

helicopterRL Code Documentation

Pedro Mediano
Imperial College London, UK
pedro.martinez-mediano13@imperial.ac.uk

August 8, 2014

1 Introduction

This is the documentation for the **helicopterRL** project. More information in
<https://github.com/pmediano/helicopterRL>
<http://wp.doc.ic.ac.uk/sml/student-projects/>

2 Main function

2.1 helicopter_learn_loop.m

Summary: Script to learn a controller for the helicopter problem

High-level steps

1. Initialize variables and compile RL-Glue trainer
2. Generate random trajectories to speed up learning
3. Apply learning loop (model learning, policy learning, policy application)

Code

```
1 % 1. Initialization
2 clear all; close all;
3 settings_hc; % Load scenario-specific settings
4
5 use_reward_model = false;
6 use_previous_history = true;
7
8 % Modify the trainer source files and compile for the desired MDP
9 whichMDP = 0;
10 setenv('tMDP', num2str(whichMDP));
11 cd([trainerDir 'consoleTrainerJavaHelicopter']);
12 !sed "s/whichTrainingMDP = [0-9]/whichTrainingMDP = ${tMDP}/" <src/↵
    consoleTrainerBackup >src/consoleTrainer.java
13 !make
14
15 % Get a suitable name for the history file
16 cd([pilcoDir 'scenarios/helicopter']);
```

```

17 getHistoryFilename;
18
19
20 % 2. Collect data with random policy to start learning, or start from the
21 % latest history file
22 if ~use_previous_history
23     cd([trainerDir 'testTrainerJavaHelicopter']);
24     !sed "s/whichTrainingMDP = [0-9]/whichTrainingMDP = ${tMDP}/" <src/↵
        testTrainerBackup >src/testTrainer.java
25     !make
26     !bash run.bash &
27     cd([agentDir 'randomAgentMatlab']);
28     delete([ 'randomDataMDP' num2str(whichMDP) '.mat']);
29     theRandomAgent = random_agent_matlab(whichMDP, codecDir, agentDir);
30     runAgent(theRandomAgent);
31
32     load([agentDir 'randomAgentMatlab/randomDataMDP' num2str(whichMDP) '.mat']);
33
34     delta_n = min(n_init, size(random_data.x, 1));
35     x = random_data.x(1:delta_n,:);
36     y = random_data.y(1:delta_n,:);
37     r_x = random_data.r_x(1:delta_n,:);
38     r_y = random_data.r_y(1:delta_n,:);
39     H = 5;
40 else
41     load(historyFilename);
42     x = history{end}.x;
43     y = history{end}.y;
44     r_x = history{end}.r_x;
45     r_y = history{end}.r_y;
46     dynmodel = history{end}.dynmodel;
47     if use_reward_model
48         if ~isfield(history{end}, 'reward_dynmodel')
49             disp('Requested reward model, but no reward model in previous history. ↵
                Abort. ');
50             return
51         else
52             reward_dynmodel = history{end}.reward_dynmodel;
53         end
54     end
55     policy = history{end}.policy;
56     H = history{end}.H;
57     last_size = history{end}.steps;
58 end
59
60
61 % Start learning loop. Feel free to Ctrl-C at any time, history file is
62 % saved automatically and you can resume later using
63 % |use_previous_history = true;|
64 cd([pilcoDir 'scenarios/helicopter']);
65 j = 1;
66 while true
67
68     % 3. Train GP's and specify cost function
69
70     % 3.1. Train GP for the helicopter dynamics
71     trainHelicopterModel

```

```

72     if exist('history', 'var'); history{end+1}.dynmodel = dynmodel; else history{1}.←
        dynmodel = dynmodel; end
73
74     % 3.2. Train GP model to predict reward to use as cost function, or
75     % set parameters of user-defined cost function
76     if use_reward_model
77         trainRewardModel;
78         cost.fcn = @Rbased_loss_hc;
79         cost.rewardgpmode = reward_dynmodel;
80         history{end}.reward_dynmodel = reward_dynmodel;
81     else
82         cost.width = 1;
83         cost.fcn = @loss_hc;
84     end
85
86     % 4. Learn policy and save structure
87     H = H + 5;
88     learnPolicy;
89     policy.date = clock;
90
91     history{end}.policy = policy;
92     history{end}.H = H;
93
94     % 5. Run the simulator with the latest policy until trajectory is
95     % longer than lower limit. If it isn't aggregate more datasets.
96     last_size = 0;
97     min_last_size = H;
98     run_counter = 0;
99     while last_size < min_last_size && run_counter < 10
100         cd([trainerDir 'consoleTrainerJavaHelicopter']);
101         !bash run.bash &
102         cd([pilcoDir 'scenarios/helicopter']);
103         theAgent = helicopter_agent(policy, codecDir, pilcoDir);
104         runAgent(theAgent);
105
106         aux_newdata = load('GPHistory.mat');
107         newdata = [newdata; newdata.helicopter_agent_struct];
108         last_size = size(newdata,1) - 1;
109
110         run_counter = run_counter + 1;
111     end
112
113     % 6. Get new data from the last trajectory. If the last trajectory
114     % was long enough, select data
115     if last_size < 5990
116         delta_n = min(last_size, max_last_size) + 1;
117         x = [x; newdata(1:delta_n-1,1:16)];
118         y = [y; newdata(2:delta_n,1:12)];
119         r_x = [r_x; newdata(2:delta_n,1:12)];
120         r_y = [r_y; newdata(2:delta_n, 17)];
121     else
122         [dynIdx, rewardIdx] = selectData(history{end}, max_last_size);
123         x = [x; history{end}.trajectory(dynIdx, 1:16)];
124         y = [y; history{end}.trajectory(dynIdx+1, 1:12)];
125         r_x = [r_x; history{end}.trajectory(rewardIdx, 1:12)];
126         r_y = [r_y; history{end}.trajectory(rewardIdx, 17)];
127     end
128

```

```

129
130     history{end}.steps = last_size;
131     history{end}.x = x;
132     history{end}.y = y;
133     history{end}.r_x = r_x;
134     history{end}.r_y = r_y;
135     history{end}.trajectory = newdata;
136
137     save(historyFilename, 'history');
138
139     disp(['Length of the last trajectory is ' num2str(last_size)]);
140
141     % 7. Plot NLPD of model when following the trajectory
142     drawNLPDplots;
143
144     j = j + 1;
145
146 end

```

2.2 settings_hc.m

Summary: Script to set up the helicopter scenario

High-Level Steps

1. Define state and important indices
2. Set up scenario
3. Set up the plant structure
4. Set up the policy structure
5. Set up the cost structure
6. Set up the GP dynamics model structure
7. Parameters for policy optimization
8. Plotting verbosity
9. Some array initializations

Code

```

1  warning('on','all'); format short; format compact
2
3  % Include some paths
4  try
5      rd = '../..';
6      addpath([rd 'base'],[rd 'util'],[rd 'gp'],[rd 'control'],[rd 'loss']);
7  catch
8      end
9
10  basename = 'helicopter_'; % filename used for saving data
11  trainerDir = '/home/pmediano/Downloads/RL/helicopter/trainers/';
12  agentDir = '/home/pmediano/Downloads/RL/helicopter/agents/';
13  setenv('AGENTDIR', agentDir);
14  codecDir = '/home/pmediano/Downloads/RL/helicopter/src/';
15  pilcoDir = '/home/pmediano/Downloads/RL/pilcoV0.9/';

```

```

16 addpath([agentDir 'GPAgentMatlab']);
17
18 rand('seed',1); randn('seed',1); format short; format compact;
19
20 % 1. Define state and important indices
21
22 % 1a. State representation
23 % 1 u_err forward velocity
24 % 2 v_err sideways velocity
25 % 3 w_err downward velocity
26 % 4 x_err forward error
27 % 5 y_err sideways error
28 % 6 z_err downward error
29 % 7 p_err angular rate around forward axis
30 % 8 q_err angular rate around sideways (to the right) axis
31 % 9 r_err angular rate around vertical (downward) axis
32 % 10 qx_err quaternion entries, x,y,z,w q = [ sin(theta/2) * axis; cos(theta/2)],
33 % 11 qy_err where axis = axis of rotation; theta is amount of rotation around that axis
34 % 12 qz_err [recall: any rotation can be represented by a single rotation around some axis]
35
36 observationIdx = 1:12;
37 nVar = 12;
38
39 % Action representation
40 % 13 longitudinal cyclic pitch
41 % 14 latitudinal (left-right) cyclic pitch
42 % 15 main rotor collective pitch
43 % 16 tail rotor collective pitch
44
45 actionIdx = 13:16;
46 nU = 4;
47 maxU = [1, 1, 1, 1];
48
49 % 17 reward
50
51 rewardIdx = 17;
52
53 % 1b. Important indices
54
55 odei = observationIdx; % indices for the ode solver
56 augi = []; % indices for variables augmented to the ode variables
57 dyno = observationIdx; % indices for the output from the dynamics model and indices to loss
58 анги = []; % indices for variables treated as angles (using sin/cos representation)
59 dyni = observationIdx; % indices for inputs to the dynamics model
60 poli = observationIdx; % indices for variables that serve as inputs to the policy
61 difi = observationIdx; % indices for training targets that are differences rather than values
62
63 % 2. Set up the scenario
64 mu0 = zeros([nVar 1]); % initial state mean (column vector)
65 % S0 = diag(zeros([1 nVar])); % initial state covariance
66 S0 = 0.0001*eye(nVar);

```

```

67 muOSim(odei,:) = mu0; SOSim(odei,odei) = S0; % Specify initial state ←
    distribution -
68 muOSim = muOSim(dyno); SOSim = SOSim(dyno,dyno); % in this case, the origin.
69
70 n_init = 100; % no. of initial data points (computed with random ←
    policy)
71 max_last_size = 100; % max no. of data points added to the dataset in ←
    each iteration
72 N = 20; % max no. of controller optimizations
73
74 % 3. Set up the plant structure
75 %plant.ctrl = @zoh; % controler is zero-order-hold
76 plant.odei = odei; % indices to the variables for the ode solver
77 plant.augi = augi; % indices of augmented variables
78 plant.angi = angi;
79 plant.poli = poli;
80 plant.dyno = dyno;
81 plant.dyni = dyni;
82 plant.difi = difi;
83 plant.prop = @propagated; % handle to function that propagates state over time
84
85
86
87 % 4. Set up the policy structure
88
89 policy.fcn = @(policy,m,s)conCat(@conlin,@gSat,policy,m,s);% controller
90 % representation
91 policy.maxU = maxU; % max. amplitude of
92 % actions
93 [mm, ss, cc] = gTrig(mu0, S0, plant.angi); % represent angles
94 mm = [mu0; mm]; cc = S0*cc; ss = [S0 cc; cc' ss]; % in complex plane
95
96 % Uncomment the following lines if policy is @conlin
97 policy.p.w = 1e-2*randn(length(policy.maxU),length(poli)); % weight matrix
98 policy.p.b = zeros(length(policy.maxU),1); % bias
99
100
101 % Uncomment the following lines if policy is @congp
102 % nc = 100; % size of controller training set
103 % policy.p.inputs = gaussian(mm(poli), ss(poli,poli), nc)'; % init. location of
104 % % basis functions
105 % policy.p.targets = 0.1*randn(nc, length(policy.maxU)); % init. policy targets
106 % % (close to zero)
107 % policy.p.hyp = ... % initialize policy
108 % repmat(log([1 1 1 1 1 1 1 1 1 1 1 1 1 1 0.01]')), 1, 4); % hyper-parameters
109
110
111 % 5. Set up the cost structure
112 cost.fcn = @loss_hc; % cost function
113 cost.gamma = 1; % discount factor
114 cost.width = 1; % cost function width
115 cost.expl = 0; % exploration parameter (UCB)
116 cost.angle = plant.angi; % index of angle (for cost function)
117 cost.target = zeros(nVar, 1); % target state
118
119 % Alternatively, define a function to translate RL-Glue rewards to cost
120 reward2loss = @(r) -exp(r/5);
121

```

```

122
123 % 6. Set up the GP dynamics model structure
124 dynmodel.fcn = @gp1d;           % function for GP predictions
125 dynmodel.train = @train;        % function to train dynamics model
126 dynmodel.induce = zeros(5000,0,1); % shared inducing inputs (sparse GP)
127 trainOpt = [300 300];          % defines the max. number of line searches
128                                % when training the GP dynamics models
129                                % trainOpt(1): full GP,
130                                % trainOpt(2): sparse GP (FITC)
131
132 % 7. Parameters for policy optimization
133 opt.length = 50;                % max. number of line searches
134 opt.MFEPLS = 7;                % max. number of function evaluations
135                                % per line search
136 opt.verbosity = 3;             % verbosity: specifies how much
137                                % information is displayed during
138                                % policy learning. Options: 0-3
139
140 % 8. Plotting verbosity
141 plotting.verbosity = 3;         % 0: no plots
142                                % 1: some plots
143                                % 2: all plots
144
145 % 9. Some initializations
146 fantasy.mean = cell(1,N); fantasy.std = cell(1,N);
147 realCost = cell(1,N); M = cell(N,1); Sigma = cell(N,1);
148 newdata = [];

```

3 Agents

3.1 random_agent_matlab.m

Summary: Random policy agent used to collect initial data. Contains the functions needed to communicate with the RL-Glue Core.

Contents

1. helicopter_agent: set useful paths and function handles
2. helicopter_agent_init: initialize data structure
3. helicopter_agent_start: take the first step
4. helicopter_agent_step: take a step
5. helicopter_agent_message: communicate between trainer and agent
6. helicopter_agent_end: finish episode
7. helicopter_agent_cleanup: save data structure

Code

```

1 function theAgent=random_agent_matlab(MDP, codec_base, agent_base)
2 % Add paths and fill agent structure
3
4     addpath([codec_base 'agent'], [codec_base 'glue'], codec_base);

```

```

5
6     theAgent.agent_init=@helicopter_agent_init;
7     theAgent.agent_start=@helicopter_agent_start;
8     theAgent.agent_step=@helicopter_agent_step;
9     theAgent.agent_end=@helicopter_agent_end;
10    theAgent.agent_cleanup=@helicopter_agent_cleanup;
11    theAgent.agent_message=@helicopter_agent_message;
12
13    global whichMDP;
14    whichMDP = MDP;
15    global agentDir;
16    agentDir = agent_base;
17
18 end
19
20 function helicopter_agent_init(taskSpec)
21 % Initialize agent
22 % Note that helicopter_agent_struct is created and saved in |start| and
23 % |end|, different from what is done in |GPAgentMatlab|
24 end
25
26 function theAction=helicopter_agent_start(theObservation)
27 % Take the first step of the agent as the episode starts, and store data
28
29     global helicopter_agent_struct;
30     global timeStep;
31     helicopter_agent_struct = zeros(2,17);
32     timeStep = 1;
33
34     theAction = org.rlcommunity.rlglue.codec.types.Action();
35     theAction.doubleArray = rand(4,1)-1;
36
37     helicopter_agent_struct(timeStep, 1:12) = theObservation.doubleArray;
38     helicopter_agent_struct(timeStep, 13:16) = theAction.doubleArray;
39     helicopter_agent_struct(timeStep, 17) = 0;
40
41 end
42
43 function theAction=helicopter_agent_step(theReward, theObservation)
44 % Take a step and store data
45
46     global helicopter_agent_struct;
47     global timeStep;
48
49     theAction = org.rlcommunity.rlglue.codec.types.Action();
50     theAction.doubleArray = 2*rand(4,1)-1;
51
52     timeStep = timeStep + 1;
53     helicopter_agent_struct(timeStep, 1:12) = theObservation.doubleArray;
54     helicopter_agent_struct(timeStep, 13:16) = theAction.doubleArray;
55     helicopter_agent_struct(timeStep, 17) = theReward;
56
57 end
58
59 function helicopter_agent_end(theReward)
60 % An episode ends and save the data
61
62     global helicopter_agent_struct;

```



```

63     global timeStep;
64     global whichMDP;
65     global agentDir;
66
67     basename = [agentDir 'randomAgentMatlab/'];
68     filename = sprintf('randomDataMDP%i.mat', whichMDP);
69     fullname = strcat(basename, filename);
70     if exist(fullname, 'file')
71         load(fullname);
72         random_data.x = [random_data.x; helicopter_agent_struct(1:end-1,1:16)];
73         random_data.y = [random_data.y; helicopter_agent_struct(2:end,1:12)];
74         random_data.r_x = [random_data.r_x; helicopter_agent_struct(2:end,1:12)];
75         random_data.r_y = [random_data.r_y; helicopter_agent_struct(2:end,17)];
76         random_data.all = [random_data.all; helicopter_agent_struct];
77     else
78         random_data.x = helicopter_agent_struct(1:end-1,1:16);
79         random_data.y = helicopter_agent_struct(2:end,1:12);
80         random_data.r_x = helicopter_agent_struct(2:end,1:12);
81         random_data.r_y = helicopter_agent_struct(2:end, 17);
82         random_data.all = helicopter_agent_struct;
83     end
84     if timeStep > 1
85         save(fullname, 'random_data');
86     end
87 end
88
89 function returnMessage=helicopter_agent_message(theMessageJavaObject)
90 % Custom function for trainer-agent communication
91
92     inMessage=char(theMessageJavaObject);
93     global whichMDP;
94     if strcmp(inMessage, 'What is your name?')==1
95         returnMessage='My name is random_agent_matlab';
96     elseif strcmp(inMessage, 'Which MDP?')==1
97         returnMessage=sprintf('I am doing MDP = %i ', whichMDP);
98     else
99         returnMessage='I don\'t know how to respond to your message';
100     end
101 end
102
103 function helicopter_agent_cleanup()
104 % Agent cleanup
105
106 end

```

3.2 helicopter_agent.m

Summary: Helicopter control agent. Contains the functions needed to communicate with the RL-Glue Core.

Contents

1. helicopter_agent: set useful paths and function handles
2. helicopter_agent_init: initialize data structure
3. helicopter_agent_start: take the first step

4. helicopter_agent_step: take a step
5. helicopter_agent_message: communicate between trainer and agent
6. helicopter_agent_end: finish episode
7. helicopter_agent_cleanup: save data structure

Code

```

1  function theAgent=helicopter_agent(policy_input, codec_base, pilco_root)
2  % Add paths and fill agent structure
3
4      global policy
5      policy = policy_input;
6
7      addpath([codec_base 'agent'], [codec_base 'glue'], codec_base);
8
9      addpath([pilco_root 'base'],[pilco_root 'util'],[pilco_root 'gp'],[pilco_root '←
        control'],[pilco_root 'loss']);
10
11     theAgent.agent_init=@helicopter_agent_init;
12     theAgent.agent_start=@helicopter_agent_start;
13     theAgent.agent_step=@helicopter_agent_step;
14     theAgent.agent_end=@helicopter_agent_end;
15     theAgent.agent_cleanup=@helicopter_agent_cleanup;
16     theAgent.agent_message=@helicopter_agent_message;
17
18 end
19
20 function helicopter_agent_init(taskSpec)
21 % This is a persistent struct we will use to store the data collected for
22 % the next learning iteration
23 global helicopter_agent_struct;
24 helicopter_agent_struct = zeros(2,17);
25
26 end
27
28 function theAction=helicopter_agent_start(theObservation)
29 % Take the first step of the agent as the episode starts, and store data
30
31     global helicopter_agent_struct;
32     global timeStep;
33     global policy
34     timeStep = 1;
35
36     theAction = org.rlcommunity.rlg glue.codec.types.Action();
37     theAction.doubleArray = policy.fcn(policy, theObservation.doubleArray, zeros(length←
        (theObservation.doubleArray)));
38
39     helicopter_agent_struct(timeStep, 1:12) = theObservation.doubleArray;
40     helicopter_agent_struct(timeStep, 13:16) = theAction.doubleArray;
41     helicopter_agent_struct(timeStep, 17) = 0;
42
43 end
44
45 function theAction=helicopter_agent_step(theReward, theObservation)
46 % Take a step and store data
47

```

```

48     global helicopter_agent_struct;
49     global timeStep;
50     global policy;
51
52     theAction = org.rlcommunity.rlglue.codec.types.Action();
53     theAction.doubleArray = policy.fcn(policy, theObservation.doubleArray, zeros(←
        length(theObservation.doubleArray)));
54
55     timeStep = timeStep + 1;
56     helicopter_agent_struct(timeStep, 1:12) = theObservation.doubleArray;
57     helicopter_agent_struct(timeStep, 13:16) = theAction.doubleArray;
58     helicopter_agent_struct(timeStep, 17) = theReward;
59
60 end
61
62 function helicopter_agent_end(theReward)
63     % An episode ends
64 end
65
66 function returnMessage=helicopter_agent_message(theMessageJavaObject)
67 % Custom function for trainer-agent communication
68
69     inMessage=char(theMessageJavaObject);
70     global policy;
71
72     if strcmp(inMessage, 'what is your name?')==1
73         returnMessage='my name is helicopter_agent , Matlab edition!';
74     elseif strcmp(inMessage, 'when')==1
75         % Print policy training timestamp
76         returnMessage = num2str(policy.date);
77     else
78         returnMessage='I don\'t know how to respond to your message';
79     end
80
81 end
82
83 function helicopter_agent_cleanup()
84 % On cleanup, save the collected data to a MAT-file
85
86     global helicopter_agent_struct;
87     save('GPHistory', 'helicopter_agent_struct');
88 end

```

4 Model evaluation functions

4.1 checkModel.m

Summary: Compute goodness-of-fit measures to assess the predictive power of the model. If no test dataset is provided, measure on train dataset.

Input arguments:

model	model to be assessed
test_x	(optional) test inputs to try model on
test_y	(optional) test outputs to try model on

Code

```

1 function checkModel( model, test_x, test_y )
2
3     if nargin == 1; x = model.inputs; y = model.targets;
4     else x = test_x; y = test_y; end;
5
6     % 1. Gather information about the model and the test dataset
7     [m, n] = size(x);
8     [m1, d] = size(y);
9     if m1~=m; return; end;
10    LP = zeros(m, 1);
11    sampleSize = 1000;
12    sampleLP = zeros(m, 1);
13    predictions = zeros(m, d);
14
15    % 2. Compute model and optimal NLPD
16    for j=1:m
17        [mu, sigma] = model.fcn(model, x(j,:) ', zeros(n));
18        %disp(mu);
19        LP(j) = NLPD(y(j,:) ', mu, sigma, d);
20        sample = mvnrnd(mu, sigma, sampleSize);
21        aux = cellfun(@(v) NLPD(v', mu, sigma, d), num2cell(sample, 2));
22        sampleLP(j) = mean(aux);
23        predictions(j, :) = mu;
24    end
25    fprintf('NLPD of model\n \t True \t Optimal \n Mean %f %f \n Std %f %f\n',mean←
        (LP), mean(sampleLP), std(LP), std(sampleLP));
26
27    % 3. Compute relative error
28    diff = abs((predictions-y)./y);
29    relDiff = mean(100*diff)';
30    fprintf('Mean relative difference of each variable (%%):\n');
31    disp(relDiff);
32    fprintf('Total average relative difference is %f%%\n', mean(relDiff));
33
34
35    % 4. Check SNR's
36    hyp = model.hyp;
37    SNR = exp(hyp(end-1,:)-hyp(end,:));
38    if any(SNR > 500)
39        fprintf('SNR is greater than 500 for variables %s\n', num2str(find(SNR>500)))←
        ;
40    else
41        fprintf('All SNRs are less than 500.\n');
42    end
43
44    % 5. Check log-signal-std
45    lsstd = hyp(end-1, :);
46    if any(lsstd<log(0.1))
47        low = lsstd<log(0.1);
48        fprintf('LSSTD is lower than log(0.1) for\n variables \t%s\n values \t%s\n', ←
        num2str(find(low)), num2str(lsstd(low)));
49    end
50    if any(lsstd>log(10))
51        high = lsstd>log(10);

```

```

52         fprintf('LSSTD is higher than log(10) for\n variables \t%s\n values \t%s\n', ←
               num2str(find(high)), num2str(lsstd(high)));
53     end
54     if all(and(lsstd>log(0.1), lsstd<log(10)))
55         fprintf('All log-signal-std are within safe range.\n');
56     end
57
58     % 6. Check lengthscales
59     lengthScale = exp(hyp(1:end-2,:));
60     inputStd = repmat(std(model.inputs)',[1,d]);
61     fprintf('LengthScale/std ratios:\n');
62     disp(lengthScale./inputStd);
63
64
65 end

```

4.2 drawNLPDplots.m

Summary: Script to assess predictive power of a model. Computes NLPD and RelDiff of model predictions along the trajectory generated by the policy trained on it.

High-level steps

1. Initialize empty matrices
2. Loop over the trajectory gathering statistics
3. Draw plots

Code

```

1  try
2      LP = zeros(last_size, 1);
3      sampleLP = zeros(last_size, 1);
4      sampleLPstd = zeros(last_size, 1);
5      relDiff = zeros(last_size, 1);
6      sampleSize = 500;
7      fprintf(1, '\nCalculating NLPD and relative differences of predictions and ←
               experiment:\nStep:      ');
8      for k=1:last_size
9          [mu, sigma] = dynmodel.fcn(dynmodel, newdata(k,1:16)', 0.0001*eye(16));
10         mu(difi) = mu(difi) + newdata(k, difi)';
11         LP(k) = NLPD(newdata(k+1,1:12)', mu, sigma, 12);
12         sample = mvnrnd(mu, sigma, sampleSize);
13         aux = cellfun(@(v) NLPD(v', mu, sigma, 12), num2cell(sample, 2));
14         sampleLP(k) = mean(aux);
15         sampleLPstd(k) = std(aux);
16         relDiff(k) = 100*mean((mu - newdata(k+1,1:12)') ./ newdata(k+1,1:12)');
17         fprintf(1, '\b\b\b\b\b%4i ', k);
18     end
19     aux = 1:last_size;
20     figure(10)
21     plot(aux, LP, 'k-', ...
22          aux, sampleLP, 'g-', ...
23          aux, sampleLP + sampleLPstd, 'r:', ...

```

```

24     aux, sampleLP - sampleLPstd, 'r:' ); drawnow;
25     xlabel('Steps');
26     ylabel('NLPD');
27     legend('Real', 'Optimal', '1-sigma belt');
28
29     figure(11)
30     plot(relDiff); drawnow;
31     xlabel('Steps');
32     ylabel('Relative Difference (%)');
33
34 catch ME
35     disp('Error computing NLPD');
36     disp('Exception: ');
37     disp(ME);
38     disp('Covariance matrix: ');
39     disp(sigma);
40 end

```

4.3 evaluateHistory.m

Summary: Evaluate the performance of the policies in a certain learning history.

Input arguments: whichMDP: number of the MDP to test policies on history: cell array, each cell contains a policy to be evaluated

Output arguments: res: matrix with the average and std of steps survived by each policy

High-level steps

1. Set up configuration for helicopter scenario (paths, variables, etc)
2. Compile java trainer to run desired MDP
3. Evaluate each policy in history file \verb@sample@ times and return statistics

Code

```

1 function res=evaluateHistory(whichMDP, history)
2
3     % 1. Set up helicopter configuration
4     settings_hc;
5
6     n_iter = numel(history);
7     res = zeros([n_iter, 5]);
8
9     % 2. Compile java trainer
10    setenv('tMDP', num2str(whichMDP));
11    cd([trainerDir 'consoleTrainerJavaHelicopter']);
12    !sed "s/whichTrainingMDP = [0-9]/whichTrainingMDP = ${tMDP}/" <src/↵
        consoleTrainerBackup >src/consoleTrainer.java
13    !make
14
15    % 3. Loop over history and evaluate policies
16    for j=1:n_iter

```

```

17     theAgent = helicopter_agent(history{j}.policy, codecDir, pilcoDir);
18     res(j, 1) = size(history{j}.dynmodel.targets, 1);
19     [res(j, 2), res(j, 3), res(j, 4), res(j, 5)] = evaluatePolicy(theAgent, ↵
        trainerDir, pilcoDir);
20 end
21
22 end
23
24 function [mean_time, std_time, mean_reward, std_reward]=evaluatePolicy(theAgent, ↵
        trainerDir, pilcoDir, nb_samples)
25 % Auxiliari function for evaluateHistory.
26
27     if nargin < 4
28         sample = 10;    % Number of trajectories to generate
29     else
30         sample = nb_samples;
31     end
32     steps = zeros([sample, 1]);
33     rewards = zeros([sample, 1]);
34
35     for j=1:sample
36         cd([trainerDir 'consoleTrainerJavaHelicopter']);
37         !bash run.bash &
38         cd([pilcoDir 'scenarios/helicopter']);
39         runAgent(theAgent);
40         newdata = load('GPHistory.mat');
41         steps(j) = size(newdata.helicopter_agent_struct, 1) - 1;
42         rewards(j) = mean(newdata.helicopter_agent_struct(:, 17));
43     end
44
45     mean_time = mean(steps);
46     std_time = std(steps);
47     mean_reward = mean(rewards);
48     std_reward = std(rewards);
49
50 end

```

4.4 NLPD.m

Summary: Computes the Negative Log Predictive Density for a multivariate gaussian distribution.

Input arguments: x observation mu mean vector of the distribution sigma covariance matrix of the distribution d (optional) dimensionality of input space

Output arguments: lp computed NLPD

Code

```

1 function [ lp ] = NLPD(x, mu, sigma, d)
2
3     if nargin==3; d = numel(x); end;
4

```

```

5     diff = x-mu;
6     lp = 0.5*(log(det(sigma)) + diff'*(sigma\diff) + d*1.837877066409345);
7
8 end

```

5 Helper functions

5.1 getHistoryFilename.m

Summary: Script to find a name for the history file to be used. If user set `use_previous_history=true`, the name of the latest existing history file is returned. If `use_previous_history=false`, return a name for a new file.

Code

```

1 historyFilename = sprintf('historyMDP%i.mat', whichMDP);
2
3 if ~use_previous_history && ~exist(historyFilename, 'file')
4     disp(['Creating first history file for this MDP, ' historyFilename]);
5
6 elseif use_previous_history && ~exist(historyFilename, 'file')           % If the user↵
7     requests a previous history but                                       % there are no ↵
7                                     saved ↵
7                                     histories, ↵
7                                     notify
8     disp(['No previous history to load. Generating random data. Saving history in ' ↵
8         historyFilename]);
9     use_previous_history = false;
10 else                             % If a history ↵
11     file exists, add a subindex
12     history_nb = 2;
13     historyFilename = sprintf('historyMDP%i.%i.mat', whichMDP, history_nb);
14     if use_previous_history && ~exist(historyFilename, 'file')
15         historyFilename = sprintf('historyMDP%i.mat', whichMDP);
16         disp(['Using data from file ' historyFilename]);
17     else
18         while exist(historyFilename, 'file')                             % Find the latest↵
19             history file for this MDP
20             history_nb = history_nb + 1;
21             historyFilename = sprintf('historyMDP%i.%i.mat', whichMDP, history_nb);
22         end
23         if use_previous_history                                           % If user ↵
24             requested previous history, go back                         % one step to get↵
25                                     the latest ↵
26                                     existing file
27
28         history_nb = history_nb - 1;
29         historyFilename = sprintf('historyMDP%i.%i.mat', whichMDP, history_nb);
30         disp(['Using history from the latest file, ' historyFilename]);
31     else
32         disp(['Creating new file ' historyFilename]);
33     end

```



```

29     end
30 end

```

5.2 loss_dp.m

Summary: Double-Pendulum loss function; the loss is $1 - \exp(-0.5 * d^2 * a)$, where $a > 0$ and d^2 is the squared difference between the actual and desired position of the tip of the outer pendulum. The mean and the variance of the loss are computed by averaging over the Gaussian distribution of the state $p(x) = \mathcal{N}(m, s)$ with mean m and covariance matrix s . Derivatives of these quantities are computed when desired.

```
function [L, dLdm, dLds, S2] = loss_dp(cost, m, s)
```

Input arguments:

cost	cost structure	
.width	array of widths of the cost (summed together)	
.expl	(optional) exploration parameter	
.angle	(optional) array of angle indices	
.target	target state	[D x 1]
m	mean of state distribution	[D x 1]
s	covariance matrix for the state distribution	[D x D]

Output arguments:

L	expected cost	[1 x 1]
dLdm	derivative of expected cost wrt. state mean vector	[1 x D]
dLds	derivative of expected cost wrt. state covariance matrix	[1 x D ²]
S2	variance of cost	[1 x 1]

High-Level Steps

1. Precomputations
2. Define static penalty as distance from target setpoint
3. Calculate loss

```

1 function [L, dLdm, dLds, S2] = loss_hc(cost, m, s)

```

Code

```

1 if isfield(cost, 'width'); cw = cost.width; else cw = 1; end
2 if ~isfield(cost, 'expl') || isempty(cost.expl); b = 0; else b = cost.expl; end
3
4
5 % 1. Some precomputations

```

```

6 D0 = size(s,2); D = D0; % state dimension
7 D1 = D0 + 2*length(cost.angle); % state dimension (with sin/cos)
8
9 M = zeros(D1,1); M(1:D0) = m; S = zeros(D1); S(1:D0,1:D0) = s;
10 Mdm = [eye(D0); zeros(D1-D0,D0)]; Sdm = zeros(D1*D1,D0);
11 Mds = zeros(D1,D0*D0); Sds = kron(Mdm,Mdm);
12
13 % 2. Define static penalty as distance from target setpoint
14 Q = diag(ones(numel(m), 1));
15
16 target = [cost.target(:); gTrig(cost.target(:), 0*s, cost.angle)];
17 % 3. Calculate loss
18 L = 0; dLdm = zeros(1,D0); dLds = zeros(1,D0*D0); S2 = 0;
19 for i = 1:length(cw) % scale mixture of immediate costs
20     cost.z = target; cost.W = Q/cw(i)^2;
21     [r, rdM, rdS, s2, s2dM, s2dS] = lossSat(cost, M, S);
22
23     L = L + r; S2 = S2 + s2;
24     dLdm = dLdm + rdM(:)'*Mdm + rdS(:)'*Sdm;
25     dLds = dLds + rdM(:)'*Mds + rdS(:)'*Sds;
26
27     if (b~=0 || ~isempty(b)) && abs(s2)>1e-12
28         L = L + b*sqrt(s2);
29         dLdm = dLdm + b/sqrt(s2) * ( s2dM(:)'*Mdm + s2dS(:)'*Sdm )/2;
30         dLds = dLds + b/sqrt(s2) * ( s2dM(:)'*Mds + s2dS(:)'*Sds )/2;
31     end
32 end
33
34 % normalize
35 n = length(cw); L = L/n; dLdm = dLdm/n; dLds = dLds/n; S2 = S2/n;

```

Rbased_loss_hc.m

```
function [L, dLdm, dLds, S2] = Rbased\_loss\_hc(cost, m, s)
```

Input arguments:

cost	cost structure	
.rewardgpmode	GP to predict reward given state	
m	mean of state distribution	[D x 1]
s	covariance matrix for the state distribution	[D x D]

Output arguments:

L	expected cost	[1 x 1]
dLdm	derivative of expected cost wrt. state mean vector	[1 x D]
dLds	derivative of expected cost wrt. state covariance matrix	[1 x D^2]
S2	variance of cost	[1 x 1]

```
1 function [L, dLdm, dLds, S2] = Rbased_loss_hc(cost, m, s)
```

Code

```
1 [L, S2, ~, dLdm, ~, ~, dLds, ~, ~] = gp1d(cost.rewardgpmodel, m, s);

1 end
```

5.3 trainHelicopterModel.m

Summary: Script to learn the dynamics model. If SNR is too high, white noise is added to the GP targets and model is trained again.

High-level-steps

1. Set up GP structure
2. Train one GP for each variable separately and merge results into the complete GP
3. Check length-scales

Code

```
1 % 1. Set up GP inputs and targets
2 Du = length(policy.maxU); Da = length(plant.angi); % no. of ctrl and angles
3 xaug = [x(:,dyno) x(:,end-Du-2*Da+1:end-Du)]; % x augmented with angles
4 dynmodel.inputs = [xaug(:,dyni) x(:,end-Du+1:end)]; % use dyni and ctrl
5 dynmodel.targets = y(:,dyno);
6 dynmodel.targets(:,difi) = dynmodel.targets(:,difi) - x(:,dyno(difi));
7
8 dynmodel.hyp = zeros([nVar+nU+2, nVar]);
9
10 % 2. For each variable, train a separate GP until SNR falls below the limit
11 for k=1:nVar
12     disp(['Training GP for variable ' num2str(k)]);
13     auxDynmodel.fcn = dynmodel.fcn;
14     auxDynmodel.train = dynmodel.train;
15     auxDynmodel.induce = dynmodel.induce;
16     auxDynmodel.inputs = dynmodel.inputs;
17     auxDynmodel.targets = dynmodel.targets(:,observationIdx(k));
18     auxDynmodel.hyp = dynmodel.hyp(:,k);
19
20     auxDynmodel = auxDynmodel.train(auxDynmodel, plant, trainOpt);
21
22     Xh = auxDynmodel.hyp;
23     SNR = exp(Xh(end-1)-Xh(end));
24     disp(['SNR = ' num2str(SNR)]);
25     while SNR>500
26         disp('SNR too high. Add noise and re-train. ');
27         std_target = std(auxDynmodel.targets);
28         auxDynmodel.targets = auxDynmodel.targets + 0.01*std_target*randn(size(auxDynmodel.targets));
29         auxDynmodel = auxDynmodel.train(auxDynmodel, plant, trainOpt);
```

```

30     Xh = auxDynmodel.hyp;
31     SNR = exp(Xh(end-1)-Xh(end));
32     disp(['Now SNR is ' num2str(SNR)]);
33 end
34
35 % 2.1. Merge the trained hyper-parameters with the complete GP
36 dynmodel.hyp(:,k) = auxDynmodel.hyp;
37
38 end
39
40 % 3. Show the result of various tests based on the model's length-scales
41 Xh = dynmodel.hyp;
42 log_signal_std = (Xh(end-1,:) < log(0.1)) + (Xh(end-1,:) > log(10));
43 if any(log_signal_std)
44     fprintf('Helicopter model log-signal-std hyperparameter is out of range for ↵
45         variables %s \n', num2str(find(log_signal_std)));
46 end
47
48 input_std = std(dynmodel.inputs);
49 if any(Xh(1:16,:) > 100*repmat(input_std',[1, nVar]))
50     fprintf('Length-scales are much bigger than input std\n');
51 end
52 if any(Xh(1:16,:) < 0.01*repmat(input_std',[1, nVar]))
53     fprintf('Length-scales are much smaller than input std\n');
54 end

```

5.4 trainRewardModel.m

Summary: Script to train the reward model.

High-level steps

1. Set up GP structure and train
2. Add noise and re-train while SNR is high
3. Check length-scales of the model

Code

```

1 % 1. Set up and train reward GP structure
2 clear reward_dynmodel;
3 reward_dynmodel.fcn = @gp1d;
4 reward_dynmodel.train = @train;
5 reward_dynmodel.induce = zeros(5000,0,1);
6 reward_dynmodel.inputs = r_x;
7 reward_dynmodel.targets = reward2loss(r_y);
8
9 reward_dynmodel = reward_dynmodel.train(reward_dynmodel); % train dynamics GP
10
11 % 2. Add noise and train again while SNR is high
12 r_hyp = reward_dynmodel.hyp;
13 highSNR = any(exp(r_hyp(end-1,:)-r_hyp(end,:)) > 500);
14

```

```

15 while highSNR
16     target_std = std(reward_dynmodel.targets);
17     reward_dynmodel.targets = reward_dynmodel.targets + 0.01*target_std*randn(size(reward_dynmodel.targets));
18     reward_dynmodel = reward_dynmodel.train(reward_dynmodel); % train dynamics GP
19     r_hyp = reward_dynmodel.hyp;
20     highSNR = exp(r_hyp(end-1)-r_hyp(end)) > 500;
21     disp(['Reward model SNR = ' num2str(exp(r_hyp(end-1)-r_hyp(end)))]);
22 end
23
24 % 3. Check length-scales
25 if r_hyp(end-1) > log(10)
26     fprintf('Reward log-signal-std hyperparameter is %.4f, greater than log(10)\n', ←
27         r_hyp(end-1));
28 end
29 if r_hyp(end-1) < log(0.1)
30     fprintf('Reward log-signal-std hyperparameter is %.4f, less than log(0.1)\n', ←
31         r_hyp(end-1));
32 end
33 input_std = std(reward_dynmodel.inputs)';
34 fprintf('Reward input lengthscales and input std:\n');
35 disp(num2str([exp(r_hyp(1:12)), input_std]));

```