Getting started with the E-puck

Johan S. Laursen

December 19, 2016

Contents

1 Introduction

This document explains how to get started with remote controlling the e-puck using C++ and the ROS framework. An *e-puck user manual* from AAI Canada is available in the lab. Their manual is a bit dated but gives a detailed introduction to programming the e-puck directly. This document mostly replaces *section 4: Getting Started* of the AAI manual and supplements with updated information and more details regarding how to control the e-puck remotely. The manuals accompanying CD contains useful guides, programs, and c-drivers for the e-puck.

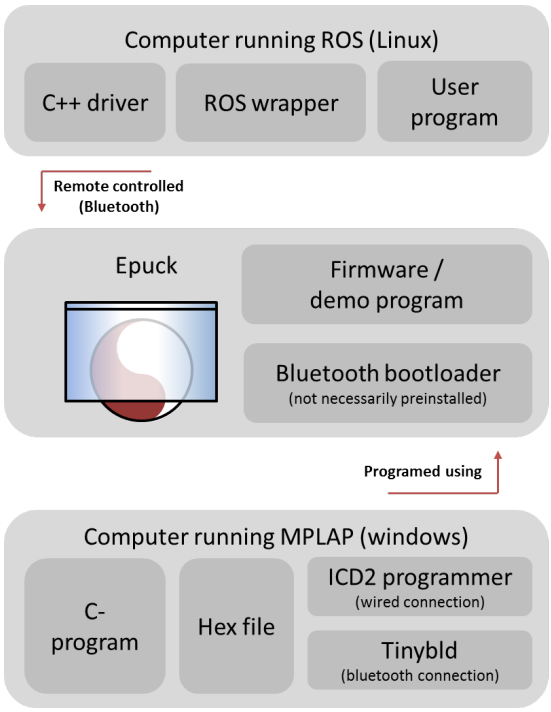

Figure 1: Pipeline and overview

Software currently installed on the e-puck

In addition to the information regarding the epuck driver this document also gives instructions on how to install both the bluetooth bootloader and the firmware onto the e-puck. This may already be installed onto the epuck depending on how it has been used privously. Likewise the demo program descripbe in AAI canadas manual may not be present on the epuck.

The current state of the driver

A C++ class is used to gain access to the e-pucks sensors and actuators raw values. This is wrapped in ROS for convenience and ROS compliance. This wrapping converts some values into physical units instead of the raw number obtain directly from the e-puck. For instance, using the ROS wrapper speed is given as a linear speed [m/s] and angular speed [rad/s] as opposed to raw integer values. Note that this driver provides a low level interface to the e-puck and does not provide high-level functionality such a tracking, wall-following, point-to-point navigation, etc. Implementation of such can be seen in the reversible e-puck project linked in Sec. 5. The code is however not suitable for direct use (as it is only a prototype) but may provide a place for inspiration.

2 Programming the e-puck

2.1 Installing and running MPLAP

Inside the epuck:

- Inside the e-puck is a dsPIC30F6014A chipset.
- This chip is part of 16 bit architecture
- This chipset is developed by the company Microchip

Installing the correct Microchip tools

To transfer compiled programs via cabled to the e-puck requires a proprietary programmer from Microchip and the accompanying toolsuite. Programmers of type `MPLAB ICD 2` is available in the lab. These programmers are however no longer supported in newer versions of Microchips programming IDE (MPLAP) and as such older versions of the IDE has to be used.

The software required to program the e-puck is separated into three parts.

- IDE: MPLAP IDE v8.70
  (Not to be confused with MPLAP-X which is the newest version)
- Compiler and chipset specification: MPLAP C30 v3.30b
  (Found under MPLAB C Compiler for PIC24 and dsPIC DSCs)

Running MPLAP:

MPLAP was originally written for Windows XP. When running the program enable windows backwards compatibility feature and run the program as an administrator. This is selected by right clicking the icon→properties→compatability. If the ICD 2 programmer is connected via USB the correct drivers has to be installed (the driver installed automatically by windows does not work). [1]

Install the correct ICD 2 USB driver:

To check the correct driver is installed click:

```
start -> Computer (right click) -> properties -> device manger
```

The driver vertion should be listed as:

```
Microchip Technology, Inc.
19-12-2007
1.0.0.6
```

To update the driver click:

```
Driver tap -> Update driver -> Browse manually -> let me pick from list ->
```

[1] Tested September 2016 using Windows 7 Service Pack 1.

```
Have disk -> Browse -> <MPLAP IDE folder>/driver64/
```

you might have to restart MPLAP and reconnet the programmer after driver changes. Also note that the driver used might change depending on which USB port is used.

Install the correct ICD 2 USB driver:

If a RS232 connection is used to interface with the programmer (using a serial calbel, not the USB cable) the settings for the COM-port has to be modified. The COM port settings can be edited from the device manger. Follow the instructions in the link and ajust the COM port settings:
http://ww1.microchip.com/downloads/en/devicedoc/51265e.pdf

COM Port settings:

- Use COM port 1 (MPLAP has truble with higher port numbers)
- Set baudrate to 11920
- Disable buffer
- Set flow to hardware

Common MPLAP errors:

Robot is not connected to programmer:

```
ICDWarn0020:  Invalid target device id (expected=0x2C3, read=0x0)
...Reading ICD Product ID
Running ICD Self Test
...  Failed Self Test.
```

Incorrect USB driver is used for the ICD 2:

```
Connecting to MPLAB ICD 2
ICD0019:  Communications:  Failed to open port (USB): (Windows::GetLastError()
= 0x0, 'The operation completed successfully.')
ICD0021:  Unable to connect with MPLAB ICD 2 (USB)
MPLAB ICD 2 ready for next operation
```

2.2 Transferring programs via cable

To install an already compiled program (hex file) on to the e-puck

1. Open the MPLAP IDE
2. Import the Hex file:
   file →import →file.hex

3. Connect the programmer:
   programmer→select programmer→MPLAP ICD 2

4. Transfer the program:
   programmer→Program

2.3 Transferring programs via Bluetooth

2.3.1 Installing the bootloader

The e-puck does not come with a Bluetooth bootloader preinstalled. However having the bootloader a huge convenience compared to continually unplugging and inserting the programmer. Too install the bootloader:

1. Download the e-puck software package linked in Sec. 5 or located on the e-puck support CD

2. Locate the compiled bootloader HEX program
   `/tool/bootloader/epuck_side/tinybld_ds6014A_7.37Mhz_115200uart1_8xPLL_with_LEDs.hex`

3. Transfer the HEX file to the e-puck using cabled approach described in Sec. 2.2.

2.3.2 using the bootloader in windows

1. Install tinyBld linked in Sec. 5

2. Browse to the hex file, select the correct COM port and choose a baud rate of 19200Hz.

3. Click write and hit the reset button on the e-puck

Tip: *If two new com ports are created during the Bluetooth pairing; the correct port is probably the one with the lowest index of the two.*

2.3.3 using the bootloader in unix

The software package (linked in 5.0.3) contains almost the same content as the CD accompanying the manual. The web-based package also includes small text-based programs which can be used to transfer hex files to the e-puck.

3 Bluetooth connection to the e-puck

Windows

In windows this is straight forward.

1. Use the normal Bluetooth paring process.
2. Select paring based on manually entering the devices paring cod e
3. Use the four digits found in the e-pucks Bluetooth name on the chip beneath the e-pucks extension board

Linux

In Linux paring (for some reason) cannot be done using the normal approach and has to be done using the terminal. The CD contains both programs and instructions on how to do the paring. Alternatively, the approach described in the links section (Sec. 5) can be used.

4 Remote controlling the e-puck

To read and control the e-pucks sensors and actuators remotely requires both an e-puck and computer to implementation and run the same interface. Subsection 4.1 deals with the driver on the e-puck side while the remaining subsection goes into more details regarding the computer side.

4.1 The e-puck firmeware

Every single firmware implementation, interfaces, or programs for reading and controlling the e-puck remotely seems to rely the same interface running on the e-puck. The interface running on the e-puck is a simple text based UART interface. Multiple implementations of the same interface exist, and as a result there is a lot of precompiled versions and variants of the need firmware can be found ready to be loaded onto the e-puck.

A version is located on the CD or inside the git-repository.
`/Software/program/BTcom/BTcom.hex`

The most stable version of this firmware is developed by Cyberbotics for their Webot platform. The can be found by downloading the trial version of webot in the folder:
`/Webots/projects/robots/e-puck/transfer/firmware/firmware-1.5.2.hex`

To install the firmware running on the e-puck side:

- Locate the hex binary.
- Transfer the binaries to the e-puck using one of the methods described previously

To confirm the firmware is correctly installed either:

- Use tinybtl or a similar program and attempt to read and control the robot. (Send 'H' to get menu of commands)
- Use the test prorgram from the CD or downloaded from 5.0.3
  `/Software/tool/e-puck_monitor/EpuckMonitor.exe`

4.2 The computer driver

The driver on the computer side is an extension and refactoring of the driver ROS C++ driver created by GCtronics. The driver adds support for controlling the leds on the e-puck using ROS topics and refactors the code into smaller more manageable pieces making it fairly easy to use the driver without ROS.

4.3 Installing and compiling the driver

The framework is based on ROS Indigo and uses Linux Bluetooth protocol stack BlueZ.

To install the framework

1. Ensure ROS is installed
2. Create or navigate to your catkin_workspace
3. Checkout the git repository (link 5.0.4) into the workspace
4. Compile the workspace using `catkin_make`

4.4 Launching the driver node

To lauch the driver:

1. Find the e-puck number (see AAI manual page 30/31)
2. Find the e-puck mac-address (type `hcitool scan` in terminal)
3. Modify the launch file accordingly `/src/epuck_driver_my/launch/epuck.launch`
4. Launch the node `roslaunch epuck_driver epuck.launch`

In addition to the e-puck name and address the launch file also contains several parameters which enables/disables specific sensors and configures the e-puck camera.

To demo the e-puck driver, create a new terminal and type:

- To run the interface demo node:
  `rosrun epuck_interface_demo epuck_interface_demo`
- To view the e-puck camera:
  `rosrun image_view image_view image:=/camera`
- To launch a ros turtle_sim_control (manual remote control):
  `roslaunch epuck_interface_demo teleop.launch`
- To launch Rviz and Gmapping:
  `roslaunch epuck_driver gmapping.launch`

4.5 Overview of the driver

The driver is split into three classes

- `BluetoothConnection`: Sets up a UART connection the e-puck
- `BasicCppDriver`: Gives a c++ interface to control the robot
- `RosWrapper`: Wraps the c++ in a ROS allowing it to run as its own node

Inside the GIT-repository hosting the driver is also a package which showcases how it is possible to connect and interface the driver using ROS.

4.6 The interface

Communication whith the e-puck is entirely based on ROS topics. The following commands can be useful to obtain information regarding these topics.

- `rostopic list`
- `rostopic echo <topic name>`
- `rostopic info <topic name>`

Outputs topics

- **proximity:**
  Topic name: `/proximityN`
  Messeage type: `sensor_msgs::Range`
- **Accelerometer:**
  Topic name: `/accel`
  Messeage type: `sensor_msgs::Imu`
- **Light:**
  Topic name: `/light`
  Messeage type: `visualization_msgs::Marker`
- **Motor speed:**
  Topic name: `/motor_speed`
  Messeage type: `visualization_msgs::Marker`
- **Selector:**
  Topic name: `/selector`
  Messeage type: `visualization_msgs::Marker`
- **Microphone:**
  Topic name: `/Microphone`
  Messeage type: `visualization_msgs::Marker`

Information not directly based on robot sensors
(manily used for integration with other ROS packages):

- **Odomery (calculated based on motor speed):**
  Topic name: `/odom`
  Messeage type: `nav_msgs/Odometry`
- **Robot transforms:**
  Topic name: `/tf`
  Messeage type: `tf2_msgs/TFMessage`
- **Laser scan (based on proximity sensor):**
  Topic name: `/scan`
  Messeage type: `sensor_msgs/LaserScan`

Inputs topics

- **Motor control**
  Topic name: `/mobile_base/cmd_vel`
  Messeage type: `geometry_msgs::Twist`
  Format: `Linear speed (msg.linear.x) and angular speed (msg.angular.z)`

- **Led:**
  Topic name: `/leds`
  Messeage type: `std_msgs::UInt32MultiArray`
  Format: `Values: 0=off, 1=on, 2=toggle,`
  `list[10]={led 0, led 1,...,led 7, body led, front led}`

5 Link collection

5.0.1 Genneral information regarding the e-puck:

- Official e-puck website:
  http://www.e-puck.org/
- GCtronis wiki
  http://www.gctronic.com/doc/index.php/E-Puck

5.0.2 Microchip - Tools for programming the e-pucks chipset:

- IDE and compiler:
  http://www.microchip.com/mplab/mplab-x-ide
- Chipset:
  http://www.microchip.com/wwwproducts/en/dsPIC30F6014A

5.0.3 E-puck software

- E-puck software
  (E-puck C drivers, Bluetooth bootloader, Linux BT bootloader scrips, etc.)
  https://gna.org/svn/?group=e-puck
- TinyBootloader
  (GUI program for uploading programs using Bluetooth)
  http://www.etc.ugal.ro/cchiculita/software/tinyblddownload.htm
- Webot
  (Includes the e-puck firmware used for remote-controlling the robot)
  https://www.cyberbotics.com/

5.0.4 Drivers for remote control of the e-puck:

- ROS driver based on C++
  (An extension and refactored version of the GCtronics driver linked below)
  https://github.com/skorbiz/epuck_driver

5.0.5 Other

- Info on Bluetooth paring in Linux
  http://www.heatxsink.com/entry/how-to-pair-a-bluetooth-device-from-command-line-on-linux
- Reversible e-puck project
  (Inspiration for the implementation of high-level epuck behviours?)
  https://github.com/skorbiz/reversible_epuck

5.0.6 Alternative drivers for remote control of the e-puck:

- Python driver
  http://abitworld.com/projects/epuck-robot-libraries-for-teleoperation/
- ROS driver (python based)
  https://github.com/verlab-ros-pkg/epuck_driver
- ROS driver (Python based, extension of the Verlab driver above)
  https://github.com/gctronic/epuck_driver
- ROS driver (C++ based, converted version of the GCtronics driver above)
  https://github.com/gctronic/epuck_driver_cpp