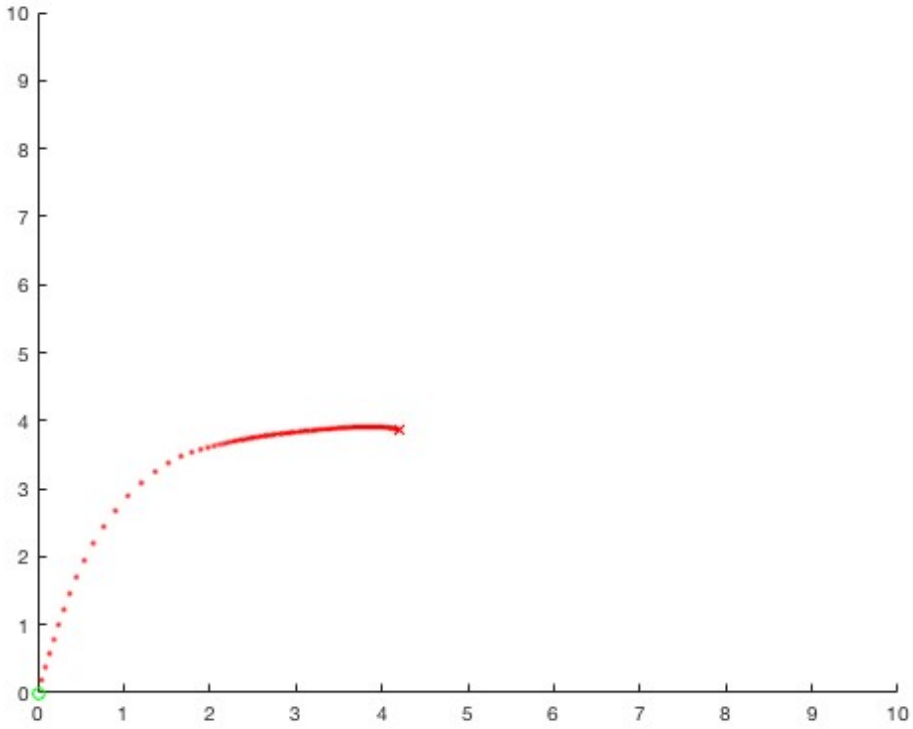
```
%==========================
%  Name:                hw7_1.m
%
%  Author:              Kairi Kozuma
%==========================


%============================== findMinima ==============================
%
%  A script to find the location of a function minimum over the planar
%  domain [0,10] x [0,10].  There should be a function called gradF
%  available to the script in order to work.
%
%============================== findMinima ==============================


%==[1] Initial guess and other parameters.
pos = [0;0];                    % Initial estimate of the location of the minimum.

fprintf('Initial guess: (%d, %d)\n', pos(1), pos(2));

c = 10   ;                              % Update step factor (sometimes called the times
tep).

nsteps = 200 ;                  % Number of steps in the gradient descent iterations.

figure(1);
  axis([0 10 0 10]);
  hold on;
  plot(pos(1), pos(2), 'go');

traj = zeros(2,nsteps);

for ii=1:nsteps
  dpos = gradF(pos);
  pos = pos - dpos;
  traj(:,ii) = pos;
  plot(pos(1), pos(2), 'r.');
  drawnow;
end

plot(pos(1), pos(2), 'rx');
fprintf('Final value is (%5.3f, %5.3f).\n\n', pos(1), pos(2));
hold off;
fprintf('Trajectory followed:\n');
for ii=1:nsteps
    fprintf('%d: (%d, %d)\n',ii,traj(1,ii),traj(2,ii));
end
```

```
Initial guess: (0, 0)
Final value is (4.206, 3.865).

Trajectory followed:
1: (4.453478e-02, 1.856410e-01)
2: (9.062233e-02, 3.774205e-01)
3: (1.387828e-01, 5.761766e-01)
4: (1.897934e-01, 7.827860e-01)
5: (2.447836e-01, 9.980611e-01)
6: (3.053263e-01, 1.222553e+00)
7: (3.734708e-01, 1.456232e+00)
8: (4.516310e-01, 1.698063e+00)
9: (5.422498e-01, 1.945591e+00)
10: (6.472581e-01, 2.194732e+00)
11: (7.674970e-01, 2.439924e+00)
12: (9.023163e-01, 2.674625e+00)
13: (1.049455e+00, 2.891960e+00)
14: (1.205140e+00, 3.085365e+00)
15: (1.364260e+00, 3.249139e+00)
16: (1.520420e+00, 3.379017e+00)
17: (1.665810e+00, 3.473223e+00)
18: (1.792013e+00, 3.535195e+00)
19: (1.895357e+00, 3.575541e+00)
20: (1.979990e+00, 3.604750e+00)
21: (2.051587e+00, 3.628038e+00)
22: (2.114125e+00, 3.647588e+00)
23: (2.170119e+00, 3.664447e+00)
24: (2.221203e+00, 3.679222e+00)
25: (2.268481e+00, 3.692322e+00)
26: (2.312737e+00, 3.704045e+00)
27: (2.354546e+00, 3.714622e+00)
28: (2.394342e+00, 3.724236e+00)
29: (2.432463e+00, 3.733038e+00)
30: (2.469175e+00, 3.741148e+00)
31: (2.504692e+00, 3.748668e+00)
32: (2.539189e+00, 3.755683e+00)
33: (2.572810e+00, 3.762263e+00)
34: (2.605677e+00, 3.768467e+00)
35: (2.637888e+00, 3.774346e+00)
36: (2.669529e+00, 3.779942e+00)
37: (2.700672e+00, 3.785290e+00)
38: (2.731377e+00, 3.790421e+00)
39: (2.761694e+00, 3.795360e+00)
40: (2.791668e+00, 3.800128e+00)
41: (2.821334e+00, 3.804745e+00)
42: (2.850722e+00, 3.809225e+00)
43: (2.879856e+00, 3.813581e+00)
44: (2.908756e+00, 3.817823e+00)
45: (2.937436e+00, 3.821961e+00)
46: (2.965909e+00, 3.826002e+00)
47: (2.994181e+00, 3.829950e+00)
48: (3.022257e+00, 3.833811e+00)
```

49: (3.050138e+00, 3.837587e+00)
50: (3.077823e+00, 3.841281e+00)
51: (3.105308e+00, 3.844893e+00)
52: (3.132588e+00, 3.848424e+00)
53: (3.159655e+00, 3.851873e+00)
54: (3.186499e+00, 3.855241e+00)
55: (3.213109e+00, 3.858525e+00)
56: (3.239474e+00, 3.861724e+00)
57: (3.265579e+00, 3.864836e+00)
58: (3.291412e+00, 3.867857e+00)
59: (3.316957e+00, 3.870787e+00)
60: (3.342200e+00, 3.873621e+00)
61: (3.367125e+00, 3.876358e+00)
62: (3.391717e+00, 3.878994e+00)
63: (3.415960e+00, 3.881527e+00)
64: (3.439840e+00, 3.883954e+00)
65: (3.463343e+00, 3.886272e+00)
66: (3.486454e+00, 3.888480e+00)
67: (3.509161e+00, 3.890576e+00)
68: (3.531450e+00, 3.892558e+00)
69: (3.553311e+00, 3.894425e+00)
70: (3.574733e+00, 3.896175e+00)
71: (3.595707e+00, 3.897809e+00)
72: (3.616225e+00, 3.899325e+00)
73: (3.636278e+00, 3.900724e+00)
74: (3.655862e+00, 3.902008e+00)
75: (3.674972e+00, 3.903175e+00)
76: (3.693604e+00, 3.904229e+00)
77: (3.711755e+00, 3.905170e+00)
78: (3.729425e+00, 3.906001e+00)
79: (3.746614e+00, 3.906724e+00)
80: (3.763321e+00, 3.907342e+00)
81: (3.779550e+00, 3.907857e+00)
82: (3.795303e+00, 3.908273e+00)
83: (3.810585e+00, 3.908594e+00)
84: (3.825398e+00, 3.908823e+00)
85: (3.839751e+00, 3.908964e+00)
86: (3.853647e+00, 3.909020e+00)
87: (3.867095e+00, 3.908996e+00)
88: (3.880102e+00, 3.908897e+00)
89: (3.892676e+00, 3.908726e+00)
90: (3.904825e+00, 3.908488e+00)
91: (3.916558e+00, 3.908186e+00)
92: (3.927884e+00, 3.907825e+00)
93: (3.938813e+00, 3.907410e+00)
94: (3.949355e+00, 3.906945e+00)
95: (3.959519e+00, 3.906433e+00)
96: (3.969315e+00, 3.905878e+00)
97: (3.978754e+00, 3.905285e+00)
98: (3.987845e+00, 3.904658e+00)
99: (3.996599e+00, 3.903999e+00)
100: (4.005026e+00, 3.903312e+00)
101: (4.013136e+00, 3.902601e+00)

102: (4.020939e+00, 3.901870e+00)
103: (4.028444e+00, 3.901120e+00)
104: (4.035662e+00, 3.900355e+00)
105: (4.042602e+00, 3.899578e+00)
106: (4.049273e+00, 3.898791e+00)
107: (4.055686e+00, 3.897996e+00)
108: (4.061848e+00, 3.897197e+00)
109: (4.067768e+00, 3.896395e+00)
110: (4.073456e+00, 3.895592e+00)
111: (4.078920e+00, 3.894789e+00)
112: (4.084168e+00, 3.893990e+00)
113: (4.089207e+00, 3.893194e+00)
114: (4.094047e+00, 3.892405e+00)
115: (4.098693e+00, 3.891622e+00)
116: (4.103154e+00, 3.890847e+00)
117: (4.107436e+00, 3.890081e+00)
118: (4.111547e+00, 3.889326e+00)
119: (4.115492e+00, 3.888582e+00)
120: (4.119279e+00, 3.887849e+00)
121: (4.122914e+00, 3.887129e+00)
122: (4.126402e+00, 3.886421e+00)
123: (4.129749e+00, 3.885727e+00)
124: (4.132961e+00, 3.885047e+00)
125: (4.136043e+00, 3.884382e+00)
126: (4.139000e+00, 3.883731e+00)
127: (4.141838e+00, 3.883094e+00)
128: (4.144561e+00, 3.882473e+00)
129: (4.147173e+00, 3.881867e+00)
130: (4.149679e+00, 3.881275e+00)
131: (4.152084e+00, 3.880699e+00)
132: (4.154391e+00, 3.880138e+00)
133: (4.156604e+00, 3.879592e+00)
134: (4.158727e+00, 3.879061e+00)
135: (4.160764e+00, 3.878545e+00)
136: (4.162719e+00, 3.878044e+00)
137: (4.164594e+00, 3.877557e+00)
138: (4.166392e+00, 3.877085e+00)
139: (4.168118e+00, 3.876627e+00)
140: (4.169773e+00, 3.876182e+00)
141: (4.171361e+00, 3.875752e+00)
142: (4.172885e+00, 3.875335e+00)
143: (4.174346e+00, 3.874931e+00)
144: (4.175748e+00, 3.874540e+00)
145: (4.177093e+00, 3.874161e+00)
146: (4.178384e+00, 3.873795e+00)
147: (4.179622e+00, 3.873441e+00)
148: (4.180809e+00, 3.873099e+00)
149: (4.181949e+00, 3.872768e+00)
150: (4.183042e+00, 3.872449e+00)
151: (4.184090e+00, 3.872140e+00)
152: (4.185096e+00, 3.871842e+00)
153: (4.186061e+00, 3.871555e+00)
154: (4.186987e+00, 3.871277e+00)

155: (4.187876e+00, 3.871009e+00)
156: (4.188728e+00, 3.870751e+00)
157: (4.189545e+00, 3.870502e+00)
158: (4.190330e+00, 3.870261e+00)
159: (4.191082e+00, 3.870030e+00)
160: (4.191804e+00, 3.869806e+00)
161: (4.192497e+00, 3.869591e+00)
162: (4.193162e+00, 3.869384e+00)
163: (4.193799e+00, 3.869184e+00)
164: (4.194411e+00, 3.868991e+00)
165: (4.194998e+00, 3.868806e+00)
166: (4.195561e+00, 3.868628e+00)
167: (4.196101e+00, 3.868456e+00)
168: (4.196620e+00, 3.868290e+00)
169: (4.197117e+00, 3.868131e+00)
170: (4.197594e+00, 3.867978e+00)
171: (4.198052e+00, 3.867830e+00)
172: (4.198491e+00, 3.867688e+00)
173: (4.198913e+00, 3.867552e+00)
174: (4.199317e+00, 3.867421e+00)
175: (4.199705e+00, 3.867294e+00)
176: (4.200077e+00, 3.867173e+00)
177: (4.200435e+00, 3.867056e+00)
178: (4.200777e+00, 3.866943e+00)
179: (4.201106e+00, 3.866835e+00)
180: (4.201422e+00, 3.866731e+00)
181: (4.201725e+00, 3.866631e+00)
182: (4.202015e+00, 3.866535e+00)
183: (4.202294e+00, 3.866443e+00)
184: (4.202561e+00, 3.866354e+00)
185: (4.202818e+00, 3.866269e+00)
186: (4.203064e+00, 3.866187e+00)
187: (4.203300e+00, 3.866108e+00)
188: (4.203527e+00, 3.866033e+00)
189: (4.203744e+00, 3.865960e+00)
190: (4.203953e+00, 3.865890e+00)
191: (4.204153e+00, 3.865823e+00)
192: (4.204346e+00, 3.865758e+00)
193: (4.204530e+00, 3.865696e+00)
194: (4.204707e+00, 3.865637e+00)
195: (4.204877e+00, 3.865579e+00)
196: (4.205040e+00, 3.865525e+00)
197: (4.205196e+00, 3.865472e+00)
198: (4.205346e+00, 3.865421e+00)
199: (4.205490e+00, 3.865372e+00)
200: (4.205628e+00, 3.865326e+00)



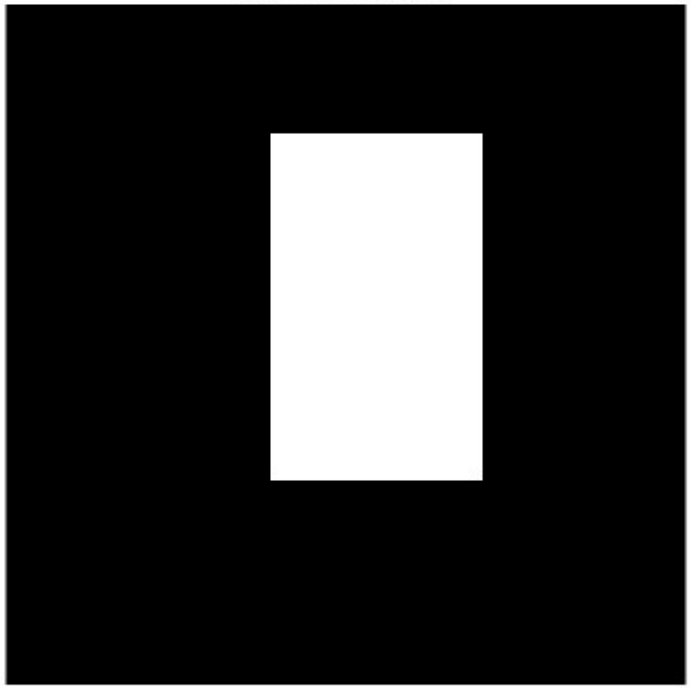
Published with MATLAB® R2016b

```
%==========================
%  Name:                hw7_2.m
%
%  Author:              Kairi Kozuma
%==========================

% 1) Diffuse for 2 images (1 blob, 1 westin)

% blob
IMAGE = double(blurme);
smoothI = isodiffuse(IMAGE, .1, 5000);

fh1 = figure(1);
  clf;
  imagesc(IMAGE);
  colormap('gray');
  axis image;
  axis off;
  title('Blob before diffusion');

fh2 = figure(2);
  clf;
  imagesc(smoothI);
  colormap('gray');
  axis image;
  axis off;
  title('Blob after diffusion');

% westin
IMAGE = westin;
smoothI = isodiffuse(IMAGE, .1, 600);

fh3 = figure(3);
  clf;
  imagesc(IMAGE);
  colormap('gray');
  axis image;
  axis off;
  title('westin before diffusion');

fh4 = figure(4);
  clf;
  imagesc(smoothI);
  colormap('gray');
  axis image;
  axis off;
  title('westin after diffusion');

% 2)

% Show Gaussian blur with increasing standard deviations
```

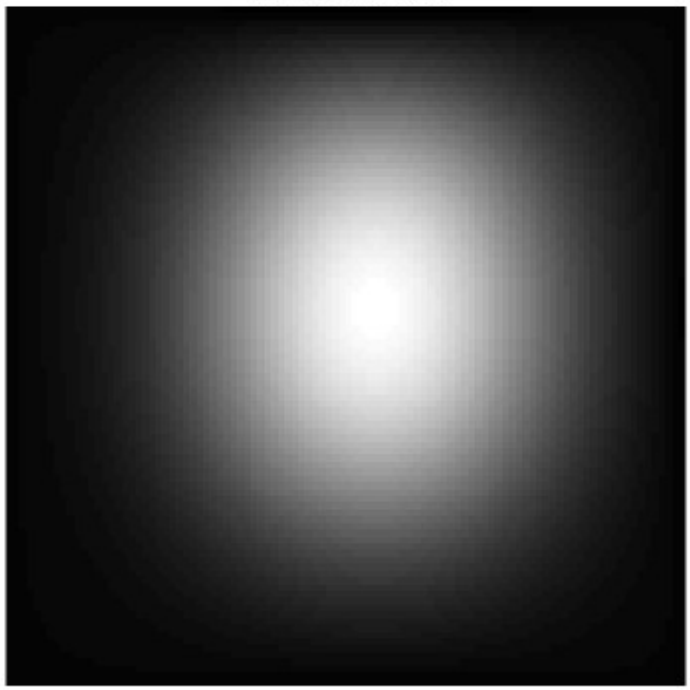
```
t = [0.25, 0.5, 0.75, 1, 1.5, 2];
gaus = gaussianFilt(double(blurme), 2);
err = zeros(1,length(t));
for i = 1:length(t)
    diff = isodiffuse(double(blurme), .001, t(i) * 1000);
    err(i) = sumsqr(gaus - diff);
end
fh5 = figure(5);
plot(t, err);
title('Error (sumsqr) vs time');


% 3)
tarr = 2.^[0:10];
avg = zeros(1,length(tarr));
fh6 = figure(6);
for i=1:length(tarr)
    smoothI3 = isodiffuse(double(rcells), 0.01, tarr(i));
    avg(i) = mean(mean(smoothI3));
end

plot(tarr, avg);
title('Average vs time');
%================================ isodiffuse ================================
%
%  Performs isotropic diffusion by running the heat equation on a grayscale
%  image.
%
%  function sI = isodiffuse(gI, dt, iter)
%
%
%  Inputs:
%    gI         - grayscale/scalar image (should be floating point format).
%    dt         - timestep to use in running the heat equation
%                      the argument should be 0 < dt < 0.5.
%    iter       - total number of iterations.
%
%  Output:
%    sI      - the smoothed image.
%
%================================ isodiffuse ================================

%
%  Name:                isodiffuse.m
%
%  Author:              Patricio A. Vela, pvela@gatech.edu
%
%  Created:             2010/01/05
%  Modified:    2010/01/05
%
%================================ isodiffuse ================================
function sI = isodiffuse(gI, dt, iter)
```

```
for i=1:iter
    dI = del2(gI);
    gI = gI + dI * dt;
end

sI = gI;

end
%
%================================ isodiffuse ================================


%============================ gaussianFilt ================================
%
%  function [image] = gaussianFilt(image, sd)
%
%
%  INPUT:
%    image  - matrix representing image
%    sd     - standard deviation
%
%============================ gaussianFilt ================================
function [image] = gaussianFilt(image, sd)

% Gaussian filter to use
sz = ceil(2.5 * sd);
if mod(sz,2) == 0
    sz = sz + 1;
end
gausfilt = fspecial('gaussian', [sz,sz], sd);

% Filter using the Gaussian blur
image = imfilter(image, gausfilt);

end
```

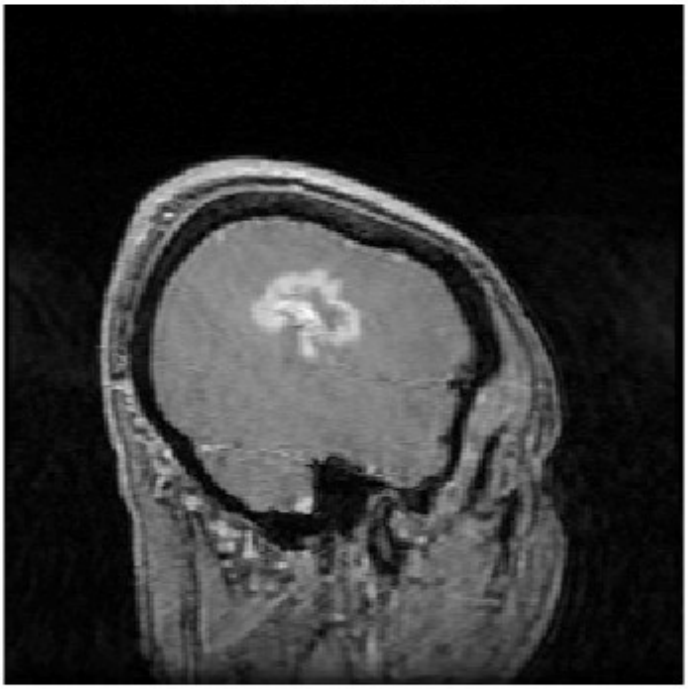
Blob before diffusion



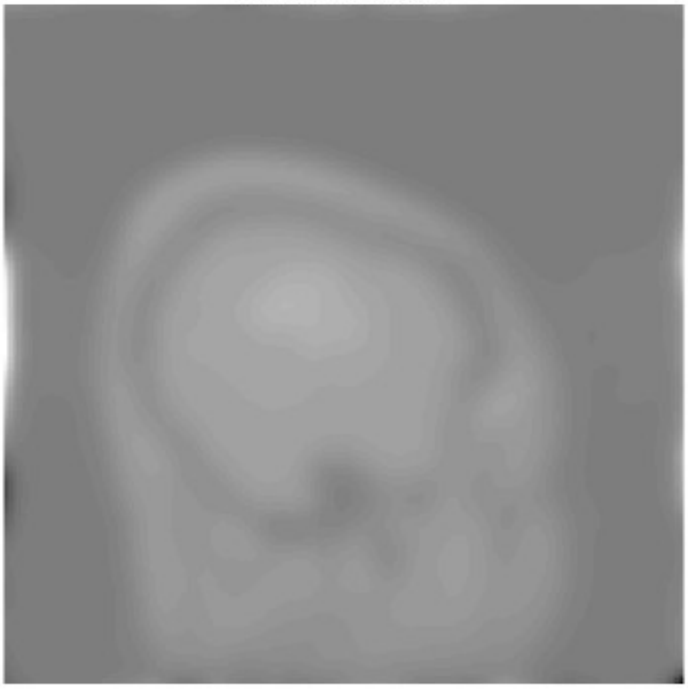
Blob after diffusion

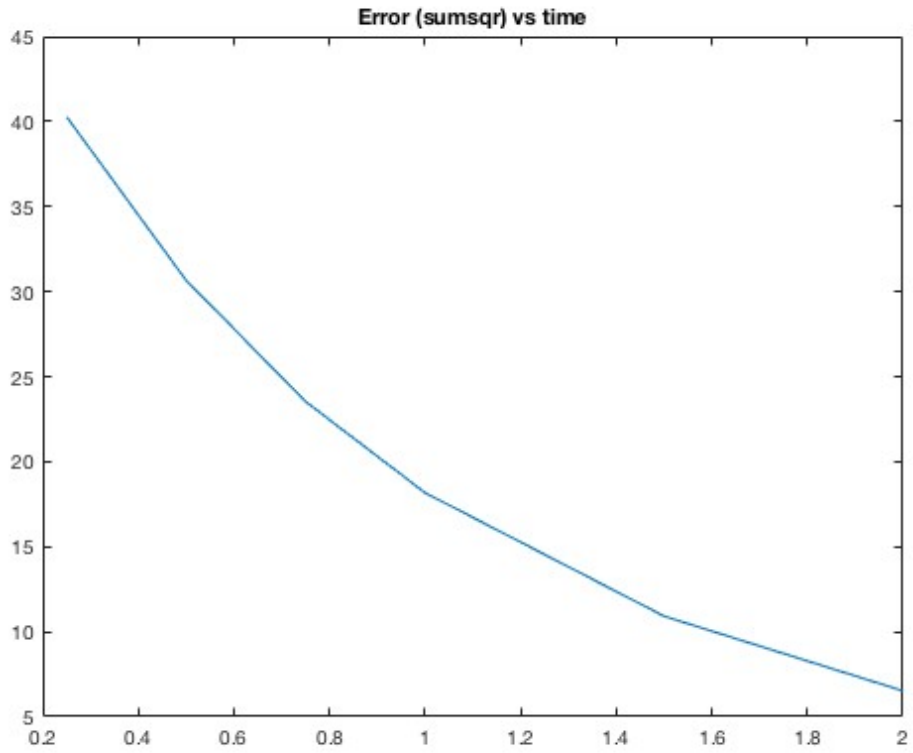


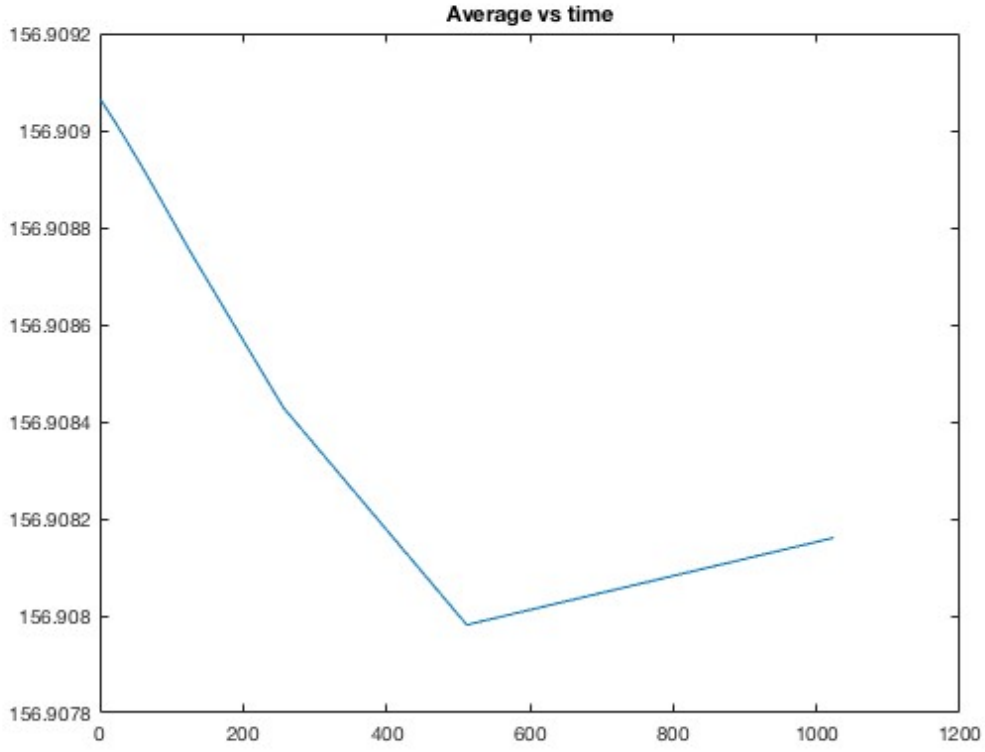
westin before diffusion



westin after diffusion




Error (sumsqr) vs time
45
40
35
30
25
20
15
10
5
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2


Average vs time
156.9092
156.909
156.9088
156.9086
156.9084
156.9082
156.908
156.9078
0
200
400
600
800
1000
1200

```
#!/usr/bin/env python
import rospy
import roslib
import math
from geometry_msgs.msg import Twist
from sensor_msgs.msg import LaserScan
from kobuki_msgs.msg import BumperEvent
from kobuki_msgs.msg import CliffEvent
from kobuki_msgs.msg import WheelDropEvent

class Move():
    STATE_STOPPED = 0   # longer pause for bumper stop, back off and
turn
    STATE_FORWARD = 1
    STATE_TURN_LEFT_SMALL = 2
    STATE_TURN_LEFT_MEDIUM = 3
    STATE_TURN_LEFT_LARGE = 4
    STATE_TURN_RIGHT_SMALL = 5
    STATE_TURN_RIGHT_MEDIUM = 6
    STATE_TURN_RIGHT_LARGE = 7
    STATE_TURN = 8              # this is for turning 180 degree
without backing off
    STATE_BACKWARD_BUMPER = 9
    STATE_PAUSED = 10   # this is for shorter pause used for turning
    STATE_WHEEL_DROP = 11
    STATE_CLIFF = 12

    def __init__(self):
        ############################
        # Setting up for LaserScane
        ############################
        # Binary values (obstacle exists or not exists)
        self.b1 = 0
        self.b2 = 0
        self.b3 = 0
        self.b4 = 0
        self.b5 = 0

        # Average value of each section
        self.sect_1 = 0.0
        self.sect_2 = 0.0
        self.sect_3 = 0.0
        self.sect_4 = 0.0
        self.sect_5 = 0.0

        # Tweaking settings
        self.threshold = 0.5

        self.LASER_FLAG = Move.STATE_FORWARD
        self.BUMPER_FLAG = Move.STATE_FORWARD
```

```
        self.WHEELDROP_FLAG = Move.STATE_FORWARD
        self.CLIFF_FLAG = Move.STATE_FORWARD

        self.states = {
                    0: Move.STATE_FORWARD,
                    1: Move.STATE_TURN_LEFT_SMALL,
                   10: Move.STATE_TURN_LEFT_MEDIUM,
                   11: Move.STATE_TURN_LEFT_MEDIUM,
                  100: Move.STATE_TURN_LEFT_LARGE,
                  101: Move.STATE_TURN_LEFT_LARGE,
                  110: Move.STATE_TURN_LEFT_LARGE,
                  111: Move.STATE_TURN_LEFT_LARGE,
                 1000: Move.STATE_TURN_RIGHT_MEDIUM,
                 1001: Move.STATE_TURN_LEFT_LARGE,
                 1010: Move.STATE_TURN_LEFT_LARGE,
                 1011: Move.STATE_TURN_LEFT_LARGE,
                 1100: Move.STATE_TURN_RIGHT_LARGE,
                 1101: Move.STATE_TURN_LEFT_LARGE,
                 1110: Move.STATE_TURN_LEFT_LARGE,
                 1111: Move.STATE_TURN_LEFT_LARGE,
                10000: Move.STATE_TURN_RIGHT_SMALL,
                10001: Move.STATE_TURN_RIGHT_LARGE,
                10010:  Move.STATE_TURN_RIGHT_LARGE,
                10011:  Move.STATE_TURN_LEFT_LARGE,
                10100:  Move.STATE_TURN_RIGHT_LARGE,
                10101:  Move.STATE_TURN_RIGHT_LARGE,
                10110:  Move.STATE_TURN_RIGHT_LARGE,
                10111:  Move.STATE_TURN_LEFT_LARGE,
                11000:  Move.STATE_TURN_RIGHT_MEDIUM,
                11001:  Move.STATE_TURN_RIGHT_LARGE,
                11010:  Move.STATE_TURN_RIGHT_LARGE,
                11011:  Move.STATE_TURN_RIGHT_LARGE,
                11100:  Move.STATE_TURN_RIGHT_LARGE,
                11101:  Move.STATE_TURN_RIGHT_LARGE,
                11110:  Move.STATE_TURN_RIGHT_LARGE,
                11111:  Move.STATE_TURN_RIGHT_LARGE}

        # Initialize LaserScan
        rospy.init_node('Move')
        rospy.Subscriber('/scan', LaserScan, self.call_back)


        ############################
        # Setting up for Move
        ############################
        # initiliaze Move Node
        rospy.Subscriber("/mobile_base/events/
bumper",BumperEvent,self.BumperEventCallback)
        rospy.Subscriber("/mobile_base/events/
cliff",CliffEvent,self.CliffEventCallback)
```

```
        rospy.Subscriber("/mobile_base/events/
wheel_drop",WheelDropEvent,self.WheelDropEventCallback)

        # tell user how to stop TurtleBot
        rospy.loginfo("To stop TurtleBot CTRL + C")

        # What function to call when you ctrl + c
        rospy.on_shutdown(self.shutdown)

        # Create a publisher which can "talk" to TurtleBot and tell it
to move
        # Tip: You may need to change cmd_vel_mux/input/navi to /
cmd_vel if you're not using TurtleBot2
        self.cmd_vel = rospy.Publisher('cmd_vel_mux/input/navi',
Twist, queue_size=10)

        #TurtleBot will stop if we don't keep telling it to move.  How
often should we tell it to move? 10 HZ
        r =rospy.Rate(10);

        # Twist is a datatype for velocity
        move_cmd = Twist()
        # let's go forward at 0.2 m/s
        move_cmd.linear.x = 0.2
        # let's turn at 0 radians/s
        move_cmd.angular.z = 0

        #stop_cmd: By default, new instance has velocity 0
        stop_cmd = Twist()

        #turn_cmd: let's turn at 45 deg/s
        turn_cmd = Twist()
        turn_cmd.linear.x = 0
        turn_cmd.angular.z = math.radians(45) #45 deg/s in radians/s
counterclockwise

        #turn_cmd_inv: let's turn at 45 deg/s
        turn_cmd_inv = Twist()
        turn_cmd_inv.linear.x = 0
        turn_cmd_inv.angular.z = math.radians(-45) #45 deg/s in
radians/s counterclockwise

        # backward_cmd
        backward_cmd = Twist()
        # backward speed at -0.2 m/s
        backward_cmd.linear.x = -0.2
        # let's turn at 0 radians/s
        backward_cmd.angular.z = 0
```

```
        # self. refers to an instance of a class
        # Move. refers to the class variable
        self.BUMPER_STATE = Move.STATE_FORWARD
        self.LASER_STATE = Move.STATE_FORWARD
        ############################
        # State Machine
        ############################
        while not rospy.is_shutdown():
                # Update states using different flags
                self.LASER_STATE = self.LASER_FLAG
                self.BUMPER_STATE = self.BUMPER_FLAG
                self.WHEELDROP_STATE = self.WHEELDROP_FLAG
                self.CLIFF_STATE = self.CLIFF_FLAG

                if (self.BUMPER_STATE == Move.STATE_FORWARD and
self.LASER_STATE == Move.STATE_FORWARD and self.WHEELDROP_STATE ==
Move.STATE_FORWARD and self.CLIFF_STATE == Move.STATE_FORWARD):
                        rospy.loginfo("In moving state")
                        self.cmd_vel.publish(move_cmd)
                        r.sleep()

                # This is where all the bumper checking should happen
                if (self.BUMPER_STATE != Move.STATE_FORWARD or
self.CLIFF_STATE != Move.STATE_FORWARD):
                        if (self.BUMPER_STATE == Move.STATE_STOPPED or
self.CLIFF_STATE == Move.STATE_STOPPED):
                                for x in range(0,20):
                                        self.cmd_vel.publish(stop_cmd)
                                        rospy.loginfo("Waiting ...")
                                        r.sleep()

                                for x in range(0,2):

self.cmd_vel.publish(backward_cmd)
                                        r.sleep()
                                for x in range(0,10):
                                        r.sleep()

                                rospy.loginfo("Turning 180 degree ")
                                for x in range(0,40):
                                        self.cmd_vel.publish(turn_cmd)
                                        r.sleep()
                                self.BUMPER_STATE = Move.STATE_FORWARD
                                self.CLIFF_STATE = Move.STATE_FORWARD
                                for x in range(0,10):
                                        r.sleep()


                if (self.WHEELDROP_STATE != Move.STATE_FORWARD):
```

```
                        self.cmd_vel.publish(stop_cmd)
                        r.sleep()

                # when laser detects obstacles
                if (self.LASER_STATE != Move.STATE_FORWARD):

                        while (self.LASER_STATE !=
Move.STATE_FORWARD):
                                # Pause for a bit
                                for x in range(0,10):
                                        self.cmd_vel.publish(stop_cmd)
                                        r.sleep()
                                # Turn Correspondingly
                                if (self.LASER_STATE ==
Move.STATE_TURN_LEFT_SMALL):
                                        for x in range(0,10):

self.cmd_vel.publish(turn_cmd)
                                                r.sleep()
                                        for x in range(0,10):
                                                r.sleep()

                                elif (self.LASER_STATE ==
Move.STATE_TURN_LEFT_MEDIUM):
                                        for x in range(0,20):

self.cmd_vel.publish(turn_cmd)
                                                r.sleep()
                                        for x in range(0,10):
                                                r.sleep()
                                elif (self.LASER_STATE ==
Move.STATE_TURN_LEFT_LARGE):
                                        for x in range(0,30):

self.cmd_vel.publish(turn_cmd)
                                                r.sleep()
                                        for x in range(0,10):
                                                r.sleep()
                                elif (self.LASER_STATE ==
Move.STATE_TURN_RIGHT_SMALL):
                                        for x in range(0,10):

self.cmd_vel.publish(turn_cmd_inv)
                                                r.sleep()
                                        for x in range(0,10):
                                                r.sleep()
                                elif (self.LASER_STATE ==
Move.STATE_TURN_RIGHT_MEDIUM):
                                        for x in range(0,20):
```

```
self.cmd_vel.publish(turn_cmd_inv)
                                                r.sleep()
                                        for x in range(0,10):
                                                r.sleep()

                                else:
                                        # elif (self.LASER_STATE ==
Move.STATE_TURN_RIGHT_LARGE):
                                        for x in range(0,30):

self.cmd_vel.publish(turn_cmd_inv)
                                                r.sleep()
                                        for x in range(0,10):
                                                r.sleep()

                                # Update flag again
                                self.LASER_STATE = self.LASER_FLAG
                r.sleep()

    ############################
    # Setting up for functions
    ############################
    def reset_sect(self):
        self.sect_1 = 0.0
        self.sect_2 = 0.0
        self.sect_3 = 0.0
        self.sect_4 = 0.0
        self.sect_5 = 0.0

    def sort(self, laserscan):
        entries = len(laserscan.ranges)
        entry_per_sect = entries / 7.0
        #for entry in range(0,entries):

        for i, entry in enumerate(laserscan.ranges):
                # Professor said that we should assume NAN as too far
away
                # Value ranges from 0.45 m to 10m
                # I use 5 out of 7 sections. Just cuz
                entry_i = entry if (not math.isnan(entry)) else 0.0
                self.sect_1 += entry_i if (entries/7 < i < 2*entries/
7) else 0
                self.sect_2 += entry_i if (2*entries/7 < i <
3*entries/7) else 0
                self.sect_3 += entry_i if (3*entries/7 < i <
4*entries/7) else 0
                self.sect_4 += entry_i if (4*entries/7 < i <
5*entries/7) else 0
                self.sect_5 += entry_i if (5*entries/7 < i <
6*entries/7) else 0
```

```
        self.sect_1 = float(self.sect_1/entry_per_sect)
        self.sect_2 = float(self.sect_2/entry_per_sect)
        self.sect_3 = float(self.sect_3/entry_per_sect)
        self.sect_4 = float(self.sect_4/entry_per_sect)
        self.sect_5 = float(self.sect_5/entry_per_sect)
        #rospy.loginfo("sort complete,sect_1: " +
'{0:.2f}'.format(self.sect_1) + " sect_2: " +
'{0:.2f}'.format(self.sect_2) + " sect_3: " +
'{0:.2f}'.format(self.sect_3) + " sect_4: " +
'{0:.2f}'.format(self.sect_4) + " sect_5:" +
'{0:.2f}'.format(self.sect_5))

    def laser_update(self):
        self.b1 = 1 if self.sect_1 < self.threshold else 0
        self.b2 = 1 if self.sect_2 < self.threshold else 0
        self.b3 = 1 if self.sect_3 < self.threshold else 0
        self.b4 = 1 if self.sect_4 < self.threshold else 0
        self.b5 = 1 if self.sect_5 < self.threshold else 0
        sect = int(str(self.b1) + str(self.b2) + str(self.b3) +
str(self.b4) + str(self.b5))
        sect_str = str(self.b1) + str(self.b2) + str(self.b3) +
str(self.b4) + str(self.b5)
        #rospy.loginfo("Sect = " + sect_str)
        #rospy.loginfo("LASER_FLAG is " + str(self.states[sect]))
        self.LASER_FLAG = self.states[sect]
        self.reset_sect()

    def call_back(self,laserscan):
        self.sort(laserscan)
        self.laser_update()


    def BumperEventCallback(self,data):
                if (data.state == BumperEvent.PRESSED):
                        # sleep for 2 seconds, if the bumper is
pressed
                        # publish the velocity
                        r =rospy.Rate(1)
                        rospy.loginfo("Bumper Pressed")
                        self.BUMPER_FLAG = Move.STATE_STOPPED

                if (data.state == BumperEvent.RELEASED):
                        r =rospy.Rate(1)
                        rospy.loginfo("Bumper Released")
                        self.BUMPER_FLAG = Move.STATE_FORWARD
    def CliffEventCallback(self,data):
                if (data.state == CliffEvent.CLIFF):
                        r =rospy.Rate(1)
                        rospy.loginfo("CliffEvent Triggered")
                        self.CLIFF_FLAG = Move.STATE_STOPPED
```

```
                else:
                        r =rospy.Rate(1)
                        self.CLIFF_FLAG = Move.STATE_FORWARD
    def WheelDropEventCallback(self,data):
                if (data.state == WheelDropEvent.RAISED):
                        r =rospy.Rate(1)
                        rospy.loginfo("WheelDropEvent Triggered")
                        self.WHEELDROP_FLAG = Move.STATE_STOPPED
                else:
                        r =rospy.Rate(1)
                        self.WHEELDROP_FLAG = Move.STATE_FORWARD


                self.CLIFF_FLAG = Move.STATE_FORWARD

    def shutdown(self):
        # stop turtlebot
        rospy.loginfo("Stop TurtleBot")
        # a default Twist has linear.x of 0 and angular.z of 0.  So
it'll stop TurtleBot
        self.cmd_vel.publish(Twist())
        # sleep just makes sure TurtleBot receives the stop command
prior to shutting down the script
        rospy.sleep(1)



if __name__ == '__main__':
        Move()
```