Project 2 consists of using Hopfield neuronal space to find an optimum configuration sequence for a robot with two-joint arm and XY dimension translation.

## Introduction of Hopfield neuronal space and finding process

The terms are identical to project 1.

In this project, comparing to the first one, some additional steps must be taken. One is to calculate the robot’s “*presence*” which means the space occupied in the work space by the robot’s arms in each configuration. It is necessary for calculating the neuronal space. Obstacles includes, in addition to all points of the obstacles in XY dimension, all configurations that the robot’s presence has overlapped space with any obstacles.

These calculations consume lots of computational resources. So various approaches that can reduce this consumption by thousands of times are proposed in the following chapters.

## Numba

Numba is a library of Python which rewrites Python code and run it in C for maximum efficiency. But the code that it can transform is very limited. However, our second method of vectorization is supported.

## Robot presence calculation

The robot arms’ special presence is calculated using two methods. One is to consider the distance between any point to the arm’s joint and to the arm’s central axis. The other is to use the equation of robots outer lines.

It is tested that the first one is faster and the result of second one is better. As we use a pre-calculation approach (introduced in next paragraph) to reduce the calculation time in this step there is no need to consider the computational resources consumed in this part. So we choose the second method.

The *pre-calculation* approach is what we use to reduce this calculation by thousands of times. The main idea is that we only need to calculate the robot with different joint configurations (the angle of two joints) in one single point. The presence of the robot in others points can be calculated directly by translating.

Numba is also applied in this step.

Using the methods mentioned above, the preparation of robot presences takes maximum 5s.

## Preparation of neuronal space

This preparation consists of calculating all the feasible configurations of the robot using the presences calculated. All neurons of configuration that has overlap with any obstacles are fixed to 0.

This takes fairly amount of time. So many computational approaches that accelerates the calculation are applied. The most powerful ones are the usage of Numba and the vectorization.

Additionally, GPU accelerations are also expected to be effective. But due to limitations of environment (Coro-virus makes me unable to reach my computer with NVidia graphic card). It is not yet applied.

## Results

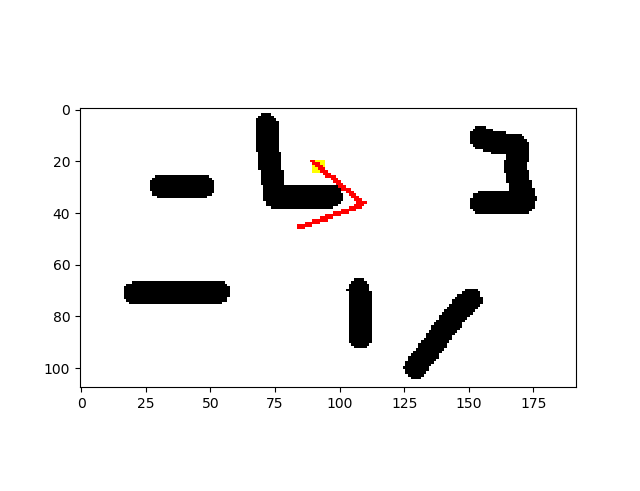
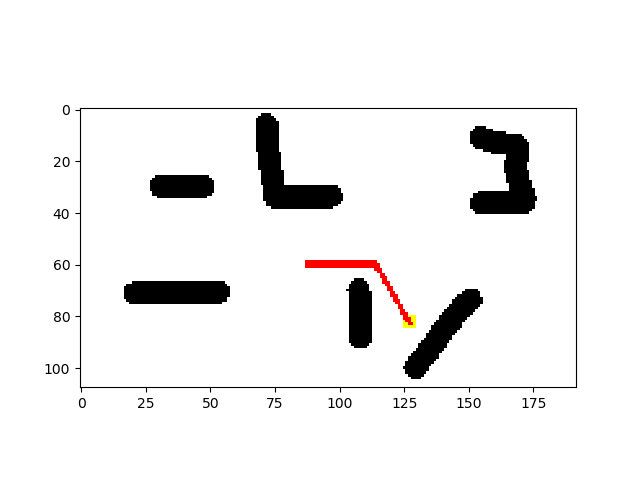
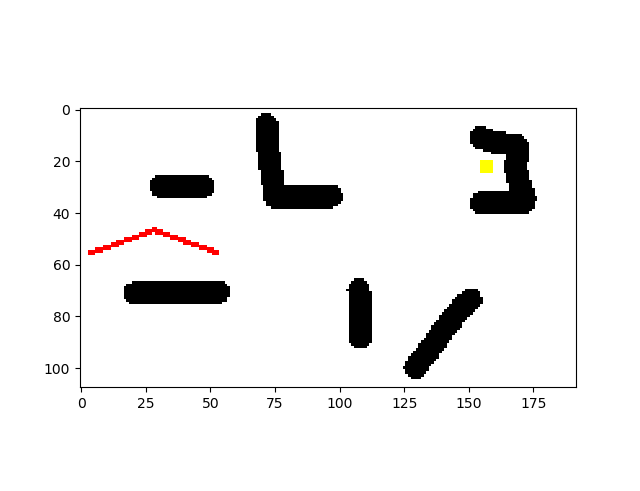
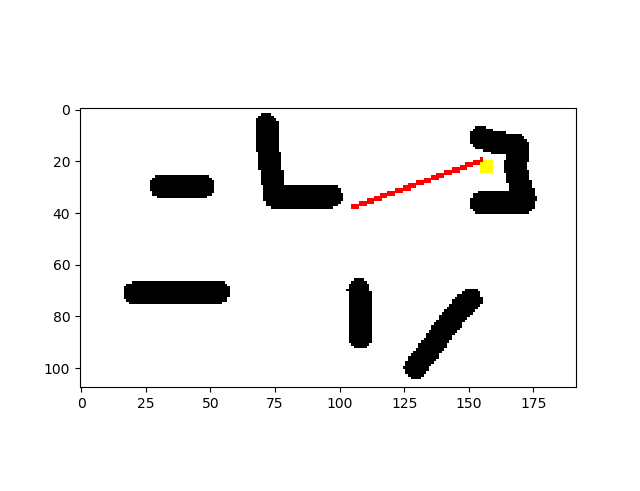
Red points are the robot. Yellow points are the target. Black points are obstacles.

Figure end2

Figure end3

Figure start

Figure end1

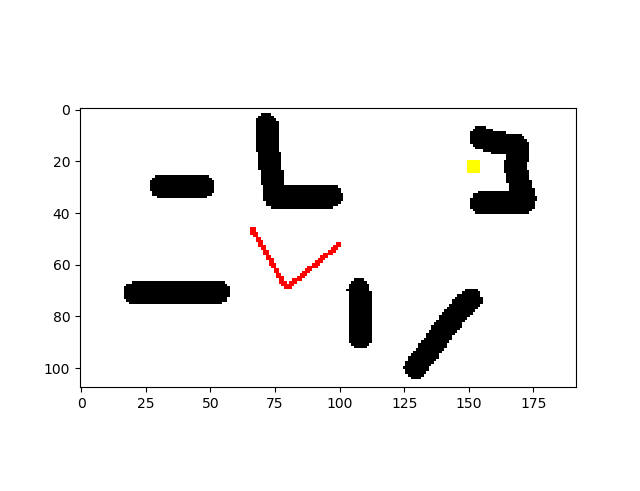
Some unpredicted behaviors are also observed. The robot tends to get away from the obstacles in the route.

Figure keep\_distance

Also, a big concern of this project is that in some extreme conditions, the method does not converge. Like when the target is behind a big obstacle.

The requirements of convergence are quite subtle. Even the overlapped space size limit could effect a lot. This is the major problematic for next stage of research.

## Simulation details

Simulations are done with Anaconda 3.7 (mainly Python 3.7 + Numpy + Numba). In this simulation, the environment is rather big (the size of neuronal space is bigger than 10e6), but by Numba the calculation time is about 1.5 minutes. With GPU acceleration, we should be able to support a neuronal space of size 10e8 ~ 10e9.

## Further work

In fact, if Numba starts to support np.roll() function with axis parameter, we will be able to accelerate much further the finding process. Unfortunately, it is still not the case. If at last we choose to apply the method in a 3D complicated space, I’ll implement this method.