# **Mini-Project**

Algorithamic Motion Planning

**Arpit Savarkar**

‘On my honor, as a University of Colorado at Boulder student, I have neither given nor received unauthorized assistance on this work.’

## 1 What space will you choose for motion planning for this vehicle? Define the topology and dimension of this space and state your reasons for this choice.

The motion planner for the drone would be undertaken in the (state)-space. A point [1] in the State-Space Space includes both configuration parameters and velocity parameters (i.e, it is the tangent bundle of the configuration space).

Let X denote the state space. i.e,

$$X : (x, y, z, \psi, \theta, v) \tag{1}$$

According to few definitions in citation [2]

- A feasible(kinodynamic) planning problem P = (X,U,I,G,F,B,D) asks to produce a trajectory y(s) : [0,S]→X and control u(s) : [0,S]→U

$$\begin{aligned} y(0) = x_I \quad (InitialState) \\ y(1)\ \epsilon\ G \subseteq X \quad (GoalState) \\ y(s)\ \epsilon F\ \ \subseteq X\ \forall s\ \epsilon\ [0, S] \quad (KinematicConstraints) \\ u(s)\ \epsilon B(y(s))\forall s\ \epsilon\ [0, S] \quad (ControlConstraints) \\ y'(s)\ \ = D(y(s), u(s))\forall s\ \epsilon\ [0, S] \quad (DynamicEquation) \end{aligned} \tag{2}$$

- An optimal planning problem P= (L,Φ,X,U,xI,G,F,B,D) asks to produce a trajectory(s) : [0,S]→X that minimizes the objective functional:

$$C(y) = \int_0^S L(y(s), u(s))ds + \Phi(y(S)) \tag{3}$$

  In addition to 2. Here L is the incremental cost and Φ is the terminal cost.

- A probabilistically-complete planner A finds a feasible solution to a problem P, when one exists, with probability approaching 1 as more time is spent planning. A planner A is asymptotically-optimal for the optimal planning problem P if the cost C(t)of the generated path approaches the optimum C* with probability 1 as more time t is spent planning.

- Theorem [2]

  The asymtotically optimal Planning problem P = (L,Φ,X,U,xI,G,F,B,D) is equivalent to state-c cost optimal planning problem withour incremental costs P= (0,ˆΦ,X×R+,U,(xI,0),G×R+,F×R+,B,ˆD), in such a way that solutions to ˆP are in one-to-one correspondence with solutions to P. Here the terminal cost ˆΦ is given by

$$\Phi^{(}[^{x}_{c}]) = c + \Phi(x) \tag{4}$$

And the dynamics ˆD are given by

$$z' = [^{x'}_{c'}] = [^{D(x,u)}_{L(x,u)}] \tag{5}$$

[2] The proof is straightforward, showing that the projection of solutions toˆPonto the first dim(X)elements are solutions to P, and solutions to P can be mapped to solutions toˆP via augmenting them with a cumulative cost dimension. Note that even if every state in the original space was reachable,not all points in the state-cost space are reachable. Moreover,the goal set is now a cylinder with infinite extent in the cost direction.

- Topology of the robot in C-Space

$$R^4 * S^1 * S^1 \tag{6}$$

- Dimension of the robot in State Space is 6
- State-Space is used during the implementation of the Algorithm
- Justification :

  The drone is fully orientable model, with translation abilities within the constraints of the room. The drone motors controls allow it to translate, and allow it to orient in the state space with states to sample within the velocity constraints and Roll Angle Constraints.he pose(x,y,z) and velocity are subsets of the R and 2 orientation degree of freedoms (S) form the state space based on systems dynamics.

## 2 Describe the advantages and disadvantages of each method of motion planning listed below for your problem.

When planning in C-Space, differential or non-holonomic constraints arise from the presence of one or more rolling contacts between rigid bodies, or from the set of controls that it is possible to apply for the drone. But when planning in State-Space, non-holonomic constraints also arise from conservation laws

### 2.1 Gradient descent planner with a potential function

According to [1], adapting randomized holonomic planning techniques to the problem of (free) State Space. The potential field method appears nicely suited because a discrete-time control can repeatedly be selected to reduce the potential. The primary problem is that dynamical systems usually have drift, which could easily cause the robot to overshoot the goal, leading to oscillations. Without a cleverly-constructed potential function(difficult nonlinear control problem), the method cannot be expected to work well. The problem of designing a good heuristic function becomes extremely complicated for the case of kinodynamic planning. Gradient Descent along Potential Function is not a trivial task. This algorithm approach is prone to local minima. Chattering Problems also do exist in the Vanilla Implementation of the Gradient Descent in Potential Functions due to cancellations of repulsive and attractive potentials. Additionally, due to large number of hyper-parameters to tune, in a even higher dimensional workspace implementation can take time. Basically, the construction of ideal potential fields could be probably difficult and time consuming. Furthermore, in higher dimensional workspaces, we might need to sample robot configurations, and the related transforms On the Positive side, potential field ensures that the robot always moves in the direction of the goal, and do not require post-processing techniques to optimize the solution for better path. Navigation potential functions have the following properties

- resolves the local minima problem
- difficult to construct
- may be possible in sphere space
- require hand tuning of parameters to find a balance between local minima and non-isolated critical points
- can map star-shaped spaces to sphere spaces. All of which are difficult in 6D state space.

Challenge in using potential functions:

- Non-Euclidean C-space hard to construct
- Use relationship between forces in workspace and C-space
- Define potential functions in workspace and map to C-space
- Using gradient decent for motion planning can lead to local minima issues.

## 2.2 Wavefront Planner

Wavefront algorithm basically consists of a breadth-first search (BFS) on the graph induced by the neighborhood connectivity (adjacency) of cells, starting at G, which is assigned the value 2 at the beginning. As BFS traverses the space, each cell is assigned a value which corresponds to the number of moves required for the shortest path from that cell to the goal.

The grid is an approximation of the space, so the gradient is an approximation of the actual distance gradient. Accounting for dynamics of the model designing a particular norm between two grids cells, can become computationally expensive. Grid based discretization requires time and space exponential in the dimension of the State-Space along with Sampling Controls within bounds accounting for dynamics. The notion of gridsize is not consistent along all the dimensions are of the state space. As some of the dimensions have bounds, they need to be accounted for in the grid size, for better heuristic development for BFS search. Accounting for the dynamics can be difficult in the BFS (A*) search would again without a cleverly-constructed heuristic (difficult nonlinear control problem), the method cannot have an upper bound on time. The problem of designing a good heuristic function becomes extremely complicated for the case of kinodynamic planning.

On the other hand if creating ESDFs or Euclidean Distance Transforms(EDTs) of 2D and 3D occupancy information is available. The main restriction of this approach is the fixed-size voxelgrid, which requires a known map size and a large amount of memory. But Voxel based wavefront planners, take advantage of Signed Distance fields as input data, and additionally allow the ESDF map to dynamically change size, and can can be used for path planning.

•Can be generalized into higher dimensions

•Can become computationally intractable in higher dimensions.

•Discretization-based approach and uses only Attractive Potential. (removes obstacles)

## 2.3 Probabilistic Roadmap

In the probabilistic roadmap approach, a graph is constructed in the configuration space by generating random configurations and attempting to connect pairs of nearby configurations with a local planner that will connect pairs of configurations. The probabilistic roadmap technique might require the connections of thousands of configurations or states to find a solution, and if each connection is akin to a nonlinear control problem, it would impact many problems of differential constraints. Furthermore, the probabilistic roadmap is designed for multiple queries. The underlying theme in that work si that is worthwhile to perform substantial pre-computation on a given environment, to enable numerous path planning queries to be solved efficiently. Additional constraints on bounds need to be accounted for by either placing the robot in the configuration space, or if that is computationally expensive, checking for collisions in the workspace for state and velocity bounds. The probabilistic roadmap technique is also amenable to kinodynamic planning. The primary requirement is the ability to design a local planner that will connect pairs of configuration that are generated at random. Steering Functions (Trajectory generation) is needed to solve a two point boundary value problem, for a purely kinematic model, Thus Control function manipulation (that is linearization or non-linear controller) is needed for exact solution, which cannot be solved analytically, and numerical solutions increase computational cost.

Probabilistic Roadmap area probabilistically complete algorithm, if a path exists, the probability of not finding it $\rightarrow 1$ as number of samples $\rightarrow$. PRM-based planners aim to construct a roadmap that captures the whole connectivity of the configuration space. Performance of sampling-based planners depend on the properties of the C-space.

•This algorithm sacrifices optimality for completeness even though there are techniques to make the search itself more optimal. Once the PRM is constructed, efficient graph search algorithms can be em-ployed to query through the node to find the find path.

•PRM can sometimes be difficult to find path through narrow corridors.

•To optimize for path length, post-processing steps are necessary, vanilla implementation of PRM does not give optimal path, smoothing would be needed to use.

•Constructing PRM also helps us find path not just from start to end goal but from any node to other.

## 2.4 Randomized tree-based planner

RRT Based planners are optimum balance between potential field heuristic analysis and uniform exploration of state space like in probabilistic roadmap methods. For the extension to kinodynamic planning, the next state of the drone can be calculated using the Runge Kutta approximation. Extending the tree, with 5 percentage, goal bias. For Holonomic based planning, the distribution of RRT vertices converge in probability to distribution that is used for sampling, even in non-convex spaces. (regardless of initial state) [1].

Approximate nearest neighbours can be used to tackle the problem of high dimentionality in state space. Special sampling techniques with cost based heuristic can be employed around narrow environments with large number of obstacles. To handle the drone for drift and other dynamics constraints, RRT* sampling can be implemented by sampling an action that brings the velocity components of x as close as possible toward the random sample.

Generating Local Trajectories, between the start node is done by sampling, and need not be done for all the possible states as required by PRM or Gradient Based Potential field methods. Thus we dont have to solve a final boundary value problem. A sampling pose saves us on computation time.

rapidly Exploring Random tree for the obstacles and workspaces, we are working in we get shorter path lengths and faster time convergence. Adding more samples to the C-space can create more optimum path lengths for the graph which is not the case in PRM which samples node all over the C-Space. Thus time taken to converge, is shorter in comparison to PRM.

1.BasicRRT, takes only one small step when adding a new branch.

(i)Tree is pulled towards random directions based on the uniform sampling of states.

(ii)RRT relies on nearest neighbours and distance metric to grow trees towards the node.

(iii)RRT adds Voronoi bias to tree growth space.

2.RRT suffers from the following :

(i)Large number of configurations, increases computational cost.

(ii)It becomes increasingly difficult to guide to guide the tree towards previously unexplored parts of the free configuration space .

(iii)Metric sensitivity still exists but sampling control inputs work faster than PRM.

## 3 Design an appropriate motion planner for the drone. Present the planner with a detailed pseudocode and explain each line. Make sure you define all the necessary elements.

Consider the following : Let X denote the state space. and x belongs to X

$$x : (x, y, z, \psi, \theta, v) \tag{7}$$

And the motion model (Velocity Space of the Kinematics of the drone is) i.e,

$$X' : \begin{cases} x' = vcos(\psi)cos(\theta) \\ y' = vsin(\psi)sin(\theta) \\ z' = vsin(\theta) \\ \psi' = \omega \\ \theta' = \alpha \\ v' = a \end{cases} \tag{8}$$

with,

$$Control\ Constraint : \begin{cases} \omega = u_1 \ \epsilon \ [\frac{-\pi}{6}, \frac{\pi}{6}] \\ \alpha = u_2 \ \epsilon \ [\frac{-\pi}{6}, \frac{\pi}{6}] \\ a = u_3 \ \epsilon \ [-0.5, 0.5] \end{cases} \tag{9}$$

that is,

$$X' = f(x, u) : \begin{cases} x' = vcos(\psi)cos(\theta) \\ y' = vsin(\psi)sin(\theta) \\ z' = vsin(\theta) \\ \psi' = u_1 \\ \theta' = u_2 \\ v' = u_3 \end{cases} \tag{10}$$

with additional constraints of

$$\begin{cases} \theta \ \epsilon \ [\frac{-\pi}{3}, \frac{\pi}{3}] \\ v = u_2 \ \epsilon \ [-1, 1] \end{cases} \tag{11}$$

Motion Update Equations (Descriptive)

$$
\begin{aligned}
v_{newState} &= (a) * \Delta t \\
\theta_{newState} &= \alpha * \Delta t \\
\psi_{newState} &= (\omega) * \Delta t \\
x_{newState} = \int_0^t & v_{newState} * cos(\theta_{newState}) * cos(\theta_{newState}) dt \\
y_{newState} = \int_0^t & v_{newState} * sin(\theta_{newState}) * sin(\theta_{newState}) dt \\
z_{newState} = \int_0^t & v_{newState} * sin(\theta_{newState}) dt
\end{aligned} \tag{12}
$$

Where the brok subscript represents the updated parameters post dead motor

Where the Integration can be performed according to Runge-Kutta Method as follows

$$
\begin{aligned}
x(\Delta t) \approx x(0) + \frac{\Delta t}{6}(w_1 + 2w_2 + 2w_3 + w_4), \\
where, \\
w_1 = f(x(0), u), \\
w_2 = f(x(0) + \frac{\Delta t}{2} w_1, u), \\
w_3 = f(x(0), + \frac{\Delta t}{2} w_2, u), \\
w_4 = f(x(0) + \Delta t \ w_3, u)
\end{aligned} \tag{13}
$$

## 3.1 Setup for the Algorithm

### 3.1.1 Cost Function

The Cost Function here is interchangeable with the term "distance between two nodes". The role of this function to create a heuristic that can be used in the motion planning algorithm further. The cost function/distance is culmination of the following

- Since the drone has to move around in the environment that can have obstacles and has a start pose and a goal pose. Both the pose constraints are added to the motion planning algorithm. The cost of the start start and goal pose constraints is expressed as a function of the state vector S and G

$$
H_1 : \begin{cases} h_1(x_s) = f(x_s, 0) - p_s \\ h_2(x_g) = f(x_g, 0) - p_g \end{cases} \tag{14}
$$

where f() is the forward integration of the motion model (Undertaken using the Runge Kutte Method) which maps any configuration toa workspace and p the desired pose of S (start) and G (goal)

- Collision Avoidance

  According to [4] As in CHOMP Algorithm [3] drones body is simplified to be a set of spheres, and the distance of the body to any point in the state-space is given by the norm of the body to the center of the sphere minus its radius. The obstacles are represented by pre-computed Signed Distance Field matrices. The norm is positive if a point is outside of the obstacle, zero if its on the surface, and negative when its lies inside the obstacle. This method can involve shifting the Drone to the configuration space for faster calculation.

  Hence, the obstacle cost function is obtained by computing the signed distance of a robot at a state

$$H_2 : \begin{cases} h_3(x_i) = c_{sdf}(f(x_i, u_i)) \end{cases} \tag{15}$$

- Done Specific Kinematic Constraints [1] Forward Kinematic Constraint, post state update. Here f() is the forward kinematics and pc is the desired position of the drone post state update

$$H_3 : \begin{cases} h_4(x_i) = f(x_i, u_i) - p_c \end{cases} \tag{16}$$

- Closeness Definition A simple metric on State space based on a weighted Euclidean Distance for position, linear and angular velocities and acceleration input along with a weighted metric to be less than or equal to unity is considered. This cost attempts to heuristically encode a measure relative to "closeness" between two states sampled from state space with a positive scalar function.

$$H_4 : \begin{cases} h_5(x_i, x_{i+1}) = w_p||(x_i, y_i, z_i) - (x_{i+1}, y_{i+1}, z_{i+1})||^2 \\ +w_{q1}||\psi_i - \psi_{i+1}||^2 \\ +w_{q2}||\theta_i - \theta_{i+1}||^2 \\ +w_v||v_i - v_{i+1}||^2 \\ +w_{cost}||(H1_i + H2_i + H3_i) - (H1_{i+1} + H2_{i+1} + H3_{i+1})||^2 \end{cases} \tag{17}$$

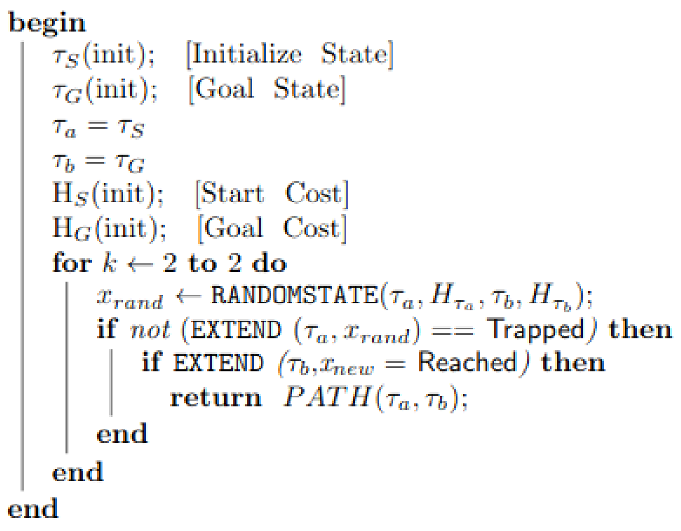
  All the weights will be normalized to be between [0,1]

- $$H : H_1 + H_2 + H_3 \tag{18}$$

### 3.1.2 Motion Planner

As discussed in Q1 Theorem[2] , state space and (state-cost) space are equivalent. And thus a cost of each state of can be calculated for reference to develop a graph.

Figure 1: Bidirectional Planning Algorithm



```
begin
    $\tau_S$(init);   [Initialize State]
    $\tau_G$(init);   [Goal State]
    $\tau_a = \tau_S$
    $\tau_b = \tau_G$
    H$_S$(init);   [Start Cost]
    H$_G$(init);   [Goal Cost]
    for $k \leftarrow 2$ to 2 do
        $x_{rand} \leftarrow$ RANDOMSTATE($\tau_a, H_{\tau_a}, \tau_b, H_{\tau_b}$);
        if not (EXTEND ($\tau_a, x_{rand}$) == Trapped) then
            if EXTEND ($\tau_b, x_{new}$ = Reached) then
                return  $PATH(\tau_a, \tau_b)$;
        end
    end
end
```

Kinodynamic Planners are used for developing a motion plan for the drone. A Bidirectional Tree Based Sampling RRT* based algorithm is developed

1. Initialization

$$\begin{gathered} \tau_S(init); H_S(init) \\ \tau_G(goal); H_G(init) \end{gathered} \tag{19}$$

   Explanation :

   - S(init) is the Initial State from which the drone starts and the cost for that particular state is calculated. Similarly G(goal) is the goal state where the robot can land in. Since Kinodynamics planners assume that the goal is not a state but a region, hard bounds according to constraints exist for velocity and $\Theta$regions.
   - RRT Bidirectional divides the computation time between two processes, exploring the state space and trying to grow the trees into each other. $\tau$are two trees maintained at all times until they become connected and a solution is found.

2. Sampling, Extend

   Explanation :

Figure 2: Extend Functionality for Bidirectonal Tree based RRT*

```
begin
    EXTEND $(\tau, x)$
    $x_{near} \leftarrow$ NEARESTNEIGHBOUR$(\tau, x)$
    $x_{new_cost} \leftarrow$ COST(x$_{near}$)
    for $i \leftarrow 2$ to 2 do
        $SET(u_{near}) \leftarrow$ SAMPLECONTROL(x$_{new}, x_{near}$)
        CONTROLSET $(x_{near}, u_{near}, cost_{near}) \leftarrow$ COST(SET(u$_{near}$)$, x_{near}$)
    end
    $x_{new}, u_{new} \leftarrow$ MINCOST(CONTROLSET(x$_{new}, u_{near}, cost_{near}$))
    if NEWSTATE $(x, x_{near}, x_{new}, u_{new})$ then
        $\tau$.addVertex(x$_{new,}$)
        $\tau$.addEdge(x$_{new}$)
        $\tau$.addCost(x$_{new}, u_{new}$)
        if $x_{new} == x_{goal}$ then
            return Reached
        ;
        else return Advanced;
    end
    return Trapped
end
```

- A state of the robot is sampled from within the velocity and Θregions from the state space.
- In each iteration, one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex.
- To prepare for extension of the tree in kinodynamic planning , the next state which is the

$$\begin{aligned} x(t) &= x_0 + \int_0^t f(x(\tau), u(\tau))d_\tau \\ such\ that\ \forall\ t\epsilon[0,T]&:\ VALID(x(t)) = TRUE \\ \epsilon\ t\epsilon[0,T]&:\ VALID(x(t)) = TRUE \end{aligned} \tag{20}$$

  which forms the next state by sampling a control from the control space at random which forms the node of the graph, and adding an edge between the two nodes.
- As shown in Figure 2 , the Extend function selects the nearest vertex already in the RRT to the sample. The "nearest" vertex is chosen according to Equation 18.
- The function NEW STATE() makes a motion update toward the new sample state x by applying an sampled input for some time increment delta t.
- NEW STATE implicitly uses a " Collision Detection " According to equation 18 being calculated at each time step to satisfy equation 23 according to the motion model specified at Equation 10

- If the New State is successful, the new state and input are represented as x New and u New .
- According to citation [1] Three situations can occur: (i) Reached, in which the new vertex reaches the sample x for a sampled input within the error bounds to this Sampled state. (ii) Advanced, in which a new vertex x New Not equal to x is added to the RRT or (iii) TRAPPED, in which case the NEW STATE fails to produce a state that lies in the Free State Space.
- RANDOMSTATE(): Samples a state from the state-space boundaries

3. Growth and Connect Samples

   (a) Along the steps as shown in figure 1 one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex. The Nearness can be measured from the cost of each node which can be calculated according to equation 15 or 16. Then, the roles are reveresed by swapping the two trees. The algorithm as defined in Citation [1] is a minor variation tot the vanilla Bidirectional RRT*. The algorithm attempts to grow the trees into each other half of the time.

   (b) NEARESTNEIGHBOUR() :
   One of the key bottlenecks in construction of RRTs has been nearest neighbour computations. The development of efficient neighbour for high dimensional state space involves computing a nearneness coefficent based on minimum cost. Since Control Inputs are sampled only once, Cost of a state forms a additonal check over Decremental cost approach undertaken in the algorithm.

   (c) SET(u Near): A Set data structure which stores the sampled controller values and resets after every iteration to store unique sampled controller parameters

   (d) SAMPLECONTROL( x New, x Near): Samples the control parameters from the controller boundaries, A Clipping function that prevents that over-rides the Velocity values between [-1, 1] if the acceleration input is too large for sampling. Similarly for yaw

$$\begin{aligned} if\ v\ \epsilon[-1,1] &\Big\{ v = a * \Delta t \\ if\ v\ !\epsilon[-1,1] &\Big\{ v = max(min((1, max(-1, a * \Delta t))) \end{aligned} \tag{21}$$

$$\begin{aligned} if \theta\ \epsilon[-\frac{\pi}{3}, \frac{\pi}{3}] &\Big\{ \theta = \alpha * \Delta t \\ if\ v\ !\epsilon[-1,1] &\Big\{ \theta = max(min((\tfrac{\pi}{3}, max(-\tfrac{\pi}{3}, \alpha * \Delta t))) \end{aligned} \tag{22}$$

(e) CONTROLSET(x Near, u Near, Cost near): Stores the next possible set of controller inputs, and its respective state from state-space and the equivalent cost.

(f) MINCOST(): Extracts the (state, controller) pair from relate to the lowest cost from the iteration

4. Steering Method for Graph

   As mentioned in Citation [1] One drawback of using a bidirectional approach is that a discontinuity in the trajectory will generally exist at the place in which the two trees connect. A number of techniques like Classical Shooting techniques can be applied to either half of the trajectory. It might be possible to slightly perturb the starting point of the second half of the trajectory to force it to begin at the end of the first half of the trajectory. In this case, the second half of the trajectory would have to be reintegrated and tested for collision.

   A Polynomial Spiral can be constructed with the velocity, pitch, roll and acceleration bounds on the uniformly sampled bounds to be given to drone controller for input.

5. PATH(): Stores the Path as state trajectory to fed into the controller from Fig 1.

## 4 Can you guarantee that your planner will find a trajectory for every given obstacle course? Justify your answer.

The modified implementation of Asymptotically-optimal meta-planning with a Randomized RRT* forms a amalgamation of Incremental Cost Optimal Planning Algorithm and RRT*. Since RRT* is a asymptotically complete planner for a feasible kinodynamic planning problem., then Bounded-Suboptimal Plan within error bounds terminates in finite time and produces a path with no Cost more than C*, which is the optimal cost.

Proof:

Let i+1 be the index of the final iteration. In the prior iterations of the run of the algorithm, a solution was found which may be greater than or equal to the most optimal Cost solution. If in the current iteration, no solution is found, but since RRT* is asymptotically complete, the current Planning problem, violates itself to not find a solution until max time is reached. Running time is finite since each loop reduces the cost by at the some epsilon value, and hence the inner loop is run no more than

$$\frac{c_0 - C}{epsilon} \tag{23}$$

According to citation [2] The need for a complete planner is too restrictive for practical use on high-dimensional problems, where only probabilistically complete planners are practical. Here, we relax this restriction. Under the un-restrictive assumption that the cost is lowered by a non-negligible fraction whenever we find a feasible path.

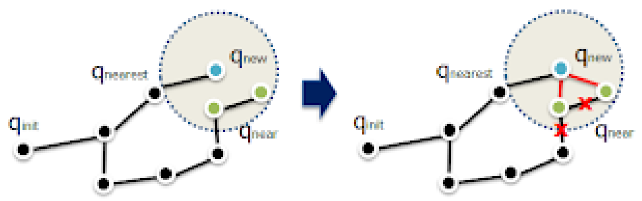
Thus In Conclusion. The motion planner is only Probabilistic complete as is RRT*.

## 5 The competition also keeps track of the time that each drone takes to complete the course. Are you be able to modify your algorithm to optimize for travel time? Justify your answer.

According to Kinodynamic Planning Analysis . Specifically in the "Sampling and Extend" Sub section of the algorithm. Approach 1 of Control Sampling is undertaken where in , a control input is sampled and an EXTEND and branch step is taken. Based on a Cost definition consisting of (Cost of a state based on pose), (Collision Avoidance kinematic constraints) and (Closeness Definition).

Based on Equation 32 it can be seen that all the cost functions are internally weighted equally. To Optimize for time indirectly implies that the the drone should travel the shortest distance in the room from start to goal. Thus we need to weight the pose weights higher. Additionally Since the costs are computed per sampled state. Optimize the Control Sampling for more than 1 control sample. That is we optimize for "m" control inputs samples, And based on the nearness Cost function, we opt for the sample which has a lower cost per state, calculated according to Equation 18.

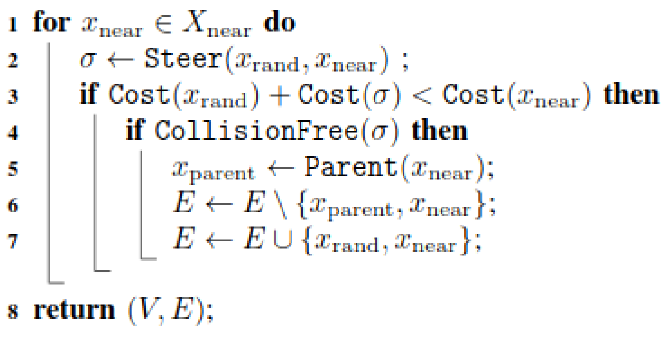
Figure 3: Optimizing for Time (SMOOTHING)- Citation [5]



Additionally, Smoothing techniques such as Rewire can be implemented after the Multiple control input sampling based on optimum cost at the between Steps 6 and 7 of Figure 1 can be applied post the graph search as shown below.

Justification: Equal Distribution Amongst all the weights of the cost function leads to development of incremental costs in such a way that a good enough distance is kept away from the obstacles also. To optimize for time, we can reduce the dyanamics impact due to drift, and model kinematic evolution. And tune the weights belonging to Spatial pose and velocities. We optimize the decremental costs of the states such that we get better costs all over. Furthermore, Smoothing techniques applied in configuration space can be employed for faster times, disregarding the energy/battery of the drone.

Figure 4: Modified Rewire - Citation [5]



```
for $x_{\text{near}} \in X_{\text{near}}$ do
    $\sigma \leftarrow$ Steer($x_{\text{rand}}, x_{\text{near}}$) ;
    if Cost($x_{\text{rand}}$) + Cost($\sigma$) < Cost($x_{\text{near}}$) then
        if CollisionFree($\sigma$) then
            $x_{\text{parent}} \leftarrow$ Parent($x_{\text{near}}$);
            $E \leftarrow E \setminus \{x_{\text{parent}}, x_{\text{near}}\}$;
            $E \leftarrow E \cup \{x_{\text{rand}}, x_{\text{near}}\}$;
return $(V, E)$;
```

## 6 You have just been notified that there are ropes hanging from the ceiling of the ASPEN Lab. The exact locations of the ropes are not given, but you are assured that they are located along a given straight line (vertical plane) across the lab. If the drone hits a rope, one of its motors will be damaged. In that case, the drone continues flying by turning off the damaged motor, which will cause a reduction in its speed and a positive or negative bias in its yaw depending on which motor is turned off. Describe the dynamics of the drone in this environment.

The initial State of the drone before it hits the rope are

$$x : (x, y, z, \psi, \theta, v) \tag{24}$$

And the motion model (Velocity Space of the Kinematics of the drone is) i.e,

$$X': \begin{cases} x' = vcos(\psi)cos(\theta) \\ y' = vsin(\psi)sin(\theta) \\ z' = vsin(\theta) \\ \psi' = \omega \\ \theta' = \alpha \\ v' = a \end{cases} \tag{25}$$

As this drone is a multi-motor system, and hitting the rope, leads to turning off the damaged motors, causing reduction in speed and a positive or negative bias depending on which motor is turned off. Assuming the control system is aware of the fact the motor is turned off.

$$x : (x, y, z, (\psi \pm \epsilon_2), \theta, (v \pm \epsilon_1)) \tag{26}$$

And the motion model (Velocity Space of the Kinematics of the drone is) i.e,

$$ModelDynamics: \begin{cases} X': \begin{cases} x' = vcos(\psi)cos(\theta) \\ y' = vsin(\psi)sin(\theta) \\ z' = vsin(\theta) \\ \psi' = \omega \\ \theta' = \alpha \\ v' = a \end{cases} \\ X'_{brok}: \begin{cases} x' = vcos(\psi)cos(\theta) \\ y' = vsin(\psi)sin(\theta) \\ z' = vsin(\theta) \\ \psi' = (\omega \pm \epsilon_1) \\ \theta' = \alpha \\ v' = (a \pm \epsilon_2) \end{cases} \end{cases} \tag{27}$$

Motion Update Equations (Descriptive)

$$\begin{aligned} v_{newState} &= (a \pm \epsilon_2) * \Delta t \\ \theta_{newState} &= \alpha * \Delta t \\ \psi_{newState} &= (\omega \pm \epsilon_1) * \Delta t \\ x_{newState} &= \int_0^t v_{newState} * cos(\theta_{newState}) * cos(\theta_{newState}) dt \\ y_{newState} &= \int_0^t v_{newState} * sin(\theta_{newState}) * sin(\theta_{newState}) dt \\ z_{newState} &= \int_0^t v_{newState} * sin(\theta_{newState}) dt \end{aligned} \tag{28}$$

Where the brok subscript represents the updated parameters post dead motor

Where the Integration can be performed according to Runge-Kutta Method as follows

$$\begin{aligned} x(\Delta t) \approx x(0) + \frac{\Delta t}{6}(w_1 + 2w_2 + 2w_3 + w_4), \\ where, \\ w_1 = f(x(0), u), \\ w_2 = f(x(0) + \frac{\Delta t}{2} w_1, u), \\ w_3 = f(x(0), + \frac{\Delta t}{2} w_2, u), \\ w_4 = f(x(0) + \Delta t w_3, u) \end{aligned} \tag{29}$$

## 7 Design a motion planner for the drone in the ASPEN Lab with hanging ropes. First, describe your method in a simple language. Then, write a detailed pseudocode for it and provide an explanation for each line. Make sure all the components are well-defined.

As the motion model is updated to account for change in dynamics, a piecewise motion update model is used to randomize sample in an online fashion. The location of the ropes i.e the map of the environment is known before hand. Thus a constant lookup is created to keep track of the fact the robot has hit the ropes. This can be done using the Signed distance fields created above and also by checking for the obstacle Cost in the configuration space. Once the collision is detected, a rigorous upgrade is made to the Cost Objective function with the addition of the following cost

$$H_5' : \Big\{ h_5(dynamic) = w_{dyan}||f(x(t),u(t)) - f_{brok}(x(t),u_{brok}(t))||^2 \tag{30}$$

Thus the new cost function becomes

$$H = H_1 + H_2 + H_3 + H_5 \tag{31}$$

And the updated Nearness function thus becomes

$$H_6 : \begin{cases} h(x_i, x_{i+1}) = w_p||(x_i, y_i, z_i) - (x_{i+1}, y_{i+1}, z_{i+1})||^2 \\ +w_{q1}||\psi_i - \psi_{i+1}||^2 \\ +w_{q2}||\theta_i - \theta_{i+1}||^2 \\ +w_v||v_i - v_{i+1}||^2 \\ +w_{cost}||(H1_i + H2_i + H3_i) - (H1_{i+1} + H2_{i+1} + H3_{i+1})||^2 \\ +w_{dyan}||f(x(t),u(t)) - f_{brok}(x(t),u_{brok}(t))||^2 \end{cases} \tag{32}$$

The above cost function will only be used only when the obstacle has detected a collision with the rope, and all the previous cost function calculated per state will be ignored and the start and goal cost will be recalculated.

As discussed in Q1 Theorem[2] , state space and (state-cost) space are equivalent. And thus a cost of each state of can be calculated for reference to develop a graph.

Kinodynamic Planners are used for developing a motion plan for the drone. A Tree Based Sampling RRT* based algorithm is developed

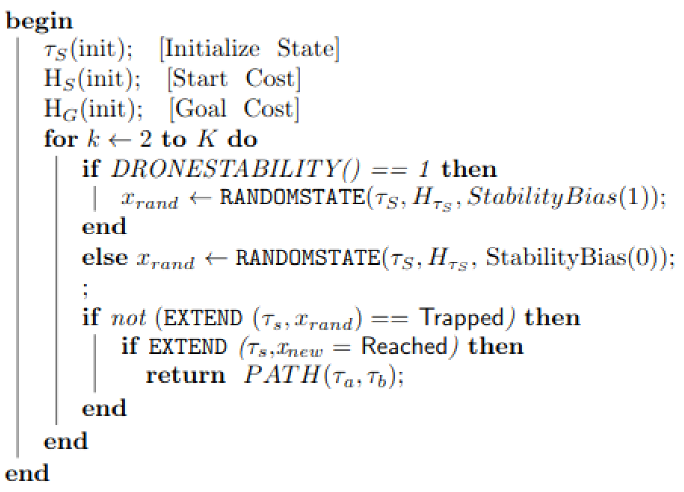
1. Initialization

$$\begin{aligned} &\tau_S(init); \\ &H_S(init) \\ &H_G(init) \end{aligned} \tag{33}$$

   Explanation :

   - S(init) is the Initial State from which the drone starts and the cost for that particular state is calculated. Similarly G(goal) is the goal state where the robot can land in. Since Kinodynamics planners assume that the goal is not a state but a region, hard bounds according to constraints exist for velocity and Θregions.
   - RRT* Algorithm is implemented here with the manipulation that the system has piece-wise dynamics

Figure 5: Modified RRT* Build



```
begin
    $\tau_S$(init);   [Initialize  State]
    H$_S$(init);   [Start  Cost]
    H$_G$(init);   [Goal  Cost]
    for $k \leftarrow 2$ to $K$ do
        if $DRONESTABILITY() == 1$ then
            $x_{rand} \leftarrow$ RANDOMSTATE($\tau_S, H_{\tau_S}, StabilityBias(1)$);
        end
        else $x_{rand} \leftarrow$ RANDOMSTATE($\tau_S, H_{\tau_S}$, StabilityBias(0));
        ;
        if not (EXTEND ($\tau_s, x_{rand}$) == Trapped) then
            if EXTEND ($\tau_s$,$x_{new}$ = Reached) then
                return  $PATH(\tau_a, \tau_b)$;
        end
    end
end
```

2. Sampling, Extend

   Explanation :

   - A state of the robot is sampled from within the velocity and Θregions from the state space.
   - In each iteration, one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex.

- To prepare for extension of the tree in kinodynamic planning , the next state which is the

$$x(t) = x_0 + \int_0^t f(x(\tau), u(\tau))d_\tau(1-\xi) + g(x(\tau), u(\tau))d_\tau(\xi) \quad (34)$$
$$such\ that\ \xi = 0\ until\ Collision\ \xi = 1\ after\ collision$$

  which forms the next state by sampling a control is undertaken from the updated NEW STATE() function considering the updated Model Dynamics and the Updated Cost Function H6 is used forms the node of the graph, and adding an edge between the two nodes.

- As shown in Figure 2 , the Extend function selects the nearest vertex already in the RRT to the sample. The "nearest" vertex is chosen according to Equation 18.

Figure 6: Extend Functionality for RRT

```
begin
    EXTEND $(\tau, x)$
    $x_{near} \leftarrow$ NEARESTNEIGHBOUR$(\tau, x)$
    $x_{new_cost} \leftarrow$ COST$(x_{near})$
    for $i \leftarrow 2$ to $N$ do
        $SET(u_{near}) \leftarrow$ SAMPLECONTROL$(x_{new}, x_{near})$
        CONTROLSET $(x_{near}, u_{near}, cost_{near}) \leftarrow$ COST(SET$(u_{near}), x_{near})$
    end
    $x_{new}, u_{new} \leftarrow$ MINCOST(CONTROLSET$(x_{new}, u_{near}, cost_{near})$)
    if NEWSTATE $(x, x_{near}, x_{new}, u_{new})$ then
        $\tau$.addVertex$(x_{new,})$
        $\tau$.addEdge$(x_{new})$
        $\tau$.addCost$(x_{new}, u_{new})$
        if $x_{new} == x_{goal}$ then
            return Reached
        ;
        else return Advanced;
    end
    return Trapped
end
```

- The function NEW STATE() makes a motion update toward the new sample state x by applying an sampled input for some time increment delta t, considering the updated system dynamics.
- NEW STATE implicitly uses a " Collision Detection " According to equation 15 being calculated at each time step to satisfy equation 34 according to the motion model specified at Equation 27
- If the New State is successful, the new state and input are represented as x New and u New .
- According to citation [1] Three situations can occur: (i) Reached, in which the new vertex reaches the sample x for a sampled input

within the error bounds to this Sampled state. (ii) Advanced, in which a new vertex x New Not equal to x is added to the RRT or (iii) TRAPPED, in which case the NEW STATE fails to produce a state that lies in the Free State Space.

- RANDOMSTATE(): Samples the a set from from the state space boundaries
- NEARESTNEIGHBOUR() :
  One of the key bottlenecks in construction of RRTs has been nearest neighbour computations. The development of efficient neighbour for high dimensional state space involves computing a nearneness coefficent based on minimum cost. Since Control Inputs are sampled only once, Cost of a state forms a additonal check over Decremental cost approach undertaken in the algorithm.
- SET(u Near): A Set data structure which stores the sampled controller values and resets after every iteration to store unique sampled controller parameters
- SAMPLECONTROL( x New, x Near): Samples the control parameters from the controller boundaries, A Clipping function that prevents that over-rides the Velocity values between [-1, 1] if the acceleration input is too large for sampling. Similarly for yaw

$$\begin{aligned} if\ v\ \epsilon[-1,1] &\Big\{ v = a * \Delta t \\ if\ v\ !\epsilon[-1,1] &\Big\{ v = max(min((1, max(-1, a * \Delta t))) \end{aligned} \tag{35}$$

$$\begin{aligned} if\theta\ \epsilon[-\frac{\pi}{3}, \frac{\pi}{3}] &\Big\{ \theta = \alpha * \Delta t \\ if\ v\ !\epsilon[-1,1] &\Big\{ \theta = max(min((\tfrac{\pi}{3}, max(-\tfrac{\pi}{3}, \alpha * \Delta t))) \end{aligned} \tag{36}$$

- CONTROLSET(x Near, u Near, Cost near): Stores the next possible set of controller inputs, and its respective state from state-space and the equivalent cost.
- MINCOST(): Extracts the (state, controller) pair from relate to the lowest cost from the iteration

3. StabilityBias(): The whole agenda of this function is optimize the weights for the sampling to minimize the cost function weights for the velocity and optimize for pose(H2) and weights corresponding to difference in dynamics(H6). This function also updates the dynamics involved by triggering Xi =1 and updating the dynamic model to the dynamics linked to dead motor.

4. Growth and Connect Samples

   Along the steps as shown in figure 1 one tree is extended, and an attempt is made to connect the nearest vertex of the other tree to the new vertex.

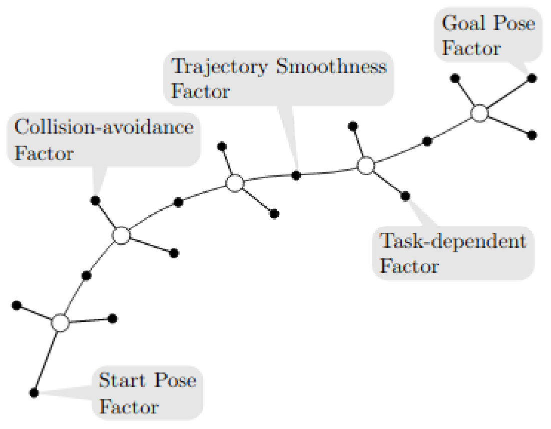
The Nearness can be measured from the cost of each node which can be calculated according to equation 18 or 32. Then, the roles are reveres ed by swapping the two trees. The algorithm as defined in Citation [1] is a minor variation of the vanilla RRT*. The algorithm attempts to grow the trees into each other half of the time.

5. Steering Method for Graphs

   Spatiotermopral Lattice Planner Steering methods can be explored here to develop Quintic Splines or Polynomial Spirals. Once these take into consideration the upgraded dynamics of the robot. A parametric curve is a curve that can be described as a set of equations with specific parameters. These parameters often denote path traversal, whether it will be through arc length or just varying from zero to one. A Polynomial Spiral can be constructed with the velocity, pitch, roll and acceleration bounds on the uniformly sampled bounds to be given to drone controller for input. Additoinally [4] suggests creating a Kinodynamic Motion Planning Factor Graph as following

[4]

Figure 7: Factor Graph Update to consider collision upgrades



**Since Kinodynamic Planners work effectively on piece-wise dynamaics of the drone and the environment, the the overview of this algorithm is pretty similar to discussed in the Section 2 / Midterm Qb, with changes in dynamics, cost function and using a Spatiotemporal Bounds to be part of factor graph to be added into nodes for Graph Search, and RRT* development instead of bi-drectional Tree growth as done earlier.**

## 8 Can you guarantee that your algorithm in part (g) will find a solution if one exists? Justify your answer.

Varational Motion Planning algorithm is subject to completeness only when the discontinuity can be recitified. Due to this discontinuity in the Dynamics. The Algorithm is only probabilisitically complete.

The modified implementation of Asymptotically-optimal meta-planning with a Randomized RRT* forms a amalgamation of Incremental Cost Optimal Planning Algorithm and RRT*. Since RRT* is a asymptotically complete planner for a feasible kinodynamic planning problem., then Bounded-Suboptimal Plan within error bounds terminates in finite time but does not guarantee optimality due to discontinuities in dynamics.

## References

[1] LaValle S.M., Kuffner J., Randomized Kinodynamic Planning, IJRR 2001.

[2] K. Hauser and Y. Zhou, ”Asymptotically Optimal Planning by Feasible Kinodynamic Planning in a State–Cost Space,” in IEEE Transactions on Robotics, vol. 32, no. 6, pp. 1431-1443, Dec. 2016, doi: 10.1109/TRO.2016.2602363.

[3] Zucker M, Ratliff N, Dragan AD, et al. CHOMP: Covariant Hamiltonian optimization for motion planning. The International Journal of Robotics Research. 2013;32(9-10):1164-1193. doi:10.1177/0278364913488805

[4] Xie, Mandy, and Frank Dellaert. ”Batch and Incremental Kinodynamic Motion Planning using Dynamic Factor Graphs.” arXiv preprint arXiv:2005.12514 (2020).

[5] Perez, A. et al. “LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics.” 2012 IEEE International Conference on Robotics and Automation (2012): 2537-2542.

[6] M. W. Mueller and R. D’Andrea, ”Stability and control of a quadrocopter despite the complete loss of one, two, or three propellers,” 2014 IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, 2014, pp. 45-52, doi: 10.1109/ICRA.2014.6906588.