# How use the code

There are different versions of the code that have been used for different scenarios.

Version 31 is for trajectory only learning, version 35 and 36 are for trajectory + stiffness learning with divergent force field and stochastic force field respectively. Version 41 is for the HW experiment, with an extra dimension (angle) and connection to the hardware.

## main.m

The software is run from “main.m”. This script reads the file “protocol\_x.txt”, where the parameters of the learning algorithm are given for the different experiments to be executed (one line per experiment). The experiments are executed one after the other in the outer for loop.

The GMM or SEDS model is learned using the demonstration trajectories in the “DemoName.mat” file specified in “protocol.txt” (demo\_set parameter). The GMM and SEDS libraries are used for this purpose (<http://lasa.epfl.ch/sourcecode/>)

The is another for loop to repeated each experiment a given number of times. The function “runPI2GMMLearning.m” is called to perform the learning.

## runPI2GMMLearning.m

The function runPI2GMMLearning() itself does nothing interesting, just some initialization and then calls runProtocol().

RunProtocol() starts by extracting the learning parameters from the initial GMM. It then enters a loop for the learning iterations.

At each iteration the roll-outs are performed by run-rollouts() and stored in the variable “D”. Their cost is computed by the cost functions which name is given as a parameter in protocol.txt, and then the policy is updated with updatePIBB(). Note that there are also other update functions such as original PI2 or PI201 but these functions haven’t been maintained in the later versions.

Run-rollouts() is the function where the system is simulated and the reference trajectory is computed. This is the place where different elements from the scenarios are added, such as the box constraints or the force fields.

## Cost functions

The name cost function that is called by runPI2GMMLearning() to assess the cost of the roll-outs must be specified in the protocol file.

In the cost functions, the first half has to do with computing the “control cost” involved in PI2. This part of the cost is usually set to a low value.

The second half computes that has directly to do with the task. Note that even if some components of the cost are computed as “immediate cost” for each time step, PI black box really only uses the total cost (the sum of the immediate cost over the all trajectory + the final cost).

## Result

For each line in the protocol file, the results are stored in a variable called “results\_this\_protocol” saved to the folder containing the main script under the name “results\_protocol$.mat”, where $ is the line number.

The parameters specified in the protocol file are stored under results\_this\_protocol.p for future reference. The parameters that are not specified in the protocol file are usually indicated by the name of the folder containing the result.

The other important information stored in the result variable are:

* results\_this\_protocol.cost : the different components of the cost at each learning iteration. The third column is the total cost
* results\_this\_protocol.thetas: parameters of the policy at each learning iteration
* results\_this\_protocol.D\_init: trajectory of the initial policy
* results\_this\_protocol.Dfin: trajectory of the final policy
* results\_this\_protocol.Ds : all the rollouts that were performed
* results\_this\_protocol.GMR : final policy in GMR form. Note that muInput and sigmaInput are necessary to compute the outputs of the policy. They are not modified during the learning itereations.
* results\_this\_protocol.random\_number\_generator\_state: use rng() function before running the experiment to restore exact same state of the random number generator and replicate the exact same results.

## Parameters

The meaning of the parameters are described in the “protocol\_xxx.txt” files.

There are some extra parameters that are not defined in the protocol files but in “main.m” or “runPI2GMMLearning.m”:

* Dimensions (positions, angles)
* Stiffness dimensions
* Number of sub-trials

## Hardware in the loop

The architecture for the hardware experiment consists of the matlab code in “PI2GMM41 HW imp\”, of the following ROS components:

/trajectory\_publisher/trajectoryPublisher.py :

Reads .txt files containing reference traj and stiffness saved by matlab ( at ~/catkin\_ws/src/trajectory\_publisher/trajectories/new/traj.txt) and publishes to a ROS topic.

/trajectory\_publisher/trajectorySubscriber.py:

Reads ROS topics published by RobotToolkit and saves as .txt file to be read by matlab at ~/catkin\_ws/src/trajectory\_publisher/recordings/rec2.txt

/gravity\_compensator/gravity\_compensator\_node:

Subscribes to F/T sensor topic and KUKU pose topic and converts F/T values from sensor frame to world frame and subtracts gravity forces due to shovel itself.

In order to operate this node, commands must be published on ros topic /GravityCompensator/ctrl (std\_msgs/Int32):

* 0: reset
* 1: get all samples and estimate model
* 2: get half of all samples and estimate model.
* 3: get 4 samples. Estimates models when 20 samples are gathered.
* 4: Turn compensation on
* 5: Display estimated parameters
* 6: Save model
* 7: load model
* 8: tare offset

A model is already saved, so you can go 7 -> 4 -> 8

/netft\_rdt\_drive/netft\_node (use arguments 128.178.145.212 and –rate=500)

Published F/T from nano25 data on ros topic

RobotToolkit package : packages/traj\_follow/KUKATRAJ

Takes care of the actual impedance control of the KUKA

## TrajDrawer.m

A class I created to draw demonstrations by hand.

When an TrajDrawer object is created, a figure is opened. Left click and drag to draw trajectories (sampling rate = 10Hz). Right click to erase latest trajectory.

Trajectories can be retrieved in object.dataPoints