ME 597
Final Project Report

TJ Wiegman
2023-04-28

Data Structures: MapCells and a MapQueue

- Each graph node is a MapCell, which each have three properties:
  - Location
  - Previous node (where it came from)
  - Cost to get there
- The graph as a whole is stored as a MapQueue, which stores each cell as a row in a table. Each row has six properties:
  - Location
  - Previous node
  - Cost to get there
  - Distance to goal
  - Priority rank (equal to cost + distance)
  - A tombstone, to note if it has been "removed" from the queue
- Getting the next item in the queue is as simple as sorting by priority rank and taking the top non-tombstoned row

A* Algorithm

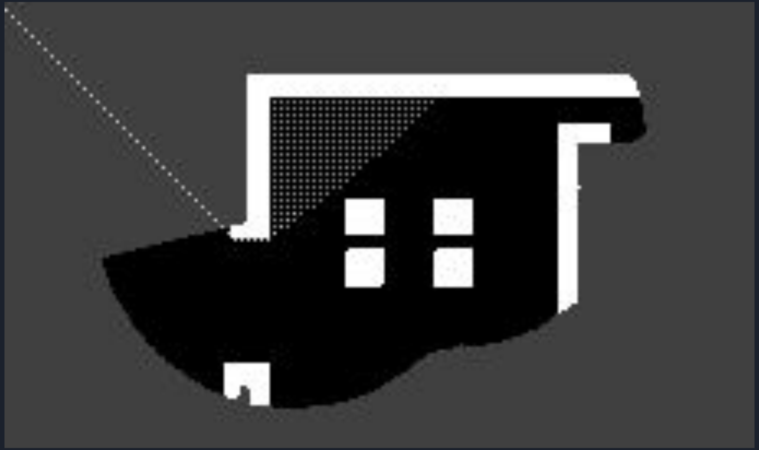
1. Look at all the neighboring cells of the next node in the queue
2. If a cell is new, add it to the queue
3. If a cell is already in the queue and this route costs less, update its record
4. Remove the central node from queue
5. Repeat until queue is empty or goal is reached!

```
while not (goal or self.update):
    # Search neighbors of current node
    nn = self.neighbors(node.coord)
    for coord in nn:
        weight = node.weight + 1
        newcell = MapCell(coord, node.coord, weight)
        queue.add(newcell)
    queue.remove(node.coord)

    # Pick next node to explore
    node = queue.next()
    if node == None:
        rospy.logerr("*** Path not found!")
        output = (None, queue.explored())
        return output
    elif node.coord == end:
        rospy.logwarn("Path created successfully!")
        goal = True
```

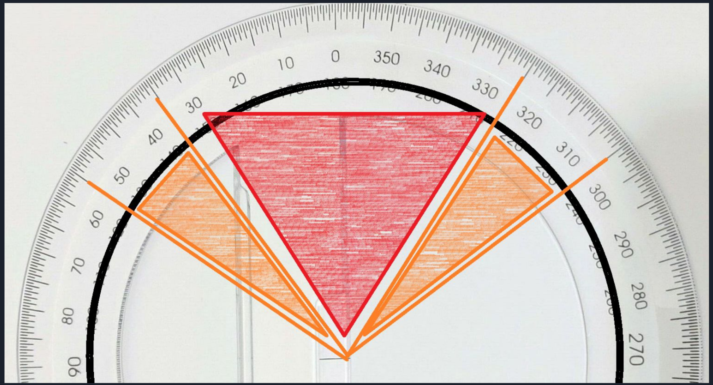
Task 1 - Strategy

1. Unexplored space is marked as traversable in the map
2. Plan a route to the top corner of the map (0,0) and follow it
3. When the map updates, plan a new route, avoiding newly-detected obstacles
4. When we eventually get stuck, try routing to the bottom corner instead (max,max)



Task 2 & 3 - Strategy

1. Use A* algorithm to plan a route
2. Use LIDAR to detect obstacles
   - Walls, if drift too close
   - Dynamic obstacles
3. Veer around obstacle, or back up
4. Either wait for obstacle to move, or re-route around it




```python
# Prevent running into obstacles
def wallstop(self, data: LaserScan) -> None:
    fRanges = circLPF(data.ranges)
    dir = np.argmin(fRanges)
    if fRanges[dir] < 0.36:
        if (dir > 35) and (dir < 55):
            self.stop = True
            rospy.logwarn("Veering around obstacle (left)!")
            self.drive(self.slow, -self.slow)
        elif (dir > 305) and (dir < 325):
            self.stop = True
            rospy.logwarn("Veering around obstacle (right)!")
            self.drive(self.slow, self.slow)
        elif (dir < 35) or (dir > 325):
            self.stop = True
            rospy.logwarn("Backing away from obstacle...")
            self.drive(-self.slow)
    else: self.stop = False
```

Tasks Performance

1. Task 1 was… pretty bad. My A* algorithm is not very fast, since it quickly generates a large numpy array and then has to sort it every time we retrieve next in queue. As a result, once more than the first half of the map is explored, the A* algorithm takes way too long to find a new path and we end up stuck.
2. Task 2 is (after squashing a few bugs in gradescope submission) actually fairly good. When the map is already constrained and explored, my A* implementation is good enough. The simple LIDAR obstacle avoidance takes care of any drifting towards walls.
3. Task 3 (a carbon copy of task 2) seems to work well. The pathfinding is just as adequate as it was on task 3, and the LIDAR algorithm I added for wall avoidance responds quickly enough to prevent colliding with dynamic obstacles as well.