Task Space Vector Field Guiding for Motion Planning*

Fernando Urra González, Jan Rosell, Raúl Suárez
Institute of Industrial and Control Engineering (IOC)
Universitat Politècnica de Catalunya (UPC) – Barcelona Tech
Barcelona, Spain
fernando.urra@upc.edu

***Abstract*—The article deals with the problem of planning in the task space in the presence of vector fields, while verifying and validating the constraints in the configuration space. The proposed approach, called the Task Space Vector Field Rapidly-exploring Random Tree (TSVF-RRT) algorithm, extends the Task-Space Rapidly-exploring Random Trees (TS-RRT) algorithm by incorporating vector fields into the task space, while avoiding non-trivial constraints on the configuration space. The planner restricts the search to a lower dimensional space, minimizing the upstream functional. To evaluate the proposed approach, graphical simulations are presented, carried out using a planar manipulator of 10 DOF. Possible advantages that encourage further research in this line are discussed.**

***Index Terms*—Motion Planning, Task-Space, Vector Field**

I. Introduction

Planning a collision-free path in real environments is a fundamental task in robotics. Many physical systems have hundreds of Degrees of Freedom (DOF). The human body, for example, has 244 DOFs, the majority of them in the hands and the feet. Most of the practical algorithms used in high-dimensional environments use sampling-based planners, including Probabilistic Route Maps (PRMs) [1] and Rapidly Exploring Random Trees (RRTs) [2]. These methods use random tests to obtain the connectivity of the free space, usually in configuration space (C-Space). Despite the success of these planners on certain classes of high-dimensional problems, the performance of the algorithms can be very sensitive to aspects like the way random samples are chosen and the distance metric used. Also, as with most planners, the complexity grows with the dimensions of the problem. In contrast, Task-Space sampling-based planners allow fast planning in high-dimensional systems by exploring actions in a low-dimensional, e.g. Task-Space RRT (TS-RRT) [3]. Usually the Task-Space is the Cartesian space where the robot operation is performed, and it usually represents the end-effector positions in the case of manipulators.

Although the computational cost per one sampling is generally more challenging in Task-space than in C-Space, planning within Task-Space has advantages such as: finding the path and goal configuration simultaneously, and easily using environmental information [4]. Instead of specifying a particular set of desired joint angles for the entire robot performing a grasping task, it can be considered only the position of the hand (solution in Task-Space), finding a feasible path and a corresponding goal configuration in the C-Space to the desired goal position in the Task-Space at the same time. In addition, planning within Task-Space allows to take advantage of environmental information directly.

Many planners in C-Space often suffer due to narrow passages or local traps problems. An approach proposed for these cases is inspired from obstacle avoidance techniques. It does not explicitly build a roadmap but, instead, builds a differentiable real-valued function, called potential function, that guides the motion of the moving object. The potential function usually consists of an attractive component, which pushes the robot towards the goal, and a repulsive component, which pushes the robot away from obstacles [5]. Planning on a configuration space in which a potential function value is associated with each point can also be cast as a vector field planning problem, by taking the (negative) gradient field of this potential function as the vector field [6]. The vector field path planning problem arises in a wide range of robotic applications, highlighting the Vector-Field RRT (VF-RRT) algorithm [7]. The VF-RRT algorithm extends a tree in a similar way to the EXTEND function of the classical RRT algorithm, but biasing the selection of random directions by means of optimal (or desired) vectors in such a way that the tree explores the directions indicated by the vector field before other directions. While vector field-based navigation combines the kinematic path planning problem with lower-level feedback controller design, which is robust despite the presence of disturbances, it cannot easily deal with obstacles in spaces with dimensions higher than two [8].

The method proposed here is called *Task Space Vector Field Rapidly-exploring Random Tree* (TSVF-RRT) and is an extension of the TS-RRT algorithm, incorporating vector fields in the Task-Space (low dimension), while validating the constraints in the configuration space. The algorithm allows to easily deal with obstacles in the Task-Space and incorporates geometric constraints for the end-effector, by orienting the vector field in the direction of said constraint, guiding the growth of the tree in C-Space.

* This work was partially supported by the Spanish Government through the project PID2020-114819GB-I00. Fernando Urra is funded by the ANID, Programa de Becas, Becas Chile de Doctorado en el Extranjero 72200501 and by the University of Talca.

After this introduction, the article is organized as follows. Section 2 exposes the background related to the proposed planner. Section 3 provides a description of the TSVF-RRT algorithm. Section 4 presents experimental results with different geometric constraints using on a planar manipulator. Section 5 presents a discussion of the results and the conclusions of the work.

II. Related Background

A. Vector-Field RRT (VF-RRT)

The VF-RRT algorithm is a planner that favors the tree growth in the direction of a vector field $f(q)$ to obtain trajectories with an upstream criterion. Its operation is similar to that of the Rapidly-Exploring Random Trees (RRT) algorithm. The difference is a biasing in the selection of random directions by means of optimal (or desired) vectors in such a way that the tree explores the directions indicated by the vector-field before any other directions [9]. Given the initial configuration q_{init} (root node of the tree) and the final configuration q_{goal}, a random node q_{rand} is generated in the C-Space, and then the nearest neighbor node q_{near} of q_{rand} in the tree is selected. Then, a unit vector v_{rand} from q_{near} to q_{rand} and a vector v_{field} of the vector field are determined.

The factor ω is a weight value that describes how much to draw v_{new} toward the v_{field} direction. As ω becomes bigger, v_{new} tends to be directed toward v_{field}. The growth direction of the tree is an intermediate direction v_{new} between the v_{rand} and the vector field v_{field}. The new node q_{new} is obtained from the extension with stepsize δ from q_{near} in the direction of v_{new} (Fig. 1). This is repeated until the distance between q_{new} and q_{goal} is less than the connectivity criterion [9].

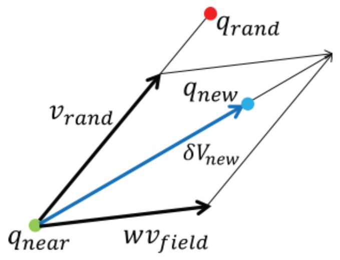


Fig. 1. Determination of a new node in the VF-RRT algorithm [7].

The VF-RRT algorithm adaptively balances between random exploration and deterministic vector field following. The algorithm retains several of the desirable features of the RRT algorithm, e.g. the Voronoi bias property, computational efficiency, and algorithmic simplicity. In [6] its application in non-honolonomic systems was demonstrated by using as an example a car-like model.

B. Task-Space RRT (TS-RRT)

The TS-RRT algorithm consists of building a tree of samples in the Task-Space, while verifying and validating the constraints in the C-Space. The implemented TS-RRT planner is based on the structure of the algorithm provided by the Open Motion Planning Library [10]. The operation of the TS-RRT is described in Algorithm 1. First, the initial and final configuration is projected onto the Task-Space (Line 4 and 5) and the sample tree T rooted at q_{start} and x_{start} is created (Line 6). Each node in the tree contains information from both the C-Space and the Task-Space. Iteratively, the tree grows until a certain stopping criterion is met (Line 7), e.g. maximum number of iterations, maximum size of the tree or specified period of time. At each iteration, a random sample, x_{rand} , is generated directly in the Task-Space. In the random process, the algorithm can choose the actual goal state with a certain probability. This bias, usually set to around 5%, allows to quickly connect T with x_{goal} (Line 9). Otherwise, the sample is obtained from a uniform distribution in a delimited region (Line 11). Subsequently, the tree grows towards x_{rand} by adding a new configuration x_{new}, calculated by the TS-EXTEND function (Line 13) detailed below. If the new configuration x_{new} matches x_{goal} (Line 14), then the algorithm has found a solution and it returns the path connecting q_{start} and q_{goal} through the configurations of the tree T (Line 16). Otherwise, this procedure is repeated with a new x_{rand} configuration. Finally, $\varnothing$ is returned (Line 19) if the stopping criterion is satisfied with no solution found.

Algorithm 1 BUILD TS-RRT

```
1: **In:** Initial Configuration $q_{start} \in C_{free}$
2:     Goal Configuration $q_{goal} \in C_{free}$
3: **Out:** Path $\mathcal{P} \in C_{free}$ between $q_{start}$ and $q_{goal}$
4: $x_{goal} \longleftarrow f(q_{goal})$
5: $x_{start} \longleftarrow f(q_{start})$
6: $T \longleftarrow$ InitTree[$q_{start}, x_{start}$]
7: **while** StopCriteria($T$) **do**
8:   **if** $p < GoalBias$ **then**
9:     $x_{rand} \longleftarrow x_{goal}$
10:  **else**
11:    $x_{rand} \longleftarrow$ SampleTask()
12:  **end if**
13:  $x_{new} \longleftarrow$ TS-EXTEND ($T$, [$x_{rand}, q_{rand}$])
14:  **if** $x_{new} = x_{goal}$ **then**
15:    $\mathcal{P} \longleftarrow$ Path($T$)
16:    **return** $\mathcal{P}$
17:  **end if**
18: **end while**
19: **return** $\varnothing$
```

The TS-EXTEND function, described in Algorithm 2, follows the procedure below to extend the tree T towards the x_{rand} configuration. First, the TS-NEAREST-NEIGHBOR() function selects from the configurations of the tree T the nearest node x_{near} to x_{rand} in Task-Space (Line 5). Then, a small step of predefined length ϵ is made from x_{near} to x_{rand}, reaching a new node x_{new} (Line 6), which is projected on to the C-Space (Line 7). If the line segment between q_{near} and q_{new} is valid, i.e. along the segment there are no collisions of the robot with itself or with the environment (Line 8), the segment is added to the tree (Line 9) and

x_{new} is returned (Line 10). If it is invalid, the tree does not grow and $\varnothing$ is returned (Line 12). A Voronoi bias in the Task-Space can dramatically improve the performance of random motion planners while avoiding non-trivial constraints on the C-Space, by encouraging exploration of empty regions in the Task-Space, rather than in the entire C-Space, which allows solutions to be found directly.

Algorithm 2 TS-EXTEND

```
1:  In: Sample tree $T$
2:      Configuration $x_{rand}$
3:      Configuration $q_{rand}$
4:  Out: Configuration $x_{new}$
5:  $[x_{near}, q_{near}] \longleftarrow$ TS-NearestNeighbor ($T$, $x_{rand}$)
6:  $x_{new} \longleftarrow$ TS-Control ($x_{near}$, $x_{rand}$)
7:  $q_{new} \longleftarrow f\,(x_{new})$
8:  if ValidSegment ($q_{near}$, $q_{new}$) then
9:    AddSegment ($T$, [$x_{near}$, $x_{new}$], [$q_{near}$, $q_{new}$])
10:   return $x_{new}$
11: end if
12: return $\varnothing$
```

III. Proposed Planner

The proposed planner, *Task Space Vector Field Rapidly-Exploring Random Tree* (TSVF-RRT), is based on the standard TS-RRT, which does not incorporate information to sample in the task space. This means that TS-RRT performs pure exploration, which causes the Voronoi bias in the task space, i.e., at each iteration, the probability that a node is selected is proportional to the volume of its Voronoi region; hence, search is biased toward those nodes with the largest Voronoi regions (representing unexplored regions of the configuration space) [11]. The Voronoi bias helps to quickly explore the entire Task-Space, but this can increase the probability of exploring areas far from the goal [4]. To solve this problem, the proposed planner TSVF-RRT, performs the exploration in the Task-Space biasing based on a vector field. Incorporating this vector field into a lower dimensional space allows to better deal with obstacles [8], constraining the growth direction in an intermediate direction between that indicated by the random sample (v_{rand}) and that of the vector field (v_{field}), i.e.,

$$v_{aux} \longleftarrow \alpha v_{field} + \beta v_{rand}, \tag{1}$$

where the weights α and β are controlled by a parameter $\lambda > 0$ such that $v_{aux} \rightarrow v_{field}$ if $\lambda \rightarrow \infty$ and $v_{aux} \rightarrow v_{rand}$ if $\lambda \rightarrow 0$. λ is initialized in a predefined λ_{init} and then updated every k iterations [6]. The proposed planner will use the method in [12] to update λ, which initializes it to λ_{max}, that is, a value high enough for the tree to follow the vector field in the first iteration, updating λ in each iteration. λ is bound in a range $[\lambda_{min}, \lambda_{max}]$ to prevent it from overflowing or growing too large unnecessarily. λ_{min} and λ_{max} are set to 10^{-3} and 10^{5}, respectively.

The position of the new candidate node x_{aux} is obtained by extending the stepsize (δ) from x_{near} in the direction of v_{aux},

$$x_{aux} \longleftarrow x_{near} + \delta v_{aux} \tag{2}$$

The new node x_{aux} is projected onto the C-Space obtaining a new configuration, which will be incorporated into the algorithm as q_{aux} in each iteration, q_{aux} guides the growth of the tree in the C-Space based on the vector field of the Task-Space. Then, the configuration q_{near} nearest to q_{aux} is selected and a defined step is taken from q_{near} in the direction of q_{aux}, thus reaching a new configuration q_{aux}. If the line segment between q_{near} and q_{aux} is valid, q_{aux} is added to the tree (Fig. 2).

Unlike the TS-RRT algorithm, which directly grows the tree in the Task-Space with a distance function defined also in the Task-Space [3], the TSVF-RRT algorithm consider the behaviors of the local planner. A Task-Space distance function frequently leads to failed local planning, because a pair of nodes has a close distance in Task-Space, but a far distance in C-Space. This causes inefficient performances, since its operations include, in addition to the inverse kinematics calculation, several collision detection calls [13].

The operation of the TSVF-RRT planner is described in Algorithm 3. First, a random node x_{rand} is generated directly in the Task-Space (Line 12). Then, in the TSVF-NEAREST-NEIGHBOR() function, the x_{near} node nearest to x_{rand} is selected and projected onto the C-Space (Line 14). Subsequently, in GetNewDirection() a new x_{aux} is obtained (Line 15).

Algorithm 3 BUILD TSVF-RRT

```
1:  In: Initial Configuration $q_{start} \in C_{free}$
2:      Goal Configuration $q_{goal} \in C_{free}$
3:      Vector Field field
4:  Out: Path $\mathcal{P} \in C_{free}$ between $q_{start}$ and $q_{goal}$
5:  $x_{goal} \longleftarrow f\,(q_{goal})$
6:  $x_{start} \longleftarrow f\,(q_{start})$
7:  $T \longleftarrow$ InitTree$[q_{start}, x_{start}]$
8:  while StopCriteria($T$) do
9:    if $p < GoalBias$ then
10:     $x_{rand} \longleftarrow x_{goal}$
11:   else
12:     $x_{rand} \longleftarrow$ RandTask()
13:   end if
14:   $[x_{near}, q_{near}] \longleftarrow$ TSVF-NearestNeighbor ($T$, $x_{rand}$)
15:   $x_{aux} \longleftarrow$ GetNewDirection ([$x_{near}, x_{rand}$], $T$, field)
16:   $q_{aux} \longleftarrow f^{-1}\,(x_{aux})$
17:   $q_{new} \longleftarrow$ TS-EXTEND ($T$, $[x_{near}, q_{near}, q_{aux}]$)
18:   if $q_{new} = q_{goal}$ then
19:     $\mathcal{P} \longleftarrow$ Path($T$)
20:     return $\mathcal{P}$
21:   end if
22: end while
23: return $\varnothing$
```

GetNewDirection(), described in Algorithm 4, returns x_{aux} using the following steps. First the unit vector v_{rand} is determined from x_{near} to x_{rand} and the vector v_{field} of the vector field. The weights α and β describe how much v_{aux} tends towards the direction of v_{field} or v_{rand} respectively, according to (1). Once v_{aux} is obtained, the position of the new node x_{aux} is obtained from q_{near} in the direction of v_{aux} with the stepsize δ. The node x_{aux} is projected onto the C-Space (Line 16). The result of the projection will be the q_{rand} random configuration, which will favor the growth of the tree in C-Space based on the direction of the vector field in Task-Space.

Algorithm 4 GetNewDirection

```
 1: In: Configuration $q_{near}, x_{near}$
 2:       Sample tree $T$
 3:       Vector Field field
 4: Out: Configuration $x_{aux}$
 5: $v_{rand} \longleftarrow \frac{x_{rand} - x_{near}}{\| x_{rand} - x_{near} \|}$
 6: $v_{field} \longleftarrow$ GetVectorField ($x_{near}$, field)
 7: $[\alpha, \beta] \longleftarrow$ GetWeights ($v_{rand}, v_{field}, T$)
 8: $v_{aux} \longleftarrow \alpha v_{field} + \beta v_{rand}$
 9: $x_{aux} \longleftarrow x_{near} + \delta v_{aux}$
10: return $x_{aux}$
```

The TSVF-EXTEND() function, described in Algorithm 5, extends the tree T to the q_{rand} configuration in a similar way to the RRT algorithm [14]. If the new configuration q_{aux} matches q_{goal}, then the algorithm has found a solution, returning the path connecting q_{start} and q_{goal}. If the stopping criterion is satisfied without having found a solution, the algorithm returns $\varnothing$.

Algorithm 5 TSVF-EXTEND

```
 1: In: Sample tree $T$
 2:       Configuration $x_{near}$, $q_{near}$, $q_{rand}$
 3: Out: Configuration $x_{aux}$
 4: $q_{aux} \longleftarrow q_{near} + \min(\epsilon, \| q_{rand} - q_{near} \|) \frac{q_{rand} - q_{near}}{\| q_{rand} - q_{near} \|}$
 5: $x_{aux} \longleftarrow f(q_{aux})$
 6: if ValidSegment ($q_{near}$, $q_{aux}$) then
 7:     AddSegment ($T$, [$x_{near}$, $x_{aux}$], [$q_{near}$, $q_{aux}$])
 8:     return $q_{aux}$
 9: end if
10: return $\varnothing$
```

IV. Results

In this section, some examples of a planar manipulator that needs to find a suitable trajectory from a given starting position to a given final configuration (Fig. 3). The Task-Space used is the cartesian coordinate of the end-effector. Four algorithms were tested in this problem, RRT, RRT-Connect, TS-RRT, and TSVF-RRT. We will compare the efficiency of the algorithms in terms of the time t required to complete the path search, the number n of iterations, and the success rate (the percentage of successful executions) within the maximum planning time limit (30 seconds). The robot has 10 links, all of equal length.

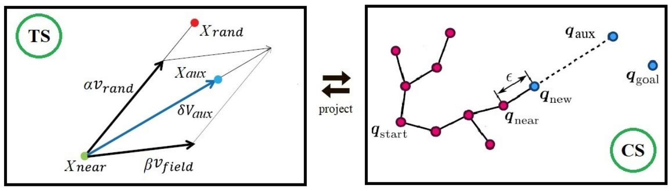


Fig. 2. Schematic of the TSVF-RRT planner. The Task-Space (left) shows the fetching of x_{aux} with an embedded vector field. The new node is projected on the C-Space (right), where the projected configuration is used as the q_{aux} for the growth of the sample tree.

The approach proposed in this work has been implemented within The Kautham Project [15], a motion planning and simulation environment developed at the Institute of Industrial and Control Engineering (IOC-UPC) for teaching and research. The experiments were run on an 1.80GHz Intel i7-8565U, 16GB RAM PC + Ubuntu 20.04.4 LTS.

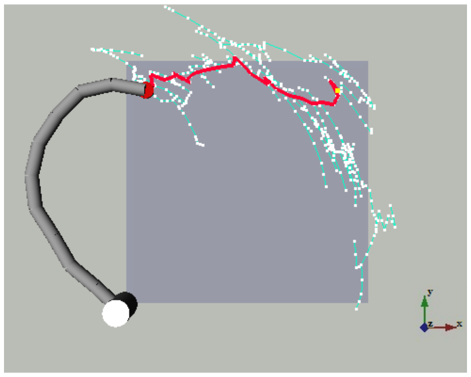
Figure 3 shows the results of the RRT algorithms (a), the RRT-Connect algorithm (b), and the TS-RRT algorithm (c). The figures show the planar manipulator trees (cyan), and solution paths (red) obtained with the planners, and the planar manipulator. Fig. 4 shows the results of the TSVF-RRT algorithm. Four regions have been artificially imposed on Task-Space, each with a vector field imposing different direction of movements (indicated by arrows). The vector-field points downwards in the left region, rightwards in the middle-bottom region, and upwards in the right region. However, it does not establish a clear direction of movement in the middle-top region. The end-effector should go from x_{start} to x_{goal}, with the vector-field guiding the growth of the tree in C-Space.

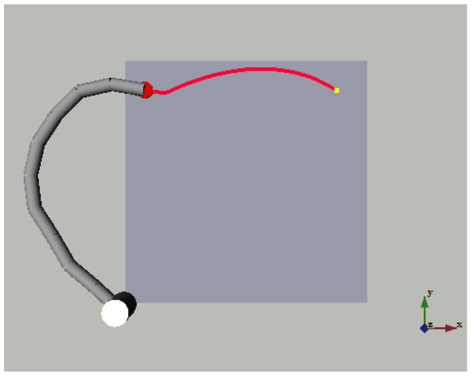
Table 1 shows the average results after 30 executions of the mentioned algorithms, with the planning time limited to 30 s. In the table 1 it is observed that for the RRT-Connect algorithm the path found is shorter and the tree has explored a smaller area in C-Space than in the case of the other algorithms.

The average time $t = 0.287$ s of the RRT algorithm, shows that the speed of this algorithm is higher than that of the TS-RRT and TSVF-RRT algorithms, even though the number of iterations $n = 715$ is five times higher than the TS-RRT algorithm with $n = 142$. RRT algorithms have been found to be quite fast in open environments [13]. In the proposed planner TSVF-RRT, it is observed that the tree tends to explore the directions indicated by the vector field before other directions. Even so, it presents a time $t = 1.727$ s smaller than the TS-RRT algorithm with $t = 3.584$ s.

TABLE I
Comparison of the four algorithms in 2D environment.

Algorithm	Success rate (%)	Planning time (t)	N° of iterations (n)
RRT	100	$0,2875$	715
RRT-Connect	100	$0,0687$	19
TS-RRT	100	$3,584$	142
TSVF-RRT	100	$1,727$	163





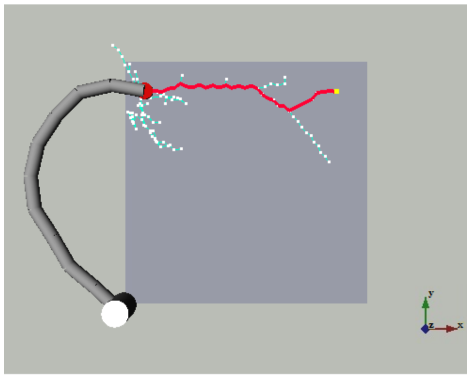
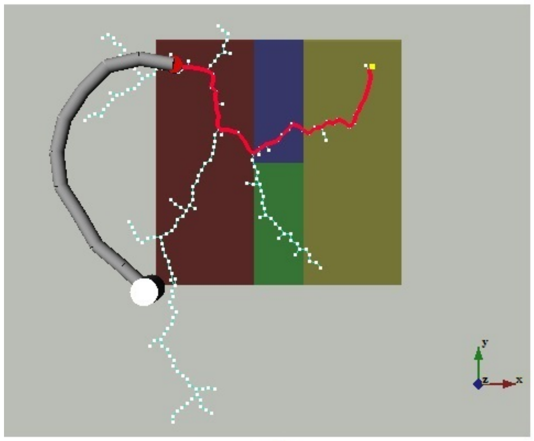


Fig. 3. Results of (a) RRT, (b) RRT-Connect and (c) TS-RRT in 2D environment. All figures show the initial configuration of the robot and the search tree in the Task-Space.



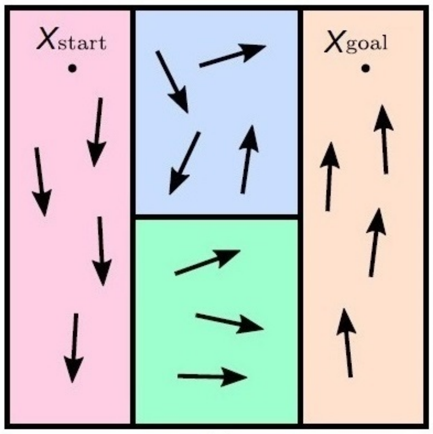


Fig. 4. Result of (a) TSVF-RRT in 2D environment and (b) the imposed vector field in Task-Space.

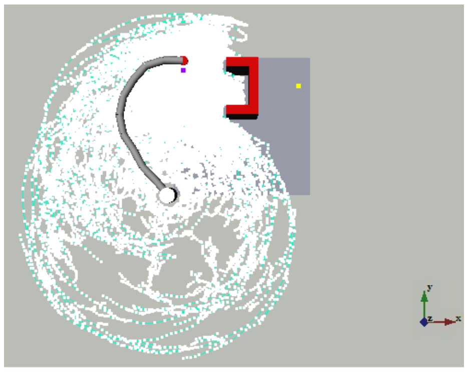
TABLE II
COMPARISON OF THE FOUR ALGORITHMS IN 2D ENVIRONMENT WITH OBSTACLE.

Algorithm	Success rate (%)	Planning time (t)	N° of iterations (n)
RRT	0	–	–
RRT-Connect	100	$5,944$	1620
TS-RRT	73	$21,299$	826
TSVF-RRT	100	$11,885$	741

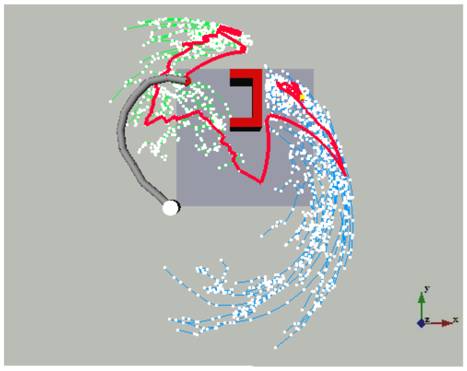
After obtaining these results in a space free of obstacles, a second problem is generated incorporating an obstacle in the path of the planners. The intention of this problem is to observe the response of the TSVF-RRT planner in a space with obstacles.

The artificially imposed regions in Task-Space (Fig. 4.b) favor the growth direction of the end-effector towards the goal and away from the obstacle. Figure 5 shows the process of the planners RRT (a), RRT-Connect (b), TS-RRT (c) and TSVF-RRT (d) with the built-in obstacle (red) in each of them.

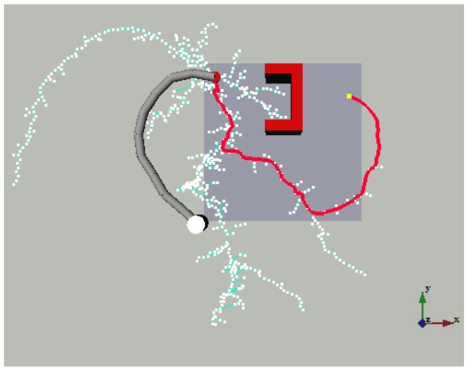
Table 2 shows the average results after 30 executions. It is observed that the RRT algorithm does not find a successful path within the time limit. In the case of the RRT-Connect algorithm, the found trajectories required an average time $t = 5.944$ s, smaller than those of the TS-RRT ($t = 21,299$ s) and TSVF-RRT ($t = 11.885$ s) algorithms. Even so, the tree of the TSVF-RRT algorithm explores a smaller area, with a lower number of iterations ($n = 741$) compared to the RRT-Connect ($n = 1620$) and TS-RRT ($n = 826$) and a success rate of 100% versus 73% presented the TS-RRT. Even when the proposed planner requires more time than RRT-Connect, it tends to explore the directions indicated by the vector field before other directions, better complying with the constraints imposed in the Task-Space.



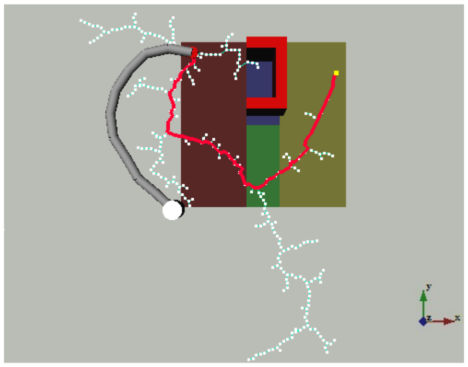
a)



b)



c)



d)

Fig. 5. Results of (a) RRT, (b) RRT-Connect, (c) TS-RRT and (d) TSVF-RRT in 2D environment with obstacle.

V. Conclusion and Future Work

The paper has presented a motion planner called *Task Space Vector Field Rapidly-exploring Random Tree* (TSVF-RRT). Based on the combination of the TS-RRT and VF-RRT algorithms, it maintains the good properties of both approaches, by adaptively balancing between random spatial exploration and tracking of deterministic vector fields in low-dimensional space. Experimental results for a planar manipulator show that this algorithm can achieve good performance in 2D environments and its efficiency in the number of iterations is superior to RRT, RRT-Connect and TS-RRT algorithms. However, at runtime it has lower efficiency than some of them. This may be because the computational cost per test is generally more challenging in the Task-Space than in the C-Space, when performing a projection (IK and FK) between both spaces. Future work is focused on extending the planner to 3D real world environments and, also, automatically define the vector field to incorporate constraints of the task to generate feasible movements from high-level requirements. Solving the problem of exploration in a constrained task space can be a powerful tool to allow a variety of planning algorithms to work in higher dimensional spaces.

References

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

[2] S. M. Lavalle. ”Rapidly-exploring random trees: A new tool for path planning,” Technical Report TR 98-11, Dept. of Computer Science, Iowa State University, Oct.1998.

[3] A. Shkolnik and R. Tedrake, ”Path planning in 1000+ dimensions using a task-space voronoi bias,” In 2009 IEEE International Conference on Robotics and Automation (ICRA), pages 2061–2067, 2009.

[4] R. Terasawa, Y. Ariki, T. Narihira, T. Tsuboi and K. Nagasaka, ”3D-CNN Based Heuristic Guided Task-Space Planner for Faster Motion Planning,” 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 9548-9554.

[5] B. Siciliano and O. Khatib, ”Springer Handbook of Robotics”. Berlin,Heidelberg: Springer-Verlag, 2007.

[6] I. Ko, B. Kim, and F. C. Park, “Randomized path planning on vector fields,” The International Journal ofRobotics Research, vol. 33, no. 13, pp. 1664–1682, 2014.

[7] I. Ko, B. Kim and F. C. Park, ”VF-RRT: Introducing optimization into randomized motion planning,” 2013 9th Asian Control Conference (ASCC), 2013, pp. 1-5.

[8] A. Jahn and L. C. A. Pimenta, ”Sampling Based Path Planning and Vector Fields for Curve Tracking by UAVs,” 2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR), 2016, pp. 223-228.

[9] N. García, J. Rosell, R. Suarez, ”Aplicación de algoritmos RRT en la planificación de movimientos óptimos en robótica,” A: Metaheuristics International Conference. ”Proceedings of the 12th Metaheuristics International Conference”. Barcelona: 2017, p. 953-962.

[10] I. A. Sucan, M. Moll and L. E. Kavraki, ”The Open Motion Planning Library,” in IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72-82, Dec. 2012.

[11] S. R. Lindemann and S. M. LaValle, ”Incrementally reducing dispersion by increasing Voronoi bias in RRTs,” IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA ’04. 2004, 2004, pp. 3251-3257 Vol.4.

[12] N. García, R. Suárez and J. Rosell, ”First-Order Synergies for Motion Planning of Anthropomorphic Dual-Arm Robots,” 20th IFAC World Congress, Toulouse, France, July 2017. IFAC-PapersOnLine, Volume 50, Issue 1, July 2017, Pages 2247-2254.

[13] J. Lee and S. Yoon, ”PROT: Productive regions oriented task space path planning for hyper-redundant manipulators,” 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 6491-6498.

[14] J. J. Kuffner and S. M. LaValle, ”RRT-connect: An efficient approach to single-query path planning,” Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings, 2000, pp. 995-1001 vol.2.

[15] J. Rosell et al., “The Kautham project: A teaching and research tool for robot motion planning,” in Proc. IEEE Int. Conf. Emerg. Technol. Factory Autom., Barcelona, Spain, Sep. 2014, pp. 1–8.

[16] M. Elbanhawi and M. Simic, ”Sampling-Based Robot Motion Planning: A Review,” in IEEE Access, vol. 2, pp. 56-77, 2014.