RBE 550 – HW3

John Hall

Valet Report

Environment and Vehicles

Thank goodness the parking lot was filled with tetrominoes, because I was able to reuse my field creation class from the previous assignments. The only difference was the field creation had to be aware of the starting and goal locations. This was to not place obstacles there and to place two obstacles on either side of the parking spot. Before the path planner even attempts a path, I first check if there is a theoretical path by using a 2D A* algorithm to determine if the start or goal locations are boxed in.

To simulate the kinematics of the vehicles, I used pygame to keep track of time and then pass to a vehicle class the change in time, and the control inputs. The control inputs for the car and truck are the speed and steering angle, and the delivery robot is directly controlled with an angular velocity. The resulting path is a set of positions and orientations, and the vehicle will travel between them using constant direction and angular velocity instructions to simulate a real vehicle. A smarter motion planner would have implemented variable velocity control depending on if the vehicle needs to turn, but that is a future improvement to be made.

Planning Algorithm

To plan my vehicle's path from the start to the goal parking location I am using a Hybrid A* algorithm using a state lattice with Reeds-Shepp analytic expansion. The algorithm takes the parking lot and discretizes it into a state with the vehicle: (x, y, heading, *trailer heading*). Each node is expanded into six neighboring nodes left, forward, right, back left, back, and back right. Traveling to different nodes incur different costs especially when changing direction and steering.

Each node's priority is then rated based on the maximum of two heuristic values. The first is a constrained-without-obstacles path, in this case the length of a Reeds-Shepp path to the goal. This determines the minimum length of a path given the vehicles kinematics and is especially important as the vehicle approaches the goal but does not take into account any obstacles. The second heuristic is an unconstrained-obstacle-aware two-dimensional A* algorithm that does not take into account the vehicle's kinematics. This heuristic determines the best bath avoiding obstacles. I dilate the obstacles and apply a gaussian blur to create a gradient for the heuristic to avoid obstacles because it is planned from the center of the vehicle's back axle. **A video of the state lattice expansion is attached in this submission demonstrating the nodes generated based on the current heuristics.**

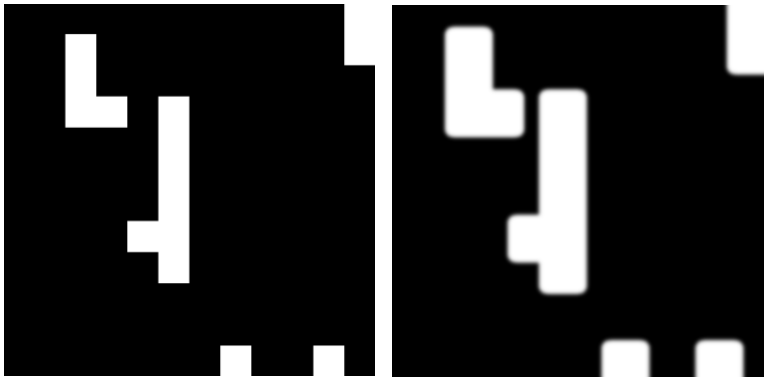

Figure 1. Dilated and blurred obstacles to assist 2D unconstrained A*

To extend the capabilities of the state lattice expansion, as the algorithm approaches the goal, nodes are probabilistically selected to generate a Reeds-Shepp path. This is known as analytic expansion and is proven to find the shortest path from a node, but the path must first be checked for collisions (explored more in the subsequent section). The probability that a node is analytically expanded is based on the ratio of the distance-to-go and the initial distance. This means that nodes closer to the goal will be probed more often than at the beginning. The Reeds-Shepp path also allows for the solution to converge exactly to the goal solution, not just to the discretized bin, and speeds up the algorithm significantly. **Analytic expansion is not available for the truck because Reeds-Shepp does not take into account the kinematics of the trailer. This means solutions are only generated through the state lattice and tend to take much longer to find.**

```
# Path Planner Pseudocode
def PathPlanner(Start, Goal, Map, Resolution, Ang_Resolution):
    frontier = PriorityQueue()
    came_from = {}
    cost_so_far = {}
    came_from[discretize(Start)] = None
    cost_so_far[discretize(Start)] = 0
    frontier.put((0,Start))

    while not frontier.empty():
        node = frontier.get()

        if collision_check(node): # check if current node is in collision
            continue

        if discretize(node) == discretize(Goal): # check if current node is in goal region
            return path(node)

        if probability(node): # calculate probability rs_path is generated from node based on distance-to-go/initial-distance
            rs_path = reeds_shepp(node, Goal)
            if not collision_check(rs_path): # check if Reeds Shepp path is viable
                return path(node) + rs_path

        for next_node in find_neighbors(node): # generate state lattice expansion with L, S, R, BL, B, BR directions
            if valid(node):
                new_cost = cost_so_far[discretize(node)] + next_node.additional_costs + Resolution # add additional costs of turning/backing up
                if next_node not in cost_so_far or new_cost < cost_so_far[discretize(next_node)]:
                    cost_so_far[discretize(next_node)] = new_cost
                    priority = new_cost + heuristic(next_node)
                    frontier.put(priority, next_node)
                    came_from[discretize(next_node)] = node
```

Figure 2. Pseudo code describing the flow of my Hybrid A* path planner.

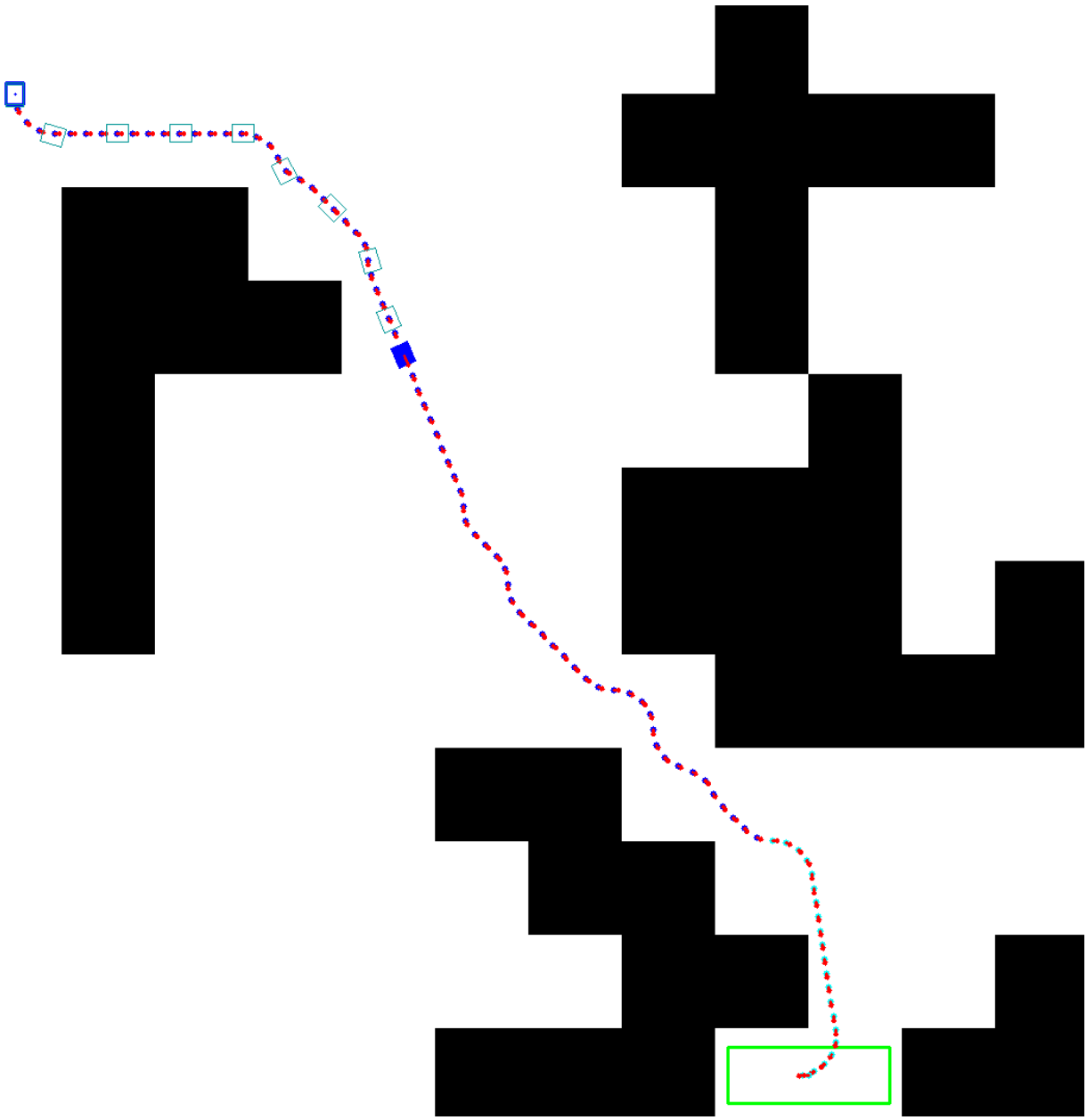

Figure 3. Example path generated for delivery robot with a map density of 30%. The last section of the path is colored slightly different and represents the Reeds-Shepp analytic expansion.

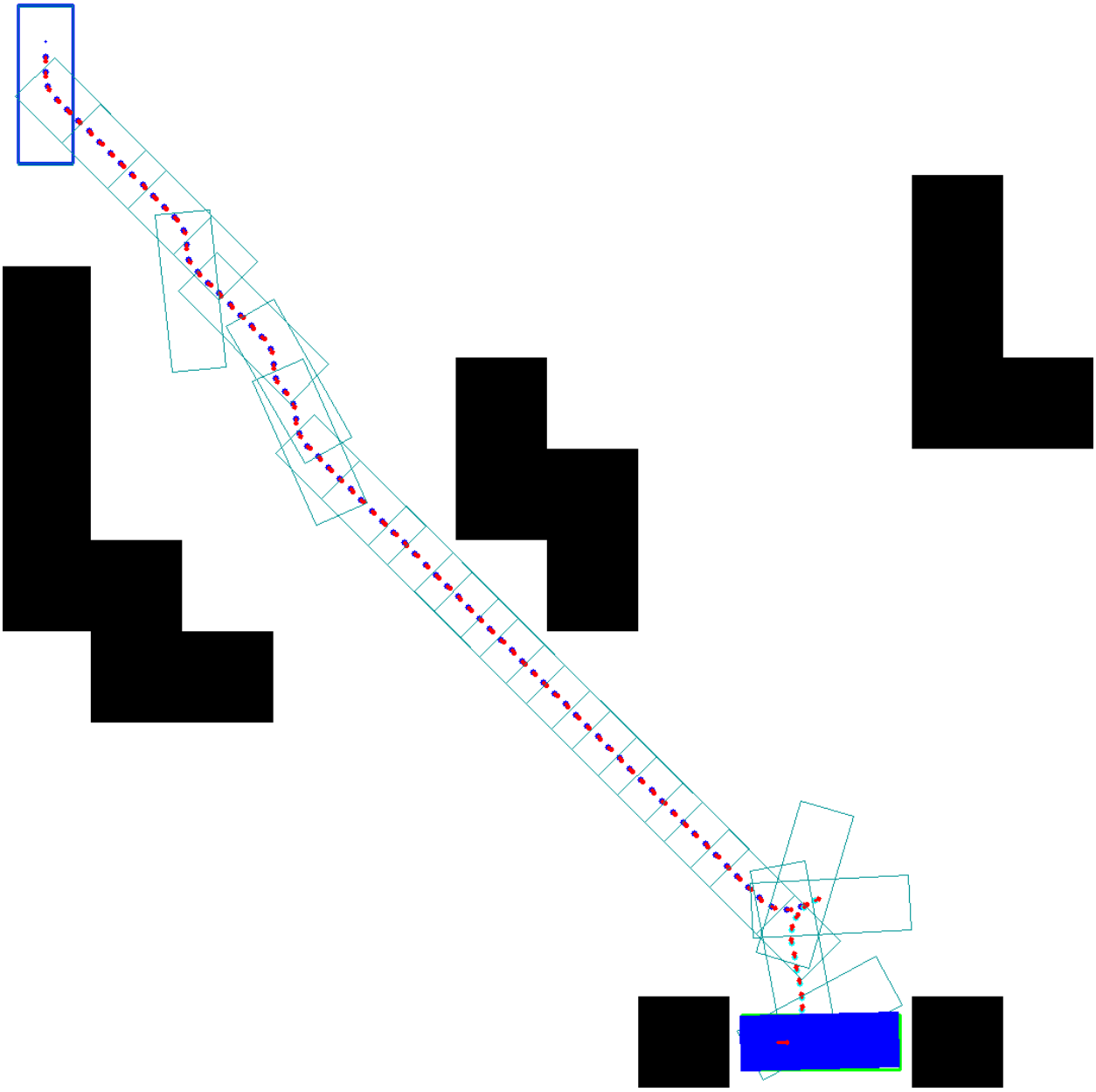

Figure 4. Path followed by a car to avoid obstacles and correctly parallel park in a spot.

Collision Checker

To check for collisions in any path, I used computer vision practices to draw oriented bounding rectangles and create masks between the path of the vehicle and any obstacles. This method checks if there are any filled pixels in the masked image to determine if there is a collision. If so, it will alter the cost of the remaining nodes after the collision in case there is a safe way to reach the subsequent nodes.

Collision checking is also employed to detect self-collisions between the truck and the trailer to prevent jackknife intersections. Using computer vision techniques was very useful for debugging

and is a concept I am very comfortable with. However, I don't believe that it is as cost efficient as using a data structure like a kd-tree.

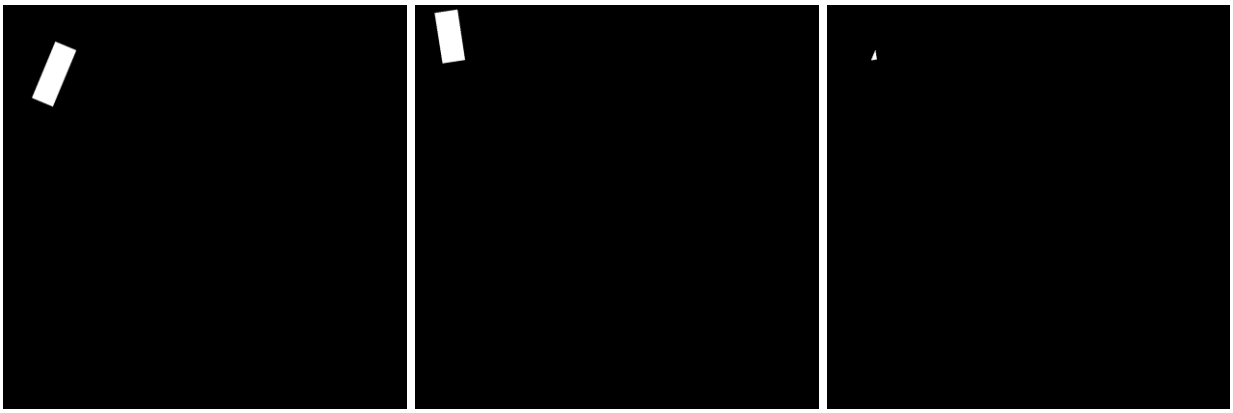

Figure 5. Self collision of the truck and trailer using masking. Truck box (left), truck trailer (center), intersection (right).

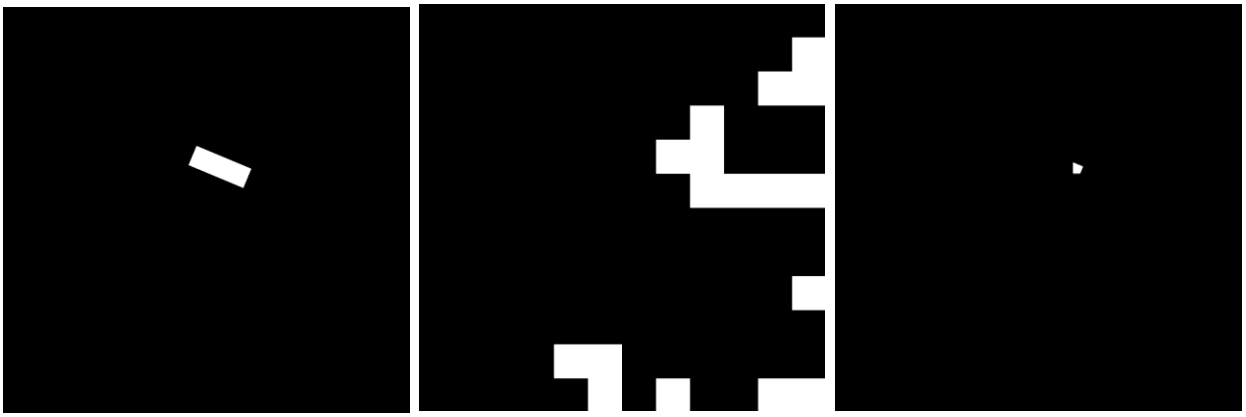

Figure 6. Collision between car and obstacle using the path thus far (left), obstacles (center), and any intersection between them (right).

Challenges

I think there were two big challenges that made this assignment difficult. The first was understanding the additional costs incurred by turning/reversing because it would result in weird or circular paths. It wasn't necessarily a problem, but many of the paths generated would be slightly jerky with sudden turns, I could have benefited from adding a path smoother to generate better paths.

I would end up getting circular paths where a node somehow found a cheaper way to arrive at its parent node. These circular paths were of varying lengths, but it would cause infinite loops while trying to check for collisions and end solutions. Currently, the way I handle it is whenever I have to loop through the entire path, if the length of the path exceeds the total number of explored nodes, break from the loop and go to the next node in the frontier. Surprisingly it worked well to

ignore, but I'm worried there is something deeper going on in my algorithm that I was unable to diagnose in the timespan of the assignment.

The other big challenge was the truck. I didn't have too much of an issue implementing its kinematics in the simulator or in node expansion, but the biggest problem was the fact that Reeds Shepp is unavailable. I did not have time (or my simulator did not have enough time to search every node) to get the truck to parallel park into a spot. Analytic expansion sped up my algorithm drastically for the robot and car, and I think with more time, I would have liked to come up with a way to analytically expand the truck-trailer solution.

Resources

https://ai.stanford.edu/~ddolgov/papers/dolgov_gpp_stair08.pdf

https://medium.com/@junbs95/gentle-introduction-to-hybrid-a-star-9ce93c0d7869

https://kth.diva-portal.org/smash/get/diva2:1057261/FULLTEXT01.pdf

https://github.com/yinflight/HybridAStarTrailer (For Jackknife cost)

https://pypi.org/project/rsplan/ (For Reeds-Shepp path generation)

The base code for this assignment was heavily borrowed from the code my partner Sean Lendrum and I were working on for our final project which uses Hybrid A*. Certain functions related to Hybrid A* may be similar, but we worked on this assignment separately.