learnOptimWBC

Description

learnOptimWBC is a Matlab framework to study the combination of learning and prioritized multi-task control for redundant robots. In such settings, a robot has to accomplish a main global task by the simultaneous execution and combination of several different elementary tasks.
Our framework is specifically used to study multi-task controllers with soft task priorities: our goal is to learn the temporal profile of the task priorities in an automatic way, without the requirement of the expert user manually tuning the task priorities and their evolution in time. We are studying these methods on redundant robots (arms, humanoids). THIS IS WORK IN PROGRESS!

If you are going to use parts of this code, please cite us:

Modugno, V.; Neumann, G.; Rueckert, E.; Oriolo, G.; Peters, J.; Ivaldi, S. (2016) Learning soft task priorities for control of redundant robots. Proc. IEEE International Conf. on Robotics and Automation (ICRA).

Installation

- Unzip /learnOptimWBC/matlab/Dependecies/robot-9.10.zip somewhere handy (or in place). This operation will create a folder called rvctools, containing the robotics toolbox that we rely upon. Add /rcvtools and its subfolder to the Matlab PATH (you can right-click on the folder name in the Matlab panel list). To install the robotics toolbox run the rvctools/startup_rvc.m (Every time you have to run a script from the Matlab shell you have to change the current Matlab folder using the directory view that is usually on the right side of the Matlab window) from the Matlab shell.
- You need to compile the mex file for the robotics toolbox. Depending on your Matlab version, you will need an appropriate compiler version.
  For example: Matlab 2013a supports gcc until version 4.4. If you have a newer gcc (probably 4.8 if you have Ubuntu 14.04) then add it to your system:
  `sudo apt-get install gcc-4.4`
  `sudo apt-get install g++-4.4`
  Then manually edit the file:
  `/home/<YOUR_NAME>/.matlab/R2013a/mexopts.sh`
  and update the name of your compiler:

`CC='gcc-4.4`
`CXX='g++-4.4`.
More instructions on how to set up your compiler:
http://fr.mathworks.com/help/matlab/matlab_external/changing-default-compiler.html

- compile the mex file for the Newton-Eulero dynamic computation in the robotic toolbox (`/rvctools/robot/mex`). Check that you added rcvtools to the Matlab PATH (including subfolders). Then run make.m from /rcvtools/robot. Note: Matlab may complain about line 304 in frne.c and lines 465-468 in ne.c, manually edit the files to change all the \ comments into /* … */ comments.
  If everything works well, you will read:

```
>> make
** building mex file
```

- Move the MEX file frne.xxx (where xxx is the extension of the mex files on your platform - for example frne.mexa64) in rvctools/robot/@SerialLink. The MEX file will now be used instead of the M-file, and thus anything that calls rne() will use the MEX file and be faster. This applies to inertia(), coriolis(), gravload(), and fdyn().

- once the installation of the robotics toolbox is done go to /learnOptimWBC/matlab and run startup_LOWBC.m. This script will add the paths to Matlab and it will update some files inside the robotics toolbox copying them from the folder /ChangedFile. After this, please remove from the PATH the folder /ChangedFile.

Now you are ready to try learnOptimWBC toolbox on your machine.

Code Structure

- *Classes* contains all the classes that define the functionalities of the toolbox,
- *Common* contains many support functions and some routines to generate the reference for the controllers,
- *DesignToolbox* contains a set of scripts to design new scenarios and new controllers,
- *Exe* contains functions all the executables for the simulation of the learned controller,
- *Interface* contains some routine to use v-rep as dynamic simulator,
- *Robot* contains the robot models and their symbolic representations,
- *TestResults* contains the results of each optimization, the scenarios and a configuration file for each implemented controller,
- *Interface* contains utility functions,
- *UsefullFile* is not used.

How to add a new controller

If you want to define a new global controller given a specific robot, you have to define the parameters in AllStaticParameters.m and UF_StaticParameters.m both in /matlab/DesignToolbox

in AllStaticParameters.m

- *CONTROLLERTYPE* : string defines the kind of controller, ‘UF’ for our method or 'GHC’, a method used for comparison,
- *subchain1* : row vector defines which frame we want to control on a given kinematic chain. Example: 6 DOF robot for an e-e cartesian controller –-> subchain1=[6],
- *[bot1]* : the robot that we want to use. pick the model from /matlab/Robot folder by specifying the Mdl***() functions,
- *traj_type = {’cartesian_x’,’cartesian_rpy’,’joint’}* : specifies the kind of controllers that we want to use (one for each elementary controller),
- *control_type = {’regulation’,’tracking’}* : defines if we want to reach a point or if we want to follow a trajectory (one for each elementary controller),
- *type_of_traj = {’none’,’sampled’,’func’}* : for tracking= ‘sampled’ if we want to generate a sampled trajectory ‘func’ for closed form trajectory
  for regulation = ‘none’ (one for each elementary controller),
- *geometric_path = {’rectilinear’,’lemniscate’,’circular’, 'none’}* : defines the kind of trajectory that we want to follow for a tracking task. ‘none’ for a regulation task (one for each elementary controller),
- *time_law = {’exponential’,’linear’,’none’}* : defines the kind of time law that we want to follow for a tracking task. ‘none’ for a regulation task (one for each elementary controller),
- *geom_parameters{1,i}* : defines the geometric structure of each task. Look inside the function in /common to know which are the parameters that we specify for each trajectory. In the regulation case is necessary to specify just the arrival point (one for each elementary controller),
- *dim_of_task{1,i}* : is a vector of 0 and 1 that define the dimension that we want to control for each task. 1 = dim active 0 = dim not active (one for each elementary controller),
- *id * : the name of the configuration file for the current controller that has to be specified in /matlab/Exe/allruntimeparameters.m if we want to use it for in a simulation.

in UF_StaticParameters.m

- *metric* : cell array of strings. According to (peters07) we can define different matrix that produces different control structure,

- *kp* : row vector that defines the proportional gain (one for each elementary controller).

How to add a new scenario

Creating a new scenario is a trial and error procedure. For the obstacles this toolbox provides a global variable called 'G_OB'. The obstacle supported in this library can be 'wall' and 'repulsive point'. The class 'obstacle' contains all the information to properly set up new obstacles.
The script /matlab/DesignToolbox/DesignScene.m is a sandbox where the user can visualize the scenario with the obstacles, the robot and the trajectories given by the low level controllers that satisfy a tracking task.
As we said the scenario design phase is a time consuming operation but in our knowledge this is a common issue for every simulator where creating a new scenario can be a painful task.
To start give a look at /matlab/TestResults/scenarios where is possible to find some simple scenarios.

How to run an experiment

Given a scenario and a controller we can run a simulation (with /matlab/Exe/MainExec.m) or learn the parameters in the controller (with /matlab/Exe/MainOptRobust.m). For both of this is necessary to properly set the parameters in /matlab/Exe/AllRuntimeParameters.m

For both MainExec.m and MainOptRobust.m we have to set in /matlab/Exe/AllRuntimeParameters.m

- *time_struct* : is a structure that defines the duration of the current experiment and the step size, time_struct.ti ; time_struct.tf; time_struct.step,
- *time_sym_struct* : the same as above but is used only for the integration inside the simulator,
- *fixed_step* : defines the kind of integration of the differential equations in the simulator,
- *torque_saturation* : defines the maximum allowed torque in the simulator (in Nm),
- *name_dat* : defines the name of the controller that we want to use in the current experiment,
- *name_scenario* : defines the scenario of the current experiment,
- *qi{1}* : starting joint positions of the robot,
- *qdi{1}* : starting joint velocities of the robot,

- *choose_alpha = 'RBF'* : defines the structure of the activation policies used in the current experiment. 'RBF' is the default selection,
- *number_of_basis* : defines the number of basis functions for the RBF,
- *redundancy* : defines the overlapping of the basis functions,
- *numeric_theta* : is a set of values that are assigned to define the time profile of each activation policy. Not used in MainOptRobust.m.

Only for MainOptRobust.m

- *explorationRate* : defines the exploration rate of CMA-ES (the optimization algorithm)
- *niter* : defines the number of generations of the optimization algorithm
- *fitness* : defines the fitness that is used to evaluate the quality of the current roll-out.