Cartesian Constrained Stochastic Trajectory Optimization for Motion Planning

Michal Dobiš 1,*, Martin Dekan 1, Adam Sojka 1, Peter Beňo 2 and František Duchoň 1

1 Institute of Robotics and Cybernetics, Slovak University of Technology in Bratislava, 812 19 Bratislava, Slovakia; martin.dekan@stuba.sk (M.D.); adam.sojka@stuba.sk (A.S.);
frantisek.duchon@stuba.sk (F.D.)
2 Department of Robot Applications, Photoneo S.R.O. Company, 821 09 Bratislava, Slovakia;
beno@photoneo.com
* Correspondence: michal.dobis@stuba.sk

Abstract: This paper presents novel extensions of the Stochastic Optimization Motion Planning (STOMP), which considers cartesian path constraints. It potentially has high usage in many autonomous applications with robotic arms, where preservation or minimization of tool-point rotation is required. The original STOMP algorithm is unable to use the cartesian path constraints in a trajectory generation because it works only in robot joint space. Therefore, the designed solution, described in this paper, extends the most important parts of the algorithm to take into account cartesian constraints. The new sampling noise generator generates trajectory samples in cartesian space, while the new cost function evaluates them and minimizes traversed distance and rotation change of the tool-point in the resulting trajectory. These improvements are verified with simple experiments and the solution is compared with the original STOMP. Results of the experiments show that the implementation satisfies the cartesian constraints requirements.

Keywords: motion planning; cartesian constraints; robotic arms

1. Introduction

Currently, there are many path planners available that can solve the problem of path planning in a complicated configuration space of an industrial robotic arm. Commonly, the effective path planners that can fast-search any complicated configuration space are sampling-based, e.g., PRM [1], RRT [2] or their modifications (e.g., [3]), which even consider robot dynamics [4]. These algorithms are effective in searching for feasible paths, but they do not meet requirements of optimization principles. On the other hand, there are optimization-based algorithms (e.g., Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [5] and Energy-Optimal Collision-Free Motion Planning [6]) that combine energy consumption and collision avoidance constraints in an optimization task. These algorithms do not use randomized sampling and are not guaranteed to find a path for each case because there is a risk the planner will be trapped in a local minimum. An interesting solution is introduced by the Stochastic Optimization Motion Planning (STOMP) [7], which combines optimization-based methods and randomized generation of noised trajectories. The best candidate is chosen from these noised trajectories by optimization of the cost function.

The disadvantage of STOMP is that it works in robot joint space but some specific industrial applications need to find optimal solutions in cartesian space of robot TCP (tool center point). Imagine an industrial application in which a manipulated object is grasped by a vacuum or magnetic gripper. The grasp could have been weak, such as when using a two-finger gripper, and the object could have been dropped during the manipulation if the external forces are greater than the grasp force. This can occur with a specific tool-point orientation or position [8–11]. Likewise, many other applications...
exist where a minimization of tool-point rotation is required, e.g., applications in human environment \([12-15]\), liquid handling \([16]\), in milling and in the study of the cutter path \([17]\) and others \([18,19]\). The restrictions of specific tool-point orientations and positions on a robot trajectory are called cartesian path constraints.

Therefore, a new requirement for cartesian path constraints is necessary in the path planner. Authors of the GSTOMP algorithm \([20]\) tried to solve a similar problem. Their solution was based on a task-informed initialization of the STOMP algorithm by generation of initial trajectories using the Learning from Demonstration method \([21]\), with their custom cost function aiming to minimize the distance between collision-free trajectory and initial trajectory.

STOMP works in robot joint space, though the cartesian path constraints are not considered in the current version. However, STOMP is extendable because the implementation is based on plugins, which can be simply switched. Therefore, the idea of our designed solution is to use cartesian space instead of joint space in parts of the original algorithm, where it is required. The most potentially beneficial plugins could be the noise generator and cost function. In the cartesian space it is much easier to generate good trajectory candidates that satisfy cartesian constraints of the robot TCP. Likewise, the important part is the cost function, which can evaluate these generated trajectory candidates in respect of cartesian constraints. The STOMP version that uses these plugins is called cartesian constrained STOMP. In contrast to GSTOMP, our research provides a solution that does not require the task-informed initialization and is usable without any third-party algorithms. The main contribution of our research are extensions of STOMP that allow the usage of cartesian constraints. The new plugins, being the sampling generator and cost function, are briefly described in the Section 2. The Section 3 focuses on the description of our proposed implementation and solution for path planning under cartesian constraints. Our research is verified with experiments in the Section 4. The first experiment verifies the satisfaction of required cartesian constraints and the second compares computation time and the quality of resulting trajectories with outputs from the original algorithm.

2. STOMP Algorithm

The STOMP algorithm mainly works in joint space; therefore, the disadvantage of the whole algorithm is that it is unable to consider any cartesian constraints in trajectory generation and it is possible only to validate constraints on a result. This research analyzes opportunities of cartesian path constraints implementation, with this section briefly describing the most important parts of the STOMP algorithm.

In the first step of STOMP, the initialization method is executed, creating a deterministic trajectory (Figure 1). The simplest case is the linear interpolation in joint space. Another option is to use a custom initial trajectory. If the initial trajectory is collision free, it can be considered to be a result of the algorithm.

![Figure 1](image.png)

**Figure 1.** An example of initialization trajectory (red line) and rollouts (grey lines). The grey polygon between the start and the goal states represents an obstacle (Reprinted Ref. [22]).

When the initialization trajectory is not collision free, the algorithm continues into a first iteration in which \(n\) different trajectories are generated. These trajectories are called rollouts (Figure 1). The next important STOMP term is timestep, which represents robot
state or waypoint in the trajectory. STOMP works with a constant number of timesteps. Rollouts are created by a noise generator, which generates noise around the initialization trajectory in each timestep. The basic noise generator uses Normal Distribution Sampling. This method uses a parameter standard deviation (stddev) that represents an amplitude of the noise applied to each joint in the rollout. Using larger values will produce larger motion for the joints. In the context of STOMP, each joint state of a trajectory is called timestep and STOMP works with a constant number of timesteps.

The next step is to evaluate rollouts adding cost values by cost functions (STOMP can use more functions). If more rollouts are valid, the resulting trajectory is rollout, which has the lowest cost (Figure 2). On the other hand, if no rollout is valid (e.g., the robot state is in collision with an obstacle), then the rollout with the lowest cost is chosen to the next iteration, where a new noise is generated around this chosen rollout. One of the standard STOMP cost functions is Collision Check, which validates if the rollout is collision free and the cost is computed by the kernel smoothing.

![Figure 2. An example of valid trajectory (blue line), initialization trajectory (red line) and rollouts (grey lines) (Reprinted Ref. [22]).](image)

STOMP also uses other methods for preprocessing and postprocessing of trajectories, including Update Filters and Noisy Filters (further details are discussed in [7,23]).

### 3. Cartesian Constrained STOMP

The cartesian constrained planning in STOMP is based on the new noise generator and cost function. The newly designed noise generator, Cartesian Normal Distribution Generator, creates noise in cartesian space as a good estimation of the satisfaction of constraints. The decision on whether the constraints are satisfied is made by the new cost function, Cartesian Distance.

#### 3.1. Noise Generator

The standard noise generators Normal Distribution Sampling and Goal Guided Multivariate Gaussian work in robot joint space, but they should be generated in cartesian space due to requirements of cartesian constraints. Therefore, in this article, Cartesian Normal Distribution Generator is introduced.

The implementation is based on the virtual plugin method `generateNoise`, which uses noiseless trajectory input. An output is noise and has noisy trajectory. Since all parameters are in joint space, a conversion between joint space and cartesian space is needed using forward and inverse kinematics. The generator works as follows:

1. Convert each timestep of noiseless joint input trajectory into cartesian waypoints using forward kinematics;
2. Each waypoint is noised by a random generator in the interval of stddev parameters by (1);
3. New timestep is computed by inverse kinematics from each generated waypoint;
4. If inverse kinematics for a particular waypoint has a solution, the new computed timestep is added to the resulting trajectory; otherwise, noiseless timestep from input trajectory is added to the resulting trajectory.

The following equation calculates the noised waypoint position:

\[ P = P' \ast T_{xyz}(r_1, r_2, r_3) \ast R_x(r_4) \ast R_y(r_5) \ast R_z(r_6) \]  

(1)

where \( P' \) is a \( 4 \times 4 \) transformation matrix (from robot base link to robot TCP) of noiseless waypoint and \( T_{xyz} \) is a \( 4 \times 4 \) translation matrix, where the rotation part is the identity matrix and the translation part is variable. \( R_x, R_y \) and \( R_z \) are \( 4 \times 4 \) rotation matrices, where the rotation part is variable and the translation part is zero vector. Values \( r_1, r_2, r_3, r_4, r_5 \) and \( r_6 \) are random numbers with amplitudes defined by parameter stddev.

Figure 3 shows the space around a waypoint in which a new noised waypoint could be created. This space is user-defined by the parameter stddev. It consists of six values that represent the amplitude of the translation noise on axes \( x, y \) and \( z \) and rotation noise on axes \( x, y \) and \( z \). For example, parameter values \((0.1, 0.1, 0.5, 0.7, 0.7, 1.5)\) mean that the TCP could be translated up to \( 0.1 \) m on axes \( x \) and \( y \) and \( 0.5 \) m on axis \( z \), and rotated up to \( 0.7 \) rad around \( x \) and \( y \) axes and \( 1.5 \) rad around \( z \) axis in one iteration. Higher values could explore greater space, but there is a risk that generated noise will be out of required cartesian constraints. Cartesian Normal Distribution Generator does not guarantee that the new rollout satisfies constraints. The first reason is that a user could set stddev parameter values higher than the allowed constraints. The second problem is that in the next iterations different rollouts are used, which could be outside the tolerance.

**Figure 3.** An example of cartesian space around a waypoint. (Blue dots represent waypoints and red dots represent start and goal).

### 3.2. Cost Function

Due to previously outlined reasons, a definition of the new cost function is needed. Therefore, a Cartesian Distance Cost Function is designed in our research. The goal is to reach the closest approximation to cartesian linear trajectory, which consists of \( n \) waypoints. The change of position and orientation of TCP is linear in each waypoint and is illustrated in Figure 4. The method calculates the distance and rotation angle between waypoints of
cartesian linear trajectory and waypoints rollout in each timestep. The count of cartesian linear trajectory waypoints is the same as the count of rollout waypoints for a simpler comparison of both trajectories.

Figure 4. An example of cartesian linear trajectory.

Rollouts with a calculated distance or rotation angle outside of user-defined tolerance are invalid. The optimization goal of this cost function is to minimize the distance and rotation angle in each STOMP iteration. Figure 5 shows the generated rollout (green curve with blue dots representing timesteps) and optimal linear trajectory (yellow line). The grey arrow illustrates the distance that should be minimized. Figure 5 displays the ideal linear trajectory and one generated rollout where it is possible to infer distances between individual tool-point poses. These are used in the cost function which is considered by the cartesian constraints.

Figure 5. Ideal linear trajectory (yellow) and one rollout (green) with each tool-point pose visualized with a frame of reference.

The first step of the cost function is to convert rollout timesteps from joint space to cartesian space of robot TCP. The transformation matrices between each waypoint of cartesian linear trajectory and rollout are calculated by the formula:

\[ \text{Diff}_i = P_{Li}^{-1} * P_{Ri} \]  

(2)

where \( P_{Li} \) is i-waypoint of cartesian linear trajectory and \( P_{Ri} \) is the i-timestep of rollout. The evaluation strategy is split into translation and rotation errors. The translation error is
denoted as $e_{ti}$ which is the Euclidean norm of the translation vector of matrix $Diff_i$. The variable $e_{ri}$ represents rotation error and is calculated as the angle of the rotation vector, which is created from the rotation part of matrix $Diff_i$ [24]. The rollout is invalid when the translation or rotation error exceeds tolerances $t_l$ and $t_r$. The final cost is given by the formula:

$$C = \sum_{j=0}^{n} w_t \frac{e_{ti}}{t_l} + w_r \frac{e_{ri}}{t_r}$$  \hspace{1cm} (3)$$

where $n$ is the number of timesteps (waypoints), with $e_{ti}$ and $e_{ri}$ obtained from the transformation matrix between the i-waypoint of cartesian linear trajectory and rollout. Parameters $t_l$ and $t_r$ work in this function as normalizers and parameters $w_t$ and $w_r$ represent cost weights. These tolerances and cost weights are parameterizable (recapitulated in Table 1).

Table 1. Parameters of Cartesian Distance Cost Function.

| Parameter | Description |
|-----------|-------------|
| $w_t$     | The position cost weight used in the cost function |
| $w_r$     | The orientation cost weight used in the cost function |
| $t_l$     | The maximal translation between waypoint of cartesian linear trajectory and rollout |
| $t_r$     | The maximal angle between waypoint of cartesian linear trajectory and rollout |

4. Results

The new STOMP plugins (Cartesian Normal Distribution Sampling Generator and Cartesian Distance Cost Function) were verified and evaluated on a simple experimental setup, which consisted of one obstacle between the start and the goal states (Figure 6). Since that trajectory generation is randomized, the resulting trajectory between these two robot states could be different in each planning attempt. Therefore, path planning was triggered multiple times in each experiment and provided a variable dataset of resulting trajectories.

![Figure 6. Experimental setup and examples of a resulting trajectory: (a) A trajectory computed by the basic STOMP which does not satisfy cartesian constraints requirements; (b) A trajectory computed by the cartesian constrained STOMP which demonstrates capability of planning under desired cartesian constraints.](image)

The configuration of basic STOMP parameters was based on our experience and need to ensure that a collision-free trajectory would be found in each path-planning attempt. Therefore, the algorithm iterated up to 100 times, which represented enough iterations, and was stopped when the first valid solution was found. The important parameter was the number of rollouts. A value smaller than that used can result in no solution being found, while a higher value can slow down the iteration. Therefore, 15 rollouts were chosen as a suitable number. Each rollout consisted of 10 timesteps that were appropriate for the trajectory length and the environment complexity chosen for experiments presented in this
The configuration of basic STOMP parameters was based on our experience and need. The experiment dataset consisted of 3000 different resulting trajectories between the same start and the same goal state, which were the output from cartesian constrained STOMP. Figure 6 shows the difference between trajectories generated by the original and the cartesian-constrained STOMP, where Figure 6a is the trajectory from the original STOMP with Normal Distribution Sampling Generator and Figure 6b is the trajectory from the new Cartesian Normal Distribution Sampling Generator with Cartesian Distance Cost Function, which satisfies the cartesian constraints.

### 4.1. Cartesian Constraints Verification

The first experiment validated tolerance keeping, with the goal being to ensure that any waypoint of a resulting trajectory did not exceed the maximal allowed translation distance and rotation angle. The translation tolerance was set to 1.0 m and rotation tolerance to 0.2 rad. The parameter stddev in the Cartesian Normal Distribution Sampling was (0.5, 0.5, 0.1, 0.1, 0.1), so that the amplitude of translation noise was 0.5 m on each axis and the rotation amplitude was 0.1 rad around each axis. This configuration has rather disproportional sampling amplitude compared to the tolerance as it is able to generate a configuration outside the desired tolerance in a single step. This was selected to ensure that the constraint worked even under such harsh conditions.

The experiment dataset consisted of 3000 different resulting trajectories between the same start and the same goal state, which were the output from cartesian constrained STOMP. Figure 7 shows the differences between waypoints of an ideal cartesian linear trajectory and waypoints of resulting trajectories. The histogram in Figure 7a shows translation distance and the histogram in Figure 7b shows rotation angle, which represents the rotation difference between these waypoints. The histograms and results satisfied our expectation because no waypoint of resulting trajectories exceeded a translation tolerance of 1.0 m and rotation tolerance of 0.2 rad. Nevertheless, noise amplitudes were relatively high and tolerances were low, with all resulting trajectories valid.

![Figure 7](image.png)

**Figure 7.** Histograms of differences between waypoints of cartesian linear trajectory and waypoints of resulting trajectories: (a) Translation distance; (b) Rotation angle.

### 4.2. Comparison of Noise Generators and Cost Functions

The goal of the second experiment was to compare configurations of original and cartesian constrained STOMP. The experiment was be executed in four different configurations, where Cartesian and Joint Normal Distribution Sampling Generator were switched. Likewise, Cartesian Distance Cost Function was activated in two configurations and deactivated.
in two configurations. Each configuration had a similar dataset as the first experiment, but there were 1000 computed resulting trajectories. The configurations were as follows:

1. Joint Normal Distribution Sampling Generator without Cartesian Distance Cost Function (green bars in Figure 8)—this represented the original STOMP and it was not expected that cartesian constraints would be satisfied.
2. Joint Normal Distribution Sampling Generator with Cartesian Distance Cost Function (red bars in Figure 8)—this represented the original STOMP, which was extended with Cartesian Distance Cost Function. It was expected that cartesian constraints would be satisfied.
3. Cartesian Normal Distribution Sampling Generator without Cartesian Distance Cost Function (blue bars in Figure 8)—this replaced the original noise generator in joint space with the noise generator in the cartesian space, but there was still a risk that cartesian constraints would not be satisfied.
4. Cartesian Normal Distribution Sampling Generator with Cartesian Distance Cost Function (orange bars in Figure 8)—the combinations of both newly implemented plugins. It used the noise generator in the cartesian space with Cartesian Distance Cost Function and cartesian constraints that should be satisfied.

For the cartesian noise generator, the amplitudes of noise were (0.2, 0.2, 0.5, 0.05, 0.05), while for the Normal Distribution Sampling Generator, the amplitudes were (0.04, 0.08, 0.1, 0.07, 0.1, 0.10). Cartesian distance cost function used a rotation tolerance of 0.4 rad and translation tolerance of 2.5 m. The cost weight was the same as the weight of the CollisionCheck cost function. This configuration of noise and tolerances was different to the first experiment, 4.1 Cartesian Constraints Verification, where the configuration was...
relatively extreme. There was a risk that generated rollouts were often out of tolerances, which increased computation time unnecessarily. The goal of the second experiment was to test functionality in situations that were close to real robotic applications; therefore, noise amplitudes were lower and tolerances were increased.

The comparison focused on four criteria, which consisted of computation time, the accumulated sum of cartesian distances of tool-points between waypoints (4) and the accumulated differences of tool-points orientations between waypoints (5) [25].

\[
x = \sum_{i=2}^{n} \| p_i - p_{i-1} \| \\
\]

where \( p_i \) is robot tool-point position, \( i \) is the waypoint id of \( n \) waypoints and the score \( x \) is assigned to each trajectory.

\[
x = \sum_{i=2}^{n} \cos^{-1} (q_{xi} * q_{xi-1} + q_{yi} * q_{yi-1} + q_{zi} * q_{zi-1} + q_{wi} * q_{wi-1})
\]

where \( q_x, q_y, q_z \) and \( q_w \) are values of tool-point quaternion of \( i \) waypoint and the score \( x \) is assigned to each trajectory.

Table 2 summarizes the averages and variances of evaluation criteria scores, with the histograms in Figure 8 providing a results comparison. The evaluation of tool-point orientation shows that the Normal Distribution Sampling Generator, which works only in robot joint space, cannot satisfy the orientation constraints as was expected. However, using the Cartesian Distance Cost Function kept STOMP rollouts within the tolerance of constraints. The combination of Cartesian Normal Distribution Sampling Generator and Cartesian Cost Function provided the best result for this criterion, but without the Cartesian Cost Function, some trajectories were outside the allowed tolerance. This was mentioned in Section 3.1 Noise Generator and is shown in Figure 8c,d.

| Table 2. Summary of results’ averages and variances. |
| --- | --- | --- | --- |
| | Cartesian Noise | | Joint Noise |
| | Avg | Var | Avg | Var | Avg | Var |
| --- | --- | --- | --- | --- | --- | --- |
| Cartesian Distance (m) | 2.4668 | 0.5871 | 2.035 | 0.244 | 1.6482 | 0.1728 |
| Tool Orientation (rad) | 0.4750 | 0.0052 | 0.383 | 0.001 | 1.5697 | 0.3733 |
| Computation Time (s) | 0.2267 | 0.0054 | 0.656 | 0.058 | 0.135 | 0.0031 |

Figure 8a shows that the Joint Normal Distance Generator is better in relation to the accumulated sum of cartesian distance between waypoints. This unexpected behavior was caused by the parameter stddev in Cartesian Normal Distribution Sampling generating samples in a larger range than stddev in Joint Normal Distribution Sampling.

The STOMP algorithm finds solutions faster without using Cartesian Distance Cost Function. However, this was expected because a set of possible solutions is smaller and a planning request is complicated using the Cartesian Distance Cost Function (Figure 8b).

5. Discussion

In this paper, we proposed two extensions for the widely used STOMP algorithm. These extensions enable STOMP to be used as a cartesian space planner. This will allow it to be used in a variety of applications such as liquid handling or ones that use a vacuum gripper.

Both extensions, Cartesian Normal Distribution Sampling Generator and the Cartesian Distance Cost Function, were tested in an environment that was comparable to common industrial applications such as pick and place or bin picking. The results were compared
with the standard STOMP algorithm. Both experiments showed that these modifications led to desirable results. However, it should be noted that in complex environments where the STOMP algorithm does not find a suitable trajectory with 100% success rate, our extensions lower the success rate even further [25]. This is caused by a further limitation of the possible state space. Compared to STOMP modifications such as GSTOMP, our extensions are based on STOMP framework, so there is no need to use other third-party libraries that allow easier use. Another limitation of our proposed extensions is the increase in computational time needed to find a suitable trajectory.

We hope to address these issues in our future work by optimizing the sampling generator, which will lead to fewer needed sample-state generations, thus lowering computational time. Additionally, we want to compare our extension of STOMP with path-planning algorithms such as RRT, PRM, CHOMP, etc.

6. Conclusions

The STOMP algorithm is an optimization-based path planner that uses random sampling of noised trajectories. The algorithm works in robot joint space and, therefore, it cannot consider any cartesian path constraints. The cartesian constraints are often required in many industrial applications. Our research was focused on extension of the STOMP algorithm, which will be able to do path planning under cartesian constraints. We designed and implemented the new Cartesian Normal Distribution Sampling Generator and the Cartesian Distance Cost Function, which work in cartesian space of a robot TCP to satisfy cartesian constraints. This implementation is available in supplementary materials https://github.com/photoneo/stomp_ros/tree/MichalD/cartesian_constraints_sampling (accessed on 5 November 2021) or online at [26].

Original noise generators in STOMP generate trajectory rollouts in robot joint space. Our Cartesian Normal Distribution Sampling Generator generates tool-point positions in cartesian space. The trajectory rollout, which is generated under cartesian constraints, could be a better candidate as a generated rollout in joint space without consideration of any constraints.

Due to STOMP being an optimization-based path planner, we defined the new Cartesian Distance Cost Function which is able to validate each trajectory rollout. This cost function tries to choose rollouts that are the most similar to a linear trajectory of tool-point in cartesian space.

In our simple experiments, STOMP tried to find collision-free trajectory under cartesian constraints between the same start state and the same goal state. The first experiment proved that the cartesian constrained STOMP satisfied cartesian path constraints in each resulting trajectory. The second experiment compared configurations of original and newly created plugins of STOMP with respect to the computation time, accumulated sum of tool-point orientation changes and cartesian distances between waypoints. The experiment showed that the accumulated sum of tool-point orientation was minimized using the Cartesian Normal Distribution Sampling Generator and the Cartesian Distance Cost Function. On the other hand, the computation time was increased using Cartesian Distance Cost Function. However, this increase was expected because the planning problem under cartesian constraints is more difficult than without constraints.

Our future work will focus on reducing computation time. It is possible to improve Cartesian Normal Distribution Sampling, which will generate better candidates and a STOMP algorithm capable of finding trajectory after less iterations. The other goal will be to evaluate STOMP plugin extensions on a real robot or a real industrial application.

Supplementary Materials: The following are available online at https://github.com/photoneo/stomp_ros/tree/MichalD/cartesian_constraints_sampling (accessed on 5 November 2021).

Author Contributions: Conceptualization, P.B. and M.D. (Michal Dobiš); methodology, P.B. and M.D. (Michal Dobiš); software, M.D. (Michal Dobiš); validation, M.D. (Martin Dekan) and A.S.; formal analysis, A.S.; investigation, M.D. (Michal Dobiš); resources, P.B.; data curation, M.D. (Michal Do-
biš); writing—original draft preparation, M.D. (Michal Dobiš); writing—review and editing, M.D. (Martin Dekan) and A.S.; visualization, M.D. (Michal Dobiš); supervision, P.B.; project administration, P.B.; funding acquisition, F.D. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by VEGA, grant number VEGA 1/0775/20; by the European Regional Development Fund, grant number 313012P386; and by Horizon 2020, grant number 824964.

**Institutional Review Board Statement:** Not applicable.

**Data Availability Statement:** The following are available online at https://github.com/photoneo/stomp_ros/tree/MichalD/cartesian_constraints_sampling (accessed on 5 November 2021).

**Acknowledgments:** This research was partially sponsored by Photoneo company. ([http://photoneo.com](http://photoneo.com)). VEGA 1/0775/20, APVV-17-0214 and DIH’2 supported this work. Operational Program Integrated Infrastructure created this publication for the project: “Robotic workplace for intelligent welding of small-scale production,” code ITMS2014+: 313012P386, and it is co-sponsored by the European Regional Development Fund.

**Conflicts of Interest:** The authors declare no conflict of interest.

**References**

1. Kavraki, L.E.; Svestka, P.; Latombe, J.-C.; Overmars, M.H. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Autom.* 1996, 12, 566–580. [CrossRef]

2. Lavalle, S.M. Rapidly-Exploring Random Trees: A New Tool for Path Planning. 1998. Available online: https://www.cs.csustan.edu/~sjiang/Courses/CS4710-215/Papers/06%20RRT.pdf (accessed on 26 November 2021).

3. Hu, B.; Cao, Z.; Zhou, M. An Efficient RRT-Based Framework for Planning Short and Smooth Wheeled Robot Motion under Kinodynamic Constraints. *IEEE Trans. Ind. Electron.* 2020, 68, 3292–3302. [CrossRef]

4. Stanko, J.; Rodina, J.; Hubinsky, P. Comparison of Approaches for UAV Dynamics Consideration in Sampling Based Path Planning Methods. In Proceedings of the 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR), Budapest, Hungary, 15–17 October 2020; pp. 1–5. [CrossRef]

5. Zucker, M.; Ratliff, N.; Bagnell, J.A.; Srinivasa, S.S. CHOMP: Covariant Hamiltonian optimization for motion planning. *Int. J. Robot. Res.* 2013, 32, 1164–1193. [CrossRef]

6. Zhao, Y.; Wang, Y.; Zhou, M.; Wu, J. Energy-Optimal Collision-Free Motion Planning for Multiaxis Motion Systems: An Alternating Quadratic Programming Approach. *IEEE Trans. Autom. Sci. Eng.* 2019, 16, 327–338. [CrossRef]

7. Kalakrishnan, M.; Chitta, S.; Theodorou, E.; Pastor, P.; Schaal, S. STOMP: Stochastic trajectory optimization for motion planning. In Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 9–13 May 2011. [CrossRef]

8. Bicchi, A.; Kumar, V. Robotic grasping and contact: A review. In Proceedings of the 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation, San Francisco, CA, USA, 24–28 April 2000; pp. 348–353. [CrossRef]

9. Fantoni, G.; Santochi, M.; Dini, G.; Tracht, K.; Scholz-Reiter, B.; Fleischer, J.; Kristoffer Lien, T.; Seliger, G.; Reinhart, G.; Franke, J.; et al. Grasping devices and methods in automated production processes. *CIRP Ann.* 2014, 63, 679–701. [CrossRef]

10. Sukop, M.; Tuleja, P.; Janos, R.; Juras, O.; Marcinko, P.; Semjon, J.; Vagas, M. Using the Vacuum in Handling Tasks in the Context of Operating Cost Savings. *J. Autom. Control* 2017, 5, 85–88. [CrossRef]

11. Björnsson, A.; Lindback, J.-E.; Eklund, D.; Jonsson, M. Low-cost Automation for Prepreg Handling—Two Cases from the Aerospace Industry. *SAE Int. J. Mater. Manuf.* 2016, 9, 68–74. [CrossRef]

12. Tölgyessy, M.; Dekan, M.; DUCHOŦ, F.; Rodina, J.; Hubinský, P.; Chovanec, L. Foundations of Visual Linear Human–Robot Interaction via Pointing Gesture Navigation. *Int. J. Soc. Robot.* 2017, 9, 509–523. [CrossRef]

13. Thakur, N.; Han, C.Y. An Ambient Intelligence-Based Human Behavior Monitoring Framework for Ubiquitous Environments. *Information* 2021, 12, 81. [CrossRef]

14. Su, H.; Yang, C.; Ferrigno, G.; De Momi, E. Improved Human–Robot Collaborative Control of Redundant Robot for Teleoperated Minimally Invasive Surgery. *IEEE Robot. Autom. Lett.* 2019, 4, 1447–1453. [CrossRef]

15. Qi, W.; Ovr, S.E.; Li, Z.; Marzullo, A.; Song, R. Multi-Sensor Guided Hand Gesture Recognition for a Teleoperated Robot Using a Recurrent Neural Network. *IEEE Robot. Autom. Lett.* 2021, 6, 6039–6045. [CrossRef]

16. Kong, F.; Yuan, L.; Zheng, Y.P.; Chen, W. Automatic Liquid Handling for Life Science. *J. Lab. Autom.* 2012, 17, 169–185. [CrossRef] [PubMed]

17. Toh, C.K. A study of the effects of cutter path strategies and orientations in milling. *J. Mater. Process. Technol.* 2004, 152, 346–356. [CrossRef]

18. Nemec, D.; Gregor, M.; Bubeníková, E.; Hruboš, M.; Pírník, R. Improving the Hybrid A* method for a non-holonomic wheeled robot. *Int. J. Adv. Robot. Syst.* 2019, 16, 1729881419826857. [CrossRef]
19. Uríček, J.; Poppeová, V.; Zahoranský, R.; Bulej, V.; Kuciak, J.; Durec, P. The potential fields application for mobile robots path planning. Trends in the development of machinery and associated technology TMT 2008. In Proceedings of the 12th international research/expert conference, Istanbul, Turkey, 26–30 August 2008; pp. 505–508.

20. Magyar, B.; Tsiogkas, N.; Brito, B.; Patel, M.; Lane, D.; Wang, S. Guided Stochastic Optimization for Motion Planning. *Front. Robot. AI* 2019, 6, 105. [CrossRef] [PubMed]

21. Asif, M.; Mukadam, M.; Ahmadzadeh, S.; Chernova, S.; Boots, B. Towards Robust Skill Generalization: Unifying Learning from Demonstration and Motion Planning; Conference on Robot Learning: Mountain View, CA, USA, 2017; pp. 109–118.

22. Dobiš, M.; Dekan, M.; Beňo, P.; Duchaň, F.; Kohút, M. The Comparison of Motion Planners for Robotic Arms. *J. Control. Eng. Appl. Inform.* 2021, 23, 87–94.

23. MoveIt! STOMP Planner Tutorial. Available online: http://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/stomp_planner/stomp_planner_tutorial.html (accessed on 31 October 2021).

24. Diebel, J. Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix* 2006, 58, 1–35.

25. Lavalle, S.M. Sampling-Based Motion Planning. In *Planning Algorithms*; Cambridge University Press: New York, NY, USA, 2014; pp. 153–206.

26. Dobiš, M.; Dekan, M.; Beňo, P.; Sojka, A.; Duchaň, F. The Cartesian Constrained STOMP (The Supplementary Material). Available online: https://github.com/photoneo/stomp_ros/tree/MichalD/cartesian_constraints_sampling (accessed on 5 November 2021).