Closed Loop RRT for Truck-Trailer System

Project Report for ME 8843 YEZ

Bhushan Pawaskar
School of Aerospace Engineering
Georgia Institute of Technology
Atlanta, US
bpawaskar3@gatech.edu

***Abstract*—Truck-trailer systems are widely used in various industries for transporting goods. These systems have to function in dynamic warehouse environments and require accurate and efficient planning to ensure a safe operation. But, due to the nonlinear dynamics of the system, traditional motion planning methods may not be suitable for generating feasible paths in real-time. This report proposes the use of a sampling-based motion planning algorithm for a truck trailer system. In particular, using Closed Loop Rapidly Exploring Random Tree (CL-RRT) to generate collision-free paths for the truck-trailer system while performing U-turn maneuver in compact spaces. The report explores the challenges faced in the implementation of such an algorithm on a non-holonomic system. The effectiveness of the proposed method has been shown through experiments in simulation. The results show that several performance benefits can be obtained and the proposed approach can prove to be a promising solution for real-time path planning while ensuring the kino-dynamic constraints are satisfied.**

***Index Terms*—Kinodynamic Planning, Rapidly Exploring Random Trees, Truck-Trailer System, Real-time planning**

I. Introduction

The logistics industry is rapidly evolving, and there is a growing need for a cost-effective transportation of goods from one place to another. Autonomous trucks have emerged as a promising solution to address these challenges, as they can reduce costs by eliminating the need for human drivers. However, the deployment of autonomous trucks in dynamically changing warehouse environments poses several challenges, including motion planning in real-time. These trucks have to plan a collision-free path in order to maneuver through compact spaces. A fast real time planning method is required for this purpose.

Sampling based motion planning methods have become very popular in the past few decades. They provide fast exploration of high-dimensional state spaces and also offer probabilistic completeness guarantees. Rapidly-Exploring Random Trees (RRT) [1] is one such sampling based algorithm used for quickly exploring a complicated, high-dimensional state space. The algorithm offers several performance benefits over other convex optimization based algorithms that allow it to be used for real-time applications. Over the past few decades, RRTs have been used in numerous robotic systems, including DARPA Urban Challenge vehicles, autonomous driving systems used in in public settings, humanoid robots.

However, Vanilla RRT algorithm has several drawbacks that make it not directly usable for real-world systems [4]. Firstly, the algorithm does not provide any optimality guarantees. It just generates a feasible path which may or may not be optimal. In addition, it is highly sensitive to initial conditions and sampling strategy used. This can have a lot of impact on its performance. Also, the algorithm suffers from the curse of dimensionality. As the dimension of the search space increases, it becomes more and more computationally expensive. Lastly, the path generated by RRT only considers a path that connects the discrete points in the state space via the interpolated points in between. It does not respect the kinodynamic constraints of the system and the generated path may or may not be actually feasible.

There are various variants of RRT algorithm that address one or many of these mentioned drawbacks. Bi-driectional RRT starts the tree exploration from both start and goal node resulting in improvement in efficiency of the algorithm. Control based RRT planners for example generate only kinodynamically [2] feasible paths for the system. Algorithms like RRT* [3] address the suboptimality of the paths generated by the algorithm. A cost function is used to guide the growth of the tree towards the goal region and the tree is rewired to ensure optimality.

Closed Loop RRT (CL-RRT) [5] is one such popular alogirthm that was first introduced for team MIT's entry vehicle for the 2007 DARPA Urban Challenge, Talos. For the competition, their team needed to develop an algorithm that was efficient, real-time and could dynamically re-plan with changing obstacles in the environment. This algorithm not only takes into consideration the feasibility of the generated path but also provides rapid exploration of the space by using a closed loop system with a low level controller for sampling the space. Moreover, this method can also provide probabilistic optimality guarantees.

The report is organized into five major sections. The introduction is the first section and talks about the need for using sampling based algorithms like RRT for motion planning, their advantages and drawbacks. The second section is a preliminaries section that gives a high level overview of the CL-RRT algorithm and its key concepts. The third section, Methodology will talk about the truck-trailer vehicle model and the changes needed in adapting the CL-RRT algorithm for that model. Lastly section 4, results and discussion talks about the benefits of using this method over standard approaches and section 5 is conclusion

II. Preliminaries

In this section, we will briefly look at the CL-RRT [5] Algorithm implemented by the MIT team and also cover the key highlights of the paper.

A. Low Level Controller

A stable closed-loop control architecture based on a reference tracking controller is crucial for the CL-RRT algorithm to function. The existing randomized kinodynamic planning algorithms first sample a reference state. Then they find a control input $u(t)$ to reach that state by sampling multiple control inputs $u(t)$ directly or by back-calculating the required input $u(t)$ to reach that state. This is computationally expensive.

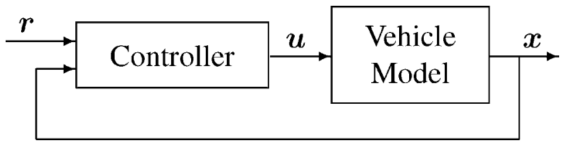


Fig. 1. Closed-loop prediction. Given a reference command , the controller generates high rate vehicle commands to close the loop with the vehicle dynamics

The approach taken by CL-RRT to this is it directly passes in the sampled state to the stable-closed loop system as a reference state. The low level controller then tries to minimize the error and drives the system towards the reference state.

In the case of the MIT vehicle, a bicycle model [8] was used to model the vehicle dynamics and thus the tracking low level controller used for that were the pure pursuit controller [10] for lateral tracking and a simple PI controller for longitudinal tracking. In that case, the reference passed to the controller were the x,y coordinates of the sampled points and the commanded velocity of the controller.

B. Cost heuristics

After the sampling step, RRT attempts to connect the sampled node to the nearest nodes in the tree. Determining the nearest nodes in the tree requires a heuristic function. Typically, $L2$ norm or the euclidean distance is used as the heuristic function. But doing so for a non-holonomic system can be highly inaccurate. Consider the example where the algorithm samples a reference configuration to the side to the vehicle. The euclidean distance would not work in such a case as the vehicle cannot move sideways. It would need to take a longer path to reach that point.

This can be addressed by performing a propogation step to estimate the distance. But doing it online can take lots of computing resources. This challenge was solved in the CL-RRT paper using a different heuristic. They used Dubin's path to estimate the distance to find the closest node. A Dubins path

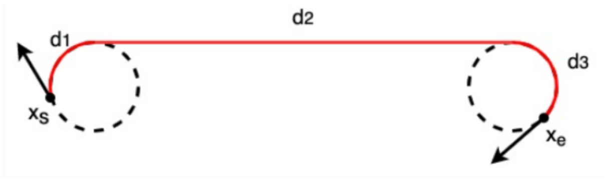


Fig. 2. A sample dubin's path consisting of three components

is a type of curve that connects two configurations of a non-holonomic vehicle, while respecting the vehicle's constraints on motion. The Dubins path is the shortest possible path and it consists of a sequence of three basic maneuvers: a constant radius turn to the left or right, a straight line segment, and another constant radius turn in the opposite direction. The nice thing about this approach is that there is a closed form analytical solution available to estimate the length of the Dubin's path which saves computation time.

C. Smart sampling

The motion planner generally is accompanied by a high level navigator that decides the vehicle behavior. This level of abstraction has more information about the system since it has access to the high level map of the system. The behavior planner knows the kind of action for which a trajectory must be planned. eg. like executing a parallel parking maneuver or executing a left-side U-turn.

This information can be used to make the planning algorithm even more efficient. Based on the kind of maneuver that is being performed we can smartly sample the state space to save time. This can make the algorithm much faster in real-time implementation.

Consider setting the values for two parameters each along two (r (radial) and θ (angular)) dimensions: θ_{mean}, θ_{var}, r_{mean}, r_{var}. Using these, we can define a normal distribution in the polar coordinates to sample points from. A straight maneuver on the highway for example will contain sampling points from a distribution in the front of the vehicle with a narrow field of view. While a left turn maneuver will require a distribution that is wider and at an angle towards the left.

D. Dynamic replanning

This is one of the most important aspects of the algorithm which makes it so much better than its predecessors. Typically, sampling algorithms will try to sample points till a path towards the goal point is found. Only then will the agent start to move as per the trajectory found. CL-RRT on the other hand takes a different approach. The algorithm tries to search for a path for a specific amount of time until it times out. After the time-out the algorithm evaluates the best possible path out of all the possible paths based on a certain heuristic. The agent then proceeds in the direction of that path and starts recomputing the paths again. At any given moment, the agent only plans forward paths for a finite time horizon. It keeps on planning further paths as it executes the best feasible trajectory at any given time step. And then again selects a

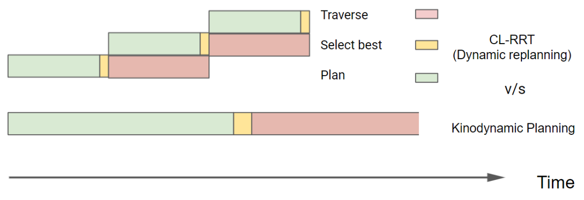


Fig. 3. Finite horizon planning for CL-RRT

new best trajectory to be executed after a particular time-step. Such finite horizon planning not only allows the planner to account for dynamic obstacles but also gives huge performance improvements

III. Methodology

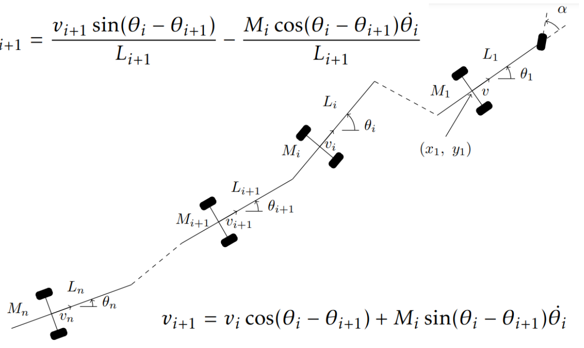
Implementation of CL-RRT on the MIT vehicle used bicycle model to model the vehicle dynamics. To use the algorithm for a truck-trailer system however several changes need to be made to the algorithm to adapt it for our system.

A. Vechicle Model

The truck-trailer system belongs to a class of general n-trailer systems. [13] Using the non-holonomic constraints, a recursive formula can be analytically derived that relates heading and velocity of each trailer to the one before in the n-trailer system using system parameters like:

- M_i: Length of hitch between trailer i and $i+1$
- L_i: Length of trailer i
- v_i: velocity of trailer i
- θ_i: global-heading of trailer i
- α: Steering angle

$$\dot{\theta}_{i+1} = \frac{v_{i+1}\sin(\theta_i - \theta_{i+1})}{L_{i+1}} - \frac{M_i\cos(\theta_i - \theta_{i+1})\dot{\theta}_i}{L_{i+1}}$$



$$v_{i+1} = v_i\cos(\theta_i - \theta_{i+1}) + M_i\sin(\theta_i - \theta_{i+1})\dot{\theta}_i$$

Fig. 4. Figure illustrating general n-trailer system

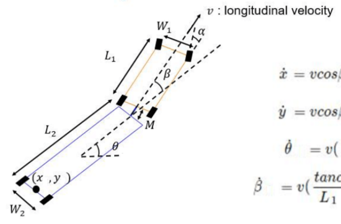
Such a system can be then simplified to a truck and trailer system with only 4 state variables:

- x: x location of rear axle of trailer
- y: y location of rear axle of trailer
- θ: global heading angle of trailer
- β: angle between truck and trailer

The inputs to this model are also obtained in a simplified form:

- v: velocity applied at the rear axle of truck

Truck-trailer Dynamics



$$\dot{x} = v\cos\beta\ (1 + \frac{M_1}{L_1}\tan\beta\ \tan\alpha)\cos\theta$$
$$\dot{y} = v\cos\beta\ (1 + \frac{M_1}{L_1}\tan\beta\ \tan\alpha)\sin\theta$$
$$\dot{\theta} = v(\frac{\sin\beta}{L_2} - \frac{M_1}{L_1 L_2}\cos\beta\ \tan\alpha)$$
$$\dot{\beta} = v(\frac{\tan\alpha}{L_1} - \frac{\sin\beta}{L_2} + \frac{M_1}{L_1 L_2}\cos\beta\ \tan\alpha)$$

Fig. 5. Vehicle model of the truck-trailer system

- α: steering angle

The model thus obtained, is almost similar to the bicycle model in that the inputs required are the same and thus, we can use a pure-pursuit controller for lateral tracking and PID controller for the longitudinal tracking of references.

B. Stabilizing Dynamics

If the system is unstable, The CL-RRT algorithm requires a stabilizing controller to track the references passed into the closed loop system. Even though the truck-trailer system is similar to the bicycle model in some ways, the system is only stabilizing in the forward direction. In reverse direction, the state variable β is not stable.

Use an LQR controller to get $\beta = \beta\text{eq}$

$$\min_{\bar{u}} J = \int_0^\infty \bar{p}^T Q_1 \bar{p} + \bar{u}^T Q_2 \bar{u}\, \mathrm{dt} = \int_0^\infty (\beta - \beta_{\mathrm{eq}})^2 Q + (\alpha - \alpha_{\mathrm{eq}})^2\, \mathrm{dt}$$

$$\alpha_{\mathrm{cmd}} = \alpha_{\mathrm{des}} + \alpha_{\mathrm{feed}} = \alpha_{\mathrm{des}} - K\left(\beta - \beta_{\mathrm{eq}}\left(\alpha_{\mathrm{des}}\right)\right)$$

Fig. 6. Using an LQ controller to stabilize β

Thus, a pure-pursuit controller cannot be used directly for the reverse motion. So we use a gain-scheduled LQ controller [11] to stabilize β to an equilibrium value β_e for the reverse motion.

C. Cost Heuristics

A dubin's path proved to be computationally efficient quick way of estimating the costs to the nearest node for the bicycle model. The truck-trailer model however cannot use such a heuristic due to the dynamics being much more complex and unstable in the reverse direction.

A modified dubin's path [13] can be used with an approach arc but computing the approach arc for each of the nodes turns out to be expensive. So we consider using lookup tables. Lookup tables are pre-computed offline and contain information about the estimated distance to each point in a discretized grid map. While running the algorithm online, the only computation that needs to be made is interpolating the values from the lookup tables.

The pure-pursuit controller is made to follow different end positions on a discretized grid map for a maximum of 1000 time steps. The tolerances for reaching the goal are relaxed in order to get estimates for points which can't be exactly reached. The cost to each point is then calculated as per the given equation [7]:

$$\Gamma_{\text{rev}} = \sum_{i=1} 10(x_2[i] - x_2[i-1])^2 + 10(y_2[i] - y_2[i-1])^2 + 10^4(\beta[i] - \beta[i-1])^2$$

$$\Gamma_{\text{fwd}} = \sum_{i=1} 10(x_1[i] - x_1[i-1])^2 + 10(y_1[i] - y_1[i-1])^2$$

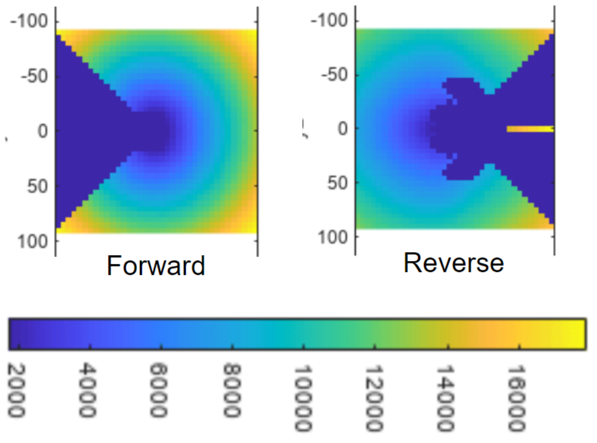


Fig. 7. Estimated costs on the discretized gridmap

D. Dynamic replanning

The CL-RRT algorithm has two major steps of functioning:

- Execution loop of RRT
- Expansion of tree

In the execution loop, the RRT algorithm chooses the best node at any given time-step and executes the trajectory towards that node. In the tree-expansion loop, the algorithm rapidly samples new points and tries to expand the tree to explore the state space. During run-time, these steps occur simultaneously and thus save a lot on computation time. However, due to limitation of computational resources, running both of these loops in parallel is not possible. So, we demonstrate the proof of concept by running the loops one after another. When the tree expansion step is running, the time is frozen so that the execution loop does not skip ahead.

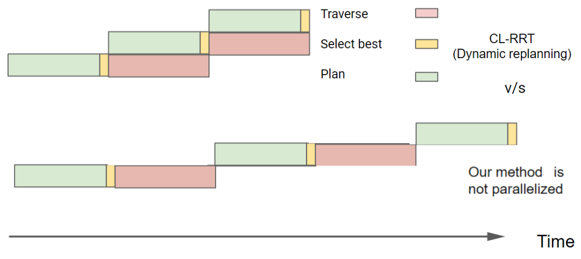


Fig. 8. Comparing CL-RRT execution to our method (not parallelized)

We can still obtain results to evaluate if this finite horizon planning gives better performance or not, however it should be noted that the time obtained in these results is not the actual time but a modified time that assumes the two loops of tree expansion and RRT propagation have taken place simultaneously.

E. Experiments

The goal of the project was to see how well a truck-trailer system could perform a u-turn maneuver in compact spaces. This was chosen as the goal because a u-turn maneuver requires going back and forth to be executed. And the reverse dynamics of the truck are not stabilizing. So this makes it challenging for the truck to execute. To test this, the following test environments were made in MATLAB. The first is a larger environment with two U-turn maneuvers in the opposite direction. The second has just a single maneuver in one direction.

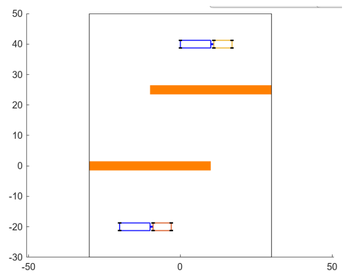


Fig. 9. Test environment 1: used to evaluate performance of the truck-trailer system for various timesteps

- In the first experiment, we ran an iteration of control based RRT to see how much time does the algorithm take to plan and then traverse to its end location. This was then compared to the CL-RRT timing for the same loop for two different time steps of 8 seconds and 15 seconds respectively.
- In the second experiment, points were sampled only from a specific region which might make a left u-turn easier. This was done to evaluate the effectiveness of smart sampling. These were sampled from a two regions of
1. $\theta_{mean} = \pi/4$, $\theta_{var} = 2\pi/3$, $r_{mean} = 12$, $r_{var} = 8$
2. $\theta_{mean} = -\pi/4$, $\theta_{var} = 2\pi/3$, $r_{mean} = 18$, $r_{var} = 12$

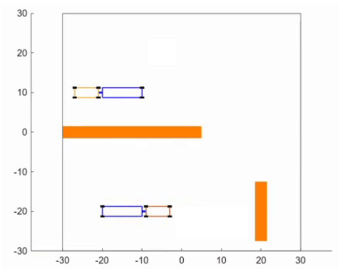


Fig. 10. Test environment 2: used to compare performance of smart sampling vs uniform sampling

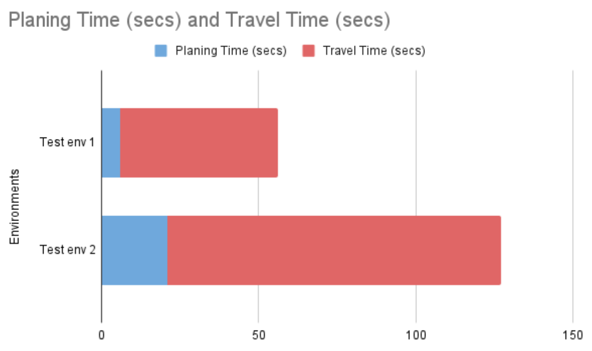


Fig. 11. Comparison of average time for both environments

IV. Results and Discussion

In general, the results were pretty consistent for the smaller environment with just one maneuver to be performed. This was because the solver was able to find a solution relatively quickly. On the other hand, the results for the larger environment were inconsistent and more dependent on the RNG. Hence, all results shown in this report have been averaged over 5 different rng runs.

A. Experiment 1

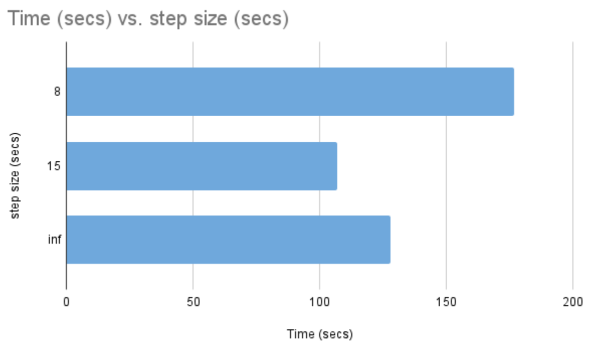


Fig. 12. Comparison of CL-RRT for various time steps

The time-step of the finite horizon for CL-RRT is an important parameter and can decide how well the algorithm performs. If the value is too large, then the algorithm performs like other kinodynamic planners and takes long time to initialize. This also makes it slow to respond to changes in the environment. If the value is too small, the algorithm cannot plan for long term, causing the agent to travel around until it gets lucky and finds a path.

This can be seen in the results. If the values for the time step is 15 seconds, time is saved as the agent can simultaneously plan as it travels v/s in the 'inf' case where the agent first finds a path and then starts to travel. But if the time step is further decreased to 8 seconds, the total time increases as the agent selects paths without long-term planning.

B. Experiment 2

It is expected that by sampling smartly, the system would perform better. However, the system tends to perform slightly worse or similar when sampling space is restricted to only a smaller region. This could be attributed to either bad selection of parameters for sampling or that sampling like this makes the system highly sensitive to rng, since if the system selects a node that takes it too far to one side, it has far lesser chance of sampling a node to compensate for that.

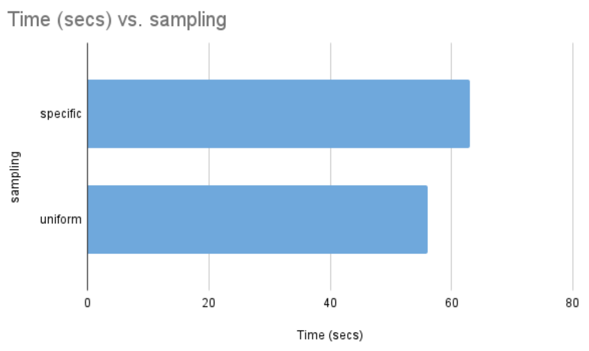


Fig. 13. Comparison of CL-RRT for various time steps

V. Conclusion

The performance of CL-RRT is inconsistent. However, if lots of samples can be generated to cover most of the sample space, the results become more and more consistent as seen with the smaller test environment. CL-RRT did show some improvements in the time required for planning and execution however the method is highly dependent on getting the time-step value right. The optimizations mentioned in sampling did not seem to yield successful results at least in the test environment.

A. Limitations

- The current method recomputes the tree every time at each time step. Ideally, CL-RRT should reuse the previous tree for efficient computation. However, due to the tree data structure being implemented in MATLAB back-end as a C++ object, it was difficult to customize it without breaking the dependencies. If the tree is reused, better performance is expected for lower time step values, but this could not be implemented. The same applies for the state validator object which is responsible for collision with obstacles. Ideally, it would have been nice to test with obstacles that move with time but it was not feasible to modify it easily.
- The lookup tables are only 2 dimensional and do not consider the orientation of the truck while estimating distance. Even though this somewhat works, better results could be obtained with 3 dimensional lookup tables for x,y and θ.

References

[1] LaValle, Steven M.. “Rapidly-exploring random trees : a new tool for path planning.” The annual research report (1998): n. pag.

[2] LaValle SM, Kuffner JJ. Randomized Kinodynamic Planning. The International Journal of Robotics Research. 2001;20(5):378-400. doi:10.1177/02783640122067453

[3] Sampling-based Algorithms for Optimal Motion Planning - Sertac Karaman, Emilio Frazzoli https://doi.org/10.48550/arXiv.1105.1186

[4] M. Elbanhawi and M. Simic, ”Sampling-Based Robot Motion Planning: A Review,” in IEEE Access, vol. 2, pp. 56-77, 2014, doi: 10.1109/ACCESS.2014.2302442.

[5] Kuwata, Y. et al. “Real-Time Motion Planning With Applications to Autonomous Urban Driving.” Control Systems Technology, IEEE Transactions on 17.5 (2009): 1105-1118. © 2009 Institute of Electrical and Electronics Engineers

[6] Article: https://www.mathworks.com/help/nav/ref/plannercontrolrrt.html

[7] Article: https://www.mathworks.com/help/nav/ug/reverse-capable-motion-planning-for-tractor-trailer-model-using-plannercontrolrrt.html

[8] Path Planning using a Dynamic Vehicle Model: Romain Pepy, Alain Lambert and Hugues Mounier

[9] Article: https://www.mathworks.com/help/mpc/ug/truck-and-trailer-automatic-parking-using-multistage-mpc.html

[10] Implementation of the Pure Pursuit Path tracking Algorithm R. Craig Conlter CMU-RI-TR-92-01

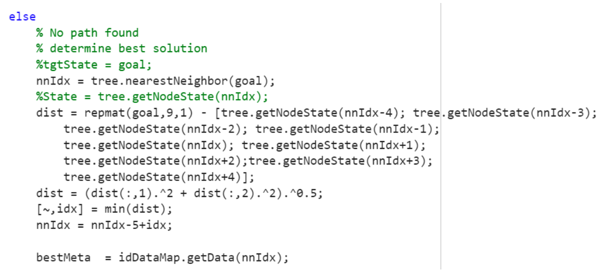
[11] Motion planning for a reversing general 2-trailer configuration using Closed-Loop RRT Niclas Evestedt1, Oskar Ljungqvist1, Daniel Axehill

[12] Motion Planning for a Reversing Full-Scale Truck and Trailer System - Olov Holmer

[13] Closed-Loop-RRT* path planning for a vehicle-trailer system - Edvard Grødem

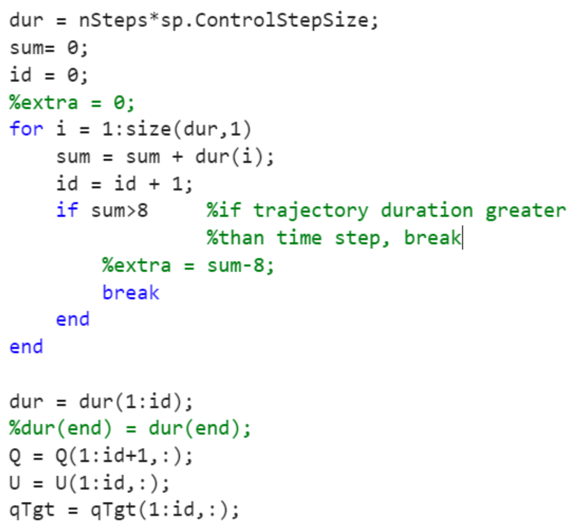
[14] A. C. Manav and I. Lazoglu, ”A Novel Cascade Path Planning Algorithm for Autonomous Truck-Trailer Parking,” in IEEE Transactions on Intelligent Transportation Systems, vol. 23, no. 7, pp. 6821-6835, July 2022, doi: 10.1109/TITS.2021.3062701.

VI. Appendix



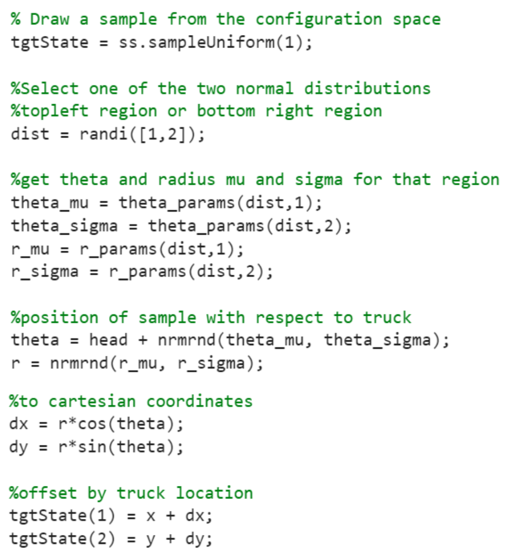
```
else
    % No path found
    % determine best solution
    %tgtState = goal;
    nnIdx = tree.nearestNeighbor(goal);
    %State = tree.getNodeState(nnIdx);
    dist = repmat(goal,9,1) - [tree.getNodeState(nnIdx-4); tree.getNodeState(nnIdx-3);
        tree.getNodeState(nnIdx-2); tree.getNodeState(nnIdx-1);
        tree.getNodeState(nnIdx); tree.getNodeState(nnIdx+1);
        tree.getNodeState(nnIdx+2);tree.getNodeState(nnIdx+3);
        tree.getNodeState(nnIdx+4)];
    dist = (dist(:,1).^2 + dist(:,2).^2).^0.5;
    [~,idx] = min(dist);
    nnIdx = nnIdx-5+idx;

    bestMeta  = idDataMap.getData(nnIdx);
```

Fig. 14. Selection of best available node if path not found



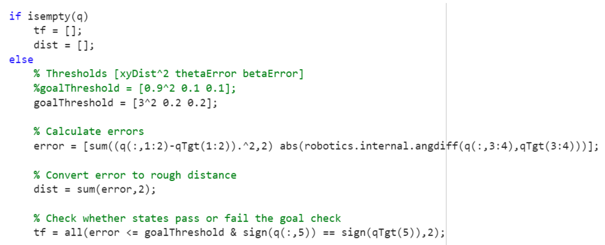
```
dur = nSteps*sp.ControlStepSize;
sum= 0;
id = 0;
%extra = 0;
for i = 1:size(dur,1)
    sum = sum + dur(i);
    id = id + 1;
    if sum>8     %if trajectory duration greater
                 %than time step, break
        %extra = sum-8;
        break
    end
end

dur = dur(1:id);
%dur(end) = dur(end);
Q = Q(1:id+1,:);
U = U(1:id,:);
qTgt = qTgt(1:id,:);
```

Fig. 15. Follow trajectory towards best node but only for a fixed time step



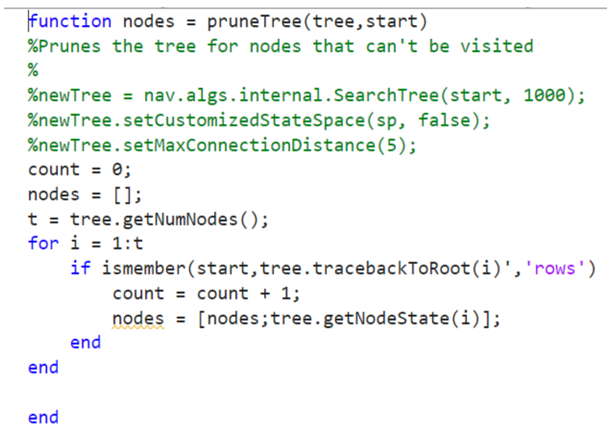
```
% Draw a sample from the configuration space
tgtState = ss.sampleUniform(1);

%Select one of the two normal distributions
%topleft region or bottom right region
dist = randi([1,2]);

%get theta and radius mu and sigma for that region
theta_mu = theta_params(dist,1);
theta_sigma = theta_params(dist,2);
r_mu = r_params(dist,1);
r_sigma = r_params(dist,2);

%position of sample with respect to truck
theta = head + nrmrnd(theta_mu, theta_sigma);
r = nrmrnd(r_mu, r_sigma);

%to cartesian coordinates
dx = r*cos(theta);
dy = r*sin(theta);

%offset by truck location
tgtState(1) = x + dx;
tgtState(2) = y + dy;
```

Fig. 16. Sample points from only specific regions around the truck



```
if isempty(q)
    tf = [];
    dist = [];
else
    % Thresholds [xyDist^2 thetaError betaError]
    %goalThreshold = [0.9^2 0.1 0.1];
    goalThreshold = [3^2 0.2 0.2];

    % Calculate errors
    error = [sum((q(:,1:2)-qTgt(1:2)).^2,2) abs(robotics.internal.angdiff(q(:,3:4),qTgt(3:4)))];

    % Convert error to rough distance
    dist = sum(error,2);

    % Check whether states pass or fail the goal check
    tf = all(error <= goalThreshold & sign(q(:,5)) == sign(qTgt(5)),2);
```

Fig. 17. Thresholds for checking if goal is reached or not



```
function nodes = pruneTree(tree,start)
%Prunes the tree for nodes that can't be visited
%
%newTree = nav.algs.internal.SearchTree(start, 1000);
%newTree.setCustomizedStateSpace(sp, false);
%newTree.setMaxConnectionDistance(5);
count = 0;
nodes = [];
t = tree.getNumNodes();
for i = 1:t
    if ismember(start,tree.tracebackToRoot(i)','rows')
        count = count + 1;
        nodes = [nodes;tree.getNodeState(i)];
    end
end

end
```

Fig. 18. Tree pruning function but implementation after exporting tree as a matlab object. Slow implementation. Worse than remaking the tree.