

Cat Hunt

Expanded Dog Chasing Cat Problem

Eric Olson

ECE 539
Professor Hu
December 19, 2003

TABLE OF CONTENTS

INTRODUCTION.....	3
MOTIVATION	3
PROBLEMS	3
WORK PERFORMED.....	3
PROGRAMMING	3
TESTING.....	5
RESULTS	5
CONCLUSION	5
APPENDIX A: Matlab Source Code, Chase Script.....	6
APPENDIX B: Matlab Source Code, Closest Dog Calculator	13
APPENDIX C: Matlab Source Code, Distance Formula	14
APPENDIX D: Matlab Source Code, Graphical Analysis.....	15
APPENDIX E: Required Files.....	16
APPENDIX F: Sample Graphical Representations.....	17
APPENDIX G: Testing Results, Max-Min	18
APPENDIX H: Testing Results, Kosko's Max-Product.....	19
APPENDIX I: Graphical Analysis, Max-Min vs. Kosko's Max-Product Rule.....	20

INTRODUCTION

The following is the final report for the semester project in ECE 539: Introduction to Artificial Neural Network and Fuzzy Systems. This project creates a modified and expanded version of the dog-cat simulation. The simulation uses Fuzzy Control Logic to determine paths taken by the dogs chasing the cat. In this expanded version, there are options for multiple dogs and the cat has been given some limited artificial intelligence to attempt to evade the dogs.

Motivation

- It was relatively boring to watch the cat run straight and have the dog circling it.
- Cats are smart, let it try to escape.
- Dogs are smarter than the original simulation.
- Dogs really don't like cats, get more dogs!

Problems

1. Expand and improve the dog-cat problem by increasing the number of dogs and improving their A.I., while also giving the cat A.I.
2. Create clearer and easier to use graphical representation of chase simulation

WORK PERFORMED

Programming

The simulation is based on the dog-cat problem and the script *dogcatfz.m* written by Yu Hen Hu. The program was built around the fuzzy systems and rules created by Hu in his simulation. From his original program, many new features were added including:

1. Control interface menu to get parameters from user
 - a. Dog locations created randomly from input number from user
 - b. Dog locations determined from data file "test"
2. Two graphical representation possibilities (Appendix F)
 - a. Animal route plotting with trails
 - b. Animal route plotting without trails
3. Code to allow for many iterations to test results
4. Artificially intelligent cat that attempts to evade and escape dogs

5. Support for unlimited number of dogs
6. Improved dog chasing A.I.

Structure

The program is run by the main script *chase.m* (Appendix A). This file runs the simulation. It has a main loop that continues calculating and plotting routes of the cat and the dogs until the cat is caught, escapes the 600x600 square, or iterates 500 times. The makes calls to all required files shown in Appendix E. New scripts *closest.m* (Appendix B) and *distance.m* (Appendix C) were written to complete important calculations for the main script program.

Theory

Cat A.I.

The cat A.I. was developed initially using FCL, but it did not fit the needs of the simulation. Instead, through testing, it was determined and developed that the cat would base it's future route off of the biggest threat: the closest dog. After determining which dog was closest, the cat would then run in the same direction that the dog is running in. This is not always directly away from the dog if that dog is "far" away from the cat as discussed in the Dog A.I. section below. I felt that this design would allow for a more appropriate decision making process by the cat.

Dog A.I.

The dog A.I is mainly handled using FCL that was implemented originally by Professor Hu in his *dogcatfz.m* simulation. This FCL system was built upon to improve the dogs' capability at catching the cat. Initially, the dog would merely run towards where the cat is at the time of calculation. Because of this, dogs that were "far" away from the cat would have little to no chance of catching the cat.

The modified method checks if the dog is close to the cat (within 5 iterations of reaching the cat's current position). If close, the dog uses FCL to determine its route. If far from the cat, the dog sets its course for a point in front of the cat as guessed upon using the cat's current position and angle it is facing. If the dog is far from the cat and the cat is moving relatively towards the dog, the route is determined using FCL. This model allows for the dogs close to the cat to swarm while the dogs far from the cat to attempt to head the cat off in its attempt to escape.

Method of Inference

The *infer.m* script allows us to choose either the Max-Min rule or Kosko's Max-Product rule for calculations. Based on results compiled for the Final Exam, it was hypothesized that the Max-Min rule would create a more successful FCL. The results of testing these two rules can be found in Appendices G, H, and I and will be discussed in the Results section of the report.

Testing

Testing was done using many different configurations of the program. Static parameters including the speed of the cat and dogs were chosen after receiving results that created a system similar to that of reality. Testing code was added to *chase.m* to aid in testing Max-Min rule or Kosko's Max-Product rule.

RESULTS

The result of this project is the simulation itself. The purpose was not to design a dog that could chase down any cat or a cat that could escape a certain number of dogs. Instead, it was designed to create a relatively realistic simulation where the cat and dog A.I. would cause actions to be taken similar to that in reality.

Testing was done to see which rule, Max-Min or Kosko's Max-Product, that would produce the most realistic results. After thorough testing, it was determined that the Max-Min rule more accurately created the desired effect. Results of this testing can be seen numerically and graphically in Appendices G, H, and I. The evaluation of this judgement was made through the numerical results and watching the graphical simulation and the perceived actions and routes chosen by the dogs.

CONCLUSION

The expansion and improvement of this problem created a very interesting and entertaining simulation. The random dog locations makes watching the simulation very fun, as every trial is different. I felt this was an extremely fun project and enjoyed developing my animals and their A.I. I felt this was a very good choice for a project because of how interesting this application was of Fuzzy Control Logic. I hope you enjoyed the simulation.

APPENDIX A: Matlab Source Code, Chase Script

Matlab Source Code (chase.m):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS539                                     %
% Semester Project                         %
% Dog Chasing Cat Script                   %
% chase.m                                  %
%                                           %
% Eric Olson - 9016984685                  %
% CAE Login - eolson                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This script calculates and portrays a modified version of the dog-cat
% problem. It allows for multiple dogs chasing a single cat. The cat is
% "smart" and knows how to avoid the dogs. The dogs know how to take
% appropriate angles to head off the cat. The script makes use of Fuzzy
% Control Logic and calls fuzzy system scripts created by Professor Yu Hen
% Hu. These files have not been modified. The only additional scripts are
% closest.m that determines the dog currently closest to the cat and
% distance.m that calculates the distance between two points.

% Code Based On:
% dogcatfz.m - fuzzy control for dog-cat problem
%
% copyright (C) 1996 by Yu Hen Hu
% created: 12/6/96
% last modified: 11/25/2001

clear all
close all

% define global variables
global xcat
global ycat
global xdog
global ydog
global t % current time in function in 1/10seconds
global ndogs % number of dogs

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% User Control Menu %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set dynamic values based on which case the user wants to do
disp('Welcome to Eric''s Dog Chasing Cat Program!')
disp(' ')
disp('Which Type of Testing Do You Want? (1 or 2) ')
disp('1 - Use random dog locations')
disp('2 - Load dog locations from data file "test"')
reply1 = input(' ');

if reply1 == 1
    disp('How many dogs do you want to use?')
```

```

reply2 = input(' ');
ndogs = reply2;

% randomly create initial locations and angles of dogs
test1 = random('Normal',0,100,ndogs,3);
elseif reply1 == 2
    disp('Initial dog locations and angles loaded from "test"')

    % loading dog locations from file "test" in form of: x y initial_angle
    load test;
    test1 = test;

    ndogs = length(test1);
else
    disp('Using default of 10 dogs in random locations')
    ndogs = 10;

    % randomly create initial locations and angles of dogs
    test1 = random('Normal',0,100,ndogs,3);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Testing Code           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This code runs 100 iterations of program to test number of catches and
% cat escapes made. Used for data collection.
%
% dogs = [1 2 3 4 5 8 10 15 20 25 30 40 50 100];
%
% for q = 1:14
%
% ndogs = dogs(q)
%
% escape(q) = 0;
% caught(q) = 0;
%
% for p = 1:100
%
% test1 = random('Normal',0,100,ndogs,3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Fuzzy Sets           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

M=[1 .67 .33 0 0 0 0 0 0 0 0 0 0 0; % LN fuzzy set
    0 .33 .67 1 .67 .33 0 0 0 0 0 0 0 0; % SN fuzzy set
    0 0 0 0 .33 .67 1 .67 .33 0 0 0 0 0; % ZO fuzzy set
    0 0 0 0 0 0 0 .33 .67 1 .67 .33 0; % SP fuzzy set
    0 0 0 0 0 0 0 0 0 0 .33 .67 1]; % LP fuzzy set

[nadj,lensp]=size(M);
ang_range=[-180 180];
dangs=(ang_range(2)-ang_range(1))/(lensp-1);
sang = [ang_range(1):dangs:ang_range(2)]; % support for ang
K=1/6; % dz = k*ang --proportional control

```

```

sdz=K*sang;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Initiation           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

terminate=0;
t=0;
vc=3; % cat speed
vd=1.5; % dog speed

% initial position of cat
xcat(1)=0;
ycat(1)=0;

azic(1)=90;      % initially the cat is facing east
angc(1)=90;
dzc(1) = 0;      % initialize dz of cat

% load initial locations of dogs
for i = 1:ndogs
    % initial position of dog
    xdog(i,1)= test1(i,1);
    ydog(i,1)= test1(i,2);

    % initial heading of dog
    azid(i,1)= test1(i,3);

    % initial angle of dog
    dzd(i,1)= 0;
    angd(i,1)=0;
end

ruledc; % initialization of the rule set.
% it will define two variables: d and rule
% d - a row vector, with 2 elements. d(1): # of fuzzy sets defined
% on the input variable, d(2): # of fuzzy sets defined on the output
% variable.
% rule: a matrix. Each row is a rule. Each row contains d(1) + d(2)
% elements. The first d(1) elements specifying which fuzzy set is
% used as the antecedent part. The last d(2) elements specifying which
% fuzzy sets in the output control variable are used.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%           Main Loop           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% loop while cat is not caught or within boundaries
while terminate == 0

% increment t
t = t+1;

% determine closest dog
closest_dog = closest;

```

```

% base angle of cat off that of closest dog
angc(t) = atan2(ycat(t)-ydog(closest_dog,t),xcat(t)-
xdog(closest_dog,t))*180/pi-azid(closest_dog,t);
if angc(t) > 180, % make sure -pi <= angc(t) <= pi
    angc(t)=angc(t)-360;
elseif angc(t) < -180,
    angc(t)=angc(t)+360;
end

% loop through all dogs and calculate ang values
for i = 1:ndogs

    % get distance of dog to cat for guessing future cat location
    dist = distance(xcat(t),ycat(t),xdog(i,t),ydog(i,t));

    % if close to cat
    if dist/vd >= 5
        % determine angle by guessing where the cat will be when dog would
reach it
        xcat_guess = xcat(t) + dist/(2*vd)*vc*cos(azic(t)*pi/180);
        ycat_guess = ycat(t) + dist/(2*vd)*vc*sin(azic(t)*pi/180);

        % check if cat is running in the direction of the dog
        if dist > distance(xcat_guess,ycat_guess,xdog(i,t),ydog(i,t))
            angd(i,t) = atan2(ycat(t)-ydog(i,t),xcat(t)-xdog(i,t))*180/pi-
azid(i,t);
        else
            angd(i,t) = atan2(ycat_guess-ydog(i,t),xcat_guess-
xdog(i,t))*180/pi-azid(i,t);
        end
    else
        % far from cat and it is not running toward dog, lead cat with
appropriate angle
        angd(i,t) = atan2(ycat(t)-ydog(i,t),xcat(t)-xdog(i,t))*180/pi-
azid(i,t);
    end

    % check for valid angle between -pi and pi radians
    if angd(i,t) > 180
        angd(i,t)=angd(i,t)-360;
    elseif angd(i,t) < -180
        angd(i,t)=angd(i,t)+360;
    end

end

% FCL control for cat, not used in final script
% Not used because considered not realistic motion of cat
%
% % fuzzify cat input variable to FLC: ang
% ac= fuzify(M,angc(t),sang);
% % fuzzy inferencing cat
% outc = infer(rule,d,M,0,ac);
% % defuzzification cat
% dzc(t+1)=defuz(outc,sdz);

```

```

% do operations on all dogs
for i = 1:ndogs
    % fuzzify dog input variable to FLC: ang
    ad(i,:) = fuzify(M,angd(i,t),sang);
    % fuzzy inferencing dog
    outd(i,:) = infer(rule,d,M,0,ad(i,:));
    % defuzzification dog
    dzd(i,t+1) = defuz(outd(i,:),sdz);

    % determine next bearing of dog
    azid(i,t+1) = azid(i,t) + dzd(i,t+1);
    if azid(i,t+1) > 180, % make sure -pi <= azi <= pi
        azid(i,t+1) = azid(i,t+1) - 360;
    elseif azid(i,t+1) < -180,
        azid(i,t+1) = azid(i,t+1) + 360;
    end

    % determine new dog position
    xdog(i,t+1) = xdog(i,t) + vd*cos(azid(i,t+1)*pi/180);
    ydog(i,t+1) = ydog(i,t) + vd*sin(azid(i,t+1)*pi/180);
end

% determine next bearing of cat based on closest dog
azic(t+1) = azid(closest_dog,t+1); % standard A.I. - run away from closest
dog
% azic(t+1) = azic(t) + dzc(t+1); % FCL bearing calculation
if azic(t+1) > 180, % make sure -pi <= azi <= pi
    azic(t+1) = azic(t+1) - 360;
elseif azic(t+1) < -180,
    azic(t+1) = azic(t+1) + 360;
end

% determine new cat position
xcat(t+1) = xcat(t) + vc*cos(azic(t+1)*pi/180);
ycat(t+1) = ycat(t) + vc*sin(azic(t+1)*pi/180);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%      Termination Code      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% dog within reach of cat
for i = 1:ndogs
    % if current dog is within reach of cat, it's caught
    if sqrt( (xcat(t+1)-xdog(i,t+1))^2 + (ycat(t+1)-ydog(i,t+1))^2 ) <= 2
        terminate=1;
    end
end

% cat was caught by at least one dog
if terminate == 1
    % caught(q) = caught(q) + 1;
    msg = ['The pack of ' num2str(ndogs) ' dogs caught the cat.'];
    disp(msg)
end

```

```

% check if the cat made it off the screen or t >= 300
if abs(xcat(t)) >= 300 | abs(ycat(t)) >= 300 | t >= 500
    terminate=1;
    %escape(q) = escape(q) + 1;
    msg = ['The cat evaded ' num2str(ndogs) ' dogs and escaped.'];
    disp(msg)
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Graphical Output (no trails) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% draw graphic of chase without trails once every 5 iterations or on
termination
if rem(t,5)==0 | terminate==1,
    clf
    hold on

    %plot cat
    plot(xcat(t),ycat(t),'ro')

    %plot dogs
    for i = 1:ndogs
        plot(xdog(i,t),ydog(i,t),'bo')
    end

    title('Dogs Chasing Cat (no trails)')
    legend('Cat','Dogs')
    xlabel('x')
    ylabel('y')
    axis([-300 300 -300 300])
    hold off

    drawnow
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Graphical Output (trails) %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% % draw graphic of chase with trails once every 5 iterations or on
termination
% if rem(t,5)==0 | terminate==1,
%     clf
%     hold on
%
%     % plot cat
%     plot(xcat,ycat,'ro')
%
%     % plot dogs' paths
%     for i = 1:ndogs
%         plot(xdog(i,:),ydog(i,:), 'bo')
%     end
%
%     title('Dogs Chasing Cat (with trails)')
%     legend('Cat','Dogs')

```

```
% xlabel('x')
% ylabel('y')
% axis([-300 300 -300 300])
% hold off
%
% drawnow
% end

end % t-loop

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Testing Code %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% end % end testing p-loop
%
% % do testing calculations for output
% trials(q) = caught(q) + escape(q);
% escape_percent(q) = 100 * escape(q) / trials(q);
% caught_percent(q) = 100 * caught(q) / trials(q);
%
% % display testing output
% disp('Total Trials = ')
% disp(trials)
% disp('Caught Percentage = ')
% disp(caught_percent)
% disp('Escaped Percentage = ')
% disp(escape_percent)
%
% end % end testing q-loop
```

APPENDIX B: Matlab Source Code, Closest Dog Calculator

Matlab Source Code (closest.m):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS539
% Semester Project
% Closest Dog Determiner
% closest.m
%
% Eric Olson - 9016984685
% CAE Login - eolson
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% determines the id number of closest dog to cat

function close = closest()

% define global variables
global xcat
global ycat
global xdog
global ydog
global t
global ndogs

% initialize close and min_dist
close = 0;
min_dist = 1000;

% get coordinates of cat
x1 = xcat(t);
y1 = ycat(t);

% loop through all dogs
for i = 1:ndogs

    % get coordinates of current dog
    x2 = xdog(i);
    y2 = ydog(i);

    % calculate distance of current dog to cat
    cur_dist = sqrt( (y2-y1)^2 + (x2-x1)^2 );

    % check if dog is newest closest
    if cur_dist < min_dist
        % store minimum distance and dog number
        min_dist = cur_dist;
        close = i;
    end

end % dog for loop

return
```

APPENDIX C: Matlab Source Code, Distance Formula

Matlab Source Code (distance.m):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS539
% Semester Projct
% Distance Function
% distance.m
%
% Eric Olson - 9016984685
% CAE Login - eolson
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% calculated distance between two points

function dist = distance(x1,y1,x2,y2)

% calculate distance of current dog to cat
dist = sqrt( (y2-y1)^2 + (x2-x1)^2 );

return
```

APPENDIX D: Matlab Source Code, Graphical Analysis

Matlab Source Code (graph.m):

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CS539
% Semester Project
% Graphical Data Analysis
% graph.m
%
% Eric Olson - 9016984685
% CAE Login - eolson
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Display data collected in massive iterations for graphical analysis of
% results

% ndogs values tested
ndogs = [1 2 3 4 5 8 10 15 20 25 30 40 50 100];

% Max-Min rule results
caught_percent1 = [0 1 14 13 24 46 55 60 72 75 77 85 89 98];
escape_percent1 = [100 99 86 87 76 54 45 40 28 25 23 15 11 2];

% Kiosko's Max-Product rule results
caught_percent2 = [0 0 4 15 24 37 53 62 72 68 70 82 84 98 ];
escape_percent2 = [100 100 96 85 76 63 47 38 28 32 30 18 16 2];

figure(1)
hold on
plot(ndogs, caught_percent1, 'b', ndogs, escape_percent1, 'r')
title('Max-Min rule: Caught and Escape Percentages vs. Number of Dogs')
xlabel('Number of Dogs')
ylabel('Percent')
legend('Caught','Escape')
hold off

figure(2)
hold on
plot(ndogs, caught_percent2, 'b', ndogs, escape_percent2, 'r')
title('Koskos max-product rule: Caught and Escape Percentages vs. Number of
Dogs')
xlabel('Number of Dogs')
ylabel('Percent')
legend('Caught','Escape')
hold off

figure(3)
hold on
plot(ndogs, caught_percent1, 'b', ndogs, caught_percent2, 'r')
title('Percent Caught for Max-Min Rule vs. Koskos max-product rule')
xlabel('Number of Dogs')
ylabel('Percent')
legend('Max-Min Rule','Koskos max-product rule')
hold off
```

APPENDIX E: Required Files

Matlab M-Files Required for Simulation

Table 1

Title	Filename	Creator
Dog Chasing Cat Simulator	chase.m	Eric Olson
Closest Dog Calculator	closest.m	Eric Olson
Distance Formula	distance.m	Eric Olson
Fuzzification Script	fuzify.m	Yu Hen Hu
Defuzzification Script	defuz.m	Yu Hen Hu
Inference Script	infer.m	Yu Hen Hu
Dog-Cat Rules	ruledc.m	Yu Hen Hu

Auxiliary Files

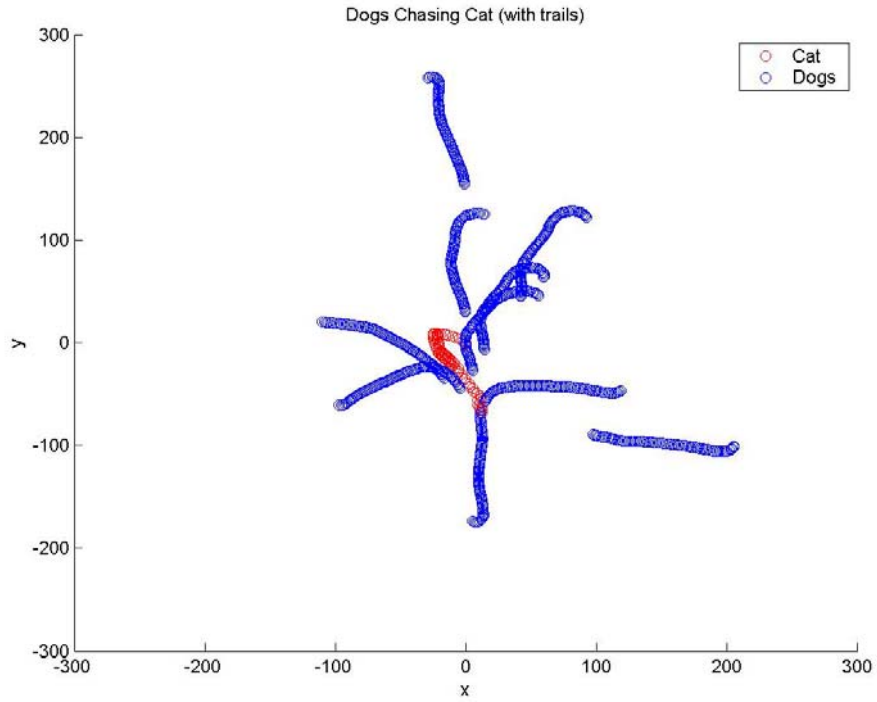
Table 2

Title	Filename	Creator	Purpose
Project Report	eolson.doc	Eric Olson	Summary and Analysis of Project
Project Presentation	eolson.ppt	Eric Olson	Powerpoint Presentation of Project
Graphical Analysis of Max-Min and Kosko's Max-Product	graph.m	Eric Olson	Graphical Analysis

APPENDIX F: Sample Graphical Representations

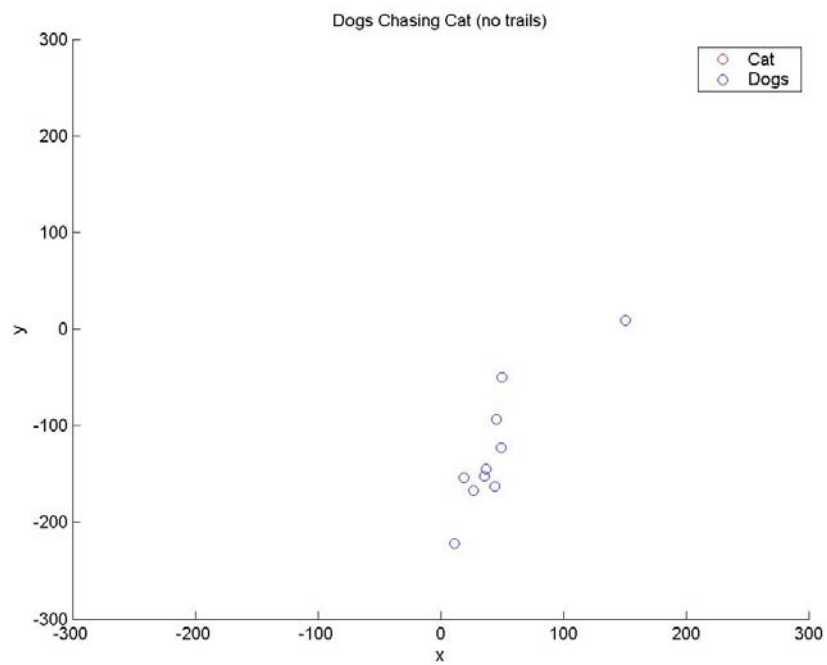
Graphical Output with Pet Trails

Figure 1



Graphical Output without Pet Trails

Figure 2



APPENDIX G: Testing Results, Max-Min

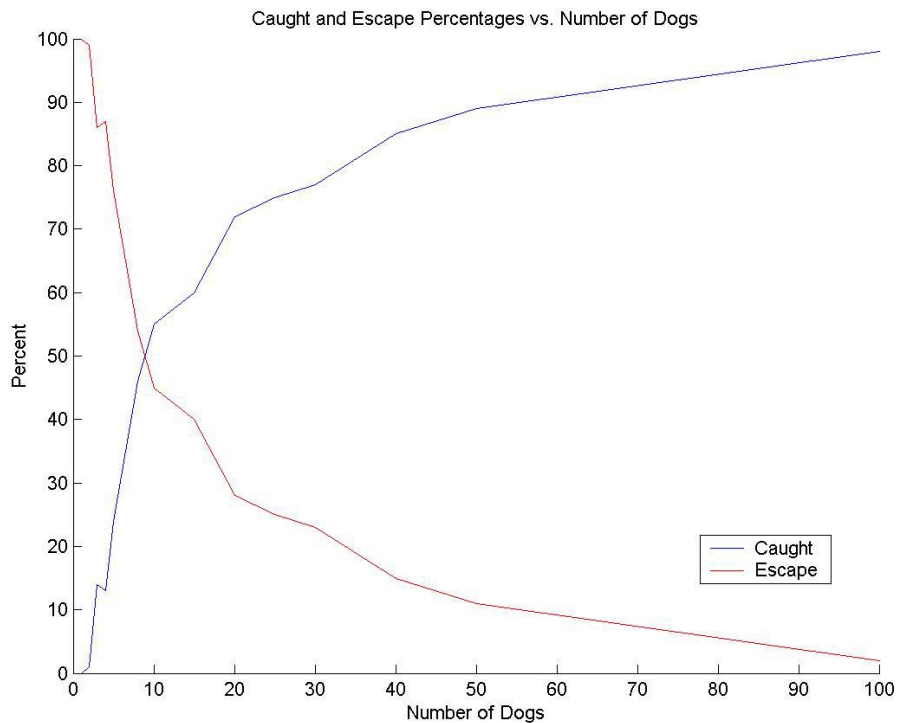
Testing using random dog locations and usual max-min rule

Table 3

Number of Dogs	Total Trials	Caught Percentage	Escaped Percentage
1	100	0	100
2	100	1	99
3	100	14	86
4	100	13	87
5	100	24	76
8	100	46	54
10	100	55	45
15	100	60	40
20	100	72	28
25	100	75	25
30	100	77	23
40	100	85	15
50	100	89	11
100	100	98	2

Graphical Representation of Results:

Figure 3



APPENDIX H: Testing Results, Kosko's Max-Product

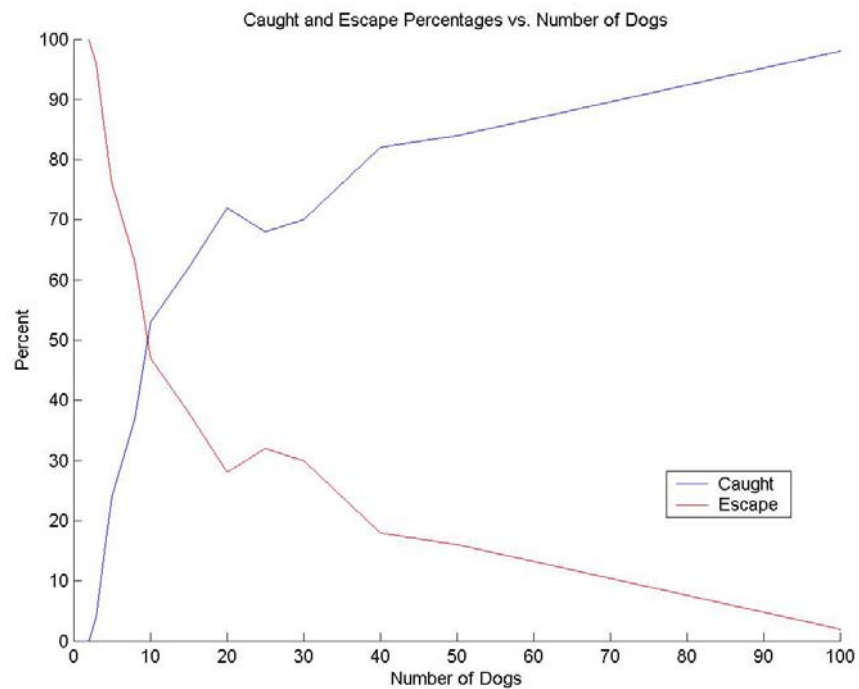
Testing using random dog locations and Kosko's max-product rule

Table 4

Number of Dogs	Total Trials	Caught Percentage	Escaped Percentage
1	100	0	100
2	100	0	100
3	100	4	96
4	100	15	85
5	100	24	76
8	100	37	63
10	100	53	47
15	100	62	38
20	100	72	28
25	100	68	32
30	100	70	30
40	100	82	18
50	100	84	16
100	100	98	2

Graphical Representation of Results:

Figure 4



APPENDIX I: Graphical Analysis, Max-Min vs. Kosko's Max-Product Rule

Max-Min Rule vs. Kosko's max-product rule

Figure 5

