulling individual components together. It loads three kinds of information from ROS `param server`: URDF of the work cell in `robot_description` parameter, SRDF of the work cell in `robot_description_semantic` parameter, and different configurations created by MSA for kinematics, trajectory control. `ur_driver` node publishes the real-time joint states on `/joint_states` and `robot_state_publisher` converts the `/joint_states` messages to corresponding `/tf` messages. It looks at the urdf of the robot and listens on the `joint_state` messages. It then calculates where each link of the robot is and then broadcasts it to the rest of the system on `tf`, which is subscribed by `move_group` too. Thus, it handles the common task of computing forward kinematics. Finally, the `move_group` node talks to the controller on the robot using the `FollowJointTrajectoryAction` interface by publishing a goal point on `follow_joint_trajectory/goal`.

TABLE I. PERFORMANCE OF DIFFERENT ALGORITHMS.

		pose goal		joint-space goal	
	algorithm	states	time (s)	states	time (s)
no obstacle	PRM	1412	5.002649	1417	5.005281
	PRMstar	868	5.001812	847	5.004916
	RRT	12	0.039855	11	0.042702
	RRTstar	2159	5.002191	2227	5.009980
	RRTconnect	4	0.034107	7	0.035261
obstacle	PRM	1552	5.003105	1568	5.002542
	PRMstar	992	5.009589	972	5.013361
	RRT	10	0.042280	26	0.062595
	RRTstar	1856	5.024425	1792	5.001743
	RRTconnect	5	0.028506	6	0.034078

B. Planning with obstacle

In this experiment, we use the PRMstar algorithm. We consider a pose goal given by the configuration vector $\{-0.1304, 0.43455, 0.19730, 1.41, -2.46, -4.88\}$ where the first three elements indicate the position and the last three indicate the orientation of the end-effector. We test the motion of the `UR5` arm to this goal configuration in the presence and absence of an obstacle. The obstacle is a human bounding box with the following vertices' coordinates $(0.046, 0.95, 0.6)$, $(0.046, 0.48, 0.6)$, $(0.046, 0.95, -0.59)$, $(0.046, 0.48, -0.59)$, $(-0.03, 0.95, 0.6)$, $(-0.03, 0.48, 0.6)$, $(-0.03, 0.95, -0.59)$, $(-0.03, 0.48, -0.59)$. Fig. 7 shows the results for this test. Fig. 7 (a) depicts the case with no obstacle and start (yellow) and goal positions of the robotic arm. In Fig. 7 (b), we see the path trail that `UR5` follows to reach the goal position in the absence of an obstacle. Fig. 7 (c) and (d) show the collision object in the workspace and the path trail. Notice that the arm follows a different path to avoid the obstacle. For case (b), the algorithm creates 808 roadmap states and the solution is found in 4.998913s. For cases (c) and (d), the algorithm creates 835 roadmap states and the solution is found in 5.014565s.

C. Performance of planning algorithms

We measure the performance of some sample-based motion-planning algorithms, namely PRM, PRMstar, RRT, RRTstar, and RRTconnect. Similar to the last experiment, we consider two cases, i.e., with and without an obstacle. Table I lists the results. RTConnect performs the best with the minimum number of states in all the experiments. RRTConnect uses two RRTs, one rooted at the source and another rooted at the goal point, and seeks to grow both trees until they get connected, thus finding the path. This methodology leads to faster solution time.

VI. Conclusions and Future work

Application development for robots is a challenging task with several dimensions, i.e., motion planning, scene acquisition, and control. A modular approach can cope with such

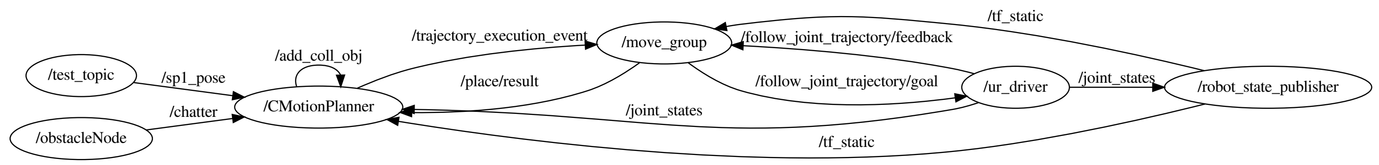


Fig. 6. `rqt_graph` view of the nodes and topics involved in actuating the model.

complexity. We showed how we could combine independently developed components namely perception and planning. In particular, we showed how we modelled the proposed solution architecture. ROS is a popular platform that offers open-source packages for different aspects of robot software development. We detailed the steps required to model a motion planning application within ROS and its motion planning framework MoveIt. Since the planner is not responsible for mapping free and occupied spaces, there is an efficiency benefit, especially for static scenarios. The paper focused on the motion planning aspect. We presented the ROS communication graph of the running system, validating the presence of all system components as indicated by respective nodes and their communication topics. For space limitation, we did not present additional capabilities of our solution, e.g., safety tracking and re-planning. Our results showed that the manipulator successfully adapts the trajectory in the presence of obstacles. Due to its unique strategy for exploring configuration space, RRTConnect performed better as a path planner in different test cases.

A complete work cell can be modelled in a urdf, containing the static objects. Thus, all the modelled objects are checked when generating a self-collision matrix. At run-time, the configuration space to be checked when planning trajectories is smaller, and hence the faster solution. This will be part of future work. We also plan to prepare a case study based on a collaborative assembly operation that involves all components of our proposed approach and demonstrates its efficacy.

Acknowledgments

The authors acknowledge the support of the ARISE Associated Laboratory (LA/P/0112/2020) and R&D Unit SYSTEC -Base (UIDB/00147/2020) and Programmatic (UIDP/00147/2020), and also the support of project RECLAIM – “RE-manufaCturing and Refurbishment LArge Industrial equipment”- GA no. 869884, funded by the European Commission under the H2020 and project PRODUTECH 4S&C - SUSTAINABLE & CIRCULAR PRODUTECH, with reference POCI-01-0247-FEDER-046102, co-funded by FEDER, through COMPETE 2020.

References

[1] W. Z. Khan, M. H. Rehman, H. M. Zangoti, M. K. Afzal, N. Armi, and K. Salah, “Industrial internet of things: Recent advances, enabling technologies and open challenges,” *Computers & Electrical Engineering*, vol. 81, p. 106522, 2020.

[2] Z. Sheng, C. Mahapatra, C. Zhu, and V. C. M. Leung, “Recent Advances in Industrial Wireless Sensor Networks Toward Efficient Management in IoT,” *IEEE Access*, vol. 3, pp. 622–637, 2015.

[3] X. Li, D. Li, J. Wan, A. V. Vasilakos, C.-F. Lai, and S. Wang, “A review of industrial wireless networks in the context of industry 4.0,” *Wireless networks*, vol. 23, pp. 23–41, 2017.

[4] B. McMorris, “A History Timeline of Industrial Robotics,” https://futura-automation.com/2019/05/15/a-history-timeline-of-industrial-robotics/, 2022, accessed: 2022-10-15.

[5] IFR, “IFR International Federation of Robotics - Robot History,” https://ifr.org/robot-history, 2021, accessed: 2022-10-10.

[6] J. A. Marvel, “Performance Metrics of Speed and Separation Monitoring in Shared Workspaces,” *Transactions on Automation Science and Engineering*, vol. 10, no. 2, pp. 405–414, 2013.

[7] P. Anderson-Sprecher, “Intelligent Monitoring of Assembly Operations,” Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-12-03, June 2011.

[8] J. Fryman and B. Matthias, “Safety of Industrial Robots: From Conventional to Collaborative Applications,” in *7th German Conference on Robotics (ROBOTIK)*. VDE, 2012, pp. 1–5.

[9] P. Tsarouchi, S. Makris, and G. Chryssolouris, “Human–robot interaction review and challenges on task planning and programming,” *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 8, pp. 916–931, 2016.

[10] P. A. Lasota, G. F. Rossano, and J. A. Shah, “Toward Safe Close-Proximity Human-Robot Interaction with Standard Industrial Robots,” in *10th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2014, pp. 339–344.

[11] P. Rybski, P. Anderson-Sprecher, D. Huber, C. Niessl, and R. Simmons, “Sensor Fusion for Human Safety in Industrial Workcells,” in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2012, pp. 3612–3619.

[12] M. Bdiwi, M. Pfeifer, and A. Sterzing, “A new strategy for ensuring human safety during various levels of interaction with industrial robots,” *CIRP Annals*, vol. 66, no. 1, pp. 453–456, 2017.

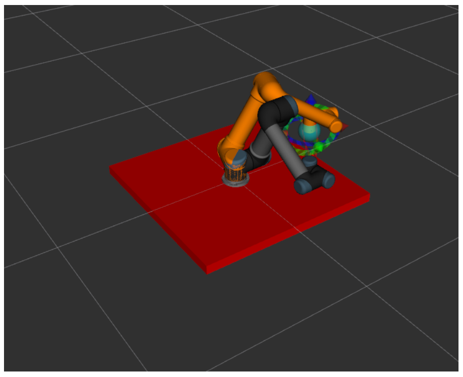
[13] U. Robots, “UR5 collaborative robotic arm,” https://www.universal-robots.com/products/ur5-robot/, 2015, accessed: 2019-12-20.

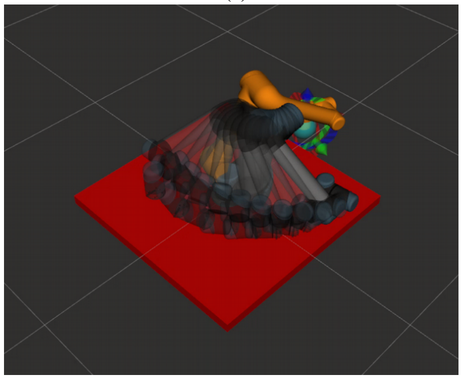
[14] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

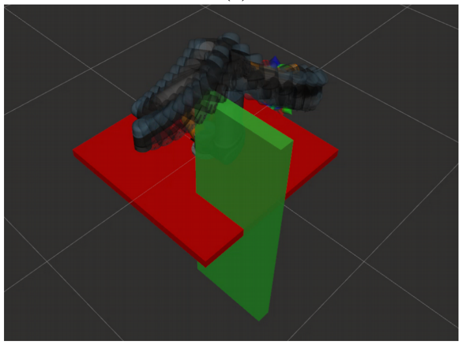
[15] S. Chitta, I. Sucan, and S. Cousins, “Moveit![ros topics],” *IEEE Robotics & Automation Magazine*, vol. 19, no. 1, pp. 18–19, 2012.

[16] A. Roncone, O. Mangin, and B. Scassellati, “Transparent Role Assignment and Task Allocation in Human Robot Collaboration,” in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 1014–1021.

[17] Y. Pyo, H. Cho, R. Jung, and T. Lim, *ROS Robot Programming*. Robotis Co., Ltd., 2017, p. 50.


(a)


(b)


(c)

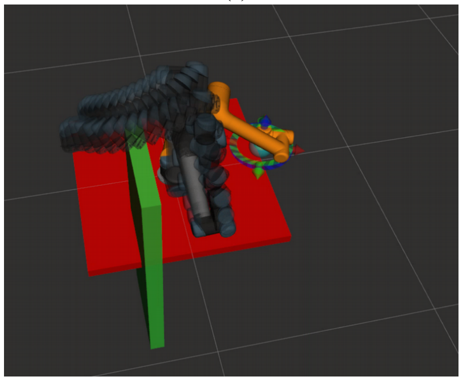

(d)

Fig. 7. Motion planning of UR5 manipulator.

[18] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[19] J. Pan, S. Chitta, and D. Manocha, "FCL: A General Purpose Library for Collision and Proximity Queries," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2012, pp. 3859–3866.

[20] D. Thornton Coleman IV, "Methods for Improving Motion Planning Using Experience," Ph.D. dissertation, University of Colorado, 2017.

[21] M. Kallman and M. Mataric, "Motion Planning Using Dynamic Roadmaps," in *International Conference on Robotics and Automation (ICRA)*, vol. 5. IEEE, 2004, pp. 4399–4404.

[22] L. Antão, J. Reis, and G. Gonçalves, "Voxel-based Space Monitoring in Human-Robot Collaboration Environments," in *24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, pp. 552–559.

[23] R. Nogueira, J. Reis, R. Pinto, and G. Gonçalves, "Self-adaptive Cobots in Cyber-Physical Production Systems," in *24th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, pp. 521–528.

[24] I. GitHub, "Universal Robot," https://github.com/ros-industrial/universal_robot, 2019.

[25] T. T. Andersen, "Optimizing the Universal Robots ROS driver." https://orbit.dtu.dk/en/publications/optimizing-the-universal-robots-ros-driver, Technical University of Denmark, Tech. Rep., 2015.

[26] P. Beeson and B. Ames, "TRAC-IK: An Open-Source Library for Improved Solving of Generic Inverse Kinematics," in *15th International Conference on Humanoid Robots (Humanoids)*. IEEE-RAS, 2015, pp. 928–935.

[27] P. Beeson and B. Ames, "trac_ik," https://bitbucket.org/traclabs/trac_ik/src/master/, 2019, accessed: 2020-01-25.