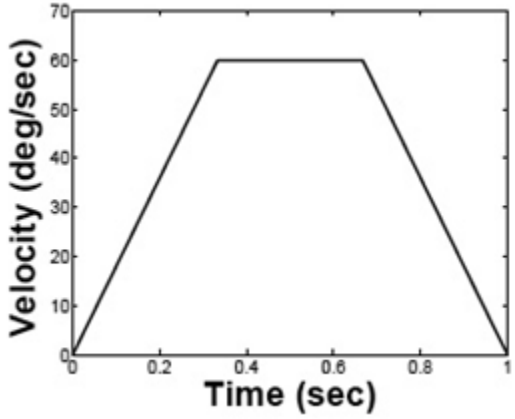
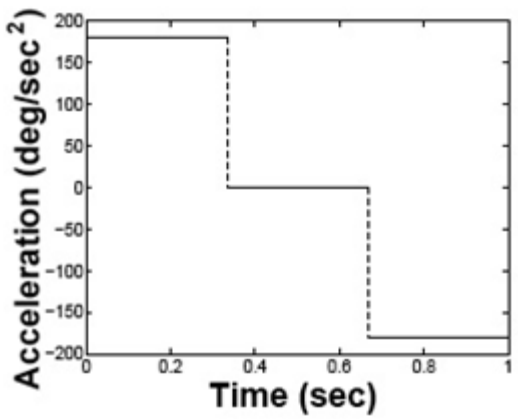
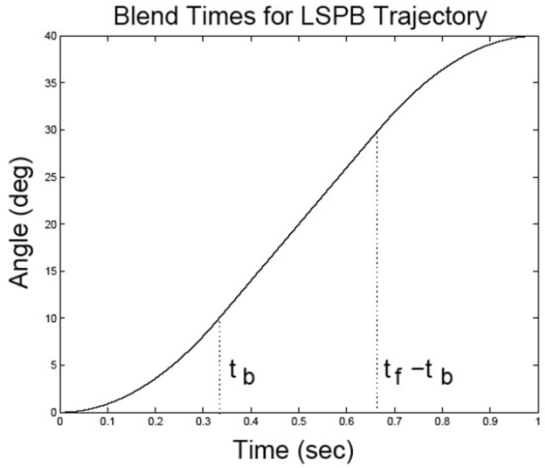
Intro to Robotics

Lecture 13

Linear Segments with Parabolic Blends (LSPB)







Interpolating Polynomials or Blending Polynomials

- Multiple via points
- Continuity constraints: velocity and accelerations
- Initial and end times

- Given initial and final times, t_0 and t_f , respectively, with

$$q^d(t_0) = q_0 \quad ; \quad q^d(t_f) = q_1$$
$$\dot{q}_d(t_0) = q'_0 \quad ; \quad \dot{q}^d(t_f) = q'_1$$

The required cubic polynomial $q^d(t)$ can be computed from

$$q^d(t_0) \quad = \quad a_0 + a_1(t - t_0) + a_2(t - t_0)^2 + a_3(t - t_0)^3$$

- Parameters can be computed

$$a_2 = \frac{3(q_1 - q_0) - (2q'_0 + q'_1)(t_f - t_0)}{(t_f - t_0)^2} \qquad a_3 = \frac{2(q_0 - q_1) + (q'_0 + q'_1)(t_f - t_0)}{(t_f - t_0)^3}$$

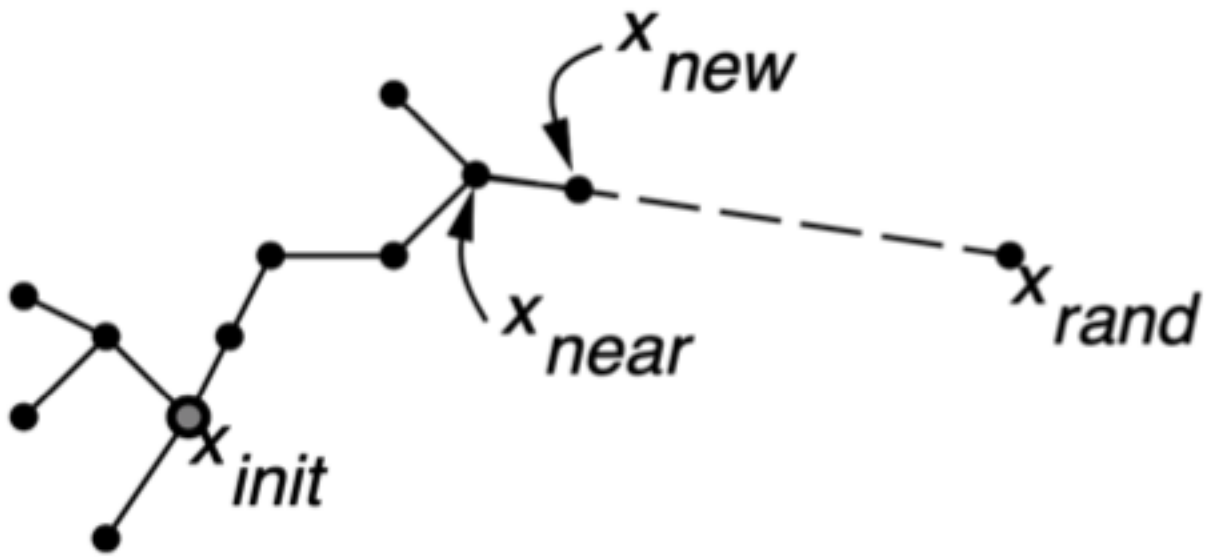
- A sequence of moves can be planned using the above formula by using the end conditions q_f, v_f of the ith move as initial conditions for the i+1st move.

Velocity and Acceleration Limitations

- Time warping

Rapidly Exploring Random Trees (RRT) -- Basic Idea

The result is a tree, $\mathcal{T}$, rooted at x_{init}.

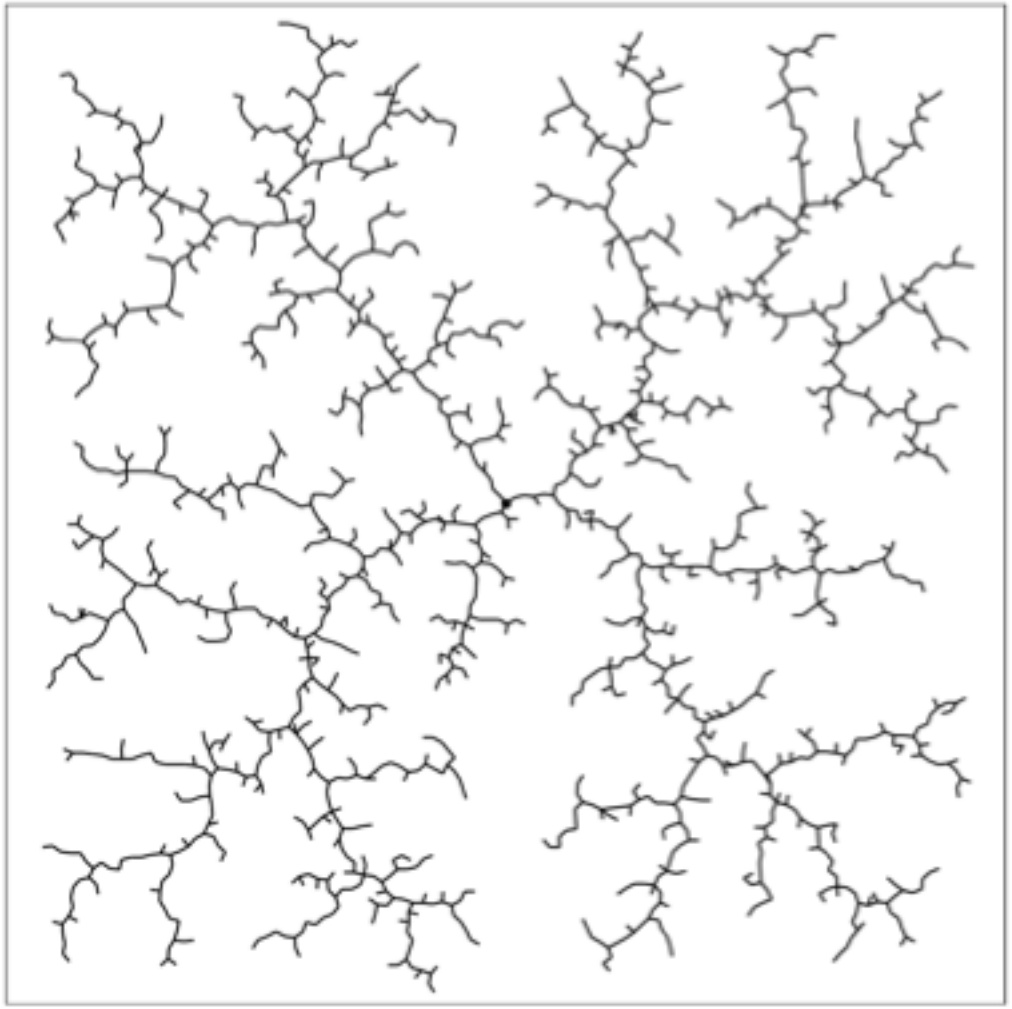


Basic Algorithm

GENERATE_RRT($x_{init}, K, \Delta t$)

1 $\mathcal{T}$.init(x_{init});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

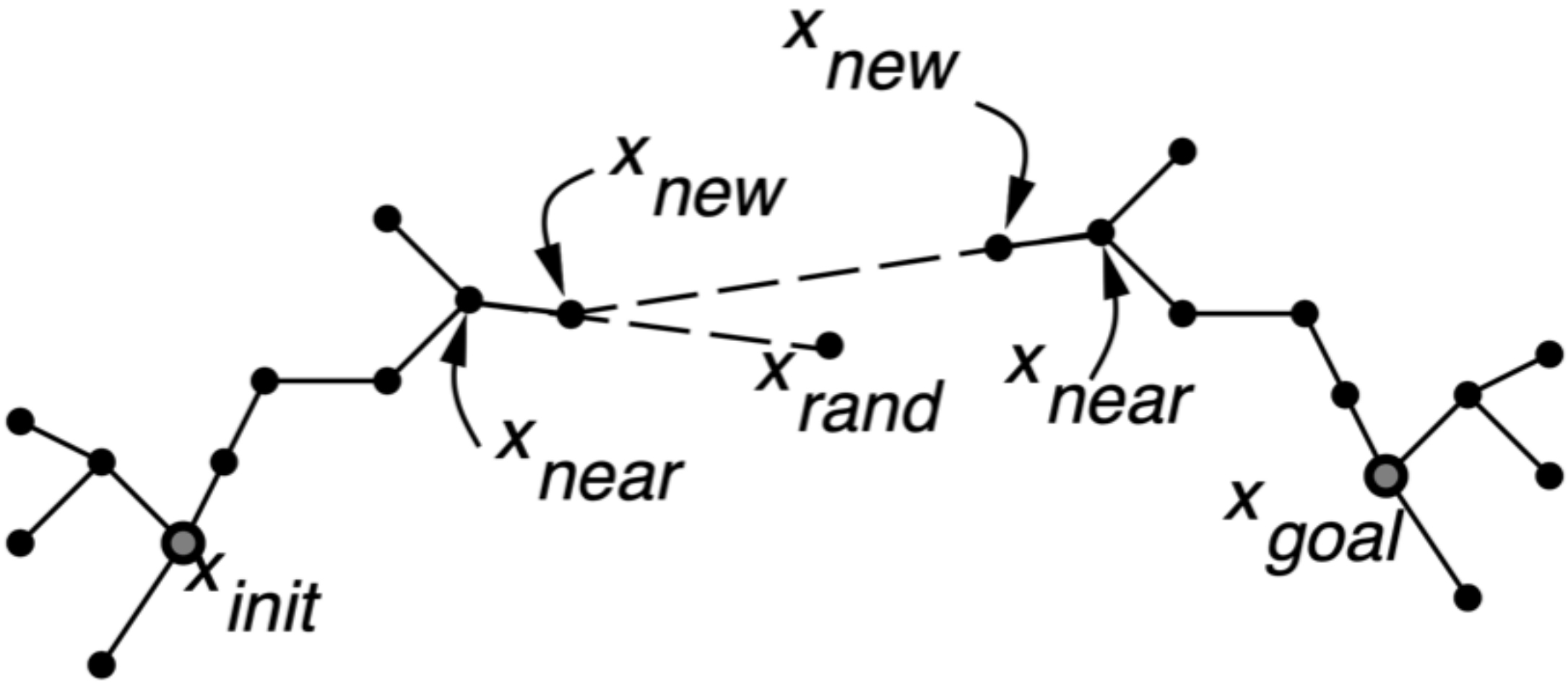
Outcome after Interations



Dual RRT

- Tree 1 has a new node using random target (RRT) – N_t
- New node becomes target for tree 2
- Find a node tree 2 that is nearest to the target in tree 1 – N_n
- Add a new node/edge in tree 2 based on the nearest node N_n and the target node N_t

Dual RRT



Bias toward Goal

- When generating a random sample, with some probability pick the goal instead of a random node when expanding
- 5-10% is the right choice

Improvements

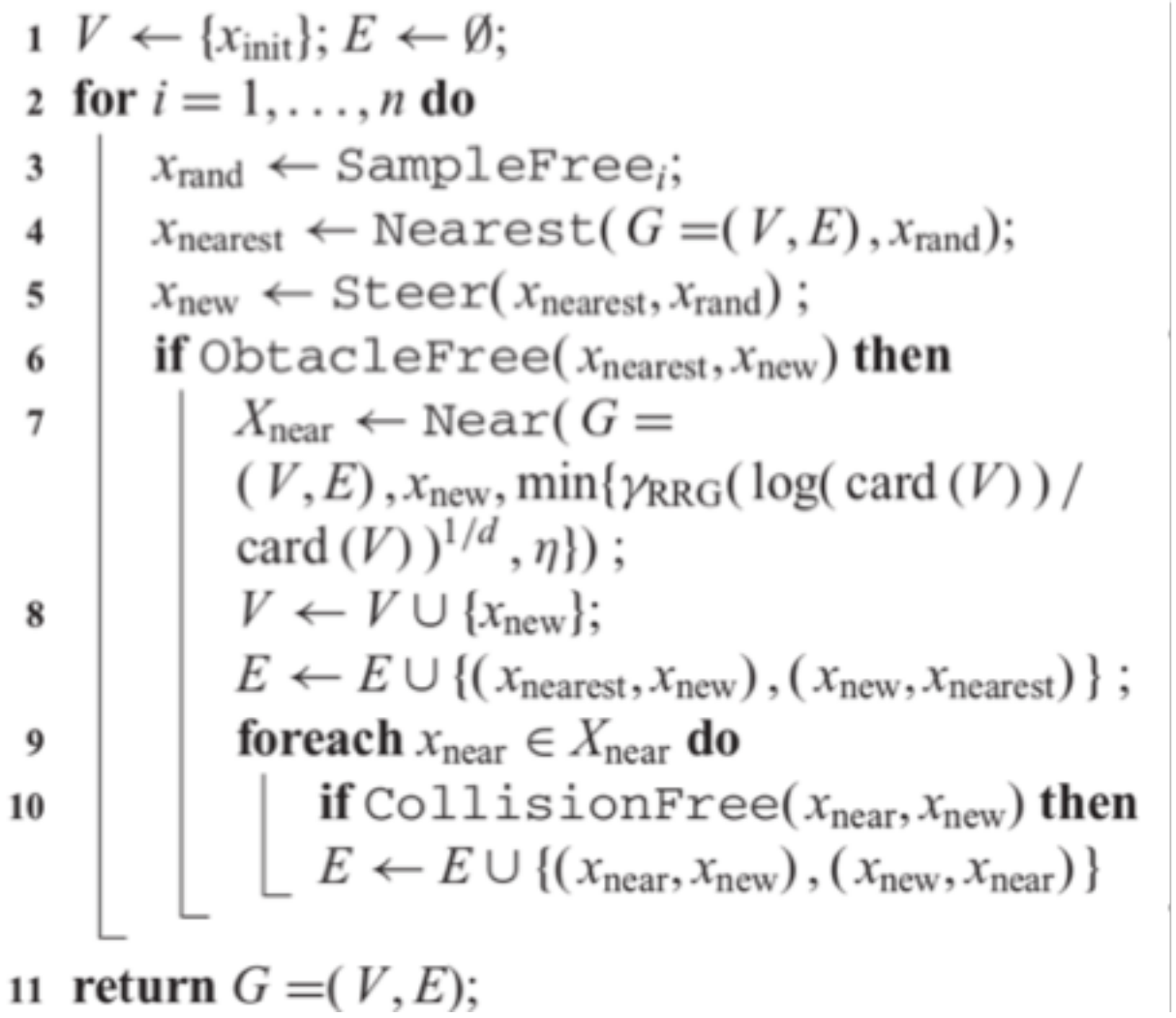
- Get u based on features of kinodynamic systems
- Keep expanding till reach to the random or goal, or have a collision
- Do smoothing before using the path
- Shortcuts: along the found path, pick two vertices xt1, xt2 and try to connect them directly (skipping over all intermediate vertices)
- Allows to specify an objective function that includes smoothness in state, control, small control inputs, etc.

RRT Summary

- Advantages
  - Single parameter
  - Balance between greedy search and exploration
  - Converges to sampling distribution in the limit
  - Simple and easy to implement
- Disadvantages
  - Metric sensitivity
  - Nearest-neighbor efficiency
  - Unknown rate of convergence
  - “long tail” in computation time distribution

RRG

- Attempt to connect to every vertex within a radius r of x_new



RRT*

- RRT* records the distance each vertex has traveled relative to its parent vertex as the cost of the vertex
- After the closest node is found in the graph, a neighborhood of vertices in a fixed radius from the new node are examined. If a node with a cheaper cost() than the proximal node is found, the cheaper node replaces the proximal node.
- Rewiring of the tree
- After a vertex has been connected to the cheapest neighbor, the neighbors are again examined. Neighbors are checked if being rewired to the newly added vertex will make their cost decrease. If the cost does indeed decrease, the neighbor is rewired to the newly added vertex. This feature makes the path more smooth.
- RRT* takes longer time to compute

RRT* Pseudo Code

```
Rad = r
G(V,E) //Graph containing edges and vertices
For itr in range(0…n)
    Xnew = RandomPosition()
    If Obstacle(Xnew) == True, try again
    Xnearest = Nearest(G(V,E), Xnew)
    Cost(Xnew) = Distance(Xnew, Xnearest)
    Xbest, Xneighbors = findNeighbors(G(V, E), Xnew, Rad)
    Link = Chain(Xnew, Xbest)
    For x' in Xneighbors
            If Cost(Xnew) + Distance(Xnew,x') < Cost(x')
                    Cost(x') = Cost(Xnew) + Distance(Xnew, x')
                    Parent(x') = Xnew
                    G += {Xnew, x'}
    G += Link
Return G
```