function [botSim] = localise(botSim,map,target)

%% Localisation code

modifiedMap = map; %you need to do this modification yourself

botSim.setMap(modifiedMap);

targetp=BotSim(modifiedMap);

targetp.setBotPos(target);

travellist=[];

scannum = 18;

botSim.setScanConfig(botSim.generateScanConfig(scannum));

cycle=0;

num =300; % number of particles

clusnum=floor(size(map,1)/2)-1; %assumed number of clusters

if clusnum<2 %this can't be less than two

clusnum=2;

end

isolated=1;

while isolated==1

%generate some random particles inside the map

particles(num,1) = BotSim; %how to set up a vector of objects

newparticles(num,1) = BotSim;

position=[];

heading=[];

for i = 1:num

particles(i) = BotSim(modifiedMap); %each particle should use the same map as the botSim object

newparticles(i)= BotSim(modifiedMap);

particles(i).randomPose(0); %spawn the particles in random locations

position=[position;particles(i).getBotPos()];

heading=[heading;particles(i).getBotAng()];

end

[clusindex, roompos]=kmeans(position,clusnum); %split particles into clusters and get rooms

if botSim.debug()

disp('clustering')

clusnum

num

roompos

end

adjacent=zeros(clusnum);

for j=1:clusnum

for i=1:clusnum

if i==j

score=0;

else

%determine if rooms can "see" each other

newparticle=BotSim(modifiedMap);

newparticle.setBotPos(roompos(j,:));

C=roompos(i,:)-roompos(j,:);

dist=norm(C);

theta=atan2(C(2),C(1));

newparticle.setBotAng(theta);

pscan=newparticle.ultraScan();

if pscan(1)>=dist

score=1;

else

score=0;

end

end

adjacent(i,j)=score;

end

end

A=sum(adjacent, 2);

if find(A==0) %a "room" is isolated. increase clusters

if botSim.debug()

disp('isolated')

adjacent

end

clusnum=clusnum+1;

num=num+50;

else

isolated=0;

end

[targetdist, targetroom]=min(pdist2(roompos,target)); %find out which room target is in

E=roompos(targetroom,:)-target;

Enorm=E/norm(E);

theta=atan2(Enorm(2),Enorm(1));

targetp.setBotAng(theta);

pscan=targetp.ultraScan();

if pscan(1)<=norm(target-roompos(targetroom,:))

if botSim.debug()

disp('target misplaced')

end

%target is in the wrong room!

clusnum=clusnum+1;

num=num+50;

isolated=1; %redo reseed

end

if isolated==0

% do wavefront/djvorak here

visited=[];

unvisited=1:size(roompos,1);

current=targetroom;

distancematrix=squareform(pdist(roompos));

djmatrix=zeros(size(roompos));%not really but they are same size

djmatrix(:,1)=Inf;

djmatrix(current,1)=0;

while ~isempty(unvisited)

%djorak

if botSim.debug()

disp('pathing')

unvisited

end

[dist, index]=min(djmatrix(unvisited,1)); %find minimum dist in unvisited

current=unvisited(index);

visitlist=find(adjacent(current,:)); %gives indexes adjacent to current

[value, index]=intersect(visitlist,visited);

visitlist(index)=[]; %kill off nodes already visited

totaldistance=dist+distancematrix(current,visitlist);

for index=find(totaldistance'<djmatrix(visitlist,1));

djmatrix(visitlist(index),1)=totaldistance(index);

djmatrix(visitlist(index),2)=current;

end

visited=[visited;current];

unvisited(unvisited==current)=[];

end

if ~isfinite(sum(djmatrix(:,1)))

isolated=1

end

end

end

roompos

adjacent

djmatrix

maxNumOfIterations = 200;

n = 0;

converged =0; %The filter has not converged yet

particleScan=[];

mparticleScan=[];

rotation=[];

travelto=[0,0];

while(converged == 0 && n < maxNumOfIterations) %%particle filter loop

n = n+1; %increment the current number of iterations

botScan = botSim.ultraScan(); %get a scan from the real robot.

[maxbS,maxbI]=max(botScan); %identify index with max

%% Write code for updating your particles scans

if botSim.debug()

disp('scanning')

end

% for i =1:num %for all the particles.

% particles(i).setScanConfig(particles(i).generateScanConfig(scannum));

% particleScan(:,i) = particles(i).ultraScan();

% [maxpS,maxpI]=max(particleScan(:,i));

% rotation(i)=maxbI-maxpI; %rotation matrix

% mparticleScan(:,i)=circshift(particleScan(:,i),rotation(i)); %align maxes

% end

%

for i=1:num

particles(i).setScanConfig(particles(i).generateScanConfig(scannum));

particleScan(:,i) = particles(i).ultraScan();

% [maxpS,maxpI]=max(particleScan(:,i));

% rotation(i)=maxbI-maxpI; %rotation matrix

[xcor,lag]=xcorr(botScan,particleScan(:,i));

[val, idx]=max(xcor);

rotation(i)=lag(idx);

mparticleScan(:,i)=circshift(particleScan(:,i),rotation(i)); %align maxes

end

%% Write code for scoring your particles

if botSim.debug()

disp('scoring')

end

%remove not finite scans

%we don't actually need to do this but \*shrug\*

index=find(isfinite(sum(particleScan,1))==0);

particleScan(:,index)=[];

mparticleScan(:,index)=[];

rotation(index)=[];

particles(index)=[];

particleScore=sqrt(sum((mparticleScan-botScan).^2,1));

mparticleScore=particleScore.^-2; %maybe think of adjusting this

% particleScore=sqrt(sum((mparticleScan-botScan).^2,1));

% mparticleScore=normpdf(particleScore,0,std(particleScore));

%rotation penalty

rotationpenalty=1-cos(rotation\*2\*pi/scannum);

mparticleScore=mparticleScore.\*(1-0.4\*rotationpenalty);

prob=mparticleScore/sum(mparticleScore);

index = find(prob==0); %finds the indices which are 0

cumprob=-1;

particleScore(index) = []; %remove infinites

particles(index)=[];%kill particles with inf. Outsiders!

rotation(index)=[]; %particles, its score, its rotation still the same

prob(index)=[];

cumprob=cumsum(prob);

%% Write code for resampling your particles

weight=zeros(num,1)+1;

U=1./num;

B=rand\*2\*max(prob);

if B>1

B=B-1;

end

if sum(cumprob)>0

if botSim.debug()

disp('respawning')

end

for i=1:num %all the particles to respawn

index=find(B<=cumprob,1);

B=B+U;

if B>1

B=B-1;

end

newposition=particles(index).getBotPos()+0.2\*rand(1,2);

newheading=particles(index).getBotAng()+rotation(index)\*2\*pi/scannum+0.05\*randn;

newparticles(i).setBotPos(newposition);

newparticles(i).setBotAng(newheading);

weight(i)=prob(index)+0.1;

end

else

if botSim.debug()

disp('oops fuckup happened')

end

for i=1:num %all the particles to respawn

newparticles(i).randomPose(5);

weight(i)=1;

end

end

particles=newparticles;

weight=weight/sum(weight); %returns prob total 1

%% Write code to check for convergence

position=[];

heading=[];

for i=1:num

position=[position;particles(i).getBotPos()];

heading=[heading;particles(i).getBotAng()];

end

botPosEs=sum(position.\*weight);

botAngEs=mod(sum(heading.\*weight),2\*pi);

if botSim.debug()

botPos=botSim.getBotPos();

botAng=mod(botSim.getBotAng(),2\*pi);

disp(n)

disp('where I think I am ')

disp(botPosEs)

disp(botAngEs)

disp('where I really am')

disp(botPos)

disp(botAng)

derror=norm(botPosEs-botPos)

herror=(botAngEs-botAng)/botAng

end

variance=norm(var(position));

xydistrib=norm(botPosEs-mean(position)); %this can work as convergence!disp('scoring')

%% Write code to take a percentage of your particles and respawn in randomised locations (important for robustness)

five=floor(0.05\*num);

for i=1:five %five percent

index=floor(num\*rand)+1;

particles(index)=[]; %kills it

newparticle = BotSim(modifiedMap);

newparticle.randomPose(min([min(botScan),10]));

particles=[particles;newparticle];

end

%% Write code to decide how to move next

confident=0;

if and(xydistrib<1, variance<250)

confident =1; % the bot knows where it is! I hope

end

if confident==0

if botSim.debug()

disp('localising')

end

%robot is lost

%move randomly to localize

if mod(cycle,5)==4

turn = (maxbI-1)\*2\*pi/scannum;

move = min([rand\*(maxbS-10),30]);

else

turn=0.5;

move= 5;

end

cycle=cycle+1;

centered=0; %is the robot at a node

travelto=[-10,-10];

else

%robot kinda knows where it is

%Lets try djestra or whatever

%nodes are roompos

cycle=0;

arrived=0;

%find out robo room

[robodist,roboroom]=pdist2(roompos,botPosEs,'euclidean','Smallest',1);

current=roboroom;

%is robot at a node

if norm(botPosEs-travelto)<2

centered=1;

previous=roboroom;

if travelto==target

converged=1;

end

end

%where will robo go

if centered==0

%robot has not centered into a node after localising

travelto=roompos(roboroom,:);

else

if previous==targetroom

travelto=target;

else

travelto=roompos(djmatrix(previous,2),:);

end

end

Z=travelto-botPosEs;

Zdist=norm(Z);

Zdir=atan2(Z(2),Z(1))-botAngEs;

move=min([Zdist,8]);

turn=Zdir;

if botSim.debug()

disp('attempting to move to')

travelto

move

turn

end

end

%collision detection

collide=botScan<9;

collision=1;

while collision==1

dir=mod(turn+2\*pi,2\*pi)/(2\*pi/scannum);

dir=round(dir+1);

%I think I can solve this with a mod something but I'm lazy

if dir<1

dir=dir+scannum;

end

if dir>scannum

dir=dir-scannum;

end

if collide(dir)

turn=rand\*2\*pi;

move=3;

else

collision=0;

end

end

botSim.turn(turn); %turn the real robot.

botSim.move(move); %move the real robot. These movements are recorded for marking

for i =1:num %for all the particles.

particles(i).turn(turn); %turn the particle in the same way as the real robot

particles(i).move(move); %move the particle in the same way as the real robot

end

if ~botSim.insideMap()

converged=1;

disp('Bot outside!')

end

%% Drawing

%only draw if you are in debug mode or it will be slow during marking

if botSim.debug()

hold off; %the drawMap() function will clear the drawing when hold is off

botSim.drawMap(); %drawMap() turns hold back on again, so you can draw the bots

botSim.drawBot(30,'g'); %draw robot with line length 30 and green

for i =1:num

particles(i).drawBot(3); %draw particle with line length 3 and default color

end

%mostly because I do not know how to draw an x

%targetp.drawBot(10,'r');

%I have learned how to draw an x

plot(target(1),target(2), 'x');

plot(roompos(:,1),roompos(:,2), '+');

drawnow;

end

end

averageCompletionTime averageDisFromTgt averagePathLength percentCollision

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Map1 4.2702 3.6486 133.55 0.02

Map2 7.0188 3.7492 194.62 0.02

Map3 14.425 8.49 389.18 0.03

averageCompletionTime averageDisFromTgt averagePathLength percentCollision

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Map1 4.8926 3.9164 156.38 0.03

Map2 6.0772 4.219 180.41 0.05

Map3 10.95 6.7526 365.49 0.03

averageCompletionTime averageDisFromTgt averagePathLength percentCollision

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Map1 3.3941 3.0521 143.69 0

Map2 4.4019 2.6997 171.44 0.01

Map3 11.891 13.723 392.35 0.08

averageCompletionTime averageDisFromTgt averagePathLength percentCollision

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Map1 4.0973 3.4928 137.84 0.01

Map2 5.1927 5.6817 164.3 0.09

Map3 15.203 8.9026 422.17 0.05

averageCompletionTime averageDisFromTgt averagePathLength percentCollision

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Map1 3.9003 4.3786 150.6 0.02

Map2 4.7652 5.9458 188.87 0.08

Map3 12.46 5.2879 411.88 0.01