A new path smoothing method based on RRT* for non-holonomic mobile robots

S.A.Eshtehardian
Department of Mechanical Engineering
Sharif University of Technology
Tehran, Iran

S.Khodaygan
Department of Mechanical Engineering
Sharif University of Technology
Tehran, Iran

***Abstract*—The conventional RRT* algorithm can generate a graph-type path for motion of mobile robots. Due to discontinuity of generated path, tracking the path for special non-holonomic mobile robots is difficult or impossible. In this paper, a new method is proposed to generate a smoother path in the output of the RRT* algorithm based on the B-Spline interpolation. The applicability of the proposed method is evaluated through considering a case study and the results are compared and discussed.**

Keywords—Path Planning, RRT*, Mobile Robots, B-Spline

I. Introduction

Nowadays, one of the most noticed fields in terms of engineering and computer science is robotics. Diversity of robots can help us implementing them to a lot of tasks. Meanwhile, there are a large number of issues in this way, and solving them will able us to utilize robots in a more intelligent and sufficient way. One of the most prominent subsets of such issues is the problem of robot's path planning, in particular for mobile robots. There are a lot of path planning algorithms [1] such as RRT [1,5,6] that each of them is appropriate for a corresponding application.

Some of the most practical groups of algorithms in this regard are sampling-based algorithms, which usually in general are divided into two basic algorithms, RRT [1,5,6] and PRM [1,2,3,4]. PRM is generally used for multi-query tasks in structured environments like factories. RRT, however, is a proper choice for single-query tasks and has become a common algorithm in recent decades, which was introduced by Steven M.Lavalle in 1998. Meanwhile, there are some problems in terms of application of RRT to the motion planning tasks. The foremost one is that it is not probability completed, which means it is not guaranteed to find the optimal solution, even if after infinite iterations. Indeed, Karaman and Frazzoli, added some options to the classic RRT in order to change it into a probabilistic completed one. As a result of that, a new algorithm named RRT*[2] was designed. RRT* and its modifications like RRT*-Smart [6] seem suitable for a large number of path planning problems from robotics to computer games. But the only issue that may puzzle designers in robotics is how to deal with non-holonomic constraints since the output of RRT* is a discrete path. A reliable solution with respect to the mentioned limitations will be a combination of RRT* with a proper CAGD curve [1] that is able to generate a smooth path observing corresponding non-holonomic constraints of a particular robot. In this essay, it is tried to tackle this problem by means of B-Spline interpolation [12].

II. Recent work

In [8] another B-Spline interpolation has been implemented to a Dynamic RRT algorithm. After that, in [9] a new method based on RRT and its combination with Bezier cure was introduced. Finally, in RRT-based algorithms, a smoothing approach was designed to prevent problems caused by non-holonomic constraints [10]. Subsequently, a novel method, according to conflating RRT*-connect and cubic Ferguson, has been proposed in [11]. Now, due to the recommended future directions in [1], it is decided to prepare a new algorithm by which non-holonomic constraints are overcome. The new algorithm can generate a smooth path so as to prepare the way in which non-holonomic mobile robots can pass without any disturbance due to the non-holonomic constraints.

III. Basic concepts

Before introducing the proposed algorithm, first of all, the aim is to introduce some basic concepts related to it as follows:

A. B-Spline

There are various CAGD tools that have been designed so far. However, each of them has its particular application, and it is not a good idea to use them arbitrarily in the prepared tasks. According to this fact, an appropriate tool that completely fitted the pre-defined problem will be looked for. In this vein, first utilizing NURBS [12] was considered, since it is a cut above other algorithms in this regard, but a considerable issue about it is that it requires a lot of parameters, even if in using it for showing a simple shape, such as a circle. Therefore, a decision was made to switch to B-Spline [12], which is another practical CAGD after NURBS, and it not only needs fewer parameters to be considered but also it is flexible enough in comparison to other CAGD curves (like Bezier). In addition, B-Spline has been applied to path planning tasks several times in the past, like in [8] and [10]. Consequently, it is deduced to use B-Spline again with RRT* in this problem. Now, the goal is to first execute the algorithm, then eliminate some of the links and then adding others new in order to consider turning radius and then implement a B-Spline interpolation for avoiding problems caused by non-holonomic constraints.

Assume n points that everyone denoted by $\boldsymbol{x_i}$. In definition, a B-Spline by which it is possible to interpolate or approximate some points is shown in eq.1[12]:

$$P(x) = \sum_{i=0}^{n} p_i N_{i,k}(x) \qquad (t_{k-1} \le x \le t_{n+1}) \qquad (1)$$

Where each p_i is one of the B-Spline coefficients that should be calculated. Moreover, $N_{i,k}(x)$is a blending function denoted in eq.2 and .3 [12]:

$$N_{i,k}(x) = \frac{(x-t_i)N_{i,k-1}(x)}{t_{i+k-1}-t_i} + \frac{(t_{i+k}-x)N_{i+1,k-1}(x)}{t_{i+k}-t_{i+1}} \tag{2}$$

$$N_{i,1}(u) = \begin{cases} 1, & t_i \le x \le t_{i+1} \\ 0, & otherwise \end{cases} \tag{3}$$

In this article, the plan is to utilize Fortran routine Parcur method for identifying the coefficient of the B-Spline [13]. In this way, assume a dataset x_r consisting of m×n elements. Let define x_r^* in which r=1,2, 3…, m as follows [13]:

$$x_{r,l}^* = a_l x_{r,l} + b_l \qquad r=1,\ldots,m\ ,\ l=1,\ldots,n \tag{4}$$

And $s^*(x^*) = (s_1^*(x^*),\ldots,s_n^*(x^*))$ is the curve corresponding to which is derived from Parcur or Clocur and owning B-spline coefficient $p_{i,l}^*$ in which i=-k,…,g .due to the normalization property mentioned in[9], for reaching the B-spline curve $s(x) = (s_1(x),\ldots,s_n(x))$ interpolating the data points x_r must have the coefficient as follows:

$$p_{i,l} = \frac{p_{i,l}^* - b_l}{a_l}\ , i = -k,\ldots,g\ , l = 1,\ldots,n. \tag{5}$$

The shape of the curve may firmly be affected by choosing parameter values u_r is a parameter by which the overall shape of the curve can be determined. They can be chosen based on the physical constraints (i.e., about four-wheeled autonomous robots, it is recommended to use centripetal method). In this method, chord length parametrization is used w.r.t. the experimental robot's kinematic constraints [13]:

$$t_1 = 0 \tag{6}$$
$$t_r = t_{r-1} + d_r \tag{7}$$
$$a = t_1, b = t_m \tag{8}$$
$$d_r = \|x_r - x_{r-1}\| \tag{9}$$

B. *RRT* Algorithm*

RRT* and its modifications are one of the most important and practical algorithms in robotic motion planning, which was first developed by Karaman and Frazzoli [2]. In these algorithms, the output usually is one of the branches of a tree, similar to the classic RRT [1,5,6]. However, unlike RRT, RRT* privileges probability completeness property. According to this property, RRT* can find the optimal path when the number of iterations is infinite. Hence, RRT* is a suitable choice. RRT* is one the most practical algorithms when the aim is finding a path in a single query problem, especially when the environment is somewhere in the real-world instead of a virtual world like a computer game. That's because in real world applications using some grid-based algorithms like A* may have some difficulties for implementation. In [1] more information about where to use RRT* will be found. RRT* is presented in Algorithm 1[14]:

```
Algorithm 1  :  RRT*
1   :    Input: initial node $x_{init}$ , max nodes $K \leftarrow N$
2   :  $V \leftarrow \{x_{init}\}; E \leftarrow \emptyset;$
3   :    for i=0 to K do
4   :  $x_{rand} \leftarrow Sample(i)$
5   :  $x_{nearest} \leftarrow Nearest(V, x_{rand})$
6   :  $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
7   :  $V \leftarrow V \cup \{x_{new}\}$
8   :    If ObstacleFree($x_{nearest}, x_{rand}$) then
9   :  $X_{near} \leftarrow Near(V, x_{new})$
10  :  $x_{nearest} \leftarrow Parent(X_{near}, x_{nearest}, x_{new})$
11  :     $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
12  :     $E \leftarrow Rewire(X_{near}, E, x_{new})$
13  :    end if
14  :    end for
15  :    return (V, E)
```

In Algorithm 1, First, x_{rand} sampled from obstacle free space, then the nearest node to the x_{rand} in the previous graph is selected. Were the Euclidean distance between them less than a constant amount, a direct line between would be added to the graph and x_{rand} would change to x_{new}. Else, rely on the constant amount which was mentioned before, on the straight line between $x_{rand}$and $x_{nearest}$at the distance which equal to the constant amount from $x_{nearest}$, x_{new} will be selected (steer function as shown in Figure 1).

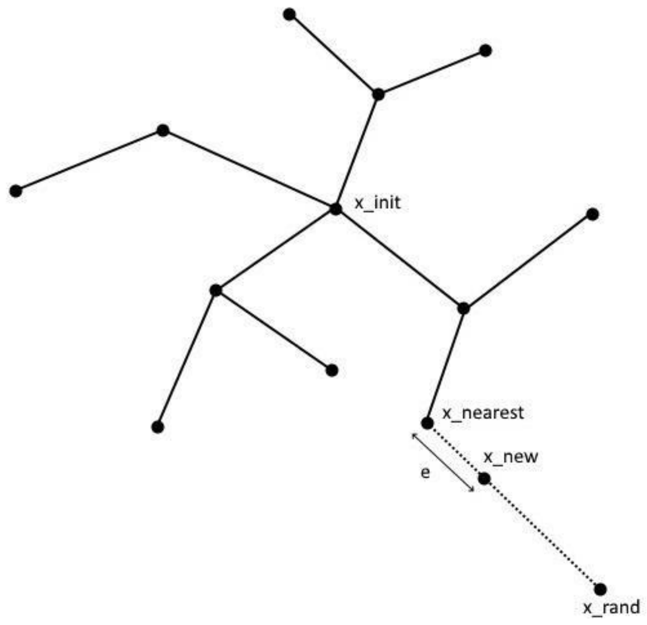

Figure 1: Steer functio

Among the neighbors, one that after connecting to the random node, has the minimum cost will be selected as the parent node as shown in Figure 2. This process is done by Parent function.

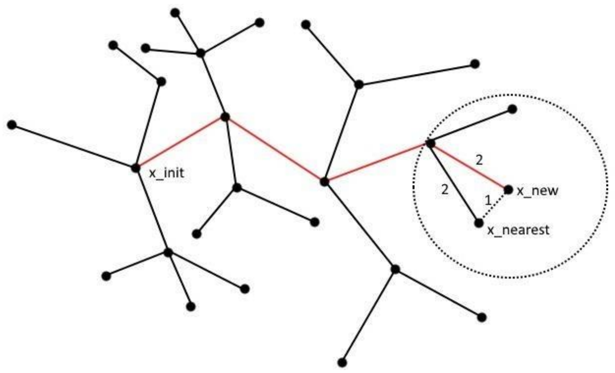

Figure 2: Parent function

Essentially, the final step is going to do, which is named rewiring by Rewire function. At this time, this approach checks the near nodes. If the path which passes x_{new} and then one of the near nodes has less cost than the present path, the link between that node and its parent will be deleted, and the algorithm connects it to the x_{new} . This process shows in Figure 3.

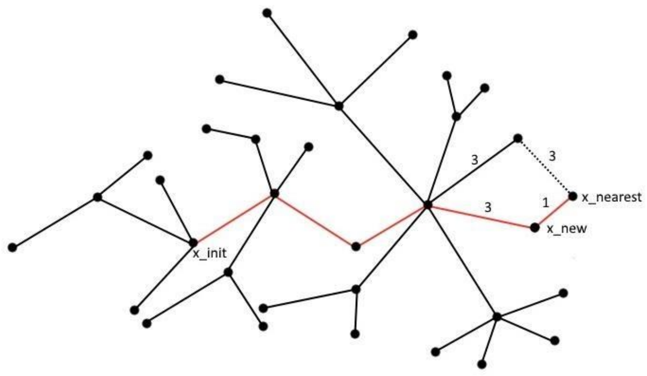

Figure 3: Rewire function

C.Turning Angle

Due to dynamic constraints, each non-holonomic vehicle (i.e., a non-holonomic mobile robot) has a limited angle (in other aspect radius) of turning, which, in turn, restricts the turning process of that vehicle. Figure 4 shows an example of this property:

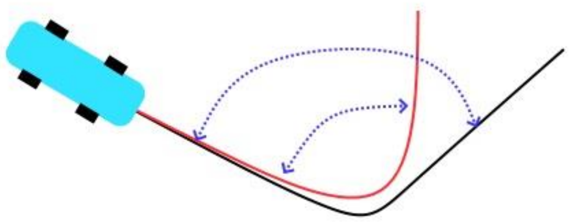

Figure 4:Turning angle of a vehicle

In figure 4 the blue vehicle is able to turn at most along the black path. However, the angle of the red path is obviously less than the angle of black turning, so it is impossible for the vehicle to go through it. To illustrate, for exploring the reason for this phenomenon, please take a look at figure 5:

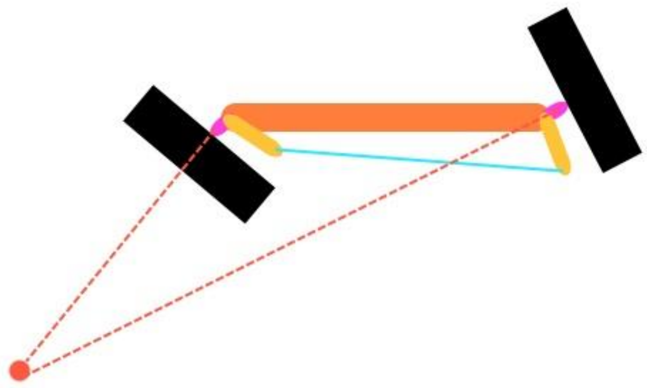

Figure 5: The design of steering systems of non-holonomic vehicles

In figure 5, the parallelogram of the steering system is obvious, by which the vehicle (like a mobile robot) can turn, but not at an arbitrary angle. In general, due to various steering systems, the turning angle will be different. However, there is something in common among all of them. They are not able to turn in discrete paths and some paths with a very low angle in their turnings. For more information, please take a look at [15] and its references.

IV. The Proposed Method

In this method, first seeking to adjust the RRT* algorithms w.r.t the turning radius of the autonomous mobile robot, and then interpolate the path by means of B-Spline interpolation is the main idea. The proposed algorithm is shown in Algorithm 2:

```
Algorithm 2    Implementation of B-Spline interpolation on RRT*

1    Input: initial node $x_{init}$ , max nodes $K \leftarrow N$
2    $V \leftarrow \{x_{init}\}; E \leftarrow \emptyset;$
3        for i=0 to K do
4    $x_{rand} \leftarrow Sample(i)$
5    $x_{nearest} \leftarrow Nearest(V, x_{rand})$
6    $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$
7    $V \leftarrow V \cup \{x_{new}\}$
8        If ObstacleFree( $x_{nearest}, x_{rand}$) then
9    $X_{near} \leftarrow Near(V, x_{new})$
10   $x_{nearest} \leftarrow Parent(X_{near}, x_{nearest}, x_{new})$
11       $E \leftarrow E \cup \{(x_{nearest}, x_{new})\}$
12       $E \leftarrow Rewire(X_{near}, E, x_{new})$
```

```
13              end if
14              end for
15          $(V, E) \leftarrow Turningconsideration(V, E)$
16              Finalpath$\leftarrow Bspline(V, E)$
17              return Finalpath
```

As shown in Algorithm 2, two new functions are added to the traditional form of RRT*. In Turningconsideration, some of the nodes are removed from RRT* according to the available turning angle of the vehicle, and then in Bspline, the left-over nodes are interpolated by B-Spline. As explained before, in this paper the aim is to use Parcur method for determining which coefficients are better for the defined B-Spline. The reason for this selection is that it is important for us to use B-Spline interpolation instead of approximation since the generated path should contain all of the points, which are parts of the RRT*'s output. Now, when the output of RRT* was created by a small number of iterations, which is not recommended, the path can be smoothed by means of just B-Spline. On the other hand, in the most practical applications using the large available number of iterations is suggested due to the accuracy of the algorithm, and in such cases as will be discussed in the results, the number of nodes are too large that the implementation of the Bspline function itself might not be able to smooth the path. This is when the next part of the algorithm is applied for the purpose of removing residual nodes from the path. The Turningconsideration function accounts for this aim. In figure 6, Turningconsideration is implemented on a given RRT* graph:

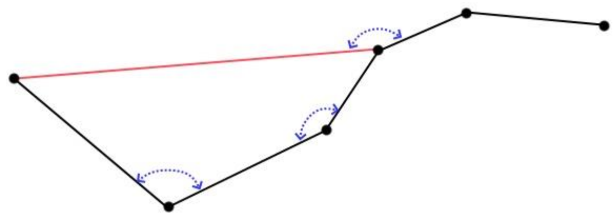

Figure 6:Turningconsideration function

In figure 6, the red line is drawn because the turning angle between the two points in both ends of it is more appropriate for the vehicle in comparison to the two nodes among them. Similarly, other nodes that disturb the angle of turning of the vehicle or robot are withdrawn, and then B-Spline method is executed.
In addition, there is something important about the proposed method. This method does not have a unique way of collision detection, so it uses the same way as RRT* for this aim.
In continue, it will be more explored how to reduce the risk of collision by combining the algorithm with other ways of path planning and AI.

V. Results

First B-Spline has been implemented and then B-spline w.r.t. the turning angle at the same time. Figure 7 shows the red path generated by pure B-spline, and figure 8 shows it after the implementation of the Turningconsideration . Then figure 9 and figure 10 show the two new paths generated by both algorithms. Then, in table I, the lengths the both paths are demonstrated. As you can observe clearly, the length of the path that considers the turning angle is less than the other, so it seems this method is feasible and optimal. In addition, in figure 10, the path generated by the new method is smoother than the previous one in the figure 10.

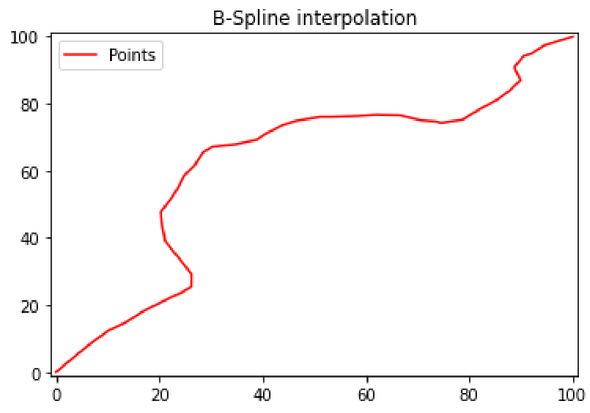

Figure 7:The output of an RRT algorithm*

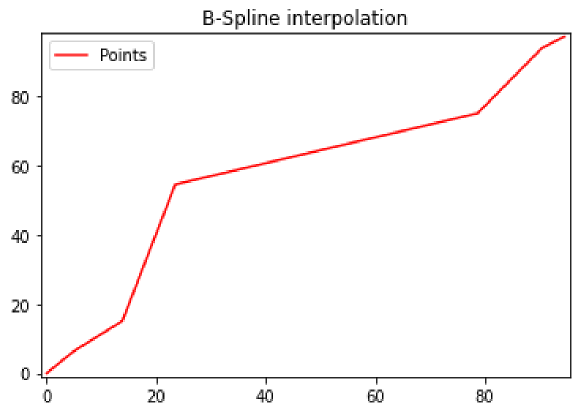

Figure 8:The output of mentioned RRT algorithm after implementation of Turningconsideration algorithm*

In both figures 7 and 8 the discrete graph-based output of the RRT* algorithm can be seen clearly. The term discrete means the path have some discontinuities, which leads to some problems for non-holonomic robots for passing it.
For addressing such problems, as shown in Figure 9, one way is to implement a B-Spline for smoothing a path. However, since usually RRT* will be executed after an adequate number of iterations, which is normally large, it causes again a path which is smoother than the former one, but still not suitable in real world. The reason is that due to the large dimensions and turning angle of most vehicles and robots, they cannot pass delicate paths as shown in Figure 9, and also a path with approximately straight lines may be preferred. In order to let the robot reach this level of compatibility, as explained before, it is better to find which nodes can be removed without being an issue, and at the same time, the turning angle of the robot should be considered.
Figure 10 shows the output after using Turningconsideration function.

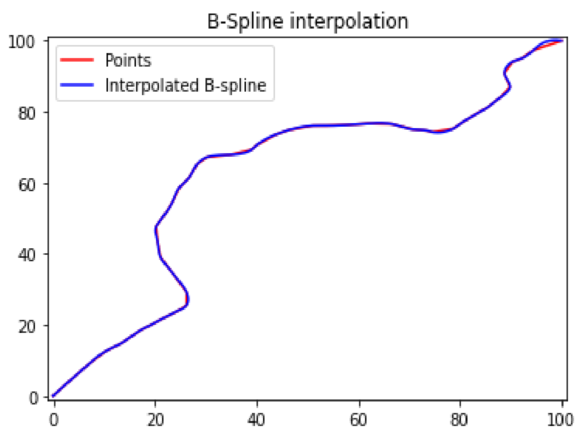

Figure 9:Implementation of B-Spline on the RRT algorithm without considering turning angle[15]*

As mentioned before, in figure 9 B-Spline is executed on the pure RRT*, and clearly, it is not smooth enough for a lot of mobile robots according to their angle of turning.

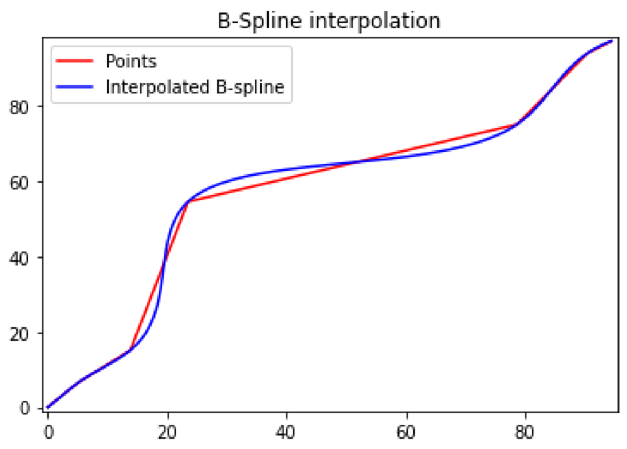

Figure 10:implementation of B-Spline on RRT with consideration of turning angle*

In the new approach in this paper, as shown in figure 10, a new function that can eliminate some residual nodes (Turningconsideration) is tried and then applied to the previous path, which was only interpolated by means of RRT* algorithm.

TABLE I. TABLE TYPE STYLES

Length of the generated curves		
Type of algorithm	***B-spline with RRT****	***B-spline with RRT* with Turning consideration***
The length of the algorithm(cm)	170.92	149.78

a. Sample of a Table footnote. (*Table footnote*)

Fig. 1. Example of a figure caption. (*figure caption*)

In Table I, as mentioned before, the length of the path with and without the implementation of the RRT* is shown. Clearly, after withdrawing residual nodes, the length will be decreased in addition to the creation of a smoother path. Since the design of steering system in some vehicles are a bit complex, then such path planning can be useful for them. Experiencing both smoothing and optimization at the same time is a considerable achievement.

VI. Conclusion

In conclusion, the presented approach can be one of the most practical methods in non-holonomic terms, especially for some known robots like rescue robots. However, there are some improvements which can be made as a suggestion in future researches. Generally, RRT* algorithms can be implemented everywhere when a path planning task is defined, and in non-holonomic cases, after using the B-Spline method, it will be completely applicable, but engineers and researchers should notice that they can enhance the way in which robot can detect its environment and obstacles. In other words, there are large numbers of ideas for being used simultaneously with RRT* such as reinforcement learning and SLAM methods.

SLAM methods itself are good approaches in order to recognize the environment by means of sensors, and reinforcement learning can do the same. Whenever they can be combined with previous and classic algorithms, they can produce a very reliable output as a path for robots, in particular mobile robots.

Finally, as a way of being sure about how practical the proposed algorithm is (and the same for other path planning methods), one of the most practical ways is using software for simulation. In this paper, Webots software has been used for this goal. An appropriate environment was defined for testing the robot as shown in figure 8:

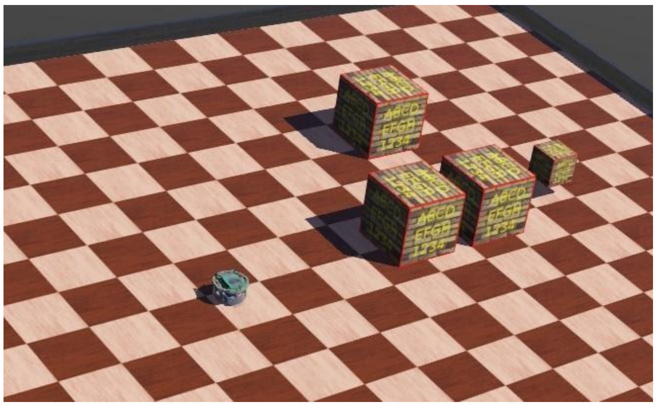

Figure 11: A typical installation for robotic path planning

In figure 8 there is a robot and some obstacles around it, which can play a role as an empirical surrounding for the robot. Indeed there are some other examples for similar aims, like utilizing ROS. Please take this fact into account that always the best policy is testing the algorithm in a real atmosphere with a suitable robot.

REFERENCES

[1] Iram Noreen, A. K. (2016). Optimal Path Planning using RRT*based Approaches: A Survey and Future Directions. *International Journal of Advanced Computer Science and Applications(IJACSA)*, Vol. 7, No. 11.

[2] S. Karaman, and E. Frazzoli, "Sampling-based algorithms for optimal motion planning", Int J Rob Res, vol. 30, pp. 846-894, 2011.

[3] M. Elbanhawi, and M. Simic, "Sampling-based robot motion planning: A review survey", IEEE Access, vol. 2, pp. 56-77, 2014.

[4] E. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces ", IEEE Transactions on Robotics and Automation, vol. 12, pp. 566-580, 1996.

[5] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning", 1998.

[6] J. J. Kuffner, and S. M. Lavalle, "RRT-connect: An efficient approach to single-query path planning", in Proceedings of IEEE International

[7] Jauwairia Nasir, F. I. (2013). RRT*-SMART: A Rapid Convergence Implementation of RRT*. International Journal of Advanced Robotic Systems, 10.5772/56718.

[8] Enzhong Shan, B. D. (2009). An Dynamic RRT Path Planning Algorithm Based On B-Spline. Second International Symposium on Computational Intelligence and Design.

[9] Kwangjin Yang, S. M. (2014). Spline-Based RRT Path Planner for Non-Holonomic Robots. Journal of Intelligent and Robotic Systems, DOI 10.1007/s10846-013-9963-y.

[10] Mohamed Elbanhawi, M. S. (2015). Continuous Path Smoothing for Car-Like Robots Using B-Spline Curves. Journal of Intelligent & Robotic Systems

[11] Priyanka Sudhakara, V. G. (2017). Optimal trajectory planning based on bidirectional spline-RRT∗ for wheeled mobile robot. Third International Conference on Sensing, Signal Processing and Security (ICSSS), (p. 10.1109/SSPS.2017.8071566).

[12] KunwooLee. (1999). Principles of CAD/CAM/CAE. PrenticeHall.

[13] Dierckx, P. (n.d.). Curve and Surface Fitting with Splines.

[14] BryanGuevara. (2018). An Overview of the Class of Rapidly-Exploring Random Trees. Utrecht University.

[15] Ettefagh, M. M.-J. (2014). Optimal synthesis of four-bar steering mechanism using AIS and genetic algorithms. *Journal of Mechanical Science and Technology*, 2351-2362.