**PI2 Library Implementation Documentation**

Date: 31 August, 2014

**Setup**

1) The C++ Eigen Library needs to be in the include directory. Needed libraries are:

**#include** <Eigen/Dense>

**#include** <Eigen/Sparse>

**#include** <Eigen/Eigen>

using namespace Eigen

2) PI2Wholesteps.cpp and the corresponding header file is the main PI2 ( with update and rollout methods) library that needs to be included for learning with PI2.

**#include** <PI2Wholesteps.h>

3) nejihebipi2controller.cpp is the starting file for calling PI2 library within PI2Wholesteps.cpp in the controller implementation.

4) nejihebipi2controller.cpp has the init( ) method that takes the ***initial state vector*** (start positions and joint angles) ***, time parameters, and goal as input***. This method calls the PI2Wholesteps.cpp

For giving the goal (x,y) as input in init( ) / from terminal, the following direction is referred:



5) The ***time parameters*** are in nejihebipi2controller.cpp file under init( ) method. The ***number of time-steps*** and need ***time-interval (dt)*** are needed to be adjusted for learning far away goals and can be done in this nejihebipi2controller.cpp file.

6) Locomotion ***control parameters to be learned*** (here 4 screw velocities, each initialized to 0) are ***initialized*** in wholesteps( ) method under PI2Wholesteps.cpp. Here, initialization can be changed if needed.

7) ***Number of Rollouts*** (int rollout)and ***Number of Updates*** (int update\_no) are defined in wholesteps( ) method under PI2Wholesteps.cpp file. If they are also need to be adjusted for any other task, it can be changed here.

Though, Number of Rollouts generally does not need to be changed. Number of Updates can be increased if far away goals or complex tasks is to be handles.

**Library Description**



8) PI2Wholesteps.cpp has the ***rolloutCost( )*** function which is called for performing the noisy rollouts. It gives the final matrix of all used noises for all parameters that are learned, and also corresponding costs of these rollouts.

All rollouts are performed from initial state vector and start position of the robot, while using noisy control parameters.

9) Matrix “***noise\_matrix***” returned by rolloutCost( ) has the dimensions as number of parameters learned +1. (The extra is for storing the noisy costs). This matrix is given as function argument to updateRule( ) while calling it.

10) ***Cost function getcost( )*** is defined in PI2Wholesteps.h header file. If the function needs to be referred, it can be done here. Presently, this function simply returns the Euclidean distance between reached robot head position and given goal position.

11) PI2Wholesteps.cpp has the ***updateRule( ) method*** and is called from wholesteps( ) method once rollouts are done in an update. This function has the main update rule steps for PI2 which ascertains the parameter update for next run.

At the end of every update, one noiseless cost is obtained to see convergence. The program runs for predefined total number of updates even if convergence occurs, to check that stable learning is there.

12) KinemodelController.cpp file has the control policy of nejiheji snake robot that is used for PI2 learning (in an offline manner). After learning, the final values are passed to nejihebilpzinterface.

13) The kinemodel( ) method in KinemodelController.cpp gives the main kinematic implementation. This is called from rollCost( ) and updateRule( ) in PI2Wholesteps.cpp file. If a different controller is used, then it should be called from rolloutCost() and updateRule() accordingly for getting the costs.

14) General Comments:

* The learned final parameters (u\_final) are limited with +/-1 using total number of timesteps in nejihebipi2controller.cpp under init( ) method. This can be changed if required.
* Also,the starting robot configuration with robot start positions and joint angles are defined in this file under init( ) method and can be changed if needed.
* The noise-free cost at the end of every update gives the distance remaining between reached position and target goal position. It should reach around zero to have convergence.
* For different tasks other than the goal-directed locomotion implemented, also the noise need to be adjusted, which is the only open factor to be tuned. The noise is now generated in rolloutCost( ) for every parameter that is learned and is done simply by having a random number from standard normal distribution. GenerateNoise.cpp is included additionally, in case other noise definitions are needed for other tasks.