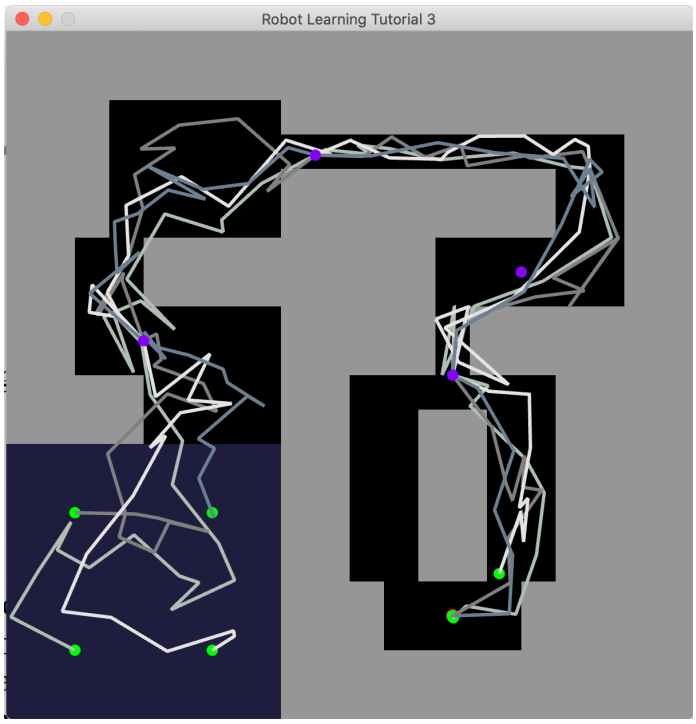
Benjamin Leroy

02107259

Coursework 2

Section 1: Demonstrator

Figure 1



With:
The green points:
- In the blue region: the initial state
- In the black region: the final state

The red point: the goal state

The purple points: the waypoints

The grey lines: the paths

The grey region: the forbidden space

Parameters:
For each waypoint:
- 100 sequences
- 15 actions per sequence

Refitted 50 times to the top 10%

Question 1a What reward function did you design? Explain how this is a better reward function than just using the negative distance between the robot's final state and the goal state

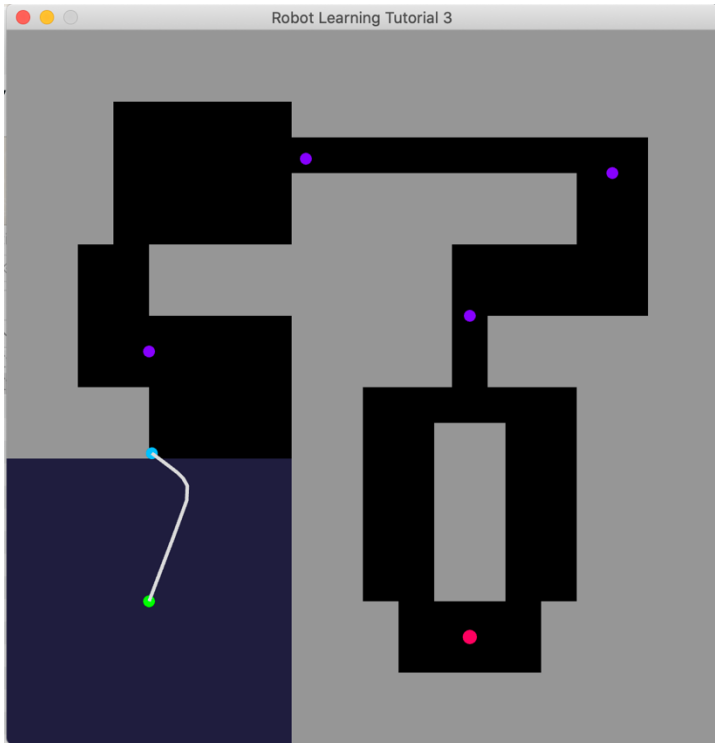
To design the reward function, we couldn't just use the negative distance between the robot's final state and the goal state because it wouldn't push the agent to go all around the maze. Therefore, we created different waypoints to indicate to the agent the right path to follow. Then we used the negative distance for each waypoint as a reward. For each waypoint, we found the best path and then we concatenated all the paths to get the demonstration to reach the goal.

Question 1b : In Lecture 5, one of the motivations I gave for using imitation learning, was that it is difficult to hand-design a good reward function for Reinforcement Learning. However, in this tutorial, you have actually been asked to hand-design a good reward function. So, explain what this reward function is really modelling, given that it is not being used for Reinforcement Learning.

This reward function is just modelling the score of each path by computing the distance of the path's final state to the waypoint (considering that we are doing what's explained previously). However, it's different from reinforcement learning because the agent doesn't receive a reward, which would then be used to calculate a policy. Furthermore, this reward function doesn't associate a reward to each (state, action) pair. Here we are doing supervised learning and the reward function allows to find the paths that will be used to execute the cross-entropy method.

Section 2: Single-Demonstration Behavioural Cloning

Figure 2



With:

The green point: the initial state

The blue point: the final state

The red point: the goal state

The purple points: the waypoints

The grey line: the path

The grey region: the forbidden space

Parameters:
For each waypoint :100 samples of 15 actions each

Refitted 50 times to the top 10%

Trained for 500 epochs

Learning rate: 0.001

Structure:
Input layer (2,100) with ReLU function

2 hidden layers (100,100) with ReLU function

Output layer (100,2) with linear function

Question 2a: Does the robot reach the goal when using this policy? Explain why this is the case, and explain the shape of the path drawn in the above figure.

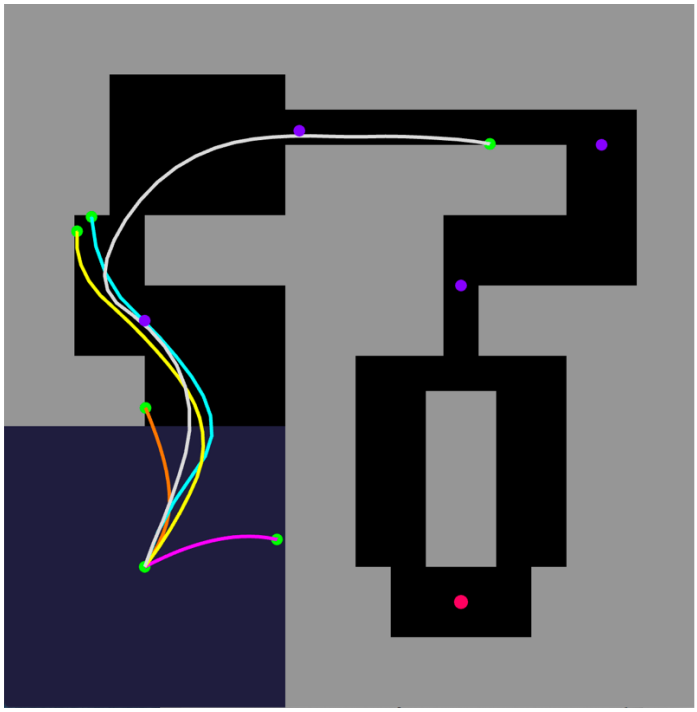
The robot doesn't reach the goal but that's normal because we trained a model using a single good demonstration. Therefore, if the prediction isn't exactly what expected then the robot will be in a new state for which he hasn't been trained on. We can see here that the robot reaches a wall and stops there. The robot got lost in the environment and couldn't change his path because the demonstration is good and doesn't include such state. We are here with a state distribution problem because testing data are out of distribution. To avoid that problem, we need to diversify those demonstrations by using different initial states.

Question 2b: In this environment, the dynamics are non-linear. Explain if and how the environment's linear / non-linear nature, would affect the optimal policy's linear / non-linear nature

The environment has a non-linear nature because it contains walls that forces the agent to go all around the maze. Therefore, if we push the agent to reach its goal without crossing walls then he must learn a non-linear optimal policy.

Section 3: Multiple-Demonstration Behavioural Cloning

Figure 3



With:
The green points:
- the initial state (middle of the blue zone)
- The other ones: the final states

The red point: the goal state

The purple points: the waypoints

The grey line: the path for 100 demonstrations

The yellow line: the path for 20 demonstrations

The blue line: the path for 5 demonstrations

The pink line: the path for 2 demonstrations

The orange line: the path for 1 demonstration

Parameters:
For each waypoint :100 samples of 15 actions each

Refitted 50 times to the top 10%

Trained for 500 epochs

Learning rate: 0.001

Structure:
Input layer (2,100) with ReLU function

2 hidden layers (100,100) with ReLU function

Output layer (100,2) with linear function

Question 3a: Is there a trend between the number of demonstrations used to train the network, and the quality of the policy? Explain why this is the case.

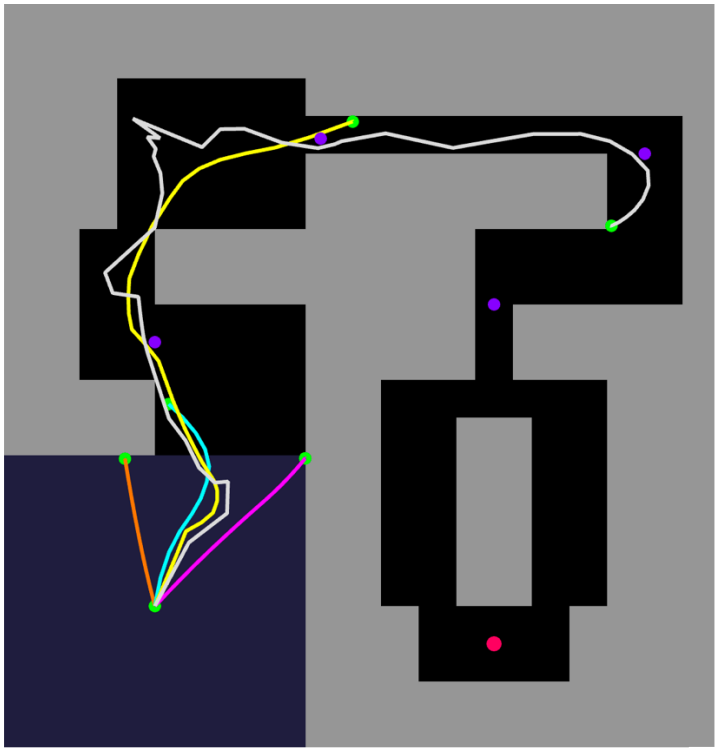
If the number of demonstrations increases, the dataset (used to train the agent) will contain more different states present in the environment. Therefore, the more demonstrations we get, the more different states the agent will be trained on. So, it allows to increase the size of the state distribution. Consequently, it reduces the chances for the the agent to get lost by discovering new states for which he doesn't know what action to execute to find the goal. To sum up, the more demonstrations used to train the network the better the policy will be.

Question 3b: about how to create the validation subset.

We must be sure that the network performs well everywhere in the environment. Therefore, we need a validation set which contains states distributed in diverse regions of the environment. By sampling 30% of all (state, action) pairs from the full dataset, we cannot assure that the states are distributed in different regions of the environment. On the contrary, by sampling 30% of the demonstrations (all demonstrations are independent and identically distributed), we are sure to have states from diverse regions because those demonstrations cross all the environment. For that reason, to create a validation set for early stopping we will prefer sampling 30% of the demonstrations.

Section 4: DAgger

Figure 4



With:

The green points:
- the initial state (middle of the blue zone)
- The other ones: the final states

The red point: the goal state

The purple points: the waypoints

The grey line: the path for 100 demonstrations

The yellow line: the path for 20 demonstrations

The blue line: the path for 5 demonstrations

The pink line: the path for 2 demonstrations

The orange line: the path for 1 demonstration

Parameters:
For each waypoint :100 samples of 15 actions each

Refitted 50 times to the top 10%

Trained for 100 epochs

Learning rate: 0.0005

Dataset initialized with 10 demonstrations

Structure:
Input layer (2,100) with ReLU function

2 hidden layers (100,100) with ReLU function

Output layer (100,2) with linear function

Question 4a: Why is there a difference in these paths as the number of DAgger iterations increases?

When the number of Dagger iterations increases, the agent gets better at finding the right path. This is coherent with the fact that at each iteration, the demonstrator indicates for each state of the path what action to take to find goal. Then this (state, action) pair is added to the dataset. Finally, the agent is trained on this new dataset which has the new indications. So, after each iteration, the agent gets better at finding the goal because each iteration allows to increase the size of the dataset and therefore the state distribution.

Question 4b : To make the agent reach any goal when he is trained to reach a fixed position, we need to first adapt the way we store the demonstrations in the dataset. A demonstration to reach the original goal is composed of (state, action) pairs. However, a new goal can be any state of this sequence. Therefore, we could consider any state of that sequence as the new goal and the previous (state, action) pairs as a way to reach it. So, the idea is to store in the dataset the vectors (state, action, new goal) for each (state, action) pair and for each new goal of the demonstration. Then we train the neural network to predict the action given the initial state and the goal state as inputs.