Cleaning Robot with Bayes Filter and Kalman Filter

New Predicted State

$X^{P} = A*X^{t-1} + B*u_{t} + w_{t}$

The state transition matrix is applied to the previous state, $B*u_{t}$ applies acceleration which provides values to update the position and velocity of A*X, and then w t is applied to it, which is a

zero mean multivariate distribution using zero as its mean and standard deviation.

$P^{P} = A*P_{t-1}*A^{T} + Q_{t}$

The process covariance matrix is updated with the state transition matrix, and then the system noise Q_{t}, the covariance of the process noise is applied.

Kalman Gain

$K = P^{P}*H^{T} / (H*P^{P}*H^{T} + R)$

To calculate the Kalman Gain, K, we compare the error in the estimate to the combined errors in the estimate as well as the covariance of the observation noise(R). H is used to get the desired dimension for output of K.

$Y_{t} = C*Y_{t} + R_{t}$

Observational data about what had just occurred,Y_{t}, is found by using C as the identity matrix in

which we choose which variables we decide to look at on the most recent state matrix, and the measurement noise is applied, R_{t}.

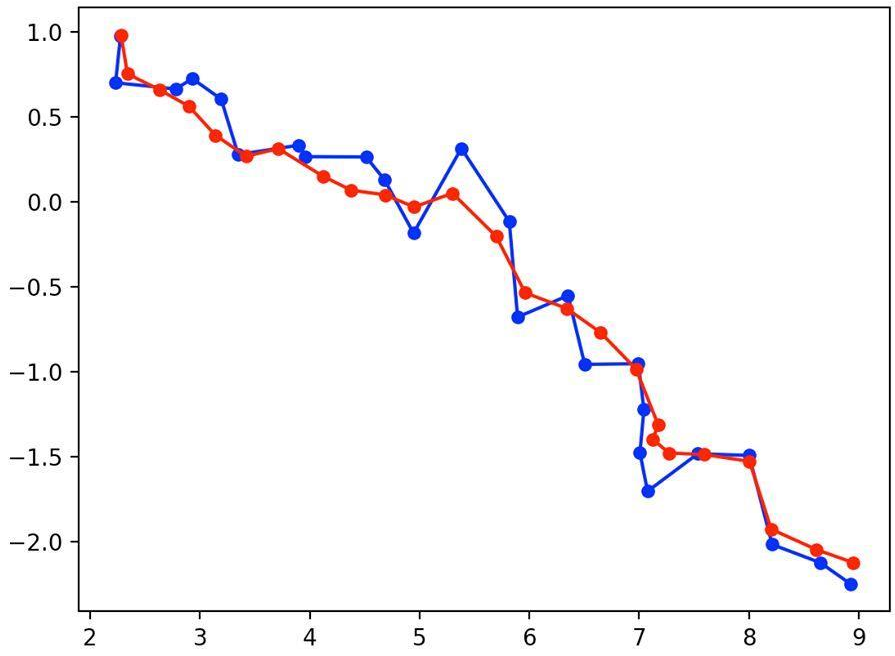
Updating

$P_{t} = (I - K*H)*P^{P}$

To get the new process covariance matrix we take the identity matrix minus the Kalman gain, K. and apply it to the previous covariance matrix. H is used to get the desired dimensions again.

$X_{t} = X^{P} + K[Y-HX^{P}]$

Update state matrix by adding the Kalman gain onto the measurement gain between now, Y, and the previous example



Robot Algorithm

This is simple cleaning robot algorithm written in Python. It assumes the following:

- Map can be interpreted as 2-d grid, with obstacles as obstructed points in the map.
- Obstacles can be arbitrary, both in quantity and position.
- The algorithm knows nothing about the surrounding environment.
- The robot only provides 3 rules: turn_left, turn_right (rotate the looking direction) and move (move ahead 1 point)

The goal of the algorithm is to make the robot travel around room, with any point in the room visited at least one.

How does it work:

The algorithm (sweeper.py) works as follow:

1. Initialize the observed map to empty, current position to (0, 0)
2. Find the nearest unvisited point in observed map using Breadth-First Search
3. If cannot find, algorithm stops, and the accessible part of the room has been cleaned. Otherwise go to step 4
4. Move the robot to the position found in step 2, with each step updating the current position.
5. If it cannot move to the desired position, mark the position as obstructed in observed map, otherwise mark it as visited.

6. Go back to step 2

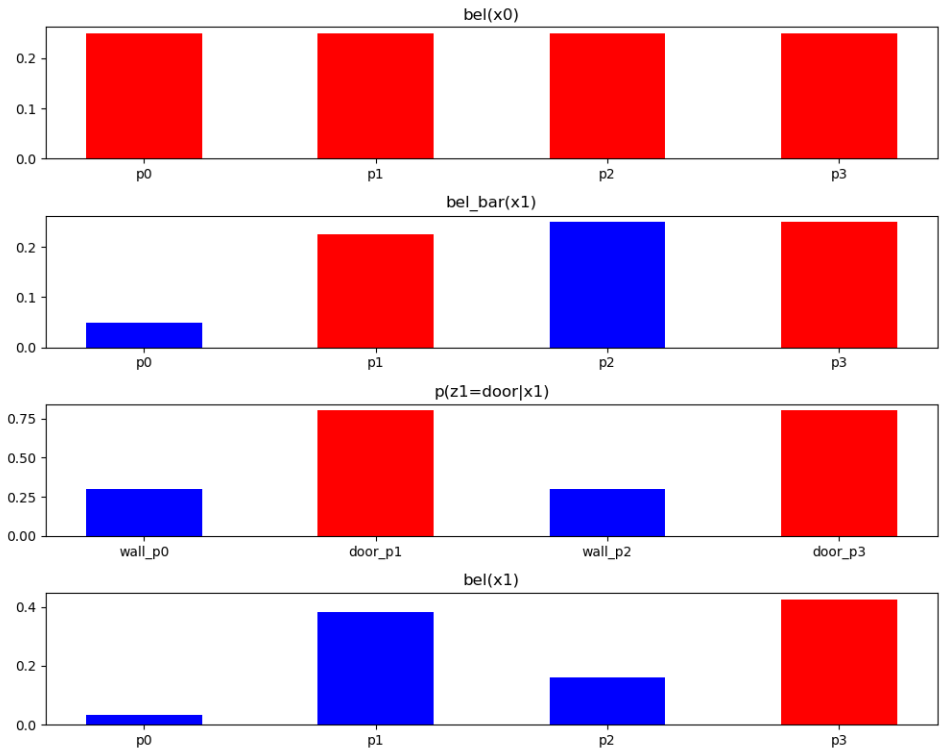
Complexity;

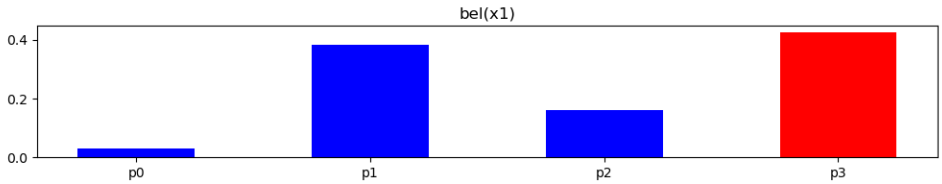
Theoretically, the time complexity of the algorithm is $O(N^2)$. It needs to find at most N unvisited points, and each need at most N steps to get there. But in practice, since it only finds the nearest unvisited position, so it can efficiently do it in O(N).

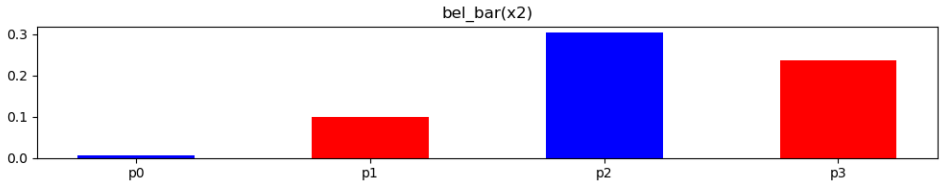
The space complexity is O(N), to store the observed map.

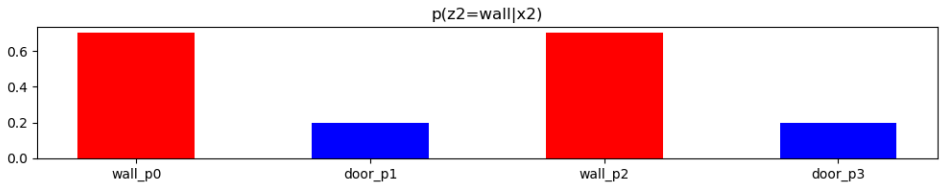
Optimization:

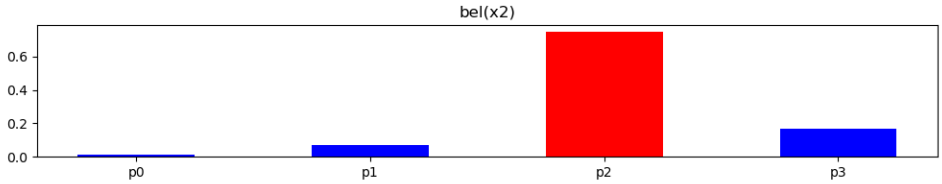
An optimized version of the algorithm, called Spiral BFS, can help to reduce the number of steps taken by 5-10%. When visualizing the algorithm, I found out that the robot occasionally misses the uncleaned part (since it favors moving in one absolute direction), and needs to waste time traveling back. By making the robot favoring turning left (or right), it will minimize the chance of missing uncleaned parts. Two modes can be switched easily by setting spiral attribute of the algorithm

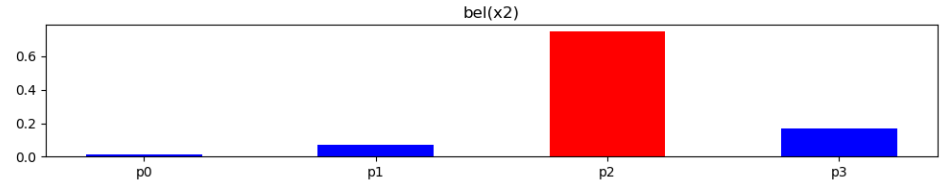


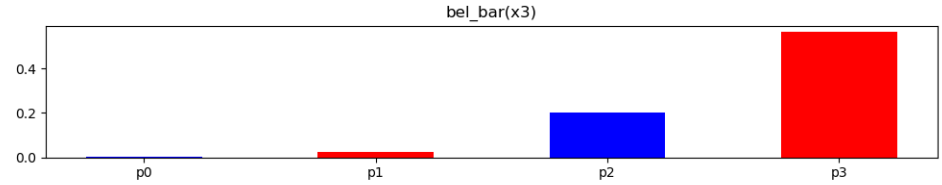

bel(x1)
0.4
0.2
0.0
p0
p1
p2
p3

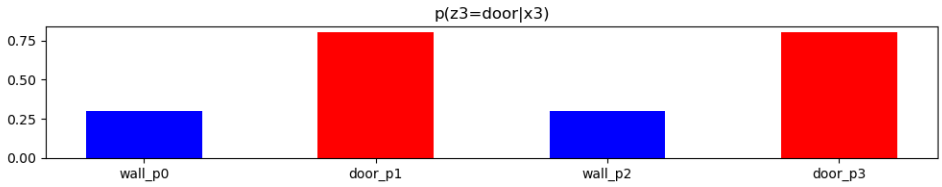

bel_bar(x2)
0.3
0.2
0.1
0.0
p0
p1
p2
p3

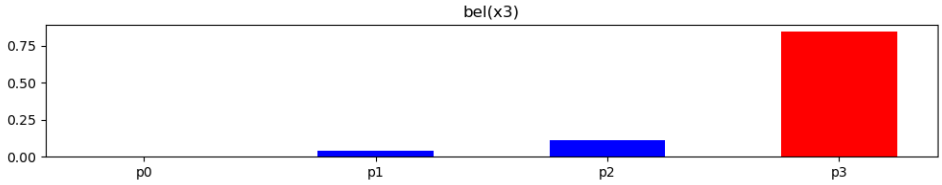

p(z2=wall|x2)
0.6
0.4
0.2
0.0
wall_p0
door_p1
wall_p2
door_p3


bel(x2)
0.6
0.4
0.2
0.0
p0
p1
p2
p3


bel(x2)
0.6
0.4
0.2
0.0
p0
p1
p2
p3


bel_bar(x3)
0.4
0.2
0.0
p0
p1
p2
p3


p(z3=door|x3)
0.75
0.50
0.25
0.00
wall_p0
door_p1
wall_p2
door_p3


bel(x3)
0.75
0.50
0.25
0.00
p0
p1
p2
p3

Robot exploring the environment using the DFS algorithm, the steps are noted and the time taken for the goal state also noted. The comparison of DFS and BFS showing the promising result of the robot the exploring the environment at the same time.

steps taken by dfs: 159, turns taken: 388, time taken: 18.45ms

steps taken by planned bfs: 0, turns taken: 0, time taken: 18.45ms

DFS: average steps taken: 159, turns taken: 388, time taken: 18.45ms

Planned BFS: average steps taken: 0, turns taken: 0, time taken: 18.45ms