Robotics
2014
Practical Challenge 1
Sensor and Actuator Wrapper for a Differential Robot and Proportional Control

Vladimir Estivill-Castro
MiPal

July 26, 2016

Objectives

1. Being able to program a robot to carry out some simple *reactive-control*.
2. Use Logic-labeled finite-state machines (*llfsms*) to construct a behavior.
3. Have an introduction to robotic middleware and the PUSH-approach with an introduction to `ROS`.

Advanced Objectives

1. Design a wrapper (to control sensors and actuators) for a differential robot that can be used with `ROS` and the `LEGO-NXT` brick. However, to create the API for the wrapper so that it meets the following requirements.
   (a) It can be generalized to *MiPal*'s Object-Oriented Whiteboard. Thus the API is designed for a `Control` message from the controller to the robot, and a `Status` message from the wrapper to the controller.
   (b) It is as portable as possible to Ubuntu 14.04 and `MacOS` 10.9 to 10.11.
   (c) It can be extended to a simulator like
      i. `Webots`.
      ii. `MORSE`.
      iii. `Gazebo`.

   It can be extended to be used with the `Giraff` in the *MiPal* lab.

2. Apply the API to develop a simple program (using `clfsm`) to maintain the robot caged inside an environment with walls (backing away from obstacles).

3. Develop (using `clfsm`) a simple example of a feedback control loop.

Warning

Be aware that every single piece of software used here is in fact, like all software, software in evolution. Note that `MacOS` rapidly has migrated from Leopard, to Mountain-lion, to Mavericks, to El Capitan. Ubuntu has moved in a few years several versions. Laptops have moved from 32-bits to 64-bits. `ROS` has moved from Indigo to Hydro. `Webots` from 6.X to 7.X, and to 8.X. Thus, you have to be patient and resilient. The *MiPal* software aims for being as cross-platform and as standard as possible.

This challenge should not require the *MiPal* infrastructure. The aim is that it should be completed with the released versions for `ROS`, and every other shareware.

Background

You should have some experience or you may want to refresh on the following.

1. POSIX tools and UNIX shell (specially Ubuntu and the MacOS terminal).

2. `C++11` programming. You may need background in threads and tools that avoid busy waiting of the CPU.

Instructions

This laboratory sessions has several parts.

1. Becoming familiar with `ROS`.

2. Creating a behavior for the `ROS`-turtle with logic-labeled finite-state machines. This behavior should illustrate reactive control. The robot does not escape. We simulate its touch sensors that recognizes the wall.

3. Create a similar reactive behavior in with a real robot, the the `LEGO-NXT`. In this case the sensors recognize obstacles and the robot backs-off. But also, observe how the need for robotic middleware appears.

   In this part you will install the `NXT_dirver` and the `NXT_controller` which are now `r2d2mipal` libraries (derived from `nxtpp0-5`) that enable to program (using the standard `C++11` compilers in LINUX-Ubuntu with 32/64 bits) the `LEGO-NXT` brick via a USB connection (or blue-tooth).

   This library provides to control the actuators; namely, the motors and the speaker. It also makes available information from the environment via a touch sensors, a rotation sensor and/or a sonar sensor.

4. We will explore programs that directly build the behavior using the library , but then, as we create an abstraction to a *llfsm*, the need for middleware will appear.

   We will use a simple `ROS` API to create a hardware abstraction (controller) and reproduce the behavior with the model of a *llfsm*.

This laboratory exercise requires you to investigate several topics. You are invited to place references to the literature in your report. Most importantly, you are to provide feedback on the instructions and the documentation of *MiPal*'s tools (feedback on this text is also welcome).

Tools

1. `ROS`-Indigo for `Linux` 14.04 as per the labs in term 1, 2015 at UPF.

2. We will be using `NXT_dirver` and the `NXT_controller` which are now `r2d2mipal` libraries (a version of `nxtpp0-5` that was updated at *MiPal* by Esteve Fernandez [extension to MacOS 64-bits], Carl Lusty [Raspberry Pi], and by Vlad Estivill-Castro [*MiPal*'s Make-file and `ROS`]. It should run for `Linux` and `MacOS`, and it has been mapped to run using `catkin` in `ROS`.

   You should be familiar with `ROS`(see the tasks). In particular with `catkin`, at least to the extent of knowing about packages and how they are placed in directories. The you can visit *MiPal*'s Website download section[1] and download the `clfsm.tar.bz2`.

   The share-ware `nxtpp0-5` was only for for LINUX, and unfortunately this share-ware that is not supported anymore, and whose documentation is lost as the host Web site for it is no longer maintained. Therefore, somewhat in *MiPal* we maintain our version, but please report and if possible fix compatibility issues (`Linux` 32-bit vs 64-bit, blue-tooth compatibility, `ROS`-compatibility, etc). For this challenge, it is fine if you run it on the host computer and connect to the robot using a USB cable (creating a tethered robot). The `README` file for `nxtpp0-5` says the following.

   > NXT++ is an interface written in C++ that allows the control LEGO MINDSTORMS robots directly through a USB connection. The interface is intended to be simple and easy to use. The interface can be used in any C++program.

   Its has the following requirements.

   - A standard C++ compiler (note that `C++11` standards continue to evolve and compiles like `gnu` and `clang` are catching up at different rates).
   - The USB library: `libusb`: `http://libusb.sourceforge.net/`. Usually installed in later version of Ubuntu with
     ```
     sudo apt-get install libusb-1.0-0-dev
     ```
   - The blue-tooth library `libbluetooth`: Usually installed in later version of Ubuntu with
     ```
     sudo apt-get install libbluetooth-dev
     ```
   - The knowledge of basic compiling operations in your compiler.

[1] mipal.net.au/downloads.php

3. The LEGO-NXT Mind-storms

   The LEGO Mindstorm series of kits contain software and hardware to create small, customizable and programmable robots. They include a programmable 'Brick' computer that controls the system, a set of modular sensors and motors, and LEGO parts from the Technics line to create the mechanical systems. See http://mindstorms.lego.com/en-us/Default.aspx. We have build them in a standard form as a differential robot, with touch sensors as bumpers, a light sensor, and a sonar sensor. If you are interested, contact the instructor and we can make available a copy of the Docs folder of the *MiPal* repositories that has a folder NXT with the documentation on how to assemble the LEGO-NXT for the current differential robot, nevertheless please do not modify the hardware.

First task.- The basics of ROS

ROS beginners tutorials

ROS should be installed in the labs. To be able to use the environment you need some set-up.

```
export ROS_HOSTNAME=localhost
export ROS_MASTER_URI=http://localhost:11311
```

However, it is ok if you install in in your own laptop use the same versions ROS-Indigo and with Ubuntu 14.04.

> We have also tested most of what we describe here in MacOS with ROS-Hydro installed with Mac-Ports; see http://wiki.ros.org/hydro/Installation/OSX/MacPorts/Repository).

Your next task consist of installing ROS by following the instructions for Ubuntu (we recommend Indigo). Then, follow the following tutorials from the Beginner Level (http://wiki.ros.org/ROS/Tutorials We basicly are looking at all the C++11 sections, but if you want to do other programming languages, that would be beneficial as well.

1. Installing and Configuring Your ROS Environment
   http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment
2. Navigating the ROS Filesystem
   http://wiki.ros.org/ROS/Tutorials/NavigatingTheFilesystem
3. Creating a ROS Package
   http://wiki.ros.org/ROS/Tutorials/CreatingPackage
4. Building a ROS Package
   http://wiki.ros.org/ROS/Tutorials/BuildingPackages
5. Understanding ROS Nodes
   http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes
6. Understanding ROS Topics
   http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

7. Understanding ROS Services and Parameters
   http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams
8. Using `rqt_console` and `roslaunch`
   http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch
9. Using `rosed` to edit files in ROS
   http://wiki.ros.org/ROS/Tutorials/UsingRosEd
10. Creating a ROS `msg` and `srv`
    http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv
11. Writing a Simple Publisher and Subscriber (`C++11`)
    http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29
12. Examining the Simple Publisher and Subscriber
    http://wiki.ros.org/ROS/Tutorials/ExaminingPublisherSubscriber
13. Writing a Simple Service and Client (`C++11`)
    http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29
14. Examining the Simple Service and Client
    http://wiki.ros.org/ROS/Tutorials/ExaminingServiceClient
15. Recording and playing back data
    http://wiki.ros.org/ROS/Tutorials/Recording%20and%20playing%20back%20data
16. Getting started with `roswtf`
    http://wiki.ros.org/ROS/Tutorials/Getting%20started%20with%20roswtf
17. Navigating the ROS wiki
    http://wiki.ros.org/ROS/Tutorials/NavigatingTheWiki
18. Where Next?
    http://wiki.ros.org/ROS/Tutorials/WhereNext

In this practical challenge we are not going to work with the *MiPal* whiteboard (the PULL-approach). However, you must read the paper "High Performance Relaying of C++11 Objects Across Processes and Logic-Labeled Finite-State Machines" [1] [2]. For your report, after completing the `ROS` tutorials, contrast what `ROS` offers and the model of a blackboard architecture (or a robotics middleware).

Second task.- A reactive behavior with the `ROS`-turtle

In the `ROS` tutorials you should have looked at *Understanding ROS Nodes*
http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

As well as *Understanding ROS Topics* http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics

At some point there are 3 programs running:

[2] link.springer.com/chapter/10.1007%2F978-3-319-11900-7_16

1. roscore which is the communication layer.
2. turtlesim as simulator of one turtle.
   ```
   rosrun turtlesim turtlesim_node
   ```
3. turtlesim as the tele-operation control.
   ```
   rosrun turtlesim turtle_teleop_key
   ```

Since you also built a catkin workspace and you compiled packages of ROS you can install clfsm to create behavior in ROS. Fist, watch the following videos:

1. www.youtube.com/watch?v=AJYA2hB4i9U&feature=youtu.be.
2. www.youtube.com/watch?v=qs-jmjxOXLI
3. www.youtube.com/watch?v=4txscEXN8lQ
4. https://www.youtube.com/watch?v=fX7ANt03Xsc

Then, follow all the steps of *How to use clfsm with ROS* [3].

The aim here is to make sure you can understand the construction with *llfsms* of the behavior where if the turtle gets too close to the wall it backs-off. The example machines are provided with the package

```
clfsmRosDemoMachines.tar.gz
```

Third task.- The basics of NXT_dirver and NXT_controller from r2d2mipal

You must have installed ROS and completed the earlier tasks. For you report, you must look at the code of testDemo.cpp in the NXT_dirver r2d2mipal and explain it. The idea is that you investigate and explain what this program does. You will needs this as your tasks is to extend the NXT_controller.

Installing NXT_dirver from r2d2mipal for Ubuntu

We recommend you use Ubuntu 14.04. Installation should be as per a standard catkin package. Obtain the package from github

```
git clone https://github.com/mipalgu/NXTdriver.git
```

Just place the package under the catkin workspace you created in the tutorials. However, because it requires libusb. You may need to install this library.

```
sudo apt-get install libusb-dev
```

[3] mipal.net.au/downloads.php

Second task

Complete the activities of the document "How to use the `NXT_dirver r2d2mipal` and the corresponding `NXT_controller`". [4] To obtain the `NXT_controller` package, clone it from `github`

```
git clone https://github.com/mipalgu/NXTcontroller.git
```

Make sure you describe the current behavior of the `motorTest.machine` in your report. The current hardware interface: `NXT_controller` does not implement tan abstraction for the sonar, for the light or for the encoders in the motor. Your tasks is to implement these sensors in two modes.

1. As a `topic`. The module (usually an *llfsm*) interested in the value of one of these sensors posts a message that turns the sensor on. From there on, the `NXT_controller` should post the values of the corresponding sensor regularly as a topic, until again, someone switches the posting off.

2. As a `service`. The module interested in the value (usually an *llfsm*) request the value as a client, and the `ROS::service` returns the value of the requested sensor.

To demonstrate this new functionality of the sensors, modify the `motorTest.machine` behavior so that if an obstacle becomes to close, then the behavior exits. Also, if the encoders reach a very high value the *llfsm* stops.

Fourth task.- Alternative with `ROS`

You are required to control with the `ROS` alternative `smach` at least one of the reactive behaviors of the earlier tasks. That is, implement and construct with `smach` one of the following behaviors:

1. the `ROS` turtle simulators walking and backing off if it gets too close to the wall backs of.

2. The `motorTest.machine` behavior with the added features that a very close objects exits the behavior as well as if a large value of the encoders is reached.

You can find the information on `smach` at wiki.ros.org/smach.

What shall you submit or demonstrate

You should submit a *Practical Challenge Report*. You should demonstrate you completed all the activities in the requested tasks. It is OK to complete the labs with your own robots if you are so inclined and acquire `LEGO-NXT` robots.

You should submit a reflective report on the experience in the lab with enough evidence that shows you have completed all the activities. You may include some screen shots. You may include links to videos. **The challenge report must be a PDF document.** Please discuss in your own words the topics covered in the challenge. That is, describe concept like

[4] Section 4 of http://mipal.net.au/downloads.php

1. robotic middleware,

2. finite-state machines, or

3. reactive architecture.

It is important to describe how would you have implemented the behavior of the last tasks (the NXT robot) if you didn't have access to `clfsm` and to a module like the `NXT_controller`. That is, what would be the structure of a program that achieves that behavior if it had evolved from the `testDemo` program of the NXT-Driver? What would be the software architecture in such case?

Some other important concepts you should investigate further are as follows.

1. The relevance of time in relation to the software controlling a robot.

2. The basics of reactive control.

3. The basic of robotic middleware.

References

[1] V. Estivill-Castro, R. Hexel, and C. Lusty. High performance relaying of C++11 objects across processes and logic-labeled finite-state machines. In D. Brugali, J. F. Broenink, T. Kroeger, and B. A. MacDonald, editors, *Simulation, Modeling, and Programming for Autonomous Robots - 4th International Conference, SIMPAR 2014*, volume 8810 of *Lecture Notes in Computer Science*, pages 182–194, Bergamo, Italy, October 20th-23rd 2014. Springer.