settings_hc.m

Table of Contents

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

```
warning('on','all'); format short; format compact

% Include some paths
try
  rd = '../../';
  addpath([rd 'base'],[rd 'util'],[rd 'gp'],[rd 'control'],[rd 'loss']);
catch
end

basename = 'helicopter_';      % filename used for saving data
trainerDir = '/home/pmediano/Downloads/RL/helicopter/trainers/';
agentDir = '/home/pmediano/Downloads/RL/helicopter/agents/';
setenv('AGENTDIR', agentDir);
codecDir = '/home/pmediano/Downloads/RL/helicopter/src/';
pilcoDir = '/home/pmediano/Downloads/RL/pilcoV0.9/';
addpath([agentDir 'GPAgentMatlab']);

rand('seed',1); randn('seed',1); format short; format compact;

% 1. Define state and important indices
```

```
% 1a. State representation
%  1  u_err   forward velocity
%  2  v_err   sideways velocity
%  3  w_err   downward velocity
%  4  x_err   forward error
%  5  y_err   sideways error
%  6  z_err   downward error
%  7  p_err   angular rate around forward axis
%  8  q_err   angular rate around sideways (to the right) axis
%  9  r_err   angular rate around vertical (downward) axis
%  10  qx_err   quaternion entries, x,y,z,w   q = [ sin(theta/2) * axis; cos(theta
%  11 qy_err    where axis = axis of rotation; theta is amount of rotation around
%  12 qz_err    [recall: any rotation can be represented by a single rotation arou

observationIdx = 1:12;
nVar = 12;

%  Action representation
%  13 longitudinal cyclic pitch
%  14 latitudinal (left-right) cyclic pitch
%  15 main rotor collective pitch
%  16 tail rotor collective pitch

actionIdx = 13:16;
nU = 4;
maxU = [1, 1, 1, 1];

%  17 reward

rewardIdx = 17;

% 1b. Important indices

odei = observationIdx;        % indicies for the ode solver
augi = [];                    % indicies for variables augmented to the ode variab
dyno = observationIdx;        % indicies for the output from the dynamics model an
angi = [];                    % indicies for variables treated as angles (using si
dyni = observationIdx;        % indicies for inputs to the dynamics model
poli = observationIdx;        % indicies for variables that serve as inputs to the
difi = observationIdx;        % indicies for training targets that are differences

% 2. Set up the scenario
mu0 = zeros([nVar 1]);               % initial state mean (column vector)
% S0 = diag(zeros([1 nVar]));          % initial state covariance
S0 = 0.0001*eye(nVar);
mu0Sim(odei,:) = mu0; S0Sim(odei,odei) = S0;        % Specify initial state distri
mu0Sim = mu0Sim(dyno); S0Sim = S0Sim(dyno,dyno);    % in this case, the origin.

n_init = 100;                       % no. of initial data points (computed with rand
max_last_size = 100;                % max no. of data points added to the dataset in
N = 20;                             % max no. of controller optimizations

% 3. Set up the plant structure
```

```
%plant.ctrl = @zoh;                  % controler is zero-order-hold
plant.odei = odei;                   % indices to the varibles for the ode solver
plant.augi = augi;                   % indices of augmented variables
plant.angi = angi;
plant.poli = poli;
plant.dyno = dyno;
plant.dyni = dyni;
plant.difi = difi;
plant.prop = @propagated;   % handle to function that propagates state over time



% 4. Set up the policy structure

policy.fcn = @(policy,m,s)conCat(@conlin,@gSat,policy,m,s);% controller
                                                          % representation
policy.maxU = maxU;                                  % max. amplitude of
                                                          % actions
[mm, ss, cc] = gTrig(mu0, S0, plant.angi);                  % represent angles
mm = [mu0; mm]; cc = S0*cc; ss = [S0 cc; cc' ss];         % in complex plane

% Uncomment the following lines if policy is @conlin
policy.p.w = 1e-2*randn(length(policy.maxU),length(poli));  % weight matrix
policy.p.b = zeros(length(policy.maxU),1);                  % bias


% Uncomment the following lines if policy is @congp
% nc = 100;                             % size of controller training set
% policy.p.inputs = gaussian(mm(poli), ss(poli,poli), nc)'; % init. location of
%                                                            % basis functions
% policy.p.targets = 0.1*randn(nc, length(policy.maxU));    % init. policy targets
%                                                            % (close to zero)
% policy.p.hyp = ...                                         % initialize policy
%  repmat(log([1 1 1 1 1 1 1 1 1 1 1 1 1 0.01]'), 1, 4);   % hyper-parameters


% 5. Set up the cost structure
cost.fcn = @loss_hc;                            % cost function
cost.gamma = 1;                                 % discount factor
cost.width = 1;                               % cost function width
cost.expl = 0;                                  % exploration parameter (UCB)
cost.angle = plant.angi;                        % index of angle (for cost function)
cost.target = zeros(nVar, 1);                       % target state

% Alternatively, define a function to translate RL-Glue rewards to cost
reward2loss = @(r) -exp(r/5);


% 6. Set up the GP dynamics model structure
dynmodel.fcn = @gp1d;                % function for GP predictions
dynmodel.train = @train;             % function to train dynamics model
dynmodel.induce = zeros(5000,0,1);    % shared inducing inputs (sparse GP)
trainOpt = [300 300];                % defines the max. number of line searches
                                     % when training the GP dynamics models
```

```
                                     % trainOpt(1): full GP,
                                     % trainOpt(2): sparse GP (FITC)

% 7. Parameters for policy optimization
opt.length = 50;                        % max. number of line searches
opt.MFEPLS = 7;                         % max. number of function evaluations
                                         % per line search
opt.verbosity = 3;                       % verbosity: specifies how much
                                         % information is displayed during
                                         % policy learning. Options: 0-3

% 8. Plotting verbosity
plotting.verbosity = 3;            % 0: no plots
                                   % 1: some plots
                                   % 2: all plots

% 9. Some initializations
fantasy.mean = cell(1,N); fantasy.std = cell(1,N);
realCost = cell(1,N); M = cell(N,1); Sigma = cell(N,1);
newdata = [];
```

Published with MATLAB® R2014a