HMM & 1st Order Logic Assignment

Carter Kruse (November 6, 2023)

Instructions

While there is no new provided code for this assignment, you may use previous code from the Mazeworld assignment, if you find it helpful. You can draw fancy graphics, or you can represent the mazes using ASCII art; either is perfectly acceptable as long as the information is presented clearly.

The following problem is based on a problem from Rob Schapire, a professor at Princeton. (This general type of problem, however, is quite standard in robotics, and the approach of using a Markov model to integrate sensor information is very standard.)

A robot is in a $ 4 \times 4 $ maze of the type we used in the Mazeworld assignment. The physics of the maze are the same as for the blind robot problem. If the robot tries to move west, but there is a wall there, then the robot fails to move.

However, in this case we will not try to come up with a clever sequence of movements to figure out where the robot is. Instead, we will rely on sensor information to figure out a probability distribution over locations that the robot might be in.

For simplicity, let us first consider the case where there are no walls. There are 16 possible locations for the robot, and we do not know which location the robot is in. The state of the robot is a single random variable, `X`, with sixteen possible values (labelled 0 through 0, perhaps). At time zero, `X` has the distribution $ \left(0.0625, 0.0625, 0.0625, \ldots, 0.0625 \right) $, since $ 0.0625 $ is $ \frac{1}{16} $. This represents the belief of the robot that it might be at any of the 16 locations with equal probability.

The robot has an initial location and will take some moves. The robot does not know either its initial location or how it moved, although it does not that it only makes one move per time step. For simplicity, let us start by assuming that the robot chooses a direction uniformly at random from north, south, east, and west. If the way is blocked, then the robot stays where it was originally. (This still takes a turn.)

So how can the robot figure out something about where it is? Each empty square of the maze has a floor that is painted with one of four colors: red, green, yellow, or blue. The robot has exactly one sensor, pointing downwards. That sensor mostly works. If the robot is in a blue square, the probability of receiving the evidence 'b' is $ 0.88 $, but the sensor might also give the evidence 'r' (with

probability $ 0.04 $), 'g' (with probability $ 0.04 $), or 'y' (with probability $ 0.04 $). The situation for squares with other colors is symmetric.

So the robot receives a sequence of colors as sensor input, one color after each attempted move. (Recall that a move might not actually change the robot location, in the case that there is a wall in the way.) For example, assume that $ \left(0 , \right) $ is red and $ \left(1, 0 \right) $ is blue. Then perhaps the above sequence of actions `e w w e` would give the sensor results `b r r b`, based on the locations visited as described above. (Or perhaps it would give `b g r b`, with the second sensor reading being an error.)

Task

So, the task is as follows. Given a sequence of sensor readings, knowledge of the maze (where the walls are), and the colors of each location of the maze, compute a sequence of probability distributions describing the possible locations of the robot at each time step.

You should use the "filtering" method described in the chapter of the book regarding "probabilistic reasoning over time". You will have to think about several things. First, what is the state transition model? Second, what is the sensor model? Third, how do you implement the filtering algorithm, given that there are several possible values for the state variable (the state variable is not boolean).

Explain the model precisely, and explain exactly how you compute each new distribution of possible states. Write the code, and in some way show the distribution after each time step. (Further, show the maze and the actual state of the robot, as well as the sequence of motions taken. Even though the robot does not know these things, knowing the ground truth will help show that the sequence of probabiliity distributions is reasonable.)

For the implementation, if you are comfortable with linear algebra, the easiest way to implement filtering is to use matrix multiplication, as described by the book. If not, you can use the more "ad hoc" method described.

Mazeworld

Here is a maze, drawn in the venerable tradition of ASCII art:

```
.......
.##....
..##...
.......
..##...
```

```
#.###..
....##.
```

The periods represent open floor tiles (places that can be walked on or over), and the number signs represent walls, places in the maze where a simple robot cannot go.

There is a robot. It can move in any of four directions: north (towards the top of the screen), east, south, and west. There are coordinates on the maze. `(0, 0)` is the bottom left corner of the maze. `(6, 0)` is the bottom right corner of the maze.

Report

The report is written in Markdown. There is no detailed description of what should be in the report, as it's up to the student to describe the implementation and results clearly and convincingly. The Markdown, PDF, and Python source files are submitted in a bundle as a single ZIP. There is a short README file that explains how to run the code.

HMM.py

In this file, we implement the filtering algorithm, alonside the transition and sensor models, as outlined in the problem statement. The objective is to use a HMM (Hidden Markov Model) to solve the state estimation / localization problem in the context of Mazeworld using filtering and forward-backward smoothing.

The methods are implemented in Python, and while the file contains quite a few comments, we will quickly review the main components/structure here.

The constructor is responsible for the initialization of the maze, sensors, and maze colors, along with the color frequency that determines how many of each color are included in the maze. This is used in the sensor model to determine the likelihood that the robot is at a given square in the Mazeworld. To create the dictionary containing the color frequencies, we update according to the values in `maze_colors` , not including those that are walls.

```
# Constructor
def __init__(self, maze, sensors, maze_colors):
    # Initialize the maze, sensor readings, and colors.
    self.maze = maze
    self.sensors = sensors
    self.maze_colors = maze_colors

    # Construct a dictionary with the color frequency values.
    self.color_frequency = {'r': 0, 'g': 0, 'y': 0, 'b': 0}
```

```
        # Update the color frequency dictionary according to the maze.
        for color in maze_colors.values():
            if color != '0':
                self.color_frequency[color] += 1
```

The `initialize_state()` method is responsible for creating the initial state of the probability distribution, by assigning the appropriate values to the correct locations. Initially, we construct a matrix of all zeros equivalent in size to the maze. Following this, we cycle through the empty locations on the grid, and set the initial probability according to the number of empty spaces that there are in total (i.e. the length of `maze_colors`).

```
    def initialize_state(self):
        # Initialize the state with an array of all zeros.
        state = np.zeros((self.maze.width, self.maze.height))

        # Update the array, according to certain constraints.
        for i in range(self.maze.width):
            for j in range(self.maze.height):
                # Check to make sure the location is a floor.
                if self.maze.is_floor(i, j):
                    # Determine the probability by the number of open positions.
                    state[j, i] = 1 / len(self.maze_colors)

        return state
```

The `filter()` method is where we perform the discrete filtering algorithm. Firstly, we initialize the state and add it to the sequence of states, which represents the list of probability distributions at each timestep/iteration. Following this, we cycle through the sensor readings and apply the transition model to each state, followed by teh sensor model, according to the color. We normalize the state to maintain the probability distribution and add it to the list of states in the sequence. As implied, the filtering algorithm is broken down into smaller components, which will be discussed later.

```
    def filter(self):
        # Initialize the state.
        state = self.initialize_state()

        # Add the start state to the sequence.
        sequence = [np.flipud(state)]

        # Cycle through the sensor readings.
        for color in self.sensors:
            # Apply the transition, followed by the sensor model.
            state = self.apply_transition(state)
```

```
            state *= self.sensor_model(color)

            # Normalize the state, which represents a probability distribution.
            state = state / np.sum(state)

            # Add the state to the sequence.
            sequence.append(np.flipud(state))

        # Return the probability distributions given sensor readings.
        return sequence
```

The `apply_transition()` method is responsible for reshaping the state to allow for matrix multiplication, and reshaping the state back to the appropriate size. This encapsulates the modifications required to actually apply the transition matrix with the way it is set up. Specifically, the state is multiplied (in terms of matrices) by the transpose of the transition model.

```
    def apply_transition(self, state):
        # Reshape the state to be a (1, n^2) matrix.
        state = np.reshape(state, (1, self.maze.width * self.maze.height))

        # Use matrix multiplication to apply the transition model.
        state = np.matmul(state, np.transpose(self.transition_model()))

        # Reshape the state back to be an (n, n) matrix.
        state = np.reshape(state, (self.maze.width, self.maze.height))

        return state
```

The `sensor_model()` method is responsible for encapsulating all of the information regarding the sensor reading, and the corresponding matrix that encodes the associated probabilities. We first construct an empty matrix of zeros to represent the sensor model, and cycle through all of the empty cells of the maze, updating the sensor model according to whether the color matches or not. This requires use of the `color_frequency` dictionary/map, which is used to determine the total number of cells that are of any given color.

```
    def sensor_model(self, color):
        # Construct an empty matrix to represent the sensor model.
        matrix = np.zeros((self.maze.width, self.maze.height))

        # Cycle through the cells of the maze.
        for i in range(self.maze.width):
            for j in range(self.maze.height):
                # Check to make sure the location is a floor.
                if self.maze.is_floor(i, j):
                    # Update the sensor model, according to whether the color matches.
```

```
                if self.maze_colors[(i, j)] == color:
                    matrix[j, i] = 0.88 / self.color_frequency[color]
                else:
                    matrix[j, i] = 0.04 / self.color_frequency[self.maze_colors[(i, j)]]

        # Return the probability matrix/distribution.
        return matrix
```

The `apply_transition()` method relies on the `transition_model()` method, which is responsible for encapsulating all of the information regarding the transition from one location to a different one within the Mazeworld. The corresponding matrix encodes the associated probabilities. We first construct an empty matrix of zeros to represent the transition model, and cycle through all of the cells of the maze. For each cell, we determine the possible moves from a given location, iterate through them, and update the transition matrix according to the transition between the "current" location and "next" location. This serves as the appropriate matrix to use for the transition model.

```
    def transition_model(self):
        # Construct an empty matrix to represent the transition model.
        matrix = np.zeros((self.maze.width * self.maze.height, self.maze.width * self.maze.he

        # Cycle through the cells of the maze.
        for j in range(self.maze.height):
            for i in range(self.maze.width):
                # Determine the 'current' cell.
                current = self.maze.width * j + i

                # Compute the possible moves from a given location.
                moves = self.get_moves((i, j))

                # Iterate through the possible moves.
                for move in moves:
                    # Update the 'next' cell, according to each move.
                    next = self.maze.width * move[1] + move[0]

                    # Update the transition model, according to the 'current' and 'next' loca
                    matrix[next, current] += 1 / len(moves)

        # Return the probability matrix/distribution.
        return matrix
```

In order to determine the appropriate moves from a given location, we must implement the `get_moves()` method, which cycles through the list of all possible moves, creates a new location representing the move of the robot, and checks if it is a valid move. If this is not the case, the robot simply stays in the location it was already in.

```
    def get_moves(self, location):
        # Initialize an empty list of possible moves from a given location.
        moves_list = []

        # Enumerate the possible moves.
        possible_moves = [(0, -1), (0, 1), (-1, 0), (1, 0)]

        # Cycle through all of the possible moves.
        for move in possible_moves:
            # Update the location, according to the original location and the move.
            new_location = (location[0] + move[0], location[1] + move[1])

            # Check if the location is a floor.
            if self.maze.is_floor(new_location[0], new_location[1]):
                moves_list.append(new_location)
            # Otherwise, do not move the robot.
            else:
                moves_list.append(tuple(location))

        return moves_list
```

HMMProblem.py

In this file, we define the problem set up, along with the respective testing for the HMM for 4x4 mazes. In essence, the `HMM.py` file contains the filtering and forward-backward smoothing algorithms, while this file is specific to the problem of state estimation and localization for the Mazeworld problem.

The methods are implemented in Python, and while the file contains quite a few comments, we will quickly review the main components/structure here.

The constructor is responsible for the initialization of the maze, possible colors, and maze colors, along with the path that the robot will take in the process of state estimation. The code is written in such a way that a specific path may be passed in, or the length of the path may be passed to create a random path that does not violate any of the constraints (i.e. the robot hitting a wall).

```
    # Constructor
    def __init__(self, maze, path):
        # Initialize the maze, possible colors, and maze_colors.
        self.maze = maze
        self.possible_colors = ['r', 'g', 'y', 'b']
        self.maze_colors = self.create_maze_colors()

        # Determine if the path is pre-set, or a path length is given.
```

```
        if type(path) is int:
            # Construct a random path using the path length.
            self.path = self.create_random_path(path)
        else:
            self.path = path
```

The `create_maze_colors()` method is responsible for setting up the maze with various colors, selected from the provided colors: red, green, yellow, and blue. In this method, we cycle through all of the cells in the maze and set the floor locations to be a random color, which is stored in a dictionary.

```
    def create_maze_colors(self):
    # Initialize an empty dictionary holding the colors of the map.
    maze_colors = {}

    # Cycle through the cells of the maze.
    for i in range(self.maze.width):
        for j in range(self.maze.height):
            # Check to make sure the location is a floor.
            if self.maze.is_floor(i, j):
                # Update the maze color at the location to be a random choice.
                maze_colors[(i, j)] = random.choice(self.possible_colors)
            else:
                # Otherwise, the maze color is not relevant.
                maze_colors[(i, j)] = '0'

    return maze_colors
```

The `create_random_path()` method is responsible for creating a random path for the robot to follow, according to the constraints of the maze and the robot's starting location. The possible moves are defined according to the problem set up: west, east, south, and north. This method takes in a parameter that provides the path length, and generates a path of exactly this length. As given by the problem, it may be the case that the robot tries to move into a wall, in which case the robot simply remains where it is. This aspect is incorporated into the code.

```
    def create_random_path(self, path_length):
    # Initialize an empty path.
    path = []

    # Enumerate the possible moves of the robot, one for each time step.
    possible_moves = [(0, -1), (0, 1), (-1, 0), (1, 0)]

    # Set the location of the robot, according to the maze.
    location = [self.maze.robotloc[0], self.maze.robotloc[1]]

    # Cycle through the pre-determined length of the path.
```

```
    for _ in range(path_length):
        # Select a random move from the choice of possible moves.
        move = random.choice(possible_moves)

        # Create a new location, according to the move.
        new_location = (location[0] + move[0], location[1] + move[1])

        # Check if the location is a floor.
        if test_maze.is_floor(new_location[0], new_location[1]):
            # Update the path, and the current location of the robot.
            path.append(new_location)
            location = new_location
        # Otherwise, the robot remains in the same place.
        else:
            path.append(tuple(location))

    return path
```

The `move_robot()` method is responsible for updating the location of the robot, alongside determining the actual color at a given cell location, along with the sensor reading. Initially, we check to make sure that the location provided is actually an open location. Following this, we determine the actual color of the square that the robot is on, and simulate the sensor reading using a probability technique.

```
    def move_robot(self, new_location):
        # Check to make sure the location is a floor.
        if self.maze.is_floor(new_location[0], new_location[1]):
            # Update the robot location, according to the specified parameter.
            self.maze.robotloc[0], self.maze.robotloc[1] = new_location[0], new_location[1]

        # Determine the actual color of the maze at the given location.
        actual_color = self.maze_colors[self.maze.robotloc[0], self.maze.robotloc[1]]

        # Determine the other colors that are possible, beyond that that is the actual one.
        other_colors = [color for color in self.possible_colors if color != actual_color]

        # Simulate the sensor, by determining the sensor color according to a random value.
        sensor_color = None
        random_value = random.random()

        # The probability values are set in accordance to the problem set up.
        if random_value < 0.88:
            sensor_color = actual_color
        elif random_value < 0.92:
            sensor_color = other_colors[0]
        elif random_value < 0.96:
            sensor_color = other_colors[1]
```

```
        else:
            sensor_color = other_colors[2]

        # Return the actual color at the location, along with the sensor reading.
        return actual_color, sensor_color
```

Finally, the `solution()` method is responsible for determining the "solution" to the problem, by applying the HMM filtering (or forward-backward smoothing) algorithm, and printing out the resuts. The `locations` array contains a string representation of the maze (with the robot location) at each timestep/iteration. Similarly, the `colors` and `sensors` arrays (stored as strings) maintain the actual color of the square that the robot is on, and the sensor reading, for each timestep/iteration.

As implied, we cycle through the locations of the robot's path to determine the actual color and sensor color, and run the HMM algorithms with the given sensors and maze colors. Following this, we apply the filtering and forward-backward smoothing, which gives us our results, which we output to the terminal.

```
    def solution(self):
        # Initialize the locations (maze), (actual) colors, and sensor readings.
        locations = [str(self.maze)]
        colors = '0'
        sensors = '0'

        # Cycle through each location in the path.
        for location in self.path:
            # Determine the actual and sensor color after the robot move.
            actual_color, sensor_color = self.move_robot(location)

            # Update the locations array with a string representation of the maze.
            locations.append(str(self.maze))

            # Add the actual and sensor colors to the arrays (given as strings).
            colors += actual_color
            sensors += sensor_color

        # Run the HMM algorithm on the maze, with the given sensors and maze colors.
        hmm = HMM(self.maze, sensors[1:], self.maze_colors)

        # Filtering Only
        start = time()
        filter_path = hmm.filter()
        filter_time = time() - start

        # Forward-Backward Smoothing
        start = time()
        smooth_path = hmm.smooth()
```

```
    smooth_time = time() - start

    # Print the maze, readings, colors, and distributions.
    for i, distribution in enumerate(filter_path):
        print('Iteration ' + str(i))
        print(self.path[:i])
        print('----------')
        print('Actual Color: ' + colors[i])
        print('Sensor Color: ' + sensors[i])
        print()
        print('Robot Location: ')
        print(locations[i])
        print('Distribution: ')
        print(distribution)
        print()
        print('Smooth Distribution: ')
        print(smooth_path[i])
        print()
```

IMPORTANT - The forward-backward smoothing is part of the extra credit included for this assignment, and thus, will be discussed later.

Results

The results of implementing the filtering and forward-backward smoothing is presented for the following cases:

```
test_maze = Maze('maze1.maz')
test_problem = HMMProblem(test_maze, 12)
```

```
Iteration 0
[]
----------
Actual Color: 0
Sensor Color: 0

Robot Location:
....
....
....
A...

Distribution:
[[0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]
```

```
 [0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]]

Smooth Distribution:
[[2.46731252e-03 8.37267059e-04 2.84415181e-04 4.84059414e-06]
 [6.11229368e-02 3.85788747e-02 1.19101800e-03 4.57246989e-05]
 [1.89795830e-01 1.17424975e-01 3.70350754e-02 6.87276732e-04]
 [3.02412550e-01 1.87905371e-01 5.93211785e-02 8.85353797e-04]]

Iteration 1
[(0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[0.008 0.176 0.008 0.01 ]
 [0.02  0.008 0.01  0.008]
 [0.176 0.176 0.01  0.02 ]
 [0.176 0.176 0.008 0.01 ]]

Smooth Distribution:
[[6.38142245e-06 1.05408619e-03 7.43767486e-07 2.04227235e-08]
 [9.87619295e-04 9.96087429e-05 4.82453971e-06 1.17635377e-07]
 [2.37350076e-01 1.46935118e-01 1.49395207e-04 1.11298737e-05]
 [3.78258662e-01 2.34985284e-01 1.53362201e-04 3.57164046e-06]]

Iteration 2
[(0, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[0.00392658 0.08149506 0.00377841 0.00083347]
 [0.00981645 0.00707525 0.00078717 0.00088904]
 [0.22329648 0.15076587 0.00495453 0.00222259]
```

```
 [0.28686263 0.21840677 0.00377841 0.0011113 ]]

Smooth Distribution:
[[2.36190623e-06 3.69889744e-04 2.75278383e-07 1.38099715e-09]
 [3.62440089e-04 6.61332496e-05 2.82757334e-07 9.97029602e-09]
 [2.25435562e-01 9.40964361e-02 5.55498038e-05 9.21000890e-07]
 [4.61257208e-01 2.18298479e-01 5.41513526e-05 2.97865332e-07]]

Iteration 3
[(0, 0), (0, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[1.51992888e-03 3.24641364e-02 1.33185400e-03 1.21361523e-04]
 [9.35406358e-03 3.72246309e-03 3.19904716e-04 7.25330272e-05]
 [2.26174728e-01 1.52999266e-01 3.01860174e-03 3.51664530e-04]
 [3.42403581e-01 2.22489880e-01 3.49847656e-03 1.57557065e-04]]

Smooth Distribution:
[[8.39899810e-07 1.40779928e-04 9.64835153e-08 2.37457533e-10]
 [3.15565717e-04 3.15500189e-05 1.06936706e-07 7.23667615e-10]
 [2.07884661e-01 8.72676557e-02 3.06642770e-05 1.35114250e-07]
 [5.01828572e-01 2.02453519e-01 4.58148853e-05 3.82826313e-08]]

Iteration 4
[(0, 0), (0, 0), (0, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[6.49229019e-04 1.24300236e-02 4.95514555e-04 2.97982043e-05]
 [8.71168119e-03 2.82421598e-03 1.47361031e-04 1.25258257e-05]
 [2.32732427e-01 1.45003530e-01 2.84338207e-03 1.30269563e-04]
 [3.60903295e-01 2.29694703e-01 3.31668959e-03 7.53544818e-05]]
```

```
Smooth Distribution:
[[3.65698151e-07 5.71037152e-05 4.53950219e-08 8.31414729e-11]
 [2.82599242e-04 2.36185521e-05 4.63006526e-08 1.36535491e-10]
 [2.07246737e-01 7.94533817e-02 2.84384741e-05 4.70385945e-08]
 [5.10388060e-01 2.02477803e-01 4.17357434e-05 1.80290168e-08]]

Iteration 5
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
.A..

Distribution:
[[3.17549631e-04 5.10534634e-03 1.85415621e-04 1.00407501e-05]
 [8.66454040e-03 2.35319828e-03 1.09238942e-04 4.52766202e-06]
 [2.32665972e-01 1.45727677e-01 2.62850101e-03 1.08308967e-04]
 [3.68676719e-01 2.30040691e-01 3.33863557e-03 6.36380290e-05]]

Smooth Distribution:
[[1.88744738e-07 3.05081960e-05 2.52443984e-08 5.96620657e-11]
 [2.82766631e-04 1.89388646e-05 3.87955865e-08 4.38250765e-11]
 [2.03707393e-01 8.04307502e-02 2.51446871e-05 4.32599277e-08]
 [5.16097453e-01 1.99364452e-01 4.22825466e-05 1.46354218e-08]]

Iteration 6
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
.A..

Distribution:
[[2.02392349e-04 2.46093061e-03 7.60119473e-05 3.68860209e-06]
 [8.57064148e-03 2.24250101e-03 9.08280029e-05 3.26127129e-06]
 [2.33600309e-01 1.44564113e-01 2.62182753e-03 9.85258879e-05]
 [3.70942784e-01 2.31142570e-01 3.31684172e-03 6.27729611e-05]]
```

```
Smooth Distribution:
[[1.62087646e-07 2.23291394e-05 2.22970528e-08 4.83175826e-11]
 [2.71063802e-04 2.05502110e-05 2.81556169e-08 6.12494529e-11]
 [2.07312045e-01 7.68761522e-02 2.81703036e-05 3.41903658e-08]
 [5.12459154e-01 2.02969718e-01 4.05541664e-05 1.63339109e-08]]

Iteration 7
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[1.60226118e-04 1.53552791e-03 3.68673777e-05 1.51749166e-06]
 [8.56781748e-03 2.18120560e-03 8.65763165e-05 2.75026309e-06]
 [2.33535488e-01 1.44744826e-01 2.59312592e-03 9.75949080e-05]
 [3.71913425e-01 2.31158596e-01 3.32244482e-03 6.20113159e-05]]

Smooth Distribution:
[[1.77634033e-07 3.01584124e-05 2.37166702e-08 6.26045693e-11]
 [3.02585854e-04 1.75619741e-05 5.33877788e-08 6.13981046e-11]
 [2.00532206e-01 8.66519772e-02 2.33050305e-05 6.44665656e-08]
 [5.15926791e-01 1.96469492e-01 4.55890062e-05 1.48539501e-08]]

Iteration 8
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
.A..

Distribution:
[[1.45914217e-04 1.20530219e-03 2.32438286e-05 7.46323998e-07]
 [8.55445483e-03 2.16880481e-03 8.42331711e-05 2.63780314e-06]
 [2.33668213e-01 1.44577496e-01 2.59406335e-03 9.64293612e-05]
 [3.72176113e-01 2.31320860e-01 3.31947540e-03 6.20130378e-05]]

Smooth Distribution:
```

```
[[3.77392827e-07 5.24135308e-05 4.22353803e-08 1.30495681e-10]
 [2.61434051e-04 3.44907460e-05 4.12677691e-08 5.05406307e-10]
 [2.24411681e-01 7.24034880e-02 4.49794726e-05 7.12465926e-08]
 [4.82810626e-01 2.19941368e-01 3.89487252e-05 3.78543532e-08]]

Iteration 9
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[1.40654030e-04 1.09079699e-03 1.83804493e-05 4.78817842e-07]
 [8.55466261e-03 2.16085357e-03 8.37625267e-05 2.57540509e-06]
 [2.33651457e-01 1.44613513e-01 2.59010332e-03 9.63833132e-05]
 [3.72296691e-01 2.31317230e-01 3.32054051e-03 6.19187814e-05]]

Smooth Distribution:
[[6.90785701e-07 1.31025132e-04 9.32686321e-08 7.63641970e-10]
 [4.72583031e-04 2.46777072e-05 2.77397834e-07 4.51033445e-09]
 [1.87782330e-01 1.38825299e-01 2.54255370e-05 6.80522990e-07]
 [4.86470989e-01 1.86189925e-01 7.58416245e-05 1.56778024e-07]]

Iteration 10
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
A...
....

Distribution:
[[1.38893235e-04 1.04987485e-03 1.66980634e-05 3.83261151e-07]
 [8.55274760e-03 2.15953092e-03 8.34594933e-05 2.56329656e-06]
 [2.33670693e-01 1.44588795e-01 2.59047821e-03 9.62278557e-05]
 [3.72326610e-01 2.31341009e-01 3.32010862e-03 6.19269812e-05]]

Smooth Distribution:
[[1.18132160e-06 3.17975446e-04 3.20413668e-07 9.32384448e-09]
```

```
 [2.77827606e-04 1.32094228e-04 1.47556223e-06 6.07395903e-08]
 [3.46587970e-01 6.74270474e-02 2.22860877e-04 5.76856369e-06]
 [2.37890987e-01 3.47067733e-01 6.46476680e-05 2.04083364e-06]]

Iteration 11
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
.A..
....

Distribution:
[[1.38238461e-04 1.03576527e-03 1.60956589e-05 3.50266832e-07]
 [8.55286623e-03 2.15848573e-03 8.34095809e-05 2.55526167e-06]
 [2.33667261e-01 1.44595331e-01 2.58991545e-03 9.62311256e-05]
 [3.72341753e-01 2.31339535e-01 3.32029337e-03 6.19142490e-05]]

Smooth Distribution:
[[4.24718351e-06 5.09159043e-04 5.12499047e-06 1.39409808e-07]
 [6.56937153e-04 3.85841205e-04 1.77637739e-05 8.35027335e-07]
 [1.57940200e-01 5.41983711e-01 5.78702054e-04 7.86178000e-05]
 [1.83034879e-01 1.13721342e-01 1.05720878e-03 2.52909961e-05]]

Iteration 12
[(0, 0), (0, 0), (0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1),
----------
Actual Color: g
Sensor Color: g

Robot Location:
....
....
..A.
....

Distribution:
[[1.15840890e-03 3.93207114e-04 1.33350102e-04 6.24893959e-05]
 [7.17808882e-02 1.81148234e-02 1.53936858e-02 2.14354648e-05]
 [8.91439238e-02 5.51609349e-02 4.78226778e-01 8.07476723e-04]
 [1.42047706e-01 8.82561568e-02 2.78662865e-02 1.14324493e-02]]

Smooth Distribution:
[[7.99209109e-05 2.71281329e-05 9.20008608e-06 1.18559827e-04]
 [1.23807620e-02 1.24977733e-03 2.92061189e-02 1.47887492e-06]
```

```
 [6.15021480e-03 3.80566149e-03 9.07329690e-01 1.39273522e-04]
 [9.80015087e-03 6.08896601e-03 1.92254997e-03 2.16905476e-02]]

-----Time Analysis-----
Filtering Only: 0.0007658004760742188
Forward-Backward Smoothing: 0.001383066177368164



test_maze = Maze('maze1.maz')
path = [(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3)]
test_problem = HMMProblem(test_maze, path)



Iteration 0
[]
----------
Actual Color: 0
Sensor Color: 0

Robot Location:
....
....
....
A...

Distribution:
[[0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]
 [0.0625 0.0625 0.0625 0.0625]]

Smooth Distribution:
[[0.02892959 0.02467271 0.02408882 0.00441052]
 [0.07191283 0.09109646 0.00701927 0.01133292]
 [0.18311122 0.09131955 0.07316009 0.01459392]
 [0.18927032 0.13940443 0.03088315 0.01479419]]

Iteration 1
[(1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
.A..
```

```
Distribution:
[[0.008 0.176 0.008 0.01 ]
 [0.02  0.008 0.01  0.008]
 [0.176 0.176 0.01  0.02 ]
 [0.176 0.176 0.008 0.01 ]]

Smooth Distribution:
[[2.33049266e-04 9.30368219e-02 1.86751625e-04 2.07930408e-04]
 [2.92130068e-03 3.52016261e-04 3.75468054e-04 3.74501658e-04]
 [2.82083843e-01 2.80850919e-01 6.36781062e-04 3.63945076e-03]
 [2.27859411e-01 1.06162674e-01 3.06271870e-04 7.72808786e-04]]

Iteration 2
[(1, 0), (1, 1)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
.A..
....

Distribution:
[[0.00392658 0.08149506 0.00377841 0.00083347]
 [0.00981645 0.00707525 0.00078717 0.00088904]
 [0.22329648 0.15076587 0.00495453 0.00222259]
 [0.28686263 0.21840677 0.00377841 0.0011113 ]]

Smooth Distribution:
[[4.90787310e-04 8.84859556e-02 1.94637458e-04 8.69323296e-05]
 [3.49283142e-03 9.40792705e-04 8.96836377e-05 6.27488765e-04]
 [6.20516165e-01 1.50700651e-01 4.23074854e-03 3.98792085e-03]
 [2.75647151e-02 9.74306248e-02 1.90234709e-04 9.69830911e-04]]

Iteration 3
[(1, 0), (1, 1), (1, 2)]
----------
Actual Color: y
Sensor Color: r

Robot Location:
....
.A..
....
....
```

```
Distribution:
[[5.59462774e-03 5.43161654e-03 4.90235264e-03 4.46713365e-04]
 [7.57479575e-01 1.37018222e-02 1.17752076e-03 2.66983076e-04]
 [3.78415855e-02 2.55985045e-02 1.11110153e-02 2.84773239e-02]
 [5.72879849e-02 3.72250688e-02 1.28773618e-02 5.79943667e-04]]

Smooth Distribution:
[[1.64928584e-04 7.37505704e-05 1.12162393e-03 7.10060162e-05]
 [8.16982586e-01 2.50287777e-03 8.60547232e-05 5.92021782e-05]
 [7.58024354e-04 1.51939946e-03 2.29098067e-03 1.72181821e-01]
 [3.05174938e-04 3.55077429e-04 1.44895387e-03 7.85388974e-05]]

Iteration 4
[(1, 0), (1, 1), (1, 2), (2, 2)]
----------
Actual Color: g
Sensor Color: g

Robot Location:
....
..A.
....
....

Distribution:
[[0.08441782 0.00323128 0.00130408 0.01818193]
 [0.22209083 0.0861176  0.0899151  0.00331177]
 [0.095771   0.01089214 0.20432073 0.01102395]
 [0.02068106 0.01450281 0.00673874 0.12749916]]

Smooth Distribution:
[[9.21372302e-03 6.22600025e-04 6.75788792e-05 3.05688418e-02]
 [8.52302316e-02 1.06767118e-03 3.81431749e-01 4.54072686e-05]
 [2.23648343e-03 1.15306041e-03 3.06112795e-01 4.49645195e-04]
 [5.59255413e-04 9.59685828e-04 3.91330548e-04 1.79889941e-01]]

Iteration 5
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
....
.A..
....
....

Distribution:
```

```
[[0.26348758 0.00531963 0.07529279 0.00155649]
 [0.03710057 0.21801178 0.01120674 0.08184416]
 [0.01061778 0.01217587 0.00450352 0.02629534]
 [0.00460754 0.00160481 0.23601542 0.01036   ]]

Smooth Distribution:
[[2.68007710e-01 2.32027603e-04 7.36944554e-02 1.52752201e-05]
 [4.28829927e-03 3.99571585e-01 1.03871574e-04 1.88486483e-02]
 [1.26894556e-03 9.82494731e-04 2.62726780e-04 6.88158744e-04]
 [7.07409338e-04 1.87592746e-04 2.31005237e-01 1.35562497e-04]]

Iteration 6
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.A..
....
....
....

Distribution:
[[0.01954232 0.42443135 0.00320476 0.00687496]
 [0.04540843 0.00225843 0.01628764 0.00414952]
 [0.04870308 0.17724254 0.01225666 0.01055402]
 [0.01618686 0.19209148 0.00866554 0.01214243]]

Smooth Distribution:
[[1.02168176e-03 4.88168508e-01 1.67546479e-04 4.49282584e-04]
 [5.93493694e-03 1.18071781e-04 1.06440698e-03 2.16939091e-04]
 [5.60168519e-02 2.03859178e-01 8.00980420e-04 1.37942312e-03]
 [1.86176484e-02 2.20937991e-01 4.53038533e-04 7.93515439e-04]]

-----Time Analysis-----
Filtering Only: 0.0004642009735107422
Forward-Backward Smoothing: 0.0006957054138183594



test_maze = Maze('maze2.maz')
path = [(1, 1), (1, 0), (1, 0), (1, 1), (1, 2), (1, 3), (1, 2)]
test_problem = HMMProblem(test_maze, path)



Iteration 0
[]
----------
```

```
Actual Color: 0
Sensor Color: 0

Robot Location:
##.#
#...
#.#.
.A#.

Distribution:
[[0.     0.     0.0625 0.    ]
 [0.     0.0625 0.0625 0.0625]
 [0.     0.0625 0.     0.0625]
 [0.0625 0.0625 0.     0.0625]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 1.71423060e-05 0.00000000e+00]
 [0.00000000e+00 2.50673231e-01 1.89882848e-03 3.78807227e-05]
 [0.00000000e+00 4.96842604e-01 0.00000000e+00 3.66043752e-05]
 [1.22605577e-03 2.49225569e-01 0.00000000e+00 4.20841890e-05]]

Iteration 1
[(1, 1)]
----------
Actual Color: r
Sensor Color: r

Robot Location:
##.#
#...
#A#.
..#.

Distribution:
[[0.         0.         0.00833333 0.        ]
 [0.         0.01388889 0.00833333 0.01388889]
 [0.         0.91666667 0.         0.00833333]
 [0.00833333 0.00833333 0.         0.01388889]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 9.69356256e-08 0.00000000e+00]
 [0.00000000e+00 1.15111211e-04 3.40245448e-07 6.61411954e-07]
 [0.00000000e+00 9.99839714e-01 0.00000000e+00 2.60546927e-07]
 [1.01740236e-06 4.20820556e-05 0.00000000e+00 7.15933437e-07]]

Iteration 2
[(1, 1), (1, 0)]
----------
Actual Color: b
```

```
Sensor Color: r

Robot Location:
##.#
#...
#.#.
.A#.

Distribution:
[[0.00000000e+00 0.00000000e+00 1.61066619e-04 0.00000000e+00]
 [0.00000000e+00 7.67303476e-03 2.14755492e-04 3.57925820e-04]
 [0.00000000e+00 9.86264597e-01 0.00000000e+00 2.14755492e-04]
 [1.61066619e-04 4.55013199e-03 0.00000000e+00 4.02666547e-04]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 8.62338836e-08 0.00000000e+00]
 [0.00000000e+00 4.06661982e-05 1.70086271e-07 6.40816491e-07]
 [0.00000000e+00 9.99905158e-01 0.00000000e+00 3.26010299e-07]
 [7.43019337e-07 5.15578962e-05 0.00000000e+00 6.52092989e-07]]

Iteration 3
[(1, 1), (1, 0), (1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
##.#
#...
#.#.
.A#.

Distribution:
[[0.00000000e+00 0.00000000e+00 4.53799860e-04 0.00000000e+00]
 [0.00000000e+00 4.93463272e-02 5.46596113e-03 5.64164550e-05]
 [0.00000000e+00 2.93285349e-01 0.00000000e+00 7.73786940e-04]
 [3.27259514e-03 6.47275685e-01 0.00000000e+00 7.00798152e-05]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 9.23538466e-06 0.00000000e+00]
 [0.00000000e+00 5.08077518e-03 4.74115607e-04 4.83849372e-07]
 [0.00000000e+00 1.71614871e-01 0.00000000e+00 1.49063465e-04]
 [2.69148781e-06 8.22666999e-01 0.00000000e+00 1.76560393e-06]]

Iteration 4
[(1, 1), (1, 0), (1, 0), (1, 1)]
----------
Actual Color: r
Sensor Color: r
```

```
Robot Location:
##.#
#...
#A#.
..#.

Distribution:
[[0.00000000e+00 0.00000000e+00 4.73668803e-05 0.00000000e+00]
 [0.00000000e+00 4.59564622e-03 3.83816612e-04 7.34549205e-05]
 [0.00000000e+00 9.79279136e-01 0.00000000e+00 1.16143681e-05]
 [4.55878472e-03 1.10388022e-02 0.00000000e+00 1.13783012e-05]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 1.02056562e-08 0.00000000e+00]
 [0.00000000e+00 1.29652594e-04 6.03463266e-06 1.88169406e-06]
 [0.00000000e+00 9.99854241e-01 0.00000000e+00 2.79906446e-07]
 [5.20051408e-08 6.92445118e-06 0.00000000e+00 9.23122190e-07]]

Iteration 5
[(1, 1), (1, 0), (1, 0), (1, 1), (1, 2)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
##.#
#A..
#.#.
..#.

Distribution:
[[0.00000000e+00 0.00000000e+00 1.11454125e-05 0.00000000e+00]
 [0.00000000e+00 7.68391306e-01 1.08086920e-04 4.21427095e-04]
 [0.00000000e+00 2.09188728e-01 0.00000000e+00 2.29008479e-06]
 [5.23771776e-04 2.13176948e-02 0.00000000e+00 3.55495694e-05]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 5.51767336e-09 0.00000000e+00]
 [0.00000000e+00 9.79256524e-01 3.52333841e-06 4.97104914e-04]
 [0.00000000e+00 2.01206984e-02 0.00000000e+00 9.06497261e-08]
 [5.93815994e-08 3.12176704e-05 0.00000000e+00 9.07763097e-05]]

Iteration 6
[(1, 1), (1, 0), (1, 0), (1, 1), (1, 2), (1, 3)]
----------
Actual Color: y
Sensor Color: y
```

```
Robot Location:
##.#
#A..
#.#.
..#.

Distribution:
[[0.00000000e+00 0.00000000e+00 1.98916375e-06 0.00000000e+00]
 [0.00000000e+00 8.99867258e-01 1.08076417e-02 4.91261467e-04]
 [0.00000000e+00 8.49006558e-02 0.00000000e+00 6.48736315e-06]
 [3.21714056e-04 3.54684902e-03 0.00000000e+00 5.61431809e-05]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 3.01356674e-09 0.00000000e+00]
 [0.00000000e+00 9.91415880e-01 3.08367367e-04 5.13950868e-04]
 [0.00000000e+00 7.66383454e-03 0.00000000e+00 1.85099686e-07]
 [4.87394153e-07 1.07468943e-05 0.00000000e+00 8.65449461e-05]]

Iteration 7
[(1, 1), (1, 0), (1, 0), (1, 1), (1, 2), (1, 3), (1, 2)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
##.#
#A..
#.#.
..#.

Distribution:
[[0.00000000e+00 0.00000000e+00 1.41680373e-04 0.00000000e+00]
 [0.00000000e+00 9.10586354e-01 1.19381643e-02 5.66720889e-03]
 [0.00000000e+00 7.03065730e-02 0.00000000e+00 7.34211468e-06]
 [5.91163027e-05 1.20952909e-03 0.00000000e+00 8.40315236e-05]]

Smooth Distribution:
[[0.00000000e+00 0.00000000e+00 4.17149480e-06 0.00000000e+00]
 [0.00000000e+00 9.83047679e-01 3.51495335e-04 6.11818585e-03]
 [0.00000000e+00 1.03501811e-02 0.00000000e+00 2.16173860e-07]
 [1.74056112e-06 3.56121614e-05 0.00000000e+00 9.07184627e-05]]

-----Time Analysis-----
Filtering Only: 0.0004942417144775391
Forward-Backward Smoothing: 0.0008168220520019531
```

Extra Credit/Bonus

The following implementations are designed to enhance the problem, thus meriting "extra credit" as defined by the outlines of the assignment. By going above and beyond to consider extensions to the problem, we consider a greater level of complexity for the problem. Due to the structure of our code, we are able to maintain much of the structure of the previously written code.

According to the instructions, any "scientifically interesting extension related to the theme of the assignment is welcome." The following represent implementation of a few of the ideas that were encouraged.

Forward-Backward Smoothing - Bonus #1

Show that a better estimate of state distribution at time steps other than the first or last time step can be computed by using a forward-backward algorithm.

To implement the forward-backward smoothing, we following the structure of the `filter()` method to construct the `smooth()` method. Firstly, we initialize the states and add them to the sequence of states, which represents the list of probability distributions at each timestep/iteration. Following this, we cycle through the sensor readings forward and in reverse, applying the transition model and sensor model to each state in the correct order, according to the color. We normalize the states to maintain the probability distribution and perform the core aspect of the "smoothing" algorithm.

The forward-backward smoothing algorithm is broken down into smaller components, which were previously discussed.

```
def smooth(self):
    # Initialize the states.
    forward_state = self.initialize_state()
    backward_state = self.initialize_state()

    # Add the start states to the sequences.
    forward_sequence = [np.flipud(forward_state)]
    backward_sequence = [np.flipud(backward_state)]

    # Cycle through the sensor readings.
    for color in self.sensors:
        # Apply the transition, followed by the sensor model.
        forward_state = self.apply_transition(forward_state)
        forward_state *= self.sensor_model(color)

        # Normalize the state, which represents a probability distribution.
        forward_state = forward_state / np.sum(forward_state)

        # Add the state to the sequence.
```

```
            forward_sequence.append(np.flipud(forward_state))

        # Cycle through the sensor readings, in reverse order
        for color in self.sensors[::-1]:
            # Apply the sensor model, followed by the transition.
            backward_state *= self.sensor_model(color)
            backward_state = self.apply_transition(backward_state)

            # Normalize the state, which represents a probability distribution.
            backward_state = backward_state / np.sum(backward_state)

            # Add the state to the sequence.
            backward_sequence.append(np.flipud(backward_state))

        # Reverse the sequence, to consider iteration from 0 to n.
        backward_sequence.reverse()

        # Construct a new sequence to represent the smoothed distributions.
        sequence = []

        # Iterate through the states in the forward sequence.
        for i, state in enumerate(forward_sequence):
            # Apply the backward sequence values to the state.
            smooth_state = state * backward_sequence[i]

            # Normalize the state, which represents a probability distribution.
            smooth_state = smooth_state / np.sum(smooth_state)

            # Add the state to the sequence.
            sequence.append(smooth_state)

        # Return the forward-backward smoothed probability distributions given sensor reading
        return sequence
```

The results of the forward-backward smoothing is that it produces a better estimate of the state distribution at various time steps. While the algorithm takes slightly longer, due to the increase in computational complexity, this is relatively insignificant, as the algorithm still takes less than a fraction of a second to complete. To view the results of the filtering algorithm and the forward-backward smoothing side by side, please see the visualization `comparison.png`, alongside the confidence graph `confidence_values.png`, which will be discussed later.

As an example, consider `Iteration 4` of the following implementation:

```
test_maze = Maze('maze1.maz')
test_problem = HMMProblem(test_maze, 12)
```

```
Iteration 4
[(0, 0), (0, 0), (0, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
....
....
....
A...

Distribution:
[[6.49229019e-04 1.24300236e-02 4.95514555e-04 2.97982043e-05]
 [8.71168119e-03 2.82421598e-03 1.47361031e-04 1.25258257e-05]
 [2.32732427e-01 1.45003530e-01 2.84338207e-03 1.30269563e-04]
 [3.60903295e-01 2.29694703e-01 3.31668959e-03 7.53544818e-05]]

Smooth Distribution:
[[3.65698151e-07 5.71037152e-05 4.53950219e-08 8.31414729e-11]
 [2.82599242e-04 2.36185521e-05 4.63006526e-08 1.36535491e-10]
 [2.07246737e-01 7.94533817e-02 2.84384741e-05 4.70385945e-08]
 [5.10388060e-01 2.02477803e-01 4.17357434e-05 1.80290168e-08]]
```

This reflects a case in which the robot has tried to move left or down multiple times, yet has remained in the same place, as it is unable to move outside of the boundaries. The smoothed distribution provides a more accurate representation of the actual location of the robot, as we consider the forward-backward smoothing instead of just the discrete filtering. This reflects the significance of including forward-backward smoothing.

Visualization & Confidence - Bonus #2

While the output to the terminal provides an indication of the probability distribution according to filtering only and forward-backward smoothing, it does not provide a good visual representation of the most likely state that the robot is in, according to the sensor and transition model.

To this end, the following code is included in `HMMProblem.py` to provide a proper visualization of the final result, which gives the user a solid understanding of the way in which the probability matrix should be interpreted. While the matrix represents a distribution, we are able to say with relatively high confidence where the robot is located after only a couple of movements in the maze, even when the maze is relatively sparse in nature (few walls).

```
# Visualization
plt.subplot(1, 2, 1)
```

```
plt.imshow(filter_path[-1], cmap = 'OrRd', interpolation = 'nearest')
plt.title('Filter')
plt.xticks([])
plt.yticks([])

plt.subplot(1, 2, 2)
plt.imshow(smooth_path[-1], cmap = 'OrRd', interpolation = 'nearest')
plt.title('Smooth')
plt.xticks([])
plt.yticks([])

plt.subplots_adjust(wspace = 0.5)
plt.savefig('comparison_maze4.png')
plt.show()

plt.plot(filter_max_values, label = 'Filter')
plt.plot(smooth_max_values, label = 'Smooth')
plt.legend()
plt.title('Confidence Values')
plt.xlabel('Iteration')
plt.ylabel('Confidence')

plt.savefig('confidence_values.png')
plt.show()
```

The heatmaps provide an understanding of the probability distribution in the visual sense, which further allows for comparison between the filtering and forward-backward smoothing algorithms. Evidently, the forward-backward smoothing algorithm produces a result that indicates a higher level of confidence of the state estimation for the robot.

Further, it is interesting to consider the ways in which the so-called "confidence" changes as the robot moves around the Mazeworld. This "confidence" simply reflects the maximum value in the NumPy array, as that reflects the likelihood that the robot is in the square that it is predicted to be in, given the sequence of color sensor readings.

These "confidence" values are reflected in the following code:

```
# Filtering Only
start = time()
filter_path = hmm.filter()
filter_time = time() - start

# Forward-Backward Smoothing
start = time()
smooth_path = hmm.smooth()
smooth_time = time() - start
```

```
  # Initializing confidence arrays.
  filter_max_values = []
  smooth_max_values = []

  # Print the maze, readings, colors, and distributions.
  for i, distribution in enumerate(filter_path):
      print('Iteration ' + str(i))
      print(self.path[:i])
      print('----------')
      print('Actual Color: ' + colors[i])
      print('Sensor Color: ' + sensors[i])
      print()
      print('Robot Location: ')
      print(locations[i])
      print('Distribution: ')
      print(distribution)
      print()
      print('Smooth Distribution: ')
      print(smooth_path[i])
      print()

      # Append the maximum values of each 2D array.
      filter_max_values.append(distribution.max())
      smooth_max_values.append(smooth_path[i].max())
```

Following the computation of the confidence values, `matplotlib` may be used to graph the way that they change across time. Please see `comparison.png` for a comparsion of the final heatmap between filtering only and forward-backward smoothing and see `confidence_values.png` for a comparison between the maximum values in the distribution matrices (over time), given the same comparison. As we previously found, forward-backward smoothing tends to produce more confident results (in terms of state estimation), though it takes slightly more time computationally.

Maze Expansion - Bonus #3

In the assignment instructions, we are tasked with solving the state estimation / localization problem for Mazeworld, specifically the $ 4 \times 4 $ case. This seems to reflect a rather simplified version of the extended problem, which is a maze of size $ n \times n $. Thus, when writing the code, I chose to design the code in such a way that it handles the $ n \times n $ case, rather than the limited case presented in the instructions. Considering that this required (slightly) further time/consideration, it seems as though solving this merits extra credit.

The code functions without loss of generality for an $ n \times n $ maze, with very little increase in the computational complexity. This is a result of using matrix multiplication, rather than an "ad hoc" method to apply the transition and sensor model to the state representing the probability distribution for a given timestep/iteration.

To experimentally test the correctness of the HMM algorithms (filtering and forward-backward smoothing) on the $ n \times n $ mazes, we present the following results. Images of the final probabiltity distributions are shown in `comparison_maze3.png` and `comparison_maze4.png`. This demonstrates that the algorithm run within less than a second for mazes that are more extensive than the original $ 4 \times 4 $ matrices, and present reasonable solutions.

```
test_maze = Maze('maze3.maz')
path = [(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3), (2, 3), (3, 3), (3, 4)]
test_problem = HMMProblem(test_maze, path)
```

```
Iteration 0
[]
----------
Actual Color: 0
Sensor Color: 0

Robot Location:
.....
.....
.....
.....
A....

Distribution:
[[0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]
 [0.04 0.04 0.04 0.04 0.04]]

Smooth Distribution:
[[0.03180496 0.02921631 0.0123468  0.02254622 0.03961662]
 [0.02378394 0.03795648 0.04528182 0.0454176  0.05597024]
 [0.05771125 0.0261098  0.03998076 0.03977016 0.04832296]
 [0.02470936 0.06915083 0.02292422 0.00904981 0.06851253]
 [0.03639543 0.01456796 0.01611104 0.0633642  0.11937869]]

Iteration 1
[(1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.....
.....
```

```
.....
.....
.A...

Distribution:
[[0.12571429 0.01       0.00444444 0.00444444 0.00444444]
 [0.00444444 0.00444444 0.008      0.008      0.12571429]
 [0.008      0.12571429 0.01       0.008      0.00444444]
 [0.12571429 0.00444444 0.00444444 0.00444444 0.008     ]
 [0.12571429 0.12571429 0.01       0.01       0.12571429]]

Smooth Distribution:
[[8.14776448e-02 5.27797302e-04 1.27019512e-03 2.86900960e-04
  4.04156150e-05]
 [1.57782600e-04 1.26727485e-03 1.07478218e-03 4.97623520e-03
  2.06401378e-01]
 [5.32750358e-04 1.79730201e-01 7.78223271e-03 1.09091334e-03
  7.26727141e-04]
 [1.23261268e-01 2.95271820e-05 1.77939599e-04 7.44384265e-04
  1.29489196e-03]
 [2.75431225e-03 7.04050253e-02 4.55713047e-04 5.56894251e-04
  3.12976814e-01]]

Iteration 2
[(1, 0), (1, 1)]
----------
Actual Color: y
Sensor Color: r

Robot Location:
.....
.....
.....
.A...
.....

Distribution:
[[0.01193448 0.01135913 0.00093877 0.00074481 0.00485454]
 [0.00497868 0.00517264 0.03717516 0.20207138 0.00640116]
 [0.36481696 0.00120699 0.01148133 0.03717516 0.00510281]
 [0.01184471 0.01332229 0.00100859 0.0010629  0.19715565]
 [0.0225722  0.01193448 0.01179554 0.01179554 0.01209408]]

Smooth Distribution:
[[3.52388583e-04 3.56403623e-03 8.34956649e-05 6.37324119e-05
  1.57868296e-04]
 [3.82645259e-04 1.45484695e-04 1.90351918e-01 2.14157227e-01
  1.78277398e-04]
 [2.05098492e-01 1.02303373e-04 3.30523059e-03 5.75466870e-02
```

```
   9.66286525e-05]
  [2.65530456e-05 1.51500862e-03 3.49461563e-05 1.27670929e-04
   3.18241393e-01]
  [4.77224248e-05 1.04196301e-04 3.11397970e-03 1.06284390e-03
   1.39271261e-04]]

Iteration 3
[(1, 0), (1, 1), (1, 2)]
----------
Actual Color: b
Sensor Color: y

Robot Location:
.....
.....
.A...
.....
.....

Distribution:
[[0.00160145 0.00204962 0.03422547 0.14217565 0.01148739]
 [0.26368958 0.03729382 0.01224898 0.00454442 0.00870012]
 [0.02134846 0.01572471 0.00533687 0.01225201 0.16754616]
 [0.01643221 0.01771647 0.02566819 0.16843228 0.01201207]
 [0.00274524 0.00237486 0.00254654 0.00256145 0.009286  ]]

Smooth Distribution:
[[3.58816580e-04 7.84778256e-04 1.22884819e-01 8.04315196e-02
  6.88730240e-03]
 [1.19452044e-01 1.34230275e-01 5.31280850e-03 1.29499558e-04
  4.64323542e-04]
 [4.43774520e-04 4.76308069e-03 5.98297203e-05 5.43382583e-03
  8.87494214e-02]
 [2.80351364e-04 9.67111734e-04 1.53326398e-01 2.72215269e-01
  3.86393011e-04]
 [1.38603805e-05 1.53843023e-04 6.49993939e-04 6.83030832e-04
  9.37629515e-04]]

Iteration 4
[(1, 0), (1, 1), (1, 2), (2, 2)]
----------
Actual Color: g
Sensor Color: g

Robot Location:
.....
.....
..A..
.....
```

```
.....

Distribution:
[[0.01534519 0.16512801 0.0084629  0.00853981 0.00771516]
 [0.01437555 0.01303443 0.00650233 0.0140092  0.01097092]
 [0.02533773 0.00466135 0.14475024 0.02762749 0.00889827]
 [0.00332317 0.00267156 0.00861078 0.00232957 0.02853947]
 [0.00138636 0.0014483  0.07282344 0.40161756 0.0018912 ]]

Smooth Distribution:
[[1.15239685e-03 2.91191475e-01 1.36362739e-04 1.50363099e-04
  6.66537447e-04]
 [4.49705109e-04 5.81068198e-04 1.57795850e-04 2.33775851e-03
  1.24801611e-03]
 [1.99776552e-03 3.22040918e-05 3.56332637e-01 5.81657821e-04
  8.56434857e-04]
 [3.93774281e-05 1.18061547e-04 2.90812708e-05 5.82921680e-05
  1.07471584e-03]
 [1.83136772e-05 8.04813783e-06 3.63647455e-02 3.04340560e-01
  7.66265380e-05]]

Iteration 5
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.....
.....
.A...
.....
.....

Distribution:
[[0.18056996 0.01380158 0.00572896 0.00117618 0.00106119]
 [0.00206804 0.00579074 0.00985422 0.0029324  0.03573152]
 [0.00260753 0.15960883 0.00323919 0.00929281 0.00230929]
 [0.02810755 0.000548   0.00675981 0.01416486 0.00227737]
 [0.00648094 0.06729016 0.03310812 0.03270916 0.37278158]]

Smooth Distribution:
[[2.77276337e-01 3.37294401e-03 1.36395151e-03 2.76239968e-04
  1.96978141e-04]
 [2.30812698e-04 4.18164754e-04 2.24755432e-03 1.09825915e-03
  1.59466674e-01]
 [3.02698869e-04 3.24256441e-01 3.44072655e-04 4.77304063e-03
  3.88530245e-04]
 [8.73652188e-03 1.13768647e-05 9.77837337e-04 5.28076739e-04
```

```
  1.14416487e-03]
 [6.03216056e-04 7.12335666e-03 1.12224458e-03 1.00576914e-02
  1.93682813e-01]]

Iteration 6
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
.....
.A...
.....
.....
.....

Distribution:
[[0.01980901 0.01893156 0.02747611 0.00979862 0.03509038]
 [0.17175306 0.16662518 0.00130136 0.00412335 0.00220859]
 [0.01415225 0.00064025 0.01705804 0.00166581 0.04460325]
 [0.00198316 0.23534363 0.04590616 0.04588726 0.02880096]
 [0.00569348 0.00564449 0.0128607  0.04163131 0.041012  ]]

Smooth Distribution:
[[7.59005294e-05 2.87688555e-04 4.18142228e-02 4.62469521e-02
  3.47133823e-03]
 [2.57128135e-01 2.49968989e-01 5.32225094e-04 9.69653663e-04
  6.17024316e-04]
 [1.75102217e-03 6.43799868e-05 4.95484077e-03 6.95353111e-04
  1.69672959e-01]
 [1.67173040e-04 2.27146622e-02 1.81245518e-02 1.67924037e-01
  6.03990749e-03]
 [1.77746230e-05 8.96647683e-06 4.57309628e-05 9.08752499e-04
  5.79775979e-03]]

Iteration 7
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3), (2, 3)]
----------
Actual Color: r
Sensor Color: r

Robot Location:
.....
..A..
.....
.....
.....
```

```
Distribution:
[[0.00933009 0.01650768 0.00181204 0.00241012 0.00258971]
 [0.01173226 0.00606957 0.26862523 0.01868472 0.00348509]
 [0.2352422  0.01754908 0.00351034 0.13934187 0.00243502]
 [0.01041865 0.001707   0.00980419 0.00371826 0.20002332]
 [0.00077033 0.01051466 0.00751806 0.01002415 0.00617636]]

Smooth Distribution:
[[5.49397096e-05 1.65390383e-04 7.91001755e-05 6.17614771e-05
  1.39826328e-05]
 [2.77857771e-04 2.64081274e-04 2.62273268e-01 5.97772158e-02
  1.12209721e-04]
 [2.27101913e-01 5.36582020e-04 6.35166347e-04 1.33706902e-01
  1.92767718e-04]
 [3.05546506e-04 6.26346836e-06 3.55813968e-05 2.96157417e-04
  3.13601630e-01]
 [1.66005061e-06 3.59456227e-05 7.22644792e-05 7.47586934e-05
  3.17055122e-04]]

Iteration 8
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3), (2, 3), (3, 3)]
----------
Actual Color: r
Sensor Color: r

Robot Location:
.....
...A.
.....
.....
.....

Distribution:
[[1.43756179e-03 1.80871539e-03 6.89825374e-03 6.07841165e-04
  2.64020260e-04]
 [6.25502520e-03 7.49566716e-03 2.83944224e-02 3.90714142e-01
  8.33554896e-04]
 [2.59564115e-01 7.55650116e-03 2.33506825e-02 2.67627590e-02
  8.23163610e-03]
 [7.60582153e-03 1.15115704e-03 3.92257009e-04 8.56320976e-03
  2.00475624e-01]
 [6.88861911e-04 6.28664746e-04 2.03087570e-03 1.47171765e-03
  6.81691204e-03]]

Smooth Distribution:
[[4.23730649e-05 1.65352116e-04 2.83314981e-04 3.52254691e-05
  1.51843814e-05]
 [2.51741728e-04 1.75136918e-04 4.61804966e-02 3.55559985e-01
  4.31325190e-05]
```

```
 [2.31691700e-01 3.99020857e-04 1.20737077e-03 4.35267703e-02
  1.89166716e-04]
 [2.20425198e-04 2.54420493e-05 1.62610818e-05 2.03844390e-04
  3.19508860e-01]
 [3.89330193e-06 1.85303006e-05 1.08523654e-04 7.86439947e-05
  4.96044730e-05]]

Iteration 9
[(1, 0), (1, 1), (1, 2), (2, 2), (1, 2), (1, 3), (2, 3), (3, 3), (3, 4)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
...A.
.....
.....
.....
.....

Distribution:
[[4.34670746e-04 1.22667549e-03 2.56397679e-02 2.70942768e-01
  1.33908577e-03]
 [1.86813321e-01 2.99270415e-02 2.38355529e-02 3.14863076e-03
  1.58962704e-02]
 [1.56312564e-02 1.15856003e-02 4.38830160e-03 2.39691185e-02
  1.60670700e-01]
 [1.06894789e-02 1.10035289e-02 2.38628926e-02 1.55774352e-01
  1.24661866e-02]
 [3.81954330e-04 1.78796153e-04 3.14559116e-04 1.31307845e-03
  8.56641282e-03]]

Smooth Distribution:
[[2.89996122e-05 1.43218630e-04 2.92700914e-02 3.09305436e-01
  1.52868634e-03]
 [2.13264137e-01 3.41643982e-02 2.22630696e-03 2.94090874e-04
  1.06053992e-03]
 [1.46000284e-03 7.72948072e-04 5.12349475e-04 2.23878236e-03
  1.83419994e-01]
 [7.13162196e-04 1.25615137e-02 2.72416292e-02 1.77830374e-01
  1.16437651e-03]
 [2.54825694e-05 1.19286130e-05 3.67258709e-05 1.53306476e-04
  5.71519138e-04]]

-----Time Analysis-----
Filtering Only: 0.0008471012115478516
Forward-Backward Smoothing: 0.00146484375
```

```
test_maze = Maze('maze4.maz')
test_problem = HMMProblem(test_maze, 15)
```

```
Iteration 0
[]
----------
Actual Color: 0
Sensor Color: 0

Robot Location:
.#...
.#.#.
.#.#.
...#.
A...#

Distribution:
[[0.04 0.   0.04 0.04 0.04]
 [0.04 0.   0.04 0.   0.04]
 [0.04 0.   0.04 0.   0.04]
 [0.04 0.04 0.04 0.   0.04]
 [0.04 0.04 0.04 0.04 0.  ]]

Smooth Distribution:
[[4.29888731e-01 0.00000000e+00 5.95667633e-08 7.68834831e-10
  2.39010482e-11]
 [1.46965053e-01 0.00000000e+00 3.21714447e-06 0.00000000e+00
  1.58242548e-12]
 [2.40920789e-02 0.00000000e+00 1.71087546e-04 0.00000000e+00
  1.55858556e-13]
 [6.54257148e-02 4.86149188e-02 1.35989201e-02 0.00000000e+00
  3.28442630e-14]
 [1.34215314e-01 8.34024259e-02 4.00432416e-02 1.35792366e-02
  0.00000000e+00]]

Iteration 1
[(0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
A...#
```

```
Distribution:
[[0.176      0.         0.00666667 0.01333333 0.01333333]
 [0.00666667 0.         0.01       0.         0.00666667]
 [0.01333333 0.         0.01       0.         0.01      ]
 [0.176      0.00666667 0.00666667 0.         0.00666667]
 [0.176      0.176      0.176      0.01       0.        ]]

Smooth Distribution:
[[5.78324458e-01 0.00000000e+00 1.11355015e-10 6.48053833e-12
  4.12749192e-13]
 [2.83647242e-04 0.00000000e+00 1.33793180e-08 0.00000000e+00
  1.65940059e-14]
 [1.81079788e-04 0.00000000e+00 7.13983666e-07 0.00000000e+00
  5.08914922e-15]
 [8.53832749e-02 9.09115830e-05 2.53067776e-05 0.00000000e+00
  5.49976528e-16]
 [1.75015492e-01 1.08598695e-01 5.20395718e-02 5.68361358e-05
  0.00000000e+00]]

Iteration 2
[(0, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
A...#

Distribution:
[[0.20662055 0.         0.00053673 0.00136623 0.00136623]
 [0.00296667 0.         0.0008051  0.         0.00053673]
 [0.00612851 0.         0.0008051  0.         0.00073191]
 [0.14375844 0.00534781 0.00291787 0.         0.00043914]
 [0.27205897 0.20662055 0.14247028 0.00452319 0.        ]]

Smooth Distribution:
[[5.78717058e-01 0.00000000e+00 7.95006108e-12 1.04549263e-12
  1.83503893e-13]
 [1.07582367e-04 0.00000000e+00 8.87291650e-10 0.00000000e+00
  8.34372262e-15]
 [6.92145068e-05 0.00000000e+00 4.69014120e-08 0.00000000e+00
  2.45618737e-15]
 [5.77252748e-02 6.02258616e-05 9.07339065e-06 0.00000000e+00
  3.27512376e-16]
```

```
 [2.23580135e-01 1.05098600e-01 3.46117423e-02 2.10464656e-05
  0.00000000e+00]]

Iteration 3
[(0, 0), (0, 0), (1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
.A..#

Distribution:
[[2.09157236e-01 0.00000000e+00 4.12750383e-05 1.17928681e-04
  1.17928681e-04]
 [2.78172655e-03 0.00000000e+00 5.63264620e-05 0.00000000e+00
  4.03440224e-05]
 [4.04463100e-03 0.00000000e+00 1.01760038e-04 0.00000000e+00
  4.65507951e-05]
 [1.43493116e-01 4.56210205e-03 1.92766222e-03 0.00000000e+00
  2.60684452e-05]
 [3.00388571e-01 2.10389454e-01 1.19729986e-01 2.97733299e-03
  0.00000000e+00]]

Smooth Distribution:
[[5.78408418e-01 0.00000000e+00 9.34837467e-13 3.78635715e-13
  1.15420637e-13]
 [9.95999478e-05 0.00000000e+00 6.27900257e-11 0.00000000e+00
  4.27708814e-15]
 [4.42663663e-05 0.00000000e+00 5.64257211e-09 0.00000000e+00
  1.40108869e-15]
 [5.53580190e-02 4.91981979e-05 5.66358035e-06 0.00000000e+00
  2.47417147e-16]
 [2.36406409e-01 1.02025391e-01 2.75899477e-02 1.30825441e-05
  0.00000000e+00]]

Iteration 4
[(0, 0), (0, 0), (1, 0), (1, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
```

```
.#.#.
...#.
.A..#

Distribution:
[[2.06406719e-01 0.00000000e+00 3.18572681e-06 9.80164406e-06
  9.77854513e-06]
 [2.71383325e-03 0.00000000e+00 4.75780118e-06 0.00000000e+00
  3.04135852e-06]
 [3.82984324e-03 0.00000000e+00 4.07048110e-05 0.00000000e+00
  2.96821193e-06]
 [1.48189039e-01 4.47050029e-03 1.56704689e-03 0.00000000e+00
  1.54762801e-06]
 [3.12649117e-01 2.07984172e-01 1.09719822e-01 2.39412151e-03
  0.00000000e+00]]

Smooth Distribution:
[[5.78377628e-01 0.00000000e+00 2.29462654e-13 2.48410588e-13
  6.52441986e-14]
 [9.84934438e-05 0.00000000e+00 7.87916969e-12 0.00000000e+00
  2.29202742e-15]
 [4.24780339e-05 0.00000000e+00 2.30036765e-09 0.00000000e+00
  1.08390040e-15]
 [5.66931930e-02 4.75188308e-05 4.48916614e-06 0.00000000e+00
  2.06240537e-16]
 [2.41832579e-01 9.83984554e-02 2.44949081e-02 1.02549679e-05
  0.00000000e+00]]

Iteration 5
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
A...#

Distribution:
[[2.02316344e-01 0.00000000e+00 2.57911453e-07 8.02597803e-07
  7.98470727e-07]
 [2.65742408e-03 0.00000000e+00 9.87109199e-07 0.00000000e+00
  2.32017605e-07]
 [3.90762958e-03 0.00000000e+00 3.05564688e-05 0.00000000e+00
  1.94541852e-07]
 [1.52611672e-01 4.46317683e-03 1.42686894e-03 0.00000000e+00
```

```
   9.37842578e-08]
  [3.19274581e-01 2.06509363e-01 1.04638307e-01 2.16071079e-03
   0.00000000e+00]]

 Smooth Distribution:
 [[5.78452019e-01 0.00000000e+00 1.60990138e-13 1.45953068e-13
   3.23830370e-14]
  [9.85916875e-05 0.00000000e+00 4.35871975e-12 0.00000000e+00
   1.80914761e-15]
  [4.78454141e-05 0.00000000e+00 2.44907965e-09 0.00000000e+00
   9.75318631e-16]
  [5.92902123e-02 4.85631807e-05 4.12804918e-06 0.00000000e+00
   1.81388620e-16]
  [2.44089198e-01 9.50807508e-02 2.28792454e-02 9.44433452e-06
   0.00000000e+00]]

 Iteration 6
 [(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0)]
 ----------
 Actual Color: b
 Sensor Color: b

 Robot Location:
 .#...
 .#.#.
 .#.#.
 ...#.
 .A..#

 Distribution:
 [[1.97799789e-01 0.00000000e+00 2.83362972e-08 6.54246635e-08
   6.46867144e-08]
  [2.59993458e-03 0.00000000e+00 6.04486287e-07 0.00000000e+00
   1.79079608e-08]
  [4.00880226e-03 0.00000000e+00 2.74504362e-05 0.00000000e+00
   1.31795362e-08]
  [1.55829624e-01 4.48619748e-03 1.35883299e-03 0.00000000e+00
   5.84902043e-09]
  [3.23715328e-01 2.06002089e-01 1.02122550e-01 2.04860178e-03
   0.00000000e+00]]

 Smooth Distribution:
 [[5.78718376e-01 0.00000000e+00 1.53110953e-13 5.31903634e-14
   2.40692624e-14]
  [1.00105379e-04 0.00000000e+00 2.36907493e-11 0.00000000e+00
   1.67416071e-15]
  [6.91156437e-05 0.00000000e+00 4.27786009e-09 0.00000000e+00
   9.97155491e-16]
  [6.62833371e-02 5.24114890e-05 5.43347445e-06 0.00000000e+00
```

```
  1.59442824e-16]
 [2.40722435e-01 9.16042337e-02 2.24325730e-02 1.19752466e-05
  0.00000000e+00]]

Iteration 7
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
A...#

Distribution:
[[1.93138660e-01 0.00000000e+00 8.91877265e-09 5.49604104e-09
  5.22190997e-09]
 [2.54101735e-03 0.00000000e+00 5.28210715e-07 0.00000000e+00
  1.39544235e-09]
 [4.08625936e-03 0.00000000e+00 2.60413870e-05 0.00000000e+00
  9.22757660e-10]
 [1.58153511e-01 4.51321160e-03 1.32563300e-03 0.00000000e+00
  3.77167277e-10]
 [3.27060083e-01 2.06206924e-01 1.00954627e-01 1.99348207e-03
  0.00000000e+00]]

Smooth Distribution:
[[5.80166054e-01 0.00000000e+00 1.48696994e-13 3.52748990e-14
  1.94092973e-14]
 [1.18744323e-04 0.00000000e+00 2.12279981e-10 0.00000000e+00
  1.89015910e-15]
 [2.03236794e-04 0.00000000e+00 3.42477942e-08 0.00000000e+00
  1.12142934e-15]
 [8.83128045e-02 9.73919063e-05 7.23748515e-06 0.00000000e+00
  1.27331576e-16]
 [2.24761352e-01 7.61870865e-02 3.01327344e-02 1.33236701e-05
  0.00000000e+00]]

Iteration 8
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1)]
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
```

```
.#.#.
.#.#.
A..#.
....#

Distribution:
[[1.88448612e-01 0.00000000e+00 6.76516703e-09 6.16550115e-10
  4.25264924e-10]
 [2.48146945e-03 0.00000000e+00 4.98731118e-07 0.00000000e+00
  1.09602264e-10]
 [4.14260025e-03 0.00000000e+00 2.53580287e-05 0.00000000e+00
  6.65691392e-11]
 [1.59905951e-01 4.54081378e-03 1.31023355e-03 0.00000000e+00
  2.51972662e-11]
 [3.29803159e-01 2.06834346e-01 1.00539474e-01 1.96747649e-03
  0.00000000e+00]]

Smooth Distribution:
[[5.83825264e-01 0.00000000e+00 9.11693236e-13 2.86510601e-14
  1.76450157e-14]
 [6.21510700e-04 0.00000000e+00 5.39902925e-10 0.00000000e+00
  2.89798896e-15]
 [1.32983603e-03 0.00000000e+00 5.11590263e-07 0.00000000e+00
  1.28141966e-15]
 [1.90858996e-01 1.72316484e-04 4.20766507e-05 0.00000000e+00
  7.61784970e-17]
 [7.31317287e-02 1.13041282e-01 3.69725261e-02 3.95166241e-06
  0.00000000e+00]]

Iteration 9
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
.#...
.#.#.
.#.#.
.A.#.
....#

Distribution:
[[3.57848522e-02 0.00000000e+00 5.92568366e-07 8.84760948e-10
  1.65605927e-10]
 [2.28249876e-01 0.00000000e+00 2.07670851e-06 0.00000000e+00
  8.21518906e-10]
 [1.79265023e-02 0.00000000e+00 1.07249218e-04 0.00000000e+00
  2.11070223e-11]
```

```
 [3.14090264e-02 4.30484137e-01 1.22950650e-01 0.00000000e+00
  1.64249732e-10]
 [6.46810423e-02 4.04414797e-02 1.95774649e-02 8.38504895e-03
  0.00000000e+00]]

Smooth Distribution:
[[7.55535251e-04 0.00000000e+00 2.28944291e-10 7.46208362e-14
  1.36070561e-14]
 [7.20112841e-01 0.00000000e+00 5.89521098e-09 0.00000000e+00
  2.63110260e-13]
 [9.64000823e-03 0.00000000e+00 1.47236139e-06 0.00000000e+00
  5.09863052e-17]
 [5.38286084e-03 2.20021906e-01 4.16257448e-02 0.00000000e+00
  4.40715240e-15]
 [1.27089557e-03 7.95521390e-04 3.38964533e-04 5.42431296e-05
  0.00000000e+00]]

Iteration 10
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1)]
----------
Actual Color: y
Sensor Color: r

Robot Location:
.#...
.#.#.
.#.#.
..A#.
....#

Distribution:
[[2.34734720e-02 0.00000000e+00 1.90173296e-07 1.52466870e-06
  5.22536852e-09]
 [2.97384484e-02 0.00000000e+00 9.79172212e-06 0.00000000e+00
  1.06649871e-10]
 [7.57872343e-01 0.00000000e+00 1.07684901e-02 0.00000000e+00
  8.98763560e-11]
 [3.80844854e-02 3.64457265e-02 3.34051621e-02 0.00000000e+00
  2.99509095e-11]
 [1.40735868e-02 3.88317249e-02 1.33840839e-02 3.91096485e-03
  0.00000000e+00]]

Smooth Distribution:
[[9.80857082e-04 0.00000000e+00 1.01101854e-12 2.19282232e-10
  6.98143589e-13]
 [1.56476842e-03 0.00000000e+00 2.27067070e-10 0.00000000e+00
  4.83430704e-16]
 [9.87957461e-01 0.00000000e+00 2.05587980e-04 0.00000000e+00
  4.12246420e-16]
```

```
 [2.28479769e-03 2.15180267e-03 1.81633324e-03 0.00000000e+00
  7.73766770e-17]
 [3.09910963e-04 2.39182494e-03 3.09976269e-04 2.66788418e-05
  0.00000000e+00]]

Iteration 11
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1), (1, 1)]
----------
Actual Color: y
Sensor Color: y

Robot Location:
.#...
.#.#.
.#.#.
.A.#.
....#

Distribution:
[[4.20371941e-03 0.00000000e+00 9.00016564e-06 2.26972088e-07
  1.07390390e-07]
 [6.46979018e-01 0.00000000e+00 5.65986270e-04 0.00000000e+00
  4.25399114e-09]
 [1.10771922e-01 0.00000000e+00 2.88295141e-03 0.00000000e+00
  1.65968999e-11]
 [3.55270420e-02 1.12931338e-01 7.23318567e-02 0.00000000e+00
  1.38294250e-10]
 [4.40956463e-03 4.31184627e-03 3.75770170e-03 1.31771574e-03
  0.00000000e+00]]

Smooth Distribution:
[[7.43040634e-04 0.00000000e+00 9.81554384e-10 3.50139395e-12
  1.64814332e-12]
 [7.27016554e-01 0.00000000e+00 1.07384513e-08 0.00000000e+00
  4.17359936e-13]
 [8.31981434e-03 0.00000000e+00 1.69321085e-06 0.00000000e+00
  6.20598292e-17]
 [4.73690499e-03 2.13133542e-01 4.36422301e-02 0.00000000e+00
  5.48437708e-15]
 [1.23000799e-03 7.77948577e-04 3.43527003e-04 5.47253983e-05
  0.00000000e+00]]

Iteration 12
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1), (1, 1),
----------
Actual Color: b
Sensor Color: b

Robot Location:
```

```
.#...
.#.#.
.#.#.
...#.
.A..#

Distribution:
[[4.98997807e-01 0.00000000e+00 1.67414155e-05 5.47994962e-07
  2.55618372e-08]
 [4.03748648e-02 0.00000000e+00 1.72966331e-04 0.00000000e+00
  3.32169731e-09]
 [5.18135001e-02 0.00000000e+00 3.38132109e-03 0.00000000e+00
  1.90226971e-10]
 [1.99450689e-01 6.45059899e-03 5.49925949e-03 0.00000000e+00
  1.23646222e-11]
 [3.68111062e-02 9.48763973e-02 6.18227247e-02 3.31446917e-04
  0.00000000e+00]]

Smooth Distribution:
[[6.02585726e-01 0.00000000e+00 3.12030521e-10 3.14247524e-11
  1.46238822e-12]
 [6.41048093e-04 0.00000000e+00 4.81464460e-09 0.00000000e+00
  5.91049760e-14]
 [1.16096814e-03 0.00000000e+00 8.05245910e-07 0.00000000e+00
  3.30341678e-15]
 [1.72921101e-01 1.68714552e-04 5.04176474e-05 0.00000000e+00
  1.11955079e-16]
 [6.73336949e-02 1.15447582e-01 3.96852945e-02 4.64254952e-06
  0.00000000e+00]]

Iteration 13
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1), (1, 1),
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
.A..#

Distribution:
[[5.84262419e-01 0.00000000e+00 2.97982314e-06 5.14291925e-07
  1.73448338e-08]
 [9.09162338e-03 0.00000000e+00 8.08449034e-05 0.00000000e+00
  4.66348133e-10]
 [9.88832776e-03 0.00000000e+00 2.68508814e-04 0.00000000e+00
```

```
   8.02083506e-11]
  [1.11931808e-01 4.40900319e-03 1.11066738e-03 0.00000000e+00
   3.27239225e-12]
  [1.39835685e-01 7.59932397e-02 6.17679391e-02 1.35642258e-03
   0.00000000e+00]]

 Smooth Distribution:
 [[5.99784223e-01 0.00000000e+00 6.03659674e-10 2.97675669e-10
   1.00393080e-11]
  [1.24128088e-04 0.00000000e+00 2.23500918e-08 0.00000000e+00
   9.23429259e-14]
  [1.96298035e-04 0.00000000e+00 1.34106553e-07 0.00000000e+00
   1.83257130e-14]
  [7.88313161e-02 9.87232858e-05 1.05082197e-05 0.00000000e+00
   4.18692014e-16]
  [2.06165319e-01 7.84717103e-02 3.62977942e-02 1.98215044e-05
   0.00000000e+00]]

 Iteration 14
 [(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1), (1, 1),
 ----------
 Actual Color: b
 Sensor Color: b

 Robot Location:
 .#...
 .#.#.
 .#.#.
 ...#.
 .A..#

 Distribution:
 [[5.94147637e-01 0.00000000e+00 1.11537874e-06 1.02846944e-07
   1.40368915e-08]
  [7.82172904e-03 0.00000000e+00 8.29989300e-06 0.00000000e+00
   2.34494994e-10]
  [3.59705696e-03 0.00000000e+00 3.31194072e-05 0.00000000e+00
   1.20717954e-11]
  [8.97234128e-02 2.47099162e-03 8.62936992e-04 0.00000000e+00
   1.14995295e-12]
  [1.57684679e-01 9.50991134e-02 4.72883212e-02 1.26147037e-03
   0.00000000e+00]]

 Smooth Distribution:
 [[5.98802166e-01 0.00000000e+00 2.92008670e-09 6.85377938e-10
   9.35426507e-11]
  [1.13184523e-04 0.00000000e+00 3.25939605e-08 0.00000000e+00
   6.13913182e-13]
  [1.07527030e-04 0.00000000e+00 1.30061032e-07 0.00000000e+00
```

```
   4.30966809e-14]
  [6.29150583e-02 6.44558775e-05 1.22817702e-05 0.00000000e+00
   2.46322043e-15]
  [2.09251239e-01 9.58441161e-02 3.28619740e-02 2.78315559e-05
   0.00000000e+00]]

Iteration 15
[(0, 0), (0, 0), (1, 0), (1, 0), (0, 0), (1, 0), (0, 0), (0, 1), (1, 1), (2, 1), (1, 1),
----------
Actual Color: b
Sensor Color: b

Robot Location:
.#...
.#.#.
.#.#.
...#.
..A.#

Distribution:
[[5.90036093e-01 0.00000000e+00 1.32749700e-07 3.33352953e-08
  3.27471116e-09]
 [7.65760218e-03 0.00000000e+00 9.51936212e-07 0.00000000e+00
  1.81243655e-10]
 [2.61515176e-03 0.00000000e+00 1.75553178e-05 0.00000000e+00
  4.86484113e-12]
 [8.35407617e-02 2.34896497e-03 6.32386956e-04 0.00000000e+00
  1.93773964e-13]
 [1.64853429e-01 9.97122704e-02 4.76282672e-02 9.56396041e-04
  0.00000000e+00]]

Smooth Distribution:
[[5.98154648e-01 0.00000000e+00 5.09758557e-09 2.56014922e-09
  2.51497674e-10]
 [2.94051756e-04 0.00000000e+00 5.48314946e-08 0.00000000e+00
  6.95975240e-12]
 [2.00843541e-04 0.00000000e+00 1.01118573e-06 0.00000000e+00
  2.80214690e-13]
 [8.46902344e-02 9.02002033e-05 2.42836453e-05 0.00000000e+00
  7.44091599e-15]
 [1.67121717e-01 1.01084254e-01 4.82836046e-02 5.50883806e-05
  0.00000000e+00]]

-----Time Analysis-----
Filtering Only: 0.0013239383697509766
Forward-Backward Smoothing: 0.002496004104614258
```

State Estimation / Localization - Discussion - Bonus #4

Prior to applying state estimation to the Mazeworld problem, it is helpful to consider a more simplistic case, particularly one in which the robot is in a 1D world/maze. The robot is able to move left or right without concern of any walls, except for the boundaries of the corridor.

Further, we may consider the case where the robot is located in a continuous space, with sonar sensors that detect the distance from the walls. In this case, a particle filter may be used to reduce the state space of the robot in the process of state estimation.

The analysis of each of these problems in turn is presented in `state_estimation.pdf`, which is located in the `Introduction` folder. This provides a (relatively) lengthy discussion surrounding the presented state estimation problems. The `state_estimation.py` file is used to mathematically compute the appropriate values to answer the first proposed question, and thus is relatively specific/simplistic.

The main idea behind the discussion of these problems was to consider the widespread applications of state estimation for various problems, particularly in the field of robotics. The first problem proposes a scenario in which there is not sensor information, but the location of the robot is unknown (after many steps) due to uncertainty/error in the motion of the robot. This certainly has applications in the real world, and is presented here primarily to provide an overview of the way in which a transition model is applied to similar problems. This gives background as to the idea of a discrete Bayesian filter.

The second problem proposes a scenario in which there is sensor information, though the robot is unaware of it's location in a continuous space, specifically a bounded rectangle. The objective is to minimize the state space of the robot through various movements, though the results highlight an issue with symmetric spaces. This situation uses a particle filter in an attempt to localize the robot, though we are unsuccessful in doing so completely. The implications of this are discussed at length.

As indicated, the goal of this extension is to provide additional background to the problem of state estimation and localization, while considering other situations that are not represented by the Mazeworld problem. While the topics presented do not directly relate to the assignment, they are tangential, and thus relevant to understanding state estimation at a deeper level.

Please view the `state_estimation.pdf` file.

Literature Review - Bonus #5

The state estimation problem from this assignment is a well-known one, and it has been studied extensively in the AI research community (alongside problems involving path planning and mapping). While there is not required additional part of the assignment, I found it relevant to complete a brief review of a somewhat-related paper on state estimation.

Thus, a paper was chosen and read through enough to get a sense of the approach and major findings. In this report, we briefly discuss the paper. The discussion should describe the problem attacked by the paper, give a quick summary of the main result(s), and discuss the basic approach of the paper.

Paper: *Probabilistic Modeling and Bayesian Filtering for Improved State Estimation for Soft Robots* by DongWook Kim, Myungsun Park, Yong-Lae Park

This paper provides an overview of an optimized method for state estimation of "soft" robots using probabilisitic modeling and Bayesian filtering, focusing primarily on reducing uncertainty. This uncertainty is derived from the idea that "soft" robots interact with the world in a different manner to rigid bodies, resulting in stochastic noise/error in the robot's "soft" components and sensor measurements.

Using Bayesian filtering techniques, such as Kalman or particle filters, is demonstrated as an effective way to fuse sensor data for state estimation. The filters leverage probabilistic models to handle uncertainty and provide robust estimates in the face of error. By improving the state estimation via appropriate modeling and inclusion of uncertainty, the Bayesian filtering framework handles the challenges presented by "soft" robot components and sensor measurements.

Abstract "State estimation is one of the key requirements in robot control, which has been achieved by kinematic and dynamic models combined with motion sensors in traditional robotics. However, it is challenging to acquire accurate proprioceptive information in soft robots due to relatively high noise levels and hysteretic responses of soft actuators and sensors. In this article, we propose a method of estimating real-time states of soft robots by filtering noisy output signals and including hysteresis in the models using a Bayesian network. This approach is useful in constructing a state observer for soft robot control when both the kinematic model of the actuator and the model of the sensor are used."

"In our method, we regard a hysteresis function as a conditional random process model. We then introduce a dynamic Bayesian network composed of the actuator and the sensor models of the target system using distribution hysteresis mapping. Finally, we show that solving a Bayesian filtering problem is equivalent to suboptimal state estimation of the soft system."

The article describes two ways for defining modeling and filtering, one is by Gaussian process regression combined with an extended Kalman filter, and the other is based on variational inference with a particle filter. As the textbook addresses, Kalman filters and particle filters are valid approaches to the problem of state estimation, particularly in the field of robotics. The first approach "relaxes the uncertainty level in modeling to Gaussian", which indicates the assumption that the paper makes about the nature of the error. The second approach "illustrates a general probability distribution", which reflects the approach of particle filters.

The results of the proposed methods are experimentally validated via "real-time state estimation of a sensor-integrated soft robotic gripper" and show improvement in state estimation compared to

conventional methods.

The key idea is that an optimized filtering algorithm removes the uncertainties when applying state estimation to "soft" robots, which creates a better way to represent the robots in the context of the world. This is used to solve problems involving the physical interaction of "soft" robots with the environment.

In summary, the paper argues that using Bayesian filtering techniques, such as Kalman or particle filters, is an effective way to improve state estimation for "soft" robotic components and sensors by reducing uncertainty through probabilitistic modeling. The focus on "soft" robots, which have compliant bodies that make modeling and state estimation more challenging compared to rigid robots, is what makes this paper particularly insightful. By properly capturing uncertainty in the state estimation of "soft" components, the methods of probabilistic modeling and Bayesian filtering may be tailored to the presented challenges.