Matlab Functions
for
Obstacle Avoidance and Light Seeking

Richard G. Goyette May 97
goyette@rmc.ca

created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

1.1 Introduction

This document should be read in conjunction with the thesis document “A Development Environment for Experiments in Autonomous Robotics” by Rich Goyette.

1.2 Overview

The Matlab sofware in this document was written to implement a controller capable of learning to approach a light source while avoiding obstacles. The Matlab software in this document is used to control a real Khepera mobile robot and a simulated version of the Khepera in real time. The *Matlab-Khepera Toolbox* provides the communication interface between the Khepera and the software in this document. See the *Khepera Toolbox User's Manual*.

The learning system of each module is based on the Simple Recurrent Network. The Matlab *Neural Network Toolbox* provides code for implementing Elman Recurrent Networks and this code was used to implement to learning system here. However, some implementation changes were required in the original code in order to make it suitable for use. See the thesis document for details on the required changes.

created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

1.3 Matlab Functions

The following table lists each function and provides a short description of what each does. The code for the functions is listed in the pages following the table.

Matlab Functions	
phoebus.m	Loads and executes trained light and obstacle avoidance modules
template.m	Most functions are interfaced to the Matlab-Khepera toolbox by writing the executable code into this template file.
avoidobj.m	Calls all the functions required to train a simple recurrent network to avoid obstacles
rpobst.m	The reinforcement program for obstacle avoidance.
seeklite.m	Calls all the functions required to train a simple recurrent network to seek light
rplight.m	The reinforcement program for light seeking
preproo.m	The input representation function for obstacle avoidance
preprol.m	The input representation function for light seeking
elmsim1c.m	Produces the simple recurrent network output in response to a single input vector
binout.m	Stochastically produces a binary valued output vector with the same length as the input vector using uniformly distributed random numbers
xplore.m	Stochastically produces a binary valued output vector with the same length as the input vector using a Gaussian distribution of random numbers
actuator.m	Executes an action on the real Khepera based on the vector input to the function
trnelm1p.m	Trains the simple recurrent network for one iteration
simavoid.m	Calls all the functions required to train a simple recurrent network to avoid obstacles in a simulated environment

created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

Matlab Functions	
simlite.m	Calls all the functions required to train a simple recurrent network to seek light in a simulated environment
sim_act.m	Executes an action on the simulated Khepera based on the vector input to the function

Table 1-1
Matlab Functions for Obstacle Avoidance/Light Seeking

created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function phoebus()
%PHOEBUS    The integration function for all behaviours
%           This function is the switch architecture.  It asks the user to
%           load pre-trained weights and biases for obstacle avoidance and
%           light seeking.  These are then used to run the Khepera in the test
%           mode.
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--


global port_id
port_id=2;

%  Load the weights, biases, and network parameters of the modules

%  Load obstacle avoidance
[save_file save_path]=uigetfile('*.mat','Load Obstacle Parameters...');
if save_file~=0
      filepath=[save_path save_file];
      load(filepath);
end

% Since the weights are called the same, convert the names and clear the
loaded
% variables

WO1=W1;
WO2=W2;
bo1=b1;
bo2=b2;
obst_context=a1_previous;
clear W1 W2 b1 b2 a1_previous

%  Load lite seeking
[save_file save_path]=uigetfile('*.mat','Load Light Parameters...');
if save_file~=0
      filepath=[save_path save_file];
      load(filepath);
end

% Since the weights are called the same, convert the names and clear the
loaded
% variables

WL1=W1;
WL2=W2;
bl1=b1;
bl2=b2;
light_context=a1_previous;
clear W1 W2 b1 b2 a1_previous
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
%  Initialize the scaling factor for the tansig layer in the network
%  and the scaling factor for the random distribution.
tscale=0.04;
randscale=40;


q=0;
while q==0

      %  DATA INPUT
      %  Get input data.  Use preproo.m for obstacle avoidance, preprol.m for
            light seeking

      %  OBSTACLE INPUT
      obst_vector(1:8)=preproo(rps(port_id));

      %  DELAY
      for i=1:2000
            % Delay so that the comm line isn't tied up
      end

      %  LIGHT INPUT
      light_vector(1:8)=preprol(rls(port_id));

      % ELMAN OUTPUT FOR EACH MODULE.
      % Notice that we don't use elmsim1c.m. simelm1c.m is used because it
      % accepts the context layer input as an argument
%OBSTACLE MODULE
[obst_a1,obst_a2]=simelm1c(obst_vector',WO1,bo1,WO2,bo2,obst_context,tscale);

% LIGHT Module
[light_a1,light_a2]=simelm1c(light_vector',WL1,bl1,WL2,bl2,light_context,tscal
e);

% Now delay the context layer
obst_context=obst_a1;
light_context=light_a1;


      % THRESHOLD OUTPUTS
      % Obst_a2 and light_a2 correspond to the network output vectors
      obst_out=round(obst_a2);
      light_out=round(light_a2);

      % SWITCH ARCHITECTURE CONTROL BEGINS HERE

      % If any of proximity sensors are on, then give control to obst avoid
      if ~any(obst_vector)
            motor_vector=light_out';
            actuator(motor_vector,port_id);
      % Otherwise, give control to obstacle avoidance
      else
            motor_vector=obst_out';
            actuator(motor_vector,port_id);
      end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  DELAY
      for i=1:2000
            % Delay so the line isn't tied up
      end
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function [%%Return
Variables%%]=%%name%%(port_id,cycles_per_run,runs,pause_stat)
%TEMPLATE   This is a function template
%           Wherever %% appears, some substitution is required
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
% V1.0
% 04/11/96
%
%-------------------------------------------------------------------------
--

%  INCLUDE ALL VARIABLES THAT NEED TO BE REMEMBERED BY THE FUNCTION IN THE
%  GLOBAL STATEMENT BELOW
%  ========================================================================

global port_id cycle_count run_count
%%global

%  IF THE FUNCTION IS CALLED WITHOUT ARGUMENT, THEN INITIALIZE ALL OF THE
%  VARIABLES ABOVE AND CALL OTHER REQUIRED FUNCTIONS WITHOUT ARGUMENT TO
%  INITIALIZE THEM TOO
%  ========================================================================
if nargin<1
      %% Initialize all variables here
      %  Loop variables
      cycle_count=0;
      run_count=0;
      return;
%  OTHERWISE, THE FUNCTION WAS CALLED WITH AN ARGUMENT (EVEN A DUMMY) SO
PERFORM
%  THE LEARNING AND DATA GATHERING AND RETURN THE APPROPRIATE VARIABLES
else
  while run_count<(runs)
    while cycle_count<(cycles_per_run)

      %% PLACE YOUR CODE HERE

      %% LOOP CODE - DO NOT REMOVE
      cycle_count=cycle_count+1;
    end
    % END OF RUN.  EVERYTHING TO BE DONE BETWEEN RUNS SHOULD BE DONE HERE
    run_count=run_count+1;
    cycle_count=0;
    if pause_stat==1
      [save_file save_path]=uiputfile('*.mat','Save Matlab Variables As...');
      if save_file~=0
            filepath=[save_path save_file];
            save(filepath);
      else
            warndlg('Data not saved!');
      end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
    end
  end
%% PLACE ANY HOUSEKEEPING COMMANDS HERE (IE SMS(2,0,0) TO STOP REAL KHEP)
return;
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function avoidobj(port_id,cycles_per_run,runs,pause_stat)
%AVOIDOBJ   Simple Recurrent Network Learning Agent
%
%     This function iteratively calls all of the functions required to train
an
%     Elman simple recurrent network to avoid obstacles.  The function
receives
%     number of cycles/run, number of runs, port_id, etc from within the
%     Roborunner GUI.  The weights and biases W1,W2,b1,b2, of the trained
%     network are saved at the end of each run, so they are not required to be
%     passed back to the GUI.
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
% V1.0
% 04/11/96
%
------------------------------------------------------------------------------
--

%  INCLUDE ALL VARIABLES THAT NEED TO BE REMEMBERED BY THE FUNCTION IN THE
%  GLOBAL STATEMENT BELOW
%  ========================================================================

%%%
% LIST THE VARIABLES THAT NEED INITIALIZATION AFTER
% THE GLOBAL STATEMENT BELOW.
%%%

%-------------------------------------------------------------------------
global P s1 s2 W1 b1 W2 b2 cycle_count run_count a1_previous learning_rate
global mse1 mse2 rwd_series tscale randscale
%-------------------------------------------------------------------------

global cycle_count run_count
%  IF THE FUNCTION IS CALLED WITHOUT ARGUMENT, THEN INITIALIZE ALL OF THE
%  VARIABLES ABOVE
%  ========================================================================
if nargin<1

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

      %%
      % Initialize all your variables here
      %%
      %  Set up the initial input vectors to initialize the elman network
      %  The following vector chooses the max and min of each input of the
      %  network.
      %  8 are for proximity, 2 are for motor state.

      P=[0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10];

      %  The network will be PXS1XS2, using logsig output layer, where P is
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
the
      %  input vector size (8 light sensors and maybe 2 motor states depending
      %  on the current implementation).
      s1=8;
      s2=2;

      %  Weights and biases
      [W1,b1,W2,b2]=initelm2(P,s1,s2);

      %  Make the following calls to other functions without arguments.  Each
of
      %  these functions has variables that will be initialized if the
function
      %  is called with nargin==0.
      %  Initialize the reinforcement program
      rpobst;

      %  Initialize the scaling factor for the tansig layer in the network
      %  and the scaling factor for the random distribution.
      tscale=0.04;
      randscale=40;

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

      %  Loop variables.  Do not remove
      cycle_count=0;
      run_count=0;
      return;

else
  while run_count<(runs)
    while cycle_count<(cycles_per_run)
      %-----------------------------------------------------------------------

      %%%
      % PLACE YOUR CODE HERE
      %%%
      %  Block 1:  Get data
      % Below is the input from the proximity sensors.
      input_vector(1:8)=preproo(rps(port_id));

      %  Block 2:  Compute output of elman network.
      [a1,a2,a1_previous]=elmsim1c(input_vector',W1,b1,W2,b2,tscale);

      %  Block 3:  Stochastic Binary Output of the search vector using either
      %  uniform or Gaussian random distribution.
      %motor_vector=xplore(a2',randscale);
      motor_vector=binout(a2');

      %  Block 4:  Execute action
      actuator(motor_vector,port_id);

      %  Block 6:  Compute the corrections to the network based on the reward
      %  function
[target_vector,learning_rate,reward_value]=rpobst(motor_vector,a2',port_id );
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  Track some statistics
      rwd_series=[rwd_series reward_value];

      %  Block 7:  Now train the Elman network for one pass using the target
      %  vector, learning rate, and input vector.
      [W1,b1,W2,b2]=trnelm1p(W1,b1,W2,b2,input_vector',...
                  target_vector',a1_previous,a1,a2,learning_rate);


      %  Go back and do it again

      %--------------------------------------------------------------------
      cycle_count=cycle_count+1;
    end

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

    %%%
    % END OF RUN.  EVERYTHING TO BE DONE BETWEEN RUNS SHOULD BE DONE HERE
    %%%
    sms(2,0,0);
    %--------------------------------------------------------------------
    run_count=run_count+1;
    cycle_count=0;
    if pause_stat==1
      %  If the GUI is set to save between runs, then save the weights and
      %  biases
      [save_file save_path]=uiputfile('*.mat','Save Matlab Variables As...');
      if save_file~=0
            filepath=[save_path save_file];
            save(filepath);
      else
            warndlg('Data not saved!');
      end
    end
    [W1,b1,W2,b2]=initelm2(P,s1,s2);
    mse=[];
    rwd_series=[];
  end

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

%%%
% PLACE ANY HOUSEKEEPING COMMANDS HERE (IE SMS(2,0,0) TO STOP REAL KHEP)
%%%
sms(2,0,0)
%--------------------------------------------------------------------
return;
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function[target_vector,learning_rate,reward_value]
      =rpobst(binary_vector,search_vector,port_id)
%RPOBST     This function provides the scalar evaluation of behaviour for the
%           obstacle avoidance.
%
%     [TARGET_VECTOR,LEARNING_RATE]=reward(BINARY_VECTOR,PORT_ID)
%       TARGET_VECTOR - The vector to which the network is to try to con-
%                   verge on this pass
%       LEARNING_RATE - The learning rate for back-prop
%       REWARD_VALUE  - Pass back the reward value for use by stats gathering
%                       fns
%       CURRENT_VAL   - The current sensor readings sent back by this
function.
%       BINARY_VECTOR - The vector that was just fed to the motors
%       SEARCH_VECTOR - The vector that was output by the network
%       PORT_ID       - The port to read sensor values from.

%
%     This is the "teacher" or "evaluator" in the system.  This
%     function should be all that is required to tailor the
%     networks to learn specific behaviours.
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

%  If this function is called without arguments, then initialize any values
that
%  must be preserved from call to call.
if nargin < 2
      return;
end


% Get sensor data.
tmp=preproo(rps(port_id));
current_val=tmp(1:6);

fwd_vect=[1 1];
left_vect=[0 1];
right_vect=[1 0];
rev_vect=[0 0];
thrshld=0.01;


% Depending on the conditions for reward and punishment, decide
% which reward value to assign

% If all the front four sensors are less than 10 and moving forward, reward.

if (all(previous_val(2:5)<10))&(binary_vector==fwd_vect);
      %  Now, if the absolute difference between the search vector and the
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  binary vector is greater than thrshld, then reward.  Otherwise, the
net   %  has learned, thus leave it alone.
      delta=abs(abs(binary_vector)-abs(search_vector));
      if all(delta<thrshld)
            disp('not rewarded for moving forward')
      else
            disp('rewarded for moving forward')
            reward_value=1;
      end

% Otherwise, if any front sensor equals 10 then punish for moving forward
elseif any(previous_val(2:5)==10)&(binary_vector==fwd_vect)
      reward_value=-1;
      disp('punishing for running into wall');


% If turning right, reward if there was a reason, punish otherwise
elseif (binary_vector==right_vect)

      %  If the right turn resulted in the sensors on the left being greater
      %  than the sensors on the left, then the move was successful.
      %  Wall following is allowed if the outer 90 degree sensors are ignored
      if
(sum(current_val(2:3))>0)&(sum(current_val(2:3))>sum(current_val(4:5)))
            delta=abs(abs(binary_vector)-abs(search_vector));
            if all(delta<thrshld)
                  disp('not rewarded for turning right')
            else
                  disp('rewarded for turning right')
                  reward_value=1;
            end
      %  Otherwise, if there is now something on the right and the right turn
      %  has made it larger than the left, then punish
      elseif
(sum(current_val(4:5))>0)&(sum(current_val(4:5))>=sum(current_val(2:3)))
            reward_value=-1;
            disp('Punished for turning right into wall');

      %  Otherwise, if there **was** nothing on the left, then punish for a
      %  spontaneous right turn.
      elseif (all(previous_val(2:3)<10))
            reward_value=-1;
            disp('punish spontaneous right turn')
      end

% If turning left, reward if there was a reason
elseif (binary_vector==left_vect)

      if
(sum(current_val(4:5))>0)&(sum(current_val(4:5))>sum(current_val(2:3)))

            delta=abs(abs(binary_vector)-abs(search_vector));
            if all(delta<thrshld)
                  disp('not rewarded for turning left')
            else
                  disp('rewarded for turning left')
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
                  reward_value=1;
            end

      elseif
(sum(current_val(2:3))>0)&(sum(current_val(2:3))>=sum(current_val(4:5)))

            reward_value=-1;
            disp('Punished for turning left into wall');

      elseif (all(previous_val(4:5)<10))
            reward_value=-1;
            disp('punish spontaneous left turn')
      end

% Punish for going backwards without reason.
elseif (binary_vector==rev_vect)
      % Reward if all the front sensors are blocked, or if the left two
      % outermost and right two outermost are 10 and the inside ones are zero
      if(all(previous_val(1:6)>0)|((previous_val(1:2)>0)&(previous_val(5:6)>0)
&
            (previous_val(3:4)==0)))
            reward_value=1;
            disp('Reward for backing out of crappy location')
      else
            reward_value=-1;
            disp('punishing for backup')
      end

end

if reward_value==[]
      reward_value=0;
      fprintf('no value.  motors=%d %d\n',binary_vector(1),binary_vector(2))
end


% Compute reward_value based on current sensor readings

if reward_value == 1
      % Rewarded!
      learning_rate=0.3;
      target_vector=(binary_vector);

elseif reward_value == -1
      % Punished!
      learning_rate=0.1;
      target_vector=([1 1]-binary_vector);

elseif reward_value == 0
      % Nothing
      learning_rate=1;
      target_vector=(binary_vector);

end

function seeklite(port_id,cycles_per_run,runs,pause_stat)
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
%SEEKLITE   Simple Recurrent Network Learning Agent
%
%     This function iteratively calls all of the functions required to train
      an Elman simple recurrent network to seek light.  The function receives
      number of cycles/run, number of runs, port_id, etc from within the
      Roborunner GUI.  The weights and biases W1,W2,b1,b2, of the trained
      network are saved at the end of each run, so they are not required to be
            passed back to the GUI.
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
% V1.0
% 04/11/96
%
--------------------------------------------------------------------------
--

%  INCLUDE ALL VARIABLES THAT NEED TO BE REMEMBERED BY THE FUNCTION IN THE
%  GLOBAL STATEMENT BELOW
%  ========================================================================

%%%
% LIST THE VARIABLES THAT NEED INITIALIZATION AFTER
% THE GLOBAL STATEMENT BELOW.
%%%

%-------------------------------------------------------------------------
global P s1 s2 W1 b1 W2 b2 cycle_count run_count a1_previous learning_rate
global mse rwd_series tscale randscale
%-------------------------------------------------------------------------

global cycle_count run_count
%  IF THE FUNCTION IS CALLED WITHOUT ARGUMENT, THEN INITIALIZE ALL OF THE
%  VARIABLES ABOVE
%  ========================================================================
if nargin<1

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

      %%
      % Initialize all your variables here
      %%

      %  Set up the initial input vectors to initialize the elman network
      %  The following vector chooses the max and min of each input of the
      %  network.
      %  8 are for light, 2 are for motor state.

      %  Init vector without motor state feedback
      P=[0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10];

      %  The network will be PXS1XS2, using logsig output layer, where P is
the
      %  input vector size (8 light sensors and maybe 2 motor states depending
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  on the current implementation).
      s1=8;
      s2=2;

      %  Initialize weights and biases
      [W1,b1,W2,b2]=initelm2(P,s1,s2);

      %  Make the following calls to other functions without arguments.  Each
of
      %  these functions has variables that will be initialized if the
function
      %  is called with nargin==0.
      %  Initialize the reinforcement program
      rplight;

      %  Initialize the scaling factor for the tansig layer in the network
      %  and the scaling factor for the random distribution.
      tscale=0.04;
      randscale=40;

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

      %  Loop variables.  Do not remove
      cycle_count=0;
      run_count=0;
      return;

else
  while run_count<(runs)
    while cycle_count<(cycles_per_run)
      %-----------------------------------------------------------------------

      %%%
      % PLACE YOUR CODE HERE
      %%%

      %  BLOCK 1:  Get data
      %  Below is the input from the light sensors
      input_vector(1:8)=preprol(rls(port_id));

      % Below is the recurrent connection from the motor sensors back
      % input_vector(9:10)=rms(port_id);

      %  BLOCK 2:  Compute output of elman network.
      [a1,a2,a1_previous]=elmsim1c(input_vector',W1,b1,W2,b2,tscale);

      %  BLOCK 3:  Confidence interpretation
      %  motor_vector=binout(a2'); % too liberal (uniform rand function)
      %  motor_vector=round(a2');  % too strict - no exploration
      motor_vector=xplore(a2',randscale);

      %  BLOCK 4:  Execute action
      actuator(motor_vector,port_id);

      %  BLOCK 6:  Compute the corrections to the network based on the reward
      %  function
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      [target_vector,learning_rate,reward_value]=rplight(motor_vector,a2',port
_id);

      %  Track error statistics for later comparison
      rwd_series=[rwd_series reward_value];

      %  BLOCK 7:  Now train the elman network for one pass using the target
      %  vector, learning rate, and input vector.
      [W1,b1,W2,b2]=trnelm1p(W1,b1,W2,b2,input_vector',...
                  target_vector',a1_previous,a1,a2,learning_rate);

      %-------------------------------------------------------------------
      cycle_count=cycle_count+1;
    end

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

    %%%
    % END OF RUN.  EVERYTHING TO BE DONE BETWEEN RUNS SHOULD BE DONE HERE
    %%%
    sms(2,0,0);
    %-------------------------------------------------------------------
    run_count=run_count+1;
    cycle_count=0;
    if pause_stat==1
      [save_file save_path]=uiputfile('*.mat','Save Matlab Variables As...');
      if save_file~=0
            filepath=[save_path save_file];
            save(filepath);
      else
            warndlg('Data not saved!');
      end
    end
    [W1,b1,W2,b2]=initelm2(P,s1,s2);
    mse=[];
    rwd_series=[];
  end

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

%%%
% PLACE ANY HOUSEKEEPING COMMANDS HERE (IE SMS(2,0,0) TO STOP REAL KHEP)
%%%
sms(2,0,0)
%-----------------------------------------------------------------------
return;
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function[target_vector,learning_rate,reward_value]
      =rplight(binary_vector,search_vector,port_id)
%RPLIGHT    This function provides the scalar evaluation of behaviour for the
%           light seeking.
%
%     [TARGET_VECTOR,LEARNING_RATE]=slrwd(BINARY_VECTOR,PORT_ID)
%       TARGET_VECTOR - The vector to which the network is to try to con-
%                   verge on this pass
%       LEARNING_RATE - The learning rate for back-prop
%       REWARD_VALUE  - Pass back the reward value for use by stats gathering
%       CURRENT_VAL   - The current sensor readings sent back by this
function.
%       BINARY_VECTOR - The vector that was just fed to the motors
%       SEARCH_VECTOR - The vector that was output by the network
%       PORT_ID       - The port to read sensor values from.
%
%
%     This is the "teacher" or "evaluator" in the system.  This
%     function should be all that is required to tailor the
%     networks to learn specific behaviours.
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

%  If this function is called without arguments, then initialize any values
that
%  must be preserved from call to call.
if nargin < 2
      return;
end


% Get sensor data.
% Put the sensor data through a pre-processor which provides a positive value
% at the minimum point to make the learning easier.
ldata=preprol(rls(port_id));

% Define some working constants
fwd_vect=[1 1];
left_vect=[0 1];
right_vect=[1 0];
rev_vect=[0 0];
thrshld=0.01;

% Depending on the conditions for reward and punishment, decide
% which reward value to assign

[value,idx]=max(ldata);
fprintf('idx=%d',idx);

% If the action chosen was to move forward,
if (binary_vector==fwd_vect)
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  If the robot moved forward, and the max prepro reading ended up
      %  on the front two sensors, then reward this action
      if ((idx==3)|(idx==4))
            % Compute how close the binary vectors are to the stochastic
      % vectors.  If within threshold, then the robot is deemed to have
      % learned.  Stop rewarding.
            delta=abs(abs(binary_vector)-abs(search_vector));
            if all(delta<thrshld)
                  fprintf(' not rewarded for moving forward\n')
            else
                  fprintf(' rewarded for moving forward\n')
                  % This is the most important behaviour, so assign a reward
            % value that results in a higher learning rate
                  reward_value=1;
            end
      %  If the robot moved forward and did not result in a max on the front
two
      %  sensors, then punish the movement.
      elseif ((idx~=3)&(idx~=4))
            fprintf(' punished for moving forward with max light not
forward\n')
            reward_value=-1;
      end

% If the action chosen was to turn right:
elseif (binary_vector==right_vect)
      % If turning right, reward if the light minimum appears on the right
      if (idx==5)|(idx==6)|(idx==7)|(idx==4)
            % Compute how close the binary vectors are to the stochastic
      %vectors.  If within threshold, then the robot is deemed to have
      %learned.  Stop rewarding.
            delta=abs(abs(binary_vector)-abs(search_vector));
            if all(delta<thrshld)
                  fprintf(' not rewarded for turning right-already learned\n')
            else
                  fprintf(' rewarded for turning right into light\n')
                  reward_value=1;
            end
      % If the right turn did not result in the minimum being on any of the
      % following, then punish (use just and elseif punish when you take out
min   % constraint)
      elseif (idx~=5)|(idx~=6)|(idx~=7)|(idx~=4)
            reward_value=-1;
            fprintf(' punishing for turning right away from light\n');
      end

% If the action chosen was to turn left:
elseif (binary_vector==left_vect)
      % If turning left, reward if the light minimum appears on the left
      if (idx==1)|(idx==2)|(idx==8)|(idx==3)
            % Compute how close the binary vectors are to the stochastic
      % vectors.  If within threshold, then the robot is deemed to have
      % learned.  Stop rewarding.
            delta=abs(abs(binary_vector)-abs(search_vector));
            if all(delta<thrshld)
                  fprintf(' not rewarded for turning left - already
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
learned\n')
            else
                  fprintf(' rewarded for turning left into light\n')
                  reward_value=1;
            end

      % If the left turn did not result in the minimum being on any of the
      % following,then punish (use just and elseif punish when you take out
min   % constraint)
      elseif (idx~=1)|(idx~=2)|(idx~=3)|(idx~=8)
            reward_value=-1;
            fprintf(' punishing for turning left away from light\n');
      end

% Punish for going backwards at any time.
elseif (binary_vector==rev_vect)
      reward_value=-1;
      fprintf(' punishing for backup\n')

end

if reward_value==[]
      reward_value=0;
      fprintf(' Nothing done.  Value=%d, binary
value=%d,%d\n',value,binary_vector(1),binary_vector(2));
end


% Compute reward_value based on current sensor readings

if reward_value == 1
      % Rewarded!
      learning_rate=0.3;
      target_vector=(binary_vector);

elseif reward_value == -1
      % Punished!
      learning_rate=0.1;
      target_vector=([1 1]-binary_vector);

elseif reward_value == 0
      % Nothing
      learning_rate=1;
      target_vector=(binary_vector);

end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function [processed_input_vector]=preproo(input_vector)
%PREPROO    Input representation function for obstacle avoidance module
%
%     [PROCESSED_INPUT_VECTOR]=preproo(INPUT_VECTOR)
%     PROCESSED_INPUT_VECTOR - The input vector reduced to a "10" in
%                         all positions where the value > 400,
%                         zero otherwise
%     INPUT_VECTOR - The original input vector in the range [0,1024]
%
%
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

vect_length=length(input_vector);

% Choose a threshold of 100 to reduce spontaneous values
if max(input_vector)>100
      % Make all entries containing a value 10, all others
      % 0.
      processed_input_vector=10*(input_vector>=400);
else
      processed_input_vector=zeros(1,vect_length);
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function [processed_input_vector]=preprol(input_vector)
%PREPROL    Input representation function for light seeking module
%
%     [PROCESSED_INPUT_VECTOR]=preprol(INPUT_VECTOR)
%     PROCESSED_INPUT_VECTOR - The input vector reduced to a "10" in
%                         the minimum value position, zeros otherwise
%     INPUT_VECTOR - The original input vector in the range [0,512]
%
%
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

vect_length=length(input_vector);
if min(input_vector)<505
      % Make all entries containing the minimum 1, all others
      % 0.
      processed_input_vector=10*(input_vector==min(input_vector));
else
      processed_input_vector=zeros(1,vect_length);
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function [a1,a2,a1_previous] = elmsim1c(p,w1,b1,w2,b2,tscale)
%ELMSIM1C   Simulates an Elman recurrent network for one column vector.
%
%     [A1,A2,A1_PREVIOUS] = SIMELM1C(P,W1,B1,W2,B2,TSCALE)
%       P  - Input (column) vectors to network arranged in time.
%       W1 - Weight matrix for recurrent layer.
%       B1 - Bias (column) vector for recurrent layer.
%       W2 - Weight matrix for output layer.
%       B2 - Bias (column) vector for output layer.
%       TSCALE - The scaling factor for the tansig layer, default 1.
%
%     Returns:
%       A1 - Output (column) vectors of recurrent layer in time.
%       A2 - Output (column) vectors of output layer in time.
%
%
%
%
%-------------------------------------------------------------------------
--

global a1_previous tmp_a1

[r,q] = size(p);
s1 = length(b1);
s2 = length(b2);
a2 = zeros(s2,q);

%  If tmp_a1 is undefined, then initialize it to zero
if tmp_a1 == []
      tmp_a1 = zeros(s1,1);
      fprintf('Info:  Initializing the context layer\n');
end

%  If tscale is undefined, set it to the default value 1.
if tscale == []
      tscale = 1;
      fprintf('Info:  Tscale has default value 1');
end

if nargout < 3
  error('Simelm1c would prefer if you took all output args.  Have a nice
day... :)');

elseif nargout == 3
  a1_previous=tmp_a1;
  a1 = zeros(s1,q);
  %  The input to the tansig (second) layer will be made up of two column
layers.  One
  %  is the network input, the other is the previous value of the context
layer.
  a1(:,1) = tansig2(tscale,w1*[p; a1_previous],b1);
  a2(:,1) = logsig(w2*a1,b2);

  %  Diagnostic Output
  %fprintf('Values to the tansig layer:')
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
  %(w1*[p;a1_previous]+b1)'
  %fprintf('Values to the logsig layer:')
  %(w2*a1+b2)'

  %  Copy the context layer
  tmp_a1=a1;
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function x=binout(vect_in)
%BINOUT           Produce a stochastic binary output vector from a probability
vector.
%           This function accepts any length vector and stochastically
produces
%           a binary output using a uniform distribution of random numbers.
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

vect_length=length(vect_in);
stoch_vect=rand(1,vect_length);
for cnt=1:vect_length
      if stoch_vect(cnt) < vect_in(cnt)
            vect_in(cnt)=1;
      else
            vect_in(cnt)=0;
      end
end
x=vect_in;
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function x=xplore(vect_in,randscale)
%XPLORE           Produce a semi-stochastic binary output vector from a
%           confidence vector produced by the network.
%           This function accepts any length vector and produces
%           a binary output.
%
%           This function presents a normal distribution of random numbers %
      with a mean of 0.5 and variable variance such that the selection %
      of the random number can fall within some definable boundaries.
%
%           X=XPLORE(VECT_IN,RANDSCALE)
%           VECT_IN   - Confidence vector input from the elman network
%           RANDSCALE - Variance control
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--


mean_adjust=0.5;

%  Choose a single random number
y=(randn(1,1)/randscale)+mean_adjust;

x=(vect_in>=y);
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function actuator(motor_vector,port_id)
%ACTUATOR   Send commands to the Khepera motors based on the network output
%           This version sends commands to the real Khepera
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

Fwd_speed=3;
Rvs_speed=-3;
mot_delay=60000;

if motor_vector == [0 0]
      %  Reverse
      %  Put in a guard "if" to prevent backing into walls
      tmp=rps(port_id);
      if any(tmp(7:8))>1020
            % Dont turn on motor
            sms(port_id,0,0)
      else
            sms(port_id,-3,-3);
            %  Block 5:  Mandatory delay
            for q=1:mot_delay
            end
      end
elseif motor_vector == [1 1]
      %Forward
      %  Put in a guard "if" to prevent moving into walls
      %  To make effective, you must enter an else statement turning the
      %  motors off
      tmp=rps(port_id);
      if all(tmp(2:5))<1000
            sms(port_id,3,3);
            %  The delay of 5000 below has shown itself to prevent bad values
            %  due to line congestion
            for q=1:5000
            end
      else
            sms(port_id,0,0);
      end
elseif motor_vector == [0 1]
      %Left
      sms(port_id,-3,3);
      %  Block 5:  Mandatory delay
      for q=1:mot_delay
      end
elseif motor_vector == [1 0]
      %Right
      sms(port_id,3,-3);
      %  Block 5:  Mandatory delay
      for q=1:mot_delay
      end
end
function [w1,b1,w2,b2] = trnelm1p(w1,b1,w2,b2,p,t,a1_previous,a1,a2,lr)
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
%TRNELM1P   One training pass of an elman recurrent network.
%
%     [W1,B1,W2,B2] = TRNELM1P(W1,B1,W2,B2,P,T,A1_PREVIOUS,A1,A2,LR)
%       W1 - Weight matrix for first layer (from input & feedback).
%       B1 - Bias (column) vector for first layer.
%       W2 - Weight matrix for second layer (from first layer).
%       B2 - Bias (column) vector for second layer.
%       P  - Input (column) vectors arranged in time.
%       T  - Target (column) vector for final output.
%       A1_PREVIOUS - Output of the context layer for the most recent
simulation
%                  less one time iteration of elman net
%       A1 - Output of context layer for the most recent simulation of elman %
            net.
%       A2 - Output layer of elman network
%       LR - Learning rate for backprop
%
%     Returns:
%       W1,B1,W2,B2 - New weights and biases.
%
%     This function follows the example on page 13-47 of NN toolbox manual. %%
      However, since there is a recurrent layer, we must keep track of the
value %     of this layer for as long as the network is valid.  This layer is
not %%      updated in either this function or simelm1c.  The old and new
values of %%      the context layer must be exchanged manually at a higher
level.
%
%
%
-------------------------------------------------------------------------
-

if nargin < 10,error('Wrong number of arguments.'),end
if lr==1
      % A LEARNING RATE OF 1 IS USED TO IMPLY NO TRAINING (ZERO REWARD)
      % IF NO TRAINING, UPDATE WEIGHTS BY ZERO
      dw1 = w1*0; db1 = b1*0;
      dw2 = w2*0; db2 = b2*0;

      % CALCULATE NEW WEIGHTS

      w1 = w1 + dw1; b1 = b1 + db1;
      w2 = w2 + dw2; b2 = b2 + db2;

else

      % INITIALIZE MATRIX SIZES
      %========================

      dw1 = w1*0; db1 = b1*0;
      dw2 = w2*0; db2 = b2*0;


      % CALCULATE ERRORS
      e = t - a2;
      % DIAGNOSTIC OUTPUT
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      fprintf('target vector: %d %d\n',t(1),t(2));
      fprintf('output vector: %d %d\n',a2(1),a2(2));
      fprintf('error:         %d %d\n\n',e(1), e(2));


      % CALCULATE DERIVATIVES

      d2 = deltalog(a2,e);
      d1 = deltatan(a1,d2,w2);


      % CALCULATE WEIGHT CHANGES

      [dw1,db1] = learnbp([p;a1_previous],d1,lr);
      [dw2,db2] = learnbp(a1,d2,lr);


      % CALCULATE NEW WEIGHTS

      w1 = w1 + dw1; b1 = b1 + db1;
      w2 = w2 + dw2; b2 = b2 + db2;

end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

The next several pages provide the code for the `simavoid.m` and `simlite.m` functions which implement the obstacle avoidance and light seeking modules using the *simulated* Khepera. Changes from the original files `avoidobj.m` and `seeklite.m` are outlined in bold. Any file that requires interaction with the robot through the serial commands must be modified for use by the simulator. This is done by replacing all serial commands with the equivalent simulator command (i.e. with sim_ attached to the front). An example of this is shown in `sim_act.m` which must activate the simulated Khepera's motors. Note that the reinforcment programs for `simavoid.m` and `simlite.m` must also read the simulated Khepera's sensors so that their respective sensor read commands must also be modified. The code for the reinforcement programs was not duplicated here since it is equivalent to `rpobj.m` and `rplight.m` with the **sim_** indicator attached to the sensor reading command.

```
function simavoid(port_id,cycles_per_run,runs,pause_stat)
%SIMAVOID   Simple Recurrent Network Learning Agent
%
%     This function operates with the simulated Khepera
%
%     This function iteratively calls all of the functions required to train
      an Elman simple recurrent network to avoid obstacles.  The function
      receives number of cycles/run, number of runs, port_id, etc from within
      the   Roborunner GUI.  The weights and biases W1,W2,b1,b2, of the
      trained network are saved at the end of each run, so they are not
      required to be passed back to the GUI.
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
% V1.0
% 04/11/96
%
--------------------------------------------------------------------------------
--

%  INCLUDE ALL VARIABLES THAT NEED TO BE REMEMBERED BY THE FUNCTION IN THE
%  GLOBAL STATEMENT BELOW
%  ========================================================================

%%%
% LIST THE VARIABLES THAT NEED INITIALIZATION AFTER
% THE GLOBAL STATEMENT BELOW.
%%%

%--------------------------------------------------------------------------
global P s1 s2 W1 b1 W2 b2 cycle_count run_count a1_previous learning_rate
global mse1 mse2 rwd_series tscale randscale
global btmap1
%--------------------------------------------------------------------------

global cycle_count run_count
%  IF THE FUNCTION IS CALLED WITHOUT ARGUMENT, THEN INITIALIZE ALL OF THE
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
%  VARIABLES ABOVE
%  ================================================================================
if nargin<1

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

      %%
      % Initialize all your variables here
      %%
      %  Set up the initial input vectors to initialize the elman network
      %  The following vector chooses the max and min of each input of the
      %  network.8 are for proximity, 2 are for motor state.

      P=[0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10];

      %  The network will be PXS1XS2, using logsig output layer, where P is
the
      %  input vector size (8 light sensors and maybe 2 motor states depending
      %  on the current implementation).
      s1=8;
      s2=2;

      %  Weights and biases
      [W1,b1,W2,b2]=initelm2(P,s1,s2);

      %  Make the following calls to other functions without arguments.  Each
of
      %  these functions has variables that will be initialized if the
function
      %  is called with nargin==0.
      simrpobj;

      %  Initialize the scaling factor for the tansig layer in the network
      %  and the scaling factor for the random distribution.
      tscale=0.04;
      randscale=40;

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

      %  Loop variables.  Do not remove
      cycle_count=0;
      run_count=0;
      return;

else
  while run_count<(runs)
    while cycle_count<(cycles_per_run)
      %-------------------------------------------------------------------------

      %%%
      % PLACE YOUR CODE HERE
      %%%
      %  Block 1:  Get data
      % Below is the input from the proximity sensors.
      input_vector(1:8)=preproo(sim_rps(port_id));
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  Block 2:  Compute output of elman network.
      [a1,a2,a1_previous]=elmsim1c(input_vector',W1,b1,W2,b2,tscale);

      %  Block 3:  Stochastic Binary Output of the search vector
      motor_vector=xplore(a2',randscale);
      %motor_vector=binout(a2');

      %  Block 4:  Execute action
      sim_act(motor_vector,port_id);

      %  Update the Khepera's Imaginary position
      kheprom(btmap1);

      %  Block 6:  Compute the corrections to the network based on the reward
      %  function
      [target_vector,learning_rate,reward_value]=simrpobj(motor_vector,a2',por
t_id);

      %  Track some statistics
      rwd_series=[rwd_series reward_value];

      %  Block 7:  Now train the elman network for one pass using the target
      %  vector, learning rate, and input vector.
      [W1,b1,W2,b2]=trnelm1p(W1,b1,W2,b2,input_vector',...
                  target_vector',a1_previous,a1,a2,learning_rate);


      %-------------------------------------------------------------------------
      cycle_count=cycle_count+1;
    end

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

    %%%
    % END OF RUN.  EVERYTHING TO BE DONE BETWEEN RUNS SHOULD BE DONE HERE
    %%%
    sms(2,0,0);
    %-------------------------------------------------------------------------
    run_count=run_count+1;
    cycle_count=0;
    if pause_stat==1
      [save_file save_path]=uiputfile('*.mat','Save Matlab Variables As...');
      if save_file~=0
            filepath=[save_path save_file];
            save(filepath);
      else
            warndlg('Data not saved!');
      end
    end
    [W1,b1,W2,b2]=initelm2(P,s1,s2);
    mse=[];
    rwd_series=[];
  end

%-------------------------------------------------------------------------
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
%%%
% PLACE ANY HOUSEKEEPING COMMANDS HERE (IE SMS(2,0,0) TO STOP REAL KHEP)
%%%
sms(2,0,0)
%-------------------------------------------------------------------------
return;
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
function simlite(port_id,cycles_per_run,runs,pause_stat)
%SIMLITE    Simple Recurrent Network Learning Agent
%
%     This function works with the simulated Khepera
%
%     This function iteratively calls all of the functions required to train
      an Elman simple recurrent network to seek light.  The function receive
      number of cycles/run, number of runs, port_id, etc from within the
      Roborunner GUI.  The weights and biases W1,W2,b1,b2, of the trained
      network are saved at the end of each run, so they are not required to be
      passed back to the GUI.
%
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
% V1.0
% 04/11/96
%
-------------------------------------------------------------------------------
-

%  INCLUDE ALL VARIABLES THAT NEED TO BE REMEMBERED BY THE FUNCTION IN THE
%  GLOBAL STATEMENT BELOW
%  ========================================================================

%%%
% LIST THE VARIABLES THAT NEED INITIALIZATION AFTER
% THE GLOBAL STATEMENT BELOW.
%%%

%-------------------------------------------------------------------------
global P s1 s2 W1 b1 W2 b2 cycle_count run_count a1_previous learning_rate
global mse rwd_series tscale randscale
global btmap1
%-------------------------------------------------------------------------

global cycle_count run_count
%  IF THE FUNCTION IS CALLED WITHOUT ARGUMENT, THEN INITIALIZE ALL OF THE
%  VARIABLES ABOVE
%  ========================================================================
if nargin<1

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

      %%
      % Initialize all your variables here
      %%

      %  Set up the initial input vectors to initialize the elman network
      %  The following vector chooses the max and min of each input of the
      %  network. 8 are for light, 2 are for motor state.

      %  Init vector without motor state feedback
      P=[0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10; 0 10];
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  The network will be PXS1XS2, using logsig output layer, where P is
the
      %  input vector size (8 light sensors and maybe 2 motor states depending
      %  on the current implementation).
      s1=8;
      s2=2;

      %  Initialize weights and biases
      [W1,b1,W2,b2]=initelm2(P,s1,s2);

      %  Make the following calls to other functions without arguments.  Each
of
      %  these functions has variables that will be initialized if the
function
      %  is called with nargin==0.
      simlrwd;

      %  Initialize the scaling factor for the tansig layer in the network
      %  and the scaling factor for the random distribution.
      tscale=0.04;
      randscale=40;

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

      %  Loop variables.  Do not remove
      cycle_count=0;
      run_count=0;
      return;

else
  while run_count<(runs)
    while cycle_count<(cycles_per_run)
      %-----------------------------------------------------------------------

      %%%
      % PLACE YOUR CODE HERE
      %%%

      %  BLOCK 1:  Get data
      %  Below is the input from the light sensors
      input_vector(1:8)=preprol(sim_rls(port_id));

      %  BLOCK 2:  Compute output of elman network.
      [a1,a2,a1_previous]=elmsim1c(input_vector',W1,b1,W2,b2,tscale);

      %  BLOCK 3:  Confidence interpretation
      %motor_vector=binout(a2'); % too liberal (uniform rand function)
      %  motor_vector=round(a2');  % too strict - no exploration
      motor_vector=xplore(a2',randscale);

      %  BLOCK 4:  Execute action
      sim_act(motor_vector,port_id);

      %  Update the kheprom
      kheprom(btmap1);
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

```
      %  BLOCK 6:  Compute the corrections to the network based on the reward
      [target_vector,learning_rate,reward_value]=simlrwd(motor_vector,a2',port
_id);

      %  Track error statistics for later comparison
      rwd_series=[rwd_series reward_value];

      %  BLOCK 7:  Now train the elman network for one pass using the target
      %  vector,learning rate, and input vector.
      [W1,b1,W2,b2]=trnelm1p(W1,b1,W2,b2,input_vector',...
                  target_vector',a1_previous,a1,a2,learning_rate);

      %-----------------------------------------------------------------------
      cycle_count=cycle_count+1;
    end

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

    %%%
    % END OF RUN.  EVERYTHING TO BE DONE BETWEEN RUNS SHOULD BE DONE HERE
    %%%
    sms(2,0,0);
    %-------------------------------------------------------------------------
    run_count=run_count+1;
    cycle_count=0;
    if pause_stat==1
      [save_file save_path]=uiputfile('*.mat','Save Matlab Variables As...');
      if save_file~=0
            filepath=[save_path save_file];
            save(filepath);
      else
            warndlg('Data not saved!');
      end
    end
    [W1,b1,W2,b2]=initelm2(P,s1,s2);
    mse=[];
    rwd_series=[];
  end

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

%%%
% PLACE ANY HOUSEKEEPING COMMANDS HERE (IE SMS(2,0,0) TO STOP REAL KHEP)
%%%
sms(2,0,0)
%-------------------------------------------------------------------------
return;
end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image

Matlab Function Code for Obstacle Avoidance/Light seeking

```
function sim_act(motor_vector,port_id)
%SIM_ACT    Send commands to the simulated Khepera based on the network output
%
%
%     Copyright (c) 1996 Capt Rich Goyette
%       Royal Military College, Canada
%
%-------------------------------------------------------------------------
--

if motor_vector == [0 0]
      %  Reverse
      sim_sms(port_id,-3,-3);

elseif motor_vector == [1 1]
      %Forward
      sim_sms(port_id,3,3);


elseif motor_vector == [0 1]
      %Left
      sim_sms(port_id,-3,3);

elseif motor_vector == [1 0]
      %Right
      sim_sms(port_id,3,-3);

end
```


created using
BCL easyPDF
Printer Driver
Click here to purchase a license to remove this image