|  |  |
| --- | --- |
| . |  |



CSE 5280

Computer Graphics

Spring 2016

|  |
| --- |
| **Class Assignment-02**  **( Animation )** |

Student Name: \_\_\_\_\_**Jay Sandeepkumar Modi**\_\_\_\_

Student ID: **\_\_\_\_\_\_\_\_\_\_\_\_902292667\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |  |
| --- | --- |
|  | **Professor:** [Dr. Eraldo Ribeiro](https://fit.instructure.com/courses/452648/users/753896)  [eribeiro@cs.fit.edu](mailto:mcarvalho@fit.edu) |
|  |  |

1. **Simple motion path planning:**

**Code:**

function pathplanning()

%

% This program demonstrates a simple

% path-planning approach based on

% minimizing a cost function.

% Particle's initial position

particle(1).StartPosition = [ 10 10 ]';

% Particle's current position

particle(1).CurrentPosition = particle(1).StartPosition;

% This is the goal position.

particle(1).GoalPosition = [ 90 90 ]';

% (x,y) coordinates of the centroid of obstacles

o(1).location=[ 30 40 ]';

o(1).R=25;

o(2).location=[ 60 50 ]';

o(2).R=25;

o(3).location=[ 30 80 ]';

o(3).R=25;

o(4).location=[ 80 20 ]';

o(4).R=25;

% Advancement step for gradient descent

lambda = 3;

% We will store the calculated path locations

% in this array so we can plot them.

X = [];

% Termination condition

GoalReached = false;

% Plot initial location of particles

s = particle(1).StartPosition;

g = particle(1).GoalPosition;

figure,

plot( s(1), s(2),'ro', 'LineWidth', 2 ) % starting point

text( s(1)+2, s(2)-5, 'Start', 'FontSize', 12 );

hold on;

plot( g(1), g(2),'bo', 'LineWidth', 2 ) % goal point

text( g(1)+2, g(2)-5, 'Goal', 'FontSize', 12 );

axis([0 100 0 100 0 100]);

view([-20,40]);

%axis square;

set(gcf, 'Color', 'w' );

% Drawing Obstacles (filled circles).

filledCylinder(30,40,10);

filledCylinder(60,50,10);

filledCylinder(30,80,10);

filledCylinder(80,20,10);

% Initialize variables for locations

x = particle(1).StartPosition;

g = particle(1).GoalPosition;

% This loop calculates the new position of the particle.

% It repeats until goal is reached

n=1;

while ~GoalReached

% Calculate next step using gradient descent

x = x - lambda \* Grad( x, g, o );

% Store location. Concatenate the previous result with current's.

X = [ X x ];

% Plot particle at new location

hold on;

plot( x(1), x(2), 'g\*' ) % trajectory points

axis([0 100 0 100 0 1]);

frame=getframe;

im = frame2im(frame);

[imind,cm] = rgb2ind(im,256);

if n==1

imwrite(imind,cm,'pathplanning.gif','gif', 'Loopcount',inf);

n=2;

else

imwrite(imind,cm,'pathplanning.gif','gif','WriteMode','append');

end

% Pause for a moment so we can see the motion

pause( .1 );

% Check if goal has been reached, i.e., distance

% between current and goal locations is less than a

% pre-defined threshold.

if ( norm( x - g ) <= 3 )

GoalReached = true;

end

end

% Gradient vector (2-D direction) of cost function at current position

function G = Grad( p, g, o)

% p: 2x1 vector with the current position

% g: 2x1 vector with goal position

% Calculate the cost of moving to locations of a 4-size neighborhood

%

% ^

% |

% y1

% |

% -- x2 --- p --- x1 -->

% |

% y2

% |

%

y1 = p + [ 0; 1 ];

y2 = p + [ 0; -1 ];

x1 = p + [ 1; 0 ];

x2 = p + [ -1; 0 ];

% Calculate the components of the gradient vector

cx = Cpathplan( x1, g, o ) - Cpathplan( x2, g, o );

cy = Cpathplan( y1, g, o ) - Cpathplan( y2, g, o );

% Resultant vector formed by cost's x and y components

r = [ cx; cy ];

% Calculate the direction vector, i.e., direction of the gradient vector

G = r / norm( r );

function c = Cpathplan( p, g, o )

% Cost function for path planning calculated at position s with respect to

% goal g and obstacle o

%

% p: 2x1 vector with the current position

% g: 2x1 vector with goal position

% o(i).location: 2x1 vector with obstacle position

% o(i).R: Radius of the obstacle

% Value of log10E

logE = 2.718281828;

% Goal cost (Euclidean distance squared)

c( 1 ) = norm(p-g);

% Collision cost

field=0;

for i=1:size(o,2)

dist = sqrt( (p(1)-o(i).location(1))^2 + (p(2)-o(i).location(2))^2);

if 0 < dist && dist <= o(i).R

field\_temp = logE ^ (log( o(i).R / dist )\* logE);

else

field\_temp=0;

end

field = field + field\_temp;

end

%display(field);

c( 2 ) = field; % TODO (Equation 3.2 of Breen's paper)

% Total cost

c = c(1) + c(2);

return

function h = filledCylinder(x,y,r)

% Drawing filled cylinder in figure.

hold on

[X,Y,Z] = cylinder(r);

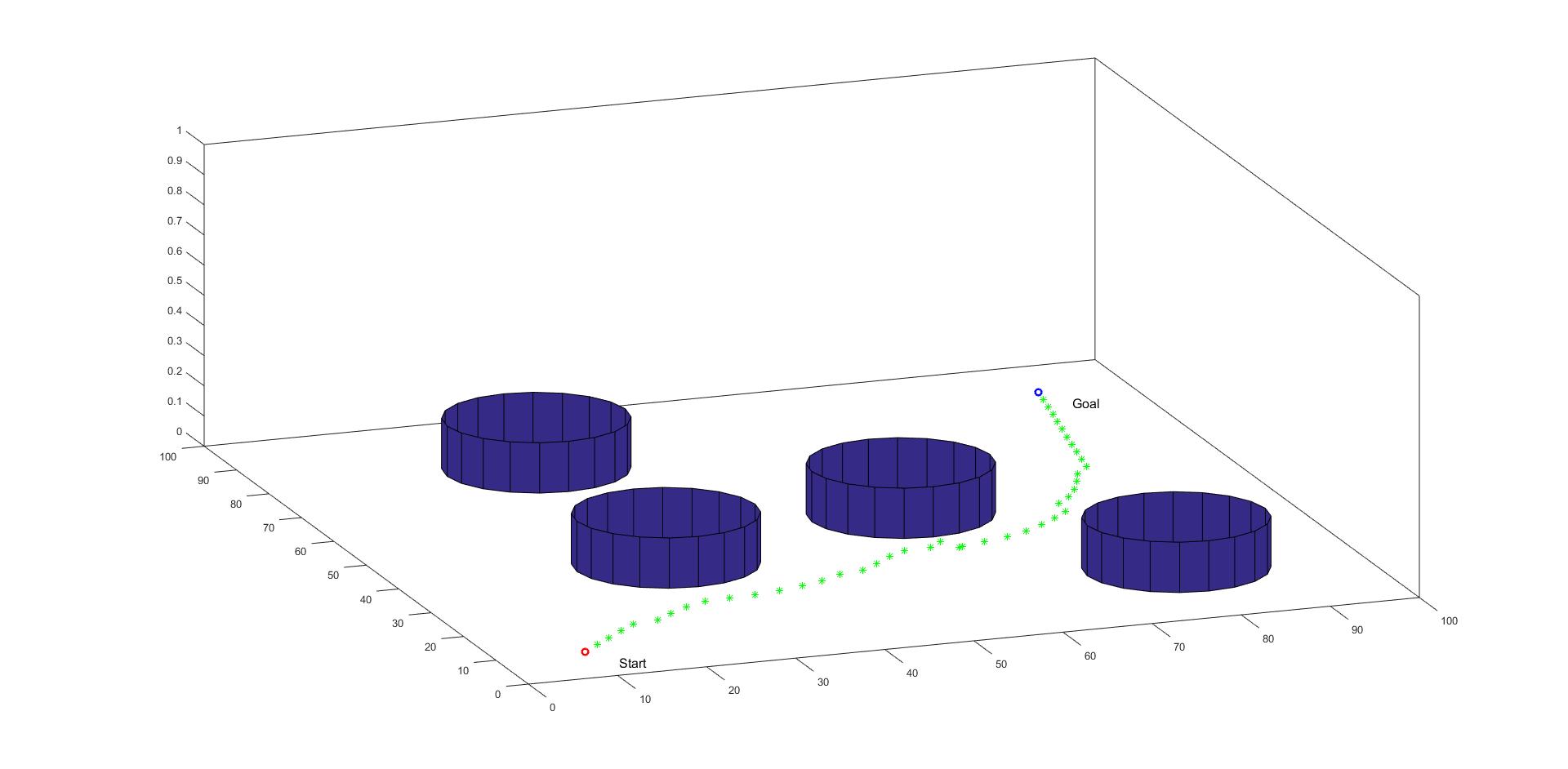
Z=Z/6;

surf(X+x,Y+y,Z)

hold off

return

**Result:**



1. **Simulation of Helbing's social-force model:**

**Code:**

function social\_force\_model()

close all;

%--------------------------------------------------------------------------

% The data structure System stores all the information about the scene.

% It makes it simpler to pass the whole system to the cost functions as

% they will need to know about the location of all particles and obstacles.

%--------------------------------------------------------------------------

% Particle 1

System.particles.x(:,1) = [ 0 40 ]'; % current location

System.particles.g(:,1) = [ 100 40 ]'; % goal location

System.particles.x(:,2) = [ 0 50 ]'; % current location

System.particles.g(:,2) = [ 100 50 ]'; % goal location

System.particles.x(:,3) = [ 0 60 ]'; % current location

System.particles.g(:,3) = [ 100 60 ]'; % goal location

% Particle 2

System.particles.x(:,4) = [ 40 0 ]'; % current location

System.particles.g(:,4) = [ 40 100 ]'; % goal location

System.particles.x(:,5) = [ 50 0 ]'; % current location

System.particles.g(:,5) = [ 50 100 ]'; % goal location

System.particles.x(:,6) = [ 60 0 ]'; % current location

System.particles.g(:,6) = [ 60 100 ]'; % goal location

% Colors for particle trajectories. Using matlab's "Lines" colormap.

System.C = colormap(lines);

% Minimum distance indicating that particle has reached its destination

System.mindist = 1;

% Acceptable personal space (radius) between particles

System.personal\_space = 5;

%--------------------------------------------------------------------------

% Start drawing scene (start and end points for particles)

DrawStartAndEndLocations( System );

axis([0 100 0 100]); % adjust limits of axes

% Step for gradient descent

lambda = 1;

GoalReached = false;

n=1;

while ~GoalReached

% Calculate next step using gradient descent.

% Simple two-particle system. Create a for-loop if using more particles

% display(size(System.particles.x,2));

for j=1:size(System.particles.x,2)

System.particles.x(:,j) = System.particles.x(:,j) - lambda \* Grad( System, j );

hold on;

plot( System.particles.x(1,j), System.particles.x(2,j), '.',...

'Color', System.C(j,:) );

drawnow;

% Has particle reached destination? Store true or false here.

reached( j ) = norm( System.particles.x(:,j) - System.particles.g(:,j) ) <= System.mindist;

end

if ( norm(System.particles.x(:,1) - System.particles.x(:,2) ) ) < 5

d = 0;

end

frame=getframe;

im = frame2im(frame);

[imind,cm] = rgb2ind(im,256);

if n==1

imwrite(imind,cm,'socialForceModel.gif','gif', 'Loopcount',inf);

n=2;

else

imwrite(imind,cm,'socialForceModel.gif','gif','WriteMode','append');

end

% let's pause for a moment so we can see the motion

pause( .05 );

% check if goal has been reached (this only works for two particles!!)

if ( reached( 1 ) && reached( 2 ) )

GoalReached = true;

end

end

% Gradient vector (2-D direction) of cost function at current position

function G = Grad( System, j )

% j: particle id

% location of particle of interest

p = System.particles.x(:,j);

% Calculate the cost of moving to locations of a 4-size neighborhood

%

% ^

% |

% y1

% |

% -- x2 --- p --- x1 -->

% |

% y2

% |

%

y1 = p + [ 0; 1 ];

y2 = p + [ 0; -1 ];

x1 = p + [ 1; 0 ];

x2 = p + [ -1; 0 ];

% Calculate the components of the gradient vector

cx = Cpathplan( System, x1, j ) - Cpathplan( System, x2, j );

cy = Cpathplan( System, y1, j ) - Cpathplan( System, y2, j );

% Resultant vector formed by cost's x and y components

r = [ cx; cy ];

% Calculate the direction vector, i.e., direction of the gradient vector

G = r / norm( r );

return

function c = Cpathplan( System, x, j )

% Cost function for path planning

%

% x: 2x1 vector with the position for which we want to calculate cost

% j: particle id: either 1 or 2 for a two-particle example

% Goal cost for particle j (Euclidean distance squared)

g = System.particles.g(:,j); % goal location for particle j

c( 1 ) = norm( x - g );

% Simple social force. It uses all particles except particle j

%c(2) = 0;

% Other cost ???

c( 3 ) = SocialForceCost( System, x, j ); % TODO

% Total cost

c = sum( c );

return

function DrawStartAndEndLocations( System )

% Just draws the start and end locations for visualization.

P = System.particles; % particles locations

C = System.C; % colors for plotting trajectories

% Number of particles in system

N = size( P.x, 2 );

for j = 1 : N

hold on;

% starting point

plot( P.x(1,j) , P.x(2,j), 'o', ...

'Color', C(j,:), 'LineWidth', 2 );

text( P.x(1,j)+.5, P.x(2,j), ...

sprintf('S%d', j ), 'FontSize', 12 );

hold on;

% goal point

plot( P.g(1,j) , P.g(2,j),'o', ...

'Color', C(j,:), 'LineWidth', 2 );

text( P.g(1,j)+.5, P.g(2,j), ...

sprintf('G%d', j ), 'FontSize', 12 );

end

return

function c = SocialForceCost( System, x, j )

% Simple social force (calculated for all particles except particle j)

%

% Input:

% System: System info (all particles)

% x: 2x1 vector with the position for which we want to calculate cost

% j: Id of the particle of interest

%

% Number of particles in system

N = size( System.particles.x, 2 );

% Set difference, e.g., [ 1 2 3 ] - [2] = [ 1 3 ]. This produces an array

% containing indices of all particles except particle j

idx = setdiff( 1:N , j );

OtherParticles = System.particles.x( :, idx );

% Radius of the region of influence of "personal space"

sigma = System.personal\_space;

field=0;

for i=1:size(OtherParticles,2);

dist=sqrt((x(1)-OtherParticles(1,i))^2+(x(2)-OtherParticles(2,i))^2);

% display(dist);

if 0 < dist && dist <= sigma

temp = log( sigma / dist);

else

temp=0;

end

field = field + temp;

end

c = field;

return

**Result:**

