```
function y = fcn(u1,u2)

%% #codegen
coder.extrinsic('clc');
persistent detSet;
if isempty(detSet)
detSet=zeros(1,3);
end
coder.varsize('detSet',[30000000 3]);

x_1=11.4591559045;
x_2 =11.45915590;


y_1=4e-10;
y_2=-4e-10;
%%Distance measures
dist_min=0;
temp_dist=0;
rDet=0;
c_s1 =(x_1+x_2)/2;
c_s2=(y_1+y_2)/2;

if (abs(y_1-y_2)>abs(x_1-x_2))
    rs =abs(x_1-x_2);
else
    rs =abs(y_1-y_2);
```

```
end
% center of self and the radious
circle(c_s1, c_s2,abs(y_1-y_2));
%drawCircle(c_s1, c_s2,rs);
%filledCircle([c_s1,c_s2],rs,1000,'r')
%Until N ==m_max
N=0; % count of not self
 %# of detectors


p=0.5; % default probability
x=0; % %Selected point /occarance
n=0; % sample size/ total number of data
m =0; % detected occurances
n_sample=10; % predefined sample size



% Detector set map(Key value pair diameter)
%detSet = containers.Map↙
('KeyType','double','ValueType','double');
% coder.extrinsic(detset);
% detSet = java.util.Hashtable;
% bool =exist(detSet);
% if bool
%detSet = zeros(1,3);
%detSet= [1 1 1];
%detSet=[detSet; o];
```

```
% end
% Estimated coverage 99%
%%%% Hyppthesis testing %%%%
m_max =max(5/p, 5/(1-p));
z_alpha =3.49;
% Sample a point
i=0;



% %%
% %two dimensional space

 a=u1;
 b =u2;

%  d=abs(sqrt(((a-c_s2).^2+(b-c_s1).^2))-rs);
%   e= [a d b];
%
%   detSet=[detSet; e]
%   y=detSet
%
%  if(abs (sqrt(((b-c_s2)^2+(a-c_s1)^2))-rs))↙
>rs % Euclidean distance
%         %%--------not-self-------
%         %%
%         N=N+1; % count not self sample
%
%             % addthe first detector to the↙
```

```
map
%              c=abs(sqrt(((b-c_s2)^2+(a-c_s1)↙
^2))-rs);
%              b= [a b c];
%              detSet=[detSet; b];
%              [rows cols] =size(detSet);
%  end

 if(a>0.1 && b>0.1) % to reduce the noise ???
if (N<m_max) % exit criterion
   n =n+1; % count the sample size
   if(abs (sqrt(((b-c_s2)^2+(a-c_s1)^2))-rs))↙
>rs % Euclidean distance -non self
        %%--------not-self-------
        %%
        N=N+1; % count (non-self) sample

        if size(detSet,1)==1 % first detector
            % addthe first detector to the map
            c=sqrt(((b-c_s2)^2+(a-c_s1)^2))-↙
rs;
            detSet=[detSet; [a b c]];
            %y=detSet;
            %xlswrite('detectors.xlsx',↙
detSet);

            circle(a,b,c);
```

```
        else
        %find the closest detector
           rDet=0;

            %finding the minimum distance to↙
the point selected
            for j=1: size(detSet,1)
                temp_dist = (b-detSet(j,2))^2+↙
(a-detSet(j,1))^2;
                if(j>1)
                    if(temp_dist<dist_min)
                    dist_min=temp_dist;
                    rDet =detSet(j,3);
                    end
                else
                    dist_min =temp_dist;
                    rDet =detSet(j,3);
                end

            end
                %% check if it is out side of↙
the closest detector
```

```
            %Check if it is within the raduis↙
of the closest detector


            %% Covered? yes

            if rDet>dist_min
                 %% Covered? yes
                m=m+1; % count covered sample
                p =N/n; % probability of not↙
self
                q= 1-N/n;
                z=m/sqrt(n*p*q)-sqrt(n*p/q); %↙
evaluate z score
                if z<z_alpha % not enough↙
coverage
                    if(N==n_sample)
                        %% Accept all saved↙
candidates as new detectors
                            N=0;
                            m=0;


                    end
                else %enough coverage
                    N=m_max+1; % exist the↙
function
                end
```

```
            else
                %% Covered? No : save the↙
candidate
                detSet=[detSet; [a b sqrt↙
(dist_min)]];

                circle(a,b,sqrt(dist_min));

            end
        end
    end

end
 end
y =a;
%This assignment writes a 'double' value into↙
a 'embedded.fi {int16}' type. Code generation↙
does not support changing types through↙
assignment. Check preceding assignments or↙
input type specifications for type mismatches.


%


 function circle(x,y,r)
%x and y are the coordinates of the center of↙
the circle
```

```
%r is the radius of the circle
%0.01 is the angle step, bigger values will↙
draw the circle faster but
%you might notice imperfections (not very↙
smooth)
ang=0:0.01:2*pi;
xp=r*cos(ang);
yp=r*sin(ang);
plot(x+xp,y+yp);
hold on
function h = filledCircle(center,r,N,color)
%↙
----------------------------------------↙
----------------------------------------↙
-
% FILLEDCIRCLE Filled circle drawing
%
% filledCircle(CENTER,R,N,COLOR) draws a↙
circle filled with COLOR that
% has CENTER as its center and R as its↙
radius, by using N points on the
% periphery.
%
% Usage Examples,
%
% filledCircle([1,3],3,1000,'b');
% filledCircle([2,4],2,1000,'r');
%
```

```
% Sadik Hava <sadik.hava@gmail.com>
% May, 2010
%
% Inspired by: circle.m [Author: Zhenhai Wang]
%↙
----------------------------------------------↙
----------------------------------------------↙
-

THETA=linspace(0,2*pi,N);
RHO=ones(1,N)*r;
[X,Y] = pol2cart(THETA,RHO);
X=X+center(1);
Y=Y+center(2);
h=fill(X,Y,color);
axis square;


% COPYRIGHT STUFF... :D (Since I am modifying↙
Zhenhai's code.)
%
% Copyright (c) 2002, Zhenhai Wang
% All rights reserved.
%
% Redistribution and use in source and binary↙
forms, with or without
% modification, are permitted provided that↙
the following conditions are
```

```
% met:
%
%     * Redistributions of source code must↙
retain the above copyright
%       notice, this list of conditions and↙
the following disclaimer.
%     * Redistributions in binary form must↙
reproduce the above copyright
%       notice, this list of conditions and↙
the following disclaimer in
%       the documentation and/or other↙
materials provided with the distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT↙
HOLDERS AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES,↙
INCLUDING, BUT NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND↙
FITNESS FOR A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE↙
COPYRIGHT OWNER OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,↙
SPECIAL, EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT↙
LIMITED TO, PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,↙
DATA, OR PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY↙
```

```
THEORY OF LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT↙
(INCLUDING NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS↙
SOFTWARE, EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.
```