Project 1 consists of using Hopfield neuronal space to find an optimum path from an origin to a destination on considering energy and time consumption.

## Introduction of Hopfield neuronal space

Comparing with deep learning and neural network that are in plain development at present, Hopfield neuronal space has a very distinctive specialty – it does not require learning. Due to this, the run time performance is usually not as good as other neural networks.

Firstly, let us introduce the basic concepts of Hopfield neuronal space that are used in this project.

* Work space: The real space where the robot is working. The dimension of this space is usually two or three.
* Configuration space: A virtual finite space where every point is a configuration of the robot. Like the position or arm joint angle. The dimension of this space equals the freedom degree of the robot. (Note that the space can be continue or discrete, finite does not mean that there are countable number of points)
* Obstacles: In the work space and the configuration space, they are places that the robot cannot reach.
* Neuronal space: It is a discrete topologically ordered representation of the configuration space. Each point (which will be called “neuron”) represents a configuration of the robot and the robot can go from every point to its adjacent point directly. Thus a path in the neuronal space will represent a feasible path for the robot.
* Neuron: Points of the neuronal space, each neuron is given a value between 0 and 1. The value will update in the finding process and finally indicate a path.

## Finding process

Before the finding process could begin, the neuronal space should be generated.

All obstacles points are set to 0 and will not be updated. All target points are set to 1 and will not be updated. Other points which represent feasible configurations will be set to 0 initially and updated in the finding process.

The finding process is in fact the process of updating values of neurons step by step. Assuming neurons exist in the neuronal space. Their value could be changed due to inputs from adjacent neurons and sensory input (like obstacles that appear suddenly, but this is not considered in this project). That is to say

Where g is a sigmoid function, T is a symmetric matrix storing the diffusion factors (discussed in detail in next chapter). The detailed study of g will not be presented in this report, we choose where n is the dimension of neuronal space. (is the number of neurons that got summed).

Step by step, the value of all neurons are updated (except for obstacles and destination), when the origin neuron (the neuron which represents the origin point of the robot) has a value greater than zero (in Python we could chose 1e-15 as zero), the path is found. From the origin neuron we go, step by step, to the next adjacent neuron whose value is the biggest among all adjacent neurons. This will give a set of neurons from the origin neuron to the destination neuron which represent a feasible path. It is what we want from this method.

## Diffusion factor

The symmetric matrix T mentioned above is what we called *diffusion factor* here. It controls the influence of one neuron to another. Thus the difficulty of going from one neuron to another. Which could be interpreted as the time or energy consumption in our case.

Since T is symmetric, the difficulty level of going form neuron 1 to neuron 2 equals that of going from neuron 2 to neuron 1. This is not always correct like climbing and descending. But this problem is not yet considered in this project.

## Test environment

Due to lack of real robots and environments, we could not simulate diffusion factors with real interpretations. But it is possible to simulate its capacity.

The test environment is a 100x100 2D space. Without any floating objects in the space, each point is dedicated a value indicating the height of the obstacle of the point (0 means plain ground with no obstacles). For simplicity, only square obstacles are considered. (see figure below). Green color represents obstacles that are easy to pass. Purple color represents obstacles that are difficult to pass and red color represents obstacles that are impossible to pass.

### Group 1

Figure Env2

Figure Env1

The choice of these two environment is to test if the method gives a route that bypasses too much obstacles (in Env1). For green obstacles, the diffusion factor is set to 0.6. The results are as following:

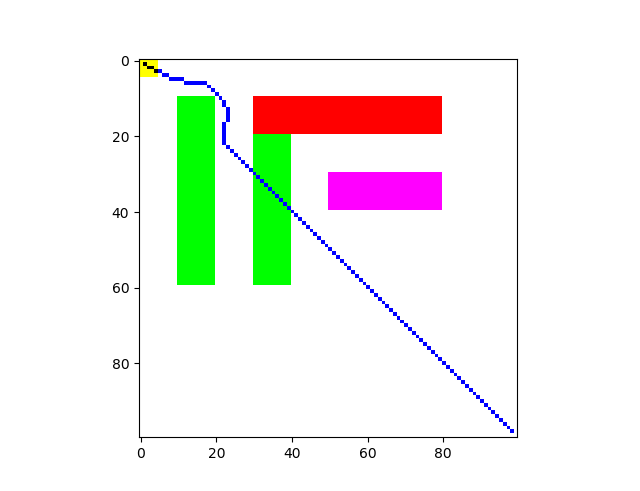
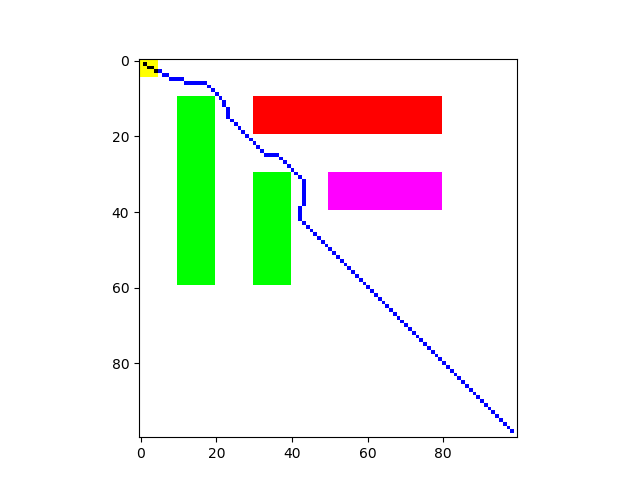


Figure Env2r

Figure Env1r

We observe that the robot will choose to bypass green obstacles if the additional distance is not too big (Env2). If the additional distance is too big the robot will choose to bypass it as the additional distance consumes more energy (time) than bypassing the obstacle.

### Group 2

Figure Env4

Figure Env3

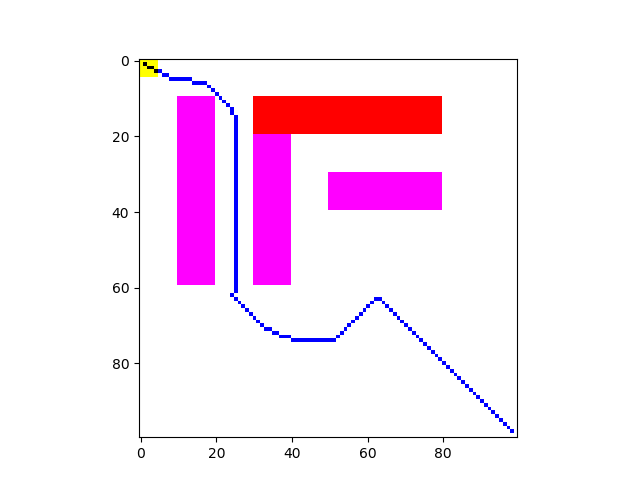
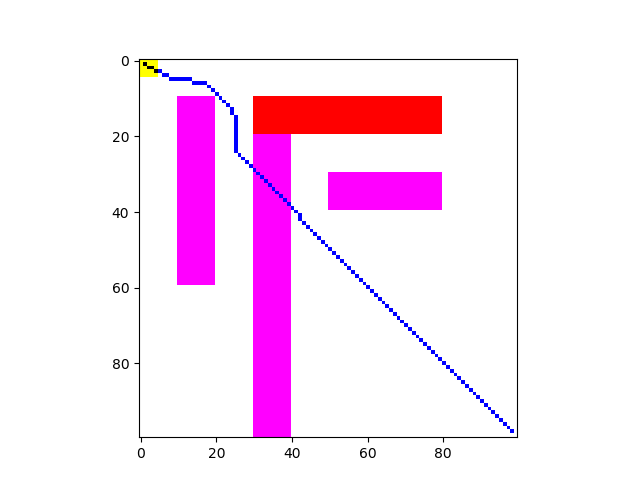
The choice of these two environment is to test if the method gives a route that bypasses too much obstacles (in Env3). For purple obstacles, the diffusion factor is set to 0.2. The results are as following:

Figure Env3r

Figure Env4r

The robot will choose to pass purple obstacles only if the bypassing distance is too long. Additionally, in Env3 we observe that the robot has an unpredicted behavior. The detailed description of this behavior is in the report “*Study of the influence of minimum target value*”.

## Further considerations

Whether the path found is optimum in the sense of energy consumption is not verified. But it is certain that the path found is better than no consideration. (The unpredicted behavior in Env3 is due to lack of iteration steps, it’s not a problem of the method)

## Simulation details

Simulations are done with Anaconda 3.7 (mainly Python 3.7 + Numpy + Numba). In this simulation, the environment is not very big so it runs rather fast (<5s).