Path Planning for an Industrial Robotic Arm

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 in modern industrial environments can increase production volumes and reduce costs. However, a robot must be equipped to perceive humans and redirect its actions under hazardous events or for cooperative tasks. Thus, dealing with dynamic obstacles appears as an essential exercise. This work presents a motion planning algorithm for robots based on Probabilistic Road Maps (PRM). For efficient nearest neighbour search, we use kd-trees in learning and query phases of the algorithm. We construct the roadmap as an undirected graph in the free space. We implement the method for a simple configuration space in $\mathbb{R}^2$ and a point robot is considered to navigate between given initial and goal configurations added to the roadmap, all specified in two dimensions. We use Euclidean distance when finding the closest neighbours. The shortest path between the start and goal configurations is found using Dijkstra's algorithm. The method is easy to implement. After the learning phase, the method can answer multiple queries. We propose to use this method in combination with a labelled voxel-based grid for solving multiple path planning queries efficiently.**

Keywords–sampling-based planning, configuration space, grid-based search, kd trees, PRMs

I. INTRODUCTION

Today, standard industrial practices use robots for improving production volumes, bringing down production costs, and for better 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 [2]. To ensure safety, typical assembly environments within industries confine robots to separate operation spaces isolated from human workers. Such a setting, on the one hand, incurs significant space and installation costs, and on the other, loses cooperation opportunity with humans. Such systems frequently bring robot operation to a halt if a human entered its operational area [3].

However, with technical progress and to realize their full potential, it became common to put robots in more open unstructured environments. Such environments can leverage the benefits of cooperation by assigning specific production tasks to robot and humans. Moving from a highly structured to an unstructured environment poses several challenges for motion planning, an important one being that robot can only possess a partial knowledge of its surroundings [4]. Most planners assume that manipulator operates in a static environment. However, for many situations, such as the collaborative environment where humans work in proximity to the manipulator, the robot must account for dynamic environments. For such environments, we need methods with two goals; firstly, to avoid collision of robots with foreign objects (environment obstacles and humans) and secondly, with the ability to detect potential collisions in advance.

The work presented in this paper concerns the first goal, i.e., obstacle avoidance for an industrial robotic arm or so-called manipulator. Mounted on a stationary platform, its links with revolute joints and end-effector can move with a certain degree of freedom. The solution deals with forward and inverse kinematics of the robotic arm. With forward kinematics, we aim to calculate the end-effector pose from the position of joints. Inverse kinematics deals with the opposite problem; given the position of the end-effector in the configuration space, we must work out the angles that each joint should have to reach that configuration. Given a robot with a description of its kinematics, a description of the environment (representation of free and occupied spaces), an initial state, and a goal state, the motion planning problem is to find a sequence of inputs that drive the robot from its initial state to the goal state while obeying the rules of the environment, e.g., not colliding with the surrounding obstacles (Figure 1). The manipulator will follow this path to reach the goal position. This work, however, explores the idea for static scenario and later we develop it for the dynamic case.

We follow a sampling-based path planning approach called Probabilistic Roadmap Planner (PRM) [5]. The method randomly samples configurations from the configuration space of the manipulator. Following this, it builds a roadmap graph of free space. Finally, it connects the initial and goal configurations to this roadmap and finds the shortest path from the start to the target using some standard algorithms such as A* [6] or Dijkstra's algorithm [7].

Our work is in the context of the project INDTECH 4.0 [8], which aims at developing new technologies for intelligent manufacturing such as Collaborative Robotics, Autonomous Drive Systems, Decision Support Systems (DSS). In the lines of INDTECH 4.0, different works have focused on these areas [9][10][11][12]. Reusable software components are major building blocks for modern Cyber-Physical Systems (CPS), efficiently managing the complexity by supporting modular development and composability. Neto and Gonçalves [9] studies component models for many industrial CPS for understanding design choices and application targets, and Neto et al. [10] designs a component framework following component-based software engineering principles. Reis et al. [11] demonstrates a collaboration scenario between human and a simple robotic manipulator in the context of Cyber-Physical Production Systems (CPPS).

The rest of the paper is organized as follows. In Sec-

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-723-8

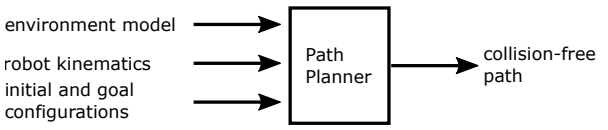

Figure 1: Robot motion planner, a top-level view.

tion II, we present some concepts and background important to the work. Section III lists some works regarding robotics path planning. Section IV presents the developed approach as well as some preliminary evaluation. In Section V, we propose a path planning architecture based on a voxelized grid. Section VI discusses merits and potential of the proposed approach. Finally in Section VII, we conclude the paper and present some future work directions.

II. Background and Concepts

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 placements of the moving object, i.e., space of all configurations [13]. It is important to distinguish between *work space* and the *configuration space*. *work space* represents the actual environment or the world where the manipulator and the obstacles exist; *configuration space* is the representation of the *work space* that we use in our implementations to aid the path planning solutions. In the following, we review two common approaches for path planning, namely, configuration space-based and sampling-based approach.

A. Path planning - configuration space approach

In order to find a trajectory in the configuration space $\mathcal{C}$, the main difficulty lies in the high computation complexity in constructing $\mathcal{C}$. The dimension of the configuration space is determined by the degrees of freedom of the robot. A configuration q is given as 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 $q = \langle q_1, q_2, q_3, q_4, q_5, q_6 \rangle$ 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 work space obstacles. A path is a continuous sequence of configurations in $\mathcal{C}_{free}$ connecting initial and goal configurations. Constructing $\mathcal{C}$ entails creating a map of $\mathcal{C}$ such that collision regions can be identified. One way is to discretize $\mathcal{C}$ and test each discretized configuration if it is collision-free. The resolution or granularity of the discretized $\mathcal{C}$ has a significant impact on the performance of path finding algorithm. A fine resolution will increase the search space as well as computation time whereas a coarse resolution might miss a valid path when there exists one. We illustrate this with an example of a robot manipulator with six revolute joints and each joint can rotate between -180° to 180°. With a discretization resolution of 1°, size of $\mathcal{C}$ is $360^6 \approx 2.18 \times 10^{15}$ points. Thus, it leads to a huge search space when explicitly constructing collision regions in $\mathcal{C}$.

B. Sample-based planning

To avoid the great computational complexity and the state explosion problem, motion planning algorithms often employ sampling-based planning together with a collision detection module. Such methods first generate a roadmap representing connectivity of $\mathcal{C}$ and later path search requests are processed on this roadmap. Sampling based planners can often create plans in high-dimensional spaces efficiently.

Approaches to the path planning problem can be divided into two classes *single-query* and *multiple-query* planning. For single-query planning, a one-time solution to a unique problem is defined, without preprocessing. Multiple-query planning applies to cases where we need to solve different problems for the same environment. Thus, such approaches form a map of the workspace, and then, multiple queries are issued at runtime to find trajectories in that map. The problems relating to dynamic environments fit into the class of single-query planners.

C. kd-trees

Initially examined in [14], a kd-tree is a data structure for storing and searching finite points in a k-dimensional space. Each node represents a subset of the dataset as well as partitioning of the subset. Each leaf node is a k-dimensional point. Each non-leaf node generates an implicit hyperplane splitting the space into two halves. Alike a binary tree, values to the left of the hyperplane are less than or equal to the value at the parent and constitute the left sub-tree, while values to the right of the hyperplane are larger and form the right sub-tree. Further, any node is assigned to one of the k dimensions with hyperplane perpendicular to that dimension's axis. The splitting dimension for each node is selected based on its level in the tree; we obtain the splitting axis for each level by cycling through the k dimensions in order, given by the following rule, $D = L \ mod \ k + 1$ where D is the splitting axis for level L and the root is defined to be at level zero [14].

Nearest neighbour search is an essential component of many path planning strategies. Some approaches generate thousand of vertices; finding closest vertices to connect to in huge search space is a challenging task. kd-trees can efficiently aid the nearest neighbour search by quickly eliminating large portions of the search space. Partitioning strategy in k-d trees allows keeping the average number of examinations small when searching for the best matches to a query item. For the experimental exercise reported in this paper, we naively use kd-trees, i.e. based on Euclidean distance. However, Yershova and LaValle [15] shows that the distance metric can be tailored to account for complex spaces encountered in motion planning scenarios.

III. Related Work

In the past years, much research has focused on path planning for industrial robots. A fundamental robotic task is to navigate from an initial position to a goal position while avoiding the set of obstacles. The developed approaches encompass static scenarios with fixed operational space of robot or dynamic situations where humans or obstacles can

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-723-8

move in the functional area of the robot. Other classifications include configuration space approaches vs the random sampling approaches, or single query based planners vs multiple-query based planners. For a detailed review of different path planning approaches, refer [16]. Below, we describe a few basic methods and some works that employ these approaches to solve motion planning problems.

Rapidly-exploring random trees (RRT) [1] and its several improvements, such as RRT-connect [17], RRT* [18] are sample-based, single-query path planning algorithms. The basic idea of randomly sampling free space is similar to PRMs [5]. The tree is rooted at the start configuration. A random free sample x_{rand} is generated, and if it is within a distance ϵ to the closest tree node $x_{nearest}$, a direct link is created between the two. Otherwise, a new node x_{new} is generated, which is at most within ϵ from $x_{nearest}$ and in the direction of x_{rand}. A connection is formed between $x_{nearest}$ and x_{new}, and x_{rand} is discarded. Random samples can be seen as controlling the expansion direction of an RRT. Local planner checks for collision avoidance when making connections. RRT effectively biases the search towards unexplored regions of the search space. As opposed to PRM, RRT remains connected even with a small number of edges. Instead of defining a goal state, a goal region can be defined. When the tree expansion falls in this region, we have a possible path connecting start to the goal. The RRT-connect approach differs from the basic RRT in two aspects. Firstly, instead of the incremental growth by ϵ, it grows the tree to the new random node x_{rand} or until an obstacle is encountered. Secondly, it uses two RRTs, one rooted at q_{init} and the second at q_{goal}. The trees are maintained until they get connected and hence a path is found. Within RRT*, the cost of reaching each vertex from q_{init} is recorded. Alike RRT, initially, RRT* extends the nearest tree node $x_{nearest}$ towards the random sample x_{rand} thus giving the new node x_{new}. However, it then examines a neighbourhood of vertices in a fixed radius around x_{new}. If any such vertex x_{near} incurs the minimum accumulated cost to reach x_{new}, then, it is made the parent of x_{new}. Another difference is that RRT* rewires the tree. It tests all x_{near} and if any such vertex can be accessed through x_{new} with a smaller cost, then x_{near} is rewired to x_{new} making the path more smooth.

PRMs (see Section IV) belong in the category of multiple-query sample-based planners. Having constructed the roadmap of $\mathcal{C}$, we can put multiple queries to the roadmap, each time specifying different end configurations.

Bertram et al. [19] presents a strategy for finding a solution when goal configuration is unreachable as it lies in a disconnected component of the configuration space with respect to the initial configuration. The main idea is to integrate the IK solution directly into the path planning process. Instead of specifying an explicit goal configuration, the planner evaluates configurations that might belong to a valid goal region. A heuristic workspace goal function calculates proximity to the target given the end-effector pose. It uses forward kinematics to find the end-effector pose from current configuration q. The function uses a weighted sum of different factors to characterize the goal region. The foremost factor is the Euclidean distance between the origin of the coordinate frame attached to the end-effector and the centre of the target object; penalty terms are added to constrain the orientation of the end-effector. It uses a modification of the RRT algorithm to guide the search. It ranks configurations based on their distance to an obstacle and their goal distance. New node goal distance must be smaller than its parent. A node which is in a collision or its goal distance is not lower than its parent is removed from the search after some failure count. The tree grows in configurations that reduce the goal distance.

Qin and Henrich [20] presents a parallel randomized approach for the path-finding problem. This work uses a discretized version of the C-space already discounting C-space regions where the arm has mechanical limitations to reach and where obstacles lie. The idea is to generate many sub-goals in free C-space. Then, it uses several parallel processes where each process will find a path connecting initial configuration with the goal configuration through a sub-goal. All processes terminate whenever a process returns a valid path. The transition from the current state to the next state is based on a cost function which selects the candidate with the minimum cost, also considering that it is within the workspace and collision-free.

Henrich et al. [21] proposes a strategy to reduce C-space size using different discretizations along each coordinate of the configuration vector, with joints closer to the base having a finer discretization resolution. The so-called optimized discretization is such that each joint results in the same movement in the Cartesian space when rotated by the chosen angle. The authors provide a formula for computing the desired angle that takes into account the distance from the joint centre to the farthest point on the end effector.

In this work, we implement a PRM approach for a simple space in $\mathbb{R}^2$, and we consider a point robot that must navigate the area. However, as the method developes, we integrate a collision detection module, that makes use of a voxel-based grid.

IV. Approach

This work is a preliminary exercise towards finding a solution to the robotic arm path planning problem. Here, we consider a static environment, and we follow a sampling-based approach called probabilistic roadmap (PRM) planner [5]. The method consists of two phases, a *learning phase* and a *query phase*. Rather than computing the configuration space explicitly, we sample it during the learning phase. This phase constructs the roadmap as a graph where sampled configurations are vertices and connections between configurations are the edges. In the query phase, we ask for a path between two free configurations. Collision detection can be an independent module and can serve in different phases of the construction of trajectory. To allow adequate connectivity of $\mathcal{C}$, we sample many configurations widely distributed over the free space.

A. Learning phase

The algorithm works as follows. We begin with an empty graph G. In the *learning phase*, we repeatedly sample a random configuration q from $\mathcal{C}$. There can be a collision detection test that checks whether the selected configuration is free i.e., $q \in \mathcal{C}_{free}$. If this check succeeds, q is added to the set of vertices V. This process continues until a given number of nodes n have been added to V. n is a parameter of the algorithm and is the desired number of nodes in the graph. In the second step, for each vertex of G, we find its k nearest neighbours denoted $P = \{p_1, p_2, \cdots, p_k\}$ according to some $dist$ metric.

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-723-8

Then, an edge is created from q to its nearest neighbours, i.e., $\forall p_i \in P, E = E \cup (q, p_i)$ where E is the list of edges in G. At the end of the *learning phase*, we have an undirected graph $G = (V, E)$. Due to the random nature of sampling, it is possible that roadmap G has disconnected components.

B. Query phase

The next step is the *query phase*. We are given the initial and goal configuration of the robot denoted respectively q_{init} and q_{goal}. The algorithm must find a feasible path connecting q_{init} and q_{goal} or return failure. We find k nearest neighbours of q_{init} and q_{goal} from V using some distance metric $dist$. These sets are denoted $P_{q_{init}}$ and $P_{q_{goal}}$ respectively for q_{init} and q_{goal}. Then, we check each element e of $P_{q_{init}}$ in order and add an edge $E = E \cup (q_{init}, e)$. Similarly, we add an edge connecting $P_{q_{goal}}$ in the roadmap. Finally, we can use any shortest path algorithm such as Dijkstra's algorithm [7] to find the path between initial and goal configurations. Algorithm might fail to return a path when roadmap has disconnected components. The working of the PRM algorithm can be explained as shown in Figure 2. We assume an Euclidean space $\mathbb{R}^2$. Robot configuration is given by a point in $\mathbb{R}^2$, depicted with an empty circle whereas shaded regions represent the obstacles. Figure 2(a) shows many random samples of the free space which are nodes of the roadmap. In Figure 2(b), we form edges between nodes using k closest neighbours. k is three, but the degree of some nodes can be greater when it is included in the closest neighbours of different nodes, or it can be smaller if it cannot find k closest neighbours due to an obstacle. In Figure 2(c), we solve a query in the roadmap. The configurations q_{init} and q_{goal} are connected to the roadmap. Then, a graph-based search algorithm returns the shortest path denoted by thick lines.

C. Remarks

Several essential aspects affect the performance of PRM; for instance, how can we choose the random configurations such that sampling of the configuration space is uniform. The original PRM uses a local planner. This local planner is instrumental for creating connections between any two sampled configurations hence creating a feasible local path. The task of the local planner is to interpolate the motion of the robot between the two samples, checking for collisions at a given resolution. If no configuration of the robot between the samples collides with an obstacle, then an edge is inserted to the roadmap. Both phases of the PRM algorithm employ the local planner. In the presented work, however, we have not used a local planner. Instead, having found the nearest neighbours using kd-tree, we make connections creating edges in the graph. We use kd-tree in both phases of PRM. In the reported implementation, kd-tree use Euclidean distance when finding nearest neighbours. However, due to the complex topology of the configuration space, such distance metric may not be directly applicable to the path planning algorithms.

D. Preliminary evaluation

We have implemented the approach in a simple 2d test scenario. We create a two-dimensional point grid that represents configuration space $\mathcal{C}$. We also specify the obstacle points which is a subset of $\mathcal{C}$ and represents the occupied or forbidden part of the space, i.e., $\mathcal{C}_{forb}$. The number of points in $\mathcal{C}$ and $\mathcal{C}_{forb}$ is a parameter of the algorithm. When creating the roadmap, we also specify the desired number of points to sample and also the number of closest neighbours to connect to. We use kd-tree and we find the nearest neighbours based on Euclidean distance, i.e., for points $p = (x_1, y_1)$ and $p_2 = (x_2, y_2)$, we calculate distance $d(p, q) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$. When forming edges, d serves as the weight on the respective edge.

Figure 3 shows an example configuration space with 40 free points (blue) and 10 collision points (red). In the next step, we sample $\mathcal{C}$ and construct a roadmap in the free region. The following 15 points were randomly selected as vertices of the roadmap.

$V = \{(11, 4), (12, 8), (14, 6), (39, 12), (12, 7), (24, 27), (46, 3), (22, 33), (15, 0), (23, 25), (11, 9), (8, 32), (6, 16), (4, 37), (29, 46)\}$

We use 2 nearest neighbours while forming the edges between nodes. Figure 4 shows the roadmap. Finally, we query the roadmap to find a path between an initial configuration $(22, 33)$ and a goal configuration $(11, 9)$. For this exercise, these configurations have been selected from the constructed roadmap. Generally, q_{init} and q_goal must be added to the roadmap at this stage. Using Dijkstra's algorithm and employing $d(p, q)$ as the cost of going from p to q, the following shortest path is returned (Figure 5). $\{(22, 33), (23, 25), (39, 12), (46, 3), (15, 0), (14, 6), (12, 8), (11, 9)\}$

In this way, we can solve a query as long as the roadmap is connected. The connectivity problem gets more complicated as the size of the configuration space increases. To this end, different measures can help, such as increasing the number of nearest neighbours to look at or increasing the number of sampled points.

V. Proposed architecture

In our proposed architecture, we integrate the probabilistic planner with a grid-based approach for efficiency. The work in [22] explores such a relation. Figure 6 shows the main components of the architecture. In particular, the framework receives the initial and goal configurations q_{init} and q_{goal} and returns a collision-free path. The main component of the architecture is the PRM which we explained in Section IV. Essential tasks of the planner are to build a roadmap and establish local paths. These tasks rely on efficient nearest neighbour search and collision detection, which we add as independent modules within the architecture. Nearest neighbour search can use kd-trees, but it also needs information for collision-free nodes as well as an appropriate distance metric. Many works have used slight modifications of Euclidean distance [5][19] whereas others specify different metrics for more complex spaces [15]. Local planner decides whether we can add an edge to the roadmap. It has the task to interpolate robot motion between two given configurations (nodes from the roadmap). For this purpose, it uses the kinematics model of the robot and checks in the voxel grid if a given configuration is collision-free. 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

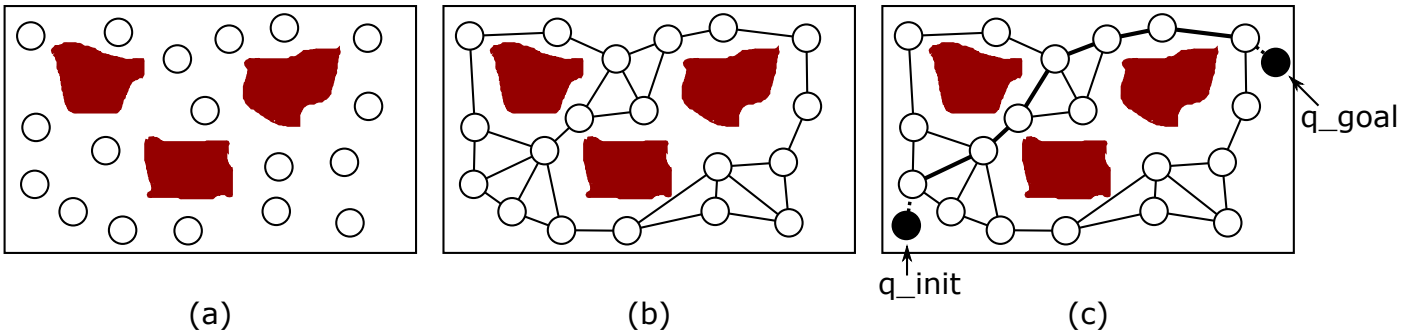

Figure 2: An example of a roadmap in a two-dimensional Euclidean space.

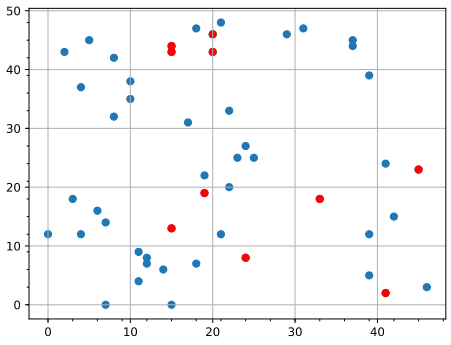

Figure 3: Two dimensional configuration space.

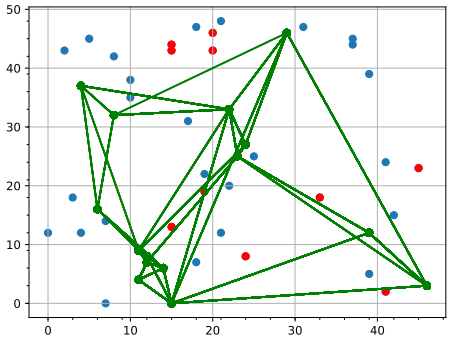

Figure 4: Randomly sampled points. Straight lines between points correspond to edges. The number of k closest neighbours for the construction of roadmap is 2.

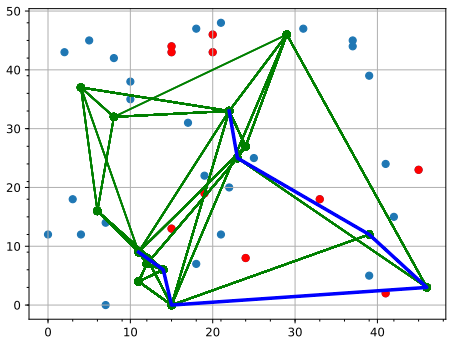

Figure 5: An example of solving a query with the roadmap. Shortest path between initial and goal configurations is depicted with thick lines.

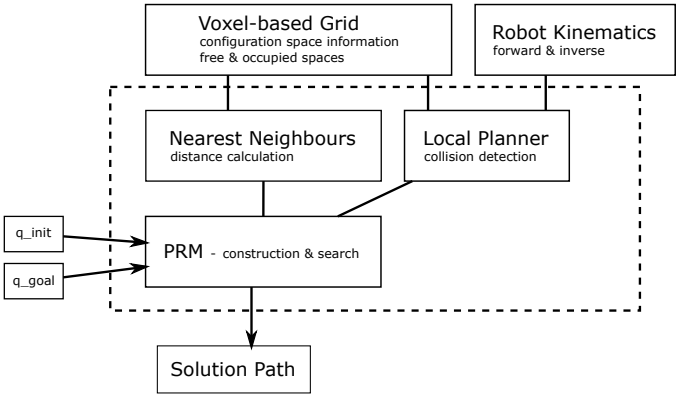

Figure 6: Robot motion planning approach - architecture, environment information is given with a labeled voxel-grid.

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 verse. When voxels in the grid are labelled, we can identify, firstly, 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. The resolution has a direct impact regarding collision detection as the local planner interpolates motion at the granularity of grid resolution. When using PRM and targeting static scenario, construction of a labelled grid can be a *preprocessing* step without impacting on the processing of queries online. The knowledge of free and occupied space is a vital component of the collision detection module within the path planning algorithm.

The modular architecture allows easy experimentation; for instance, we can use different planners or strategies for nearest neighbour searching or local planning, in future.

VI. Discussion

A desirable property of the developed solution is its real-time performance. High computational complexity is the main limitation of many path planning algorithms, preventing online recalculation of trajectories within the response time of the manipulator [23]. Thus, dynamic environments, which inevitably occur during collaborative scenarios, may not be handled in real-time, and most planners remain applicable to static environments, only. We contend that using a sampling-based approach combined with a voxel-based grid search would support real-time performance, as well as enhance the accuracy of the plan. In the planning algorithm, we can attribute a significant computation time to the following key components, i.e.,

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-723-8

nearest neighbour searching, and collision checking. Firstly, we see how the voxel-based grid supports faster computations for the above two elements. Secondly, we see in which ways can we optimize the nearest neighbour searching and collision checking to enhance algorithm performance.

Nearest neighbour search implicitly checks for collision when it finds a new neighbour. We can efficiently find neighbours in the search space using data structure, such as kd-trees. Since we build the kd-tree from the roadmap, we must account for the time it takes to construct the tree. With an offline preprocessing phase, as in PRM, this computation effort is irrelevant. However, with online planning, we must find the tradeoff between the time taken to construct the tree, and time saved in neighbour searching.

We envision that voxel-based grid search would offer most savings in collision checking for both static and dynamic scenarios. The collision checking in PRM is managed with a so-called local planner. Local planner interpolates robot motion between configurations q_s and q_f. For this purpose, at a given discretization level δ of the configuration space, it advances coordinates of q by δ to reach an intermediate configuration q_i. It keeps advancing until it reaches q_s, testing each q_i if it is collision free. Checking for a collision at a given q_i is equivalent to getting the voxels corresponding to a q_i (robot joint values) in the grid, and their occupancy status. Hence, the local planner can easily find step path. Apart from testing for environment obstacles that lie outside the robot body, collision checking must also check for self-collisions which involve motions where different links might obstruct each other.

The proposed approach will be more flexible in terms of implementation. Since path planning and voxel grid are independent modules, thus, other planners could be combined with the voxel grid.

For a uniform random sampling of $\mathcal{C}$, when choosing a configuration q, we use a uniform probability distribution over an interval of values corresponding to dof of each coordinate in q.

In terms of practical impact, this approach will benefit the collaborative environments. Collaborative spaces reduce costs and improve production volumes by allowing humans and robots to work side-by-side. In this way, humans and robots can perform specific tasks they are best at, e.g., robots perform heavy tasks while humans can inspect for quality. The main concern here is the safety; robot must react in real-time in case of hazardous events, necessitating efficient motion planners. An example of manipulator deployment is within an industrial assembly line which typically distributes the workload among several robots. The main concern is to make the production process efficient such that the final product fulfils its functional requirements, as well as reduces the production time and time to market. A small variation in single parts may propagate such that the final product does not comply with specifications. In such a setting, deriving collision-free paths for each worker is of particular importance. To maximize the number of units assembled means reducing the time needed at each station (robot) to perform its specific task, referred to as the process cycle time [24]. Practical efficiency favours sample-based planners over the configuration space approach. Such planners can derive more robust motions with shorter cycle times [25].

VII. Conclusion and Future Work

The robotic path planning is a classic problem. In this paper, we presented a simple implementation of a robot motion planner based on a sampling approach. We used kd trees for efficient nearest neighbours search. Random sampling might result in roadmaps with disconnected components, and thus, it will fail to find a path when start and goal configurations lie in disconnected components. We would investigate methods to find adequate connectivity of $\mathcal{C}_{free}$. Therefore, to guide the sample selection, our future work would study techniques that reduce the number of samples as well as improve the final roadmap quality. Next step involves testing the presented approach for real scenarios. For such cases, we need to represent configurations in terms of robot joint values instead of the point robot, and to implement the kinematics equations as shown in Figure 1. However, we can find a way to combine the two planning approaches, namely the configuration space approach and the sample based approach for efficiency. A voxel-based representation of the configuration space with additional information on the environment (e.g., the *reachability grid* [26]) can simplify the local planner used to detect collisions in the sampling approach.

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] 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.

[2] “MHI - The Industry That Makes Supply Chains Work,” http://www.mhi.org/fundamentals/robots, accessed: 2019-05-19.

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

[4] 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.

[5] 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, pp. 566–580, August 1996.

[6] 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, pp. 100–107, July 1968.

[7] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” *Numerische Mathematik*, vol. 1, pp. 269–271, December 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[8] “INDTECH 4.0 – Novas tecnologias para fabricação Inteligente, INDTECH 4.0 - New Technologies for Intelligent Manufacturing,” https://site.groupe-

psa.com/mangualde/pt-pt/atualidades/atividade/centro-de-mangualde-do-groupe-psa-na-vanguarda-4a-revolucao-industrial-projeto-indtech-4-0-em-execucao/, accessed: 2019-05-29.

[9] L. Neto and G. M. Gonçalves, "Component Models for Embedded Systems in Industrial Cyber-Physical Systems," in *Proceedings of the 7th International Conference on Intelligent Systems and Applications (INTELLI 2018)*. IARIA, June 2018, pp. 24–29.

[10] L. Neto *et al.*, "A component framework as an enabler for industrial cyber physical systems," in *Proceedings of the 1st IEEE International Conference on Industrial Cyber-Physical Systems (ICPS 2018)*. IEEE, May 2018, pp. 339–344.

[11] J. Reis, R. Pinto, and G. Gonçalves, "Human-Centered Application using Cyber-Physical Production System," in *Proceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society (IECON 2017)*. IEEE, November 2017, pp. 8634–8639.

[12] L. Antão, R. Pinto, J. Reis, G. Gonçalves, and F. L. Pereira, "Cooperative Human-Machine Interaction in Industrial Environments," in *Proceedings of the 13th APCA International Conference on Control and Soft Computing (CONTROLO 2018)*. IEEE, June 2018, pp. 430–435.

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

[14] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software*, vol. 3, no. SLAC-PUB-1549-REV. 2, pp. 209–226, 1976.

[15] A. Yershova and S. M. LaValle, "Improving Motion Planning Algorithms by Efficient Nearest-Neighbor Searching," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 151–157, February 2007.

[16] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[17] 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.

[18] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[19] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An Integrated Approach to Inverse Kinematics and Path Planning for Redundant Manipulators," in *Proceedings of the 23rd IEEE International Conference on Robotics and Automation (ICRA 2006)*. IEEE, May 2006, pp. 1874–1879.

[20] C. Qin and D. Henrich, "Path Planning for Industrial Robot arms - A Parallel Randomized Approach," in *Proceedings of the 4th International Symposium on Intelligent Robotic Systems (SIRS 1996)*. Citeseer, July 1996, pp. 65–72.

[21] D. Henrich, C. Wurll, and H. Wörn, "Online path planning with optimal C-space discretization," in *Proceedings of the 11th IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 1998). Innovations in Theory, Practice and Applications*. IEEE, October 1998, pp. 1479–1484.

[22] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the Relationship Between Classical Grid Search and Probabilistic Roadmaps," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 673–692, 2004.

[23] T. Petrič, A. Gams, N. Likar, and L. Žlajpah, "Obstacle Avoidance with Industrial Robots," in *Motion and Operation Planning of Robotic Systems*. Springer, March 2015, pp. 113–145.

[24] N. Papakostas, G. Michalos, S. Makris, D. Zouzias, and G. Chryssolouris, "Industrial Applications with Cooperating Robots for the Flexible Assembly," *International Journal of Computer Integrated Manufacturing*, vol. 24, no. 7, pp. 650–660, June 2011.

[25] D. Spensieri, "Planning Robotic Assembly Sequences," Ph.D. dissertation, Chalmers University of Technology, March 2017.

[26] P. Anderson-Sprecher and R. Simmons, "Voxel-Based Motion Bounding and Workspace Estimation for Robotic Manipulators," in *Proceedings of the 29th IEEE International Conference on Robotics and Automation (ICRA 2012)*. IEEE, May 2012, pp. 2141–2146.

Copyright (c) IARIA, 2019. ISBN: 978-1-61208-723-8