ASSIGNMENT REPORT

Yi Liu 44028156

Hengchen Guo 44380881

1. [5 points] Please define the Configuration Space of the problem and describe which method for searching in continuous space do you use.

The Configuration space (C-space) of the problem is based on the configuration of ASVs. The brooms & ASVs is represented by the coordinates of ASVs in workspace. In the C-space it is the coordinates of start point (ASV) and the angle of the following points (ASVs) with the previous point (ASV). The expression looks like this

$$[x, y, \theta_1, \theta_2, \theta_3, \dots \theta_{n-1}]$$

Thus the θ_1 is angle between the first broom and the x axis. The number of variable in this configuration is n-1+2=n+1, where n is the number of ASVs. The obstacle is hard to illustrate in the C-space so we use collision-free check to check whether there is an obstacle or not.

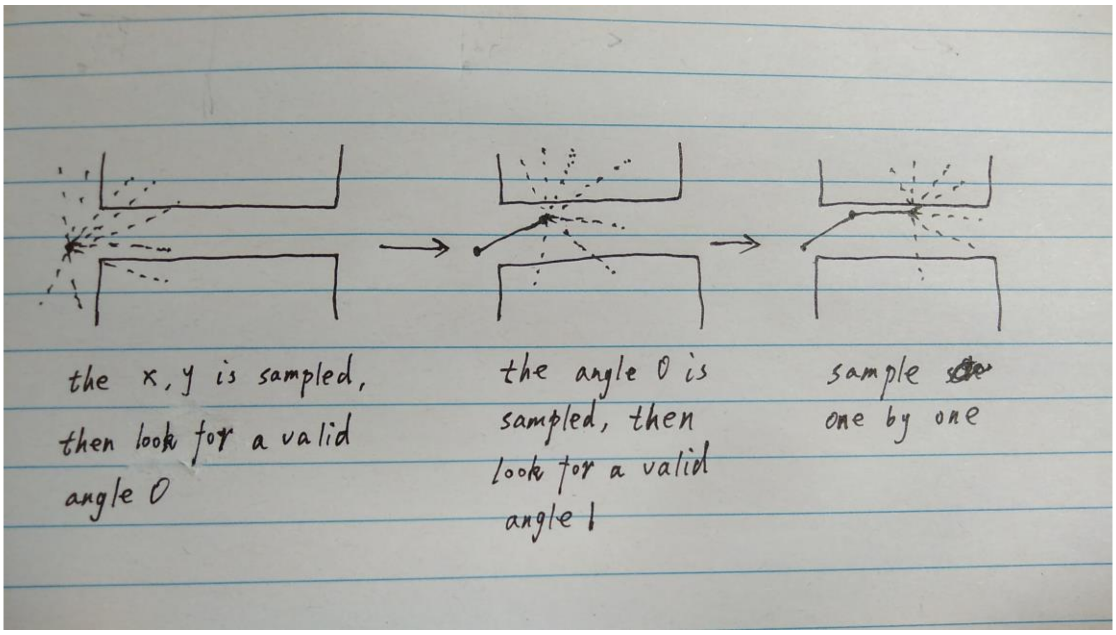
The search method we use is RRT (Rapidly-Exploring Random Trees) -connect. It is a bidirectional RRT. The basic idea is that given a random sample q_{rand}, extend from both q_{init} and q_{goal} until they reached the sample or has collision with the obstacle. The extend process for each branch would be like follow. After randomly generated a configuration q_{rand} which is collision free, extend the tree by 0.001 (primitive step) every time from both initial and goal until they meet each other or have collision. If they have collision, resample and redo the process, until the two trees can be connected.

2. [10 points] If you use sampling-based method, please describe the strategy you apply or develop for each of its four components. Otherwise, please describe the details of your discretization method.

In this assignment, we use RRT algorithm, which is a kind of sampling-based method. As illustrated in question 1, a c-space state contains the coordinates of the first ASV, the angle between the first line segment and x axis, and the angles between any two neighbour line segments, so it has this format: [x, y, angle0, angle1, angle2, … ...]

(1) sampling strategy.

We mix two different sampling strategies together to improve the efficiency. The one is generally sampling and the other one is corridor-specific sampling. In the first strategy, each dimension of a c-space state is generated step by step. Firstly, we randomly select two numbers as the “x” and “y” the state. The “Math.Random.nextDouble()” method, which generates a number between 0 and 1 uniformly. Then, we generate following parameters (angles) one by one, that is, for example, after generating a valid parameter at the nth dimension, then generate the parameter for the next dimension. The following photo illustrates


the x, y is sampled, then look for a valid angle 0
the angle 0 is sampled, then look for a valid angle 1
sample one by one

this procedure.

This method is much more efficient than generating all parameters in one time. In our experiment, if generating all parameters in one time, only several valid c-space states are found after 1 million experiments. However, if we sample step by step, nearly one of ten experiments can generate a valid c-space state.

The second sample strategy is similar as the first one. The only difference is that, in this strategy, we will restrict the first ASV located in the corridor area

(2) checking the configuration is valid.

The orientation can be indicated by the sign (positive or negative) of the angle1, angle2, … parameters of a c-space state. First, we will transfer the initial and goal ASVs from ASVConfig to our C-space config. If they are clockwise, angle1, angle2, … … will be negative, otherwise positive. Then, in the sampling stage, when sampling these parameters, we will only sample positive or negative number, depending on the orientation, so that the newly sampled states will have same orientation as the initial and goal state.

Except orientation, we also check the following aspects.

Whether it has enough area

whether it is convex

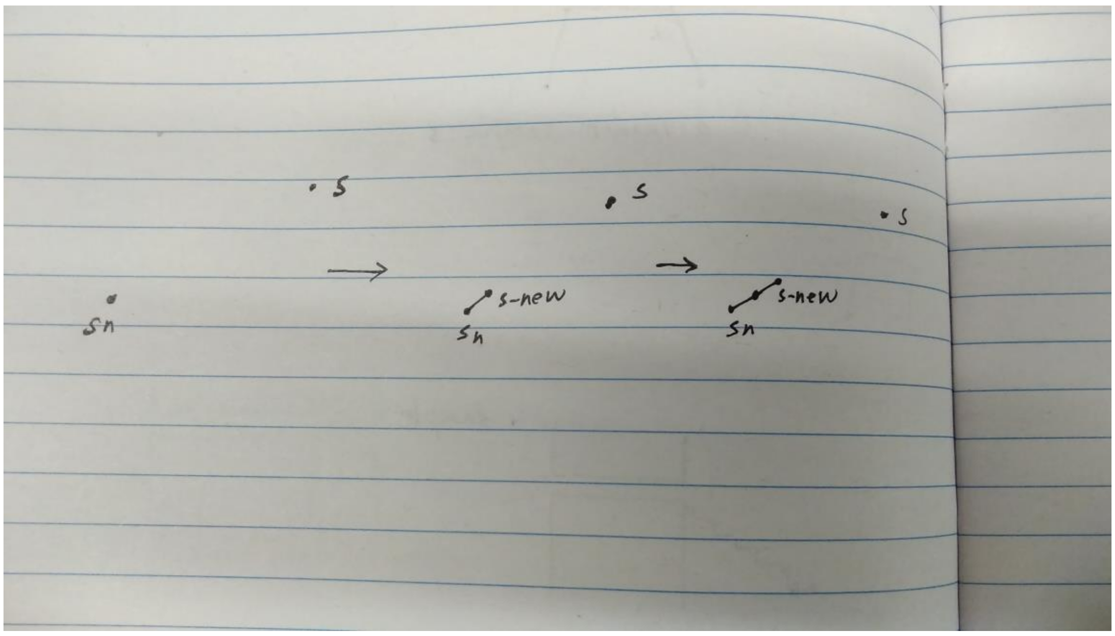
whether it fits the bounds

whether it has collisions with any obstacles

The methods we used for checking the above aspects are similar as the methods in the supporting Java code. After each sampling, we will check whether it is valid, if not, we will sample again until obtain a valid sample.

(3) connection.

As we are using bidirectional-RRT, the strategy used to ensure valid connections is extending the tree step by step towards the target. For example, we have sampled a state in c-space, say ***s***, , then implement same operations on each side. First, find out the nearest state, say ***sn***, in the existing tree. Originally, we assume the ASVs can move from ***sn*** to **s** straightly, then we compute the maximum distance between the two states, if it is larger than step size (0.001), we will find a new state, say ***s-new***, between ***sn*** and ***s***, and check whether the ASVs can move from ***sn*** to ***s-new*** within one step size. Repeat the above operations until the maximum distance is not larger than 0.001. Now, one edge is added into our tree. Then, we will continue to add edges towards the sample ***s*** until encounter ***s*** or collisions. The photo below shows this procedure.



(4) checking validity of line segment.

As shown in previous part, each time our tree will only extend no more than one step size, and the validity of line segments will be checked each time. After extending to a new state, we will compute the corresponding coordinates of ASVs, then every two neighbour ASVs will form a Line2D object in Java, and every obstacle will be converted to a Rectangle2D object in Java. Next, we call the intersects method of Line2D to check whether there are any Line2D objects has collisions with any Rectangle2D objects.

3. [12.5 points] Which class of scenarios do you think your program will be able to solve? Please explain your answer.

Our program can solve most of the scenarios even with long narrow passageways. It is efficient in terms of find a path comparing to other sampling-based algorithms, even for those scenarios with a lot of obstacles. We will define three scenarios, which is "simple", "within long narrow passageways" and "within multiple obstacles".

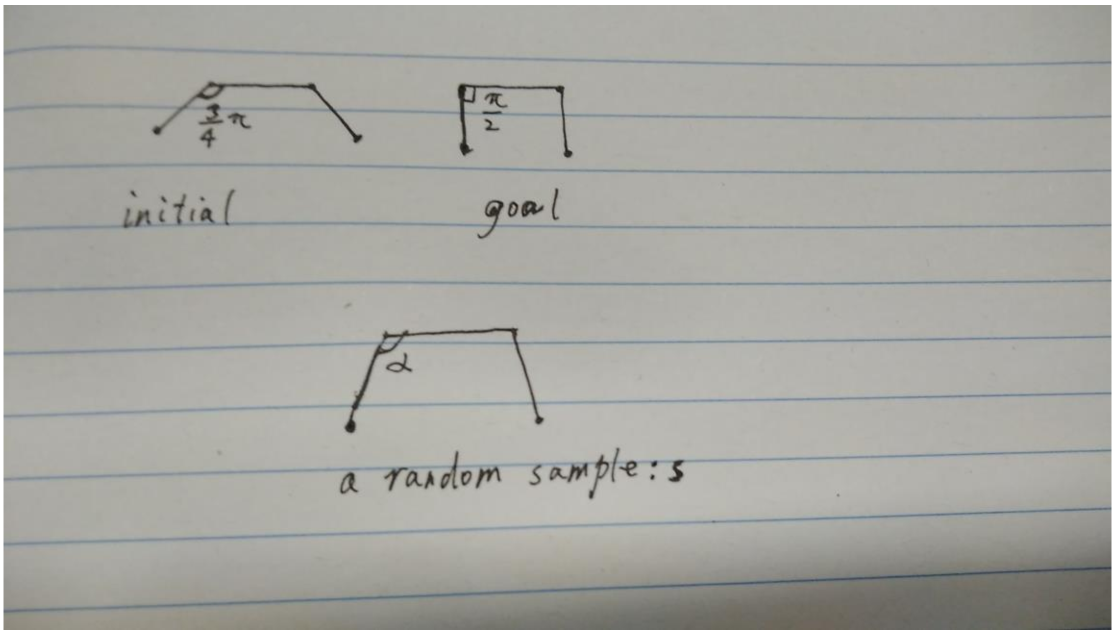
There are 3 main strategies that we used to improve the overall efficiency.

(1) Sampling strategies

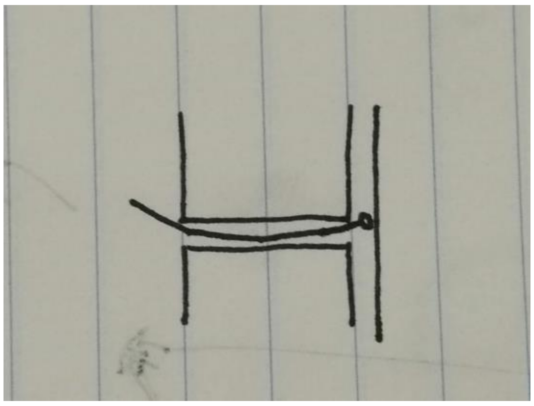
Combined sampling strategies improve the sampling efficiency, so that we can obtain more valid samples in short time, which will help to solve some cluttered map searching.

First strategy is about sampling. As we are using the start point and following angles to represent the configuration in C-space, the greatest advantage of it is that no matter how we sampled in C-space, the broom length in work space will be fixed.

Also, we have limit the angle range of those intermediate state. The limit has been set as π-θ where θ is the smallest angle between initial and goal configuration. For example, as shown in the following photo, when sampling a new state, say s, we only sample the angle alpha in the range of [3PI/4, PI/2], because there is no need to sample an angle smaller than PI/2 (to pass a narrow passage, the larger the angle, the better performance). These have improved the success rate of our sampling thus increased the overall efficiency.



However, there might be some extreme situations which require smaller angles, such as the passage in the following photo. Therefore, in our program, if sampling fails too many times, then the angle range limit will be cancel temporally.

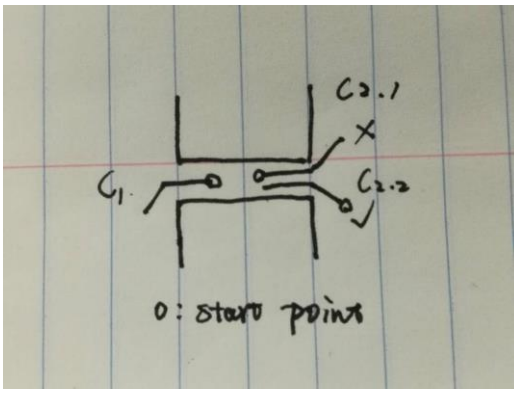


For further sampling strategy, we used the given obstacle to calculate the intersection part when doing projection on other obstacles. As the question was described as fully observable, I guess this is reasonable, which allowed us to sample more into the passageways, so that we can solve those scenarios with relatively narrow passageways a bit more efficient.

(2) Distance measurement between c-space states

This strategy improves the performance of passing a narrow passage.

The strategy that we used to use is to calculate the total distance between two states in C-space, which is time consuming and not necessarily accurate. In fact, among two configurations, the most representative valve would be the position of start point and the first angle. The distance between start point represent the distance of two configuration while the distance between first angle represent its vague direction. This solve the situation describe as below.



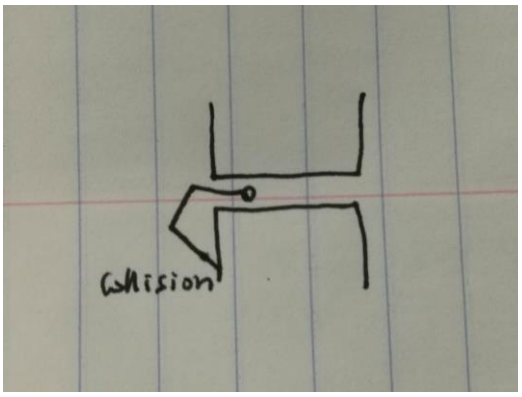
Distance measurement

For the original method, it will consider C1 is closer to C2.1, which will need to rotate those ASVs 180°, however, this is impossible in such a narrow passage. In the new method, we increased the weight of first angle in the distance measurement, so that the C2.2 will be considered closer, which can lead the way out of the passage. This will also help us improved the efficiency of tackle with scenarios have narrow passageways.

(3) Enlarge angle

This strategy can help ASVs to pass long narrow passage significantly.

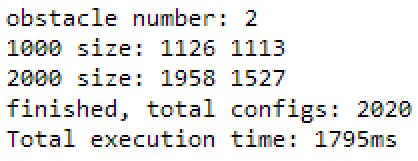
In the state illustrated below, part of the state has already in the passage. If ASVs wants to pass, the most intuitive way is to enlarge the angles between every two brooms. Therefore, in our strategy, when collisions happen, we will enlarge these angles and see whether the ASVs can pass. This should improve overall efficiency, especially in the scenarios when there are multiple obstacles in the workspace.



Enlarge angle

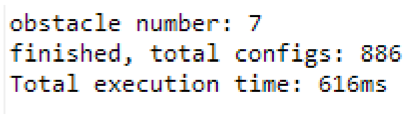
Experimental result

Because we are using rapidly-exploring random tree, instead of sampling all the way, we are doing sampling and explorations alternatives with each other. In that case, the runtime of program will not be strongly affected by the complexity of obstacle as long as they do not include and narrow passages.



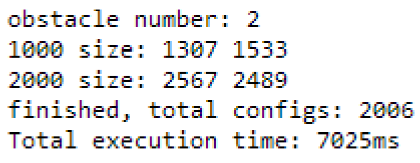
```
obstacle number: 2
1000 size: 1126 1113
2000 size: 1958 1527
finished, total configs: 2020
Total execution time: 1795ms
```

3ASV-x4.txt



```
obstacle number: 7
finished, total configs: 886
Total execution time: 616ms
```

3ASV.txt



```
obstacle number: 2
1000 size: 1307 1533
2000 size: 2567 2489
finished, total configs: 2006
Total execution time: 7025ms
```

7ASV-x4.txt

```
obstacle number: 7
1000 size: 872 975
2000 size: 1876 1030
3000 size: 3054 1075
4000 size: 4098 2110
finished, total configs: 4376
Total execution time: 10019ms
```

7ASV.txt

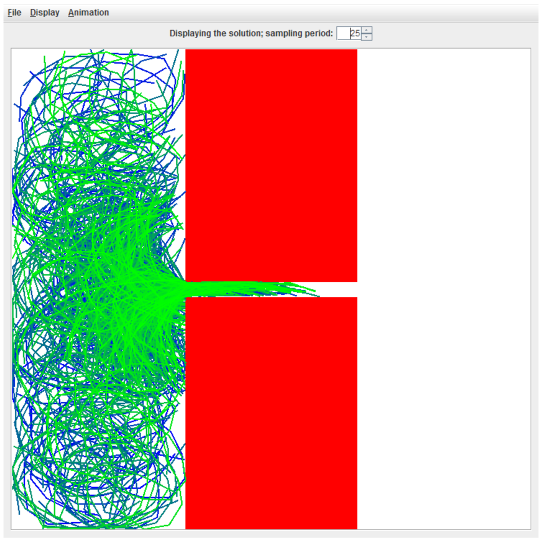
The results of our experiments have proved what I suggest above, the increase of obstacle number of will not affect the execution time a lot in low dimension situation. Also, because we are using the RRT-connect instead of normal RRT, the efficiency of search for a tree will be improved as the algorithm builds up two different parts at same time.

To conclude, our algorithm can solve most of those uncluttered problem and the strategies that we used helped us improved the overall efficiency. Also, as long as the passage is not too narrow and the number of ASV is not too much, we can solve most of those cluttered problems with narrow passageways and multiple obstacles.

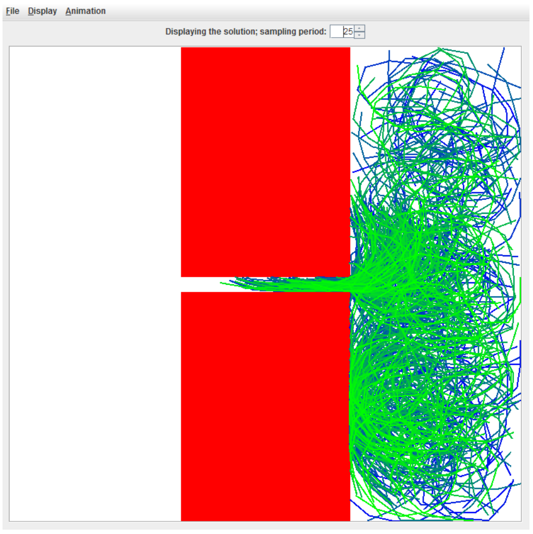
4. [12.5 points] Under what situation do you think your program will fail? Please explain your answer.

It is high chance that we will still fail when it comes to scenarios with very long and narrow passage that is too narrow.

For the strategy we used to use, we will have the graphs as below.



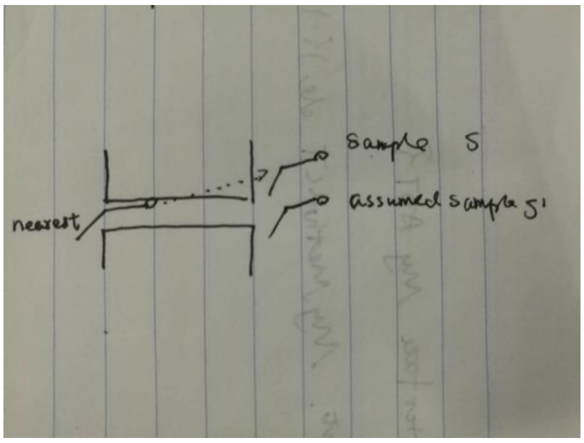
7ASV-x2-leftTree.txt



7ASV-x2-rightTree.txt

From those plots, we can tell that both side of the tree is well extending in their own part, but they can never meet each other. In another word, the tree can grow into the narrow passage, but it is difficult for it to get out again.

Thus, we introduced the "Ladder" movement method.



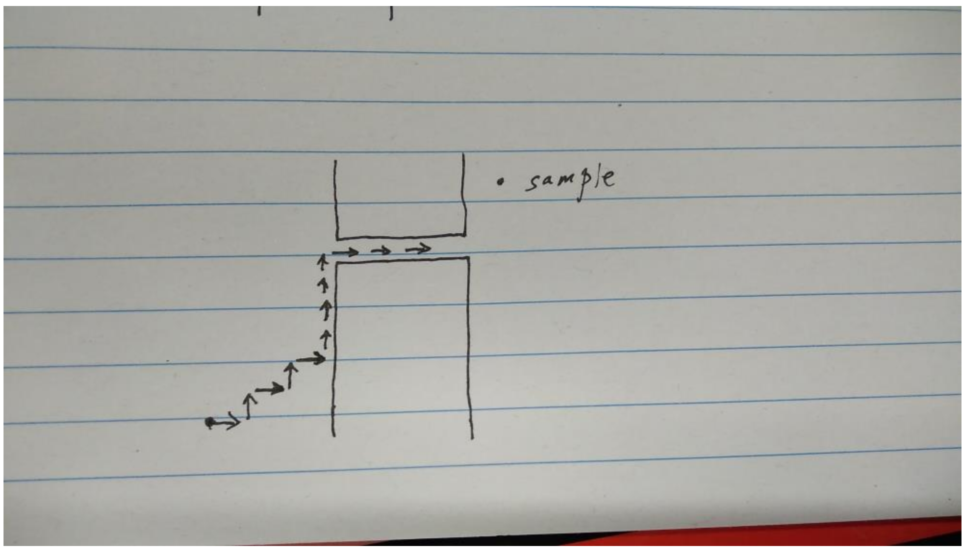
"Ladder" Move

Sometimes when the nearest wants to move to sample S, it will have collision immediately. But as the passage is narrow, the chance that sampled configuration in the position of S' is rather low. Thus, we think that we can limit the moving on y-axis direction and lessen the distance on the direction of x-axis first. This helped us a lot as it can get the configuration in the narrow passage out. The pseudo code is like below

```
LadderMove (start, target):
 result := start
 while true:
     result := move one step on the direction of x axis
     result := move one step on the direction of y axis
```

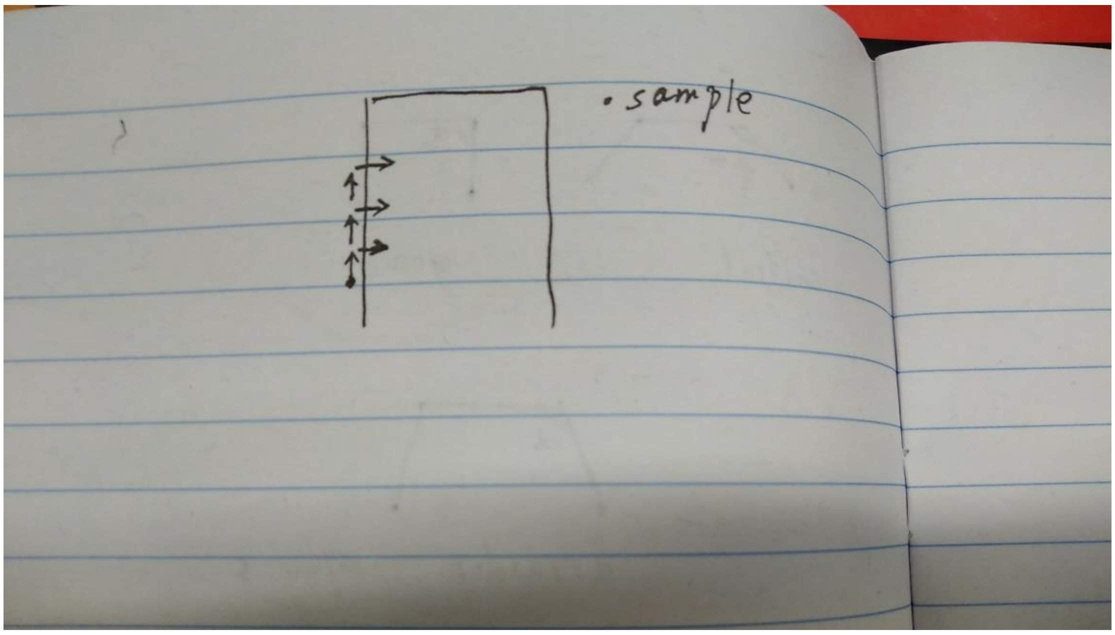
```
    if both the two moves fail
      return result
    if encounter the target
      return result
```

The following photo presents the procedure. (for convenience, we just use a point to represent ASVs)

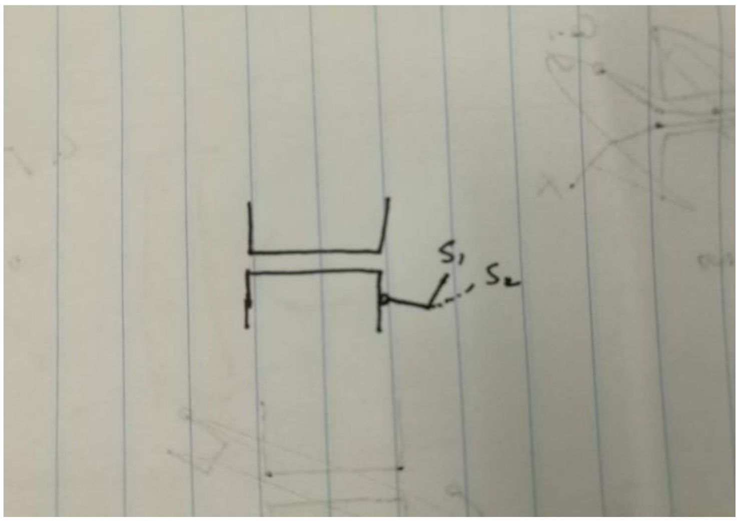


However, if the passage is very long and narrow, it might still get stuck in this area. Besides, such a strategy will make the searching slower.

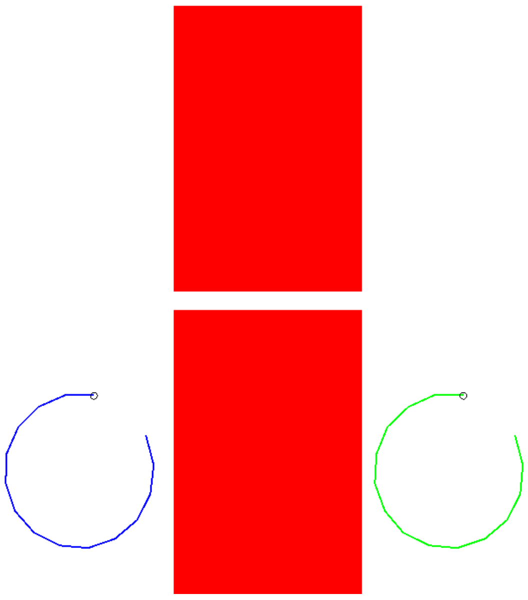
Another weakness of this ladder strategy is shown in the following photo. When a point is moving along a bound of an obstacle, based on the pseudo code, we can know that, in each step, the point will try to move right, which could be time consuming.



The second weakness is showed as above. The angle enlarge strategy did helps a lot when the configuration needs to get across long narrow passageways, but it can get stuck when only its start point is about to have collision. The angle will keep enlarge but the configuration will still have collision. Although we have set the maximum repeat time to allow it pass narrow passage so that it will not get stuck for too long, it is still a short come that needs further work to tackle.



To conclude, we can tackle some scenarios with narrow passage ways, but if there are some very long and narrow passages, and the number of ASVs is large, such as the scenario shown below.



Role of Member

Yi Liu

Report Question 2,4

Summary of Tester part2

Main structure of RRT including the use of different strategy and functions including findNearest, findNext, cSpaceDist, stepMove, cSpaceCollisionCheck and cutDist

Move strategy including enlarge angle and ladder move.

Config class and its method

Efficiency improvement including getAngleRange, increaseAngle, reversePosition and cutArray

Improvement on output file, different sample strategies (getRandomPoint)

Hengchen Guo

Report Question 1,3,4

Summary of Tester part1

Proposed use angle as C-space configuration

Output file including getSol and printPosition

Config transformation including asvConfigToCfg and cfgToArray

Basic sample including getStartPosition and getNextPoint

Further sample strategy including SampleSpace and getSinCor

Test file of 15ASV