RBE550 Motion Planning

Prof. Daniel Flickinger

Assignment 4 Valet

Submitted by:

Shreyansh Goyal

Contents

Environment setup 3
- Collision detector 4

Delivery Robot 4
- Path planning 4
- Algorithm: 4
- Planned path: 6
- Differential Drive Kinematics 6
- Simulation 7

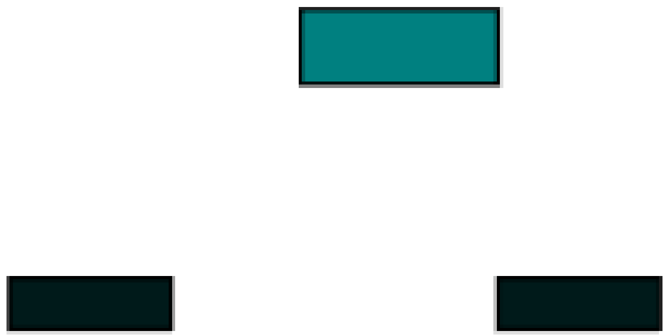
The Car 8
- Path Planning 8
- Ackermann Kinematics 8
- Simulation: 10

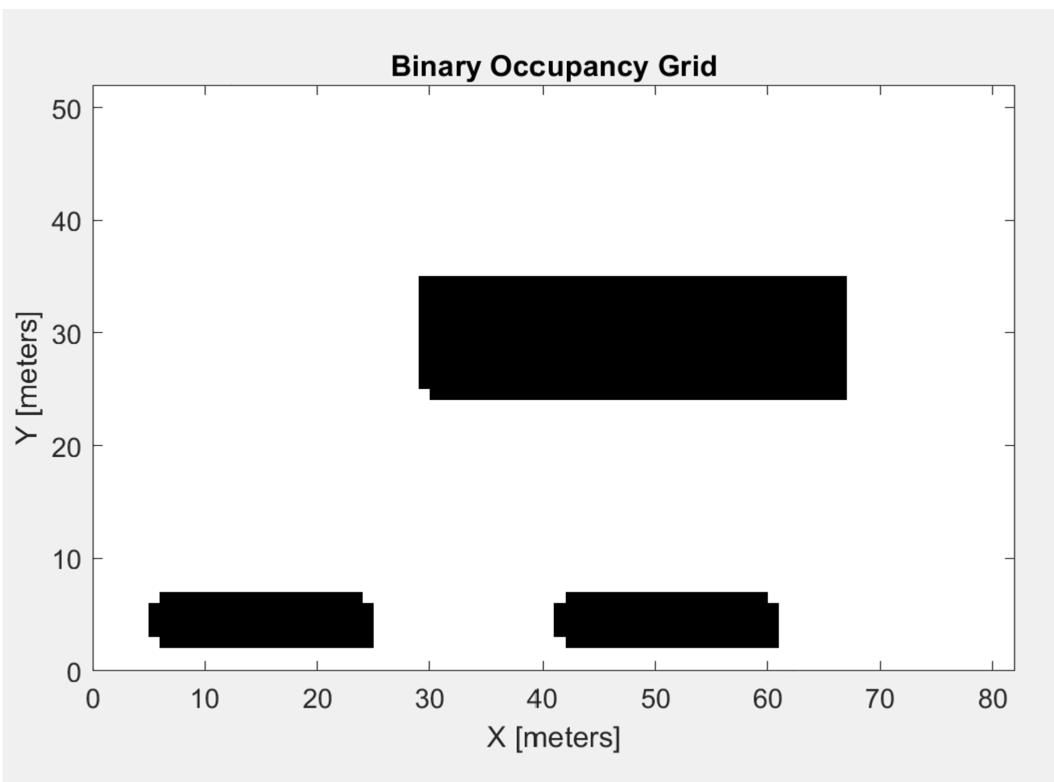
Truck 11
- Collision check: 11
- Kinematics: 12
- Control laws: 13
- Path Planning: 13
- Algorithm: 13
- Simulation: 15

References: 16

Environment setup

Firstly, the obstacles are created using rectangle function in MATLAB and then the environment for this project is obtained by taking the input map image and then using the imread function to create the binary occupancy grid.





Collision detector
For collision check, we can inflate the obstacles for the vehicle to avoid them and move without colliding.

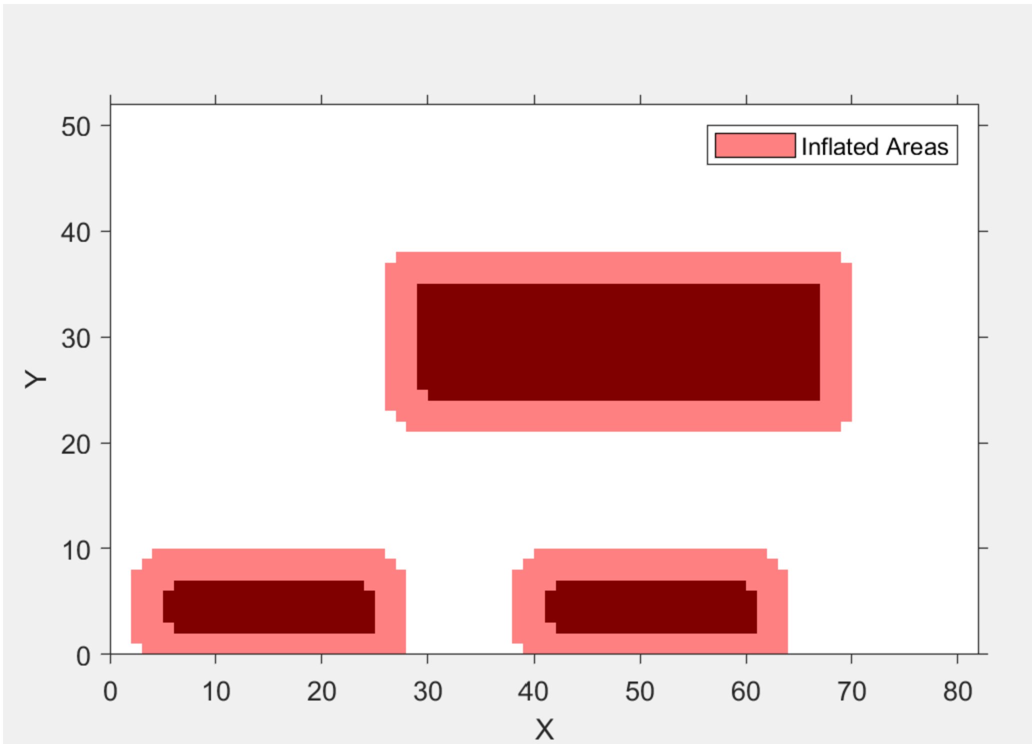


Figure 1 Collision check

Delivery Robot

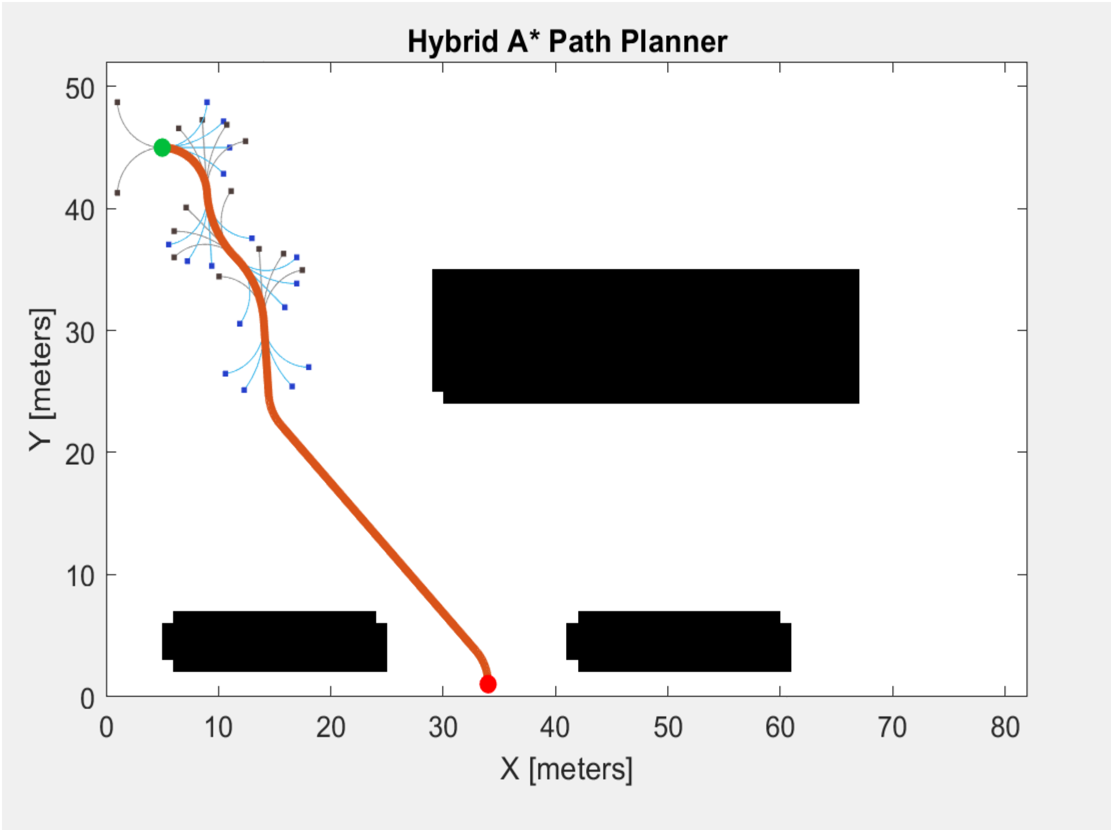
Path planning
In this problem I have used Hybrid A* algorithm to a nonholonomic mobile outdoor robot in order to plan near optimal paths. Unlike the regular A*, the Hybrid A* algorithm is capable of taking into account the continuous nature of the search space representing the real world.

Algorithm:
Each node contains some bookkeeping data for the A* algorithm: a back pointer np to its parent for the final path reconstruction and the priority queue key f (containing the sum of real costs g and the estimated costs h to the goal). θs is the start heading. The open set O contains the neighboring nodes of nodes already expanded during the search. The closed set C contains all nodes which have been conclusively processed. [1]

```
procedure PLANPATH($m, \mu, x_s, \theta_s, G$)
    $n_s \leftarrow (\tilde{x}_s, \tilde{\theta}_s, x_s, 0, h(x_s, G), -)$
    $O \leftarrow \{n_s\}$
    $C \leftarrow \emptyset$
    while $O \neq \emptyset$ do
        $n \leftarrow$ node with minimum $f$ value in $O$
        $O \leftarrow O \setminus \{n\}$
        $C \leftarrow C \cup \{n\}$
        if $n_x \in G$ then
            return reconstructed path starting at $n$
        else
            UPDATENEIGHBORS($m, \mu, O, C, n$)
        end if
    end while
    return no path found
end procedure

procedure UPDATENEIGHBORS($m, \mu, O, C, n$)
    for all $\delta$ do
        $n' \leftarrow$ succeeding state of $n$ using $\mu(n_\theta, \delta)$
        if $n' \notin C$ then
            if $m_o(n'_{\tilde{x}}) =$ obstacle then
                $C \leftarrow C \cup \{n'\}$
            else if $\exists n \in O : n_{\tilde{x}} = n'_{\tilde{x}}$ then
                compute new costs $g'$
                if $g' < g$ value of existing node in $O$ then
                    replace existing node in $O$ with $n'$
                end if
            else
                $O \leftarrow O \cup \{n'\}$
            end if
        end if
    end for
end procedure
```

Planned path:



Differential Drive Kinematics

The differential drive model state is [x y ϑ].

Variables:

- x: Global vehicle x-position, in meters
- y: Global vehicle y-position, in meters
- ϑ: Global vehicle heading, in radians
- $\dot{\phi}_L$: Left wheel speed in meters/s
- $\dot{\phi}_R$: Right wheel speed in meters/s
- r: Wheel radius in meters
- d: Track width in meters
- v: Vehicle speed in meters/s
- ω: Vehicle heading angular velocity in radians/s

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

where, $v = r(\dot{\phi}_R + \dot{\phi}_L)/2$ *and* $\omega = r(\dot{\phi}_R - \dot{\phi}_L)/2d$

The differential drive kinematics equations model a vehicle [2] where the wheels on the left and right may spin independently using the differentialDriveKinematics object in MATLAB.

For this problem, controller pure pursuit is used. The controllerPurePursuit System object in MATLAB creates a controller object used to make a differential-drive vehicle follow a set of waypoints. The object computes the linear and angular velocities for the vehicle given the current pose.

```
[v, omega] = controller(robotCurrentPose)
```

Now, the derivative function is used to obtain the time derivative of the current vehicle state. It returns the three-state vector as a three-element vector [*xDot yDot thetaDot*] which is used to update the vehicle's current pose.

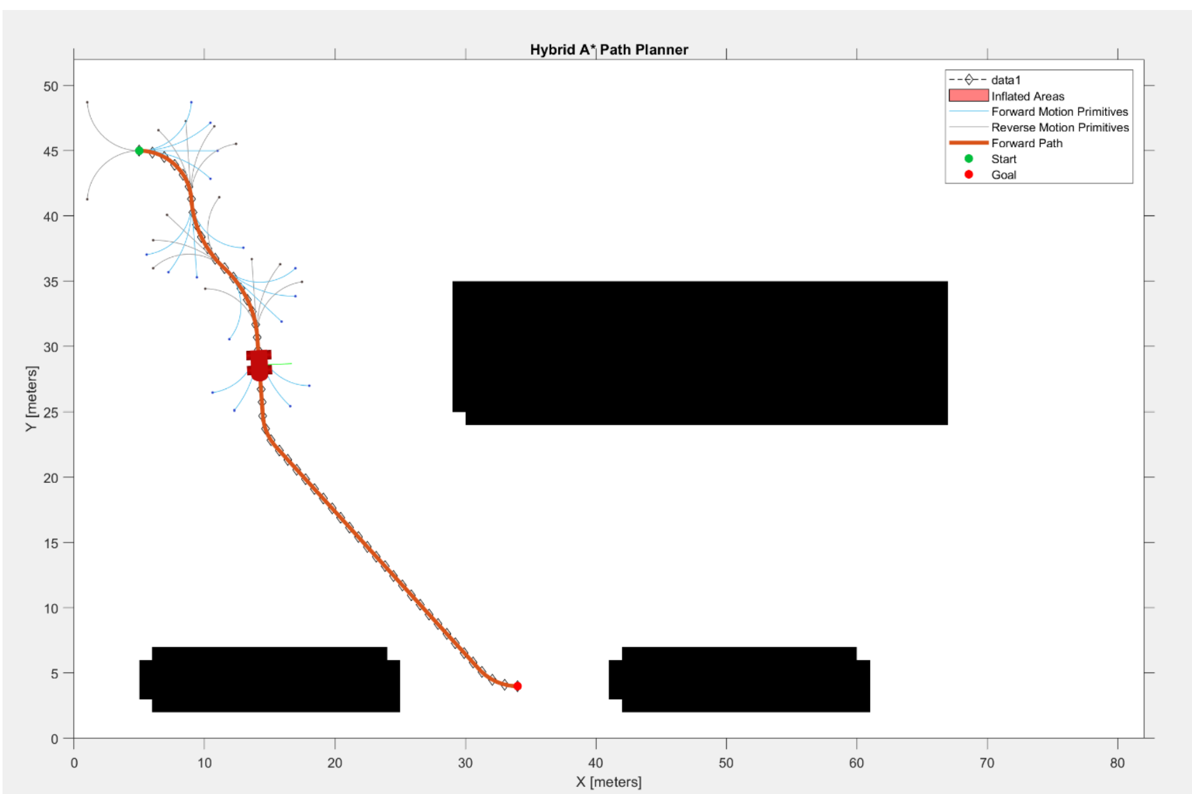
```
stateDot = derivative(robot, robotCurrentPose, [v omega]);
```

Finally, the output of the derivative is used to update the current pose of the vehicle.

```
robotCurrentPose = robotCurrentPose + stateDot*sampleTime;
```

Simulation

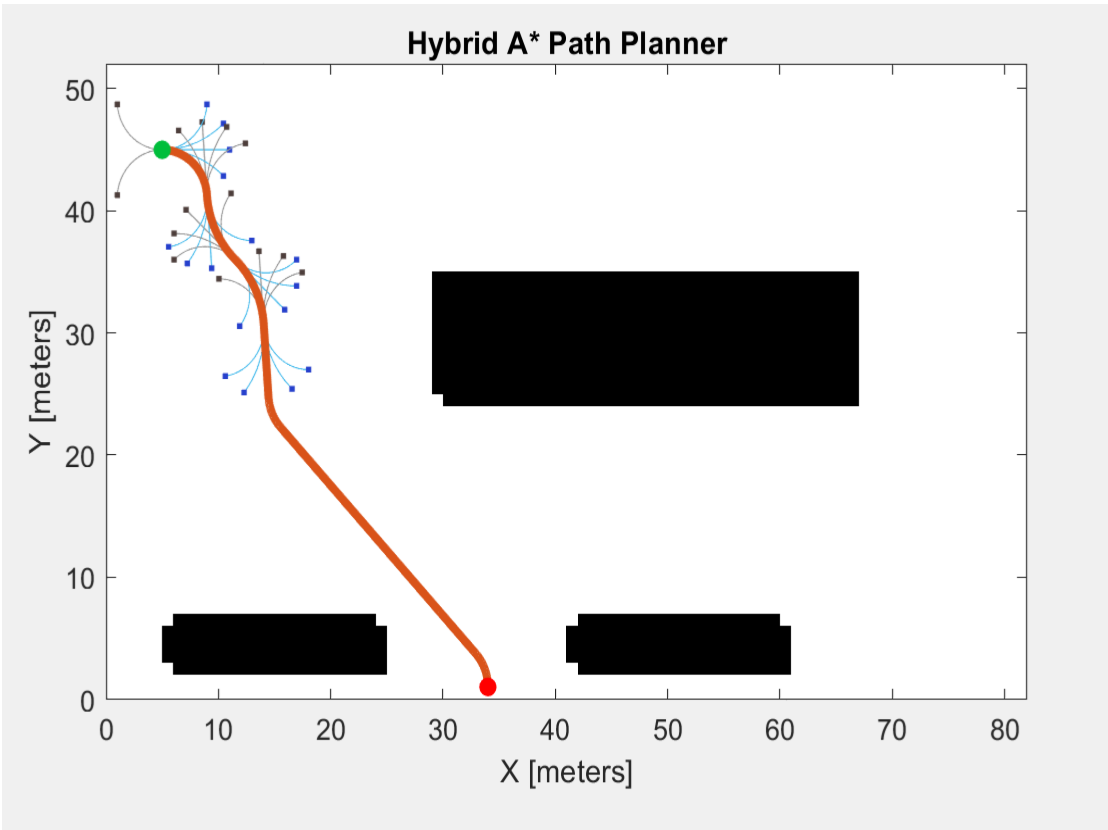
Simulation is done in MATLAB using the groundvehicle.stl file which is the input argument to the plotTranforms function. Video for the simulation has been attached with the submission.



The Car

Path Planning

The path planning is done in the same environment as for the delivery bot with the help of the Hybrid A* algorithm as explained in the previous part.



Ackermann Kinematics

The drive model state is [x y ϑ ψ]. The Variables to be used are described below.

- x: Global vehicle x-position in meters
- y: Global vehicle y-position in meters
- ϑ: Global vehicle heading in radians
- ψ: Vehicle steering angle in radians
- l: Wheel base in meters
- v: Vehicle speed in meters/s

Kinematic equation for Ackermann model:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ \tan(\psi)/l & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \dot{\psi} \end{bmatrix}$$

In MATLAB ackermannKinematics creates a car-like vehicle model that uses Ackermann steering. The state of the vehicle is defined as a four-element vector, [*x y theta psi*], with a global *xy*-position, specified in meters. The vehicle heading, *theta*, and steering angle, *psi* are specified in radians. Wheelbase (l) is 2.8 m.

```
robot = ackermannKinematics("WheelBase",2.8);
```

For this problem, controller pure pursuit is used. The controllerPurePursuit System object in MATLAB creates a controller object used to make a vehicle follow a set of waypoints. The object computes the linear and angular velocities for the vehicle given the current pose. Controller output now will be the linear velocity of the vehicle and the angular velocity of steering.

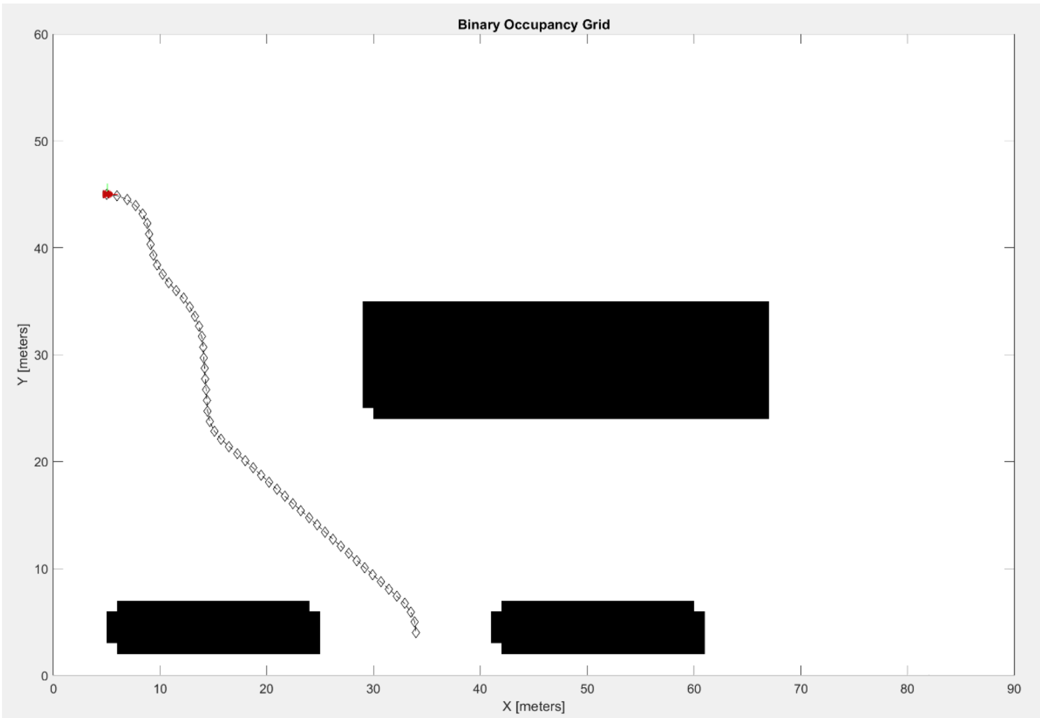
```
[v, psiDot] = controller(robotCurrentPose)
```

Now, the derivative function is used to obtain the time derivative of the current state of the vehicle. It returns state as a four-element vector, [*xDot yDot thetaDot psiDot*]. *xDot* and *yDot* refer to the vehicle velocity, specified in meters per second. *thetaDot* is the angular velocity of the vehicle heading and *psiDot* is the angular velocity of the vehicle steering, both specified in radians per second.

```
stateDot = derivative(robot, currentState, [v psiDot]);
```

Simulation:

The simulation is done similarly as for the delivery robot using the groundvehicle.stl file with wheelbase 2.8m. See the attached video.



Truck

Environment:

The environment is represented by a list of rectangles with their locations, orientations, and dimensions specified in a 1-by-N structure.

```
Length1='Length'; value1= {40};
Width1='Width'; value2={15};
Theta1='Theta'; value3={0};
XY1='XY';value4=[0 0];
Length='Length'; value5= {14};
Width='Width'; value6={8};
Theta='Theta'; value7={0};
XY='XY';value8=[-30 -34];

obstacles=[struct(Length1,value1,Width1,value2,Theta1,value3,XY,value4),struct(Length
,value5,Width,value6,Theta,value7,XY,value8)];
```

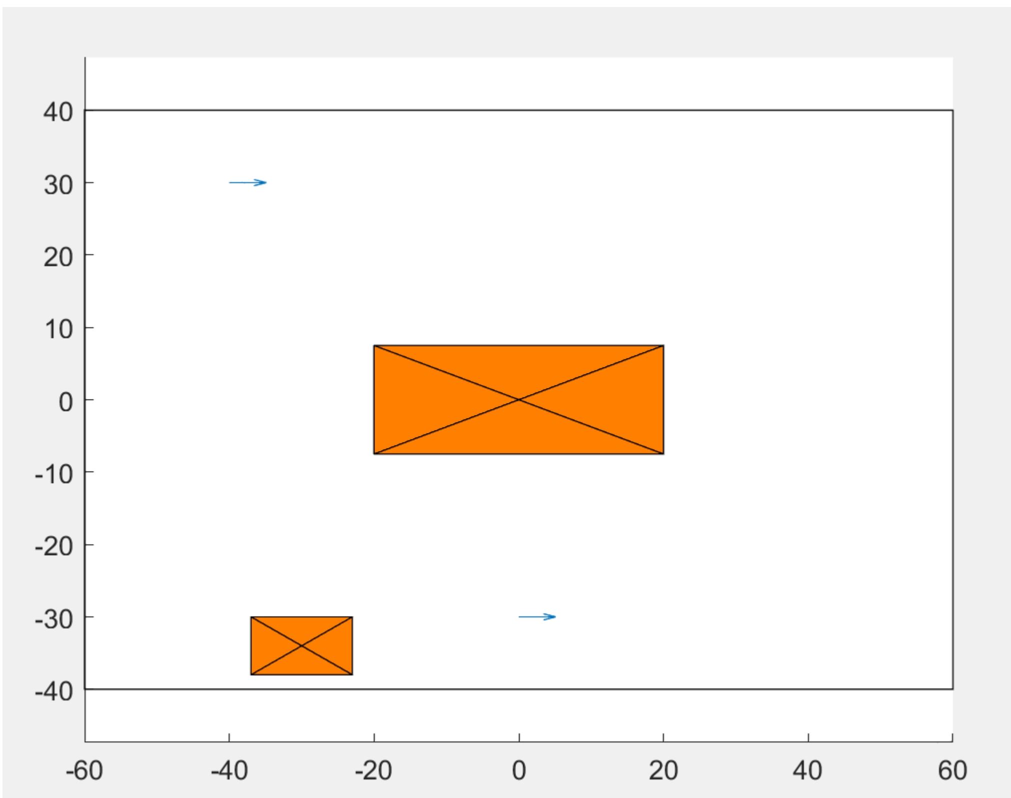
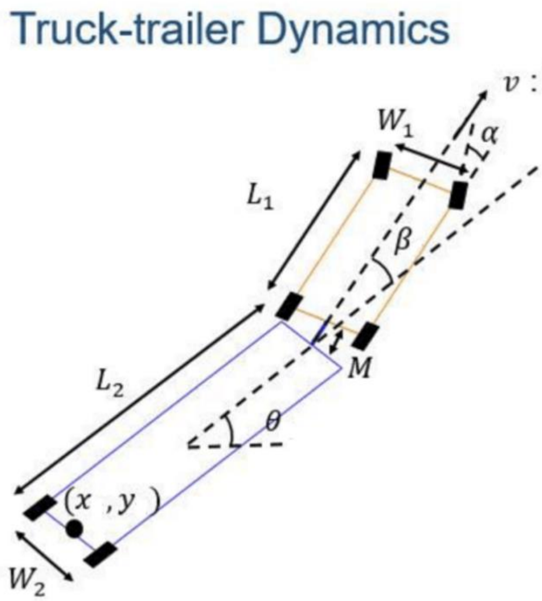


Figure 2 Environment with obstacles

Collision check:

The state validator uses oriented bounding boxes (OBBs) to verify whether the vehicle is in collision with the environment. In MATALB this is done using the exampleHelperTruckValidator custom class which inherit nav.StateValidator subclass.

Kinematics:



$$\dot{x} = v\cos\beta\ \left(1 + \frac{M_1}{L_1}\tan\beta\ \tan\alpha\right)\cos\theta$$

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

$$\dot{\theta} = v\left(\frac{\sin\beta}{L_2} - \frac{M_1}{L_1\,L_2}\cos\beta\ \tan\alpha\right)$$

$$\dot{\beta} = v\left(\frac{\tan\alpha}{L_1} - \frac{\sin\beta}{L_2} + \frac{M_1}{L_1\,L_2}\cos\beta\ \tan\alpha\right)$$

The states for this model are:

1. x (center of the trailer rear axle, global x position)
2. y (center of the trailer rear axle, global y position)
3. theta (trailer orientation, global angle, 0 = east)
4. beta (truck orientation with respect to trailer, 0 = aligned)

The inputs for this model are:

1. alpha (truck steering angle)
2. v (truck longitudinal velocity)

Parameters:

- M (hitch length)
- L1 (truck length)
- W1 (truck width)
- L2 (trailer length)
- W2 (trailer width)
- Lwheel (wheel diameter)
- Wwheel (wheel width)

State-Transition function:

Use the geometric configuration for a two-body truck-trailer system expressed as:

$q_{sys}=[x_2\ y_2\ \theta_2\ \beta]$

For planning purposes, append a direction flag, v_{sign}, and the total distance traveled from the start configuration, s_{tot} to the state vector, for the final state notation:

$q=[x_2\ y_2\ \theta_2\ \beta\ v_{sign}\ s_{tot}]$

Control laws:

At the top level, a pure pursuit controller calculates a reference point, between a current pose and goal state. The controller has two modes, for forward and reverse. For forward motion, the controller calculates a steering angle that, when held constant, drives the rear axle of the truck along an arc that intersects the reference point.[3]

LQ Feedback Stabilization:

The LQ controller is used to calculate the desired steering angle and feedback gains. The gains are the optimal solution to the Algebraic Ricatti equation, stored as a lookup table dependent on the desired steering angle as mentioned in the above refereed thesis.

Path Planning:

The plannerControlRRT object in MATLAB is a rapidly exploring random tree (RRT) planner for solving kinematic and dynamic (kinodynamic) planning problems using controls.[4] In kinematic planners, the tree grows by randomly sampling states in system configuration space, and then attempts to propagate the nearest node toward that state. The state propagator samples controls for reaching the state based on the kinematic model and control policies. As the tree adds nodes, the sampled states span the search space and eventually connect the start and goal states.

Algorithm:

To begin with, two RRTs are defined T_{init} and T_{goal} each initialized to contain to contain a single node representing x_{init} and x_{goal} .We pick a random state and x_{state} and generate new node and both trees via function GEN_STATE. From the nearest of x_{rand}, all possible controls are applied to the system for interval t.

GROW_RANDOM_TREES()

InsertState($\mathcal{T}_{init}$, nil, x_{init});
InsertState($\mathcal{T}_{goal}$, nil, x_{goal});
while $continuePlan$ **do**
    $x_{rand} \leftarrow$ RandomState();
    $x_i \leftarrow$ GEN_STATE($\mathcal{T}_{init}$, x_{rand}, FWD);
    if $x_i \neq nil$ **and** NearbyState($\mathcal{T}_{goal}$, x_i) **then**
        record candidate solution τ connecting $\mathcal{T}_{init}$ and $\mathcal{T}_{goal}$ through x_i;
    $x_g \leftarrow$ GEN_STATE($\mathcal{T}_{goal}$, x_{rand}, $BACK$);
    if $x_g \neq nil$ **and** NearbyState($\mathcal{T}_{init}$, x_g) **then**
        record candidate solution τ connecting $\mathcal{T}_{init}$ and $\mathcal{T}_{goal}$ through x_g;

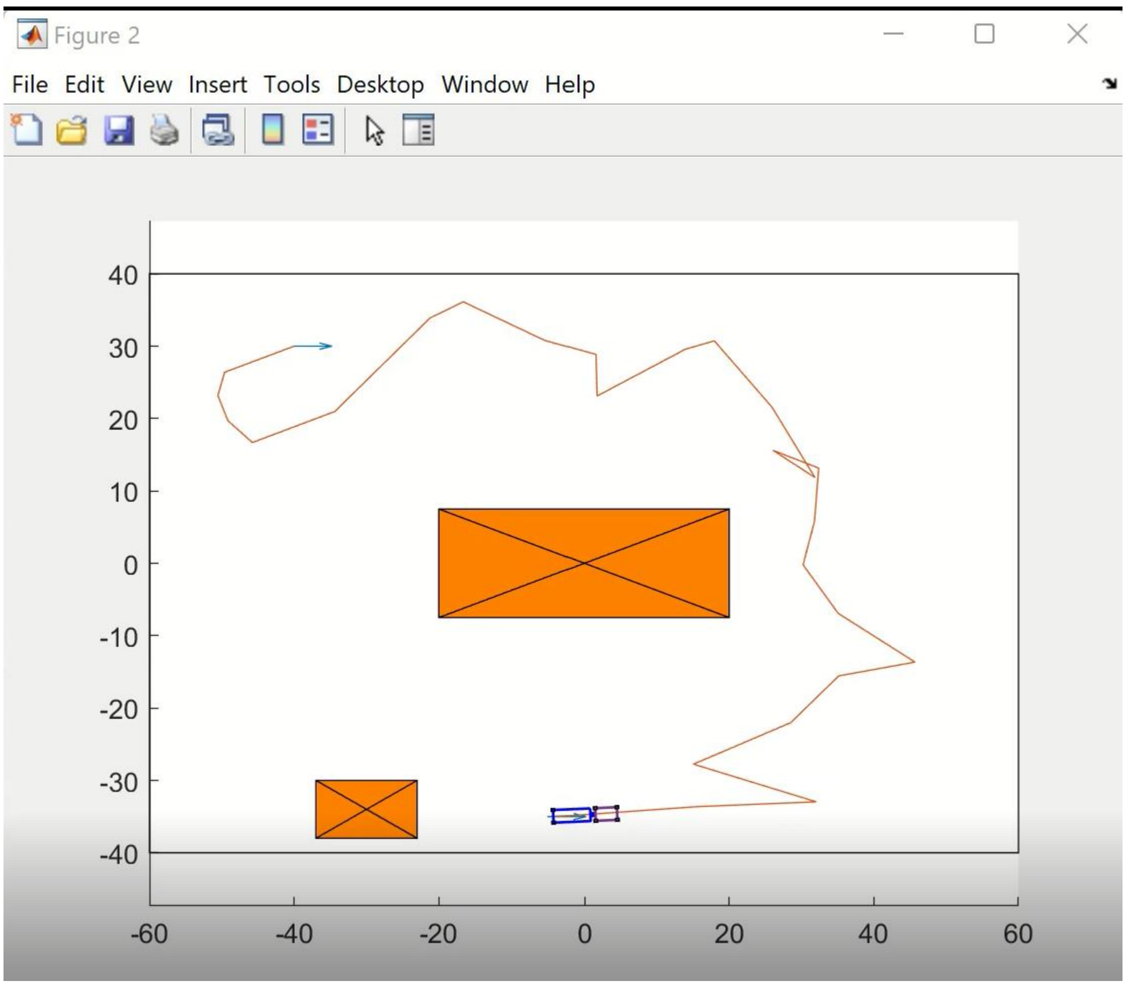


Figure 3 Planned path

Simulation:

Simulation is done for the truck with Axle width is 1.75 m, wheelbase is 3.0 m, and the distance between the rear axle of the truck and the axle center of the trailer is 5.0 m.

See the attached video with the file for the simulation.

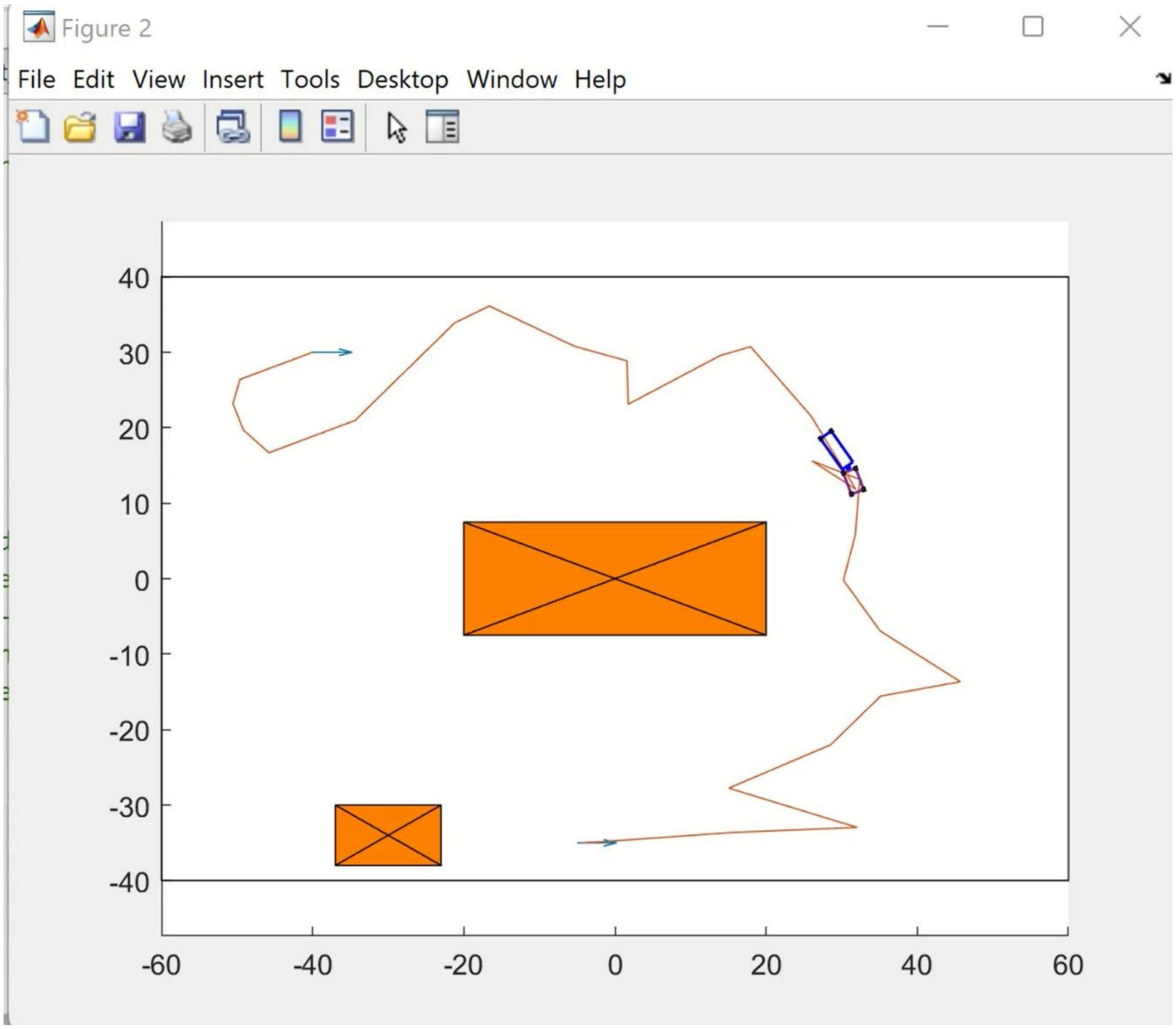


Figure 4 Snapshot of truck simulation

References:

[1] Petereit, Janko, Thomas Emter, Christian W. Frey, Thomas Kopfstedt, and Andreas Beutel. "Application of Hybrid A* to an Autonomous Mobile Robot for Path Planning in Unstructured Outdoor Environments." *ROBOTIK 2012: 7th German Conference on Robotics*. 2012, pp. 1-6.

[2] Lynch, Kevin M., and Frank C. Park. Modern Robotics: Mechanics, Planning, and Control. Cambridge University Press, 2017.

[3] Holmer, Olov. "Motion Planning for a Reversing Full-Scale Truck and Trailer System". M.S. thesis, Linköping University, 2016.

[4] S.M. Lavalle, J.J. Kuffner, "Randomized kinodynamic planning", *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378-400, May 2001.