California Institute of Technology

Pasadena, California

ME/CS/EE 133B : Robotics

Fall 2022

Final Project Report

for

RRT Based Motion Planner for Non-Holonomic Mobile Robots

Submitted by

Team Name : CaRRT Cruisers

Team Members : Sri Aditya Deevi & Jeff Chen

17 March, 2023

I Problem Description

1 Problem Statement

We considering the problem of motion planning for non-holonomic mobile robots (wheeled systems) such as cars using RRT-based (**R**apidly-exploring **R**andom **T**rees) algorithms. The planner has to be able to efficiently and effectively plan through a map avoiding obstacles in various situations such as:

- Narrow Parking Garages
- Parallel Parking
- Narrow Streets

2 Problem Formulation

2.1 Wheeled System-1: Car

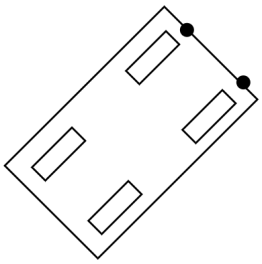


Figure 1: Schematic of a simple Car

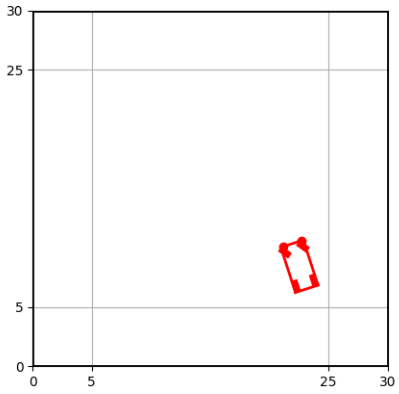
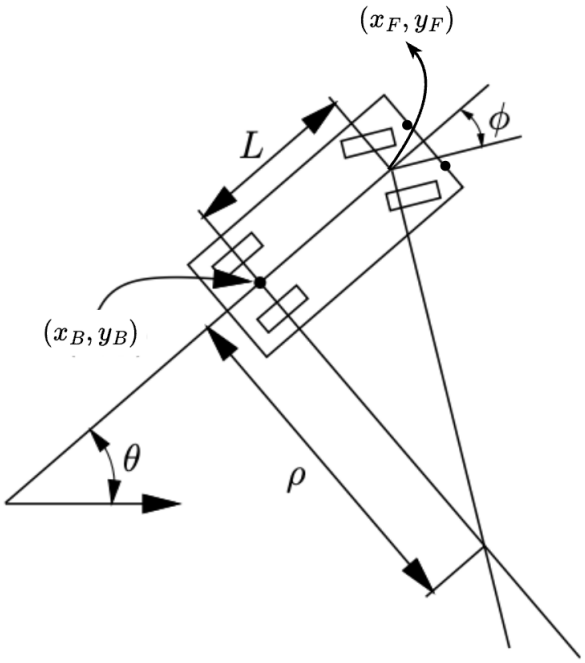


Figure 2: Visual of a simple car with turning front wheels in a Blank Map

Kinematic Equations:

Consider the following :

$$\frac{\Delta y}{\Delta x} = \tan\theta$$
$$\frac{\left(\frac{\Delta y}{\Delta t}\right)}{\left(\frac{\Delta x}{\Delta t}\right)} = \tan\theta$$
$$\frac{\dot{y}_B}{\dot{x}_B} = \tan\theta$$
$$\implies \dot{y}_B\cos\theta - \dot{x}_B\sin\theta = 0$$



To satisfy this constraint we can have (s is the signed speed of the car):

$$\dot{x}_B = s\cos\theta$$
$$\dot{y}_B = s\sin\theta$$

We can write $\rho = \frac{L}{tan(\phi)}$ and noting that $dw = \rho d\theta_B$ we get:

$$d\theta_B = \frac{\tan\phi}{L} dw$$

Dividing both sides by dt and using the fact that $\dot{w} = s$ (signed speed of the car) yields:

$$\dot{\theta}_B = \frac{s}{L}\tan\phi$$

So by considering the center of the car to be at the center of the rear axle we have the following kinematic equations describing the motion of the car*:

$$\dot{x}_B = s\cos\theta \tag{1}$$
$$\dot{y}_B = s\sin\theta \tag{2}$$
$$\dot{\theta}_B = \frac{s}{L}\tan\phi \tag{3}$$

Now we can also consider the center of the car to be at the center of the front axle, we get the following modified constraint:

$$-\dot{x}_F \sin\theta + \dot{y}_F \cos\theta - L\dot{\theta}_F = 0$$

The modified set of kinematic equations describing the motion of the car are:

$$\dot{x}_F = s\cos(\theta_F + \phi)$$
$$\dot{y}_F = s\sin(\theta_F + \phi)$$
$$\dot{\theta}_F = \frac{s}{L}\tan(\theta + \phi)$$

Both of these are equivalent representations of the wheeled system. In this work we will be primarily considering the center of the car to be at the center of the rear axle.

We should note that ($\phi \in [-\pi, \pi]$):

- If $s > 0$: FORWARD
- If $s < 0$: REVERSE
- If $\phi > 0$: LEFT
- If $\phi < 0$: RIGHT

Additional Considerations:

Each state of a car has three variable (x, y, θ), describing the position of the rear axle. For an input (s, ϕ), the next state of the car is:

$$x_{next} = x + \dot{x}\Delta t \tag{4}$$
$$y_{next} = y + \dot{y}\Delta t \tag{5}$$
$$\theta_{next} = \theta + \dot{\theta}\Delta t \tag{6}$$

where $\dot{x}, \dot{y}, \dot{\theta}$ can be obtained from equations (1), (2) and (3) respectively.

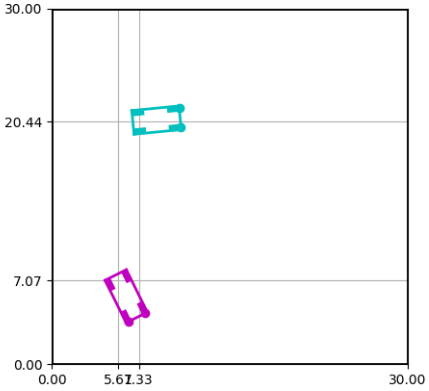
Also, the range of ϕ is limited to $[-\phi_{max}, \phi_{max}]$ and also $s \in \{-1, 1\}$.

Maps:

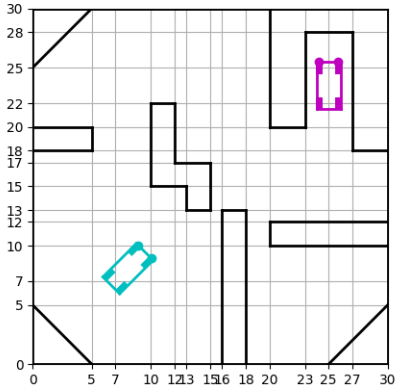
We have considered several maps for different tasks we want the car to accomplish. Note that the car shown in cyan color denotes the START position and the purple colored car denotes the GOAL position.

*Note that, here $\theta = \theta_F = \theta_B$

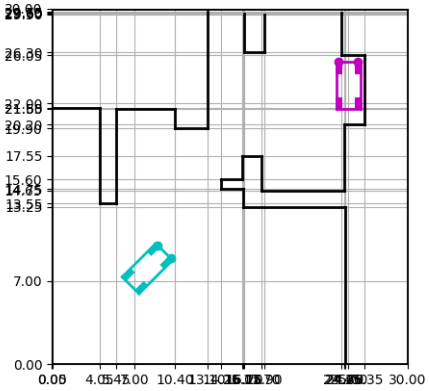
- Blank Map[†]



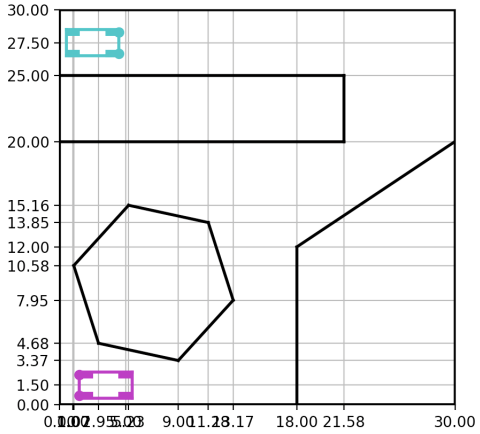
- Garage



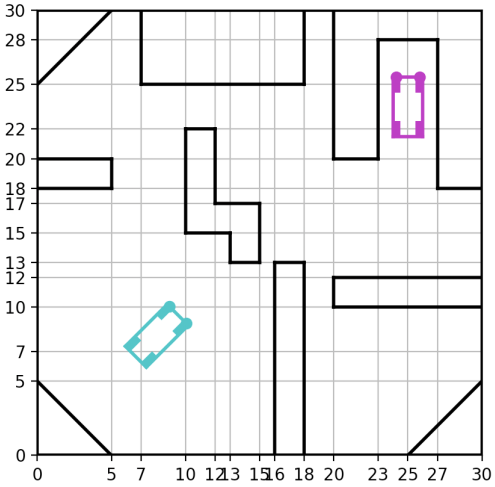
- Parallel Parking



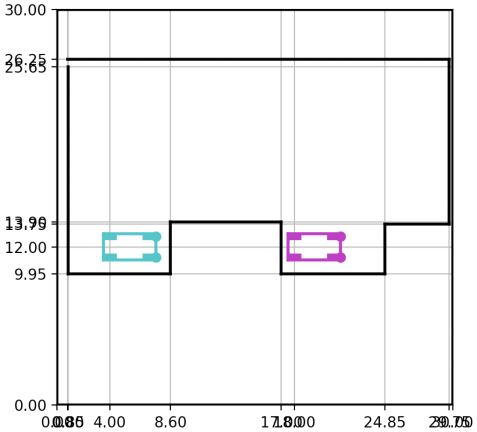
- Narrow Passage



- Narrow Garage



- Dual Parallel Parking



Note that, we also considered some other maps for Move Over and U-turn too but we are including the most interesting (and difficult!) ones in the report.

[†]There's no obstacles in the map, and the start and goal nodes are randomly generated.

2.2 Wheeled System-2: Car+Trailer

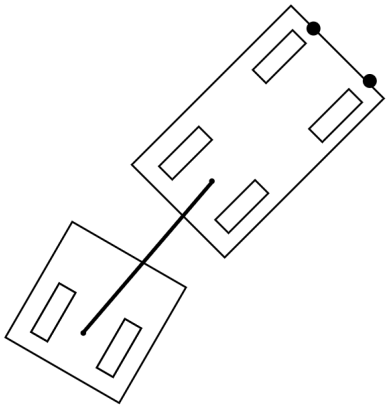


Figure 3: Schematic of a Car with a Trailer

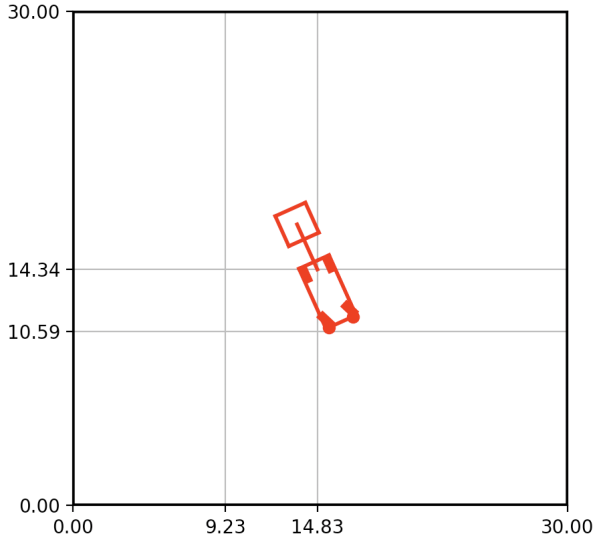


Figure 4: Visual of Car+Trailer with turning front wheels in a Blank Map

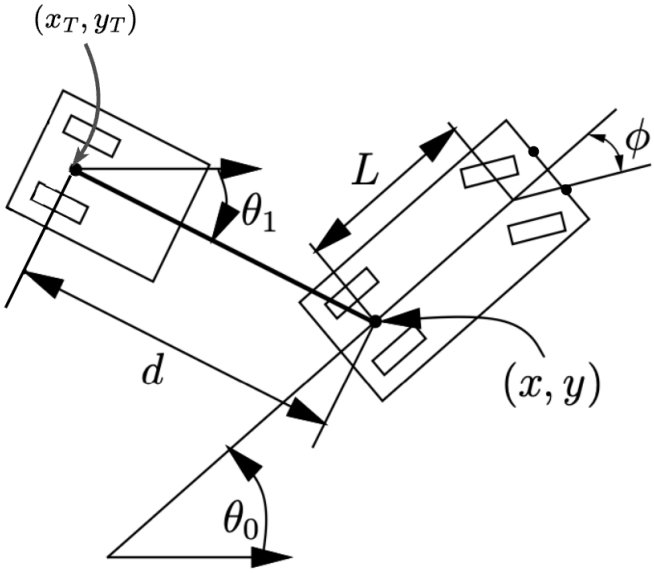
Kinematic Equations:

We can write the following:

$$y_T = y - d\sin\theta_1$$
$$x_T = x - d\cos\theta_1$$

Then:

$$\dot{y}_T = \dot{y} - d\cos\theta_1\dot{\theta}_1 \quad (7)$$
$$\dot{x}_T = \dot{x} + d\sin\theta_1\dot{\theta}_1 \quad (8)$$



Consider the following :

$$\frac{\Delta y_T}{\Delta x_T} = \tan\theta_1$$
$$\frac{\left(\frac{\Delta y_T}{\Delta t}\right)}{\left(\frac{\Delta x_T}{\Delta t}\right)} = \tan\theta_1$$
$$\frac{\dot{y}_T}{\dot{x}_T} = \tan\theta_1$$

Then substituting (7) and (8):

$$\frac{\dot{y} - d\cos\theta_1\dot{\theta}_1}{\dot{x} + d\sin\theta_1\dot{\theta}_1} = \tan\theta_1$$

As we know from the previous section that $\dot{y} = s\sin\theta$ and $\dot{x} = s\cos\theta$ (where s is the signed speed of the car) :

$$\frac{\dot{y} - d\cos\theta_1\dot{\theta}_1}{\dot{x} + d\sin\theta_1\dot{\theta}_1} = \tan\theta_1$$

$$\implies \frac{s\sin\theta_0 - d\cos\theta_1\dot{\theta}_1}{s\cos\theta_0 + d\sin\theta_1\dot{\theta}_1} = \frac{\sin\theta_1}{\cos\theta_1}$$

$$s\cos\theta_0\sin\theta_1 + d\sin^2\theta_1\dot{\theta}_1 = s\sin\theta_0\cos\theta_1 - d\cos^2\theta_1\dot{\theta}_1$$

$$d\dot{\theta}_1 = s\sin(\theta_0 - \theta_1)$$

$$\implies \dot{\theta}_1 = \frac{s}{d}\sin(\theta_0 - \theta_1)$$

The final set of kinematic equations for car with a trailer system is :

$$\dot{x} = s\cos\theta_0 \tag{9}$$
$$\dot{y} = s\sin\theta_0 \tag{10}$$
$$\dot{\theta}_0 = \frac{s}{L}\tan\phi \tag{11}$$
$$\dot{\theta}_1 = \frac{s}{d}\sin(\theta_0 - \theta_1) \tag{12}$$

Again we should note that :

- If $s > 0$: FORWARD
- If $s < 0$: REVERSE
- If $\phi > 0$: LEFT
- If $\phi < 0$: RIGHT

Additional Considerations:

Each state of the wheeled system has four variable $(x, y, \theta_0, \theta_1)$, describing the position of the rear axle.

For an input (s, ϕ), the next state of the car is:

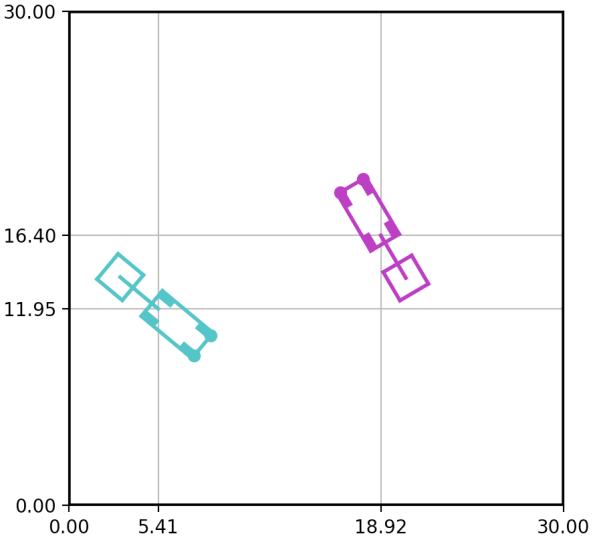
$$x^{next} = x + \dot{x}\Delta t \tag{13}$$
$$y^{next} = y + \dot{y}\Delta t \tag{14}$$
$$\theta_0^{next} = \theta_0 + \dot{\theta}_0\Delta t \tag{15}$$
$$\theta_1^{next} = \theta_1 + \dot{\theta}_1\Delta t \tag{16}$$

where $\dot{x}, \dot{y}, \dot{\theta}$, $\dot{\theta}_1$ can be obtained from equations (9), (10), (11) and (12) respectively. The range of ϕ is limited to $[-\phi_{max}, \phi_{max}]$ and also $s \in \{-1, 1\}$.

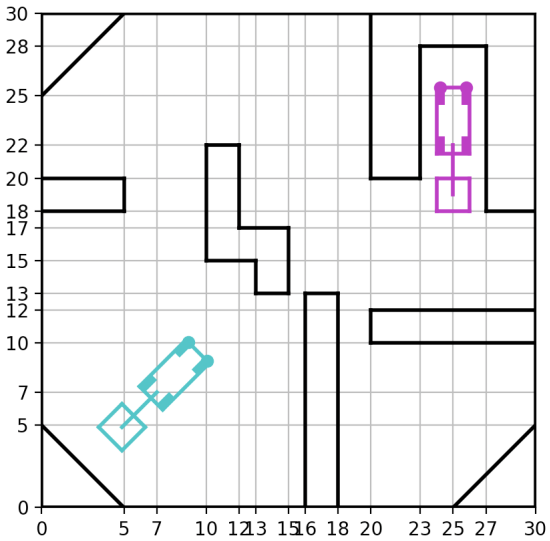
Maps:

We have considered several maps for different tasks we want the wheeled system to accomplish, similar to the car case. Note that the car-trailer shown in cyan color denotes the START position and the purple colored car-trailer denotes the GOAL position.

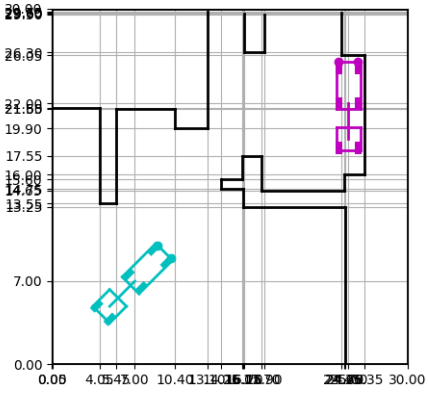
- Blank Map[‡]



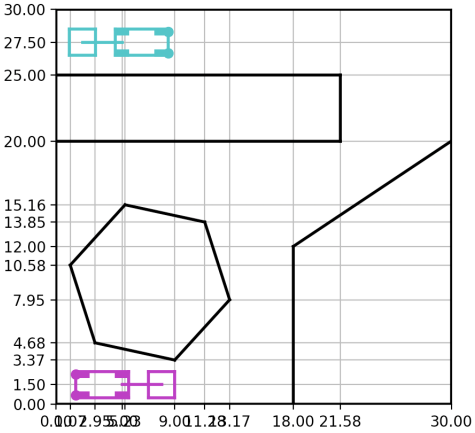
- Getting into Garage



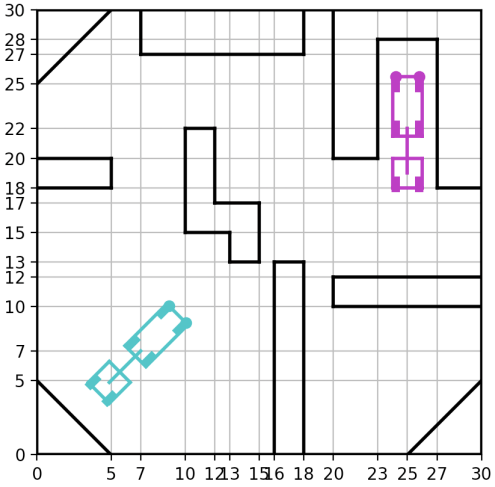
- Parallel Parking



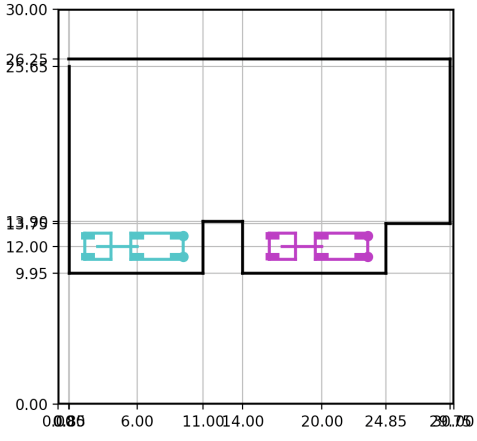
- Narrow Passage



- Narrow Garage



- Dual Parallel Parking



3 Applications

We developed and implemented the RRT algorithm for both car and car with trailer scenarios and tested them on different maps. Our results show that the algorithm can efficiently generate optimal paths while avoiding obstacles, making it a

[‡]There's no obstacles in the map, and the start and goal nodes are randomly generated.

suitable solution for various applications.

3.1 Automotive Vehicles

With the increasing demand for autonomous cars, the RRT algorithm can be integrated into autonomous vehicles to enhance their navigation capabilities, allowing them to plan efficient and safe routes in real-time while avoiding obstacles. The RRT algorithm can take into account the vehicle's kinematic constraints, such as the turning radius and maximum steering angle, and optimize the trajectory accordingly. Furthermore, the algorithm can be used to plan maneuvers such as parking and reversing, making it useful for different driving scenarios.

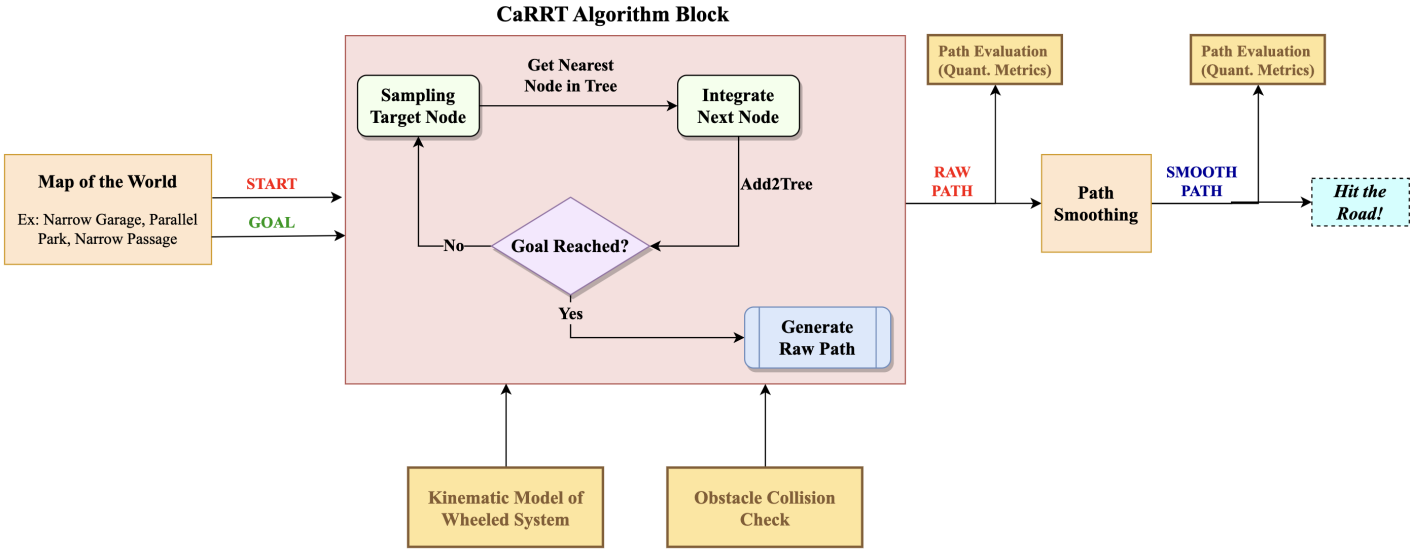
3.2 Mars Rover

Mars Rover is a robotic vehicle used for exploration on the surface of Mars. The RRT algorithm can be utilized to generate optimal paths for the Mars Rover while avoiding obstacles on the Martian surface. With the help of RRT, the Mars Rover can efficiently navigate through complex terrain, providing more information and data to researchers.

II Methodology

1 High Level Block Diagram

The following block diagram[§] illustrates the different phases of a typical experiment:



Now we will focus and zoom into different aspects of the methodology.

2 Notion of Distance

We define two kinds of distance for different purpose :

2.1 Euclidean Distance

The Euclidean distance is a measure used to determine the proximity of two nodes. When the Euclidean distance between two nodes is smaller than a predefined tolerance value, usually set to 1, we consider them to be identical. To Account for the 360° wrapping we use $cos(\theta)$ and $sin(\theta)$ for calculating Euclidean Distance.

In the case of a car without trailer, the Euclidean Distance is

$$d(q^a, q^b) = \sqrt{(x^a - x^b)^2 + (y^a - y^b)^2 + L^2(sin(\theta^a) - sin(\theta^b))^2 + L^2(cos(\theta^a) - cos(\theta^b))^2}$$

[§]Note that this block diagram mainly focuses on single tree RRT algorithm, we will discuss about the modified algorithm with two trees in the subsequent sections

In the case of a car with trailer, the Euclidean Distance is

$$d\left(q^a,q^b\right)=\sqrt{\begin{array}{c}\left(x^a-x^b\right)^2+\left(y^a-y^b\right)^2+L^2\left(\sin\left(\theta_0^a\right)-\sin\left(\theta_0^b\right)\right)^2+L^2\left(\cos\left(\theta_0^b\right)-\cos\left(\theta_0^a\right)\right)^2\\ +d^2\left(\sin\left(\theta_1^a\right)-\sin\left(\theta_1^b\right)\right)^2+d^2\left(\cos\left(\theta_1^b\right)-\cos\left(\theta_1^a\right)\right)^2\end{array}}$$

2.2 CSC Distance

The CSC distance is a measure used to determine the path length between two nodes. In the RRT algorithm, the near_node is the node that having smallest CSC distance towards the target_node.

For calculating this distance, we consider the fact that we move from one state to another using a path that consists of a **C**urve+**S**traight Line+**C**urve as shown in the figure below. For any two configurations of the car, we have to consider the counter-clockwise/clockwise turning circle (based on sign of ϕ) for both configurations. Let the center of the circle be $P_{+/-}$, we will discuss the distance of straight line and curves separately.

- Straight line distance –

  For $P_{1,\pm}$ to $P_{2,\pm}$, the distance is same as the distance between two center of circles.

  $$S=\overline{P_{1,\pm}P_{2,\pm}}$$

  For $P_{1,\pm}$ to $P_{2,\pm}$, the centers of two circles and the tangent point form a right triangle. We can calculate the distance by Pythagorean theorem.

  $$S=\sqrt{\overline{P_{1,\pm}P_{2,\pm}}^2-(2R_{min})^2}$$

- Curves –

  The arc length of the curve determined by the angle difference between the car and the tangent line.
  For $P_{1,pm}$ to $P_{2,pm}$, the tangent line is parallel to $\overline{P_1P_2}$. Thus the tangent line is

  $$\tau=atan2(y_2-y_1,x_2-x_1)$$

  For $P_{1,\pm}$ to $P_{2,\pm}$, the tangent line is tilted from $\overline{P_1P_2}$ based on the angle of the right triangle. Thus the tangent line is

  $$\tau=atan2(y_2-y_1,x_2-x_1)\pm asin(\frac{2R_{min}}{\overline{P_1P_2}})$$

  After knowing the angle of the tangent line, we can calculate the arc length

  For $P_{1,pm}$,

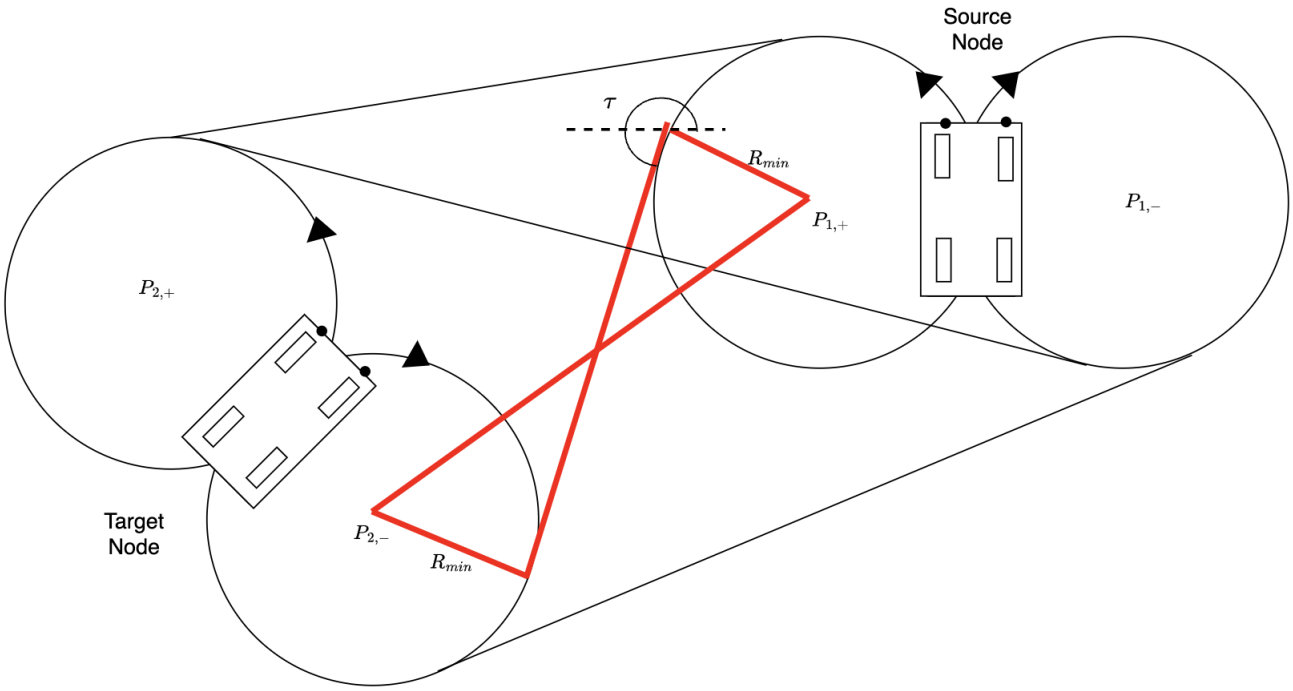
  $$C_1=R_{min}*wrap360(\pm(\tau-\theta_1))$$

  For $P_{2,pm}$,

  $$C_2=R_{min}*wrap360(\pm(\theta_2-\tau))$$

The total distance of the CSC curve would be:

$$CSC=C_1+S+C_2$$



The above discussion only applies to the forward case. For the backwards case, simply switch the source and goal node and then apply the calculation again.

3 Sampling Strategy

Uniformly random sampling of nodes will cause the tree to explore the area having no directionality towards goal whereas greedy sampling with goal node to be the target node always can cause the tree to get stuck leading to no exploration. So, we considered two different sampling strategies in this work, to address this issue.

3.1 Almost Random Sampling

We sample the target 95% pure random and 5% as the goal. This way the tree can explore the world very fast while having a tendency to approach the goal. We used this strategy in the modified CaRRT algorithm with two trees as mentioned in Section 6.

3.2 Goal Biased Random Sampling

We sample the target with 5% chance as the goal and 50% as a random node. The remaining 45% of the time we sample randomly somewhere near the goal (normally distributed with mean as goal node). Our intuition is that this can help the planner to plan faster to the goal node. We used this strategy in the CaRRT algorithm with single tree as mentioned in Section 4.

4 CaRRT Algorithm (Single Tree)

A high level RRT algorithm for one tree is presented as follows:

Algorithm 1: CaRRT Algorithm for one tree (High Level)

Data: START, GOAL, MAP
Result: PATH
Define Parameters r, p, TOL;
TREE = [START] ;
while *TRUE* **do**
  q_t = Get_Target(GOAL, r) ; // Goal Biased Random Sampling
  q_{nt} = NearestNode_TREE(q_t,TREE) ; // Uses CSC Distance
  q_{next} = NextNode_Integrate(q_{nt},q_t) ; // Uses CSC Distance and avoids collisions
  add2tree(q_{next}) ;
  d = Euclid_Distance(GOAL, q_{next}) ;
  r = $p \times d$;
  if d <*TOL* **then**
    break
  end
end
PATH = Get_Path(q_{next}, TREE) ;
Return PATH

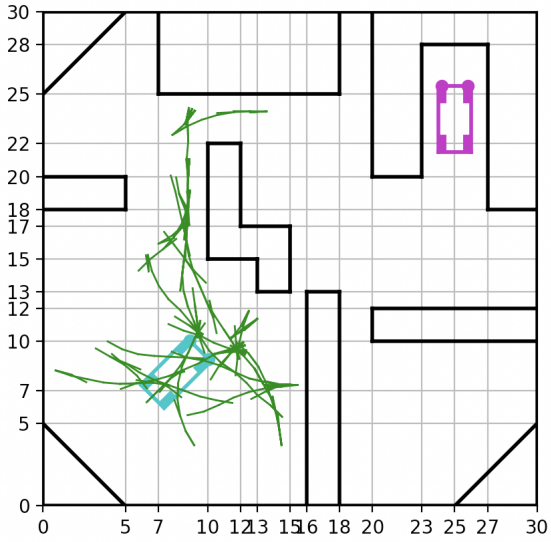


Figure 5: A Still of the tree growing.

Attempt for Optimization of ϕ

In this section, we want to describe our attempt to determine the optimal ϕ for going from the nearest node in the tree to target. We considered Euclidean distance (for simplicity) as the distance function and let the car's center be at the center of the front axle. The problem was to determine the optimal ϕ (steer angle) that would minimize the distance between the next node and the target node. In this optimization problem, we did not consider constraints (for simplicity).

Consider:

$$d(q_n,q_t)^2 = \left(x_{nt} + s\cos\left(\theta_{nt}+\phi\right)\Delta t - x_t\right)^2 + \left(y_{nt} + s\sin\left(\theta_{nt}+\phi\right)\Delta t - y_t\right)^2 +$$
$$L^2\left(\sin\left(\theta_{nt} + s\frac{\sin\phi}{L}\Delta t\right) - \sin\theta_t\right)^2 + L^2\left(\cos\left(\theta_{nt} + s\frac{\sin\phi}{L}\Delta t\right) - \cos\theta_t\right)^2$$

where

$$q_t = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} \mapsto \text{Target Node}$$

$$q_{nt} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} \mapsto \text{Nearest Node to } q_t \text{ in the tree.}$$

$$q_n = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} \mapsto \text{Potential Next Node to } q_{nt} \text{ [Using (3),(4),(5)]}$$

Then we considered $\frac{d}{d\phi}\left(d(q_n, q_t)^2\right) = 0$ and simplified but we finally got a non-linear (not very well behaved) equation at the end (Even after considering many simplifying assumptions). We tried to use a optimizer to get the roots of the equation but it sometimes led to extremely high values. We should note that even if we manage to succeed to get the optimal ϕ it may not be considered because it may be out of allowable range or there may be obstacles in the map.

Therefore, we finally decided to search for values of ϕ in the range $[-\phi_{max}, \phi_{max}]$ instead that minimized the distance and were without collision.

5 Smoothing Function

Postprocessing is commonly used in RRT based algorithms to smoothen the path as RRT incorporates randomness in its methodology. But simple postprocessing is not feasible when we are dealing with non-holonomic motion planning problems. So we have considered a smoothing algorithm that respects the kinematic constraints of the wheeled system. The following is the high level pseudo code of the algorithm :

Algorithm 2: Smoothing Function for Non-Holonomic Mobile Robots

```
Function PostProcess_smooth(PATH):
    if len(PATH) <= 2 then
        Return PATH
    end
    NEW_PATH = [PATH[0]] ;
    while len(NEW_PATH) < len(PATH) do
        q_next = NextNode_Integrate(NEW_PATH[-1], PATH[-1]) ;        // Uses CSC Distance and avoids
         collisions
        NEW_PATH.append(q_next);
        if Euclid_Distance(PATH[-1], q_next) < TOL then
            Return NEW_PATH
        end
    end
    PATH1, PATH2 = SPLIT(PATH);
    PATH1 = PostProcess_smooth(PATH1);
    PATH2 = PostProcess_smooth(PATH2);
    NEW_PATH = MERGE(PATH1,PATH2);
    Return NEW_PATH
end
```

In this algorithm, we basically reuse the $NextNode\_Integrate(.)$ function to find if there is a shorter (in terms of number of nodes) *feasible* path to the end of the path. If for the raw path, there is no such post processed path, we recursively split the raw path and repeat the smoothing algorithm for each of the sub paths. Eventually, we will get a path that is *atleast* as smooth as the raw path. But in practice, we find that this algorithm always returns a smoother path than the raw one.

6 Modified CaRRT Algorithm with Two Trees

We grow two trees T1 and T2 from start node and goal node. In each iteration, we randomly pick a target node and find the nearest node from T1, called q_{near}, then we find the next node $q_{next,1}$ from near node that has the smallest CSC distance and append it to T1. Then let $q_{next,1}$ be the target for T2, find the nearest node q_{near} and the corresponding next node $q_{next,2}$.

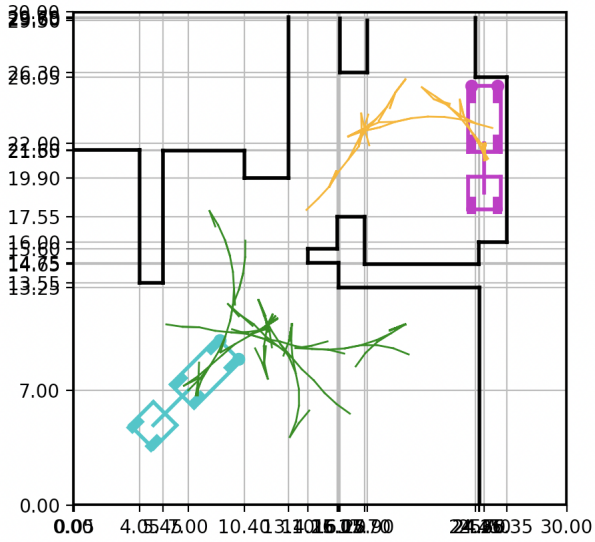


Figure 6: A Still of the Two Trees (one from START and other from GOAL) growing.

Algorithm 3: Modified CaRRT Algorithm with Two Trees (High Level)

Data: START, GOAL, MAP
Result: PATH
Define Parameters TOL;
TREE1 = [START] ;
TREE2 = [GOAL] ;
while *TRUE* **do**
  q_t = Get_Target() ; // Totally Random Sampling
  q_{nt} = NearestNode_TREE(q_t,TREE1) ; // Uses CSC Distance
  $q_{next,1}$ = NextNode_Integrate(q_{nt},q_t) ; // Uses CSC Distance and avoids collisions
  add2tree($q_{next,1}$, TREE1) ;
  q_t = $q_{next,1}$; // set next node as the target for another tree
  q_{nt} = NearestNode_TREE(q_t,TREE2) ; // Uses CSC Distance
  $q_{next,2}$ = NextNode_Integrate(q_{nt},q_t) ; // Uses CSC Distance and avoids collisions
  add2tree($q_{next,2}$, TREE2) ;
  d = Euclid_Distance($q_{next,1}$, $q_{next,2}$) ;
  if d <TOL **then**
    break
  end
  SWAP(TREE1, TREE2);
end
PATH1 = Get_Path($q_{next,1}$, TREE1) ;
PATH2 = Get_Path($q_{next,2}$, TREE2) ;
PATH = MERGE(PATH1, PATH2) ;
Return PATH

If $q_{next,1}$ and $q_{next,2}$ are close enough, we assume two tree connect so we break the while loop, otherwise we switch T1 and T2 and continue the loop. Note that we switch T1 and T2 in every iteration so both tree will try to grow towards the random space.

7 CaRRT* Algorithm

We introduced a notion of optimality into the CARRT algorithm for single tree as a experiment[¶]. We considered only the Wheeled system-1: Car, in this case. The high level pseudo code of the algorithm is as follows:

Algorithm 4: CaRRT* Algorithm for one tree (High Level)

```
Data: START, GOAL, MAP
Result: PATH
Define Parameters r,r_near, p, TOL;
TREE = [START] ;
while TRUE do
    q_t = Get_Target(GOAL, r) ;                          // Goal Biased Random Sampling
    q_nt = NearestNode_TREE(q_t,TREE) ;                  // Uses CSC Distance
    q_next = NextNode_Integrate(q_nt,q_t) ;              // Uses CSC Distance and avoids collisions
    q_near_all = Get_Near_Nodes(q_next,r_near);          // Gets Nodes in the Neighbourhood of q_next
    q_min = q_nt;
    c_min = cost(q_nt) + CSC_Distance(q_next,q_nt);
    for q_near ∈ q_near_all do ;                         // Connecting along a minimum-cost path

        if cost(q_near) + CSC_Distance(q_next,q_near) < c_min then
            q_min = q_near;
            c_min = cost(q_near) + CSC_Distance(q_next,q_near);
        end
    end
    q_nt = q_min ;
    for q_near ∈ q_near_all do ;                         // Rewiring the Tree

        if cost(q_next) + CSC_Distance(q_near,q_next) < cost(q_near) then
            Parent(q_near) = q_next;
            cost(q_near = cost(q_next) + CSC_Distance(q_near,q_next);
        end
    end
    add2tree(q_next) ;                                   // Parent is q_nt
    d = Euclid_Distance(GOAL, q_next) ;
    r = p × d;
    if d <TOL then
        break
    end
end
PATH = Get_Path(q_next, TREE) ;
Return PATH
```

Here, we store compute, store and update, when necessary, a cost (For example, path length) associated with each node. CaRRT* is basically a modified version of CaRRT algorithm, where we also consider rewiring the tree for optimality considerations. We will mention some results and inferences in subsequent sections.

[¶]This is not a main part of our project because we could not conduct extensive experiments as in the other algorithms. But some experiments and results are attached at the end.

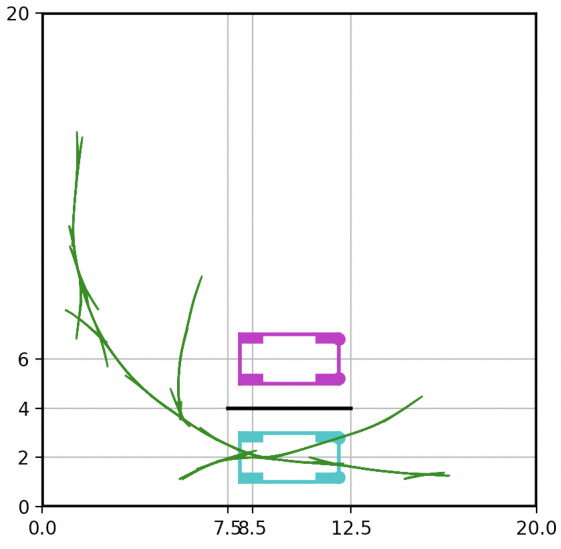


Figure 7: A Still of the tree growing. As we will give our inferences later, we observed that the tree grows slower, but branches less and the overall “behvaiour" of the growth seems to different qualitatively.

Note on Parallelization : Implementation Detail

We realized that as the number of nodes in the tree increases, computing the CSC distance sequentially (Ex: using for loops) can be highly time consuming and can slow down the algorithm. So in order to cut down the time in this stage, we parallelized the computation and used `cupy`, which utilizes the GPU (CUDA cores), instead of `numpy`.

III Results, Analysis and Inferences

1 Description of Quantitative Metrics

We realized that we need to also consider proper quantitative metrics to understand and analyze the performance of various aspects of the problem. So we define the following quantitative metrics as follows:

1. # Nodes Sampled : We record the number of nodes sampled as a part of the algorithm.
2. Tree Size : We record the number of nodes that are added to the tree, which is the size of the tree.
3. # Nodes in Path : We determine number of nodes that are a part of the path found (Raw or Smooth).
4. Path Length: We determine distance travelled by the mobile robot while traversing from the START to GOAL. This includes the both the curved segments and the straight line segments. For straight line segments the length of the segment is the 2D Euclidean distance between the (x, y) coordinates of the end nodes of the segment. For curved segments, the length of the segment is:

$$L_{curve} = \left| \frac{L}{\tan\phi} \left(\theta_0^A - \theta_0^B \right) \right|$$

   where θ_0^A and θ_0^B correspond to the orientations of end nodes of the segment.

5. Path Smoothness : The overall smoothness of the path. A score close to zero indicate smoother paths. We consider that straight line paths have a score of zero ($\phi = 0$ and $R = \frac{L}{\tan\phi} \to \infty$) and paths with larger radius of curvature ($R = \frac{L}{\tan\phi}$) to have lower score. Therefore, the final metric can be calculated as follows:

$$\text{Path Smoothness} = \sum_{segments} \left(\frac{L}{R_{segment}} \right)^2 = \sum_{segments} \tan^2 \left(\phi_{segment} \right)$$

6. Planning Time (in s) : We record the time taken to find a path frmo START to GOAL.

6. Smoothing Time (in s) : We record the time taken to find postprocess the raw path to get the smooth path using the smoothing function.

6. Outcome : If the planner cannot find a path, the outcome is FAILURE, otherwise it is SUCCESS.

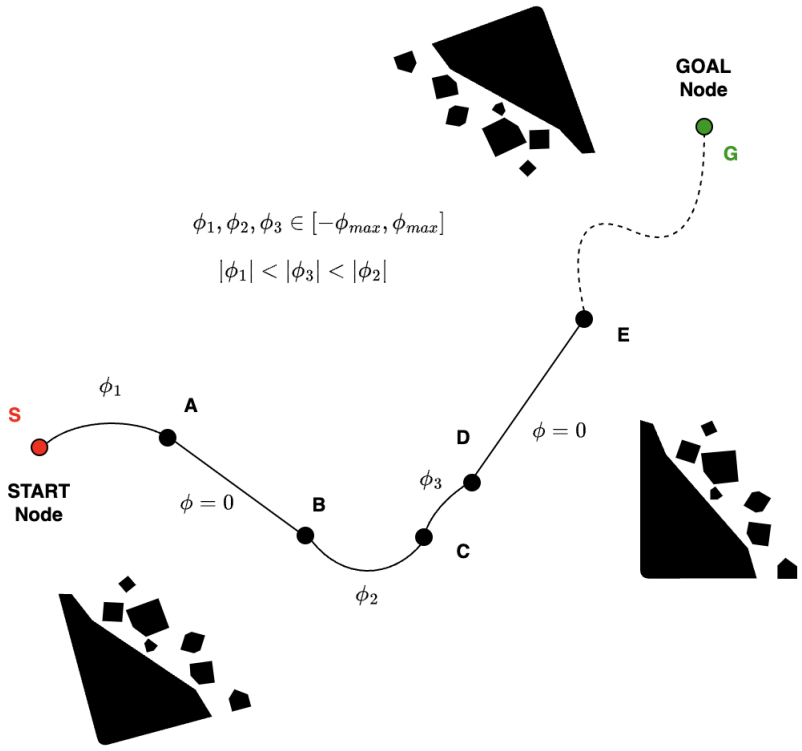


Figure 8: Defining a Typical Path. Here, AB & DE are some straight line segments and SA, BC & CD are some curved segments of the path.

2 Experimental Configuration

We perform experiments differing in the following factors:

(i) **Wheeled System Types**. Car, Car+Trailer

(ii) **Algorithm**. Single Tree, Two Tree

(iii) **Scenarios/Maps**. Garage, Narrow Garage, Blank Map, Parallel Parking, Narrow Passage, Dual Parallel Parking

For each experimental configuration, we conducted 5 experiments and recorded mean and standard deviation of the quantitative metrics mentioned in the previous section and some interesting stills of the path. The stills of the path included the traced raw and smooth paths.

3 Results of Notable Experiments

In this section, we will first present notable quantitative and qualitative results obtained from the experiments we conducted according to the experimental configuration mentioned in the previous section and then we try to make interesting inferences from the results after analysis.

3.1 Quantitative Results

Note

In all the tables shown in this subsection, for each experimental configuraion

- 1^{st} sub-row corresponds to the raw path (**Before Smoothing**) metrics.
- 2^{nd} sub-row corresponds to the smooth path (**After Smoothing**) metrics.

Also, whenever a metric is represented as $a \pm b$, a corresponds to the mean value of the metric and b corresponds to the standard deviation of the metric.

Metrics / Expt. Config	Planning Time (in s)	Smoothing Time (in s)	# Nodes Sampled	# Nodes in Path	Tree Size	Path Length	Path Smoothness	Success Rate (in %)
Car - Single Tree	436.09±235.8782	-	40821.4±13501.51	82.8±11.2321	3324.4±1087.787	81.8±11.2321	14.7666±1.888363	100
	436.09 ± 235.6237875	1.636±0.240216569	40821.4±13501.50788	69±9.859006035	3324.4±1087.787222	70.802±8.406245892	10.9418±1.864881272	100
Car - Two Trees	276.136±46.7977	-	29314.6±10313.91	69.2±2.227106	2546±895.7384	68.0608±2.648857	12.1518±1.180364	100
	276.136±146.7977422	1.232±0.2352360517	29314.6±10313.90565	60±4.604345773	2546±895.7383547	68.8978±19.8127333	10.1698±2.30889111	100
Car+Trailer - Single Tree	1228.786±2440.1042	-	78331.8±18078.82	92.4±3.006659	6105.8±1368.479	75.4±34.26135	15.8798±0.880078	100
	1228.786±440.1041783	2.176±0.5993529845	78331.8±18078.82265	72.8±2.638181192	6105.8±1368.47863	74.4332±3.942138019	10.8124±0.4809667764	100
Car+Trailer - Two Trees	4860.748±2409.682	-	155997±40644.35	114.2±15.72768	13002.4±3626.077	152.768±20.24997	19.3496±2.852108	100
	4860.748±2409.682336	3.648±0.883864243	155997±40644.34587	85.2±15.86694678	9904.5296±5929.298131	81.8884±41.23193535	12.5066±1.842791752	100

Table 1: Quantitative Results for Scenario/Map : Garage

Metric / Expt. Config	Planning Time (in s)	Smoothing Time (in s)	# Nodes Sampled	# Nodes in Path	Tree Size	Path Length	Path Smoothness	Success Rate
Car - Single Tree	436.09±235.8782	-	40821.4±13501.51	82.8±11.2321	3324.4±1087.787	81.8±11.2321	14.7666±1.888363	100
	436.09±235.6237875	1.636±0.240216569	40821.4±13501.50788	69±9.859006035	3324.4±1087.787222	70.802±8.406245892	10.9418±1.864881272	100
Car - Two Trees	276.136±146.7977	-	29314.6±10313.91	69.2±2.227106	2546±895.7384	68.0608±2.648857	12.1518±1.180364	100
	276.136±146.7977422	1.232±0.2352360517	29314.6±10313.90565	60±4.604345773	2546±895.7383547	68.8978±19.8127333	10.1698±2.30889111	100
Car+Trailer - Single Tree	1228.786±440.1042	-	78331.8±18078.82	92.4±3.006659	6105.8±1368.479	75.4±34.26135	15.8798±0.880078	100
	1228.786±440.1041783	2.176±0.5993529845	78331.8±18078.82265	72.8±2.638181192	6105.8±1368.47863	74.4332±3.942138019	10.8124±0.4809667764	100
Car+Trailer - Two Trees	4860.748±2409.682	-	155997±40644.35	114.2±15.72768	13002.4±3626.077	152.768±20.24997	19.3496±2.852108	100
	4860.748±2409.682336	3.648±0.883864243	155997±40644.34587	85.2±15.86694678	9904.5296±5929.298131	81.8884±41.23193535	12.5066±1.842791752	100

Table 2: Quantitative Results for Scenario/Map : Narrow Garage

Expt. Config \ Metric	Planning Time (in s)	Smoothing Time (in s)	# Nodes Sampled	# Nodes in Path	Tree Size	Path Length	Path Smoothness	Success Rate
Car - Single Tree	43.142±47.1955	-	8306.4±8257.103	24±7.694154	696.2±691.2457	23±7.694154	4.1188±1.199961	100
	43.142±47.1955008	0.044±0.02727636339	8306.4±8257.102993	21.2±6.177378085	696.2±691.2456582	20.0218±6.047834402	3.0012±0.5990871055	100
Car - Two Trees	1.984±0.926188	-	678±316.9031	21.6±3.611094	60.4±27.52163	36.7868±24.52982	4.8122±1.127642	100
	1.984±0.9261878859	0.07±0.03577708764	678±316.9031398	19.2±3.18747549	60.4±27.52162786	34.4524±22.88387108	4.167±1.290455888	100
Car+Trailer - Single Tree	12.78±14.44246	-	3477.4±3175.954	29.2±6.305553	291±264.5653	28.2±6.305553	6.058±0.535468	100
	12.78±14.44246378	0.086±0.07310266753	3477.4±3175.954068	25.2±5.491812087	291±264.5653038	24.6372±6.284400955	3.8182±1.056201761	100
Car+Trailer - Two Trees	11.986±4.775494	-	3368.6±1243.502	39.2±10.62826	294.8±108.3225	79.2192±38.89215	8.363±2.658769	100
	11.986±4.775494111	0.21±0.08414273587	2685.104±1816.853829	32.8±9.6	294.8±108.3224815	47.801±24.35116221	6.1628±1.882673886	100

Table 3: Quantitative Results for Scenario/Map : Blank Map

Expt. Config \ Metric	Planning Time (in s)	Smoothing Time (in s)	# Nodes Sampled	# Nodes in Path	Tree Size	Path Length	Path Smoothness	Success Rate
Car - Single Tree	355.382±205.9814	-	37559±17510.23	54.4±5.817216	3084.4±1441.605	53.4±5.817216	9.7334±1.240262	100
	355.382±205.9814099	0.502±0.114437756	37559±17510.22987	43.4±5.238320341	3084.4±1441.605161	45.7706±7.799649392	6.6416±1.998647202	100
Car - Two Trees	56.964±25.21891	-	7020.4±2170.244	57.8±7.082372	606.2±181.6154	106.9978±39.07029	10.2496±1.287854	100
	56.964±25.21891163	1.11±0.6649812027	7020.4±2170.243544	45.6±3.261901286	606.2±181.6154178	96.373±4.545409597	7.5536±0.876439068	100
Car+Trailer - Single Tree	4489.717±3139.704	-	165590.7±72540.47	81±2.160247	13543.67±5911.529	80±2.160247	11.61533±0.645674	60
	4489.716667±3139.704145	1.556666667±0.106249183	165590.6667±72540.46937	50.66666667±1.699673171	13543.66667±5911.52853	48.89866667±2.06129835	6.980666667±0.9715174843	60
Car+Trailer - Two Trees	278.9929±94.51824	-	19900±7320.629	69.85714±5.054802	1682.286±637.7159	133.3483±30.92307	12.48871±1.554309	100
	278.9928571±94.51824355	5.272857143±1.795221548	19900±7320.628935	61.57142857±6.779320296	1682.285714±637.7159498	113.603±23.92894812	9.335571429±1.735187998	100

Table 4: Quantitative Results for Scenario/Map : Parallel Parking

Expt. Config \ Metric	Planning Time (in s)	Smoothing Time (in s)	# Nodes Sampled	# Nodes in Path	Tree Size	Path Length	Path Smoothness	Success Rate
Car - Single Tree	38.242±33.7216	-	8869.6±4718.3	69.2±2.856571	620.2±372.8299	68.2±2.856571	11.3322±1.526625	100
	38.242±33.7216	0.454±0.2662780502	8869.6±4718.3	60.2±1.6	620.2±372.8299	60.0316±1.09416	7.1116±0.600939	100
Car - Two Trees	41.54±11.40847	-	8367.143±1811.35	77.28571±9.705879	643.4286±111.4628	153.0609±31.43939	13.23757±2.212673	100
	41.54±11.40847	0.5685714286±0.3132515453	8367.143±1811.35	68±7.855844	643.4286±111.4628	136.603±36.74084	9.1±1.839312	100
Car+Trailer - Single Tree	276.53±181.6557	-	47242.6±16338.34	82.6±4.673329	2755.8±1255.01	81.6±4.673329	14.6358±1.263574	100
	276.53±181.6557	0.446±0.04841487375	47242.6±16338.34	69.2±5.268776	2755.8±1255.01	66.3802±5.087768	9.6008±1.151339	100
Car+Trailer - Two Trees	696.99±77.79975	-	77632.33±18895.27	82.66667±3.299832	4774.333±1035.98	132.791±40.57216	12.42233±0.491789	60
	696.99±77.79975	0.5533333333±0.03091206165	77632.33±18895.27	66±8.041559	4774.333±1035.98	80.73±27.1415	8.421±1.696607	60

Table 5: Quantitative Results for Scenario/Map :Narrow Passage

Expt. Config \ Metric	Planning Time (in s)	Smoothing Time (in s)	# Nodes Sampled	# Nodes in Path	Tree Size	Path Length	Path Smoothness	Success Rate
Car - Single Tree	95.534±74.85489	-	15305.4±9385.231	41.6±7.605261	1252±759.1147	36.6±9.308061	7.8642±1.170802	100
	95.494±74.88808	0.272±0.05706137047	15305.4±9385.231	32.4±5.678028	1252±759.1147	33.4792±7.287483	5.5618±1.10188	100
Car - Two Trees	13.088±3.099577	-	3693.6±844.6399	44.6±6.343501	279.6±57.45468	44.2666±6.327144	8.1596±1.612154	100
	13.088±3.099576745	0.284±0.02653299832	3693.6±844.639947	36.2±4.166533331	279.6±57.45467779	36.56±7.453337829	5.8074±1.432936928	100
Car+Trailer - Single Tree	284.7775±240.1502	-	37434.25±31469.53	54.75±21.33512	2638±1917.038	53.75±21.33512	9.42375±3.079055	80
	284.7775±240.1502	0.53±0.341101158	37434.25±31469.53	39.25±13.49768	2638±1917.038	37.818±12.55129	7.00575±1.604448	80
Car+Trailer - Two Trees	43.95±10.13641	-	9386.4±1963.358	54.4±15.05457	742.4±132.0342	53.0082±14.11212	10.718±3.410023	100
	43.95±10.13641	0.784±0.3903639328	9386.4±1963.358	45.2±15.71496	742.4±132.0342	45.7014±15.44001	7.8064±3.621778	100

Table 6: Quantitative Results for Scenario/Map : Dual Parallel Parking

3.2 Interesting Inferences and Graphical Results

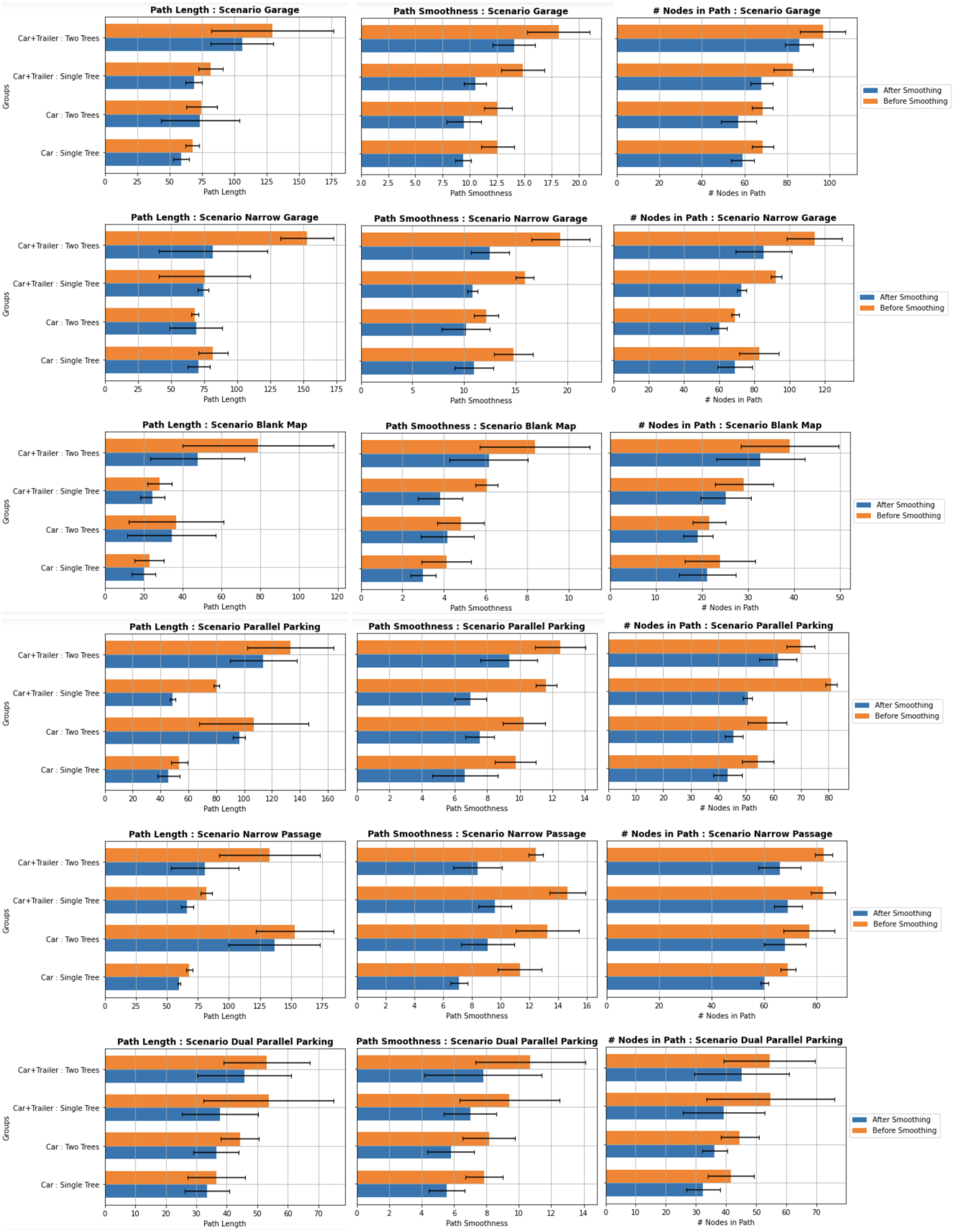


Figure 9: Graphical Representation of Quantitative Metrics across various different experiment dimensions. The orange bars indicate metrics *Before Smoothing* whereas the the blue bars indicate metrics *After Smoothing*.

■ **Effect of Smoothing Function :**

Looking at both the bars in each plot of Figure. 9, we can clearly see that for all the metrics such as Path Length, Path Smoothness[‖] and # Nodes in Path, the after smoothing values are almost always less (and in most cases very less) across various scenarios. This shows the efficacy of the developed smoothing function. Note that, this post-processing is handled by respected the kinematic non-holonomic constraints of the wheeled system. This makes the final path better and feasible for the system to accomplish.

■ **Comparison of Different Algorithms across Scenarios :**

In this part, we analyse the effectiveness of Single Tree CaRRT and Two tree CaRRT algorithms in different maps for both wheeled systems.

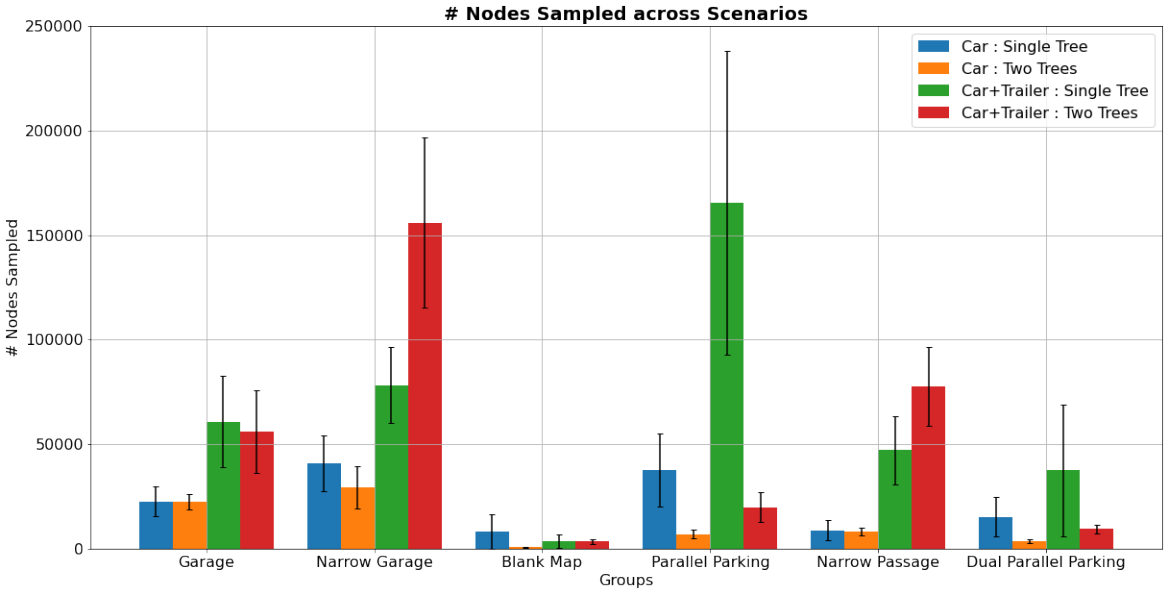


Figure 10: Comparison of # Nodes sampled across Scenarios

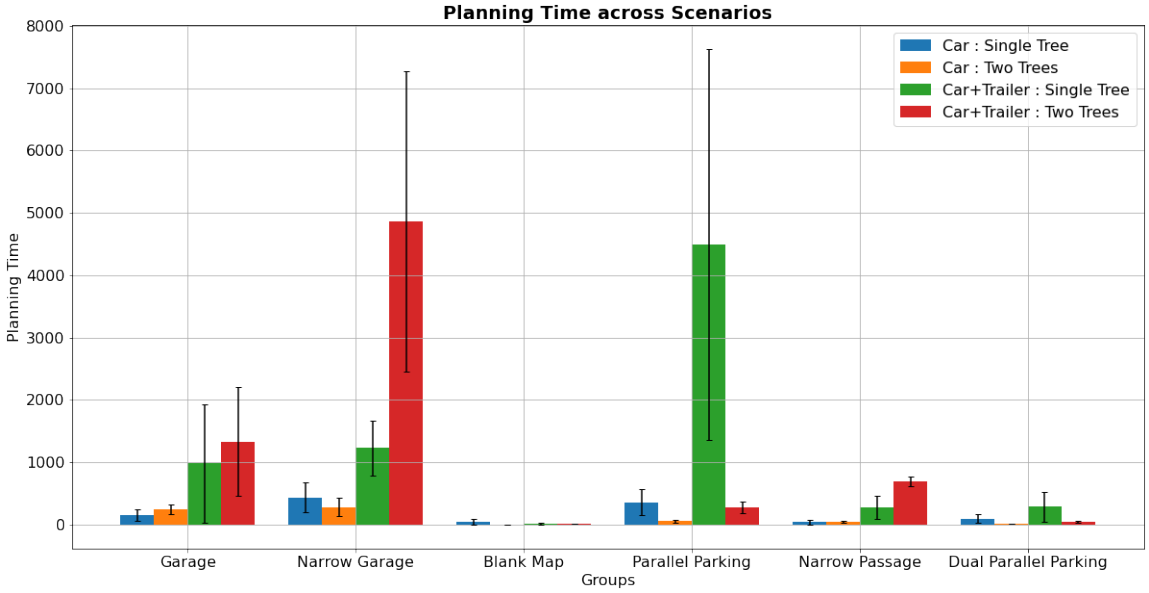


Figure 11: Comparison of Planning Time (in s) across Scenarios

[‖] Here, we should note that a lower value of smoothness indicates a smoother path

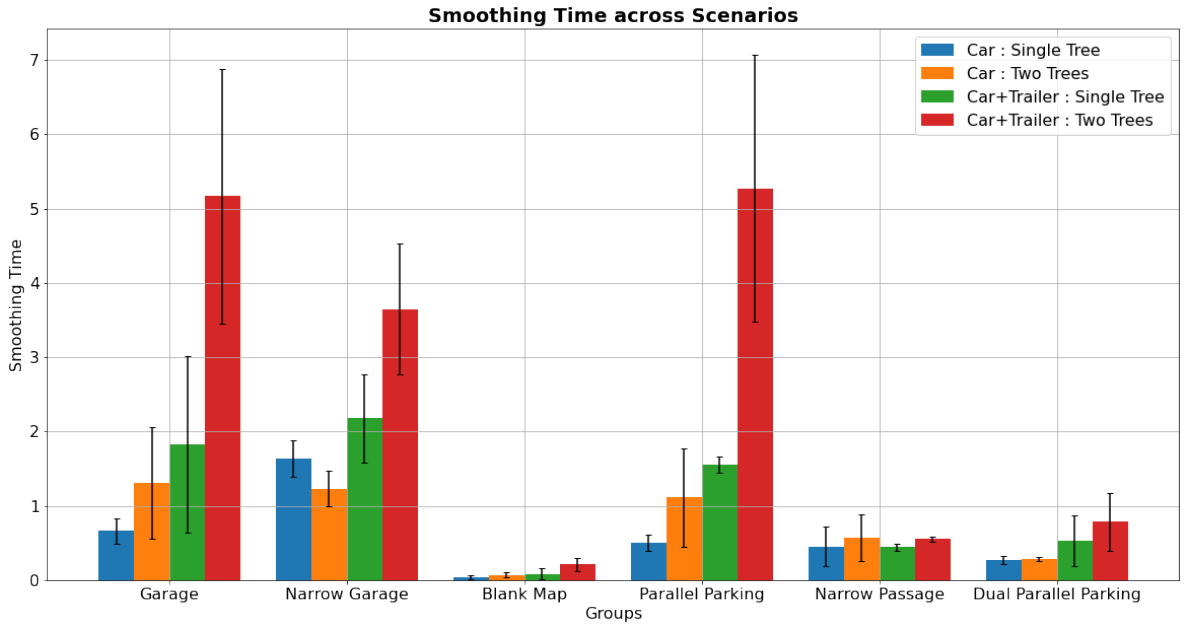


Figure 12: Comparison of Smoothing Time (in s) across Scenarios

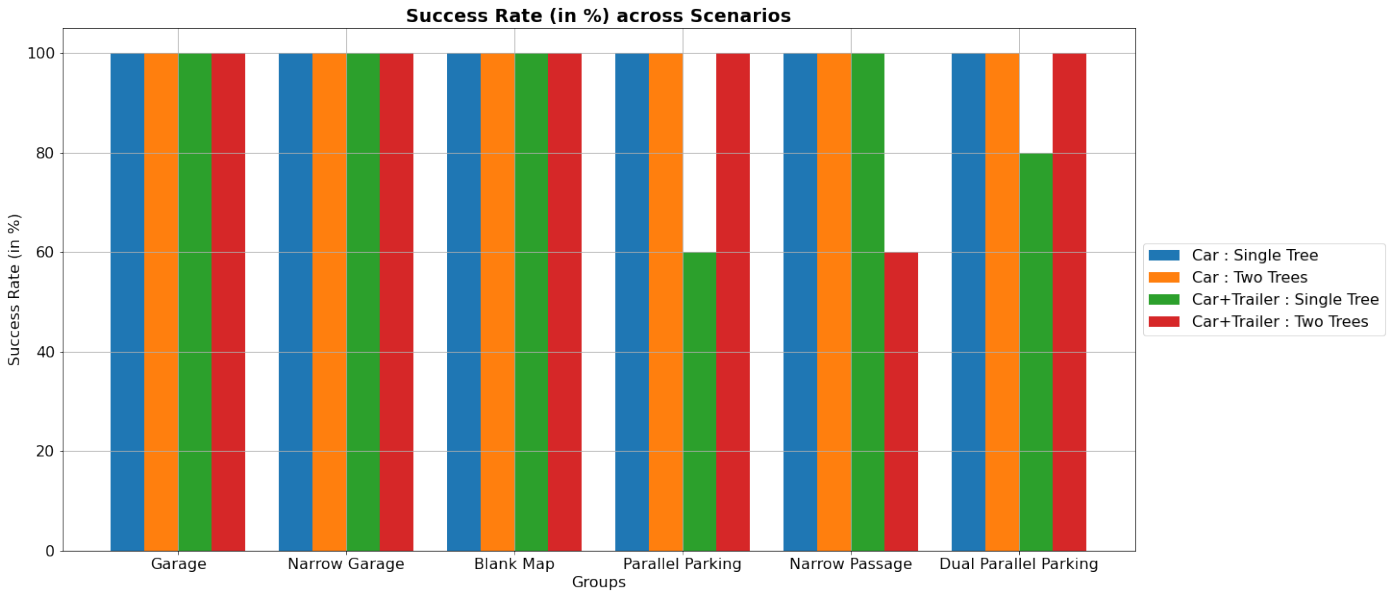


Figure 13: Comparison of Success Rate across Scenarios

1. **Narrow Garage & Narrow Passage–**

   Mainly looking at Figures 10, 11, 12, 13 (and the quantitative results from the previous section) we can see that (especially for the *Car+Trailer* case), the Single Tree CaRRT algorithm performs much better than the two tree version.

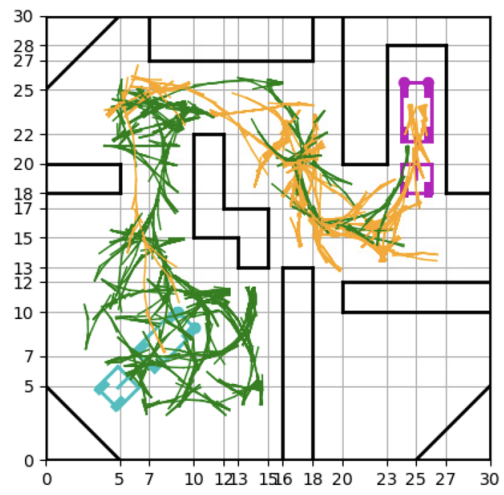


Figure 14: START Tree reaches the GOAL almost without merging (Two Tree CaRRT - Narrow Garage)

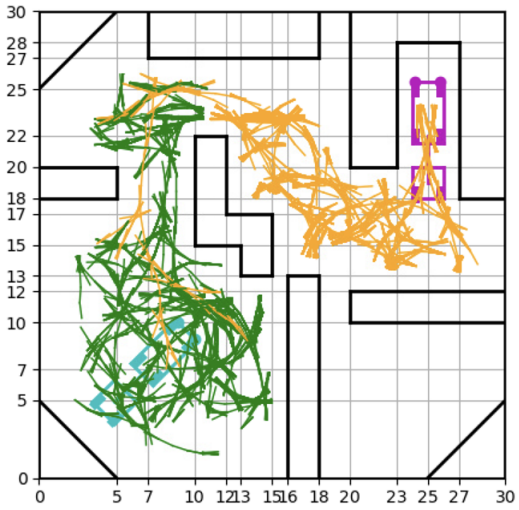


Figure 15: GOAL Tree reaches the START almost without merging (Two Tree CaRRT - Narrow Garage)

From the above figures, we can see that the two trees don't help much in this case, as for Car+Trailer it is very hard both the trees to configuration such that the two trees can merge.

2. **Blank Map**–

   The Blank Map has no obstacles so it takes very less effort for all the algorithms to find a collision free path as expected.

3. **Parallel Parking and Dual Parallel Parking**–

   We observed the the Two Tree CaRRT is *significantly* helpful for improving the parallel parking performance, by observing Figures 10, 11, 12, 13 (and the quantitative results from the previous section). This is because it takes a lot of search for a single tree to find the perfect path for the system to follow in order to park. This is observed in both *Parallel Parking* and *Dual Parallel Parking* maps.

4. **Smoothing Time (in s)**–

   Considering Figure 12, we can see that the Smoothing Times of *Car+Trailer* system is higher than the Smoothing Times for *Car* system. This is expected because the *Car+Trailer* is more constrained comparatively and post-processing by considering all the constraints can be time-consuming. But all the Smoothing Times are very small in comparison to the Planning Times.

5. **Car+Trailer System** –

   As there are additional kinematic and collision based constraints on the *Car+Trailer* system, we can observe in all scenarios,except the ones involving Parallel Parking with Two Tree CaRRT algorithm, the metric values are much higher as compared to just the *Car*. In the parallel parking maps, the Two Tree CaRRT regardless of higher constraints seems to perform very effectively and efficiently.

■ **Some Stills and Path Uniqueness Analysis :**

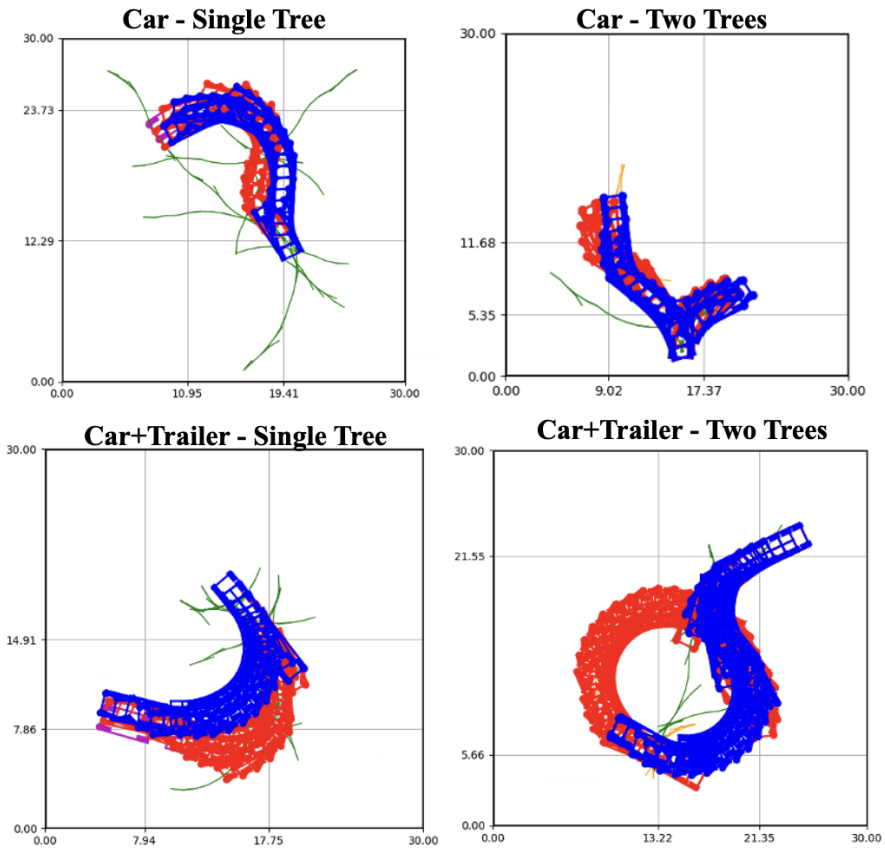


Figure 16: Some Stills of the Blank Map. Both Raw Path (shown in red) and Smooth Path (shown in blue) are displayed.

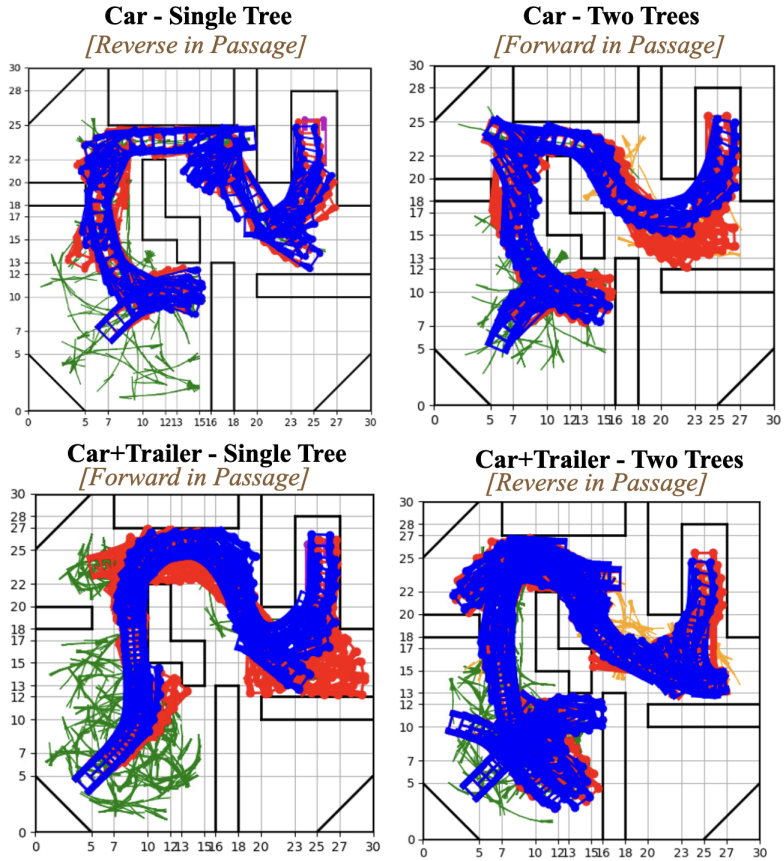


Figure 17: Some Stills of the Narrow Garage Map. Both Raw Path (shown in red) and Smooth Path (shown in blue) are displayed.

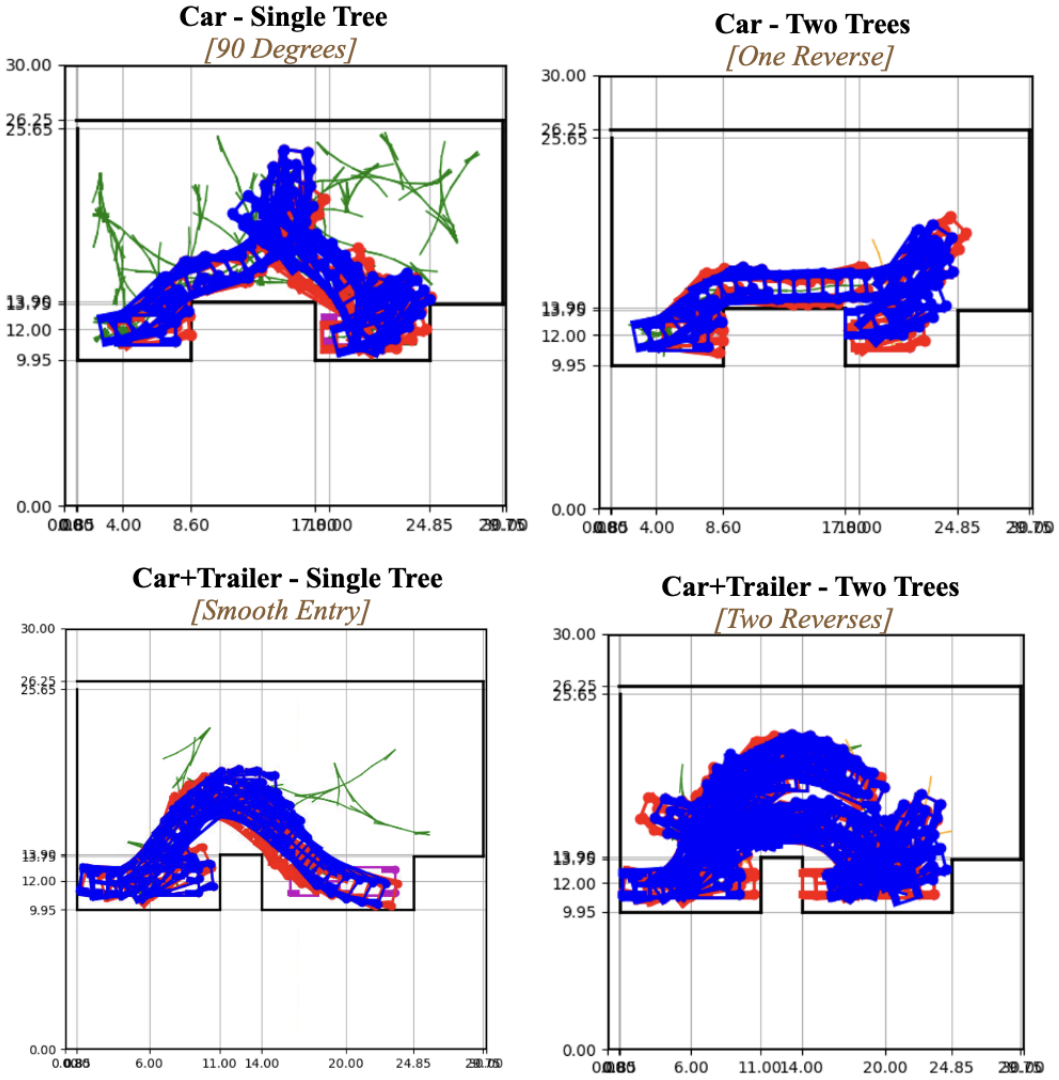


Figure 18: Some Stills of the Dual Parallel Parking Map. Both Raw Path (shown in red) and Smooth Path (shown in blue) are displayed.

Consider the following inferences :

- Some of the stills of the paths found in the Blank Map** as shown in Figure 16.
- Now let us consider the Narrow Garage Map. Here we observed all experiments and we found two “classes" of paths depending upon the orientation of the wheeled system, when it passes through the narrow passage in the map. They are *Forward in Passage* and *Reverse in Passage*. Some of the stills are shown in Figure 17.
- Now let us consider the Dual Parallel Parking Map. Here we observed all experiments and we found four “classes" of paths depending upon the movement of the system in the path. They are *90 degrees* (the system’s orientation changes by 90° in the path), *One Reverse*, *Smooth Entry*, *Two Reverses*. Some of the stills are shown in Figure 19.

Some Results and Inferences for CaRRT* Algorithm

As we mentioned in Section 7, we experimented briefly by introducing a notion of optimality into our Single Tree CaRRT algorithm for Car by considering:

- Storing Costs (like Path Length) in each node.
- Using Rewiring of Tree to find *locally* optimal paths.

**Note that here even the START and GOAL positions are randomized

We found in this case, based on our experiments on Blank Map and Move Over[††] that the algorithm typically runs much slower than the normal CaRRT algorithm but it generally finds path of lower path lengths. Also we noticed that, the growth behaviour of the tree is different and it has less branches, closely spaced nodes and “straighter" paths. Some of the results we obtained are as follows:

Metric / S.No.	Experiment #1		Experiment #2		Experiment #3		Experiment #4		Experiment #5	
Planning Time (in s)	134.55	134.55	-	-	125.08	125.08	340.08	340.08	-	-
Smoothing Time (in s)	-	0.29	-	-	-	0.18	-	0.25	-	-
# Nodes Sampled	31560	31560	103000	103000	29932	29932	31793	31793	107000	107000
# Nodes in Path	44	31	-	-	30	21	40	27	-	-
Tree Size	2436	2436	-	-	2354	2354	2573	2573	-	-
Path Length	14.421	13.178	-	-	17.369	17.055	16.308	17.737	-	-
Path Smoothness	10.855	7.124	-	-	5.649	3.654	8.749	5.219	-	-
Success Rate	Success	Success	Failure	Failure	Success	Success	Success	Success	Failure	Failure

Table 7: Single CaRRT Algorithm for Car in Scenario: Move Over. Values in red correspond to before smoothing whereas values in blue correspond to after smoothing.

Metric / S.No.	Experiment #1		Experiment #2		Experiment #3		Experiment #4		Experiment #5	
Planning Time (in s)	8.55	8.55	152.49	152.49	-	-	-	-	307.27	307.27
Smoothing Time (in s)	-	0.08	-	0.56	-	-	-	-	-	0.52
# Nodes Sampled	5376	5376	32856	32856	125000	125000	104000	104000	45906	45902
# Nodes in Path	15	13	106	58	-	-	-	-	86	44
Tree Size	450	450	2751	2751	-	-	-	-	3842	3842
Path Length	4.68	4.506	17.374	16.674	-	-	-	-	21.809	21.746
Path Smoothness	1.853	1.588	31.184	16.389	-	-	-	-	23.513	10.316
Success Rate	Success	Success	Success	Success	Failure	Failure	Failure		Success	Success

Table 8: Single CaRRT Algorithm for Car in Scenario: Blank Map

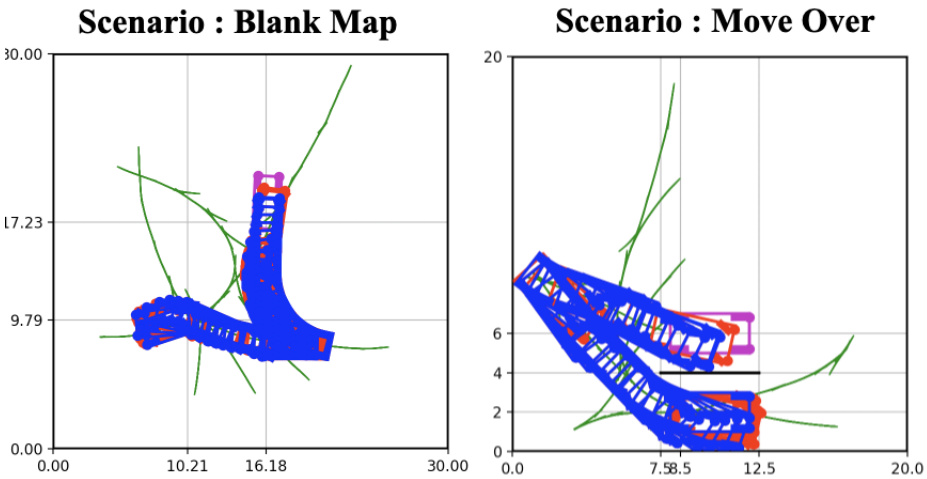


Figure 19: Some Stills of Paths found by CaRRT* Algorithm. Both Raw Path (shown in red) and Smooth Path (shown in blue) are displayed. Values in red correspond to before smoothing whereas values in blue correspond to after smoothing.

[††]The same map as the one given in the Homework Set

IV Conclusion

We successfully built RRT based motion planners for non-holonomic mobile robots for two wheeled systems : *Car* and *Car+Trailer* that can plan a effective path efficiently in challenging maps. These algorithms are initially given a map along with START and GOAL. They use the kinematic model of the wheeled system, an obstacle collision checker and CSC (Curve-Straight-Curve) notion of distance under the hood. We considered variants of this CaRRT algorithm such as Single Tree, Two Tree and an RRT* version. The output of these algorithms is a raw path. A postprocessing function that respects the non-holonomic kinematic constraints of the wheeled system is used to produce smooth, collision-free path. We also recorded multiple quantitative metrics to analyze their performances of both the raw and smooth paths. We analyzed the collected metrics across various experiments.

We presented the inferences from analyses, along with stills and interesting graphical results. We found that the CaRRT algorithm with a single tree and goal biased random sampling performs the best in maps like narrow passage and narrow garage. Also CaRRT algorithm with two trees and almost random sampling performs the best in maps involving parallel parking.Finally, the smoothing algorithm almost always finds a better collision-free path given the raw one.

One avenue of future research is further investigate and the make the CaRRT* algorithm more efficient and incorporate other kind of costs in it, so that we can find smooth, effective and *optimal* (atleast locally) paths.

Autonomous systems like smart cars and planetary rovers need to be able to make decisions quickly and accurately to avoid obstacles and reach their intended destinations. One of the challenges faced by autonomous systems is navigating through complex terrain. This can include navigating through crowded urban environments with pedestrians and other vehicles, or exploring rocky and uneven planetary surfaces. In these situations, it is important for the autonomous system to generate smooth paths that minimize sudden changes in direction or speed, as these can lead to instability or collisions. By using the developed algorithms, autonomous systems can navigate through complex terrain more efficiently and safely.

We would like to acknowledge Professor Günter Niemeyer for the advice on the project topic and his support in general.

The code repository for our project can be found at:

`https://github.com/Claude0311/final133b`

The drive link for the accompanying final video is as follows:

`https://drive.google.com/file/d/1USkcb3FE1OYir1-HiPoQMdeCHV_cHZ2c/view?usp=share_link`