Gage

June 7, 2024

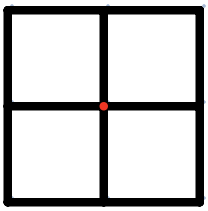
AACS lab

This is an outline for creating a 2D map while traveling, sort of like a simplified SLAM algorithm.

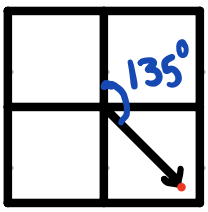
We can record where we have traveled (and possibly other info) on a coordinate grid and use that information to explore while also building a map.

Traveling

initialize at (0,0)



On forward motion, compute distance and direction traveled



$$x' = x + (d \cdot \cos(135° \cdot \frac{\pi}{180}))$$
$$y' = y + (d \cdot \sin(135° \cdot \frac{\pi}{180}))$$

Old box = (int(x), int(y))

Current box = (int(x'), int(y'))

If (x',y') is in another box than (x,y), add 1 to the current box's entry in the dict/map.

dict[Current box] += 1

Mapping

Keep track of the min and max xy coordinates to retain map size.

Update the map size when traveling outside of the map. (Current x > max x
Current x < min x
current y > max y
current y < min y)

The map will be a data structure like this:

```
Class map:
    Coordinate dictionary = { }
    max_x = 0
    max_y = 0
    min_x = 0
    min_y = 0
```

It is essentially a dynamically sized sparse matrix data structure

We could retain other information within the map's dictionary, like actions taken and anomalies.

Scanning

Using current box as the center, check some N x M region around the current box.

While checking, rotate the region based on the heading. This will allow our model to understand what direction the robot is facing.

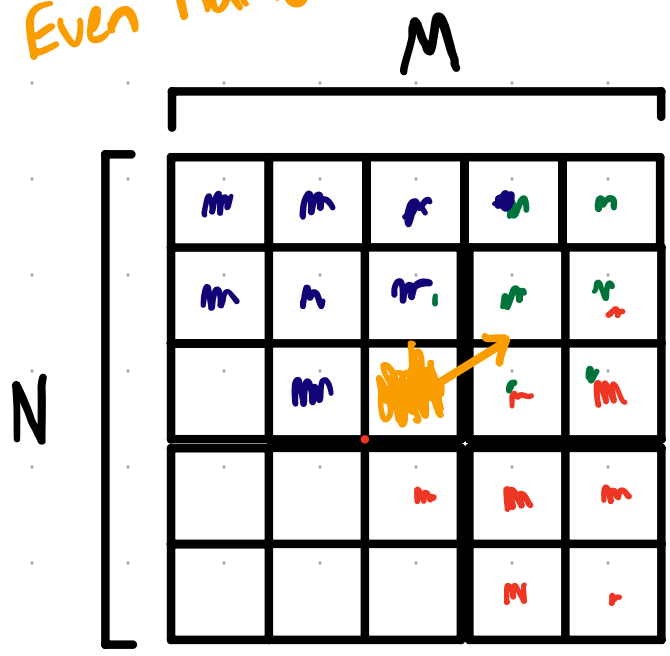
```
Matrix = np.zeros(n,m)
    for x in range(curr_x - N/2, curr_x + N/2):
        for y in range(curr_y - M/2, curr_y + M/2):
            Check dict listing for box value at (x,y)
            if no dict entry, pass

            Find the coordinate in the rotated matrix.
            a,b = rotated (x, y, heading angle)

            add value to matrix[a][b]
```

Now we should have an N x M matrix that is rotated based on heading

* N + M Should be Even numbers



Rotate

Regions of Influence

= forward
= right
= left

= Current location

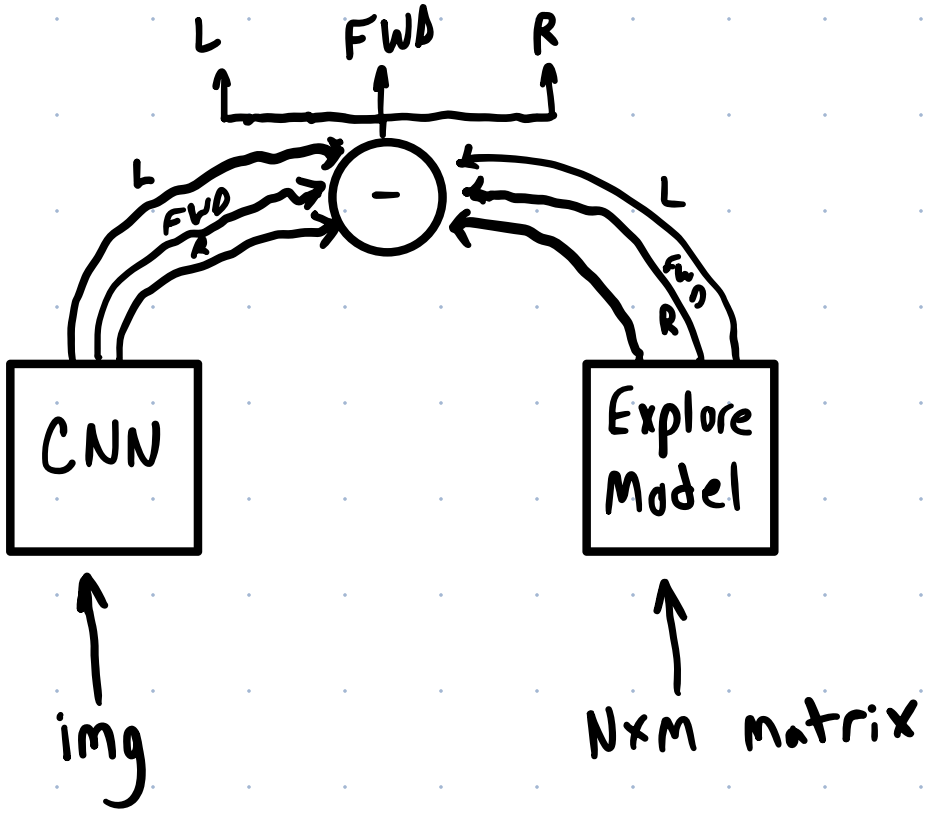
Regions of Influence

= forward
= right
= left

= Current location

Integrating

We should subtract whatever the exploration output is from the predicted actions that the navigation model outputs. Something like this:



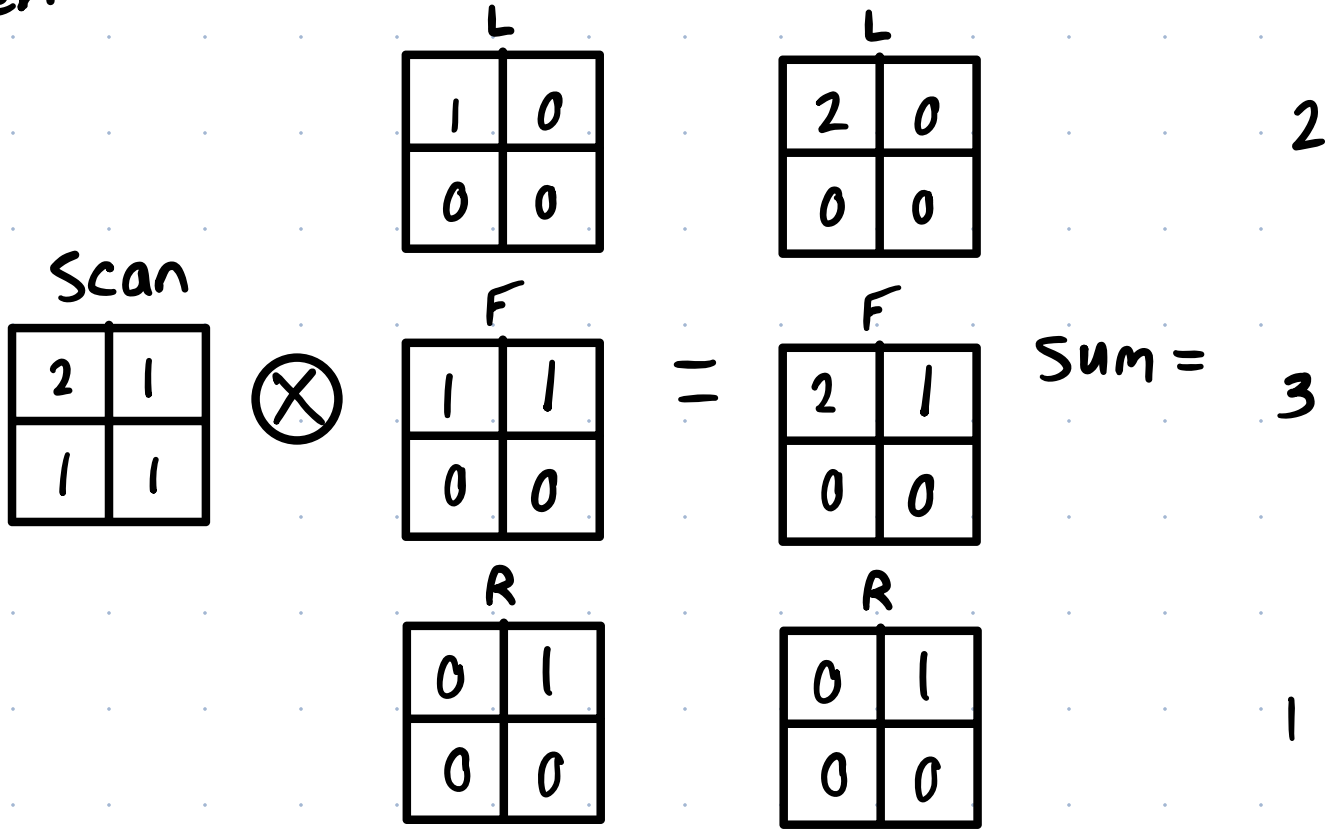
We only need the exploration model when there is a choice to be made between paths. We can compare the confidence between each

Ex:

```
If    abs(left-right) < .1 || abs(left-fwd) < .1
      || abs(right-fwd) < .1 :
      scan = Gridmap.interpolated_scan(position,heading)
      L R F = ExploreModel.forward(scan)
      left -= L    right -= R    fwd -= F
```

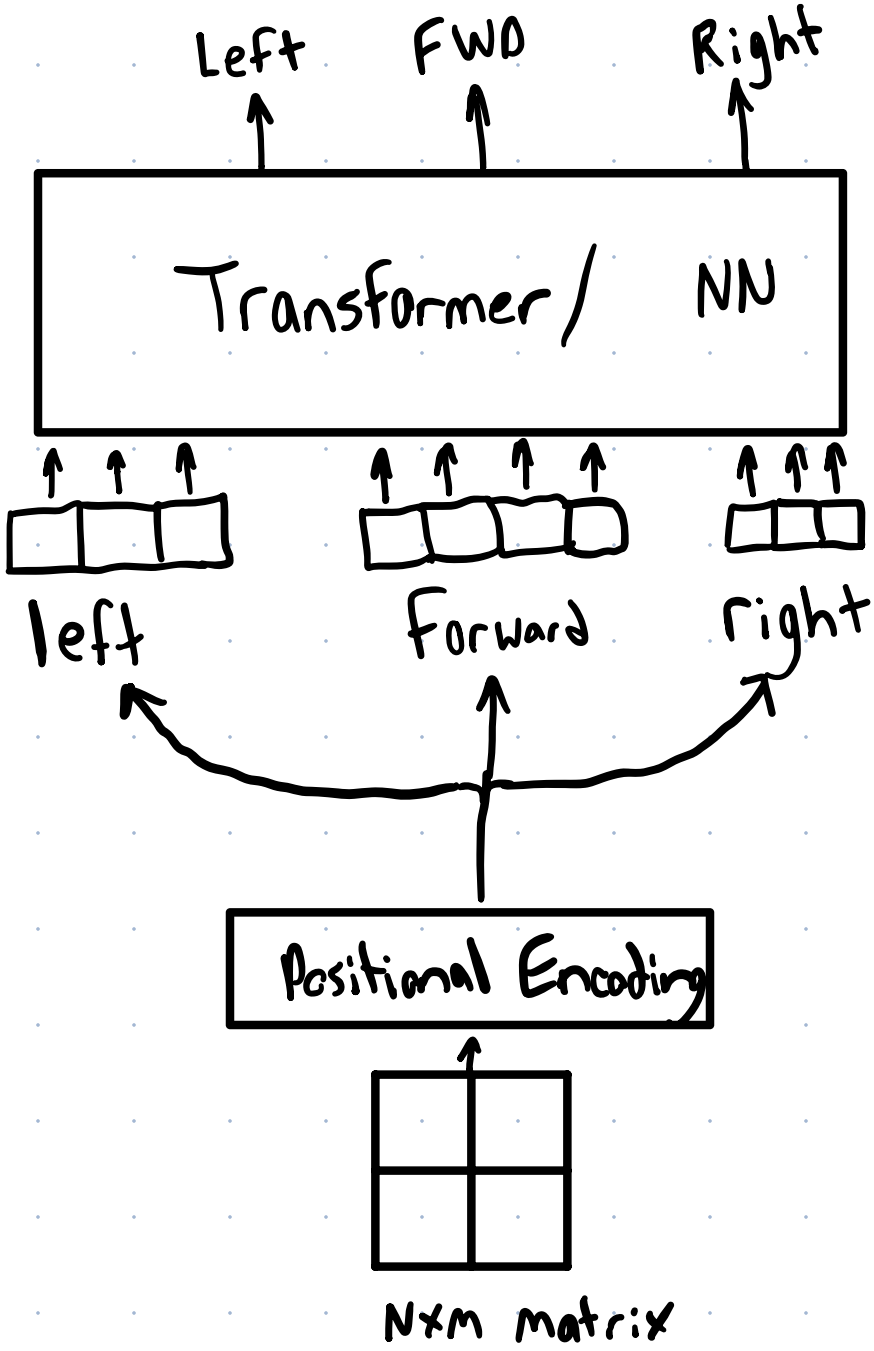
Explore Model

We can use a simple form of convolution to our scanned matrix to extract our left, right, and forward information. Gradient filters of the same size as the scan can be multiplied elementwise and then summed for each direction



This way we can create the gradient filters manually and have a working model without training it on any data

Alternatively, we can feed the scan into some NN and train it until it gives accurate predictions.



This method will require a massive exploration dataset ∴