Jordan Grotz

RBE 550

Valet Assignment

3/20/2023

Methods

Matlab was used for the implementation of this assignment due to the convenience of integrating the velocity kinematics and for Matlab's predefined occupancy grid objects. As Matlab doesn't have the same data types as other coding languages, some novel implementations had to be used. The following subsections depict how everything was implemented.

Map Creation

The maps were initially created in a graphical editing software (inkscape), allowing complete flexibility in easy placement of the obstacles. These maps were saved as image files (Figure 1), which can then be loaded into MATLAB as an OccupancyGrid. MATLAB's built in VehicleCostMap, allows vehicle dimensions to be added, and it will adjust the inflation of the obstacles accordingly. This is seen in Figure 2. The defined vehicle dimensions are saved as a collisionChecker object, which should verify if a pose of the vehicle is valid or not. Through testing, the results seemed inconsistent. The benefits of the VehicleCostMap is that it collision can be checked on a discrete grid base, or in the actual position of the robot in meters. With a proper map depiction, the search algorithm was then created.

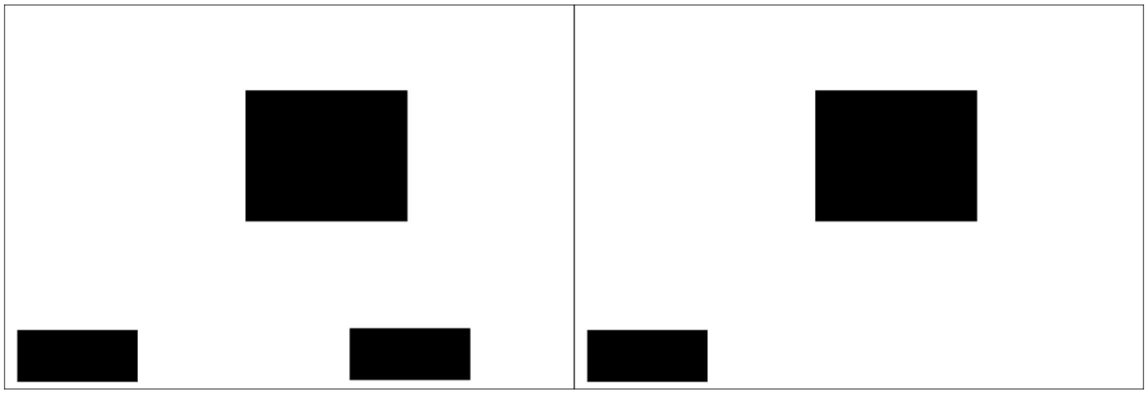

Figure 1: Pictures for different map types

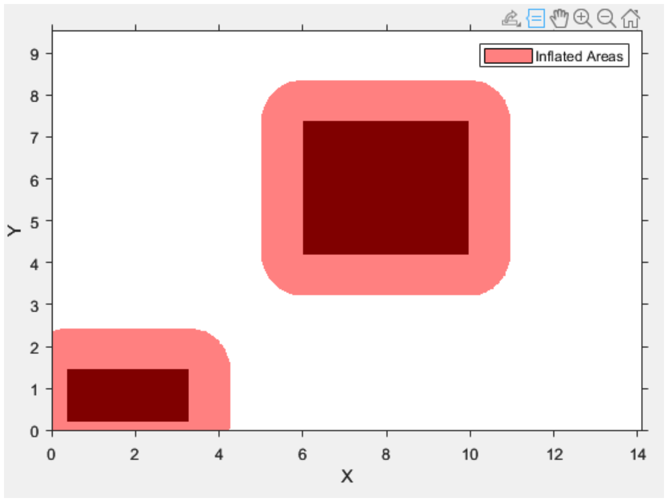

Figure 2: Matlab Representation of Map with inflation

Path Planning Algorithm

The algorithm chosen for the assignment based on the Hybrid A* described by Dolgov et al (2008). The A* pseudocode is described below, however, the data types in Matlab are also noted. The key points are the g, h, and f values associated with each point. The g value is set as the cost from the starting point to get to the current point. The h value is the heuristic which is set as the distance from the current point to the goal point. The F value is the sum of the g and h values. Because of the kinematic modelling, the cost of movement is set as the sum of the linear and angular movement. In addition, there is a penalty added for the car doing a reverse movement. For the differential drive only, there is also a penalty added for doing a turn only movement. The unique part of this Hybrid A* implementation is how the children are generated. This is described in the next section.

```
Initialize struct for storing h, g, previous point, control inputs, and
previous angular input.

Initialize dictionary storing struct using the coordinates as the key

Initialize list for open nodes (chosen as a table which can easily be sorted in
Matlab)

Initialize list for closed nodes


Add starting point to open list with a value of 0.

While the open list is not empty:

    Sort open list, and remove the node with the least value (lv_node)

    Place lv_node on the closed list

If lv_node is the goal position or if iteration count >= max iterations:

    Return path from lv_node to the starting point

Else:

    Find the children of the lv_node
```

```
        For each child in the list of children:
        If the child is in the closed list:
                Continue to the next child
        else
current cost = g value of the lv_node + cost of the movement to the child
if current cost <= child's current g value:
        update child's struct with new values for g, h, previous point….
        New h value = Euclidean distance from the child to the goal
If open list does not contain child:
        Add child to open list
Else:
        Update child's f value in the open list
```

Determining Child Nodes with Kinematic Modelling

The flow chart in Figure 3 shows the process for determining the children for each node. The children nodes can be changed by altering the input commands the algorithm searches over. In addition, the input velocity is scaled by how close the robot is to the goal location, to allow for more minute changes as it tries to reach the goal state and heading. The kinematics are integrated using the ode45 command, which is a 4th or 5th order approximation, balancing speed and accuracy. The out of bounds and collision check are done on the entire integrated path, not just the end points. The kinematic integration and collision check are semi-unique to each vehicle type. The code each kinematic model is saved as a different class which contains the kinematic equations, and collision checker. The collision checker for the trailer needs to track the trailer position and vehicle position. This is why the extra branch exists in the flow chart.

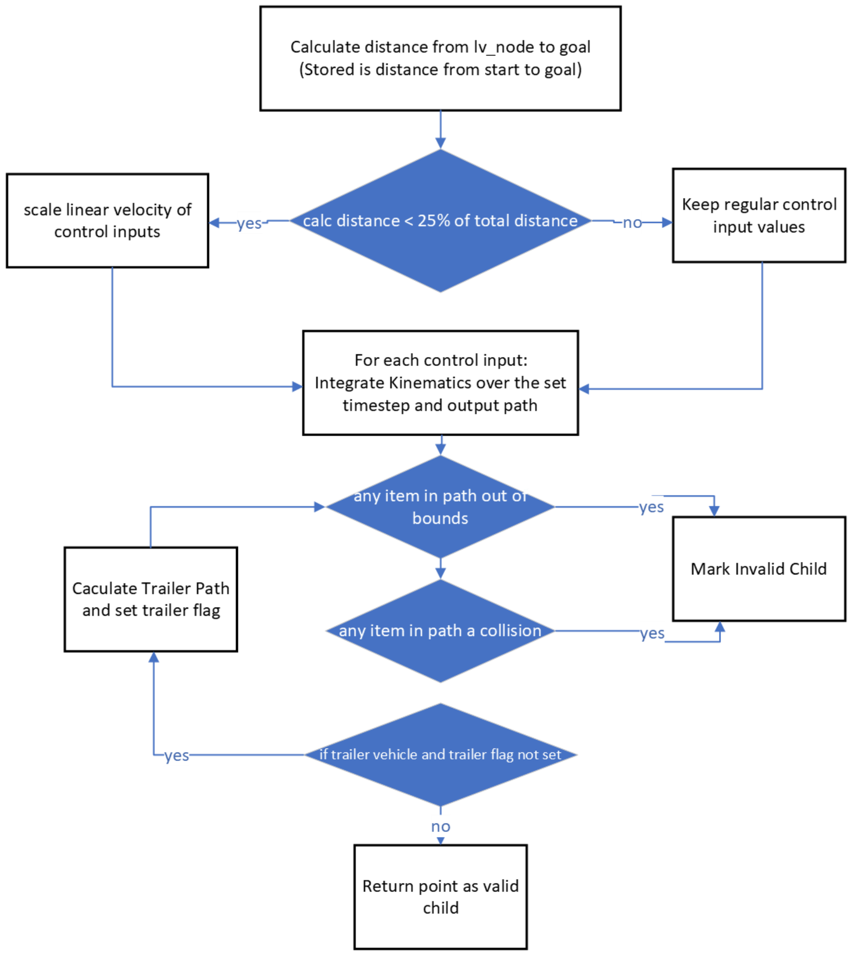
Calculate distance from lv_node to goal (Stored is distance from start to goal)
calc distance < 25% of total distance
yes
scale linear velocity of control inputs
no
Keep regular control input values
For each control input: Integrate Kinematics over the set timestep and output path
any item in path out of bounds
yes
Mark Invalid Child
any item in path a collision
yes
Caculate Trailer Path and set trailer flag
if trailer vehicle and trailer flag not set
yes
no
Return point as valid child

Figure 3: Flow chart for Child Generation

Path Depiction

The path planning function returns both the points on the path and the control inputs to get to those points. This allows the fully integrated path to be depicted. For this exercise, three output visualizations are generated. The first is the plotting the individual points with the heading of vehicle shown as a unit arrow in the direction of the heading. The second visualization is plotting the fully integrated path using the control inputs. Lastly, the robot (rectangular shape) is plotted at each point on the integrated path to represent the movement of the robot. This is saved as a video format.

Results

Figure 4,Figure 5, and Figure 6 showcase the resulting paths from the Hybrid A* search. Each picture includes what the integrated path looks like using the control inputs (Left), Each point and associated heading (Middle), and the final position of the robot (Right). The final position is graph is animated in the script, but that can't be shown in a paper report. The videos have been submitted in a separate folder. The slight discontinuities in the integrated path are due to the rounding of the actual position of the robot to limit the number of nodes that can be searched. For these graphs the robots position was rounded to a 10th of a meter, and the angle (in radians) was rounded to two decimal places. The trailer path does not show the full path, which is explained in the discussion. Lastly, Figure 7 shows an example of the searched nodes for the Ackerman steering kinematics.

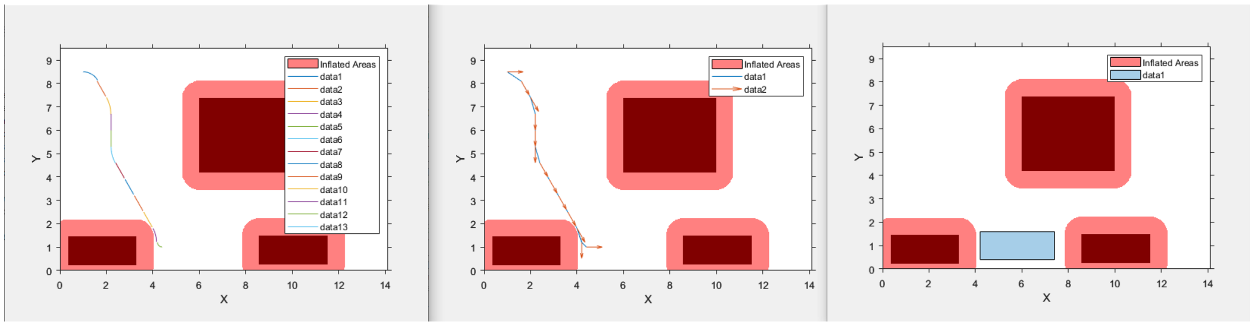

Figure 4 Differential Steering Path (Left Integrated path; Middle Each point with heading; Right: robot in final position)

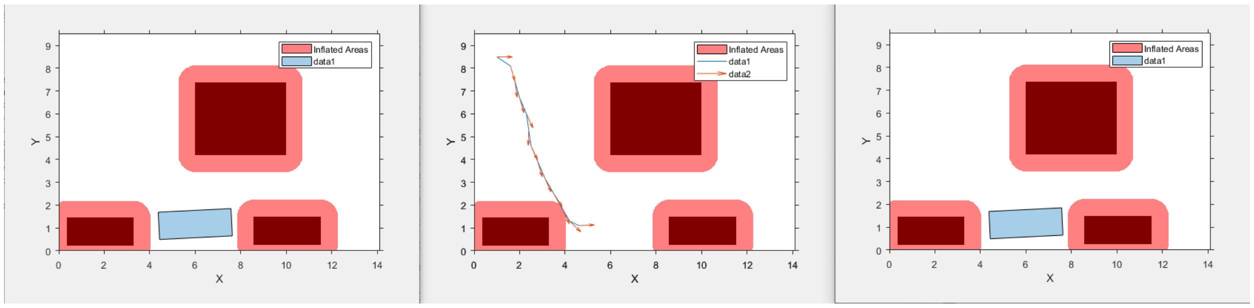

Figure 5: Ackerman Steering Path (Left Integrated path; Middle Each point with heading; Right: robot in final position)

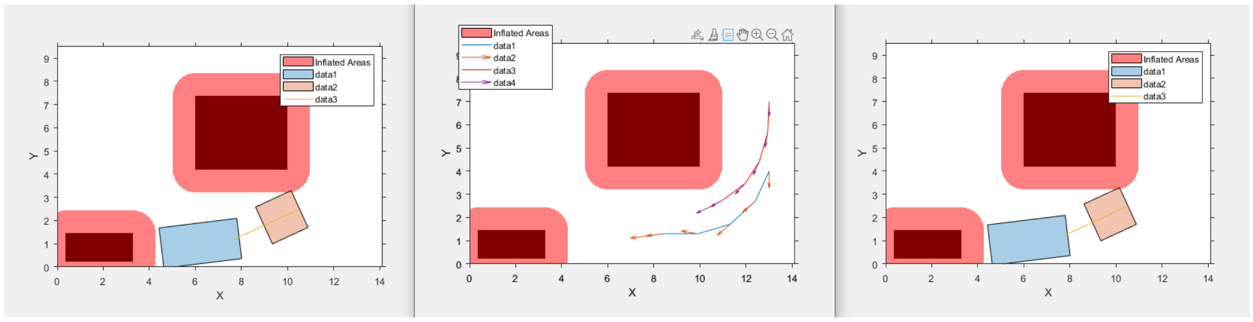

Figure 6: Trailer Steering Path (Left Integrated path; Middle Each point with heading; Right: robot in final position)

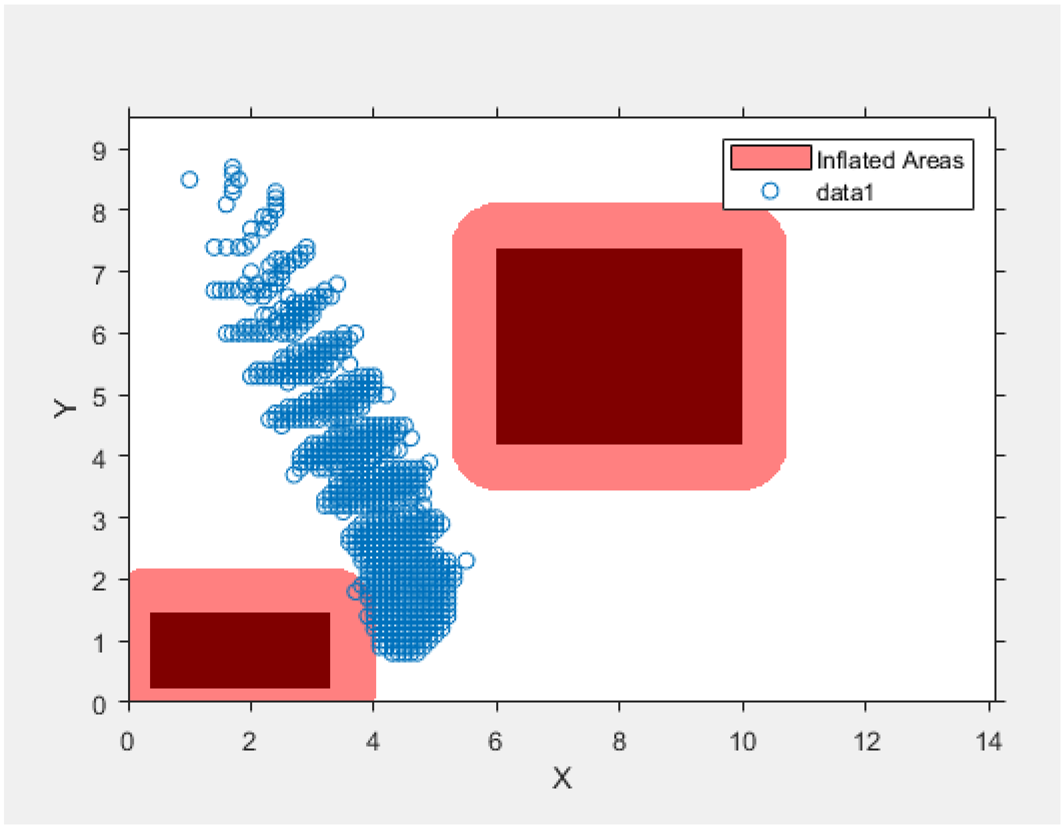

Figure 7: Example Searched Nodes for Ackerman Planning

Discussion

Trailer Kinematics issues

The trailer doesn't show a full path due to the inability of the code to find a path. The path shown was created by moving the robot closer to the goal, and changing the acceptable heading tolerance. This is representative of that 'one guy' who refuses to park is trailer properly. The trailer could not properly start at the top of the map due to the size of the map. MATLAB's occupancy grid uses a buffer on the perimeter of the map, which the trailer was always inside. In retrospect, a large workspace should have been created, but changing the pngs used for map creation changed the scale of the map cause more issues. Also due to the complexity of the trailers uncontrollable movement, it was harder for this implementation of the algorithm to adjust for the movement of the trailer.

Areas of improvement

One area of neglect which is seen in the videos is that the primitive shapes drive off the bottom edge of the map while parking. This would be the equivalent of driving up onto the sidewalk to park. The 'occupancyChecker' should have caught this or additional code would need to be developed. Due to time constraints, this was not feasibly attainable.

In addition, the algorithm is not effective at checking alternate pathways. This is due to the resolution of the map with the heading stored for each point. The algorithm can not effectively mark an entire area as searched. For example, and 1x1 meter area contains 31,400 possible nodes when accounting for angle resolution of 0.01, and position resolution of 0.1. An additional metric could be created incentivize searched a different branch after x amount of nodes in a sector have been searched.