Assignment 0: Setup

ETH Zurich

Recommended due date: Thursday, 03.10.2013

The long wait is over. Your robot has finally arrived. The robot has many sensors and actuators, and you think about all the things you can do with this new robot. Unfortunately, you have not quite figured out how to use these work. What do you need to do to make the robot work? Can you make the robot move and respond as you desire?

Note: This is an ungraded assignment.

1 Hardware

1.1 Thymio-II

Thymio-II [1] is a small differential-drive robot with a large number of sensors and actuators, as shown in Figure 1. The robot is programmable using Aseba - an event-based architecture. The robot comes with pre-programmed behaviours, namely:

- Friendly (green) - Thymio-II follows the object in front of it.
- Explorer (yellow) - Thymio-II explores the environment while avoiding obstacles.
- Fearful (red) - Thymio-II detects shocks and free falls, and shows the direction of gravity.
- Investigator (cyan) - Thymio-II follows a trail. The trail must be at least 4 cm wide with high contrast, the best is black on white.
- Obedient (purple) - Thymio-II follows the commands from the buttons or from a remote control.
- Attentive (blue) - Thymio-II responds to sound. We can control the robot by clapping. 1 clap = go straight ahead / turn. 2 claps = run / stop. 3 claps = turns in a circle.

Play with these pre-programmed behaviours to better understand how different sensors and actuators of Thymio-II work.

1.2 PrimeSense Sensor

PrimeSense 3D sensor [2] is an RGB+D camera that gives both color and depth information of an environment. It projects infrared structured light into the environment, and by analyzing the distortion pattern of the structured light, the sensor can calculate the change of depth in the scene. In addition, it has a standard image camera to capture color information of the scene. The sensor provide an RGB+D image by registering the depth and color images.

PrimeSense will be the eye of the robot. PrimeSense Camine 1.9 datasheet contains more detailed information on the sensor technology and specification.

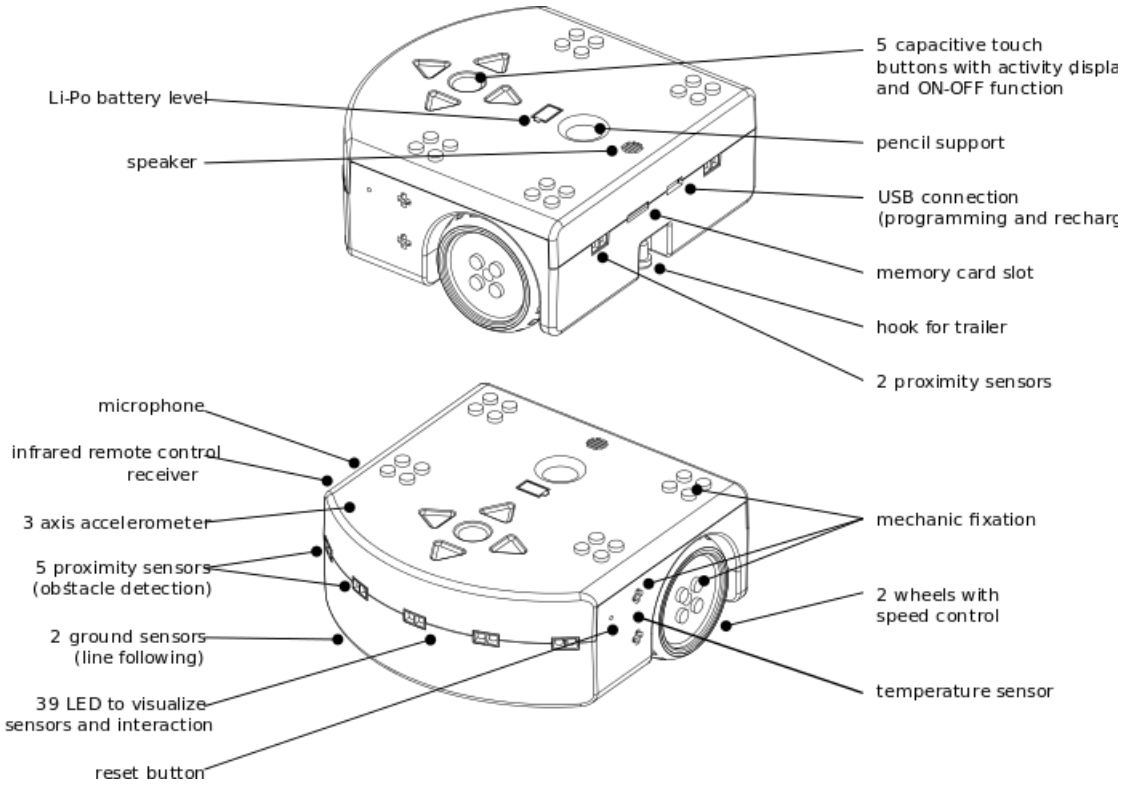

Figure 1: Sensors and actuators of Thymio II

2 Software

2.1 Operating System

Ubuntu is the officially-supported operating system of many robotics frameworks. It answers all the requirements of our class - that's why we gonna use it. For this course we recommend you to use Ubuntu 12.04 (Precise). Free distribution of Ubuntu 12.04 can be found here: http://releases.ubuntu.com/precise.

2.2 ROS

ROS (Robot Operating System) [3] provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. To install **ROS Groovy**, perform the following steps:

1. Configure your Ubuntu repositories to allow "restricted," "universe," and "multiverse" as explained in Ubuntu repository guide.

2. Set up your computer to accept software from packages.ros.org. For Ubuntu 12.04 (Precise), the command is:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise main" > /etc/apt/sources.
    list.d/ros-latest.list'
```

3. Set up your keys

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

4. Make sure your Debian package index is up-to-date then install ROS packages.

```
sudo apt-get update
sudo apt-get install ros-groovy-desktop-full
```

5. Initialize `rosdep`. `rosdep` enables easy-installation of system dependencies and is required to run some core components in ROS.

```
sudo rosdep init
rosdep update
```

6. Get `rosinstall`. `rosinstall` is a frequently used command-line tool in ROS that is distributed separately. It enables you to easily download many source trees for ROS packages with one command.To install this tool on Ubuntu, run:

```
sudo apt-get install python-rosinstall
```

7. Create ROS Workspaces. When working with ROS source code, it is often useful to do so in a "workspace". For our class you will need an area for working on and creating new ROS stacks and packages. At the moment there are two ways for creating ROS packages: using `catkin` or using `rosbuild`. We will work with both types of packages, so you have to create two different ROS Workspaces.

   - Catkin Workspace
     First, create a `catkin` workspace:

```
source /opt/ros/groovy/setup.bash
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

Even though the workspace is empty (there are no packages in the `src` folder, just a single `CMakeLists.txt` link) you can still "build" the workspace:

```
cd ~/catkin_ws/
catkin_make
```

The `catkin_make` command is a convenient tool for working with `catkin` workspaces. If you look in your current directory you should now have `build` and `devel` folders.

- Rosbuild Workspace

  In addition to the `catkin` workspace, you will need to create a workspace to work with the `rosbuild` build system. Create a new workspace in `~/groovy_workspace` and refer it to `catkin_ws`:

```
rosws init ~/groovy_workspace ~/catkin_ws/devel
```

  It can be very convenient to have a sandbox directory where packages can be put without requiring any additional rosws commands. For that we create a new directory sandbox:

```
cd ~/groovy_workspace/
mkdir sandbox
rosws set sandbox
```

  Source the `groovy_workspace/setup.bash` file to make your system work with both `groovy_workspace` and `catkin_ws`:

```
echo 'source ~/groovy_workspace/setup.bash' >> ~/.bashrc
source ~/.bashrc
```

- Confirmation

  To confirm that your package path has been set, echo the `ROS_PACKAGE_PATH` variable:

```
echo $ROS_PACKAGE_PATH
```

  You should see something similar to:

```
/home/your_user_name/catkin_ws/src:/opt/ros/groovy/share:/opt/ros/groovy/stacks:/
    home/your_user_name/groovy_workspace/sandbox
```

For more details on ROS installation, read ROS Groovy installation guide for Ubuntu.

2.3 Asebaros

`asebaros` is a bridge between ROS and Thymio-II robot. It allows to load scripts onto the robot, inspect the network structure, read and write variables, and send and receive events from ROS. It maps the robot's events to ROS topics in a dynamic way, when loading source code.

`asebaros` is implemented as a ROS package which uses `rosbuild`, so you need to install this package inside your `sandbox`.The following commands will fetch and compile the `asebaros` package.

First, navigate to your sandbox directory and download the package:

```
cd ~/groovy_workspace/sandbox
git clone --recursive git://github.com/ethz-asl/ros-aseba.git ethzasl_aseba
```

Then install ROS dependencies and finally build `asebaros`:

```
rosdep install asebaros
rosmake asebaros
```

To make the integration with your future packages easier add the location of `asebaros` into your `.bashrc` file by executing:

```
echo 'export ROBOLAB_ASEBA_PATH=~/groovy_workspace/sandbox/ethzasl_aseba' >> ~/.bashrc
source ~/.bashrc
```

When installation is finished you can try to connect to the Thymio-II robot. First, run `roscore` if it is not running already:

```
roscore
```

Then, open a new terminal and run `asebaros`:

```
rosrun asebaros asebaros "ser:device=/dev/ttyACM0;fc=hard;baud=921600"
```

This connects Aseba to the serial port `/dev/ttyACM0` with hardware flow control and a speed of 921.6 kbps. If your connection is successful, you'll get a message like this:

```
[ INFO] [1379317031.204330148]: Incoming connection from ser:device=/dev/ttyACM0;baud=921600;stop=1;parity=none;fc=hard;bits=8
```

If you cannot connect to Thymio-II, make sure you have specified the correct port name. For example, you may need to connect to `ttyUSB0` instead of `ttyACM0`.

You may also face the connection problem because of your access rights. Ensure that you are in the `dialout` group:

```
groups your_user_name
```

You should see `dialout` as one of the groups. If not, then you need to add yourself to the `dialout` group:

```
sudo adduser your_user_name dialout
```

Log out then log in again, then try to connect to Thymio-II again.

2.4 OpenNI SDK

OpenNI is an open-source framework for "natural interaction" - using your hands and body to interact with your digital devices. It provides the drivers for PrimeSense sensor and additional libraries, which we are going to use during the course. You can download it here: http://www.openni.org/openni-sdk. To install the SDK first unzip and untar the downloaded file into your home directory:

```
cd ~/Downloads
unzip OpenNI-Linux-x64-2.2.tar.zip
cd ~/
tar xvfj ~/Downloads/OpenNI-Linux-x64-2.2.tar.bz2
```

Then install `OpenNI` by executing:

```
cd ~/OpenNI-Linux-x64-2.2
sudo ./ install .sh
```

Add information about your location of the OpenNI SDK to the `.bashrc` file:

```
echo 'export ROBOLAB_OPENNI_PATH=~/OpenNI-Linux-x64-2.2' >> ~/.bashrc
source ~/.bashrc
```

`Important!` Use only **USB 2.0** port on your machine to connect PrimeSense. The sensor is incompatible with USB 3.0.

2.5 EiffelStudio

EiffelStudio [4] is an interactive development environment, specifically crafted for the Eiffel method and language. Download a free version of **EiffelStudio 7.3** for Linux from SourceForge: http://sourceforge.net/projects/eiffelstudio/files. To install **EiffelStudio 7.3**, perform the following steps:

1. Extract the contents of the archive you just downloaded into your working directory (home directory in this case).

   ```
   cd ~/
   tar xvfj ~/Downloads/Eiffel73-your-version-here.tar.bz2
   ```

2. Add the following environment variables to your `.bashrc` file.

   ```
   echo 'export ISE_EIFFEL=~/Eiffel73' >> ~/.bashrc
   echo 'export ISE_PLATFORM=linux-x86-64' >> ~/.bashrc
   echo 'export PATH=$PATH:$ISE_EIFFEL/studio/spec/$ISE_PLATFORM/bin' >> ~/.
       bashrc
   source ~/.bashrc
   ```

   If your version is `linux-x86`, then `ISE_PLATFORM` should be set to `linux-x86` instead of `linux-x86-64`.

3. Check the installation by opening the new terminal and launching Eiffel Studio.

   ```
   ec -gui
   ```

For more details on Eiffel Studio installation, read Eiffel Studio installation guide for Linux.

2.6 Roboscoop

Get the source code for Roboscoop from Dropbox. The link will be emailed to you on Monday, 23 of September. The source code of Roboscoop is split into three parts: ROS package `roboscoop_ros`, which is responsible for external communication and external execution (robot's software, reused ROS functionality), Eiffel project of Roboscoop library (`roboscoop_lib` - top level of Roboscoop) and Eiffel project `roboscoop_app` where you can create your applications.

1. Download the `roboscoop` folder from the Dropbox link then move the content to `catkin` workspace.

   ```
   mv ~/Download/roboscoop ~/catkin_ws/src
   ```

2. Set the following environment variables. These variables enable you to use external libraries in the Eiffel project.

```
echo 'export ROS_LOCATION=/opt/ros/groovy' >> ~/.bashrc
echo 'export CATKIN_WORKSPACE=/home/your_user_name/catkin_ws' >> ~/.bashrc
source ~/.bashrc
```

3. Download the `aseba` folder from the Dropbox. In the `aseba` folder, you will find `thymio_roboscoop.aesl`, `aseba_thymio.launch` and `roboscoop_thymio_navigation_driver.py`. Move these files to the `thymio` package:

```
cd $ROBOLAB_ASEBA_PATH/thymio_navigation_driver/
mv ~/Downloads/aseba/thymio_roboscoop.aesl aseba/
mv ~/Downloads/aseba/aseba_thymio.launch launch/
mv ~/Downloads/aseba/roboscoop_thymio_navigation_driver.py nodes/
mv ~/Downloads/aseba/thymio_constants.py nodes/
```

Since the Roboscoop project consists of several parts, namely, ROS package `roboscoop_ros` and Eiffel library and project `roboscoop_lib` and `roboscoop_app`, the compilation of `roboscoop_ros` must be executed before the Eiffel compilation. The scripts necessary for Roboscoop compilation are located in the `roboscoop` folder.

1. First, build `roboscoop_ros` together with the other ROS packages:

```
cd $CATKIN_WORKSPACE
catkin_make
```

2. Then, generate necessary C++ code for the Eiffel project:

```
cd $CATKIN_WORKSPACE/src/roboscoop/scripts
./gen_callbacks.sh
```

Successful generation should output something like this:

```
...
Topics found: 1
Successfully generated file : /home/your_user_name/catkin/...
```

3. Compile the Eiffel project:

```
./ build_eiffel .sh
```

If you get asked *"Should the precompile be built? [y/n]"*, then type *y*.

4. You can now launch the project by running

```
roslaunch thymio_navigation_driver aseba_thymio.launch
```

Run Eiffel in the new terminal:

```
./ run_eiffel .sh
```

2.7 SVN Repository

We recommend you to create an SVN repository for the course. VIS provides code and project hosting service at `https://code.vis.ethz.ch/`, and you can access it using your ETH account. Version control allows you to keep track of changes

1. Go to `https://code.vis.ethz.ch/` and log in using your ETH account.
2. Under **Create a new project**, type in the name of new project as `your_user_name_rpl2013` and press *create*. For example, if your ETH username is `foo`, then your project name should be `foo_rpl2013`. This will create a new project and direct you to the next page.
3. Under **Project "your_user_name_rpl2013"**, you will see **Create Repository**. Click on *svn* and give the svn repository the same name, i.e., `your_user_name_rpl2013`.
4. At the end of each assignment, we will collect your work from your repository. For this, you need to give us read access to your account. Click on *add/modify* under **Project Access Permissions**. Type in `jshin` as the username and mark *read*. Press *Set Permissions*. *Ji Won Shin* should appear as a user with *read* access right. Repeat the procedure and grant `arusakov` read access to your repository as well. At the end of this you will see three users to your project: yourself, *Ji Won Shin* and *Andrey Rusakov*.
5. Now that the repository is created, you can check it out on your computer. To check out your repository to `catkin_ws/src`, open a terminal and type:

```
svn co https://code.vis.ethz.ch/svn/your_user_name_rpl2013 $CATKIN_WORKSPACE/src
```

6. Add the roboscoop directory to the repository

```
cd $CATKIN_WORKSPACE/src
svn add roboscoop
```

3 Tutorials

At the end of these tutorials, you should be able to communicate with Thymio II using Roboscoop. In particular, you should be able to write code to activate and receive information from all sensors and actuators of Thymio II.

3.1 Ubuntu

If you have never used a Linux operating system, get yourself familiar with the basics of Linux. Internet has numerous sites and videos that explain Linux operating system in general and Ubuntu in particular. Some recommended introductions include the official Ubuntu documentation and "Introduction to Linux" by Garrels [6].

3.2 ROS

Read the ROS tutorials [7]. Follow the beginner level tutorials and perform the suggested exercises. By the end of the tutorials, you should understand the difference between topics and services and be able to write a publisher and subscriber node and a service and client node.

TF and RViz packages are useful packages for this class. Follow the tutorials for these packages and learn how they work.

3.3 Eiffel and SCOOP

Eiffel is an object-oriented language with design by contract and is the base language for SCOOP (Simple Concurrent Object-Oriented Programming), on which Roboscoop builds. There is a list of resources for learning Eiffel. In particular, an online tutorial of Eiffel is available here.

The SCOOP concurrency model seeks to remove the gap between the object-oriented concepts and the techniques used for handling the multithreaded or concurrent parts. It brings to the world of concurrency the same systematic O-O development techniques that have made their mark in the sequential world. Concurrent Eiffel with SCOOP contains a short description of SCOOP, and "Practical framework for contract-based concurrent object-oriented programming" by Nienaltowski [8] contains a more thorough description.

3.4 Aseba and asebaros

Aseba [5] is a set of tools used to program Thymio II. Aseba is an event-based architecture for real-time distributed control of mobile robots and targets integrated multi-processor robots or groups of single-processor units. The Aseba language is described in `https://aseba.wikidot.com/en:asebalanguage`, and the programming interface is found in `https://aseba.wikidot.com/en:thymioapi`.

Write a program in Roboscoop that can receive information from various sensors and activate various actuators of Thymio II and ensure that every sensor and actuator is working.

3.5 SVN

Subversion [9] is an open-source version control system, which allows you to keep old versions of files and directories (usually source code), keep a log of who, when, and why changes occurred, etc. Subversion keeps a single copy of the master sources, called the source "repository". The source repository contains all the information to permit extraction of previous versions of its files at any time.

Some basic commands of SVN are as follows:

- Update: "Update" brings changes from the repository into the working copy.

```
svn update
```

- Commit: "Commit" sends changes from your working copy to the repository.

```
svn commit -m "... commit message ..."
```

- Add: "Add" puts files and directories under version control, scheduling them for addition to the repository. These files and directories will be added in next commit.

```
svn add PATH...
```

- Delete: "Delete" removes files and directories from version control.

```
svn delete PATH...
```

- Help: "Help" brings up a list of commands.

```
svn help
```

In general, you will `svn update` to get the latest version controlled files and `svn commit` to commit your latest changes.

When you create a new file, you must `svn add` the file for the file to be under version control. Any file that is in your local svn directory but is not version controlled cannot be viewed by other users of the repository. Likewise, any file that is locally deleted but not `svn delete`'d is still viewable by other users. Do not forget to svn add files that you want us to see and delete files that you do not want to be part of your assignment submission.

References

[1] http://www.thymio.org

[2] http://www.primesense.com

[3] http://www.ros.org

[4] http://docs.eiffel.com

[5] https://aseba.wikidot.com

[6] Garrels, M. *Introduction to Linux*. Fultus Corporation. 2010.

[7] http://www.ros.org/wiki/ROS/Tutorials

[8] Nienaltowski, Piotr. *Practical framework for contract-based concurrent object-oriented programming*. PhD Dissertation, ETH Zurich, 2007.

[9] http://subversion.apache.org/