Taking the ideas from [Hsu02], I implemented the algorithm in a similar way. Let's start with describing the state space first.
The state space, S, of our autonomous car C, composes of (x, v, t) where x is the position of C in its lane, v is the velocity of C at position x and t is the timestep at which it reached (x, v). At any timestep t, we choose a value for acceleration, a, and integrate its value with given Δt (which is inverse of frequency f in our case) to get velocity change for timestep Δt by equation:

$$v_{t'} = v_t + a\Delta t$$

We further integrate its value to get value of Δx which is given by (assuming that $v_{t'}$ is reached instantaneously as suggested):

$$x_{t'} = x_t + v_t\Delta t + \frac{a\Delta t^2}{2}$$

Let us look at the algorithm now then I will explain each design choice.

input: $\left(initial_{pos}, final_{pos}, N\right)$ where $initial_{pos}$ is initial position, $final_{pos}$ is target position and N is the number of samples
output: $\begin{cases} (G, & controls) \\ & None \end{cases}$,
where graph, G, contains multiple paths from initial to final position
$control$ is the corresponding controls command for each transition from state s *to* s'.

$create\_graph\left(initial_{pos}, final_{pos}, N\right)$:
$\Rightarrow$ $G \leftarrow (initial_{pos}, 0,0)$
$\Rightarrow$ $T \leftarrow \{\}$
$\Rightarrow$ $repeat\ for\ N\ steps$:
$\Rightarrow$ $\quad m_g \leftarrow select\_node(G)$ # select an existing node from the graph
$\Rightarrow$ $\quad m_{new}, u \leftarrow sample\_new\_milestone\left(m_g\right)$ # sample new milestone using m_g
$\Rightarrow$ $\quad satisfy\_constraints(m_{new})$ # check if m_{new} satisfy dynamic constraints
$\Rightarrow$ $\quad check\_collision(m_g, m_{new})$ # check if m_g, m_{new} is collision free
$\Rightarrow$ $\quad if\ check\_collision\ is\ successful$:
$\Rightarrow$ $\quad\quad add\ m_{new}\ to\ G\ with\ weight\ 1$
$\Rightarrow$ $\quad\quad if\ m_{new}\ lies\ in\ final_{pos}\ region$:
$\Rightarrow$ $\quad\quad\quad add\ m_{new}\ to\ T$
$\Rightarrow$ $\quad\quad\quad T_{max} \leftarrow m_{new}[t]$ # reduce the scope of expansion
$\Rightarrow$ $return\ G$

$select\_node(G)$:
$\Rightarrow$ $s \leftarrow random\ value\ between\ [0, final_{pos})$ # get a random position s
$\Rightarrow$ $t \leftarrow random\ value\ between\ [0, T_{max})$ # get a random timestep in the given scope
$\Rightarrow$ $node \leftarrow (s', v', t')from\ G\ such\ that\ distance\left((s,t),(s',t')\right)is\ minimum$
$\Rightarrow$ $return\ node$

$sample\_new\_milestone\left(m_g\right)$:
$\Rightarrow$ $\Delta t \leftarrow 0.1$ # $as\ f\ is\ given\ 10, so\ \frac{1}{f} = 0.1$
$\Rightarrow$ $(s_t, v_t, t) \leftarrow m_g$
$\Rightarrow$ $u \leftarrow random\ value\ of\ control\ (acceleration)$ # random acceleration with bias
$\Rightarrow$ $v_{t'} \leftarrow v_t + a\Delta t$

$\Rightarrow s_{t'} \leftarrow s_t + v_t \Delta t + \frac{a\Delta t^2}{2}$
$\Rightarrow t' \leftarrow t + \Delta t$
$\Rightarrow return\ (s_{t'}, v_{t'}, t')$

$check\_collision(m_g, m_{new})$:
$\Rightarrow pos_t \leftarrow get\ robots' position\ at\ timestep\ t$
$\Rightarrow pos_{t'} \leftarrow get\ robots' position\ at\ timestep\ t'$
$\Rightarrow for\ each\ robot$:
$\Rightarrow$ $check\ if\ m_{new}\ is\ free\ horizontally$ (*by expanding size of robots and including their velocity*)
$\Rightarrow$ $check\ if\ m_{new}\ is\ free\ vertically$ (*by expanding size of robots and including our velocity(safe distance)*
$\Rightarrow$ $if\ m_{new} is\ not\ free\ horizontally\ and\ vertically$:
$\Rightarrow$ $return\ False$
$\Rightarrow return\ True$

The algorithm first selects a node, m_g, to expand via random sampling from given range (*note: range changes as we reach a goal path to further concentrate on finding a path with shorter total time*) and exploring in the free space (like we do in RRT). After m_g has been selected, it selects a random value of acceleration to apply on the state of m_g with some bias to direct the graph more towards final position. As Δt is given to be 0.1 second, so, we calculate the new state, m_{new}, by integrating value of acceleration and Δt. The trajectory from m_g to m_{new}, is then checked for collisions. If trajectory is collision free, m_{new} is added to the graph and corresponding control is stored with edge (m_g, m_{new}). After the limit on number of samples is reached, we get a graph with multiple trajectories from initial position to goal position. We then select the trajectory which took the shortest time and execute it.

Design choices:

⇒ **Selection of node to expand**: One of the key element was to design *select_node*. Uniform selection of nodes in the graph would have resulted in overloading of an area with many nodes and not much free area would have been explored. So, some kind of bias was needed to direct the exploration away from the current density. We could use EST like weighted sampling by assigning each node with a weight inversely proportional to its density as done in [Hsu02]. But I found that calculating spatial density for each node was computational expensive. Alternatively, we could have divided nodes into bins and then choose a bin uniformly and then pick a node from selected bin. It would have also resulted in better exploration than standard uniform selection of nodes. Another way was to give weight inversely proportional to degree of the node, but this resulted in requiring exponential number of nodes to reach target position (very bad). *I found implementing RRT style selection of a graph node easy and not much computationally expensive.* I controlled the range of sampling of random point (by controlling the value of T_{max}) to direct the graph to a particular region of interest. Particularly, once we find a path from initial to final position, we want to focus the scope of expansion to nodes which occurred prior to time taken by final position to get a better path taking lesser time.

⇒ **Checking collision:** *check_collision* was also important piece of the algorithm. To check for collision, I used the approach taught in Lecture 1, in which we reduce our robot to point object and increase the size of obstacles. As this is a car and we care for it, I further added some padding to this increased obstacle size to prevent cases of near miss and have some buffer space. But my work did not end here as there was one more problem to be solved. We can reach a position *s* at time *t* with multiple possible velocity. Some of them might be so high that even if we apply maximum deceleration, we could not prevent it from crashing in (call this area as area of imminent collision). So, every obstacle vehicle was given with extra padding, given by $\left(x = \frac{-v_{t'}^2}{2 \times a_{min}}\right)$ (e.g. we will need a

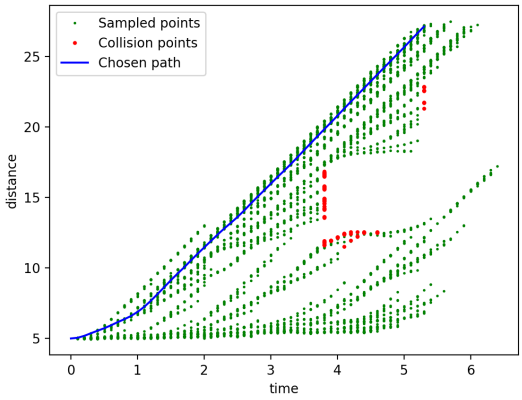
padding of $\left(\frac{25}{20}\right)$ if $v = 5\ and\ a_{min} = -10$). In this way, we are better able to sample those points which are feasible. Further, I had to control the size of padding, because as I increased the padding, completeness of my algorithm decreased.

⇒ **Number of Samples:** Running time of my algorithm was directly impacted by number of samples drawn. As number of samples increased, completeness and optimality of the algorithm increased. Due to much more samples available, algorithm could find a path if it exists with more probability of success or could focus more on finding a better path than current best. I used around 5,000 samples per data and it returned the shortest possible path almost every time.

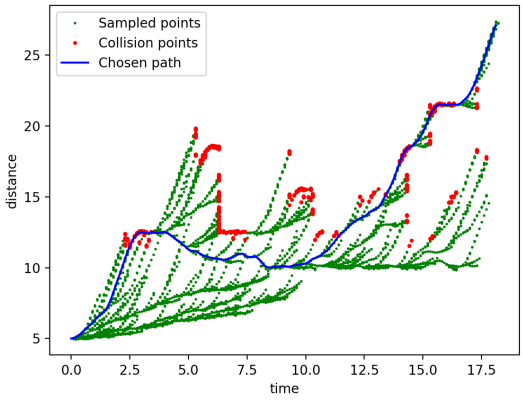
We will see from the graph that we received multiple paths to the final position but we selected the path which took minimum time. Also, we can see that due to forcing the sampling towards finding path with shorter time, area with lower ***t*** is densely sampled.

Following are some sample trees formed by the algorithm (it doesn't show the edge, only sampled points in $space \times time$):

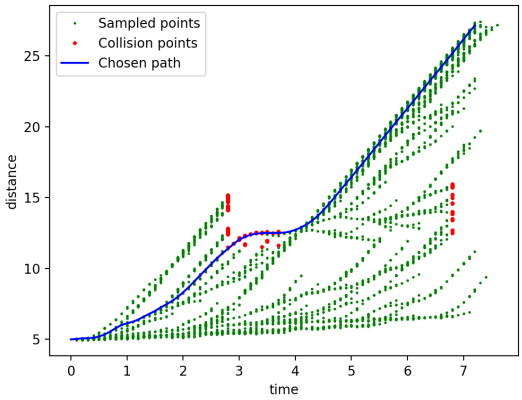
Data 1



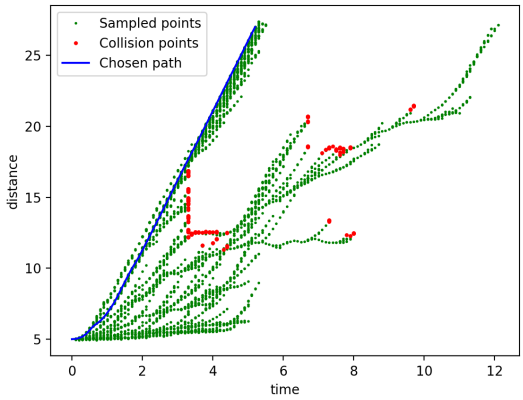
Data 2



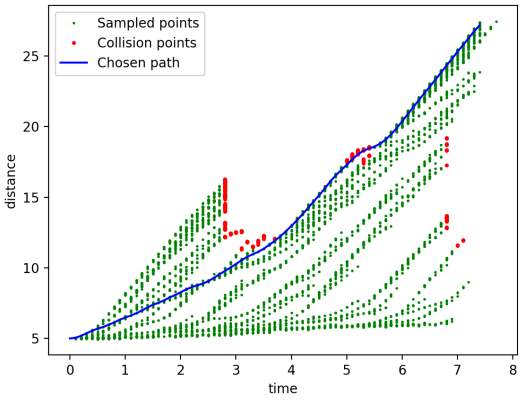
Data 3



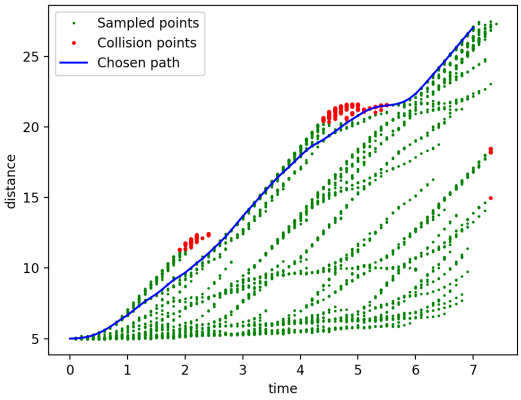
Data 4



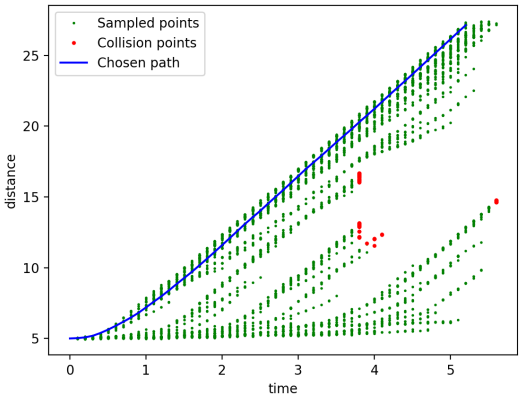
Test 1



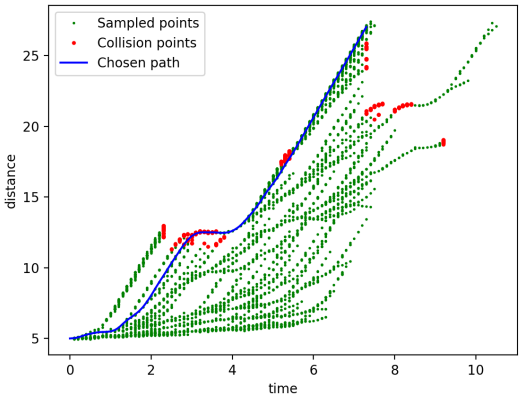
Test 2



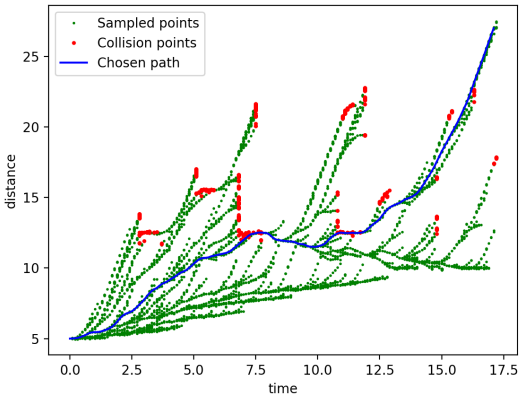
Test 3



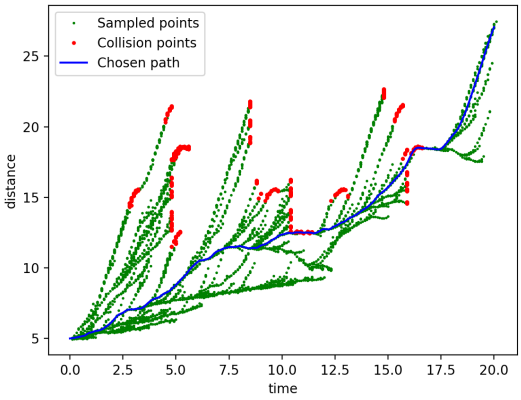
Test 4



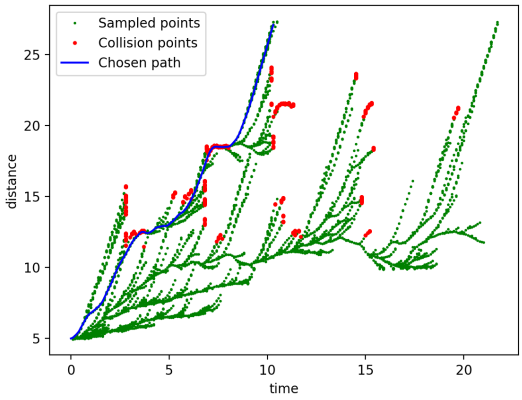
Test 5



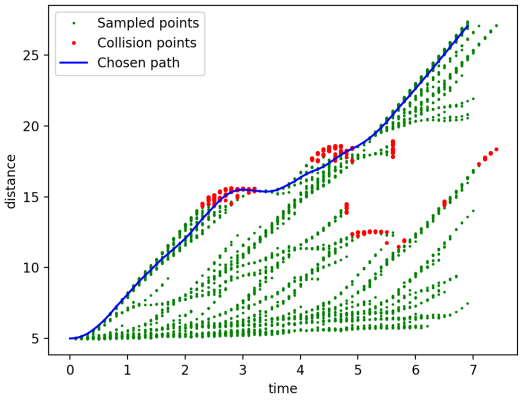
Test 6



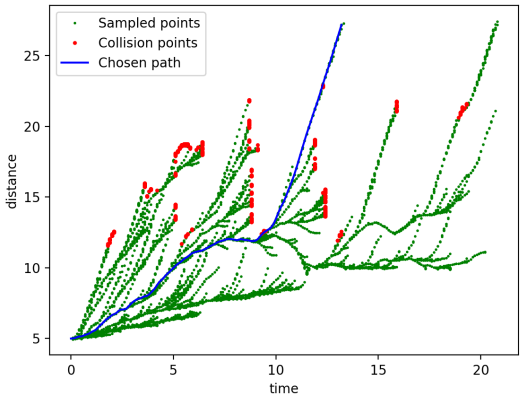
Test 7



Test 8



Test 9



Test 10