A Comparison of Motion Planners for Robotic Arms

M. Dobiš*, M. Dekan*, F. Duchoň*, P. Beňo, M. Kohút***

**Institute of Robotics and Cybernetics, Slovak University of Technology in Bratislava, Slovakia (e-mail: michal.dobis@stuba.sk, martin.dekan@stuba.sk, frantisek.duchon@stuba.sk, miroslav.kohut@stuba.sk)*
***Department of Robot Applications, Photoneo s.r.o. company, Slovakia (e-mail: beno@photoneo.com)*

Abstract: This paper presents a comparison of two motion planning approaches for industrial manipulators. The first approach implements sampling-based methods and the second approach uses optimization-based methods. Specific algorithms compared in this article are RRT, RRTConnect (sampling-based motion planning) and STOMP (optimization-based motion planning). Both approaches are able to find collision-free trajectory and are available for Robotic Operating System via MoveIt package. In our experiments computation time, success rate and trajectory length are evaluated. Our results show that RRT and RRTConnect are faster, but a risk exists, that the length of trajectory will be extremely long. On the other hand, STOMP is slower, but the length of found trajectory depends on parametrization.

Keywords: Path Planning, Collision Avoidance, Sampling-based Methods, Optimization-based Methods, robotic arms.

1. INTRODUCTION

Industrial manipulators are often used for object handling and manipulation in the manufacturing industry. Traditionally, movements of the robot are taught by the programmer and the motions are repeatedly executed for each object. The robot programmer iteratively defines each movement from source to destination, ensuring that defined trajectory is fast to execute, collision-free and respects wear of mechanical parts of the robot. This is possible when the picked object has a fixed grasping position. This often requires the usage of mechanical fixtures, human workers and specialized gripper design.

Modern manufacturing lines are, however, optimized for flexibility. It becomes increasingly important to create manufacturing setups that can adapt to changes in production faster, thus saving costs of manual reconfiguration. Furthermore, more processing power and modern types of sensors are available on the industrial market. New 3D vision systems allow us to precisely recover object positions and transform them into robot cartesian space. To move such a dynamic object, the robot has to be able to compute its motion to pick the object dynamically. Furthermore, robot trajectories have to be collision-free, thus environment where the motion takes place has to be known.

Algorithms, which solve this problem are called path planners. The task of the path planner is to find a sequence of valid configurations that moves the robot from source to destination.

The configuration space of 6-DOF robotic arm is 6-dimensional, which is too large for explicit calculation, and usage of analytic and deterministic algorithms is very difficult (Lindemann and LaValle, 2005). The configuration space and disadvantage of explicit construction are simple described in the next section.

The main goal of the article is to compare two approaches commonly used by Robotic Operating System (ROS) community. One of them are **sampling-based** motion planners. These planners use and create topological trees and graphs. Second presented type of motion planner are **optimization-based** methods, which minimize cost function. The Sampling-based motion planners from Open Motion Planning Library (**OMPL**), and the optimization-based planner, **STOMP** will be described and compared in this article. For the comparison we devised two experimental setups on an environment, which simulates a production cell. The first experiment simulates pick of randomly placed parts in the bin and the second simulates placing. The compared algorithms are RRT and RRTConnect from the OMPL and the STOMP with specific parameters. Both algorithms are free available in ROS melodic MoveIt.

2. CONFIGURATION SPACE

The configuration space C is a set of all possible states of the robot. The set includes states, which are collision-free, whether in self-collision or in collision with another obstacle. The set of collision-free states is C_{free} and the set of states in collision is C_{obs}. (Lozano-Peréz, 1990)

The configuration space can be sampled into a grid map, and then the solution can be found in the grid map by a deterministic algorithm like Dijkstra or A*. The fig. 2 illustrates the 2-axis robotic arm and the orange circle represents an obstacle. On the right the configuration space is shown. The white space represents collision free states, and

the orange space represents robot states, which are in collision.

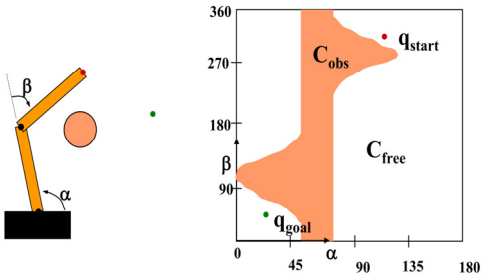

Fig. 1. Configuration space, C_{free} and C_{obs} for an articulated robot with two joints (Gasparetto et al., 2015).

Actually, the explicit definition of C_{obs} is a computationally difficult problem, because computational complexity increases exponentially with the dimension of C. (Lindemann and LaValle, 2005)

3.OPEN MOTION PLANNING LIBRARY

The OMPL library contains sampling-based motion planners, which are divided into two basic groups of algorithms – single-query and multi-query (Sucan et al., 2012).

Both groups of path planners need to define and implement fundamental parts of the algorithm:

1. Sampling method (Normal distribution, Gauss distribution, or other specified methods) (Booret al., 1999)
2. Vertex selection method - Selection of nearest neighborhoods (Especially important in multi-query, if the graph too large) (LaValle, 2014, pp.153-206)
3. Local planning methods – Between two nodes trajectory is created and collision is checked on this trajectory. Most often a linear trajectory is used with defined density of points. And in each point the collision is checked. Collision detection is a very important part of the path planning, takes the most computation time. One of the collision detectors is a library FCL, which is also used in MoveIt (Pan et al., 2012).

A single-query algorithm is useful for a fast trajectory search. The algorithm builds a tree in configuration space and tree is being iteratively grown until the goal state is reached. The base algorithm of this type is RRT – Rapidly-exploring random tree (Bircher et al., 2016).

The Tree is grown from an initialized state x_{init}, which is the root of the tree. The first step of the algorithm, after initialization, is to create a new random sample x_{new} from the configuration space of the robot. Then the nearest neighborhood x_{near} from the tree is searched. In the first iteration, the nearest neighborhood is the start state x_{init}. States x_{new} and x_{near} are connected with a linear line, on which new state x in the distance ε from x_{near} is created. If the state x and the connection between x and x_{near} is collision-free, the state x is added as new branch of the tree. This one iteration is illustrated on the fig. 2. Contrariwise, the fig. 3. demonstrates the example of fail when collision is between these two states.

The algorithm continues to the next iteration and new states are being randomly sampled, while the goal state x_{goal} is not reached in the specified time. An example of the whole the tree is displayed on fig. 4. Sometimes the x_{goal}, instead of the random sample, may be chosen. By choice of the growth in the direction towards the goal, the algorithm converges faster. But if the goal state is too often chosen, the algorithm can be trapped into a local minimum and may have a problem with obstacle avoidance. (Rodriguez et al., 2019)

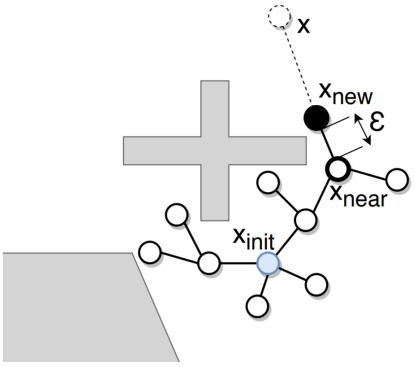

Fig. 2. Example of one iteration of RRT.

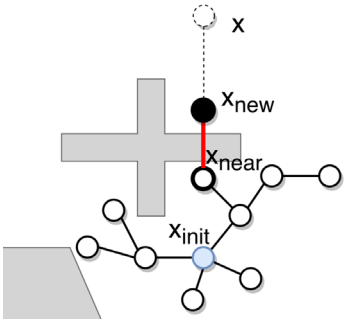

Fig. 3. Example of connection failure.

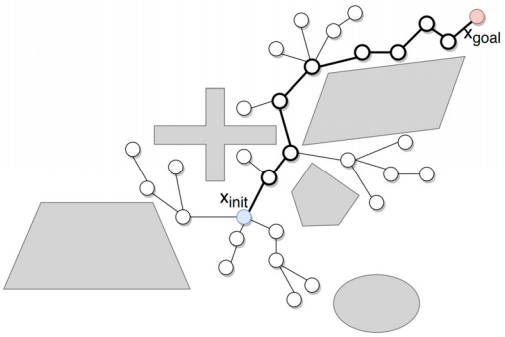

Fig. 4. Example of finding solution and growing a topological tree from $x_{init}$to x_{goal} by RRT.

RRT is often useful for fast computation of a feasible plan, but the path is not optimized. For this reason, modified versions, that can smooth the trajectory further, like RRT* exist (Karaman and Frazzoli, 2011). Another type of RRT is a bidirectional version, which builds two trees one from the start and second from the goal state. This strategy is used in RRTConnect (Kuffner and Lavalle, 2000). Transition-based RRT, which is inspired by molecular modeling with Bolzmann probability, uses additional parameters for avoiding local minimums in searched space (Jaillet et al., 2008).

Another group of sampling-based methods are multi-query methods. These algorithms are used for searching trajectories from multiple start states to multiple goal states. The Algorithms consist of two steps – built and query. In the first step a space is randomly sampled and collision is checked in each sample x_{new}. Then the collision-free samples are tried to connect with the nearest nodes x_{near} from the graph. If the connection between two states x_{new} and x_{near} is collision-free, the connection is added into the graph. In the second step, the solution in the graph is, by a simple deterministic algorithm e.g. (Dijkstra, 1959), found. The graph and solution is illustrated on fig. 5. The basic version of a multi-query algorithm is called PRM – Probabilistic roadmap (Kavraki et al., 1996).

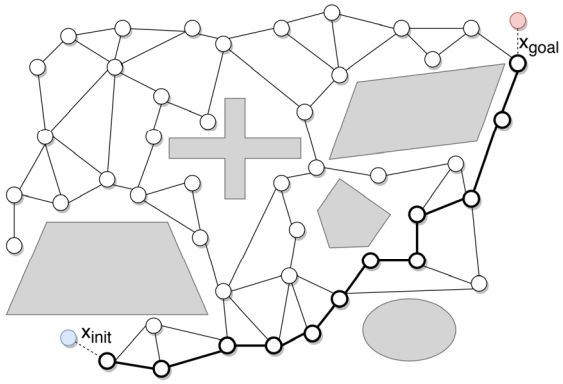

Fig. 5. Example of creation of topological graph by PRM and founding a solution from the graph.

4. STOCHASTICS TRAJECTORY OPTIMIZED MOTION PLANNER

Another class of motion planners, which can be applied to robotics manipulators, are optimization-based planners. In (Ratliff et al., 2009) planner called CHOMP is described, which formulates a trajectory optimization procedure based on covariant gradient descent. This algorithm minimizes a combination of smoothness and obstacle cost and to derive the gradients from obstacle in environment distance field representation of environment is used.

The planner called STOMP has similar approach and adopt a similar cost function, but instead the covariant gradient descent, generating of noisy trajectories is used and the best trajectory is chosen by cost function (Kalakrishnan et al., 2011).

First, the method creates a guess of a initialization trajectory, which can be linear interpolation, cubic polynomial or minimum control cost. STOMP considers trajectories of a fixed duration and trajectories are discretized into N waypoints, which is called *num_timesteps* in MoveIt implementation. The number of generated noisy trajectories is defined by parameter number of rollouts *num_rollouts* and algorithm is executed until maximum number of iterations *num_iterations* is reached. Important part of the algorithm is noise generator and in experiments of this article normal distribution sampling is used. The noise generator uses parameter *stddev* – standard deviations of noise on each joint of trajectory. Too low value of noise allows to explore configuration space of manipulation only near to the initial trajectory. In case of large obstacle on the path, the algorithm may not find a collision free trajectory (Fig. 6). Contrariwise, too high value of the noise can help to explore larger space of manipulation, but the trajectory may not be optimized (Fig 7.). This behavior can be improved by higher number of noisy trajectories, but it requires more time for computation. (Ros.org, 2011)

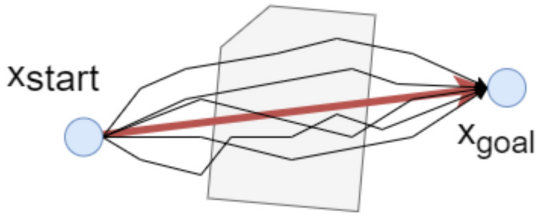

Fig. 6. Trajectories are generated with too low values of noise and STOMP is trapped in local minimum.

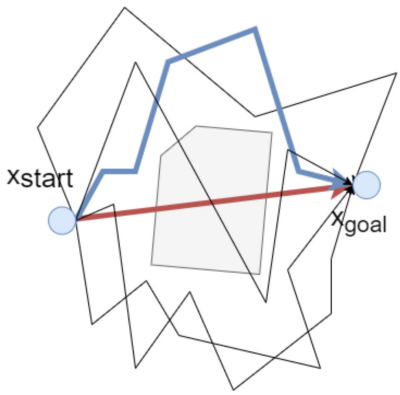

Fig. 7. Trajectories are generated with too high values of noise and chosen trajectory is not optimal.

5. EXPERIMENT SETUPS

These two MoveIt plugins OMPL and STOMP are compared on a simulation of a simplified working station with the robot ABB IRB 2100, which is illustrated on fig. 8. Experiment setups consist of two steps of production.

In the first experiment setup and in the first part of production the robot is autonomously picking a part from a bin. The start position is fixed, but positions and orientations of parts are randomized, and an application of some vision system is necessary for the part detection. Therefore, in this situation

autonomous execution of trajectory, which cannot be taught by programmer, is important to use.

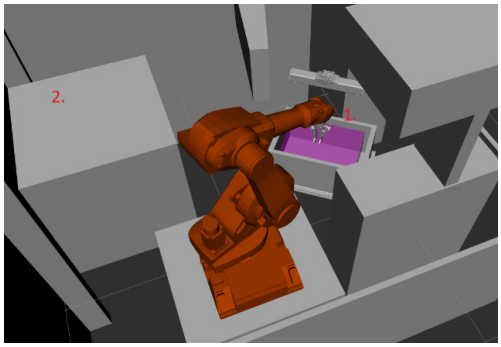

Fig. 8. Working station, on which experiments are executed and comparison of path planners is done.

In the simulation, where positions of parts are generated with defined step, motion planners are being tried to find trajectories. Joint values of start position are fixed: [-1.07, -0.23, 0.62, -0.14, 1.18, 1.39]. This scenario is marked as number 1 on Fig. 8. The example of one trajectory is displayed on Fig. 9.

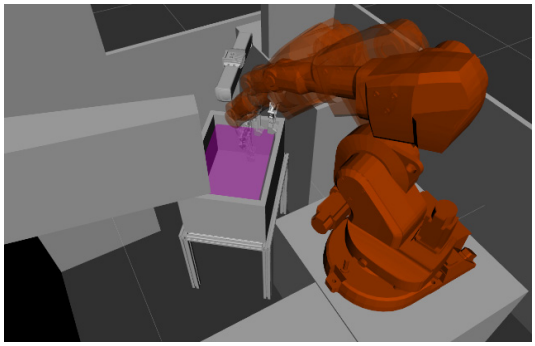

Fig. 9. Example of trajectory in first experiment.

In the second experiment setup the robot is moved to another station, where the picked part is placed on a fixed position. For simplification we imagine, that picked part is small (E.g., screw) and for the collision checking is insignificant. Therefore, any model of the part is not considered. An obstacle is located between the fixed start and the fixed goal state and the collision free trajectory must be found. Fig. 10. shows a example of trajectory, which is collision free. Algorithms are based on randomized sampling, so the trajectory is not the same in each attempt. The placing position is fixed with joint values: [0.70, 0.45, -0.27, -0.26, 1.12, -1.08] and the fixed start state is same as in previous operation from the first experiment.

After object handling in the station, the robot is returned to pick next part from box and production continue in next cycle.

Due to the fact, that the collision checking is a significant part of the time spent motion planning, it is important to define collision checking library. Both algorithms are using mentioned FCL library (Pan et al., 2012), which is supported in MoveIt. RRT and RRTConnect algorithm have no configuration parameters. Contrariwise, STOMP uses parameters, which has been described in STOMP chapter of this article.

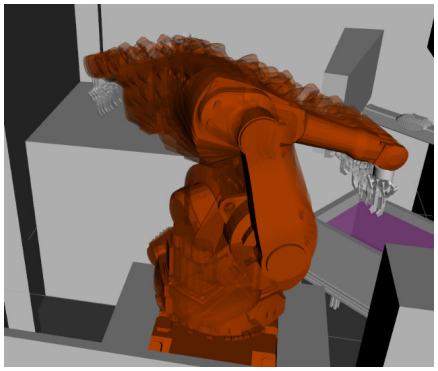

Fig. 10. Example of trajectory in second experiment.

In these experiments minimum control cost method as initialization method is used. The noise generator uses the normal distribution sampling and cost function is obstacle distance gradient.

The comparison consists of watching computation time, rate of success, and increase of tool point trajectory length with respect to forward linear path between the start and the goal states, what let us denote as *Trajectory increase** in the article. The Trajectory increase* *TI* is defined by:

$$TI = \sum_{i=1}^{n} \rho(p_i, p_{i-1}) \; / \; \rho(p_s, p_g) \tag{1}$$

Where $\rho(p_s, p_g)$ is Euclidean distance between tool point position in the start state p_s and tool point position in the goal state p_g. The sum of $\rho(p_i, p_{i-1})$ represents trajectory length of tool point, where n is number of waypoints and p_i is tool point position of waypoint on i position. The Euclidean distance is calculated by:

$$\rho(p_2, p_1) = \sqrt{(p_{x2} - p_{x1})^2 + (p_{y2} - p_{y1})^2 + (p_{z2} - p_{z1})^2} \tag{2}$$

6. RESULTS

The goal of the first experiment setup is finding of solution from the start state to the different goal states in bin.

STOMP parameters (Table 1.) are chosen from Photoneo Company experimental testing, which are optimized for finding a trajectory in the bin with minimum computation time.

Table 1. STOMP parameters from Photoneo Company experimental testing.

Parameter	Value
num_timesteps	5
num_iterations	5
num_rollouts	5
Standard deviations of noise on each joint [rad]	[0.2, 0.4, 0.5, 0.1, 0.1, 0.1]

In this case RRTConnect can find solution faster as simple RRT. Also, RRTConnect has better success rate, because RRT is failed, if timeout is reached. STOMP algorithm is the slowest with 97.9% success rate, which can be seen in Table 2.

Table 2. Experiment setup 1 – duration and success rate comparison.

Algorithm	Duration [s]	Success Rate [%]
RRT	0.15	91.5
RRTConnect	0.07	100
STOMP	0.2	97.9

Figures 11. – 13. show the trajectory increase in each attempt of the first experiment. If RRT or RRTConnect is used, significant deviations are seen on figures 11., 12. RRT had 887% maximal deviation from average trajectory increase and RRTConnect had 362% (The standard deviation is about 15%). But maximal deviation is very random and theoretical the maximal deviation can be infinity. This behavior is typical for these sampling-based planners, because they are not optimal and sometimes planner can create a rambling trajectory. STOMP had maximal deviations 82% and standard deviation 11%, which is depend on parameter stddev.

The comparison of trajectory increase can be seen on Table 3.

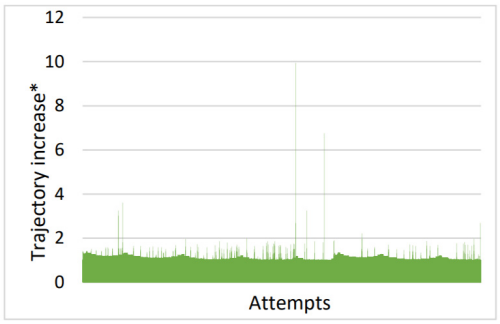

Fig. 11. OMPL – RRT and trajectory increase on each attempt.

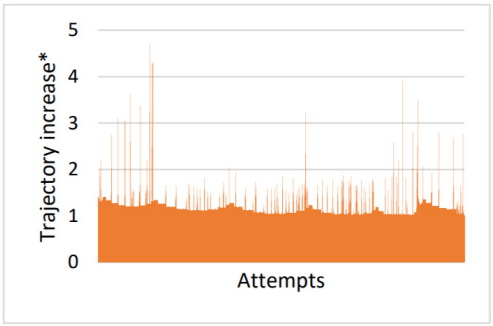

Fig. 12. OMPL – RRTConnect and trajectory increase in each attempt.

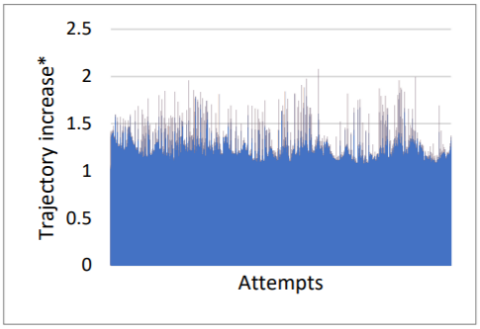

Fig. 13. STOMP and trajectory increase in each attempt.

Table 3. Experiment setup 1 – Comparison of average trajectory increase, standard deviation and maximal deviation of the trajectory increase.

Algorithm	Average	Standard deviation	Max. deviation
RRT	1.075	0.155	8.87
RRTConnect	1.064	0.155	3.62
STOMP	1.079	0.114	0.82

In this experimental setup case RRT and RRTConnect are faster with a risk, that the trajectory can be rambling. Possible solution, how to integrate this planner is usage of statistical methods for filtering wrong trajectories. STOMP is slower, but without risk for rambling trajectory. The final comparison of trajectory increase for each planners is showed on histogram Fig.14.

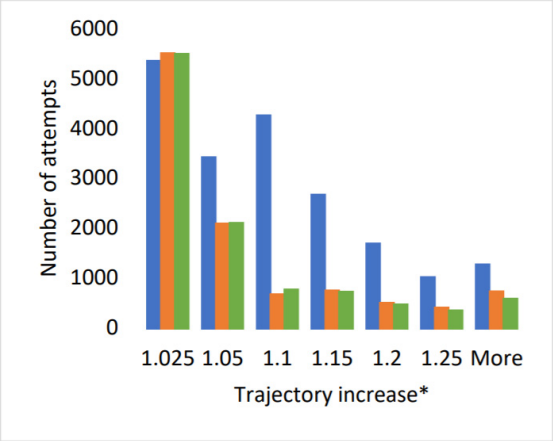

Fig. 14. Comparison STOMP (blue), RRTConnect (orange), RRT (green) by histogram, which shows number of attempts on intervals with different trajectory increase.

The second experiment setup consists of testing 10000 samples from the one start state to the one goal state. If STOMP used the same parameters as in the previous experiment, it would be impossible to find a solution, because the noise parameter is too low, and the solver has problems to find solution effectively (Fig. 15.).

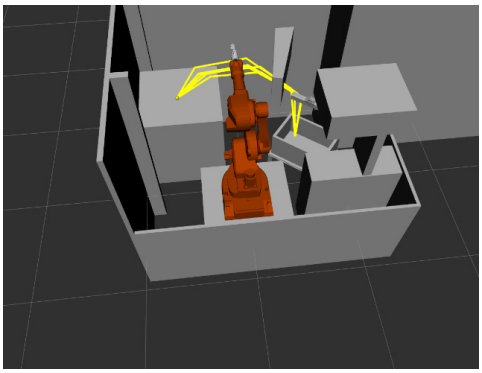

Fig. 15. Failure of STOMP, which used parameters from previous experiment. Noise is too low, and solver cannot find trajectory.

Thus, the parameters are changed to Table 4. and Fig. 16. shows, that STOMP with new parameters is able to find solution.

Table 4. STOMP parameters for the second experiment setup.

Parameter	Value
num_timesteps	10
num_iterations	10
num_rollouts	10
Standard deviations of noise on each joint [rad]	[0.3, 0.4, 0.4, 0.6, 0.6, 0.6]

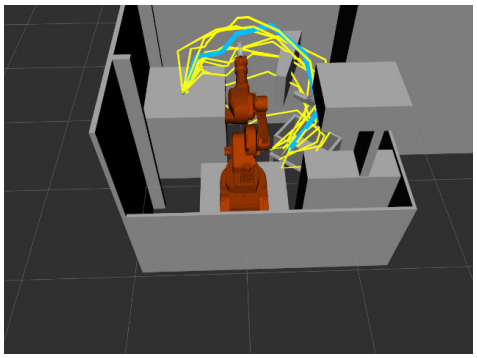

Fig. 16. STOMP trajectories from noise generator with increased gained noise.

In this experiment setup only RRTConnect as sampling-based planner is used, because it was better as RRT. Thus, RRTConnect is compared with STOMP.

According to Tab. 5. RRTConnect is faster with better success rate.

Table 5. Experiment setup 2 – duration and success rate comparison.

Algorithm	Duration [s]	Success Rate [%]
RRTConnect	0.22	99.8
STOMP	1.14	95.7

But in comparison of trajectory increase STOMP is again better, which has standard deviation 38.5%. RRTConnect has more significant peaks (Fig. 17.) with 104.9% standard deviation and the worst trajectory has 986% deviation. The completed comparison of trajectory increase is in Tab. 6.

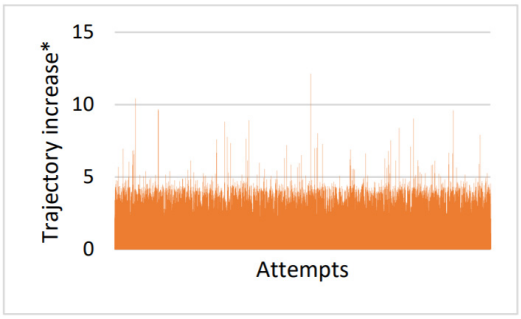

Fig. 17. OMPL RRTConnect and trajectory increase in each attempt.

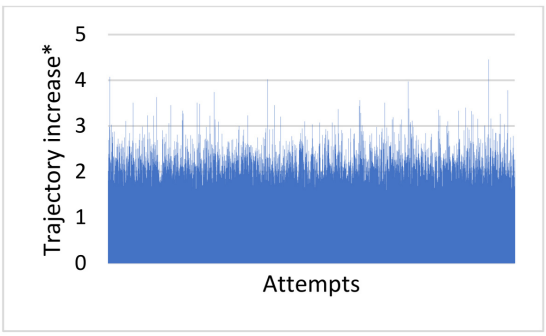

Fig. 18. STOMP and trajectory increase in each attempt.

Table 6. Experiment setup 2 – Comparison of average trajectory increase, standard deviation and maximal deviation.

Algorithm	Average	Standard deviation	Max. deviation
RRTConnect	2.24	1.049	9.86
STOMP	1.67	0.385	4.45

In this second experiment RRTConnect is faster again, but if in case of obstacle in scene, the chance, that the trajectory will be rambling, is increased. This can be seen in histogram Fig. 19., where trajectories, which are generated by STOMP, have nearer to average of trajectory increase.

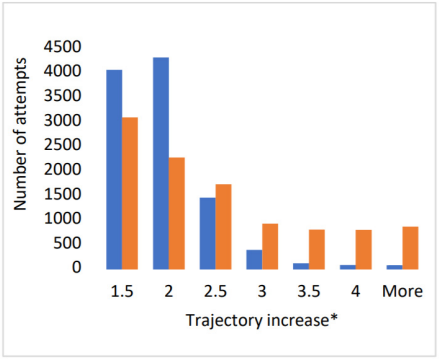

Fig. 19. Comparison STOMP (blue), RRTConnect (orange) by histogram, which shows number of attempts on intervals with different trajectory increase.

The recapitulation of results are show in Tab. 7., where motion planners are finale compared.

Table 7. Summary of results.

	RRTConnect	STOMP
Duration	faster	Slower
Parameters	No parameters required	Different cases of movement
Success rate	failures are only if timeout is reached	Needs to define different parameters in different cases of movement
Optimal trajectory	Optimal and the deviation in length trajectory is lower	Optimal and the deviation in length trajectory is lower
Setup joint limits	Setup joint limits is recommended, but if robot must work in large volume of his workspace, limitation of joint values can be impossible	Not required

CONCLUSION

RRTConnect is faster than RRT, therefore in the first experiment RRTConnect has better success rate than RRT, which fails only on timeout. In case of STOMP algorithm more timesteps and more rollouts causes longer computation time. On the other hand, when values are too low, STOMP is not be able to find a solution in a complicated environment (The second experiment). In case of moving to a bin, (The first experiment) lower values of the parameters are adequate, because the environment is not complicated. Also, in a production decreasing of cycle time is often required, therefore the motion planner must find the solution immediately. In this case RRTConnect is faster and has better success rate than STOMP. Problem is that a quality of trajectory is not guaranteed and sometimes the trajectory can be rambling.

We propose some methods for avoidance or minimization of this problem. One approach is reduction of a robot workspace by setup of joint limits. If the workspace is reduced the computed trajectory cannot contain any value outside of limits and the rambling trajectory is minimized in limited workspace. Also, the computation time is faster, because algorithm is searching only in limited workspace. Another possible approach is to use statistical methods for rejecting the rambling trajectories. If the environment is defined, it is possible to run a simulation and record lengths of trajectories in lot of attempts. From this dataset is possible to compute a standard deviation. A newly created trajectory will be rejected, if length will be longer than a defined threshold. This threshold can be derived as multiple of the standard deviation from simulation.

ACKNOWLEDGEMENT

This research was partially sponsored by Photoneo company. (http://photoneo.com). This work was supported by APVV-16-0006, APVV-17-0214.

REFERENCES

Bircher, A., Alexis, K., Schwesinger, U., Omari, S., Burri, M. and Siegwart, R. (2016). An incremental sampling-based approach to inspection planning: the rapidly exploring random tree of trees. *Robotica*, 35(6), pp.1327–1340.

Boor, V., Overmars, M.H. and van der Stappen, A.F. (1999). The Gaussian sampling strategy for probabilistic roadmap planners. *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, [online] 2, pp.1018–1023. Available at: https://ieeexplore.ieee.org/document/772447/ [Accessed 16 Oct. 2019].

Dijkstra, E.W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), pp.269–271.

Gasparetto, A., Boscariol, P., Lanzutti, A. and Vidoni, R. (2015). Path Planning and Trajectory Planning Algorithms: A General Overview. *Motion and Operation Planning of Robotic Systems*, [online] pp.3–27. Available at: https://link.springer.com/chapter/10.1007%2F978-3-319-14705-5_1.

Jaillet, L., Cortes, J. and Simeon, T. (2008). Transition-based RRT for path planning in continuous cost spaces. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp.2145–2150.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P. and Schaal, S. (2011). STOMP: Stochastic trajectory optimization for motion planning. *2011 IEEE International Conference on Robotics and Automation*, pp.4569–4574.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, [online] 30(7), pp.846–894. Available at: http://roboticsproceedings.org/rss06/p34.pdf [Accessed 16 Oct. 2019].

Kavraki, L.E., Svestka, P., Latombe, J.-C. . and Overmars, M.H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), pp.566–580.

Kuffner, J.J. and LaValle, S.M. (2000). RRT-connect: An efficient approach to single-query path planning. *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, [online] 2, pp.995–1001. Available at: https://ieeexplore.ieee.org/abstract/document/844730 [Accessed 16 Oct. 2019].

Lindemann, S.R. and LaValle, S.M. (2005). Current Issues in Sampling-Based Motion Planning. In: *Robotics Research. The Eleventh International Symposium*. Berlin, Heidelberg, New York: Springer, pp.36–54.

Lozano-Pérez, T. (1990). Spatial Planning: A Configuration Space Approach. *Autonomous Robot Vehicles*, pp.259–271.

Pan, J., Chitta, S. and Manocha, D. (2012). FCL: A general purpose library for collision and proximity queries. *2012 IEEE International Conference on Robotics and Automation*, [online] pp.3859–3866. Available at: https://ieeexplore.ieee.org/abstract/document/6225337/ [Accessed 16 Oct. 2019].

Ratliff, N., Zucker, M., Andrew, B.J. and Srinivasa, S. (2009). CHOMP: Gradient Optimization Techniques for Efficient Motion Planning. *2009 IEEE International Conference on Robotics and Automation*, [online] pp.489–494. Available at: https://kilthub.cmu.edu/articles/CHOMP_Gradient_Optimization_Techniques_for_Efficient_Motion_Planning/6552254 [Accessed 16 Oct. 2019].

Rodriguez, S., Xinyu Tang, Jyh-Ming Lien and Amato, N.M. (2019). An obstacle-based rapidly-exploring random tree. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* [online] Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1641823 [Accessed 16 Oct. 2019].

Ros.org. (2011). *STOMP Planner — moveit_tutorials Kinetic documentation*. [online] Available at: http://docs.ros.org/kinetic/api/moveit_tutorials/html/doc/stomp_planner/stomp_planner_tutorial.html [Accessed 16 Oct. 2019].

Steven Michael Lavalle (2014). *Planning algorithms*. New York: Cambridge University Press, pp.153–206.

Sucan, I.A., Moll, M. and Kavraki, L.E. (2012). The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4), pp.72–82.