HW5 for CSE276A

Yunhai Han
A53307224
y8han@eng.ucsd.edu

1 Problem description

In homework5, I am required to design a "roomba" like system. The robot should be able to navigate an environment and provide a level of coverage of the area. I can choose any architecture for this assignment, and subsumption is recommended to be a good starting point.

1.1 Environment

Here, I put an image which shows how I place the the QR codes and obstacles.

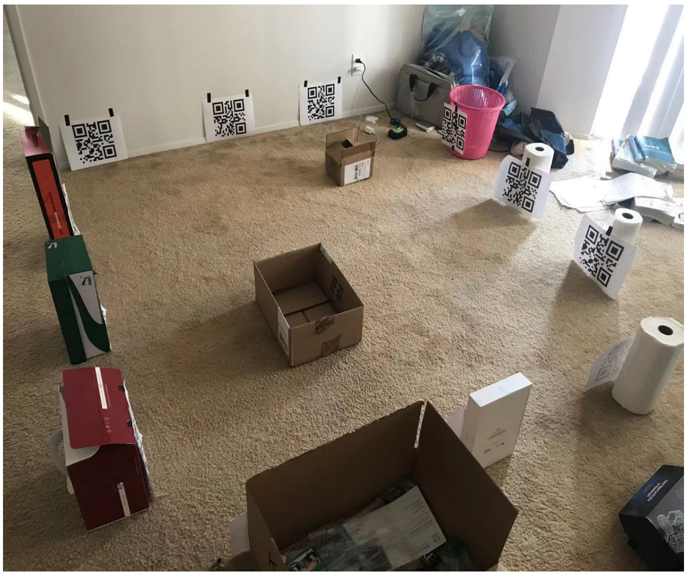

Figure 1: Environment

From my perspective, two methods could are suitable for this homework. The first one is that the robot do not make any planning and it only takes actions when the surrounding environment changes. For example, roomba dose not know where it is and how many areas have been covered. It just keeps going forward and turns right/left when it encounters some obstacles or walls. This method is really easy for implementation and professor Christensen said some robotics vacuum cleaner adopts this method because it could confuse the consumers whether they do a good job or just do nothing(I think it may just be a joke).

The second one is much more difficult and entails more efforts. It requires to know the map and find the optimal path from the map which would cover the largest areas. Then, the robot needs to follow the existed path and the accuracy of the localization and control becomes very significant. As suggested, I should divide the task into different parts and try to deal with each part at a time. Then simply by the combination of each parts, the whole task could be fulfilled. This is called the decomposition of the problem, which is widely used in robotics industry because it could make life easier for the modularity and high design efficiency.

In this case, the robot is required to navigate through an environment and provide a level of coverage. I could divide the task into four parts:mapping, motion planning, sensing and control.

In the next section, I would demonstrate the results using the first method and after that, I would show how I design each parts for the second method.

2 The first method

I use Matlab to build the simulated environment based on the real configuration in Figure 1.

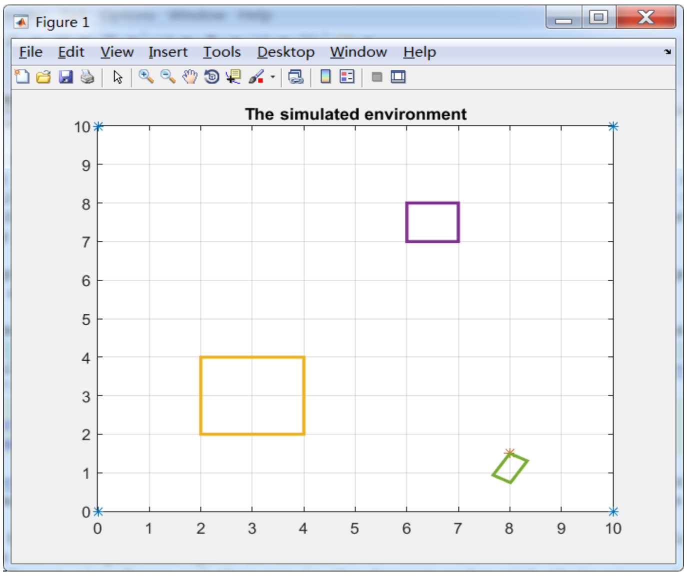

Figure 2: The simulated environment

In Figure 2, you can see there are two obstacles. However, though the simulated environment is similar with the real one, it is still difficult for us to make further analysis because the robot has its unique shape and various orientations.

In homework4, the robot could be considered as a point if I introduce the configuration space and this would greatly simplify the problem. I use the same method in homework4 and obtain the configuration space as follow.

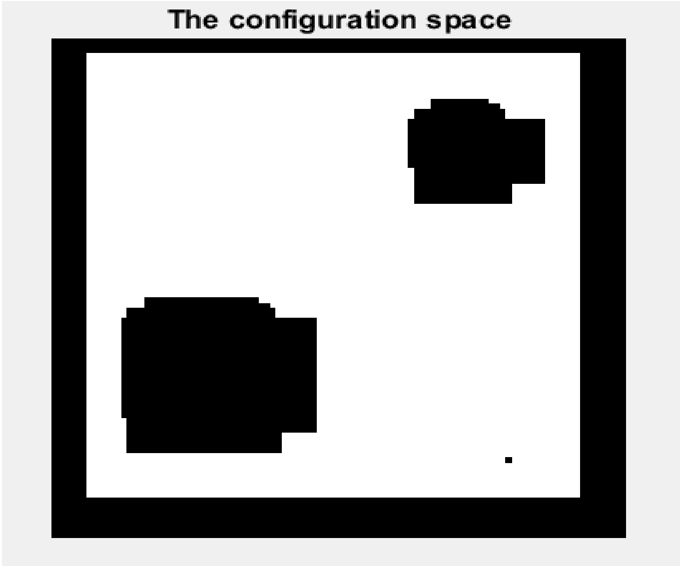

Figure 3: The configuration space

In Figure 3, you can see the reachable space for the robot(right now it is a point) is much smaller than before because I take all the possible orientations into consideration at the same time, which is unrealistic but could simplify the model.

I suppose at the beginning, the orientation of the robot is $60°$ and it would move forward until it encounters some obstacles or walls and then it would turn left/right for $90°$. The robot could use any kinds of sensors to detect obstacles, like ultrasonic sensors(the most widely used) or cameras. If the robot could move forever without the limitation of its battery, maybe it could cover nearly all the areas.

I use Matlab to make a simulation and calculate the the ratio of covered areas to the all areas every 100 iterations.

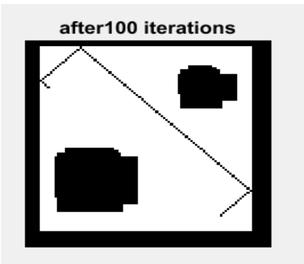

Figure 4: 100 iterations

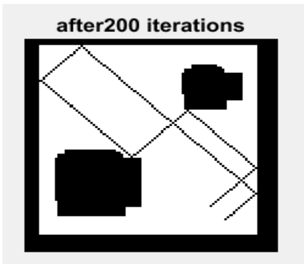

Figure 5: 200 iterations

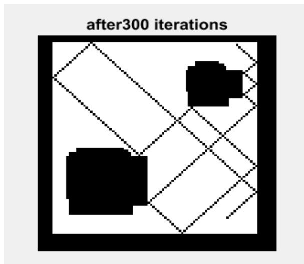

Figure 6: 300 iterations

In the following figures, these black pixels represent the routes the robot moves through. Because the robot only reacts to its surroundings and doesn't make any motion planning, it is not the optimal solution for the robot to cover all the areas. However, in most cases, people never expect their robots to do the best job, instead they would be happy if their robots keep working(even in a stupid way)(It's a joke).

In order to examine whether the robot could cover all the areas in such a way, there should be more iterations.

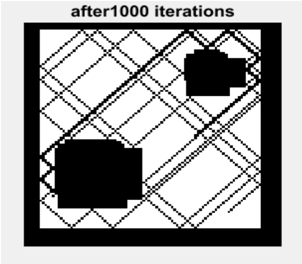

Figure 7: 1000 iterations

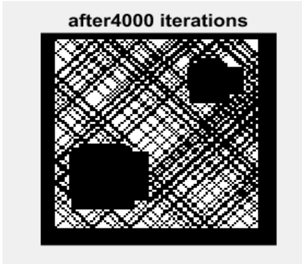

Figure 8: 4000 iterations

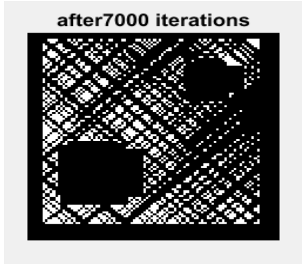

Figure 9: 7000 iterations

In figure 9, after 7000 iterations, it seems that most areas has been covered, which means this method works. Also, I plot the ratio of coverage versus iterations(time).

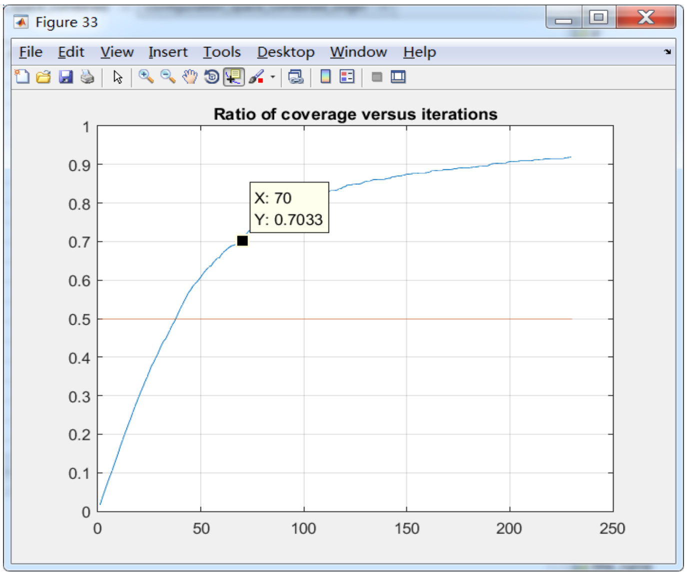

Figure 10: Ratio of coverage versus iterations

From figure 10, you could see that indeed only after about 4000 iterations, half of the areas have been covered. And after 7000 iterations, more than $\frac{2}{3}$ areas have been covered and this result is satisfactory.

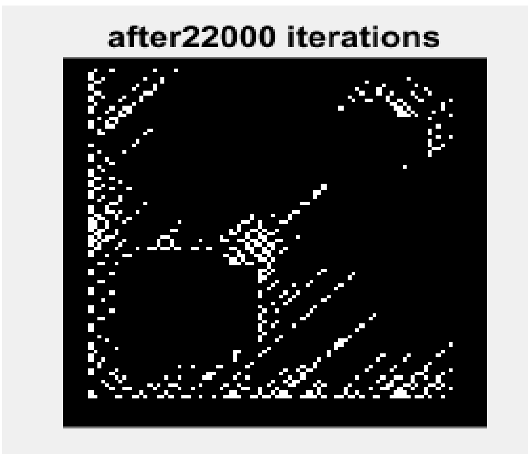

Figure 11: 22000 iterations

In figure 11, after 22000 iterations, almost all areas have been covered and the ratio is 0.9204.

To sum up, for the first method, the robot could cover almost all areas if it could run for enough time. Besides, because this method doesn't need any prior knowledge of the environment ant it could also work for dynamic environment, which is a great advantage over the second method. So, I could tell we don't have to provide with anything(no special behaviours or performance guarantees are needed), the robot could always covers nearly all the areas.

Another advantage of it is that it doesn't need high localization and sensing accuracy. It could work for the robot with cheap motors and sensors(like Picar).

The biggest disadvantage of the first method is efficiency. It is really a time-consuming method. During the first 4000 iterations, most areas are uncovered, so the ratio of coverage increases quickly. Afterwards, because most areas have already been covered, the subsequent motion actions contribute less to the coverage. Take robotics vacuum cleaner as an example, most areas have already been cleaned and the next time it reaches the same areas, it has nothing to do.

The control flow of this method is simple.

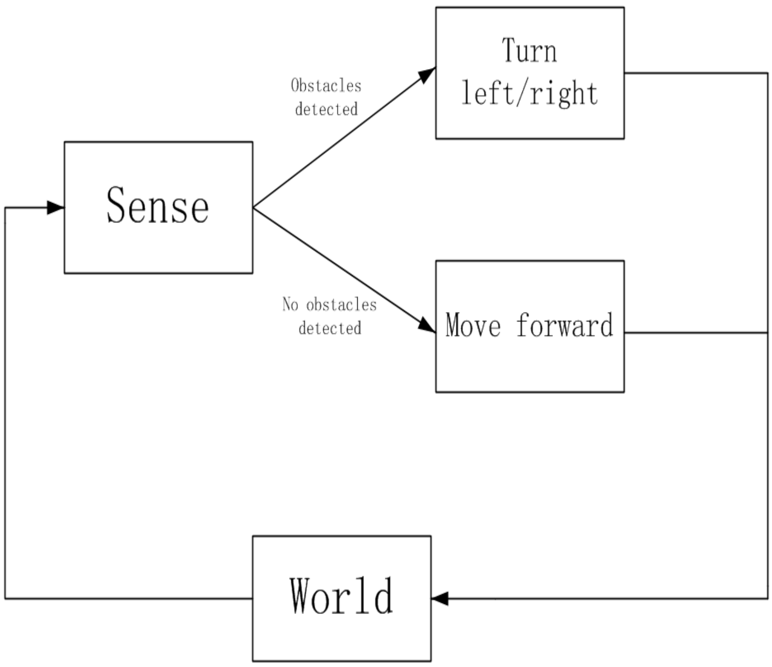

Figure 12: The control flow of the first method

3 The second method

The second method requires motion planning based on an existed map. With hybrid architectures, I could divide the task into different parts.

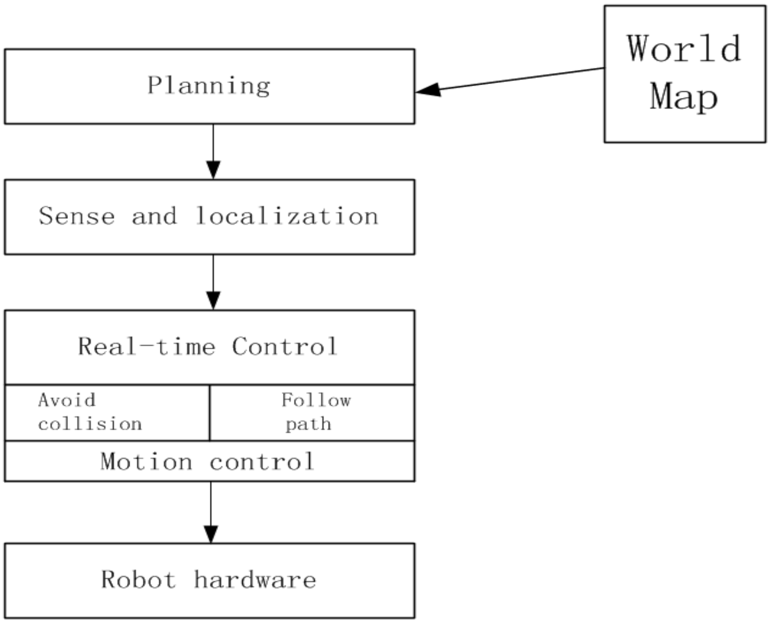

Figure 13: The hybrid architecture

From figure 13, there are five different modules: World Map, Planning, Sense, Control and robot hardware. For the first four modules, I would show how I design each of them. However, robot hardware depends on the robot itself and different robots could have their own hardwares. Hence, I only take Picar as an example to show how to combine the use four modules onto a real robot.

3.1 World Map

In order to build a world map, the robot has to navigate through the environment at the beginning. During this process, the robot build a map based on the sensor data and its own position. In homework3, I have successfully built a map containing all the positions of QR landmarks, but in this homework, because I put some obstacles inside, I need to figure out a method to detect the obstacles and obtain their positions.

The most common practise is to use rangefinders(laser rangefinders or stereo cameras) which is very useful to build a 3D map. These sensors could emit laser beams and these beams bounces off distant objects. The rangefinders measure the total time it took from when the beams left the unit until they returned and as a result, they could feedback a depth image in front of the robot. If there is an obstacle in front of the robot, the laser beam would return much more quickly than others so, the depth of this point could also be smaller then others. By the combination of robot's wheel odometry and on-board rangefinders, it is possible for us to build a map of the environment even there are various obstacles inside. The map includes different boundaries(walls) and various obstacles(boxes in this case).

If there is a robot with encoders or IMUS for the implementation of wheel odometry and some on-board sensors for building a map, I could I assume I've built a world map from that robot for the design of next module as the one in figure 2. I think this is what robotics company do when they design their products:different people are responsible for each module and when they design their own module, they don't consider the goodness of other modules and just assume others could do as good as they can.

Also, in order to consider the robot as a point for the simplicity, I build a configuration space map as the one in figure 3.

3.2 Planning

The main purpose of planning is trying to find a path for the robot to cover the most areas. However, there are indeed infinite solutions for this problem. How can we choose the best solution? Here, I add one more constraint:trying to find the shortest path. If the speed of the robot does not change, it would also take the least time if it follows the shortest path.

This problem is like a travelling salesman problem:Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city. The robot is analogous to the salesman and the waypoints are analogous to the cities. if the cities are located everywhere in the map, the salesman visiting each city and returning to the origin city means that he also goes through the whole map. Moreovers, because it is still a travelling salesman problem, the obtained path is also the shortest one.

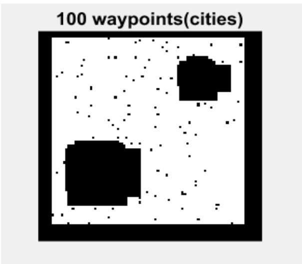

Figure 14: 100 waypoints

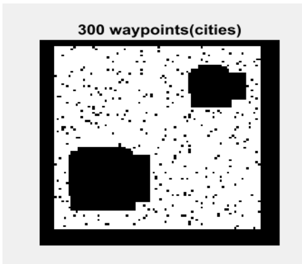

Figure 15: 300 waypoints

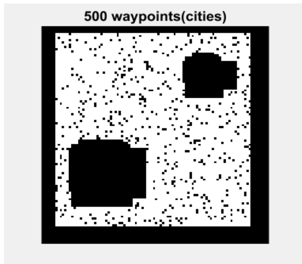

Figure 16: 500 waypoints

The waypoints are created randomly, and from figure 15, you can see that 300 waypoints are enough to cover almost every possible places in the map.

Then, I need to solve the travelling salesman problem and calculate the length of the shortest path. However, as we all know, it is a NP-hard problem and it is really hard for us to find a global optimal solution especially I have so many cities and its running-time would increase superpolynomially with the number of cities. Instead of trying to find a closed-form solution, I introduce a heuristic algorithm called simulated annealing(SA). Here is a brief introduction of this algorithm from wiki:Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. Specifically, it is a metaheuristic to approximate global optimization in a large search space for an optimization problem.

Because of its robustness for global optimization in a large scale, it is very suitable for solving travelling salesman problem. There is some significant parameters and the most important one is called cooling speed. If it is set very small(far away from 1), the iterations are not enough to find a global optimal solution. On the other side, if it is set very high(close to 1), it would converge very slowly, consuming much more time.

First, I set the cooling speed as 0.9 and obtained the best solutions for different number of waypoints.

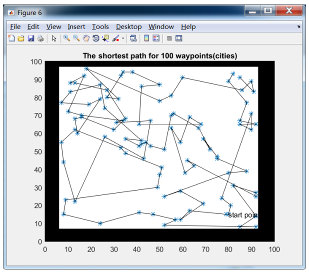

Figure 17: 100 waypoints

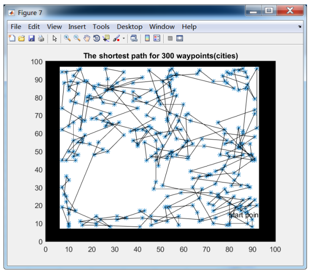

Figure 18: 300 waypoints

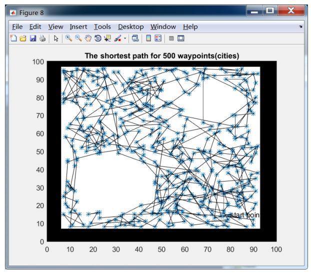

Figure 19: 500 waypoints

The length of shortest paths for each are 1257.6, 2795.9 and 4174.2(ft). From figure 17 to figure 19, you can see that the ratio of coverage increases as the number of waypoints increases. In figure 17, because there are only 100 waypoints, which are not enough to cover all the areas, the shortest path even passes through the obstacles(the two obstacles are not shown in the figures). At first, I don't take obstacles into consideration when I calculate the distance between waypoints because it is much easier for the implementation. How they perform would help me decide whether I need to improve this algorithm to take the connectability between different waypoints into consideration.

And the result shown in figure 19 could answer the question:two dents in this figure illustrate that I don't need to make any improvements if the number of waypoints are large enough. Because when there are enough waypoints, it is impossible for the shortest path to pass through the obstacles. The reason for that is really simple:such a path must not be the shortest one since the distance between two waypoints separated by an obstacle is larger than others. It is less possible for the results to converge at this solution except for that the cooling speed is too small so that it could not expand enough solution space to find a global optimal solution.

In order to see whether it is true or not, I add more waypoints into the map:700 and 900.

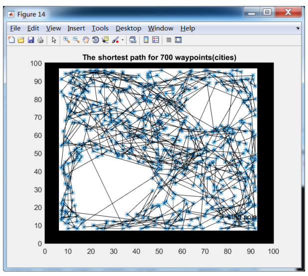

Figure 20: 700 waypoints

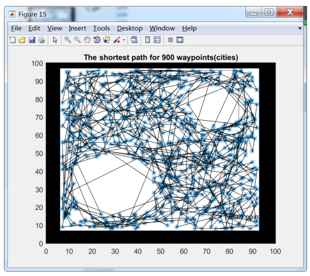

Figure 21: 900 waypoints

The length of shortest paths for each are 7643.8 and 8964.8(ft).

However, in figure 20 and figure 21, there are still some lines passing through the obstacles. I think maybe the reason is that the cooling speed is a little small,so it could not converged to a global optimal solution.

Hence, I set the cooling speed as 0.99 and do the tests again.

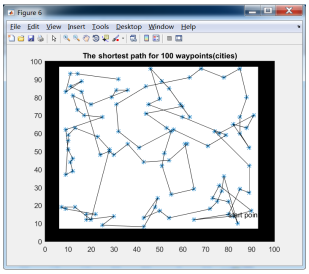

Figure 22: 100 waypoints

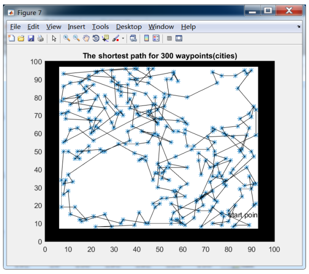

Figure 23: 300 waypoints

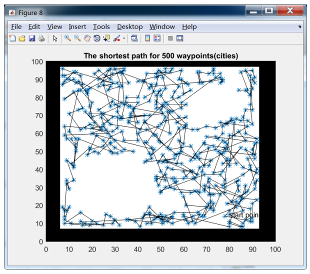

Figure 24: 500 waypoints

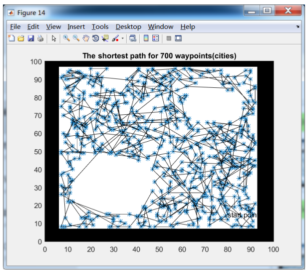

Figure 25: 700 waypoints

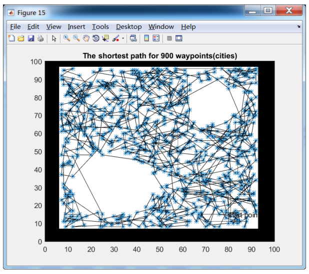

Figure 26: 900 waypoints

The length of shortest paths for each are 1106.9, 2228.2, 3261.5, 4826.3 and 6196.3(ft).

From figure 22 to figure 26, the assumption could be proved because if I set the cooling speed a little higher to guarantee enough iterations for the convergence, the robot could always avoid the two obstacles by following the optimal paths(two dents in figure 25 and figure 26).

However, it takes me almost five minites to obtain these results when I set it as 0.99.

Finally, I set the cooling speed as 0.999.

It takes really a long time to find the solutions. I went to a market to buy some fruits and milk and had the dinner at a nearby restaurant while the Matlab was working. After I came back, it still kept on working and I waited for another twenty minutes until I obtained the results for the 700 waypoints. So I choose not to obtain all of them and the results for 700 waypoints could prove what I tell before.

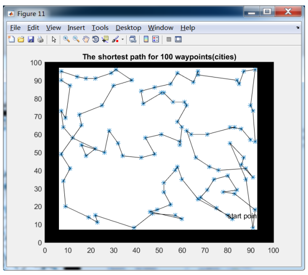

Figure 27: 100 waypoints

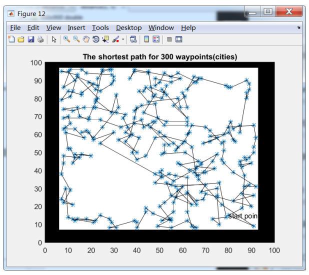

Figure 28: 300 waypoints

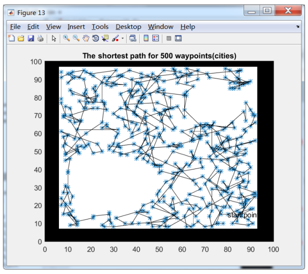

Figure 29: 500 waypoints

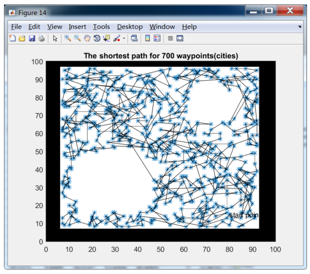

Figure 30: 700 waypoints

The length of shortest paths for each are 826.2, 1878.1, 2951.3, 4826.3 and 3983.7(ft). It is obvious that the solutions are getting closer to the literally optimal solutions(It exist somewhere in the solution space).

In figure 29 and figure 30, most areas are covered and the two dents are clear, which shows the robot could absolutely avoid any contacts and finish the cleaning mission in the shortest time meanwhile if it follows the optimal paths.

To sum up, simulated annealing algorithm is really suitable for this mission. It could find the shortest path covering most areas and avoid any obstacles at the same time. There are two requirements for providing coverage and avoidance. The first one is that the cooling speed is large(0.999) and the second one is that the number of waypoints is large(more than 500). If they are satisfied, no more behaviours are needed and nearly 100% of coverage could be guaranteed as shown in figure 30. Moreover, for a robotics vaccum cleaner, when consumers do not use them, they could solve the TSP no matter how much time it would take. When the consumers press the 'start' button, the robot could follow the optimal path it calculated before and the consumers would think the robot is so intelligent and are willing to buy more products from that robotics company!

3.3 Sense and localization

The previous module could provide the robot with a best path which covers most areas in the shortest time. The purpose of this module is to keep the robot on track. From the homework1 to homework4, I already know that open-loop control could always cause a diaster. This module could be designed on the previous homeworks:The robot could rely on its own wheel odometry and on-board camera to improve the accuracy of localization.

Since all the waypoints are known before the robot embarks on the journey, if it follows each waypoints in the path in the order, it would cover most areas. In the homework1 and homework2, this problem was solved. I just briefly restated the main methods used before.

First, the robot measures its position and the target waypoint. If it deviates from the right direction, PID-based control algorithm could help it back on the track. For example, if the steering angle is larger than expected and as a result, the robot would finally reach a point left/right to the waypoint. PID-based control algorithm could tune the steering angle in real time to make the angle error smaller.

Second, we all know that only wheel odometry is not enough for accurate measurement of localization. With the on-board camera, the robot could detect the QR landmarks placed on the walls. The positions of all the QR landmarks are known, so it provides with another way to measure the position of the robot in the map by detecting QR codes and the relative positions between them and the robot measured from camera data.

It is more like SLAM without mapping and the full SLAM is needed for the first module(World Map) to build an accurate world map and I could use that map with configuration space to solve the motion planning problem in the second module.

3.4 Real-time control

There seems no more theoretical design for this module. In the second module, the robot could avoid collision and cover most areas. In the third part, the robot could always follow the optimal path. Hence, in this module, no more improvements could be added into the system. However, dose it mean this module is insignificant? The answer is no. In the real world, all the data are noisy and if we want to get a really result, we need to tune different parameters for each parts. For example, there are at lease three parameters to tune for PID controller(P->proportional, I->integration, D->differential) and there are two noise matrix Q and R to tune for the Extended Kalman Filter(EKF) as in the homework3. The robot could do a good job in real world(not just in simulation), if and only if all these parameters are well tuned. This is much more laboring than designing each module because we may have to do lots of repeated work. However, it is indispensable for every robotics company if they want to produce wonderful products.

3.5 Robot hardware

This module includes the lowest-level designs including mechanical design and electronics design.

For mechanical design, these mechanical engineers are responsible to build a robot using 3D modelling softwares like Solidworks or ProE. There are certain requirements of the robot that must be satisfied, including size, mobility, costs of available manufacturing methods, appearance and so on. If the mechanical design is poor, the robot could not behave very well even if the algorithms are perfect. Just imagine, how about the performance of a robot whose wheels fall off or the motor sitting gets loose?

For electronics design, these electronics engineers are responsible to design a PCB to activate all the motors and sensors on the robot. Also, they need to decide how the PCB get connected to the microprocessor and the power supply. Besides, from my previous project experience, they may also need to decide how to place various sensors like ultrasonic sensors and cameras. I think this is hard to deal with, because in most cases, we don't know how well the robot would perform until all the components on the robot are installed. Sometimes, the bad performance is not due to the deficiency of the algorithms but due to the bad installation of sensors.

The intersection of softwares and hardwares makes building a good robot a really challenging problem, and I think this is why robotics is so fascinating because there are always some components that could be improved and no one knows how well a robot could perform in the future!

3.6 Control flow of the second method

The control of the second is different from that of the first method shown in figure 12.

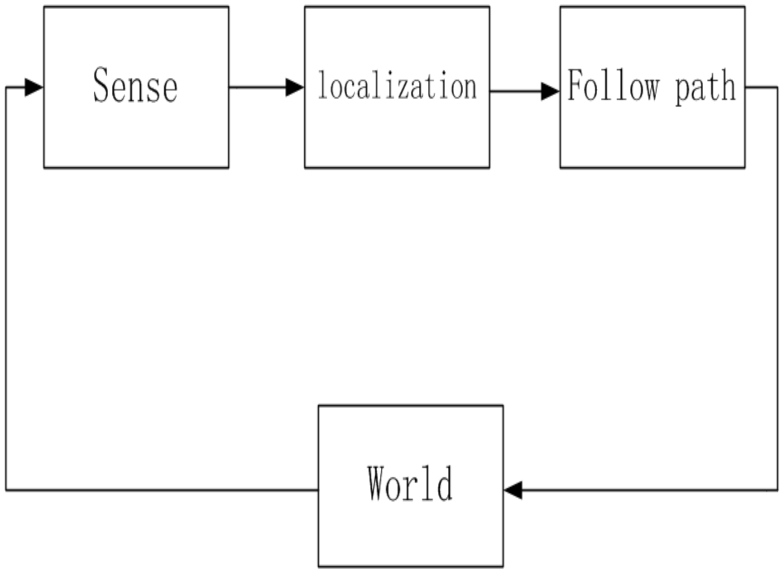

Figure 31: The control flow of the second method

Matlab Code

I put all the Matlab codes here.

```
%cse276A hw5
%The first method
close all
corners=[0 0 10 10;0 10 0 10];
start=[8,1.5];
obstacle1=[2 4 4 2 2;2 2 4 4 2];
obstacle2=[7 8 8 7 7;7 7 8 8 7];
figure(1)
plot(corners(1,:),corners(2,:),'*');
grid on
hold on
plot(start(1),start(2),'*')
plot(obstacle1(1,:),obstacle1(2,:),'LineWidth',2);
plot(obstacle2(1,:),obstacle2(2,:),'LineWidth',2);
%robot orientation
rect=cse_276A_robot_ori(start,60);
plot([rect(1,:),start(1)],[rect(2,:),start(2)],'LineWidth',2);
title('The simulated environment')
for index=1:7
    configuration_spac0e=ones(100,100);
    for i=20:40
        for j=20:40
            configuration_spac0e(i,j)=0;
        end
    end
    for i=70:80
        for j=70:80
            configuration_spac0e(i,j)=0;
        end
```

```
    end
    for i=80:84
        for j=85:91
            configuration_spac0e(i,j)=0;
        end
    end
    configuration_spac0e=configuration_spac0e';
    for i=1:100
        for j=100:-1:1
            point=[i/10,(100-j)/10];
            point_x=point(1,1);
            point_y=point(1,2);
            rect=cse_276A_robot_ori(point,0+30*(index-1));
%            plot([rect(1,:),point_x],[rect(2,:),point_y],'LineWidth',2);
%            pause(0.05)
            point1_x=rect(1,2);
            point2_x=rect(2,2);
            point3_x=rect(1,3);
            point1_y=rect(2,3);
            point2_y=rect(1,4);
            point3_y=rect(2,4);
            point_in=1;
            if(point_x>2&&point_x<4)
               if(point_y>2&&point_y<4)
                   point_in=0;
               end
            end
            if(point_x>7&&point_x<8)
               if(point_y>7&&point_y<8)
                   point_in=0;
               end
            end
            point1_in=point1_x<=10&&point1_x>=0&&point1_y<=10&&point1_y>=0;
            if(point1_in)
               if(point1_x>=2&&point1_x<=4)
                   if(point1_y>2&&point1_y<4)
                       point1_in=0;
                   end
               end
               if(point1_x>=7&&point1_x<=8)
                   if(point1_y>7&&point1_y<8)
                       point1_in=0;
                   end
               end
            end
            point2_in=point2_x<=10&&point2_x>=0&&point2_y<=10&&point2_y>=0;
            if(point2_in)
               if(point2_x>=2&&point2_x<=4)
                   if(point2_y>2&&point2_y<4)
                       point2_in=0;
                   end
               end
               if(point2_x>=7&&point2_x<=8)
                   if(point2_y>7&&point2_y<8)
                       point2_in=0;
                   end
               end
            end
            point3_in=point3_x<=10&&point3_x>=0&&point3_y<=10&&point3_y>=0;
            if(point3_in)
               if(point3_x>=2&&point3_x<=4)
                   if(point3_y>2&&point3_y<4)
                       point3_in=0;
```

```
                    end
                end
                if(point3_x>=7&&point3_x<=8)
                    if(point3_y>7&&point3_y<8)
                        point3_in=0;
                    end
                end
            end
            if(point_in&&point1_in&&point2_in&&point3_in)
                configuration_spac0e(i,j)=1;
            else
                configuration_spac0e(i,j)=0;
            end
        end
    end
    configuration_spac0e(80,85)=0;
    configuration_spac0e=configuration_spac0e';
%     figure(1+index)
%     imshow(configuration_spac0e)
    configuration_space{index}=configuration_spac0e;
end
configurations_space_combined=ones(100,100);
for i=1:100
    for j=1:100
        zero_=[configuration_space{1}(i,j),configuration_space{2}(i,j),
            configuration_space{3}(i,j),configuration_space{4}(i,j),
            configuration_space{5}(i,j),configuration_space{6}(i,j),
            configuration_space{7}(i,j)];
        if(0==min(zero_))
            configurations_space_combined(i,j)=0;
        end
    end
end
% figure(9)
% imshow(configurations_space_combined)
title('The configuration space')
%the start point(80,85) orientation 45
configuration_space_combined_origin=configurations_space_combined;
occupied=zeros(100,100);
restart=0;
iterations=0;
index=1;
total_areas=sum(sum(configuration_space_combined_origin));
while(1)
    if restart==0
        x_now=100-85;
        y_now=80;
    else
        x_now=round(rand*100);
        y_now=round(rand*100);
        while(x_now==100||x_now==0||y_now==100||y_now==0)
            x_now=round(rand*100);
            y_now=round(rand*100);
        end
        while(configurations_space_combined(100-x_now,y_now)==0&&occupied(100-x_now,
            y_now)==0)
            x_now=round(rand*100);
            y_now=round(rand*100);
            while(x_now==100||x_now==0||y_now==100||y_now==0)
                x_now=round(rand*100);
                y_now=round(rand*100);
            end
        end
```

```
    end
    occupied(100-x_now,y_now)=1;
    orientation=45;
    while(1)
        ori=orientation/45;
        switch ori
            case 0
                x_next=x_now;
                y_next=y_now+1;
            case 1
                x_next=x_now+1;
                y_next=y_now+1;
            case 2
                x_next=x_now+1;
                y_next=y_now;
            case 3
                x_next=x_now+1;
                y_next=y_now-1;
            case 4
                x_next=x_now;
                y_next=y_now-1;
            case 5
                x_next=x_now-1;
                y_next=y_now-1;
            case 6
                x_next=x_now-1;
                y_next=y_now;
            case 7
                x_next=x_now-1;
                y_next=y_now+1;
            case 8
                x_next=x_now;
                y_next=y_now+1;
        end
        if configurations_space_combined(100-x_next,y_next)==0&&occupied(100-x_next,
            y_next)==0
            state=0;
            if configurations_space_combined(100-(x_now-1),y_now-1)==1
                state=1;
                orientation=225;
            else if configurations_space_combined(100-(x_now+1),y_now-1)==1
                    state=1;
                    orientation=135;
                else if configurations_space_combined(100-(x_now+1),y_now+1)==1
                        orientation=45;
                        state=1;
                        else if configurations_space_combined(100-(x_now-1),y_now+1)
                            ==1
                            orientation=315;
                            state=1;
                            end
                    end
                end
            end
            if state==0
                restart=restart+1;
                break;
            end
            %force to change a start point, like being kidnapped
        else
            configurations_space_combined(100-x_next,y_next)=0;
            iterations=iterations+1;
            if(mod(iterations,100)==0)
```

```
                ratio{index}=sum(sum(occupied))/total_areas;
                title_name=strcat('after ',int2str(index*100),' iterations');
%                 if mod(index,10)==0
% %                     figure(9+index/10)
% %                     imshow(configurations_space_combined)
% %                     title(title_name)
%                 end
                index=index+1;
            end
            x_now=x_next;
            y_now=y_next;
            occupied(100-x_now,y_now)=1;
        end
        if index==230
            break;
        end
    end
    if index==230
        break;
    end
end
index=index-1;
ratios_vector=ones(1,index);
for i=1:index
    ratios_vector(1,i)=ratio{i};
end
% figure(34)
% plot(ratios_vector)
% hold on
% grid on
% title('Ratio of coverage versus iterations')
% ratios_half=0.5*ones(1,index);
% plot(ratios_half)
figure(10)
imshow(configuration_space_combined_origin);
configuration_space_combined_origina=configuration_space_combined_origin;
title('The configuration space')
number_waypoints=[100,300,500,700,900];
for z=1:5
    configuration_space_combined_origin=configuration_space_combined_origina;
    waypoints_x=zeros(1,number_waypoints(1,z));
    waypoints_y=zeros(1,number_waypoints(1,z));
    index=1;
    while(index<=number_waypoints(1,z))
         x_now=round(rand*100);
         y_now=round(rand*100);
         while(x_now==100||x_now==0||y_now==100||y_now==0)
             x_now=round(rand*100);
             y_now=round(rand*100);
         end
         while(configuration_space_combined_origin(100-x_now,y_now)==0)
             x_now=round(rand*100);
             y_now=round(rand*100);
             while(x_now==100||x_now==0||y_now==100||y_now==0)
                 x_now=round(rand*100);
                 y_now=round(rand*100);
             end
         end
         if index==1
            x_now=15;
            y_now=80;
         end
         waypoints_x(1,index)=x_now;
```

```
            waypoints_y(1,index)=y_now;
            configuration_space_combined_origin(100-waypoints_x(1,index),waypoints_y(1,
                index))=0;
            index=index+1;
        end
        figure(1+z)
        imshow(configuration_space_combined_origin)
        name=strcat(int2str(number_waypoints(1,z)),' waypoints(cities)');
        title(name);
        dist_matrix=zeros(number_waypoints(1,z),number_waypoints(1,z));
        for i=1:number_waypoints(1,z)
            point_x=waypoints_x(1,i);
            point_y=waypoints_y(1,i);
            for j=i+1:number_waypoints(1,z)
                point_x2=waypoints_x(1,j);
                point_y2=waypoints_y(1,j);
%                 points_number=4;
%                 points_x=round(linspace(point_x,point_x2,points_number));
%                 points_y=round(linspace(point_y,point_y2,points_number));
                connectable=1;
%                 for t=1:points_number
%                     if(configuration_space_combined_origina(100-points_x(1,t),points_y
        (1,t))==0&&100-points_x(1,t)~=15&&points_y(1,t)~=80)
%                         connectable=0;
%                         break;
%                     end
%                 end
                if connectable==1
                    dist_matrix(i,j)=sqrt((point_x-point_x2)^2+(point_y-point_y2)^2);
                else
                    dist_matrix(i,j)=inf;
                end
            end
        end
        dist_matrix1=dist_matrix';
        dist_matrix=dist_matrix+dist_matrix1;
        sol_best=cse276hw5_SA(dist_matrix,waypoints_x,waypoints_y);
        figure(10+z)
        plot(waypoints_y,waypoints_x,'*');
        hold on
        text(80,15,'start point')
        for i=1:number_waypoints(1,z)-1
            plot([waypoints_y(1,sol_best(1,i)),waypoints_y(1,sol_best(1,i+1))],[
                waypoints_x(1,sol_best(1,i)),waypoints_x(1,sol_best(1,i+1))],'-k');
        end
        x1=[0,6,6,0];
        y1=[0,0,100,100];
        x2=[0,0,100,100];
        y2=[100,97,97,100];
        x3=[93,100,100,93];
        y3=[0,0,100,100];
        x4=[0,0,100,100];
        y4=[0,7,7,0];
        fill(x1,y1,'k')
        fill(x2,y2,'k')
        fill(x3,y3,'k')
        fill(x4,y4,'k')
        name=strcat(int2str(number_waypoints(1,z)),' waypoints(cities)');
        name=strcat('The shortest path for',32,name);
        title(name);
        distance{z}=dist_matrix;
    end
    %get the distance matrix
```

```
function [sol_best] = cse276hw5_SA( dist_matrix,waypoints_x,waypoints_y)
a=0.999;
t0=200;
tf=3;
t=t0;
Markov_length=10000;
amount=size(dist_matrix,1);
sol_new=1:amount;
E_current=inf;
E_best=inf;
sol_current=sol_new;
sol_best=sol_new;
p=1;
while t>=tf
    zz=0;
    for r=1:Markov_length
        if(rand<0.5)
            ind1=0; ind2=0;
            while(ind1==ind2)
                ind1=ceil(rand.*amount);
                ind2=ceil(rand.*amount);
            end
            tmp1=sol_new(ind1);
            sol_new(ind1)=sol_new(ind2);
            sol_new(ind2)=tmp1;
        else
            ind1=0;ind2=0;ind3=0;
            while (ind1==ind2) || (ind2==ind3) || (ind3==ind1) || abs(ind1-ind2)==1
                ind1=ceil(rand.*amount);
                ind2=ceil(rand.*amount);
                ind3=ceil(rand.*amount);
            end
            tmp1=ind1;
            tmp2=ind2;
            tmo3=ind3;
            if  ind3<ind2 && ind1<ind3
                tmp2=ind2;ind2=ind3;ind3=tmp2;
            end
            if  ind1<ind2 && ind3<ind1
                tmp3=ind3;tmp1=ind1;tmp2=ind2;
                ind1=tmp3;ind2=tmp1;ind3=tmp2;
            end
            if ind2<ind3 && ind3<ind1
                tmp3=ind3;tmp1=ind1;tmp2=ind2;
                ind1=tmp2;ind2=tmp3;ind3=tmp1;
            end
            if ind3<ind2 && ind2<ind1
                tmp3=ind3;tmp1=ind1;tmp2=ind2;
                ind1=tmp3;ind2=tmp2;ind3=tmp1;
            end
            if ind2<ind1 && ind1<ind3
                tmp1=ind1;tmp2=ind2;
                ind1=tmp2;ind2=tmp1;
            end

            temlist1=sol_new((ind1+1):(ind2-1));
            sol_new((ind1+1):(ind1+ind3-ind2+1))=sol_new((ind2):(ind3));
            sol_new((ind1+ind3-ind2+2):ind3)=temlist1;
        end
        E_new=0;
        for i=1:amount-1
             E_new=E_new+dist_matrix(sol_new(i),sol_new(i+1));
        end
```

```
        E_new=E_new+dist_matrix(sol_new(amount),sol_new(1));
        for l=1:52
            if sol_new(l) == 1
                zz=1;
            end
        end
         if E_new < E_current && zz==0
            E_current=E_new;
            sol_current=sol_new;
                if E_new < E_best
                     E_best=E_new;
                     sol_best=sol_new;
                end
        else
            if rand < exp(-(E_new-E_current)./t)
                E_current=E_new;
                sol_current=sol_new;
            else
                sol_new=sol_current;
            end
        end
    end
    t=t.*a;
end
disp('The best solution:');
disp(sol_best);
disp('The shortest length:');
disp(E_best);
% % figure(20)
% han=eye(amount);
% coordinates=linspace(1,amount,amount)';
% coordinates=[coordinates,waypoints_y',waypoints_x'];
% for t=1:amount
%     han(t,sol_best(t))=1;
% end
% size(coordinates);
% size(han);
% gplot(han,coordinates,'-*');
end
```