34th International Symposium on Automation and Robotics in Construction (ISARC 2017)

Cartesian-Space Motion Planning for Autonomous Construction Machines

M. Ragaglia[a], A. Argiolas[a] and M. Niccolini[a]

[a] Yanmar Research & Development Europe, Viale Galileo 3/A, 50125, Firenze, Italy
E-mail: matteo_ragaglia@yanmar.com, alfredo_argiolas@yanmar.com, marta_niccolini@yanmar.com

Abstract -

Nowadays, the construction industry is probably the least productive and most dangerous among the various industry sectors. Given this scenario, it is quite clear that the introduction of Autonomous Construction Machines (ACMs) could represent a great opportunity to improve both productivity and safety. To this purpose, a fundamental problem that has to be tackled is trajectory planning. In the last 15 years, several sample-based algorithms have been proposed, that relies on Joint-Space sampling. Unfortunately, this feature often results in trajectories that are quite counterintuitive from the point of view of a human being. In this work we propose "cart-RRT", a Cartesian-Space randomized algorithm that improves the intuitiveness of the output trajectory, while ensuring both its safety (in terms of collision avoidance) and its feasibility.

Keywords -

Robotics; Motion Planning; Construction Machines;

1 Introduction

Nowadays, in the construction industry, several operations that require both high power and high accuracy (such as panel positioning, plumbing, material handling) are still manually performed by human workers, in very inefficient and dangerous ways. After a thorough field investigation, we believe that the construction industry could definitely benefit from the introduction of Autonomous Construction Machines (ACMs), in terms of increased productivity and human workers' safety. Among the huge number of functionalities that are required to develop an ACM, motion planning plays a fundamental role. As a matter of fact, the planner has the responsibility to compute a trajectory that allows the ACM to travel from its original configuration to a desired one, while satisfying kino-dynamic constraints and avoiding collision with obstacles.

Given the fact that robots are complex systems, often characterized by nonlinear dynamics and actuation constraints, the problem of trajectory planning can easily become computationally intractable. For this reason, randomized sample-based planning algorithms [7] have emerged as a reliable and appealing alternative to search-based planning techniques [9] and model predictive control approaches [14]. Randomized sample-based planners like Rapidly-exploring Random Trees (RRT) [8] have experienced a growing popularity since their introduction in the late '90s. The main reason behind this success relies in a rather simple, yet effective idea: sample multiple robot configurations that do not entail collisions with obstacles and connect them to build either a graph or a tree of feasible trajectories. Then, a solution is extracted from this tree (or graph), in terms of a sequence of edges that connect couples of nodes. Furthermore, it has been demonstrated in [1] that sample-based planning algorithms are probabilistically complete. Even though they have been originally introduced to solve the trajectory planning problem for holonomic robots, RRT-based algorithms have been extended to tackle more complex problems, like for instance the optimal and constrained trajectory planning. Several solutions have been proposed in the last few years, including optimal and non-linear RRTs [2], Rapidly-exploring Random Graphs (RRG), and RRT* [5, 4]. Several approaches have also been proposed to solve the optimal and constrained trajectory planning problem for arbitrary kino-dynamic systems, like for instance [3, 11, 12, 13].

A typical feature of all these randomized planning algorithms consists in sampling the nodes directly in the robot Configuration-Space, also known as Joint-Space. Unfortunately, Joint-Space randomized algorithms tend to output quite counterintuitive trajectories from the point of view of a human being. Given the fact that in construction sites humans and machines usually work side-by-side, and considering the results presented in [15], it is clear that the intuitiveness of the planned trajectories can have a strong impact on the comfort, the productivity and the level of safety perceived by human workers sharing their workspace with ACMs. In order to avoid these drawbacks, a possible solution is represented by addressing the motion planning problem directly at the Cartesian-Space level. We here propose "cart-RRT", a Cartesian-Space randomized algorithm that allows the tree to be extended directly towards the goal, as it has been originally proposed in [6]. In cart-RRT edges are computed as linear segments in the Cartesian-Space and the corresponding joint positions profiles are determined using Inverse Kinematics. Once the tree reaches the goal, a path shortening procedure is performed to reduce the length of the

output trajectory while keeping it collision-free. Then, a spline-based interpolation is used to eliminate Joint-Space velocity discontinuities. Finally, kinematic scaling is applied to guarantee that joint velocities do not exceed their lower and upper bounds. The proposed algorithm has been tested and validated in multiple simulated scenarios, using the model of a UR10 manipulator as test bench.

2 Proposed Algorithm

One of the fundamental features of cart-RRT is the fact that nodes are computed as linear segments in the Cartesian-Space. In order to draw linear edges in the Cartesian-Space, while computing their counterpart in Joint-Space, the algorithm relies on the following data-structures:

- **node**: the basic element of the random tree. It is defined by: a set of joint positions $\boldsymbol{q}$, the corresponding Cartesian pose $\boldsymbol{p}$, and a reference to its parent node within the tree n_{par};
- **edge**: the element that connects two distinct nodes. It is defined by two matrices: $\boldsymbol{Q}$ and $\boldsymbol{P}$ that contain, respectively, the series of intermediate Joint-Space and Cartesian-Space poses visited by the robot while traveling from the starting node to the destination one;
- **Tree**: the random tree built by the algorithm. It contains two main fields: a list of nodes named **Nodes**, and a list of edges named **Edges**.

Algorithm 1 explains how the cart-RRT planner works. initially, the tree only contains the starting node n_0. Then, following the approach originally proposed in [6], a random selector is sampled to select between the two possible extension modalities: random and goal-oriented. To this purpose, the value of the parametric threshold "$rand_{th}$" plays a fundamental role, since it affects the balance between exploration (random expansion) and exploitation (goal-oriented expansion). Once the tree has been extended, the algorithms checks if the goal region "$cart_g$" has been reached. If the tree reaches the goal region before its cardinality overcomes the value of the "maxNodes" parameter, the algorithm extracts a trajectory from the tree (function "GetTrajectory"). Then, it applies a post-processing technique (procedure "PostProcess", whose details are given in Section 2.4) and, finally, it returns the output trajectory. Otherwise, the algorithm returns a failure (i.e. a "null" value).

Algorithm 1 cart-RRT Algorithm pseudo-code

```
1:  Global Parameters: rand_th, maxNodes
2:  procedure CART-RRT(n_0, cart_g)
3:      Tree.Nodes ← n_0
4:      Tree.Edges ← ∅
5:      while (|Tree.Nodes| ≤ maxNodes)
6:          rand ← RAND (0, 1)
7:          if (rand ≤ rand_th)
8:              Tree ← ExtendRandomly (Tree)
9:          else
10:             Tree ← ExtendTowardsGoal (Tree, cart_g)
11:         end if
12:         if (∃ n ∈ Tree.Nodes ⟹ FKine (n.q) ∈ cart_g)
13:             trajectory ← GetTrajectory (Tree)
14:             trajectory ← PostProcess (trajectory)
15:             return trajectory
16:         end if
17:     end while
18:     return null
19: end procedure
```

2.1 Tree Extension

The two procedures that allow to extend the random tree are detailed in Algorithms 2 and 3. The first procedure implements the standard RRRT extension strategy, while the latter tries to establish a series of nodes connecting the tree directly with the goal region.

Algorithm 2 Random extension procedure

```
1:  Parameters: maxLength
2:  procedure EXTENDRANDOMLY(Tree)
3:      n_rnd ← RandomSampling ()
4:      n_near ← NearestNode (Tree.nodes, n_rnd)
5:      n_new ← GetNewNode (n_near, n_rnd, maxLength)
6:      edge ← EdgeCalc (n_near, n_new)
7:      if (edge ≠ null)
8:          n_new.n_par ← n_near
9:          Tree.Nodes ← Tree.Nodes ∪ n_new
10:         Tree.Edges ← Tree.Edges ∪ edge
11:     end if
12:     return Tree
13: end procedure
```

2.2 Edge Calculation

One of the fundamental features of cart-RRT is the fact that nodes are computed as linear segments in the Cartesian-Space. Given two distinct nodes in the tree, the connecting edge is computed according to Algorithm 4. Starting from the first node, the procedure iteratively applies differential inverse kinematics in order to reach the destination node. Intermediate Joint-Space and Cartesian-Space positions are stored in the output variable "edge". A threshold distance (parameter "edgeThresh") is set in order to exit the procedure once a given neighborhood of the destination node is reached. The procedure returns a failure in two distinct cases: whenever a kinematic singularity is encountered and in case the inverse kinematics does not converge after a given number of iterations (parameter "edgeCountMax").

Algorithm 3 Goal-oriented extension procedure

Parameters: maxLength
procedure EXTENDTOWARDSGOAL(**Tree**, cart_g)
  $\text{n}_{rnd} \leftarrow \text{SampleFromGoal}\left(\text{cart}_g\right)$
  $\text{n}_{old} \leftarrow \text{NearestNode}\left(\textbf{Tree.nodes}, \text{n}_{rnd}\right)$
  while (1)
    $\text{n}_{new} \leftarrow \text{GetNewNode}\left(\text{n}_{old}, \text{n}_{rnd}, \text{maxLength}\right)$
    $\text{edge} \leftarrow \text{EdgeCalc}\left(\text{n}_{old}, \text{n}_{new}\right)$
    if (edge $\neq$ null)
      $\text{n}_{new}.\text{n}_{par} \leftarrow \text{n}_{old}$
      Tree.Nodes $\leftarrow$ **Tree.Nodes** $\cup\ \text{n}_{new}$
      Tree.Edges $\leftarrow$ **Tree.Edges** $\cup$ edge
      if $\left(\text{FKine}\left(\text{n}_{new}.\boldsymbol{q}\right) \in \text{cart}_g\right)$
        return Tree
      else
        $\text{n}_{old} \leftarrow \text{n}_{new}$
      end if
    else
      return Tree
    end if
  end while
end procedure

Algorithm 4 Edge calculation procedure pseudo-code

Parameters: edgeThresh, edgeCountMax
procedure EDGECALC(n_0, n_f)
  $\text{edge}.\boldsymbol{Q} \leftarrow \emptyset$
  $\text{edge}.\boldsymbol{P} \leftarrow \emptyset$
  $\text{reached} \leftarrow 0$
  $\text{counter} \leftarrow 0$
  $\boldsymbol{q} = \text{n}_0.\boldsymbol{q}$
  while (reached = 0)
    if (counter > edgeCountMax)
      return null
    end if
    $\text{edge}.\boldsymbol{Q} \leftarrow \text{edge}.\boldsymbol{Q} \cup \boldsymbol{q}$
    $\text{edge}.\boldsymbol{P} \leftarrow \text{edge}.\boldsymbol{P} \cup \boldsymbol{p}$
    $\boldsymbol{p} = \text{FKine}\left(\boldsymbol{q}\right)$
    $\Delta\boldsymbol{p} = \text{n}_f.\boldsymbol{p} - \boldsymbol{p}$
    if $(\|\Delta\boldsymbol{p}\| < \text{edgeThresh})$
      $\text{reached} \leftarrow 1$
    else
      if $(\det\left(\boldsymbol{J}\left(\boldsymbol{q}\right)\right) \approx 0)$
        return null
      else
        $\boldsymbol{q} = \boldsymbol{q} + \left(\boldsymbol{J}\left(\boldsymbol{q}\right)^{-1} \Delta\boldsymbol{p}\right)$
      end if
    end if
    $\text{counter} \leftarrow \text{counter} + 1$
  end while
  return edge
end procedure

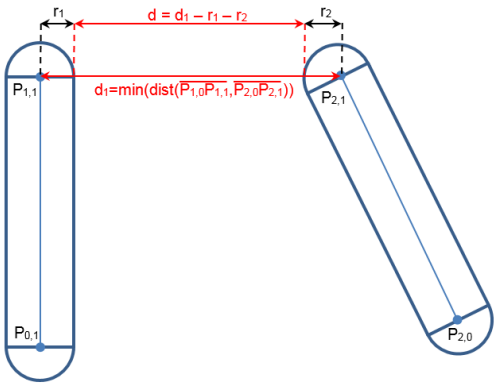


Figure 1. Minimum distance between two capsules.

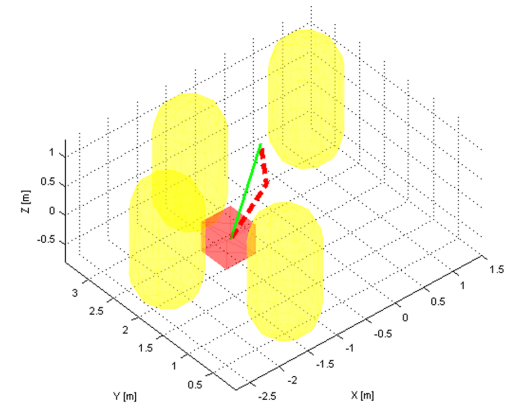
2.3 Collision Checking Strategy

One of the fundamental element of every sample-based planning algorithm (and also one of their major bottlenecks) is represented by the collision-checking strategy. As a matter of fact, directly taking into account the complex geometry of a robot and of generic obstacles would almost surely result in enormous computation time spent on checking possible collisions. For this reason, a capsule-based geometric model of both the robot and the obstacles has been adopted. We chose capsules since they represent a quite simple (yet effective) tool to model complex geometries without being too conservative. To define a capsule only two points in the Cartesian Space and a radius are needed. The main computational advantage provided by capsules is represented by the fact that to compute the minimum distance between two capsules it is sufficient to compute the minimum distance between the two segments P_0-P_1 (that can be determined analytically) and then subtract the radius of each capsule, as it is shown in Figure 1. Moving back to our algorithm, every time a new edge is computed the "**CollisionCheck**" (see Algorithm 5) procedure is invoked to verify if the motion of the entire robot along the edge may cause collisions with the obstacles. For the sake of clarity, the "RobotCapsules" function is responsible for computing the capsule-based geometry model of the manipulator, see for instance Figures 3(a)-3(c).

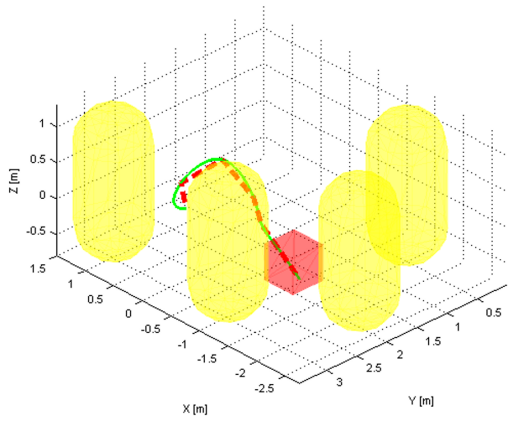
2.4 Trajectory Post Processing

As soon as the tree reaches the goal region, the sequence of nodes connecting the starting configuration to the final one is extracted from the tree and a three-stage post-processing procedure is performed:

1. **Path Shortening**: according to the algorithm proposed in [10], two distinct configuration are sampled along the trajectory. Then, the edge connecting the two sampled configurations is computed and checked for collisions. In case the edge is collision-free, the



(a) Path Shortening



(b) Spline Interpolation

Figure 2. Trajectory post-processing stages: original trajectory (dashed red), post-processed trajectory (solid green), obstacles (yellow capsules) and gaol region (red cube).

Algorithm 5 Collision checking procedure pseudo-code

```
1:  procedure CollisionCheck(edge, obstCaps)
2:      collFree ← 1
3:      for all q ∈ edge.Q
4:          robCaps ← RobotCapsules(q)
5:          for all rCap ∈ robCaps
6:              for all oCap ∈ obstCaps
7:                  minDist ← MinDistance(rCap, oCap)
8:                  if (minDist < 0)
9:                      collFree ← 0
10:                 end if
11:             end for
12:         end for
13:     end for
14:     return collFree
15: end procedure
```

trajectory is modified by inserting the new edge between the two sampled configuration. This procedure is repeated until a given number of iterations is reached. As a matter of fact, this path shortening procedure results in an a-posteriori optimization of the trajectory in the Cartesian Space. This approach can be thought as a convenient alternative with respect to tree optimization strategies normally exploited by optimal algorithms. An example of path shortening is shown in Figure 2(a);

2. **Spline Interpolation**: a series of evenly spaced configurations are selected along the trajectory to be used as via points for a cubic spline interpolation. In this way, sharp corners due to the connection of linear Cartesian-Space edges can be removed, ensuring that the trajectory belongs to the C^2 class of functions. An example of spline interpolation is shown in Figure 2(b);

3. **Kinematic Scaling**: up to this point the trajectory is computed taking into consideration a virtual time coordinate τ_v, ranging from 0 to 1. In this stage, joint positions are differentiated to find joint velocities and, in turn, joint accelerations. Then, a time-scaling parameter k_τ is computed to ensure that both joint velocities and accelerations do not exceed the imposed upper and lower bounds. Consequently, a real time coordinate τ_r can be computed as: $\tau_r = k_\tau \tau_v$.

3 Simulation Results

In order to validate the cart-RRT algorithm, a MATLAB implementation has been developed and a simulated UR10 manipulator has been considered as test bench. As far as actuation limits are concerned, we took into consideration the maximum and minimum joint velocities and accelerations displayed in Table 1. In order to resemble the operations that an ACM could possibly perform in an actual construction yard, a pick-and-place task involving a panel has been chosen. Then, the three simulation scenarios pictured in Figure 3 have been designed. More in depth:

- Scenario #1 (grasping panel "easy"), where the robot has to reach the goal region, while avoiding three obstacles. See Figure 3(a);
- Scenario #2 (grasping panel "difficult"), similar to Scenario #1, with an additional obstacle. See Figure 3(b);
- Scenario #3 (depositing panel), where the robot has grasped the panel and has to deposit it in the goal region. See Figure 3(c);

Table 1. UR10 Actuation Limits

Joint	$\dot{q}_{min}$ $[^\circ/s]$	$\dot{q}_{max}$ $[^\circ/s]$	$\ddot{q}_{min}$ $[^\circ/s^2]$	$\ddot{q}_{max}$ $[^\circ/s^2]$
1	-120	+120	-20	+20
2	-120	+120	-20	+20
3	-180	+180	-20	+20
4	-180	+180	-20	+20
5	-180	+180	-20	+20
6	-180	+180	-20	+20

For the sake of clarity, in each scenario the robot is represented in its starting configuration with links pictured with solid black lines and the corresponding capsules drawn in blue. On the other hand, capsules representing the obstacles are shown in yellow. The goal region is highlighted with a red cube and the panel is pictured with green lines and capsules. Finally, the Cartesian reference frames corresponding to the robot base, the robot end-effector and the Cartesian goal are pictured with red (X), green (Y) and blue (Z) dashed lines. Examples of possible solutions are shown in Figures 3(d)-3(f), thus demonstrating the effectiveness of the proposed algorithm. Furthermore, Figure 4(a) and 4(b) shows the joint velocity and acceleration profiles, respectively. These data clearly prove that the kinematic scaling procedure correctly scales the trajectory in order to always satisfy the aforementioned actuation bounds.

3.1 Comparative Evaluation

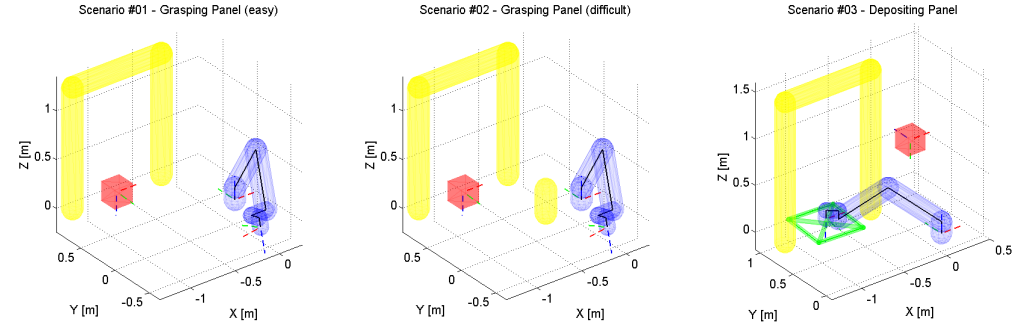
To properly assess the cart-RRT algorithm in terms of effectiveness, efficiency, and intuitiveness of the output, a comparative evaluation against state-of-the-art alternatives has been performed. More in depth, we took into account the following planning algorithms: RRT [7], RRT* [5] and SJRRT [6]. Similarly to cart-RRT, the selected state-of-the-art algorithms have been modified to return a failure if the maximum cardinality value (set to 500 nodes) is reached without finding a solution. For the sake of completeness, notice that SJRRT has been modified by removing the resolution of the kinematic redundancy, since the UR10 is not kinematically redundant. Five thousand simulations have been performed for each algorithm in each scenario in order to collect statistically significant datasets. In the following, the results of these simulations are presented and discussed. More in detail, Figures 5(a)-5(c) contain histograms that represent the average success rate of the different algorithms, with respect to each scenario. On the other hand, Figures 5(d)-5(f), Figures 5(g)-5(i), and Figures 5(j)-5(l) show box plots representing execution time, tree cardinality, and linear length of the output trajectory, respectively.

As regards the success rate, it is clear that cart-RRT and SJRRT perform consistently better than the others algorithms in all the considered scenarios. This situation is almost surely determined by the possibility to extend the random tree directly towards the goal. Moreover, it is worth noticing that RRT* was not able to provide solutions in scenario #3. Moving to the execution time, in Scenarios #1 and #3 cart-RRT and SJRRT are the fastest algorithms able to find a solution. On the other hand, in Scenario #2, the execution time of cart-RRT is significantly greater than the one of SJRRT and it is also greater than the one of RRT. Not surprisingly, RRT* is the slowest algorithm to converge in all the scenarios, since it optimizes the random tree at each iteration. Moving to tree cardinality, it is quite clear that cart-RRT and SJRRT are able to find solutions without having to reach the maximum allowed tree cardinality, as RRT normally does. As far as RRT* is concerned, the tree cardinality is always equal to the maximum, since the algorithm keeps adding nodes to the tree in order to optimize the output trajectory. Finally, if we take into consideration trajectory length, we can state that in Scenario #1 the outputs of cart-RRT, RRT* and SJRRT can be considered similar, while RRT finds longer trajectories. In Scenario #2, RRT* still provides very short trajectories, while cart-RRT performs worse than RRT, but better than SJRRT. In Scenario #3, finally, cart-RRT finds significantly shorter trajectories with respect to RRT and SJRRT.

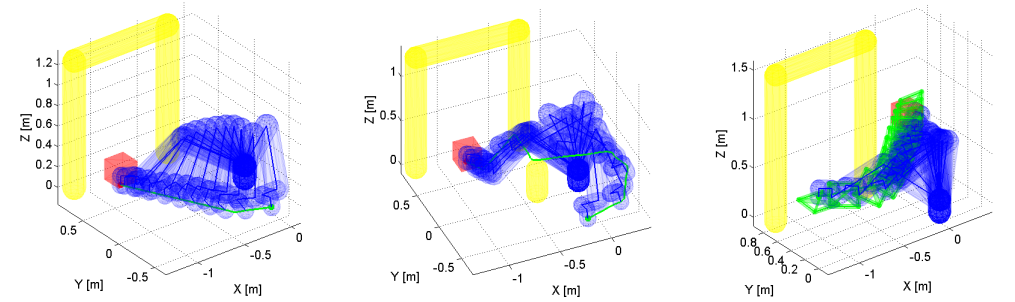
Finally, in order to evaluate and compare the intuitiveness of the trajectories computed by the different algorithms, a questionnaire has been designed and submitted to 15 workers. For each scenario, the interviewed workers were presented one example of output trajectory for each algorithm. Then, they were asked to rate the intuitiveness of each trajectory, i.e. its similarity with respect to the trajectory they would have followed. Interviewed workers were asked to rate each trajectory, by assigning a score between "0" (meaning very counterintuitive trajectory) and "5" (i.e. very intuitive trajectory). Results were aggregated and they are displayed in Figures 6, once again in the form of box plots. The comparison clearly demonstrate that the outputs of cart-RRT and RRT* (if present) are considered much more intuitive by the interviewed workers with respect to the outputs of RRT and SJRRT. To sum up, cart-RRT guarantees much better performances than RRT* in terms of success rate and execution time, while RRT* is able to provide shorter trajectories only in Scenario #2. Consequently, we can state that our algorithm represents a promising solution to the motion planning problem for ACMs sharing their workspace with human workers in actual construction yards.

4 Conclusions and Future Developments

In this work, an innovative Cartesian-Space randomized algorithm, named cart-RRT, is proposed. It consists in a modified version of RRT and it is able to provide more intu-

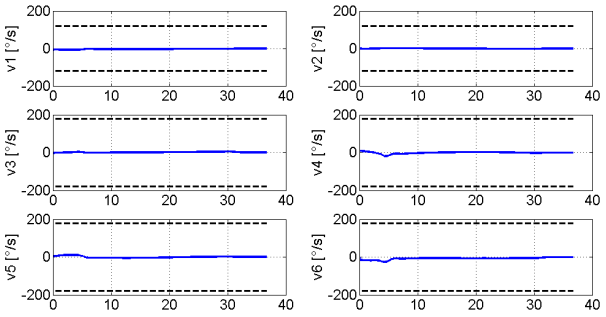


(a) Scenario #1 - grasping panel "easy" (b) Scenario #2 - grasping panel "difficult" (c) Scenario #3 - depositing panel

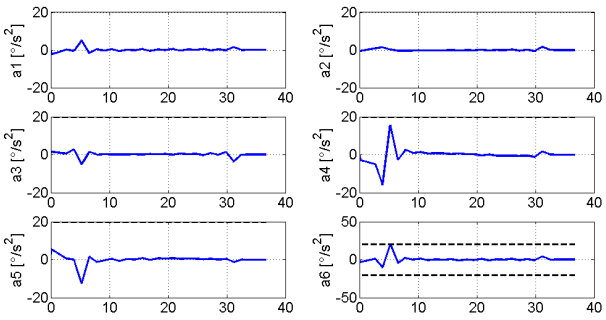


(d) Scenario #1 - grasping panel "easy" (e) Scenario #2 - grasping panel "difficult" (f) Scenario #3 - depositing panel

Figure 3. Simulation scenarios and examples of solutions.



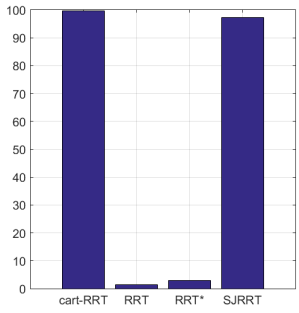
(a) Joint velocity profiles



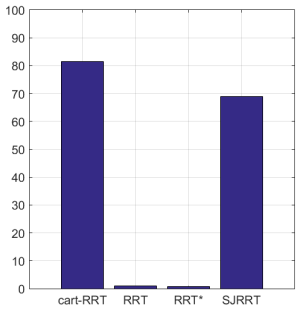
(b) Joint acceleration profiles

Figure 4. Example of solution for Scenario #1.

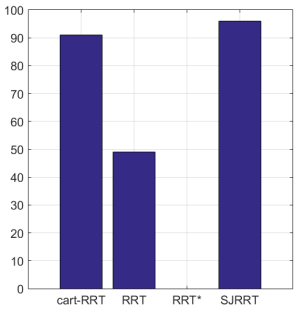
itive output trajectories thanks to two distinct features: the fact that nodes are sampled directly in the Cartesian-space, and the possibility to extend the tree directly towards the goal. Furthermore, the algorithm ensures that the output trajectories are collision-free and consistent with respect to actuation constraints. The effectiveness of the proposed approach is demonstrated in multiple simulated scenarios. Moreover, a comparative analysis of the results obtained by our algorithm with respect to the ones produced by other state-of-the-art planners is shown. In the end, not only we demonstrate that the efficiency of our algorithm is comparable with respect to state-of-the-art alternative, but we can also state that our algorithm is able to provide more intuitive trajectories with respect to state of the art planners, from the point of view of a human construction worker. At the moment, the most promising future development is represented by the possibility to extend this algorithm to the case of kinematically redundant ACMs. A study will be conducted to investigate how extra-DoFs can be used in order to improve optimality and safety (in terms of distance with respect to the obstacles) of the resulting trajectory.



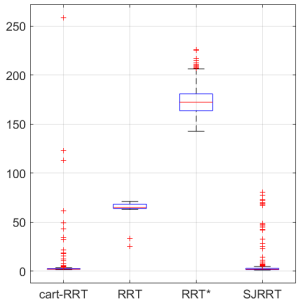
(a) Sc. #1 - Success Rate (%)



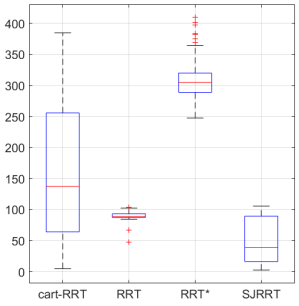
(b) Sc. #2 - Success Rate (%)



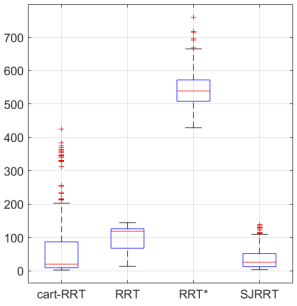
(c) Sc. #3 - Success Rate (%)



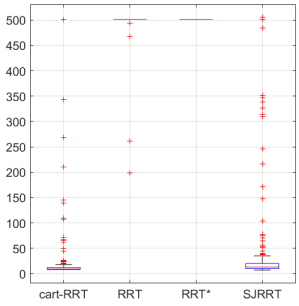
(d) Sc. #1 - Execution time (s)



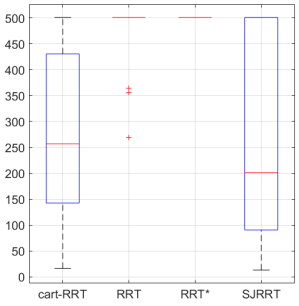
(e) Sc. #2 - Execution time (s)



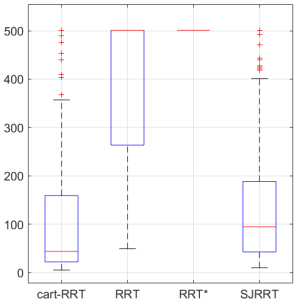
(f) Sc. #3 - Execution time (s)



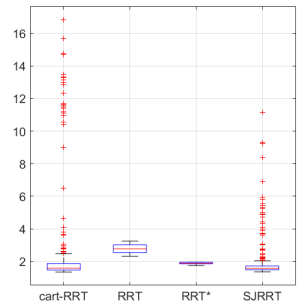
(g) Sc. #1 - Tree cardinality ()



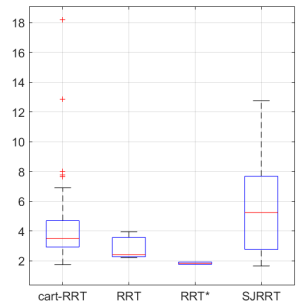
(h) Sc. #2 - Tree cardinality ()



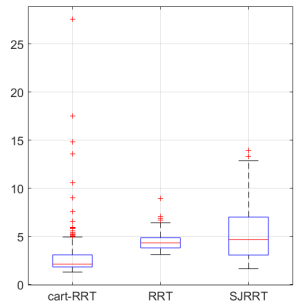
(i) Sc. #3 - Tree cardinality ()



(j) Sc. #1 - Trajectory lenght (m)

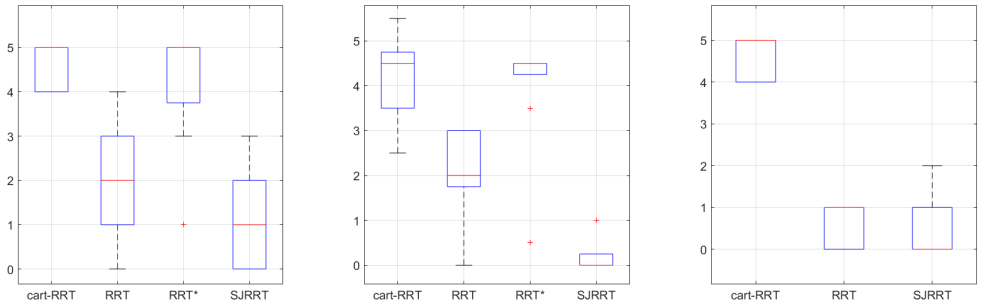


(k) Sc. #2 - Trajectory lenght (m)



(l) Sc. #3 - Trajectory lenght (m)

Figure 5. Comparative evaluation results



(a) Scenario #1 (b) Scenario #2 (c) Scenario #3

Figure 6. Evaluated intuitiveness of the output trajectory

References

[1] Jerome Barraquand, Lydia Kavraki, Jean-Claude Latombe, Rajeev Motwani, Tsai-Yen Li, and Prabhakar Raghavan. A random sampling scheme for path planning. The International Journal of Robotics Research, 16(6):759–774, 1997.

[2] M. S. Branicky, M. M. Curtiss, J. Levine, and S. Morgan. Sampling-based planning, control and verification of hybrid systems. IEE Proceedings - Control Theory and Applications, 153(5):575–590, Sept 2006.

[3] J. h. Jeon, S. Karaman, and E. Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the rrt*. In Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on, pages 3276–3282, Dec 2011.

[4] S. Karaman and E. Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In Robotics and Automation (ICRA), 2013 IEEE International Conference on, pages 5041–5047, May 2013.

[5] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. Int. J. Rob. Res., 30(7):846–894, June 2011.

[6] Bakir Lacevic and Paolo Rocco. Safety-oriented path planning for articulated robots. Robotica, 31(6):861âĂŞ874, Feb 2013.

[7] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. In Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, volume 1, pages 473–479 vol.1, 1999.

[8] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. The International Journal of Robotics Research, 20(5):378–400, 2001.

[9] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. The International Journal of Robotics Research, 28(8):933–945, 2009.

[10] College of Engineering Control Systems Laboratory University of Illinois at Urbana-Champaign. Rapidly exploring random tree (rrt) path planning.

[11] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In 2012 IEEE International Conference on Robotics and Automation, pages 2537–2542, May 2012.

[12] M. Ragaglia, M. Prandini, and L. Bascetta. Poli-rrt*: Optimal rrt-based planning for constrained and feedback linearisable vehicle dynamics. In Control Conference (ECC), 2015 European, pages 2521–2526, July 2015.

[13] M. Ragaglia, M. Prandini, and L. Bascetta. Multi-Agent Poli-RRT*, pages 261–270. Springer International Publishing, 2016.

[14] A. Tahirovic and G. Magnani. General framework for mobile robot navigation using passivity-based mpc. IEEE Transactions on Automatic Control, 56(1):184–190, Jan 2011.

[15] A. M. Zanchettin, L. Bascetta, and P. Rocco. Achieving humanlike motion: Resolving redundancy for anthropomorphic industrial manipulators. IEEE Robotics Automation Magazine, 20(4):131–138, Dec 2013.