2017

ECE 172A

PROJECT 2 REPORT

SATYAM PATEL A99053252

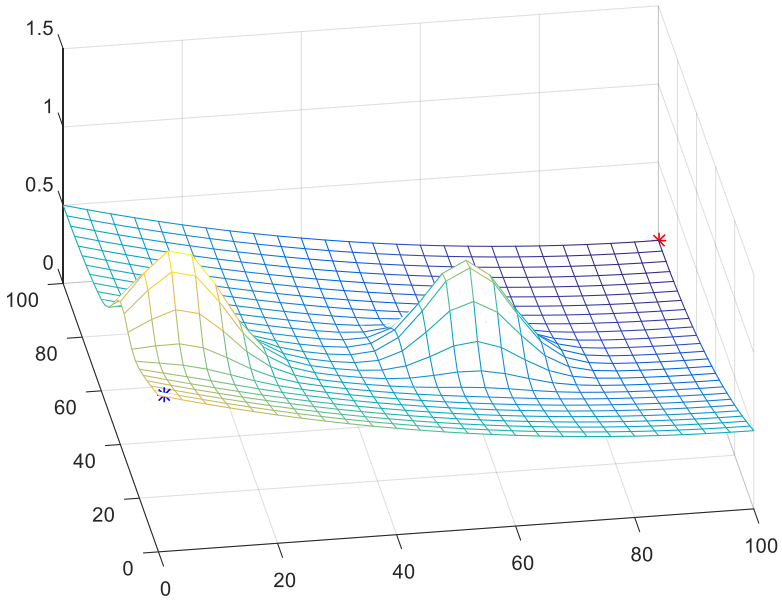
Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind. By including this in my report, I agree to abide by the Academic Integrity Policy mentioned above.

Problem 1

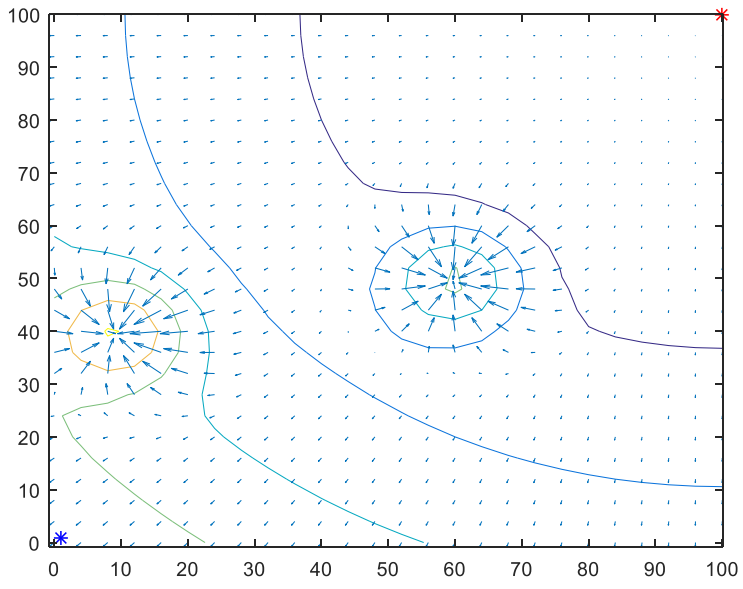
Bot guidance using potential fields

The purpose of this problem was to show an alternative approach to the sense-act paradigm that was more robust and simpler to implement.

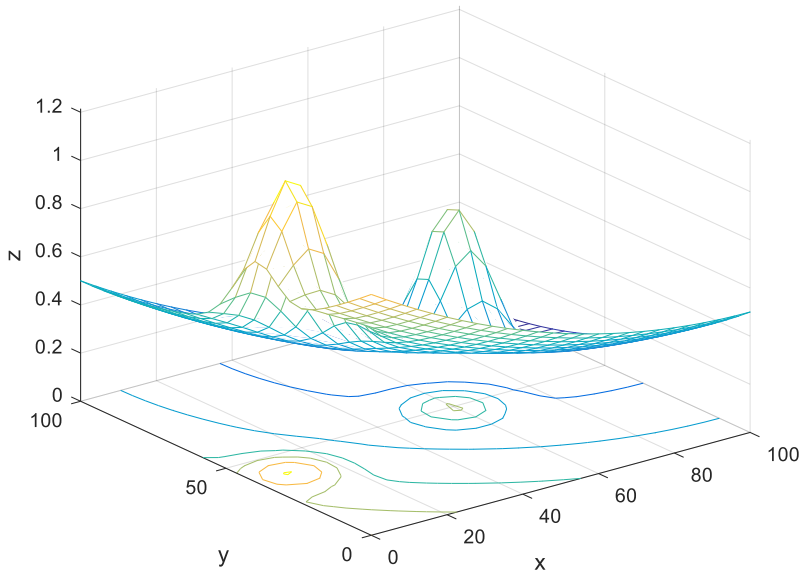
(a) Plots of contour and mesh:



a.



b.



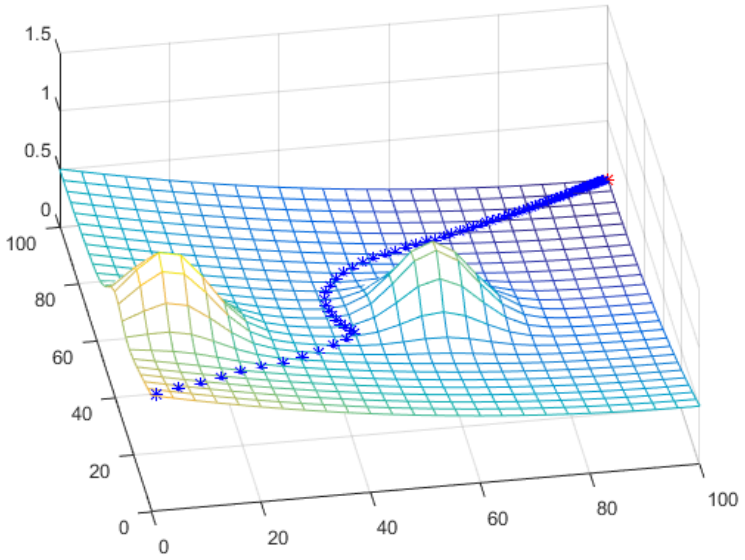
c.

Plot a is the terrain. Plot b is the contour with quivers (gradient) superimposed. Plot c is the terrain and contour plots superimposed. Starting point is marked with a blue asterisk and ending point with red.

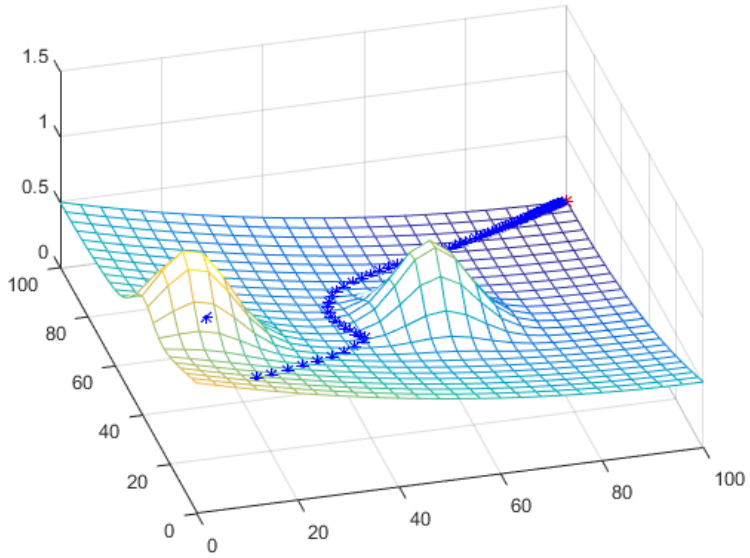
The obstacles are located at the means of each Gaussian. They look the way they do because the Gaussian is an upside down parabolic shape that decays exponentially. From the quiver plot we see that the gradient decreases in magnitude as we approach the destination point.

(b) The reason why the gradient descent method is superior to the sense-act paradigm is due to the robustness and simplicity of the actions involved. With a relatively simple algorithm the gradient descent method works for a huge variety of obstacle placements without specific cases needed. The sense-act paradigm however is highly terrain dependent. The GD algorithm is basically adding a scalar multiple of the gradient at the current location to the current location iteratively until the new location is close to the destination location. Since the overall directional tendency of the gradient is toward the destination, and the obstacles have a repulsive gradient effect, the bot will eventually end up close enough to the destination.

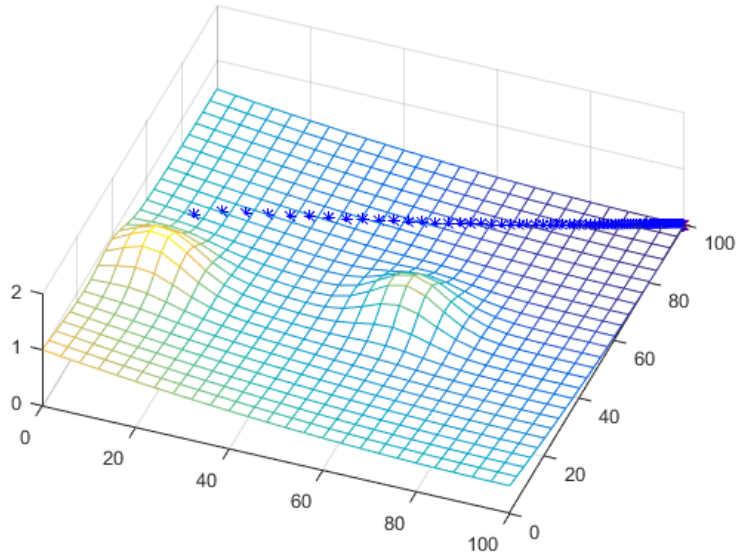
(c) Figures of various cases:


1.5
1
0.5
0
100
80
60
40
20
0
0
20
40
60
80
100

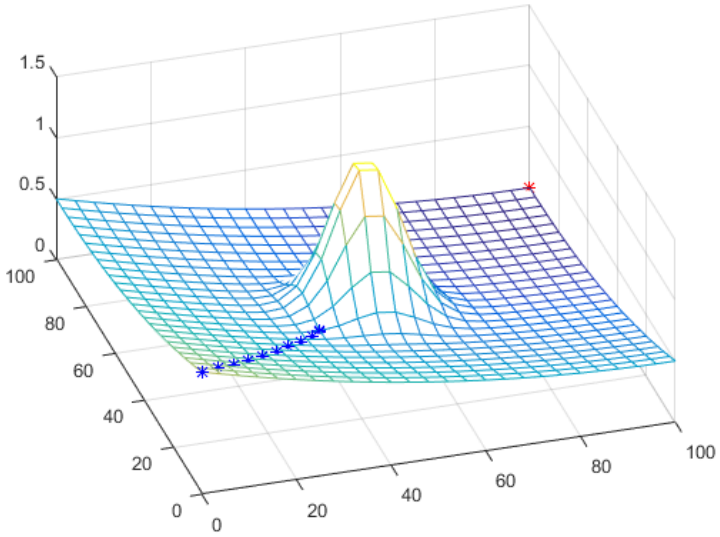
a. (0,0)


1.5
1
0.5
0
100
80
60
40
20
0
0
20
40
60
80
100

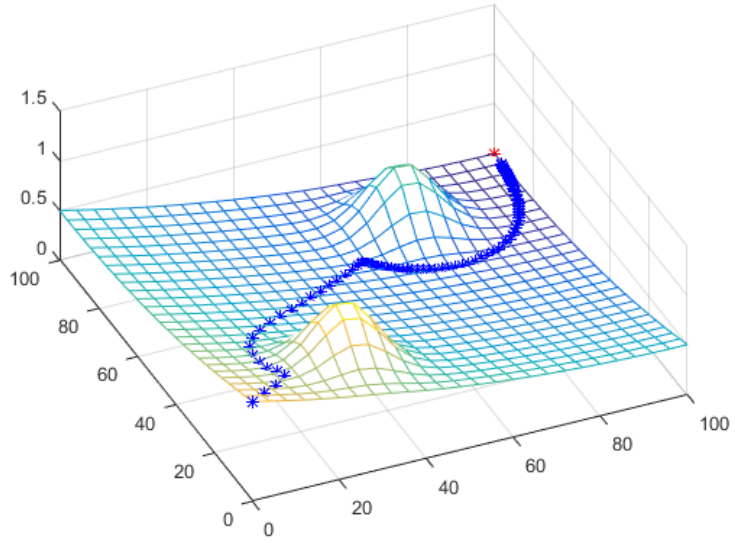
b. (10,30)


2
1
0
0
20
40
60
80
100
0
20
40
60
80
100

c. (10,60)


1.5
1
0.5
0
100
80
60
40
20
0
0
20
40
60
80
100

d.



e.

Figure a *is the standard path with the given obstacle locations and initial position.* ***Figure b*** *is the same obstacle locations with the initial point being (10,30). Plot* ***c*** *is the same as b but with initial point (10,60).* ***Figure d*** *is an unsuccessful case with the obstacles directly in the path of least resistance such that the gradient is parallel to the path at all times.* ***Figure e*** *is the path with the initial position (0,0) and obstacles at (30,20) and (70,80).*

Problem 2

Robotic swarm management

The purpose of this problem was to show an example of an intelligent way to manage multiple bots exploring an area.

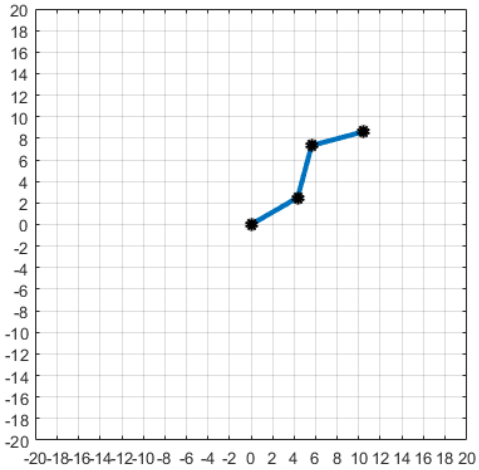
Code is in Appendix.

Problem 3

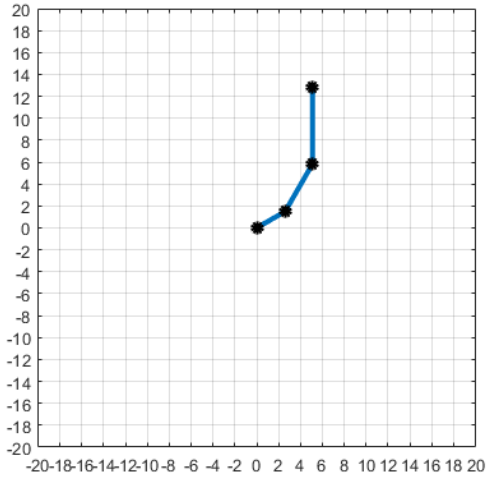
Forward and inverse kinematics of robotic arms

This problem demonstrated the incredible difference in complexity between solving forward problems and solving inverse problems.

(a) Figures of forward kinematics for two cases:



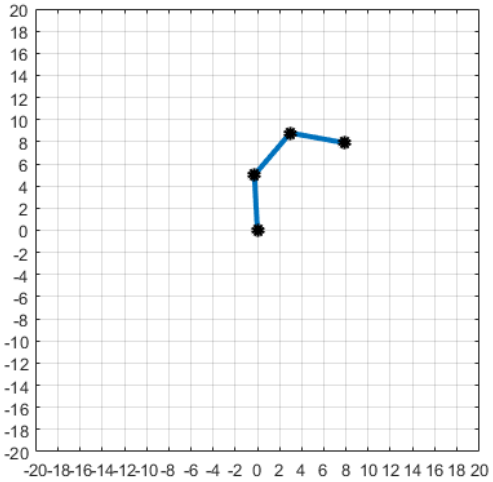
a.



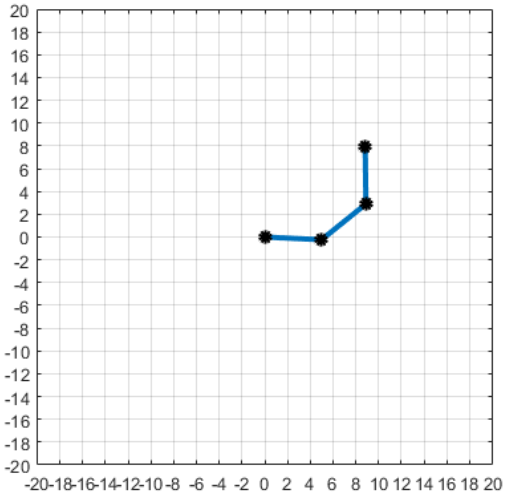
b.

Figure a *is the arm with initial conditions:* $\vartheta 0 = \pi/6, \vartheta 1 = \pi/4, \vartheta 2 = -\pi/3, l1 = 5, l2 = 5, l3 = 5$. ***Figure b*** *is the arm with initial conditions* $\vartheta 0 = \pi/6, \vartheta 1 = \pi/6, \vartheta 2 = \pi/6, l1 = 3, l2 = 5, l3 = 7$.

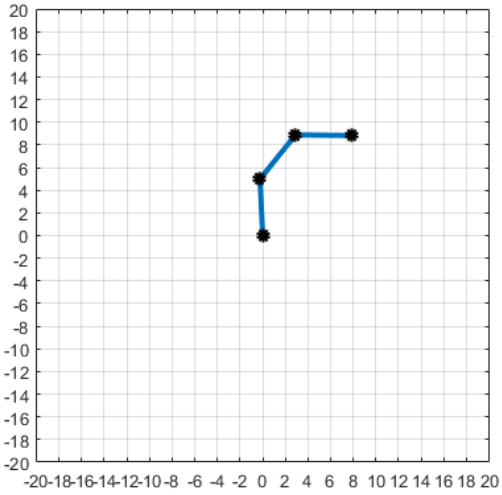
(b) Inverse Kinematics images and plots


20 18 16 14 12 10 8 6 4 2 0 -2 -4 -6 -8 -10 -12 -14 -16 -18 -20
-20 -18 -16 -14 -12 -10 -8 -6 -4 -2 0 2 4 6 8 10 12 14 16 18 20

a.


20 18 16 14 12 10 8 6 4 2 0 -2 -4 -6 -8 -10 -12 -14 -16 -18 -20
-20 -18 -16 -14 -12 -10 -8 -6 -4 -2 0 2 4 6 8 10 12 14 16 18 20

b.


20 18 16 14 12 10 8 6 4 2 0 -2 -4 -6 -8 -10 -12 -14 -16 -18 -20
-20 -18 -16 -14 -12 -10 -8 -6 -4 -2 0 2 4 6 8 10 12 14 16 18 20

c.

Figure a *is the solution with initial conditions set to zero.* ***Figure b*** *is the solution with initial conditions set to (pi/6, 0, 0) for theta 0, theta 1, theta 2 respectively.* ***Figure c*** *is the solution with initial conditions set to (pi/3, 0, 0).*

Appendix

Problem 1 code

```
clc; clear; close all;

initial_loc = [0,0];
cur_loc = initial_loc;
final_loc = [100,100];
mu = [30, 20; 70, 80];
sigma = [50, 0; 0, 50];

% Set up vector potential field
syms u v; % define symbolic variables to denote the coordintes on the potential field

% Symbolic equation that represents the potential field at any point (u, v)
z_sym(u, v) = (((final_loc(1,1)-u).^2 + (final_loc(1,2)-v).^2)/20000) + ...,
    1e4*(1/(det(sigma)*2*pi))*exp(-(1/2) .* [(u-mu(1,1)); (v-mu(1,2))]'*pinv(sigma)*[(u-mu(1,1));
(v-mu(1,2))]) + ...,
    1e4*(1/(det(sigma)*2*pi))*exp(-(1/2) .* [(u-mu(2,1)); (v-mu(2,2))]'*pinv(sigma)*[(u-mu(2,1));
(v-mu(2,2))]);

% Calculate symbolic derivatives of the field with respect to the coordinates u and v
dzdu_sym = diff(z_sym, u);
dzdv_sym = diff(z_sym, v);

% Convert symbolic functions into MATLAB functions for ease of use
z = matlabFunction(z_sym);
dx = matlabFunction(dzdu_sym);
dy = matlabFunction(dzdv_sym);
[x, y] = meshgrid(0:4:100);
```

Part 1

```
%The obstacles are gaussians and are located at their respective Means.
%The gradient of a gaussian looks like that
%the gradient becomes smaller in magnitude
figure();
meshc(x, y, z(x,y)); hold on;
xlabel('x');
ylabel('y');
zlabel('z');

% Above, we plot the mesh and contour plot as a single figure.
%
% a) Plot the contour plot and mesh of the vector field as separate figures.
% Useful functions are contour, mesh, surf, meshc, etc.
% Use x, y, and z(x, y) to plot the vector field.
figure
mesh(x,y,z(x,y))
hold on;
plot3(initial_loc(1),initial_loc(2),z(initial_loc(1),initial_loc(2)),'b*:');
```

```
plot3(100,100,z(100,100),'r*:');

figure
contour(x,y,z(x,y))
hold on;
% b) Plot the gradient (dx, dy) as quivers over the contour plot
% Use x, y, dx(x,y), and dy(x,y) to plot the quivers
quiver(x,y,dx(x,y),dy(x,y))

% c) Indicate where the bot will begin, blue asterisk, and where the
% bot will finish, red asterisk. (Look through the code).
plot(initial_loc(1),initial_loc(2),'b*:');
plot(100,100,'r*:');
hold off;
```

Part 2

Implement the algorithm from the discussion slides to control the bot through the vector field helping it reach it's final location. Plot each location it passes through as a blue asterisk, the final image should be the complete trajectory the bot takes.

```
%Press any key to begin t he animation
pause;

for i=1:100

    % This line must be deleted and replaced with your gradient descent update,
    % it's here to show you how the bot might move.
    cur_loc = cur_loc - 500*[dx(cur_loc(1),cur_loc(2)),dy(cur_loc(1),cur_loc(2))];

    %Plot the bots current location on the mesh.
    plot3(cur_loc(1,1), cur_loc(1,2), z(cur_loc(1,1), cur_loc(1,2)), '*b');
    pause(.1);
end


hold off;
```

Problem 2 code

```
function unexplored_areas = get_unexplored_areas(explore_map, UNMAPPED)
% Write this function so that unexplored_areas is a Nx2 matrix
% where N is the number of locations the bots have not explored, and each
% row of unexplored_areas represents a location (row,col) in the map.
% We define unexplored areas as locations which have values UNMAPPED in
% explore_map. Locations with values PLANNED/MAPPED/WALL are not considered
% unexplored areas in this case. If there are no unexplored areas, then
% unexplored_areas should return empty [], i.e. unexplored_areas = [];

sizeofmap = [size(explore_map,1),size(explore_map,2)];
linindices = find(explore_map==UNMAPPED);
```

```
if isempty(linindices)
    unexplored_areas = [];
    return
end
[i,j] = ind2sub(sizeofmap,linindices);
unexplored_areas = [i,j];

end
```

```
function dest = get_new_destination(curPos, unexplored_areas)
% Write this function so that it will pick the closest unexplored area
% as the new destination dest. We will keep this function simple by
% ignoring any walls that may block our path to the new destination. Here
% we define "closest" using the euclidean distance measure,
% e.g. sqrt((x1-x2)^2 + (y1-y2)^2).

% The lines below are not part of the solution and are only written here
% so that runMe.m can actually run without having written the functions yet
%dest = unexplored_areas(randi(size(unexplored_areas,1)),:);

sizeofunexplored = size(unexplored_areas,1);
distances = zeros(sizeofunexplored,1);
for i = 1:sizeofunexplored
    distances(i) = sqrt((curPos(1)-unexplored_areas(i,1))^2 + (curPos(2)-
unexplored_areas(i,2))^2);
end
minindex = find(distances==min(distances));
dest = unexplored_areas(datasample(minindex,1),:);
end
```

```
function explore_map = update_explore_map(dest, route, explore_map, PLANNED, UNMAPPED)
% Write this function so that all the locations specified in route and dest
% are marked as PLANNED only if it was previous UNMAPPED in the explore_map
% variable.

if explore_map(dest) == UNMAPPED
    explore_map(dest) = PLANNED;
end

for i=1:size(route,1)
    if explore_map(route(i,:)) == UNMAPPED
        explore_map(route(i,:)) = PLANNED;
    end
end

end
```

```
function [curPos, route, dest, explore_map] = update_position(curPos, route, dest, explore_map,
MAPPED)
% Write this function so that:
% 1) update curPos 1 step closer to the destination using route
% 2) mark the new location of the bot as MAPPED
% 3) if the new location of the bot is at the destination, set destination
%    to be empty, i.e. dest = []
% 4) update the route by removing the location that the bot was just
%    updated to e.g. if route was inputted as an Nx2 matrix, it should
%    output as a (N-1)x2 matrix.


curPos = route(1,:);
explore_map(route(1,1),route(1,2)) = MAPPED;
if curPos == dest
    dest = [];
end
route = route(2:end,:);


end
```

Problem 3 Code

```
%
forward kinematics for a robot arm in R2 with 3 joints
%}

function [x_1,y_1,x_2,y_2,x_e,y_e] = ForwardKinematics(l0,l1,l2, theta0, theta1, theta2)

x_1 = l0*cos(theta0);
y_1 = l0*sin(theta0);
x_2 = x_1+l1*cos(theta1+theta0);
y_2 = y_1+l1*sin(theta1+theta0);
x_e = x_2+l2*cos(theta2+theta1+theta0);
y_e = y_2+l2*sin(theta2+theta1+theta0);

end
```

Published with MATLAB® R2015b

```
% calculates the jacobian matrix
% inputs [l0,l1,l2,theta0,theta1,theta2]
% outputs [dxdt0,dxdt1,dxdt2,dydt0,dydt1,dydt2]
```

```
function [dxdt0,dxdt1,dxdt2,dydt0,dydt1,dydt2]=jacobian(l0,l1,l2,theta0,theta1,theta2)
    dxdt0 = -l0*sin(theta0)-l1*sin(theta0+theta1)-l2*sin(theta0+theta1+theta2);
    dxdt1 = -l1*sin(theta0+theta1)-l2*sin(theta0+theta1+theta2);
    dxdt2 = -l2*sin(theta0+theta1+theta2);
    dydt0 = l0*cos(theta0)+l1*cos(theta0+theta1)+l2*cos(theta0+theta1+theta2);
    dydt1 = l1*cos(theta0+theta1)+l2*cos(theta0+theta1+theta2);
    dydt2 = l2*cos(theta0+theta1+theta2);
end
```

```
%Run each section separately!
%Script used to test problem 3 functions
close all
%pick some theta values to draw
theta0 = pi/6;
theta1 = pi/4;
theta2 = -1*pi/3;
l0=5;
l1=5;
l2=5;
[x_1,y_1,x_2,y_2,x_e,y_e] = ForwardKinematics(l0,l1,l2,theta0,theta1,theta2);
drawRobot(x_1,y_1,x_2,y_2,x_e,y_e);
```

```
%do it again for some other values
theta0 = pi/6;
theta1 = pi/6;
theta2 = pi/6;
l0=3;
l1=5;
l2=7;
[x_1,y_1,x_2,y_2,x_e,y_e] = ForwardKinematics(l0,l1,l2,theta0,theta1,theta2);
drawRobot(x_1,y_1,x_2,y_2,x_e,y_e);
```

```
%Inverse Kinematics
close all
[theta0_target, theta1_target, theta2_target] = InverseKinematics(5,5,5,8,8)
```

Published with MATLAB® R2015b