Animating Motion Planning Topics

Peter Murray
Robotics Engineering
Worcester Polytechnic Institute
Worcester, USA

I. Introduction

Motion planning is a uniquely visual field of robotics whose fundamental ideas are inherently spatial and geometric in nature. Among these ideas are planning algorithms, a method used to find some path between a start and a goal in physical space, and the concept of a configuration space, a complete specification of a given robot's parameters which can be interpreted as a n-dimensional plot. I have taken on the challenge of both animating and explaining these two ideas in the form of a simplified *configuration space* as well as the *rapidly exploring random tree* planning algorithm as first developed by LaValle in 1998 [1]. These were chosen for their relative simplicity and visual structure.

Throughout this project, I make use of *Manim Community*, a powerful python library initially built by Grant Sanderson for simple mathematical animations. This gives a great deal of programmatic control and design uniformity throughout the project.

II. Lesson 1: Configuration Space

To introduce the concept of the configuration space, I first attempted to extract information about a simple 2R robot, and make simple connections to plot their configuration both in physical space, then in configuration space. To unite the physical space to the configuration space, I animated 'unwrapping' both θ and ϕ (the first and second angle of our 2R robot). This is achieved by transforming the circle formed by θ into a straight line, as seen in Figure 1. Do the same for ϕ, and you can form a two-dimensional plot as seen in the left of Figure 2. The white dot represents a pose of the robot: θ and ϕ. The line is simply a visual aid to track the path of the dot over time.

To make this more interesting, I have been able to implement different kinds of obstacles and robots into SE2EZ, a program written in C++ which has implemented many different kind of motion planning algorithms. It comes with a configuration space visualizer, plotting both the robot and its obstacles onto a 2D plane in real time. Creating obstacles in SE2EZ and pulling their visualiztions into my animation is a clear diretion to improve this explaination. More unique visualizations could include stacking 2d plots and creating a 3D volume which might represent a 3R robot's configuration space.

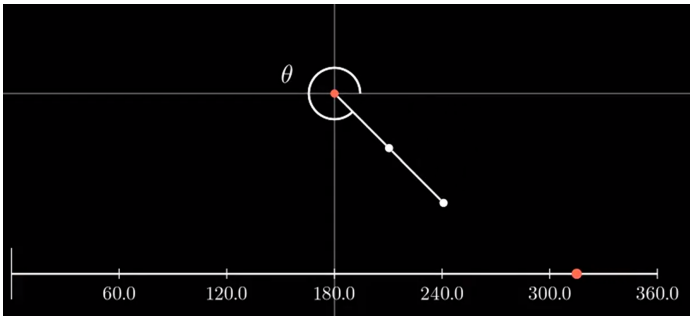

Fig. 1. "Unwrapping" joint 1.

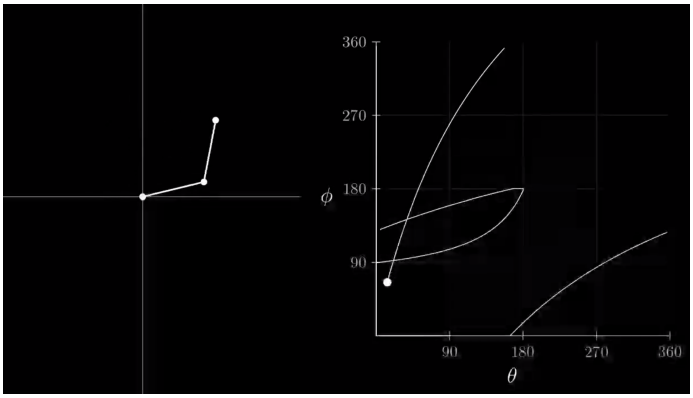

Fig. 2. Plotting a 2R robot with a 2D configuration space.

III. Lesson 2: Rapidly exploring random tree

The rapidly exploring random tree (RRT) algorithm is a particularly compelling sampling-based planner that 'grows' a tree from some start position to some goal position in an n-dimensional space, and is often included in introductory motion planning courses. Myself and my advisor, Prof. Constantinos, have developed a three-step plan to introduce RRT by incremental explaination.

1) Intuition: To ease our audience into an initial introduction to the topic, I animated a complete RRT growth based on a simple premise: "say you are a student at their home apartment, and you would like to find a path to your class." This grounds the otherwise ephemeral visual plot with white shapes representing obstacles to a tangible and relatable reality: a student going to class. With this premise, I grow a pre-planned RRT tree from the edge of the map to a building in the map, showing off RRT's capabilities to grab the audience's

attention.

Fig. 3. Initial RRT intuidion tree growth.

2) RRT Basics: In this section, we introduce the basics of RRT, step by step. First, we establish q_{start}, and q_{rand}. Drawing a line from one to the other, we extend some distance d, establishing the step size of our algorithm. Having established a q_1, we continue, but this time q_{rand} always establishes a connection to the nearest node in the tree. This is made clear with a growing circle in one step of the explanation. Following this, further nodes are added, repeating the structure of introducing q_{rand}, drawing a line towards closest node, and establishing a new node at some distance d. For one step we cover the edge case of q_{rand} existing at a closer distance than d, and in this case we establish the new node at q_{rand}. At the end of this section we explain what a voronoi bias is, and draw each node's voronoi cell color coded for size (and probability of choosing q_{rand} within that cell). This can be seen in Figure 4 below.

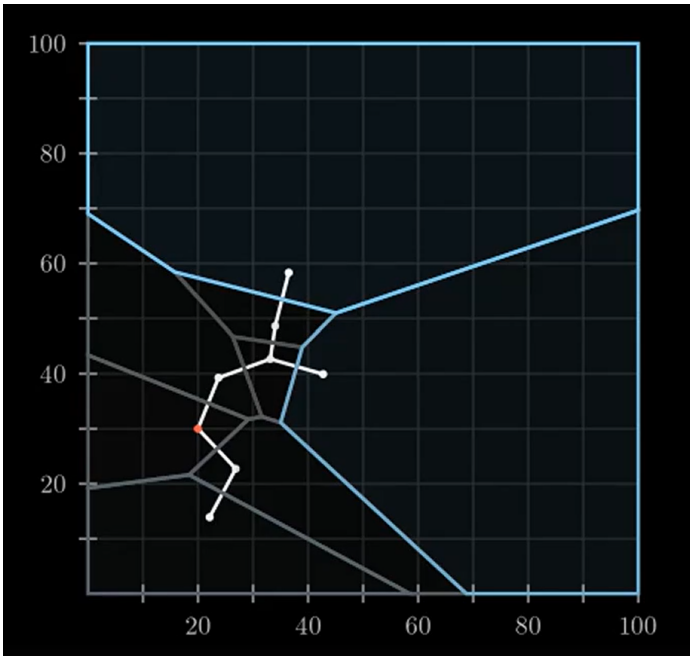

Fig. 4. 'RRT Basics' section after drawing all Voronoi cells.

3) Application: The third and final section to this explanation is the application of the concepts learned in section 2 to a 'real' scenario. By re-introducing the obstacle plot in section 1, we show our workspace in a new light. This time, we take the same stepwise approach as 'RRT Basics': first introducing q_{rand}, drawing a line towards closest node, and establishing a new node at some distance d. This time, we must account for obstacles. First, that q_{rand} is capable of being chosen within an obstacle, but produces a valid q_{new} so long as q_{new} is not in collision. We also draw a potential q_{new} that *is* within an obstacle, drawing a cross over it to clearly mark it as invalid. To explain the goal bias, we show pause the execution of the tree over an iteration with the goal bias shown: q_{rand} becomes q_{goal} for that iteration, which only happens for some percentage of nodes. An iteration where this happens is seen in Figure 5 below. This tree is continually grown until q_{goal} is found.

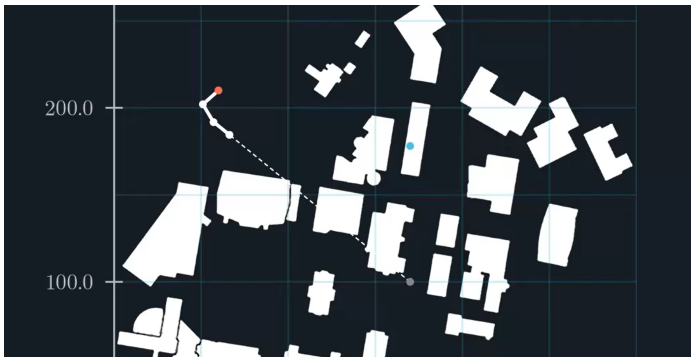

Fig. 5. 'Application' section during a goal bias iteration.

REFERENCES

[1] S. LaValle (1998) Randomly Exploring Rapid Trees: A New Tool For Path Planning . [Online]. Available: http://msl.cs.uiuc.edu/ lavalle/papers/Lav98c.pdf

[2] Manim Community Edition. (n.d.). Manim Community Documentation. Retrieved December 22, 2023, from https://docs.manim.community/index.html