CENG786 - Spring 2018
Term Project
RRT for six legged robot (RHex)

JeanPiere Demir
Middle East Technical University
jeanpiere.demir@gmail.com

Abstract

Motion planning algorithms for non-holonomic robots have practical and computational problems. Rapidly-exploring Random Trees are developed to handle these problems but still it is not very appropriate as a real-time robot motion planner due to huge computational time cost. Using euclidean metric based methods or do not sampling the whole state space can shorten the computational cost, if they are chosen relevantly. In this work, I tried to combine these methods with RRT to obtain a planner algorithm which is used by RHex to cross a step.

Keywords kynodynamic planning, non-holonomic robots

1 Introduction

Non-holonomic robots can not move snappily over rough terrain as animals due to differential constraints. Also, limited mobility of the robot is currently a preventing factor for robots to been utilized more in society [1]. Motion planner algorithms,which can achieve agile locomotion, have to be applied to the robot to handle this problem. In this work, I presented a bit developed version of RRT to increase the agility of the robot while crossing a step. As a robot I chose planar RHex which is explained in [2] detailedly. Also, the development of the RRT mostly is based on [3]. In this work, firstly, I explained RRT and RG-RRT in Section 1.1,1.2. Thereafter, I explained planar RHex briefly in Section 1.3. In Section 2, I gave the details of my implementation. Later on, I compared the results in Section 3. Finally, I finished my work by concluding all done works in Section 4.

1.1 Rapidly-exploring Random Trees

Rapidly-exploring Random Tree was introduced as a data structure and sampling methodology for non-holonomic/kinodynamic motion planning algorithms [4]. The main advantage of RRT is its sampling scheme to quickly cover high-dimensional configuration spaces which have differential and algebraic constraints. The algebraic constraints defined as movement limitations in workspace due to existence of obstacles and differential constraints defined as velocity/acceleration constraints due to nonholonomy of the robot and actuator limitations [5]. The main idea of the RRT is to bias the sampling of state space toward unexplored parts of the workspace and iteratively pull the search tree toward these unexplored parts of the workspace.

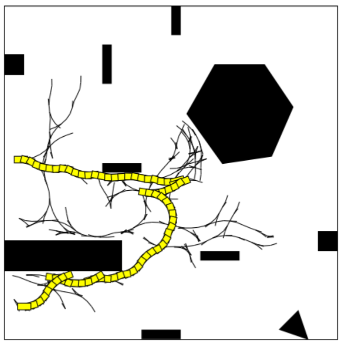

Figure 1: Generated RRT for the car which is allowed to move in forward and reverse direction. The figure is taken from [4].

As you can see from Figure 1-2, the generated tree changes according to different kinodynamic limitations.

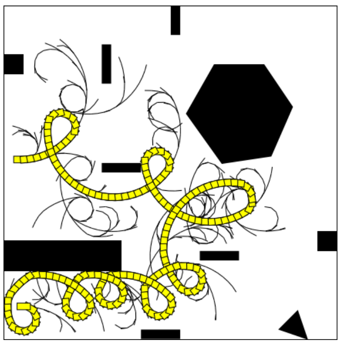

Figure 2: Generated RRT for the car which is allowed to turn left in varying degrees. The figure is taken from [4].

Additionally, RRT structure has some important properties such as,

- An RRT always remains connected due to tree structure,
- The whole path can be generated without requiring the ability to steer system between two or more predefined states
- An RRT is probabilistically complete under very general conditions

1.2 Reachable Guided-RRT

RRT is very fast and appropriate for certain applications. However, if governing equations of motion restrict the direction which tree grows up, the generated tree will not find an appropriate path for non-holonomic robot in the given environment [1]. To deal with this problem, Shkolnik et. al. (2010) developed a modified version of the RRT algorithm which is called Reachability-Guided RRT. In this algorithm, reachability-guidance biases tree grow direction toward portion of state space which are locally-reachable from the tree [6].

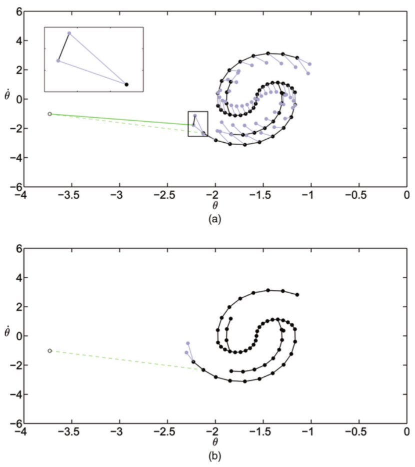

Figure 3: Two sequential steps for RG-RRT generation of underactuated pendulum. (a) a random sample is drawn from state space and the closest euclidean distance is calculated between random point and tree or reachable set. (b) the algorithm then expands the tree towards closest reachable set point if reachable set is closer, else it rejects the current sample and generate a new random sample. The figure is taken from [6]

1.3 Planar Research Hexapod(RHex)

The considered planar hexapod model is illustrated in Figure 4. The model has a rigid boy with six compliant legs and the attachment point of legs are fixed in the body. The rigid body has m_b mass, I_b inertia. Each leg is modeled as linear spring-damper pair and each one of the legs has m_t mass. Planar hexapod model has a hybrid structure and legs of the hexapod alternates between stance and flight phases [2]. I chose RHex because it is highly mobile and agile robot and I want to show that RRT can be implemented without losing agility of a robot.

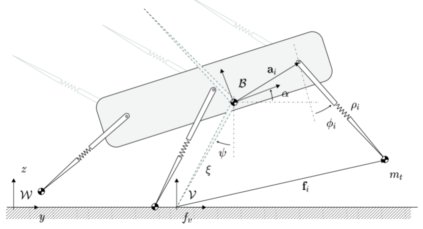

Figure 4: The planar hexapod model which is used as a non-holonomic robot. The figure is taken from [2]

In stance phase, the legs are fixed on the ground and all leg forces acted upon rigid body. Moreover, the motion of each leg is governed by the dynamics of the associated leg force. On the contrary, when legs are in flight phase they do not apply any force to the rigid body, so velocity and position of each toe become independent from the overall dynamics. As it is explained in [2], the continuous dynamics are simply dynamics of a planar rigid body under the influence of external forces generated by compliant legs. The equations are given by:

$$\begin{aligned} m\ddot{b} &= \sum_{i=1}^{6} s_i \cdot F_i \\ I\ddot{\alpha} &= \sum_{i=1}^{6} s_i \cdot (f_i - b) \times F_i \\ \ddot{f}_i &= (1 - s_i) \cdot F_i \end{aligned} \tag{1}$$

where f_i is each one of the toe position, b is body vertical/horizontal position, α is body angular position, and finally F_i is each one of the leg force.

2 Implementation

In this section, I explained the detail of my own implementation. Firstly, I gave some information about the simulation environment that is used to check collisions and generate path is appropriate or not in section 2.1. Later on, I explained my overall pseudo code and I gave some details about some of the used functions in section 1.2.

2.1 Simulation Environment

In this section, I explained simulation environment which simulates the hybrid dynamics of planar RHex and checks the generated path is appropriate or not.

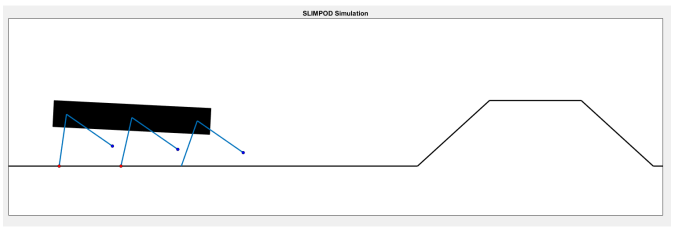

Figure 5: Matlab based simulation environment for planar Hexapod

The simulations run on Matlab, by applying ode45 (Variable-step Runge Kutta) solver to solve governing equations of motion. The model has a hybrid dynamic nature which means that it has different dynamic model/equations for different phases and alternates between these phases. Figure 5 illustrates the overall structure of the simulation environment. Current controller of planar RHex can handle only flat ground to move. So, I added a bump to the ground which is not passable by using current controller parameter values. The purpose of RRT planner algorithm is to pass this bump and reach to a certain goal position in the given environment. The bump can be seen clearly in the Figure 5.

2.2 RG-RRT like Planner Algorithm

Algorithm 1 illustrates the overall structure of the planner algorithm. Almost all kind of RRT algorithm have similar structure and they are consisted of two parts which are generating the random sample according to work/state space and checking the random sample if it is feasible or not. In this algorithm, I followed almost same structure, but I added several different parts to increase the performance of the algorithm. These additions have been done according to [1].

Algorithm 1: RG-RRT like planner algorithm for planar RHex

```
Result: Generated Tree for Planar RHex
T ← $InitializeTree()$
while goalIsAchieved do
    randSamp ← $RSG()$
    R ← $RGFinder()$
    $(dist_T,\ index_T) \leftarrow NearestState(randSamp, T)$
    $(index_R,\ index_R) \leftarrow NearestState(randSamp, R)$
    if $index_R < dist_T$ then
        continue;
    end
    else
        $(dist_T,\ index_T) \leftarrow NearestState(R($index$_R)$, T);
        $N_{prev} \leftarrow T($index$_T)$;
        controller ← $RCG($N$_{prev})$;
        $(N_{new}$, isFeasible) ← $SI($N$_{prev}$, controller);
        if isFeasible then
            T ← $InsertNode(T,$N$_{prev},\ N_{new}$, controller);
        end
    end
end
```

where RCG is RandomControllerGenetator, RSG is RandomSampleGenerator, SI is SolveInput.

The algorithm can be understood intuitively. The algorithm samples state space randomly then it creates reachable-guidance which will bias the tree grow direction according to differential limitations. The NearestState(sample, tree) function calculates shortest distance between sample and given tree and return closest point of tree to the sample, distance. If reachable set is closer than tree, the NearestState calculates distance between point $(R(index_r))$, which is closest point to the random sample in reachable set, to tree. On the contrary, if tree is closer, it rejects the current sample and it generates a new sample. Later on, the algorithm chooses returned closest point from the tree as a previous node and generates controller values as explained in 2.3. When the algorithm has previous node and controller values, it chooses previous node values as a initial value and solves the system according to given parameters by calling SolveInput function. If isFeasible flag is true, it inserts the new node to the tree. These procedure is done until the robot arrives the goal point.

Moreover, as It is explained in section 1.3, planar RHex's state space is 30 dimensional and its action space is 3 dimensional. In the beginning, I was investigating whether the whole state space has to be sampled or not, but after figuring out that the algorithm can find appropriate tree/path even the whole sampling strategy depends on several state space parameters, I based my sampling strategy according to first 6 parameters of the state space which are $[y_{body}, z_{body}, \theta_{body}, \dot{y}_{body}, \dot{z}_{body}, \dot{\theta}_{body}]$.

2.3 Controller Sampling

In the beginning of the project, I tried two different randomly sampling methodology for sampling controller space. Firstly, I generated random samples by putting minimum/maximum limit, but there was not any connection between previous node controller parameters. As it is expected, the result was unsatisfying because the robot is changing its state unexpectedly and it is starting to jump or robot's leg is starting to turn very fast. The reason of this problem is there is not any connection between previous node controller parameters. To handle that, I derived a second method which is giving a initial controller value to the root node of the tree and trying to find appropriate path by biasing these initial value. The biasing procedure has be done by adding Gaussian noise to the previous node controller values. The Gaussian noise can be show as N(0,σ) which means that it has zero mean and a specific standard deviation value.

$$controller_{curr} = controller_{prev} + \eta \tag{2}$$

where η is the Gaussian noise and controller is a struct which attributes are Buehler Clock controller values (t_c, t_s, ϕ).

3 Results & Discussions

In this section, I gave results of my implementation and discussed about them. Even the simulation environment has some deficiencies, the implementation is worked well. I compared two different situations which are controller values are constant and controller values are generated according to RRT respectively. I showed that the path which is generated by RRT like algorithm can handle bumps on the road and the robot is not losing its agility even the controller values change. Generated trees showed in Figure 6-7. Generated paths showed in Figure 10-13.

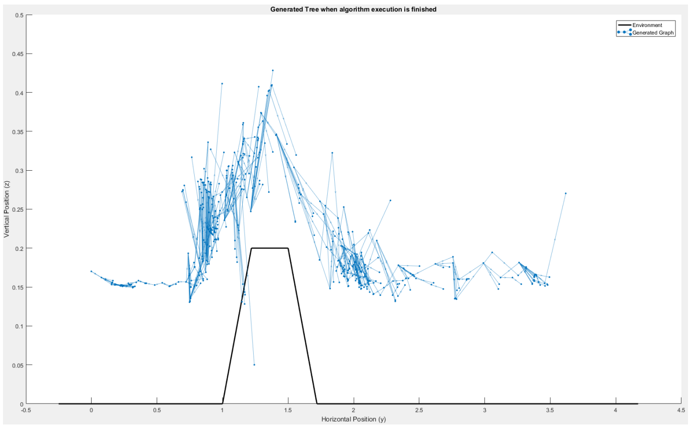

Figure 6: Generated Tree has 483 nodes and some of the nodes is colliding with environment due to some deficiency in simulation environment. The generated path showed with color blue

As you can see from Figure 7, the path to followed is collision-free and body position follows the environment texture.

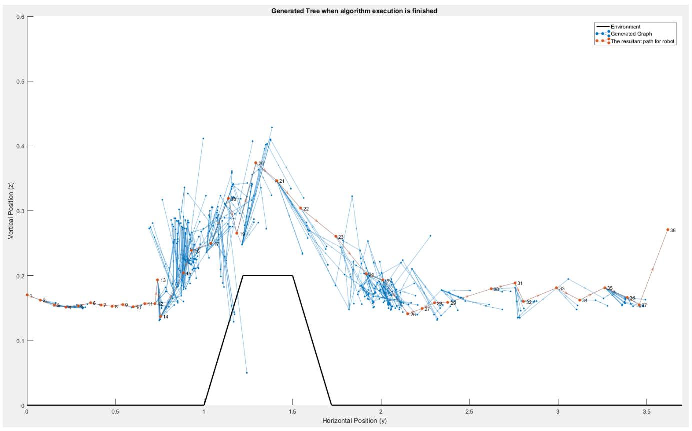

Figure 7: After generation of tree, the path to follow is generated according to MatLab shortestPath function. The shortest path showed with color red.

If we can see from Figure 6 and 7, the generated path is smooth, connected and does not change instantaneously.

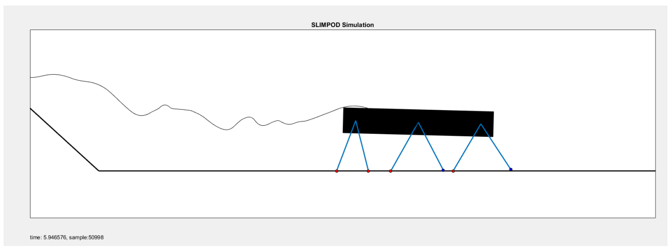

Figure 8: Followed path with RRT. The robot arrived the goal successfully

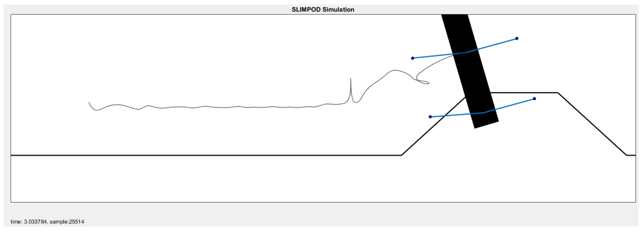

Figure 9: Followed path without RRT. The robot falled while trying to reach the goal.

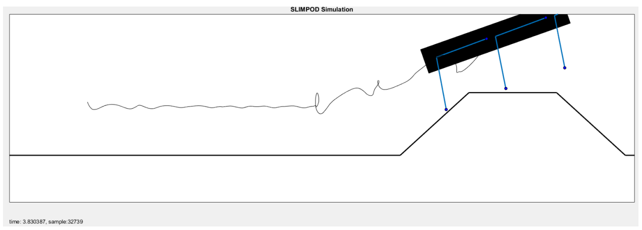

Figure 10: Body vertical vs horizontal position plot for robot uses RRT planner. The red plot is showing us the followed path over the bump.

The Figure 10 and 9 are showing us the end of the path and as it is shown, the robot is arriving the goal in Figure 10 and falling in Figure 9. Moreover, I showed the path followed over the bump in Figure 11 and 12. Here, the red plot is showing us the trajectory followed on the bump and it is approximately same with the environment texture in both of the figures.

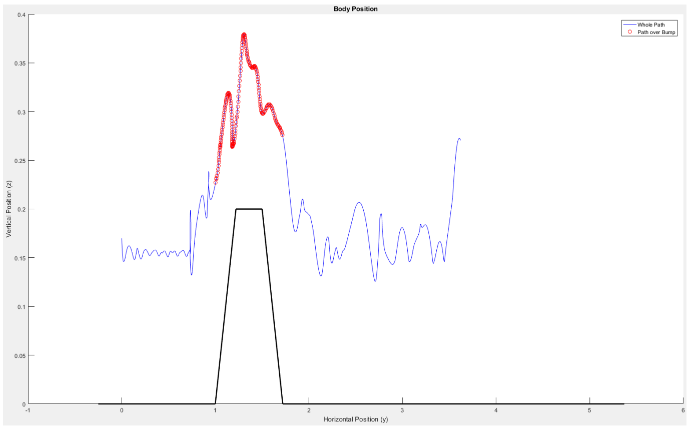

Figure 11: Body vertical vs horizontal position plot for robot does not use RRT planner. The red plot is showing us the followed path over the bump.

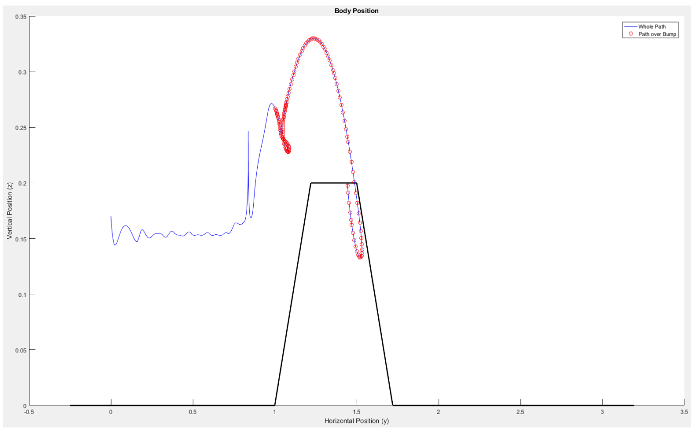

Figure 12: Body angle vs horizontal position plot for robot uses RRT planner. The red plot is the climbing angle, the green plot is going down the hill angle.

Figure 13 is showing us the trajectory of body angle while passing the bump when RRT is used. It can be seen that, the red plot is the body angle while climbing the incline and it is approximately monotonous decreasing. On the other hand, green plot is showing us the going down to the decline and body angle has monotonous increasing structure.

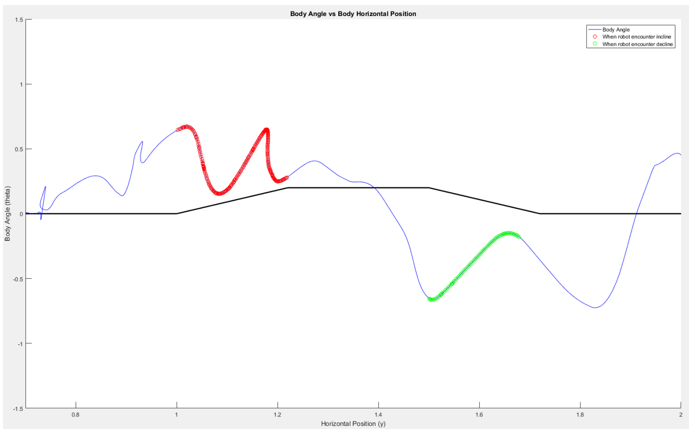

Figure 13: Matlab based simulation environment for planar Hexapod

4 Conclusion

In this work, I tried to implement RRT for a highly mobile robot in a slightly rough terrain and I developed RRT a bit instead of using traditional RRT. To create the roughness, I added a bump to the ground.

When constant controller values are applied to the robot controller, the robot could not generate appropriate path from starting point to the goal point. So, RRT which is one of the sample-based motion planning algorithm is applied to find a solution by sampling some partition of the state space according to some limitations. RRT based generated path could find the path and the path was smooth, appropriate for given environment.

As a future work, I will improve the simulation environment to make more real world based simulations. Also, I will investigate Buehler Clocks controller to improve the performance of controllers in RRT. I hope, I will develop an online path planner algorithm which can be applied to the robot while moving over the rough-terrain and it will work kind of reactively.

This project increased my knowledge about legged robots, sampling based motion planning algorithms. I could not investigate and find much more appropriate, reasonable intuitive controller generator and it will be my first work to do after the project.

5 Usage Manual

To generate a tree, please follow the given procedure:

- Open $slimpod\_RRT.m$ scriot
- If you want to change time steps change it from line 50
- If you want to change bump structure:
  - To change starting point of the incline, change line 25 $slimpod\_params.incStart$
  - To change ending point of the incline, change line 26 $slimpod\_params.incEnd$
  - To change starting point of the decline, change line 27 $slimpod\_params.decStart$
  - To change ending point of the decline, change line 28 $slimpod\_params.decEnd$
  - To change height of the bump, change line 30 $slimpod\_params.stepH$
- Hit the run button, to start to create a new tree for given environment

To animate a found tree, please follow the given procedure:

- $slimpod\_6leg\_animate('GeneratadMatFilePath', fps)$

References

[1] A. Shkolnik and M. Levashov, "Motion planning for bounding on rough terrain with the little dog robot," *Proc. of Int. Conf. on ...*, 2010.

[2] U. Saranli, *Dynamic Locomotion with a Hexapod Robot*. PhD thesis, Ann Arbor, MI, USA, 2002. AAI3068951.

[3] A. C. Shkolnik and R. Tedrake, "Sample-Based Motion Planning in High-Dimensional and Differentially-Constrained Systems," 2010.

[4] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *4th Workshop on Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000.

[5] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," *In*, vol. 129, pp. 98–11, 1998.

[6] A. Shkolnik, M. Levashov, I. R. Manchester, and R. Tedrake, "Bounding on rough terrain with the Little-Dog robot," *International Journal of Robotics Research*, vol. 30, no. 2, pp. 192–215, 2011.