Practical Course Robotics

Marc Toussaint

April 7, 2016

Contents

1 Introduction

2 Setting up your work environment

Prelimiminaries

- You need a gitlab account; access to `mlr_students`
- Connect to the local mlr-robolab WIFI

Install from a fresh Ubuntu

- install fresh Ubuntu 14.04.4 LTS
- google 'ros install indigo'; copy&paste steps; install package ros-indigo-desktop
- install packages: synaptic, git, qtcreator, ros-indigo-alvar-msgs, ros-baxter-...
- create ssh key:

```
cd
ssh-keygen
cat .ssh/id_rsa.pub
```

- enter ssh key in gitlab: gitlab start page; profile settings; ssh keys; copy&paste the key (without linebreaks!!!); 'Add key'
- in gitlab go to the project page; see the ssh URL ending with ...git
- checkout our code

  ```
  cd
  mkdir git
  cd git
  git clone <SSH-GIT-URL>
  ```

- Install the code dependency ubuntu packages:

  ```
  cd ~/git/mlr/install
  ./INSTALL_ALL_UBUNTU_PACKAGES.sh
  ```

  Trouble shooting: read the README.md in /git/mlr

- configure code and test make:

  ```
  cd ~/git/mlr/share/
  git checkout baxter
  cp gofMake/config.mk.default gofMake/config.mk
  bin/createMakefileLinks.sh
  cd src/Ors
  make
  ```

- goto project page and test make

  ```
  cd ~/git/mlr/share/teaching/RoboticsPractical/01-...
  make
  ```

  Test starting to run `./x.exe`

Make the baxter move

- setup the WIFI connection to the baxters ros server

  `source ~/git/mlr/share/bin/baxterwifisetup`

- In a project folder, try to run `./x.exe -useRos 1`

Get comfortable

- put all extra documentation useful for others in text files in ./doc
- use qtcreater; learn create 'new project' (using 'import existing project' for a path with makefile); learn to set 'include paths'
- create own folder `groupX`, maybe own branch

3 Plan

3.1 Milestone 1: Pick-and-place

Target: The robot perceives objects on the table (= segment, localize). The robot grasps them and puts them into a bin.

3.1.1 Subproblem: Basic Motion

Learn how to use our code to generate targets in various task spaces. Learn how create `CtrlTask`s directly in C++. Optionally, have a look at the much mroe abstract RAP interface.

3.1.2 Lecture: Basic Motion revisited

- Task spaces, general problem
- linear acceleration laws in task spaces
- maths to project them down to configuration space
- Discuss (practial is later): impedance, stiffness

3.1.3 Subproblem: Segmenting & tracking objects

Understand how the `tabletop` ROS packages can extract planes (the table) and point cloud clusters on top of the plane. Learn how the objects are imported in our system.

3.1.4 Lecture: Basic perception

- The pain of computer vision...
- Keep it simple: point clouds, planes, clusters, markers
- Practical packages

3.1.5 Subproblem: Pick & Place

Realize the whole pick-and-place scenario. Core issues are

- Designing the motion tasks
- Sequening, ideally failure detection & reaction

3.2 Milestone 2: System Identification, Machine Learning & Compliant Optimal Control

Target: The robot is controlled on the lowest level, sending direct 'torques' (or alike). Using system identification (ML) we learnt a perfect model of both, the dynamics and the observations. Using Bayesian filtering we can perfectly track the state—giving nice and smooth velocity estimates. The robot 'intelligently' explores its state-space to collect data for the previous tasks.

3.2.1 Lecture: Dynamics Basics; and motivation

- Dynamics & optimal control revisited
- Compliance, impedance control, manipulation & teleoperation
- (Do we have F/T sensors?)
- caveats of real robots: 'non-Markovian', sticktion, time lag, gear clearance

3.2.2 Subproblem: Collect data, formulate model, ML

Think about motion patterns to collect data. Formulate models for the robot dynamics as well as observation model. Apply ML.

3.2.3 Subproblem: Use the model for (extended/unscented) Kalman filtering of the state

3.2.4 Subproblem: Use the model to translate desired q-accelerations directly to torques

3.3 Define your own project!