Manipulation
Final Project

Contents

Overview

This documents presents a number of projects to work on as part of your final project in the ROS class. The final projects can be performed either on the real Baxter or the UR5 robot. We will give you as much support as possible to do that. You can also include a simulation of the work, but you must perform the project on the real robot.

If you can finish the simulation successfully, then you can try on the real robot. Your project counts for 30% of your final grade (HWs count 70%). If you can get your code to work on the real robot, I will give you 20% extra points on your final grade!! So if your final grade was going to be 80, you would get 100%!!

We will have a **Project Presentation Day** in which all students will demo their work together on Friday, January 13th.

Deliverables

At the end of the project you are required to

A) Document your demo on the official birl_baxter demo wiki page or birl_ur5 demo wiki:
https://github.com/birlrobotics/birl_baxter/wiki and provide a clear English and Chinese explanation of the demo, video, and working code. See the face follower demo as an example:
https://github.com/birlrobotics/birl_baxter/wiki/face_follower_en

B) Push your force controller

You can push your code to either birl_baxter_controllers, or birl_ur5_controllers according to what robot you end up using. Dr. Juan will give your read/write permissions on GIT after you provide him with your username. This will allow you to push directly to either of the repos.

C) Present your project during project day

You will give a description of what you did to the group, then you will show how your demo works.

GitHub

The first step is for you to register on **github.com.** This is important because I will need to add your username as part of a Team in the birlrobotics github page.

So, please sign up and then send this information to my (Dr. Juan's) email: stating your name, email, and github user name. After that, I will add you to the appropriate github team.

How to Push the Code to the birlbaxter repository

Please look at GIT Tutorials in depth before doing this part. It's important you have a good understanding of what GIT is.

Before we push any code, we need to make sure that we do the following steps:

1. For Baxter, install the birl_baxter code by following our wiki:
   http://www.github.com/birlrobotics/birl_baxter/wiki

   After you clone the packages, you should find the folders for birl_baxter_controllers or birl_ur5_controllers (i.e. ~/ros/indigo/baxter_ws/birl_baxter/birl_baxter_controllers).

   For ur5, you need to clone directly.
   You can create folder inside baxter_ws called birl_ur5, and then do:
   git clone https://github.com/birlrobotics/birl_ur5_controllers.git

   Note that currently this package is empty. You would need to create your own ros node inside to emulate the code that is available later.

2. Then you will integrate your code. Adding code to git requires 2 stages.

   Stage 1: Create a branch for your code.
   In order to not hurt the existing code, it is better for you to create a git branch inside the repository you want to change. Search for online tutorials if you are not well aware of a git branch.

   For example, if you want to work with baxter, you first go inside birl_baxter_controller, there you can type: "git branch", you will find a list with two existing branches:
   $ * master
   $ simplifiedPID

   Here you can create a new branch by typing:
   git checkout –b #nameofnewbranch#

This command will do 2 things: create the new branch and switch the new branch. Now when you change the code, it will not affect the original code. At the end, if your code is successful we can do something called “merging the code”

The first stage 3 steps (add, commit, push). Before I describe them below, let me introduce “git status” and “git log” which help you to understand the state of git.

git status: “git status” will help you to see what has changed, what has been added if any.

git log: “git log” will help you to see what has been committed as well as the commit message and the author of the change.

a. **Add**: This means you tell GIT which files you would like to set apart to add to the repository. Why do this? You don’t always want to add ALL your files at one time. Think of GIT as a system that keeps track of all your file changes over time. It also allows you to undo. So, if you undo, you want to undo a set of files that are related to the same problem, but you would not want to undo files that were good changes. For example, if you changed kinematics files and also joint_trajectory files, don't mix these two together, add changes in kinematics first, and then add and commit changes in joint_trajectory files second. This way, if you ever need to go back in history, you can isolate the changes to a single area. It becomes more difficult when you mix changes in many places.

```
// cd to the right directory
git add filenames
```

b. **Commit**: The commit officially records the changes that exist for the files that you added. It will create a commit number which will be kept in a history database. This number is incredibly important, as you can always to back in time to this commit and recover the exact code that the git repository had at this time.

Every commit needs a “commit message”. This commit message needs to be very clear and detailed to help a programmer know what happened here.

You can start a commit by typing:

```
$ git commit
```

At this point an editor window will show up. I recommend putting in your commit message one of three types of messages: General Description; Enhancement; or Bug Fix. After you put the message, just save the file and close it, the message will be saved with

the commit. You can run "$git log" to see the latest commit.

General Description: For example, let's say this is the first time you create an impedance controller, then your message could look like this:

```
Impedance Force Controller Implementation:
- ROS package that implements an impedance controllers in C++.
- Applies a mass-spring-damper system between target position and desired position of tool tip of robot.
```

Enhancement: Let's say that you already created your impedance controller. But now, you made it better, say you improved performance and cleaned up the code. Here you can explain that part:

```
Enhancement:
- Improved performance by using threads
- Clean up the code
```

Bug Fix: Thi is the last type of message you might want to include. It is when you find that something was wrong in your code and then you fix it.

```
Bug Fix:
- Fixed wrong math computation in force controller loop.
```

c. **Push:**
Once you have committed, it's time to send it to the cloud (git) server. The push commands takes two arguments: (i) the repository name, and (ii) the branch. For your case, they will be "origin" and the name of the branch you created earlier. So you can push by typing:

```
$ git push origin master
```

It will ask you for your username and password. This will only work, if I have added your git username as a team member to our repository, so remember to send me your account username!

ReadMe

All packages, always include a descriptive ReadMe files. It's always important to include a descriptive Readme file to explain to new users what the package does. Create a ReadMe file at the top of your

package and then add, commit, push. This ReadMe file contents, will also go on a wiki, I describe how to do this in the next section.

Documentation

1. Format

Create a tutorial guide in English and in Chinese. The format of the tutorial should follow that of the Rethink Robotics' SDK (see example here: http://sdk.rethinkrobotics.com/wiki/Enable_Robot_Example).

Again, an example of such documentation can be found at:
https://github.com/birlrobotics/birl_baxter/wiki/face_follower_en

It's very important that you include all the following sections:

- Summary
- Quickstart
- Overview
- Structure
- Video
- FAQ

Markdown Language
These demos are written using a language called markdown language. Markdown is very easy to use and makes your documentation look great. Github supports markdown language natively.

https://stackedit.io/ is a webpage where you can learn to use markdown and see how your document looks in real-time. You can test your documentation here first, or you can also try github's preview function.

Updating the Package Wiki
1. Please include your package description in the wiki for the corresponding package: birl_baxter_controllers wiki or birl_ur5_controllers wiki. When you finish, documenting you can create a new page (one for English another one for Chinese) in our wiki and paste and edit the documentation you created in stackedit there. Please include a link on the top of your wiki page to the English and Chinese versions. I.e. you could have an "EN | CN" text at the top. If you are in the English page, make sure the "CN" has a link, if you are in the Chinese page, make sure the "EN" has a link.

2. Video

Your package demo must include an (embedded) professional looking video. That means, your video must have:

- Slide 1: A title page at the beginning of the video:
  - ↘ name of project,
  - ↘ names of authors,
  - ↘ name of Professor: Dr. Juan Rojas,
  - ↘ name of course: 2016F-Manipulation, and
  - ↘ link to our robotics lab web page: http://github.com/birlrobotics
- Slide 4: A brief description of what the demo is about, and
- Slide 5: A video of the demo with a vocal explanation in English.

The video should have a 720 or higher resolution. Such videos can be made with Windows Movie Maker, Adobe Premier Pro, and others. See what a professional video might look like here.

Videos should be hosted on a site like youku/youtube (so you can embed them in the wiki page) and also provided them to Dr. Juan on a USB/CD, I will add them to our BIRL channel on both youtube and youku. The name of the video should contain the name of the force controller you implemented in the file.

Grading

Working Force Controller Project		70%
GIT	- Proper GIT code and package	10%
Wiki	- Complete 6 section description	10%
Video	- Professional Video	10%

Project Overview

There will be one project per two students.

A number of projects will be described below. Some of these projects needed to be created from scratch, others have been created by others before, and need to be implemented by following their instructions (sometimes there may be no instructions). Each project has its own set of challenges, and there may be no single one that is easier than the others.

General References for Force Control

I will be including:

- A PDF along with your project description in a zip file called “Force Control” from the Springer Handbook of robotics. It overviews the main force controller techniques that are needed in this project. This will be a very useful for your reference. If you want more details, you can look at the citations (good for task 1)
- Force Sensors in Robotics: This general file will also be good to you as it’s a good introductory text and points you to useful ROS packages.

Books

- Robotics: Modelling, Planning and Control by B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. Available in the lab library. See Ch 9 for force control.
- Introduction to Robotics, by Roger Craig. See Ch 11 for force Control. Will attach as a PDF (good for task 2)

Slides

- Few other slides for impedance control, hybrid force/position control, and control basis.

What Task should I implement?

You can try different types of task. Here are a couple of recommendations:

- Touch Avoidance
  Set the set-point to zero. Let the manipulator hang in the air, then with your finger press the end-effector in any direction (linear or angular motion). The manipulator should move away from the direction of push to return to a 0 force 0 torque configuration. See an example here.
- Polishing
  The end-effector has to touch the table flat and move sideways to polish. This requires the set-point to be some number in the z-direction (5N is around 500g of force) and then some lateral motion command. See an example here.

- More Advanced
  You can also try to implement the snap assembly, see example here and here.

Simulation Details

Here I will provide some details for simulation preparation with Baxter.

1. Make sure you have the latest code **consistently** (i.e. do this everyday at the beginning of the day).

   You can update your code by moving inside the birl_baxter folder:
   ~/ros/indigo/baxter_ws/src/birl_baxter

   And then running the following command:
   wstool update // will do git pull origin master on all git repositories.

2. Launch Baxter in Gazebo
   - We have 3 simulations that are in good shape. You can learn more about them here (these are updated continually):
   https://github.com/birlrobotics/birl_baxter_simulator

   - For the "Avoid touch" and the "Polishing Task", you can probably use the gazebo setup found here:
   roslaunch birl_baxter_description pick_n_place_box_gazebo.launch

   - How to move the arm to the table?
   If you are doing the polishing task, you can follow/modify the python script found in:
   baxter_ws/src/birl_baxter/birl_baxter_simulator/birl_sim_examples/scripts/pick_n_place_box.py

   - This will teach you how to move the arm to the table with the object in hand. However, you will need to stop as soon as the object has touched the table.

   - After that you need to start the force controller. At first, you can do this manually using a launch file to start the force controller (see the next point) after the robot's end effector is on the table, a more advanced way (and a better way) is to use a state machine using SMACH. There are examples of smach in birl_baxter_examples: example_smach.

3. Set Point
   - If you want to launch the setpoint similar to the control basis setpoint launch file, you can do that for simulation by including the argument sim:=true in the following way:

roslaunch force_controller setPoint_publisher.launch sim:=true

4. Applying Forces in Gazebo
   - If doing the “Avoid Touch” experiment, you can follow the following tutorial to add a manual force to the simulation:
   http://gazebosim.org/tutorials?tut=apply_force_torque&cat=tools_utilities
   http://answers.gazebosim.org/question/241/how-to-insert-a-force-or-torque-sensor-to-gazebo-model/

   - If you are doing the “Polishing Task” then you can use the table provided in the pick_n_place_box world in gazebo.

Force Control

105 “Baymax” Workstation

This workstation is already set up in many ways to run experiments: it has the latest:

- ROS,
- Baxter,
- BIRL code.
- It has the proper Ethernet and Wacoh internet configurations to connect to the right device. And it is a very powerful computer.

This computer is shared by *many people*, so you have to be very careful how you use it.
<< Please never disconnect it from the internet, do not kill the TeamViewer server etc. >>

- To talk to Baxter, you need to be connected to the BIRL 105 wifi connection.
  Note: the FT is connected through the Ethernet cable on the back.

- If the FT sensor is connected by USB, you need to select the Wacoh connection on the network menu.

- Running Code inside Baxter
  You can also run your code inside Baxter’s computer using SSH, ask me if you want to try this. (This method is the fastest and provides the best performance).

Force Torque Sensor

Currently in the lab, we only have ONE force-torque sensor, called the Wacoh FT Sensor. If you want to use it, do the following:

1. Connect the FT sensor to the power source, and verify the Ethernet cable is also connected on the back of the workstation.
2. Make sure all connections are properly setup in the Anaolog-to-Digital Converter (you should see a green light on the black box),
3. Finally make sure you connect to the **pre-configured** “Wacoh” ethernet configuration in the linux network configuration.

Force Control Approaches

Indirect Force Control

These methods are primarily motion controllers, which only apply a force constraint when the position of the robot deviates from the target position. These controllers do not explicitly "close the force feedback loop." This means that, technically, they don't need force torque sensors to be implemented. However, often it is a good idea to use force torque sensor data to improve the controller. This type of control has variations that lead to different names like (Admittance Control, Compliance Stiffness Control, or Damping Control).

1. Impedance Force Control

Direct Force Control

This method incorporate both force and motion into the control of the robot. They require two different inputs - the target position/motion and the target force. As a result of this, some direct force controllers must have a very clear model of the task.

2. Hybird Force/Position Control

Modular Force Control

This method takes force or moment goals and combines them through the use of null space projections. This code has been implemented, but needs parameter tuning.

3. Control Basis Control

Project Details

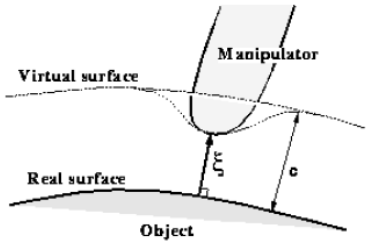
Please refer to the following document for a simple and general introduction to force control (also available in the zip Project file):
http://blog.robotiq.com/hubfs/Force_Sensors_in_Robotics_Research.pdf

Also see this video on PID controllers, most force controllers use PID to run their loop. Some force controllers may sit on top of a position controller or a velocity controller that also uses a PID to finally move the arm: https://www.youtube.com/watch?v=JEpWlTl95Tw

In this project, we have no model of the robot or the environment, so we will depend on PID control to implement the work. For this reasons, it is very important that you feel familiar with PID. You can also do an tutorial in matlab that provides a lot of insight. You need the Control toolbox to do this, which is available in the Baymax workstation in our lab:
http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID

1. Impedance Control

This applies a mass-spring-damper between the target position and the actual position of the robot. Imagine that you attach a spring between your finger and some target position in the air. The further you move your finger away from that point, the more force the spring applies to pull you back to that point.



Reference
See the following link, p. 4-9.

Videos
Here are a few videos that demonstrate what impedance control looks like:

Nice demo:	https://www.youtube.com/watch?v=WS1gSRcJbJQ
Detailed presentation:	https://www.youtube.com/watch?v=DPTSorK3QDE
	https://www.youtube.com/watch?v=Ln4EqJntDBY
Holding an egg:	https://www.youtube.com/watch?v=Mq89FDmQfvA
Another nice Demo:	https://www.youtube.com/watch?v=-rev_0oPy24

Sample Code
You can find two sample codes for an impedance controller in ROS for the pr2 robot. This code would have to be tested and adapted either for Baxter or the UR5. The first one is very detailed, with tutorials, but it has not been maintained for some time. The second one is another example.
http://wiki.ros.org/ee_cart_imped (find the git code link inside)
https://github.com/birlrobotics/ImpedController

2. Hybird Position and Force Control

Reference
See the general reference mentioned at the beginning of the project demos section. Very well documented.

Videos
Here are a few videos that demonstrate what impedance control looks like:

Nice presentation: https://www.youtube.com/watch?v=W4jrnlV-JEM
Descriptive Simulation: https://www.youtube.com/watch?v=DqEGZF-PbjY
Robot doing massage: https://www.youtube.com/watch?v=53kpMPN9hOY

Sample Code
There is an incomplete version of the code with Baxter. Need to change the code to work with the Wacoh FT sensor rather than the robots internal torque sensor. You need to make this into an official package and get it to work.

Wiki:
https://github.com/birlrobotics/birl_baxter/wiki/Hybrid-Position-and-Force-Controller-(unfinished)

And

Code:
https://github.com/birlrobotics/birl_baxter_demos/tree/master/hybrid_position_force_controller

You can also look at the control basis code which is very similar, see below.

3. Control Basis Control

Reference
See the control basis folder in the reference folder for the project for some guidance on this. Most code has been implemented here. But needs parameter tuning and some debugging.

Videos

Here are a few videos of what control basis can do:

Two robots performing an assembly: https://www.youtube.com/watch?v=ACz_Y9SztIc

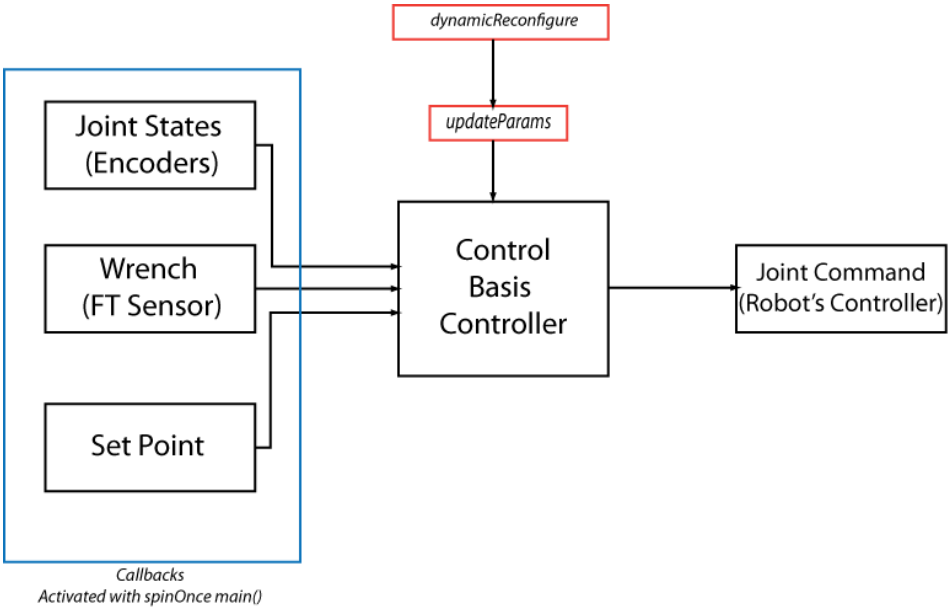
Snap assembly: https://www.youtube.com/watch?v=paBixne_d6w

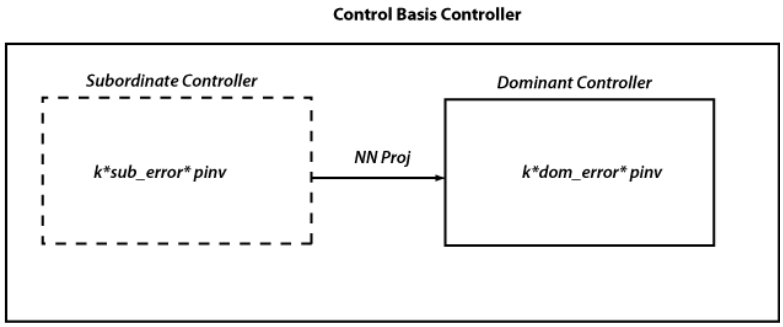
Sample Code

Sample code can be found at:

https://github.com/birlrobotics/birl_baxter_controllers/tree/master/force_controller

Overview of Code





Further Documentation

This package has extensive documentation in the ReadMe file, including descriptions about how to launch the file. For the control basis force controller package, you will only running the "topics" based controller and not the "service" based controller.

..birl_baxter_controllers/force_controller/documentation/Readme.txt

Also, check out the API of the class, which was created with rosdoc lite, by opening the index.html file in a browser.
..birl_baxter_controllers/ documentation/doxygen-doc/html/index.html

- A note on PID Gains:
  I have simplified the code so that the three gains you put in the setpoint launch file in fact are numbers for the P gain, the I gain, and the D gain in that order.

  You will only use one number per gain (i.e. just a scalar) to multiply by the error vector (instead of three numbers).

Project Day

Project Day will be an opportunity for you to show what you accomplished. This event will be open to the public. All your classmates will be there as well and will look at each other's work. Every team is required to present their project in the real-robot and if you have a simulation that too.

Date: Monday, January 16th.
Location: Our robotics lab (maybe we will have moved by then), so exact location is TBA.
Time: 3:45pm – 6:00pm.

Each time will have 20 minutes to setup and run their demos.

3:45-4:00 Welcome
4:00:4:20 – Team 1
4:20-4:40 – Team 2
4:40-5:00 – Team 3
5:00-5:20 – Team 4
5:20-5:40 Conclusion