Roadmap-based Planning in Human-Robot Collaboration Environments

Zahid Iqbal, João Reis, Gil Gonçalves

SYSTEC, Research Center for Systems and Technologies
Faculty of Engineering, University of Porto
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
Email: {zahid,jpcreis,gil} @fe.up.pt

***Abstract*—Enabling humans and robots to work together allows for cost savings and workplace efficiency. The robot must be equipped to perceive humans, and redirect its actions for co-operative tasks or under hazardous situations. Thus, dynamic motion planning appears as an essential exercise. We use dynamic roadmaps for online motion planning in changing environments combined with a voxel-based grid. The presented approach can answer path planning queries efficiently. Visualized simulation is an important technique for rapid verification of algorithms or prototypes. We present an architecture that implements a simulation of the robotic manipulator using the Robot Operating System (ROS) and MoveIt.**

Keywords–robotics technology, intelligent robotics, sampling-based planning, perception and sensing, dynamic environments, voxel-based grid.

I. INTRODUCTION

Today, there are opportunities for automation in many industries, such as automotive assembly lines, material transport, space exploration. Frequently, robots support this automation, improving thus the production volumes, bringing down costs, and improving the precision and accuracy of the production process. Additionally, as robots capabilities increase, they can take on jobs that might be impossible or dangerous for humans [1]. The ability to calculate and execute motion plans is central to the operation of these robots. Fundamental motion planning problem is to compute a set of inputs to the robot that drive it from a start to a goal position while avoiding collisions with the environment obstacles. Early works on robot motion planning assumed a robot's world to be known and fixed at the time of planning. During the execution of the plan, however, the robot might discover a deviation from the assumed world state, such as a human entering its operational area. Typical industrial assembly environments, that confined robots to separate operation spaces isolated from human workers, would bring robot operation to a halt under such scenario [2], safety being the primary concern. In other cases, the robot must replan its trajectory from scratch, which might be computationally intensive for real-time operation.

Random sampling is an important technique employed by popular planning algorithms such as Probabilistic Roadmaps (PRM) [3], Rapidly Exploring Random Trees (RRT) [4] to build network of feasible paths. For static environments, these methods can efficiently make and execute plans, even when there are obstacles. With some of these algorithms, such as PRM that are *multi-query*, we can delegate roadmap building as an offline exercise and run multiple queries online on the available roadmap. However, we can realize a better potential of the robots by operating them in unstructured open environments that change dynamically. A robot would only possess partial information of surroundings before it commences operation; obstacles could appear or go away at any time during its operation. This situation poses several challenges for motion planning [5]. With efficient planning, we can leverage the benefits of cooperation by assigning specific production tasks to robot and humans, as well as make the robot more autonomous. Specifically, it requires the ability to account for new obstacles or changes in the environment and continue to plan on the fly and quickly with some accuracy.

This work is an incremental exercise concerning building a collabarative planning solution. We define collaboration as working as a team to reach a common goal. In the collaborative environment, physical contact between human and robot may be inevitable and indeed desirable, enabling the robot to learn from experience. To achieve accurate and efficient collaboration, both robot and human need to perceive each other's intentions and must know their task set. With that knowledge, the robot can plan and adapt its actions accordingly, ultimately leading to the achievement of the mutual objective, all the while assuring safety. A typical scenario could be the robot as a work assistant that can hand in the next tool piece to the human. Robot predicts that this piece is the one needed for the person to continue its job and is shelved and outside human's access. Collaboration scenario primarily comprises a dynamic environment, and any motion planning strategy that tackles the dynamic obstacles must allow dealing with the collaboration episodes accordingly. To this end, different approaches exist, such as Incremental Path Planning [6] and Experienced based Motion Planning (EBMP) [7]. The later employs a database of previous searches and recalls an experience from the database when it encounters a similar planning exercise. This may involve *repairing* the previous plan and reduce the time of the current query as opposed to planning from scratch. Likewise, incremental search methods use previous search results to solve similar path planning problems faster.

Our work concerns path planning with obstacle avoidance in dynamic collaboration scenarios. For efficient collaboration, an intelligent monitoring system is inevitable [8, 9]. The present work uses the monitoring solution in [8] and combines with a PRM based approach, dynamic roadmaps. We organize the work presented here in two distinct parts. The first part explains dynamic motion planning for robots and reviews some strategies that address dynamic environments (Section II). In the second part (Section IV), we present the tools that allow developing motion planning algorithms on a simulated model

 ISBN: 978-1-61208-798-6

of the robot. In Section III, we describe how we integrate the monitoring solution with planning algorithm for efficient motion planning. In Section V, we present the preliminary path planning architecture using the popular tool support and based on a voxelized grid. Finally, in Section VI, we conclude the paper and present some future work directions.

II. MOTION PLANNING IN DYNAMIC AND UNKNOWN ENVIRONEMNTS

The work presented here concerns industrial robotic arms or so-called manipulators. In particular, we do not consider mobile robots; proposed ideas will be applicable to trajectory planning of a Universal Robot with a lift ability of 5 kg (UR5) [10]. Mounted on a stationary platform, its links with revolute joints and end-effector can move with a certain degree of freedom.

A. Definitions

Fundamental to the path planning problem is the concept of *configuration* and *configuration space*. The configuration of a robot is a set of independent parameters that characterize the position of every point in the object whereas configuration space (denoted $\mathcal{C}$) is the space of all possible configurations [11]. The dimension of the configuration space is determined by the degrees of freedom of the robot. A configuration q is a vector of robot joint values. For instance, a robotic arm with six revolute joints has six degrees of freedom and its configuration is denoted $q = \{q_1, q_2, q_3, q_4, q_5, q_6\}$ where q_i denotes i^{th} joint angle. The configuration space $\mathcal{C}$ is a space of these configurations, i.e., for 6 dimensions, we have $\mathcal{C} = R^6$. Any path finding strategy places configurations in $\mathcal{C}$ into two categories, those that are free $\mathcal{C}_{free}$, and others that are in collision $\mathcal{C}_{forb}$, i.e., occupied by obstacles. A configuration $q \in \mathcal{C}_{free}$ if the robot placed at q does not intersect with workspace obstacles. A *path* is a continuous sequence of configurations in $\mathcal{C}_{free}$ connecting initial and goal configurations. For dynamic scenarios, a notion of time is incorporated in $\mathcal{C}$, and resulting space may be termed as *state-time* space [12]. It consists of pair (x, t) where $x \in \mathcal{C}$ and t denotes the time. A path obeying dynamic constraints is termed as *trajectory*.

B. PRM based approaches for dynamic environments

PRM approach, in the planning phase, randomly samples configurations in the free space and builds a roadmap, i.e., $\mathcal{C}_{free}$. In the query phase, it finds paths in the roadmap to guide robot from start to the goal. Dynamic obstacles in the environment can invalidate parts of the previously constructed roadmap in what concerns free space. Therefore, we need methods to update the roadmap taking into consideration the current state.

The works in [13] and [12] use PRM-based approach to handle dynamic scenarios. The work in [12] extends PRM by augmenting configuration space with a dimension of time. The preprocessing step is like regular PRM. Online, checking for static obstacles is not required for collision testing. It incorporates the notion of *free interval*, the maximum continuous segment of time that a configuration (a roadmap node) is collision-free. Arriving earliest in a free interval allows reaching the goal faster. Implicitly, this approach builds a free interval graph that corresponds to the roadmap built in the preprocessing phase, i.e., each vertex in free interval graph maps to one node in the roadmap. The approach builds a local trajectory first, i.e. traversing a single arc between two roadmap nodes, and then extends to the complete interval graph. Robot movement velocity is given maximum v_{max}. The approach interpolates motion along a single arc using a two-dimensional state-time grid where distance travelled along the arc and time are two dimensions. The state-time space is discretized by time step τ and velocity value $v_p < v_{max}$; l is the length of the arc. From a given state (x, t), three states are reachable (Figure 1).

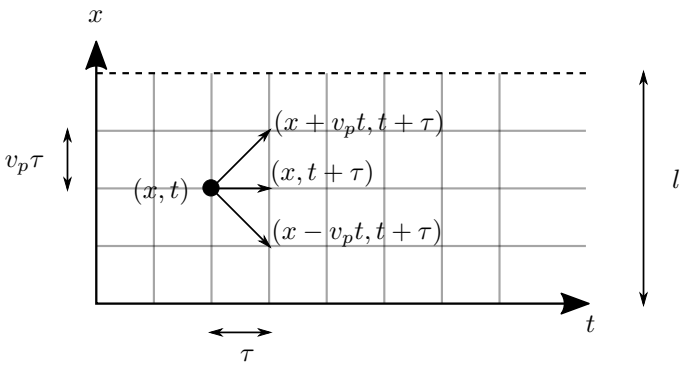


Figure 1. A state-time grid of a single roadmap arc [12].

For an arc a with start state-time (x_s, t_s) and destination state x_d, the approach maintains a stack where initially (x_s, t_s) is present. In a loop, it pops a stack element. If it is collision-free and not visited before, it pushes the reachable grid points in an order that favours movement towards the goal. The algorithm runs until the stack is empty or goal state x_d has been reached. Backpointers and the visit status of each state are maintained. So if a certain movement encounters an obstacle, we move back to the next unexplored state on the stack. The algorithm chooses a short time step τ for better accuracy of the algorithm. For finding a global trajectory, the algorithm sends $probes$ corresponding to single arcs and evaluates a function, $f(p) = g(p) + h(p)$, to determine how promising a given probe p is. $g(p)$ is the cost so far and $h(p)$ is the estimate until the goal state. Assuming (x_u, t_u) to be the top state on the stack for probe p, the trivial estimate for $g(p)$ is t_u. To estimate $h(p)$, it uses a roadmap distance function $D(x_u.g)$ from the current state x_u until the goal state g. This estimate could come from running a simple Dijkstra's algorithm on the roadmap before the query phase, giving $h(p) = D(x_u, g)/v_{max}$. The probe with minimum $f(p)$ value has the highest priority. For a start state s, the algorithm initializes probes on all arcs starting from s and puts them on the priority queue, from which it examines probes one by one. If the local trajectory found by a prob reaches the goal, we have a solution. Otherwise, it starts probes on outgoing edges and continues. The algorithm terminates either when it finds the goal node or if no probes are remaining.

The work in [13] start their approach alike the [12], i.e., building the roadmap in the preprocessing phase, accounting for static obstacles. In the query phase, at first, the approach connects query nodes to the roadmap. Next, it divides roadmap nodes into three categories, those that are reachable from the start node, those that are reachable from goal node, and those that are not reachable from query nodes. A first attempt to

find path labels some edges as blocked; edges that are not collision-free due to dynamic obstacles. An essential element of this approach is the connectivity extension in two ways, from a query node to a node reachable from the other query node, and from query nodes to the roadmap portion which is not yet connected to query nodes. The approach searches for a path with enhanced connectivity. When the approach cannot retrieve path from the roadmap, it uses RRT-connect [14] which incrementally builds two random trees one rooted at the start node and the other rooted at the goal node, and tries to find a path. Finally, it can create new nodes if the existing roadmap does not allow any path. This method can store the key positions of the moving obstacle in a database. For subsequent edge labelling, it compares the current position of an obstacle to the stored value. If it's the same, the previous exercise (collision testing, and pathfinding) is still valid, thus, avoiding new effort. Since dynamic obstacles can move at any time different positions, so, practically, there can be infinite positions that must be stored. For efficient memory usage, the approach stores only the last checked positions. However, this approach is beneficial in certain situations; for example, when moving obstacles are doors that could be closed or open, then storing a few positions might save significant future recalculations.

C. Experienced based motion planning

Experience-based approach to motion planning builds on knowledge from past planning exercises, i.e., [7]. It maintains a database of past experiences from which it can retrieve a partial or complete solution to the current problem. Such an approach may be particularly beneficial in situations where a robot is supposed to be operational in a fixed environment for long periods. In such a situation, the robot would frequently encounter similar obstacles and environments. Thus, planning from scratch is not desirable because of redundant computation effort. Repairing an available solution, instead, is faster. The approach maintains a sparse roadmap for efficient memory usage. Further, for speed up, it delays the collision checking until after it has found a candidate path. Later, it tests the path for collisions and removes any invalid segments. When the path gets broken, the search is commenced. In a post-processing step, it smoothes repaired paths before sending these to the robot. The approach discretizes the solution path into an ordered set of vertices and incorporates it into the experience roadmap. In the query phase, it follows the general approach, connecting query nodes to the graph, if this is successful, then A* algorithm can return the optimal path between start and goal nodes. As stated, it checks for possible collisions with environmental obstacles after it has found the candidate path. For this case, invariant constraints do not lend any recomputation. Moving obstacles, however, can appear and invalidate path segments. For this case, a new A* search is run to find the path. The approach to finding a path through disconnected components is similar to that of [13].

D. Incremental path planning

Incremental search methods reuse recent solutions to speed up searching for similar problems. Assuming that we have incomplete information about the world, replanning occurs from the current state until the goal whenever an obstacle encounters. When we have to execute the next action in the plan, we receive changes, if any, in the graph. This information could come from different sensors outside of the robot, such as laser scanners or RGB cameras. Using this information, we update the previous map instead of replanning from scratch. We continue to perform this loop until we reach the goal (Figure 2).

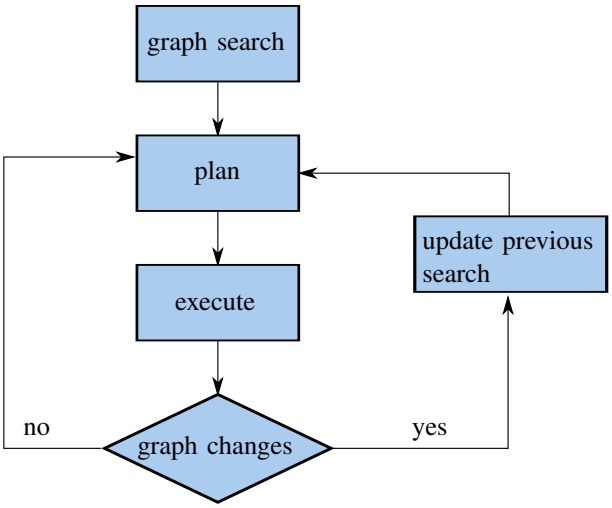


Figure 2. General methodology of incremental path planning.

In the following, we briefly discuss an incremental planning algorithm D* [15].

1) D algorithm:* D* is an incremental search algorithm, considered to be the dynamic version of A* algorithm [16] since it permits edge costs to change as the algorithm runs. The algorithm begins at the goal state donated G and works its way back to the start state; on its way, it keeps on reiterating and updating costs to reflect optimal paths. Each graph node can be in one of the following states NEW, CLOSED, OPEN, RAISE or LOWER. The OPEN list maintains nodes needing expansion. The algorithm runs until the current robot state is CLOSED. The algorithm maintains two cost estimates for each node, namely path cost function $h(G, X)$ and key function $k(G, X)$. Path cost function gives the current estimate of the sum of arc costs from state X to goal state G, and key function denotes the minimum of $h(G, X)$ which we can regard as the previous estimate of $h(G, X)$. The algorithm sorts the OPEN list according to nodes key values. When expanding a node on the OPEN list, the key function allows changing the current node state as RAISE or LOWER. When $k(G, X) < h(G, X)$, current estimate indicates an increased cost, so X state is changed as RAISE, otherwise as LOWER. Next, it propagates changes from expansion to the neighbours that are now placed on the OPEN list and expanded in turn. Each node has a back pointer which leads to the target. Algorithm reports a solution path when the next node for expansion is the start node. We follow back pointers from the start until the goal.

There have been different improvements to the original D* such as D* Lite [17], and Focussed D* [18]. Focussed D* is an informed incremental heuristic search algorithm that combines ideas from A* and D*. D* Lite, though not directly based on D*, implements similar ideas to path planning in unknown environments as D*. With lower implementation overhead, it yields better performance.

III. Proposed approach

In this section, we describe the approach that makes use of the voxel-based grid for motion planning. We envisage using a roadmap based approach for its simplicity and ease of development. This is an extension of previous work [19], now addressing the dynamic environment.

A. Dynamic roadmap with voxel-based occupancy grid

A voxel represents a value on a regular grid in three-dimensional space, useful for many applications such as Dynamic Roadmaps [20] to create a mapping from an area in the workspace to states in $\mathcal{C}$. The work in [8] presents a monitoring approach that outputs a voxel grid. 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 resolution of the grid presents a tradeoff between the accuracy of the planning queries and computation effort. The monitoring solution can also label the workspace objects. When voxels in the grid are labelled, we can identify, at first, if an object in the collaborative environment occupies the voxel, and secondly, which type of object, i.e., either a robotic part itself, human, or an obstacle object. A labelled voxel-based grid can efficiently solve planning queries, in particular, in the presence of static obstacles. In the query phase, before the search commences, occupied voxels would indicate $\mathcal{C}_{forb}$ and the remaining nodes would make the $\mathcal{C}_{free}$.

In our system, we need an accurate mapping of workspace objects from the voxel grid to the coordinates that planning algorithms can work with. Monitoring solution considers the origin $(0, 0, 0)$ to be at the base-center of the robotic arm. For the tools used in this work (section IV), every point on the robotic arm, and in the workspace can be calculated in reference to the planning-frame, which is the frame of the "base-link" of the arm, and its origin is at position $(0, 0, 0)$. When we have a common reference point, and with some additional information from the voxel grid such as dimensions of individual voxels as well as of the hyper-cube that represents the complete voxel grid, we can map the goal positions and obstacle positions from the voxel grid into the planning solution, which allows solving subsequent path-planning queries efficiently. Figure 3 shows an example voxel grid with extreme-points and dimensions identified.

However, for dynamic obstacles, we incorporate further information, such as input from prediction algorithms dictating the temporal occupancy status of voxels. Accounting for dynamic scenarios, here we stipulate two possible ways to handle it. We can have an additional dimension of time to the voxel-based grid. And, we can associate a probability of occupancy pr_{occ} for any voxel c at any given time t, i.e., $pr_{occ}(c, t) \in [0, 1]$. This input, might come from the perception sensor, and aid in redirecting paths of the manipulator in the presence of dynamic obstacles.

Secondly, we employ ideas from the reviewed approaches, in particular, those based on random roadmaps in combination with voxel-based grid for dynamic path planning of robotic manipulator. Consider the roadmap R and the voxel-based grid G. The nodes in R are configurations q_i in C_{free}, and cells of grid G are denoted c. For each c in G, we check configurations q_i representing a position of the end-effector and that fall in c. With a large number of cells in G having many such configurations, we can stop computing the roadmap R, lending a uniform distribution of end-effector positions over the workspace. An occupied c invalidates the respective q in R. Obstacles appearing, disappearing, or changing their positions represent dynamic scenarios. The grid G can inform us when a cell gets occupied or becomes free. The scene can update based on actions of an external agent, e.g., a human that shares the workspace with the robot. Whenever one or more objects occupy a cell, a reference counter keeps increasing in respective node or edge in R, denoting invalid nodes. The approach keeps renewing the state of some nodes or edges as these become occupied or free. An RRT algorithm is used when the path gets broken, and the algorithm cannot compute the query.

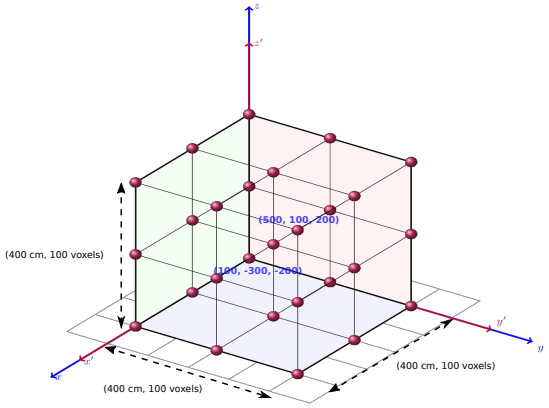


Figure 3. An example voxel grid.

IV. Tool Support for developing motion planning algorithms

In this section, we briefly describe the tools that we have used to develop motion-planning solution.

A. Robot Operating System (ROS)

ROS [21] provides a framework to develop and test robot software as well as to deploy such software to the real robots. As the scope of the robots has significantly grown, a fully functional code must contain software for hardware drivers, perception algorithms, abstract reasoning, trajectory planning, control and more. For any single user, it might not be possible to cover all aspects of software development for a robot. For the same reason, the final software architecture would incur a significant integration effort. The main objective of ROS is to manage the complexity of software development for robots. It contains helpful tools and open software packages for perception, navigation, transforms and simulations, allowing rapid prototyping of software for experiments. Emulating the motion of robots and the world in a virtual workspace without physically depending on an actual robot saves cost and time. ROS supports development in different programming languages, mainly C++ and Python.

 ISBN: 978-1-61208-798-6

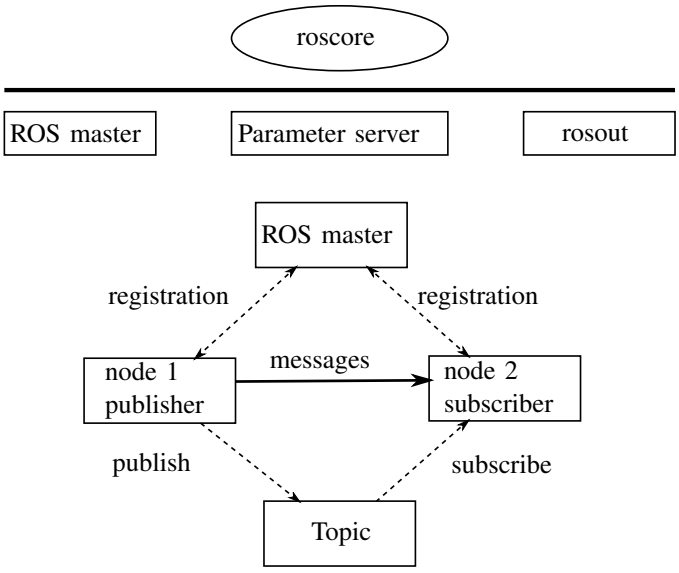


Figure 4. ROS core communication system and an example of nodes publish & subscribe to topics.

Conceptually, the computation graph within ROS is the peer-to-peer network of ROS processes that are processing data together. *roscore* is the core of the system providing a collection of nodes and programs. roscore contains three main functional module suits, which are *ROS Master*, *Parameter Server* and *rosout* logging node. ROS Master provides naming and registration to nodes in the ROS system; it allows nodes to find each other. A *node* is an executable unit of code that implements a specific functionality, similar to a software module in traditional usage. A typical system within ROS comprises many nodes and is a graph where links represent nodes communication. ROS has a publish-subscribe model of communications. Nodes communicate with each other using *topics* as communication channels. Nodes publish to a topic to send *messages* and subscribe to it for receiving messages. There can be multiple publishers and subscribers of a given topic (Figure 4). The basic message types include boolean, float, integer or string; users may use custom types as well.

B. URDF

Unified Robot Description Format (URDF) is an XML format for describing a robot model in ROS. Different ROS programs can interpret the URDF model of physical robot, e.g., tools such as *rviz* that allows to visualize it. In the most basic form, URDF describes the topology of the robot, listing its joints and links, where a joint connects two links. Topology makes a tree structure, where each link has precisely one parent, but it can have multiple child nodes. The geometry of the robot is represented through coordinate frames and transforms between these frames. Transforms have a translation and rotation components; translation measured in meters is specified in coordinate axes x, y and z whereas rotations in radians along x, y and z-axis are known as roll, pitch and yaw respectively. For ROS system to know about robot model, a launch script loads URDF file to the parameter server. The same script publishes joint state values. The kinematics libraries within ROS can read joint states, and carry out respective forward and inverse kinematics analysis to produce the relative transforms. The model also allows specifying visuals for robot parts using notations as a cylinder, sphere or box, and colour values with material tags. More complex information, such as collision information, inertia, joint and velocity limits, can also be given here. Specific utilities, e.g., *urdf_parser* can check for the validity of the URDF file.

C. MoveIt

MoveIt [22, 23] is a set of software packages with specific capabilities for mobile manipulation, such as kinematics, motion planning and control, 3D perception and navigation. *MoveIt* is integrated with standard ROS. It uses plugins for most of its functionality; motion planning (Open Motion Planning Library (OMPL) [24]), collision detection (default: Fast Collision Library (FCL) [25], and kinematics (default: OROCOS Kinematics and Dynamics Library (KDL) for forward and inverse kinematics for generic arms.

1) MoveIt setup assistant (SA): has the objective to reduce the entry barrier for developing robot software. It can automatically set up the task pipeline for producing an initial configuration quickly. Robot model URDF (section IV-B) is a prerequisite for MoveIt setup assistant. The configurations set by the assistant include self-collision matrix, planning group definitions, robot poses, end effector semantics, virtual joints list, and passive joints list. The first step of the SA is the generation of a self-collision matrix for the robot that can be used in future planning to speed up collision checking. This collision matrix encodes pairs of links on a robot that can safely be discarded from collision checking due to the kinematic infeasibility of there actually being a collision. During the step-by-step process user can provide information on different motion planning aspects. 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. SA allows adding certain fixed poses of the manipulator. We can define query positions such as the initial and the goal configurations of the manipulator, and end-effectors can be labelled. In the last step, SA generates several configurations and launch files that can be used inside a ROS package.

V. Proposed solution architecture

In this section, we present the general architecture for developing a dynamic motion planning approach using the tools described in the last section. For a simulation of the robotic manipulator, we identify three key functional modules in the architecture, perception, modelling and planning (Figure 5).

The central component *Motion planning framework* is developed as a ROS node and integrates different modules into the final motion-planning solution. Perception of the world is available through the voxel-based grid (discussed in section III-A). This grid is formed using data from vision sensor ZED 2K stereo vision camera [8].

The model of the manipulator, a prerquisite for the MoveIt setup assistant is given as a URDF file; central node looks for the *robot_description* parameter on the ROS param server to get the URDF. We used the model available in the UR5 repositories [26] that we have installed and built. An initial set of configurations generated by MoveIt setup assistant defining query poses, self-collision matrix, planning groups, is also

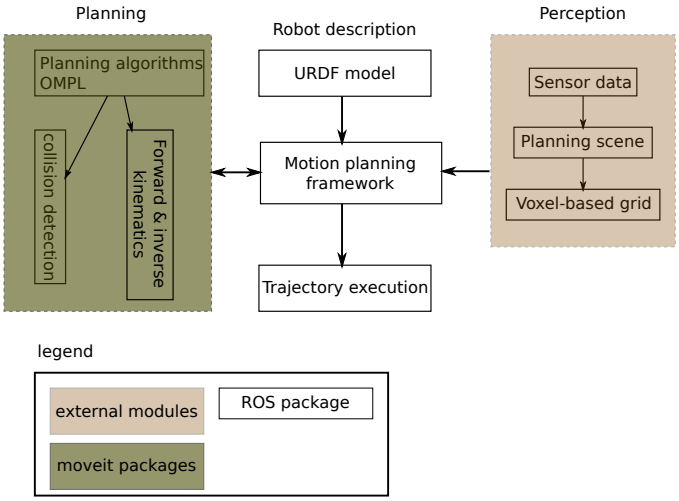


Figure 5. Robot manipulator motion planning - simulation architecture and module relationship.

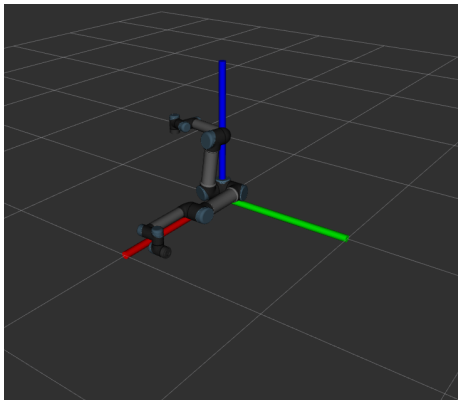
available on the param server, that Motion planning framework can look.

Concerning motion planning, MoveIt has a plugin based on the OMPL library that primarily implements randomized motion planners, such as PRM (Probabilistic RoadMaps), RRT (Rapidly Exploring Random Trees), RRTConnect etc. The central node offers an interface to the motion planner through ROS actions or services (a communication paradigm in ROS). Collision checking in MoveIt happens inside the planning scene. In particular, we can specify either pose goals or joint-space goals. Pose goals define a position of the end-effector 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 graphical simulator Rviz as well as on the UR5 robotic arm. The framework allows us to add collision objects (obstacles) to the workspace. Collision objects are geometric primitives such as a box or a cylinder, and can be easily specified through their key dimensions and 3d position. In this case, the planned trajectory will avoid the obstacle or fail to find a solution when the target configuration cannot be achieved in the presence of the obstacle.

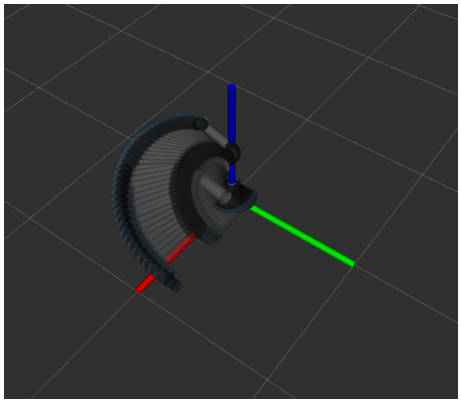
A. *Preliminary test*

We perform a simple test with regards to chosen tools. In this experiment, we use PRMstar [27] algorithm. We consider a joint-space goal given by the following configuration vector $\{-1.83, -1.732, 1.8, -1.634, -1.57, 2.88\}$ where joint angles are listed in radians. We test the motion of UR5 arm to this goal configuration in the absence of a collision object. Figure 6 shows the results for this test. For reference, see Figure 6 (a) that considers the case with no obstacle object and start and goal positions of the robotic arm. Start position is the manipulator lying in the x-y plane. In Figure 6 (b) we see the path trail that UR5 follows to reach the goal position. For the cases (a) and (b), the algorithm creates 785 roadmap states and solution is found in 5.009850s.

It is important to note that monitoring solution that outputs



(a)



(b)

Figure 6. Motion planning of UR5 manipulator.

the voxel-grid is a complementary component of intelligent motion planning. Its main objective is to provide online information on the workspace occupancy by different objects in the scene. Path planning component availabe via MoveIt is a standalone component. The reported experiment, validates the later.

VI. Conclusion

Robot path planning is a classic problem and is complicated in human-robot collaborative environments that present dynamic scenarios for motion planning. In this paper, we have reviewed some motion planning approaches that address such environments, i.e., where complete information is not available at planning time. We combine a voxel-based grid with a roadmap-based approach, for its simplicity and ease of development. Random sampling might result in roadmaps with disconnected components, and thus, it fails to find a path when start and goal configurations lie in disconnected components. For such cases, an online planner such as RRT would be helpful as it incrementally builds the complete path to the target. The software development for robots requires a breadth of knowledge and steep learning curve. For this reason, tools such as ROS and MoveIt with available functional packages can improve development time. We briefly described some essential concepts from these tools. And, presented a simulation architecture based on these tools.

The temporal validity of free spaces is not apparent in the

 ISBN: 978-1-61208-798-6

reviewed approaches. The cell marked as occupied may get free before the execution, but the algorithm has rendered it as occupied. It can be marked free only in the following planning episode. In this case, the planner might find an alternative path without the high cost. The contrary is more costly when space gets occupied between planning episodes, and the algorithm has not yet marked it occupied; leading to a failed query. Our future work will investigate this scenario. To efficiently handle dynamic scenarios, 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 previous planning query. Currently, we are looking into solutions to address these issues.

ACKNOWLEDGMENTS

INDTECH 4.0 – New technologies for intelligent manufacturing. Support on behalf of IS for Technological Research and Development (SI à Investigação e Desenvolvimento Tecnológico). POCI-01-0247-FEDER-026653

REFERENCES

[1] R. I. Association, "How Robots Are Taking on the Dirty, Dangerous, and Dull Job," https://www.robotics.org/blog-article.cfm/How-Robots-Are-Taking-on-the-Dirty-Dangerous-and-Dull-Jobs/209, accessed: 2020-05-19.

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

[3] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," IEEE Transactions on Robotics and Automation, vol. 12, no. 4, August 1996, pp. 566–580.

[4] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Department of Computer Science, Iowa State University, Tech. Rep. TR 98-11, October 1998.

[5] D. Katz, J. Kenney, and O. Brock, "How Can Robots Succeed in Unstructured Environments," in Workshop on Robot Manipulation: Intelligence in Human Environments at 4th Robotics: Science and Systems Conference (RSS 2008). Citeseer, June 2008.

[6] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong Planning A*," Artificial Intelligence, vol. 155, no. 1-2, 2004, pp. 93–146.

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

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

[9] R. Nogueira, J. Reis, R. Pinto, and G. Gonçalves, "Self-adaptive cobots in cyber-physical production systems," in 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2019, pp. 521–528.

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

[11] T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach," in Autonomous Robot Vehicles. Springer-Verlag, 1990, pp. 259–271.

[12] J. P. Van Den Berg and M. H. Overmars, "Roadmap-based Motion Planning in Dynamic Environments," IEEE Transactions on Robotics, vol. 21, no. 5, 2005, pp. 885–897.

[13] L. Jaillet and T. Siméon, "A PRM-based Motion Planner for Dynamically Changing Environments," in 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566), vol. 2. IEEE, 2004, pp. 1606–1611.

[14] J. J. Kuffner Jr and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-Query Path Planning," in Proceedings of the 17th IEEE International Conference on Robotics and Automation (ICRA 2000), vol. 2. IEEE, April 2000, pp. 995–1001.

[15] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in Intelligent Unmanned Ground Vehicles. Springer, 1997, pp. 203–220.

[16] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, vol. 4, no. 2, July 1968, pp. 100–107.

[17] S. Koenig and M. Likhachev, "Fast Replanning for Navigation in Unknown Terrain," IEEE Transactions on Robotics, vol. 21, no. 3, 2005, pp. 354–363.

[18] A. Stentz et al., "The Focussed Dˆ* Algorithm for Real-Time Replanning," in IJCAI, vol. 95, 1995, pp. 1652–1659.

[19] Z. Iqbal, J. Reis, and G. Gonçalves, "Path planning for an industrial robotic arm," in The Eighth International Conference on Intelligent Systems and Applications (INTELLI 2019). IARIA, 2019, pp. 30–36.

[20] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004, vol. 5. IEEE, 2004, pp. 4399–4404.

[21] 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.

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

[23] S. Chitta and I. Sucan, "Moveit," https://moveit.ros.org/, 2012, accessed: 2019-10-31.

[24] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," IEEE Robotics & Automation Magazine, vol. 19, no. 4, 2012, pp. 72–82.

[25] J. Pan, S. Chitta, and D. Manocha, "Fcl: A general purpose library for collision and proximity queries," in 2012 IEEE International Conference on Robotics and Automation. IEEE, 2012, pp. 3859–3866.

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

[27] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The international journal of robotics research, vol. 30, no. 7, 2011, pp. 846–894.

 ISBN: 978-1-61208-798-6