STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

LORRIS TOOLBOX
Set of tools for developement and control of robots

Vojtěch Boček

Brno 2013

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor SOČ: 18. Informatika

LORRIS TOOLBOX
Set of tools for developement and control of robots

Author: **Vojtěch Boček**

School: **SPŠ a VOŠ technická,**
Sokolská 1, 602 00 Brno

Consultant: **Jakub Streit**

Brno 2013

Acknowledgement

Thanks to Jakub Streit for his advices, help, and much patience he provided during my work on this project, to Martin Vejnár for his Shupito programmer, to Mgr. Miroslav Burda for great help with text part of this work and last but not least, to Bc. Martin Fouček for his advices and help with Qt Framework. Thanks goes also to DDM Junior for their support.

This work was made with financial support from JMK a JCMM.

Annotation

This work describes a complex set of tools designed for developement and control of any device capable of connecting to serial port or TCP socket.

Because Lorris isn't a simple application nor is it a toolbox focused on one narrow area of use, because the whole set is continuously growing and because scope of use for all features and modules of Lorris is too big, it is impossible to briefly describe it all in limited scope of annotation.

To get better notion of what can one achieve with Lorris, please refer to introductory chapter of this work (English version of this text is available on enclosed CD as PDF file).

Main asset of this software package is the ability to significantly speed-up and simplify developement and testing of various applications for microcontrollers, typically programming and controlling various kinds of robots.

Key words: binary data analysis, programming and control of robots, developement for microcontrollers, programming of microchips

Contents

Introduction

Lorris is extensive set of tools which all share the same goal – to help with development, debugging and control of electronic devices, mainly robots. This chapter briefly describes the most important features of all modules and each module is throughly described in it's own chapter further on.

Module: analyzer

- Its main purpose is to graphically display data from the device.
- Analyzer uses widgets to display data – small "windows", each showing certain part of data.
- Each widget has individual settings and the user can place them anywhere on the workspace.
- Thanks to widgets, it is possible to assemble interface suitable for virtually any device.
- Several types of widgets are available in Lorris, for example *Number, Color, Column bar, Circle* (displaying angle within circle) or *Graph*.
- Analyzer is also ideal for easy displaying of data from components for which using only numbers is not eligible, e.g. color sensor.
- Some widgets can also send data to the device. Consequently, it means that beside displaying data, widgets can also control the device.
- Of all the widget types, *Script* is the most notable one. The user writes his own script, which processes the incoming data. User's script can use other widgets and other parts of Lorris, which means that it can display or in other ways interpret virtually any data.
- Using script, the user can modify the behavior of Lorris itself.

Module: programmer

- Graphical interface for several types of bootloaders and programmers of microchips.
- It can write program to chip, read and erase chip's memory or program chip's fuses.
- Official GUI for Shupito programmer.
- Shupito is microchip programmer. One end goes to computer, the other one to the chip – you need programmer to program some types of chips.

Module: terminal

- Regular terminal – displays incoming data either as text or as hexadecimal byte dump.

Module: proxy between serial port and TCP socket

- Proxy creates server connected to serial port. The serial port is then accessible from anywhere via the internet.
- It makes it possible to debug, control or otherwise communicate with the device via internet network.

Enclosed CD contains promotional poster as PDF file.

1 Motivation

I'm a member of one of the teams which build robots for various competitions, and I've met a problem while we were building one of our robots – such robot usually contains a pretty large number of various sensors (ultrasound range meters, encoders for measuring covered distance, buttons which detect collision with borders of the game field ...), and there was no way to show data from those sensors comfortably and clearly.

In order to simplify and speed-up development of the robot, I decided to find some computer application which would show data from the robot in clear and well-arranged manner. Requirements which I set for this application are specified in chapter 1.1.

1.1 Requirements for application

I require following features from the application:

1. Ability to process data from device and show them clearly
2. Support for many formats of incoming data
3. Quick and simple to use
4. Support for other operating systems than MS Windows
5. Low price
6. Ability to easily expand program, ideally open-source
7. No dependencies on other applications (eg. MS Office Excel)

1.2 Present applications

I've found only several programs which have at least similiar function (reading data from serial port and displaying them). Basically only two types of applications are available – commercial, which cost a lot of money (and still

don't meet all the requirements) or applications which can display data in only one format, typically in graph.

- **SerialChart**[1] is open-source program[1] which can parse and display data from serial port. SerialChart is simple and well arranged, but it can display data only in graph and it is configured by hand-written configuration file.

- **WinWedge**[2] is commercial program. It can process data from serial port and display them as a graph in MS Excel or as a web page. It can also send commands back to connected device, but it has bad user interface and the need for another program (like MS Excel) to actually show the data is not ideal. It is available only for MS Windows and basic version costs $ 259.

- **Advanced Serial Data Logger**[3] is designed to be used primarily to collect data from serial port and export them, thus you have to use another application to display the data (eg. MS Excel), similarly to WinWedge.

- **StampPlot Pro**[4] can process incoming data in widgets created by user, but it is not simple to use, it is not open-source, it is available only for MS Windows and I haven't managed to get it working under Windows 7.

- **LabVIEW**[5] is very large software package with long history and is is possible to use it for wide variety of operations – various laboratory measurements, analysis of data signals, controlling of robots, controlling of whole systems for laboratory measurements and much more. However, LabVIEW is closed proprietary software and because of its specialization, it is also very expensive. Basic license costs about $ 1150.

[1] Program with publicly available source code, free to modify and use

1.3 Comparison of applications

Following table lists features of each application. Numbering of requirements matches the list in chapter "Requirements for application".

Requirements	1	2	3	4	5	6	7
SerialChart	✔	✗	✔	✗	✔	✔	✔
WinWedge	✗	✔	✔	✗	✗	✗	✗
Advanced Serial Data Logger	✗	✔	✔	✗	✗	✗	✗
StampPlot Pro	✔	✔	✗	✗	✔	✗	✔
LabVIEW	✔	✔	✗	✔	✗	✗	✔

I've decided to write my own program which will meet all the requirements because no such application exists.

2 Lorris

Lorris is a program written in C++ with use of Qt Framework[6]. Qt is multiplatform framework, which (among other things) makes it possible to run Lorris on multiple operating systems – I'm using Debian Linux[7] (Wheezy, 64bit) and Windows 7 for testing.

2.1 Website and repository

Lorris' GIT[2] repository is hosted on GitHub[8]. GitHub also provides hosting for project's website, which contains links to prebuilt Lorris binaries for Windows, description of program, video introduction to Lorris (6 min.), screenshots of Lorris and information how to build Lorris under MS Windows and Linux.

- Repository: https://github.com/Tasssadar/Lorris

[2] *GIT* – distributed version control system

- Website (Czech):
  http://tasssadar.github.com/Lorris/cz/index.html
- Website (English):
  http://tasssadar.github.com/Lorris/index.html
- SOČ presentation:
  http://www.sokolska.cz/soc-2012/bocek-vojtech-lorris-sada-nastroju-pro-robotiku/

There is still an ongoing developement in application's repository.

2.2 Application's structure

The program is designed as modular application, so that it can accommodate several parts which, although they are separate, share the same area of use. Base part of application provides connection to device (e.g. to robot or to development board with chip), tab-based user interface and storage for application settings, but data processing itself takes place in individual modules.

Modules are opened as tabs, much alike pages in web browser. Lorris can open several windows at once and it can split each window to multiple parts like presented in image 2 – window is divided in the middle, you can see two tabs at once. The one on the left is analyzer and the other one is terminal.

Connection options:

- Serial port
- Shupito Tunel (virtual serial port, more in chapter 5.1.1)
- TCP socket[3]
- Loading data from file

[3] *Transmission Control Protocol* – connection via internet.

It is possible to connect multiple modules to one device.

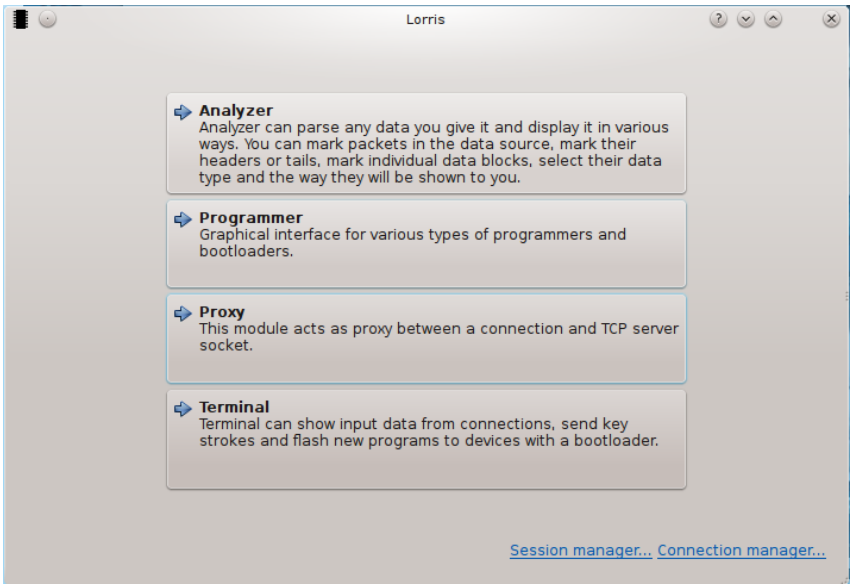


Figure 1: Tab creation dialog

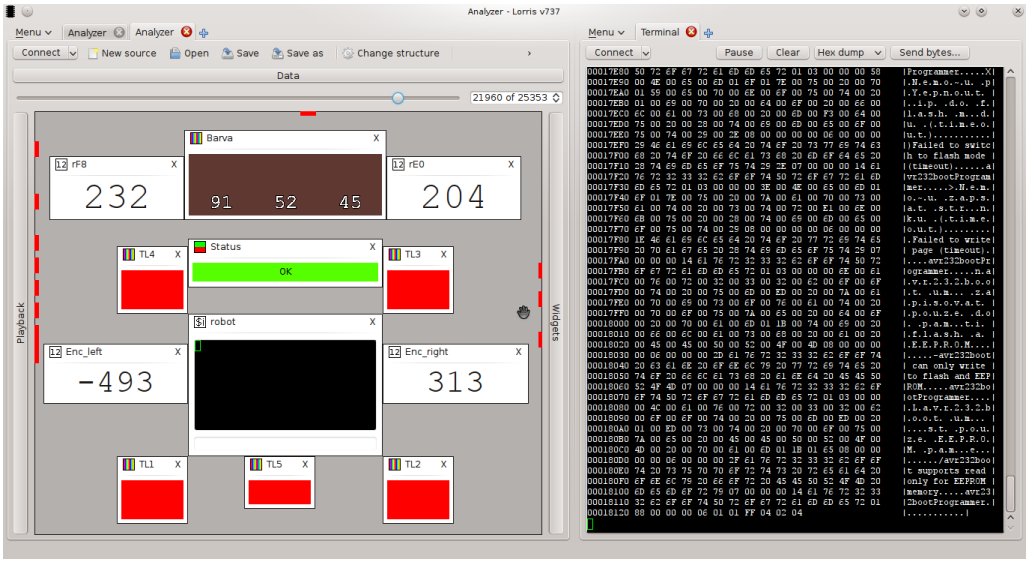


Figure 2: Window divided to multiple parts

2.3 Session

Lorris can save everything user opened (tabs, their layout, connection, data of each tab, ...) as session. User can later load saved session and thus return to his previous work. Lorris automatically saves session before it is closed, so when user starts Lorris again, all his work is in the same state as it was before he left.

2.4 Automatic updates

Lorris can update itself under MS Windows. It checks for new version on start, and if there is one available, it shows little notification:

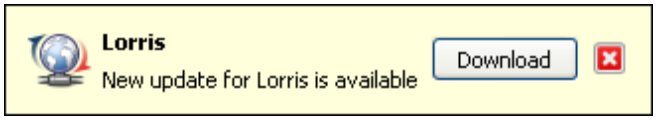


Figure 3: New update notification

In case user confirms the update, Lorris closes itself and runs little updater application. Updater shows changelog and downloads new version and installs it.

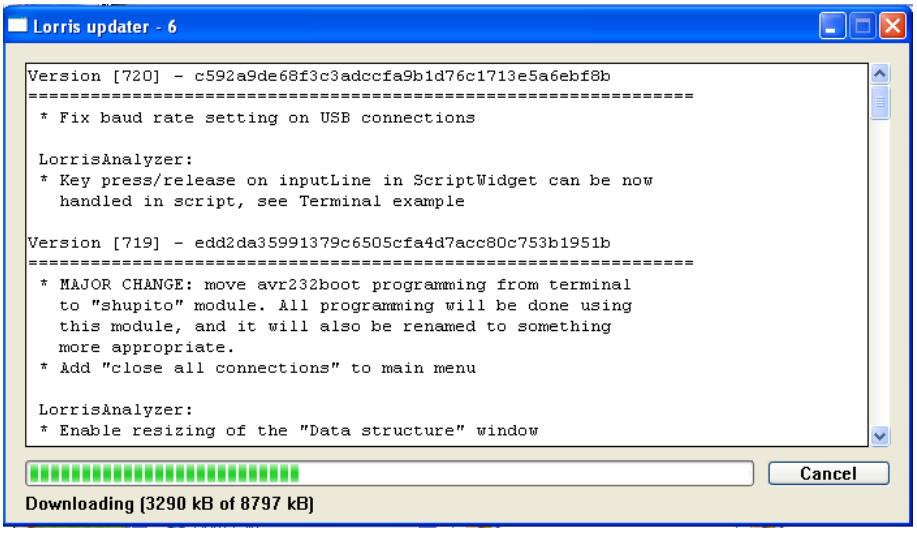


Figure 4: Ongoing update

3 Module: Analyzer

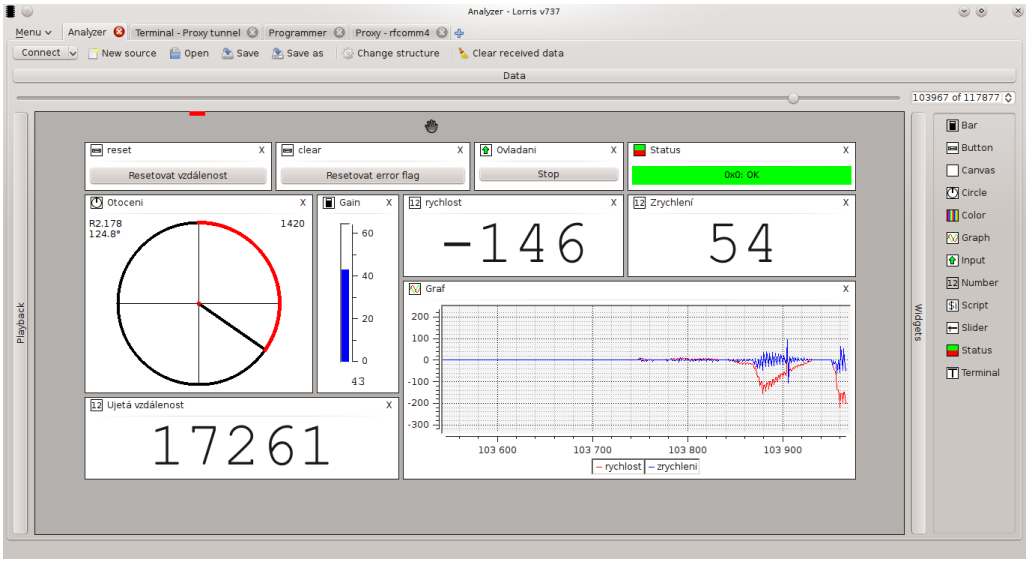


Figure 5: Module analyzer

This module parses incoming data (structured as packets) and displays them in graphical widgets. Application saves processed data into memory – user can go through received packets using slider and textbox in upper part of the window. All data (received packets, packet structure and widgets positions and settings) can be saved to file.

Packet structure is configured in dialog window (image 7). It is possible to set packet's length, endianness[4], packet's header and its content – static data ("start byte"), dynamic lenght od packet and command and device ID. Packets can be later filtered by command or device ID.

[4] *Endianness* – order of bytes in numbers

Incoming data show up in upper part of the window when packet structure is set and user can then "drag" widgets from the list in right part of the window to workspace. Data are assigned to widget again using drag&drop, this time user has to drag first byte of data to widget.

Widget then displays data from that byte (or several bytes if needed). Assigned byte is highlighted when user puts mouse over the widget, so that he can find out which data belong to which widget.

Widget settings are available in context menu under right-click. User can set title and other parameters different for each widgets – these parameters will be described in each widget's section later. Widgets can also be locked, which means the widget can't be closed nor moved or resized.

It is possible to precisely position widgets using grid or by using "aligment lines" (see image 6). User can also easily clone widgets by moving them while holding the control key.

Some widgets might profit from following feature: if user grasps widget with mouse as if he wanted to move it and then "shakes it" from right to left, the widget will expand itself to cover all of the visible workspace. When it is moved, it will shrink to it's original size.

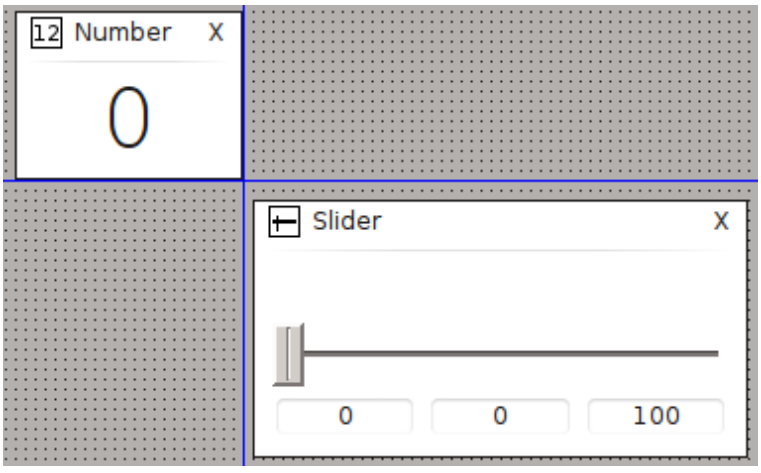


Figure 6: Widget aligment using grid and lines

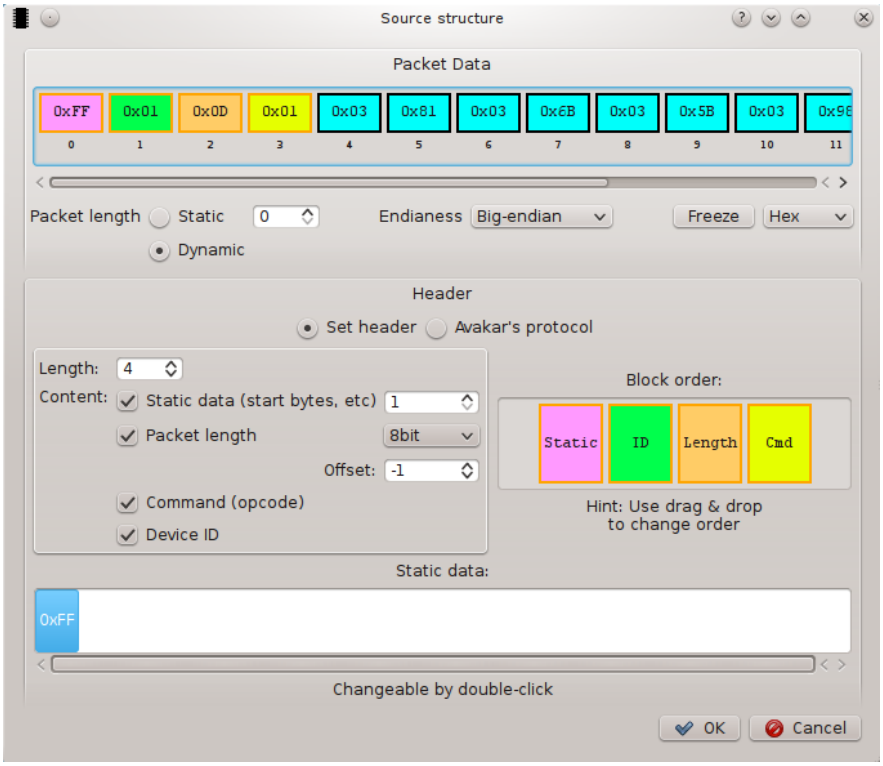
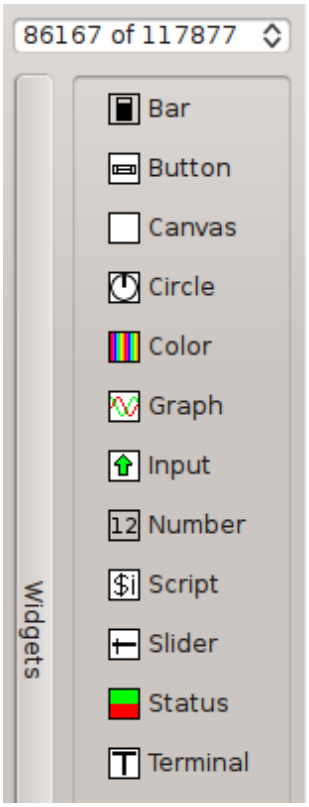
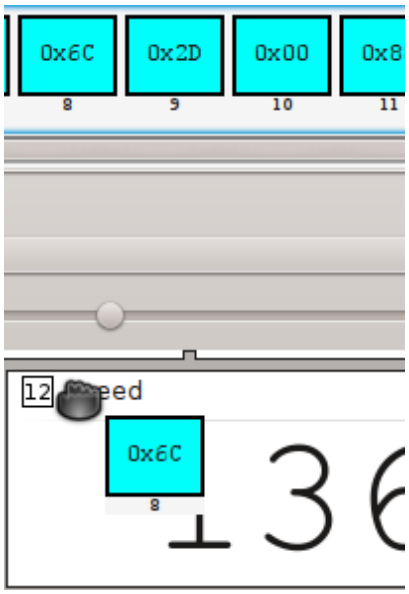


Figure 7: Packet structure dialog



(a) List of widgets



(b) Assigning data using drag&drop

Figure 8: Widgety

3.1 Filters

Analyzer can filter incoming data and each filter may contain several conditions, which determine if packet is filtered out or not.

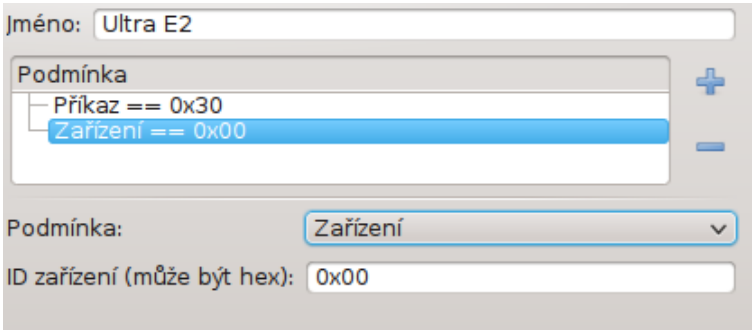


Figure 9: Filter settings

Each condition can check command or device ID from packet's header, value of byte in packet or it can run simple user script. Thanks to the script, it is possible to write almost any kind of condition.

```
// Return true if passes, false if it
// should be filtered out
function dataPass(data, dev, cmd) {
    return false;
}
```

Example 1: Script filter condition

3.2 Widget: number

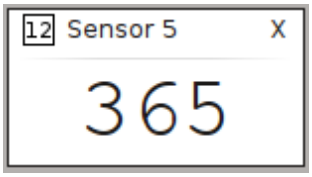


Figure 10: Widget: number

This widget displays integers (both signed and unsigned, 8 to 64bits) and decimal numbers (single-precision[5], 32 and 64 bit). Widget can align the number to max lenght of it's data type and format as follows:

- Decimal – number as base 10
- Decimal with exponent – uses exponent to display big numbers, available only for decimal numbers
- Hexadecimal – number as base 16, available only for unsigned numbers
- Binary – number as base 2, available only for unsigned numbers

[5]Standard floating-point number format used in C and other languages (IEEE 754-2008)

Another feature is option to recalculate widget's value using formula specified by the user. This is useful for example while showing data from infrared range finders, because their output value must be converted to centimeters using equasion. Formula can look like this:

```
2914/(%n+5)-1
```

where `%n` is alias for number which would otherwise be displayed in the widget. This particular formula converts distance measured by Sharp GP2Y0A41 infrared range finder to centimeters.

3.3 Widget: bar

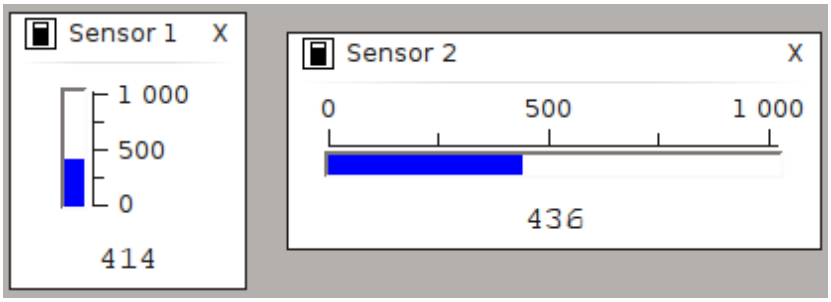


Figure 11: Widget: bar

Data in this widget are displayed as bar. User can set data type (same as widget *number*), orientation (vertical or horizontal) and range of displayed values. It can also use formula to re-calculate it's value in the same way as widget *number*.

3.4 Widget: color

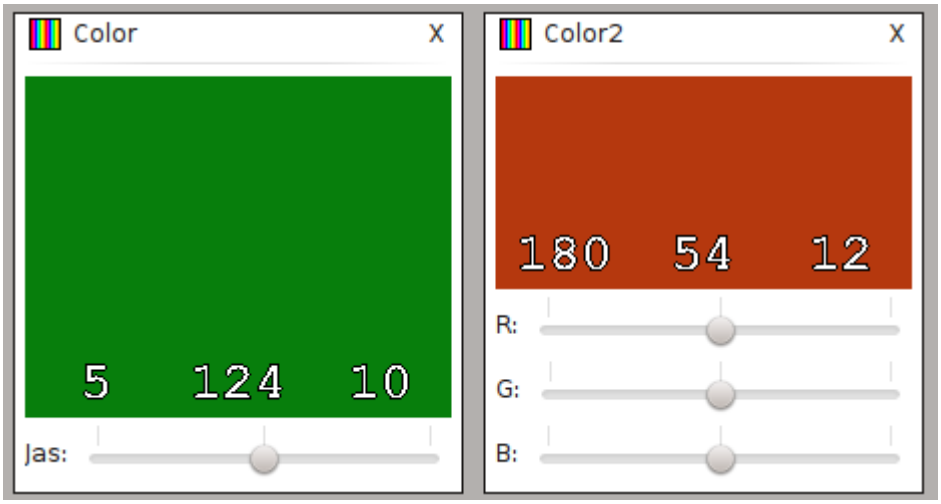


Figure 12: Widget: color

This widget shows incoming data as colored rectangle. Supported color formats:

- **RGB** (8b/channel, 3x uint8)
- **RGB** (10b/channel, 3x uint16)
- **RGB** (10b/channel, 1x uint32)
- **Shades of gray** (8b/channel, 1x uint8)
- **Shades of gray** (10b/channel, 1x uint16)

Widget supports brightness correction for all colors at once or for each color of RGB space separately.

3.5 Widget: graph

This widget shows data in graph – order of the data is on the x axis and data values on the y axis. User can set name, color and data type of each graph

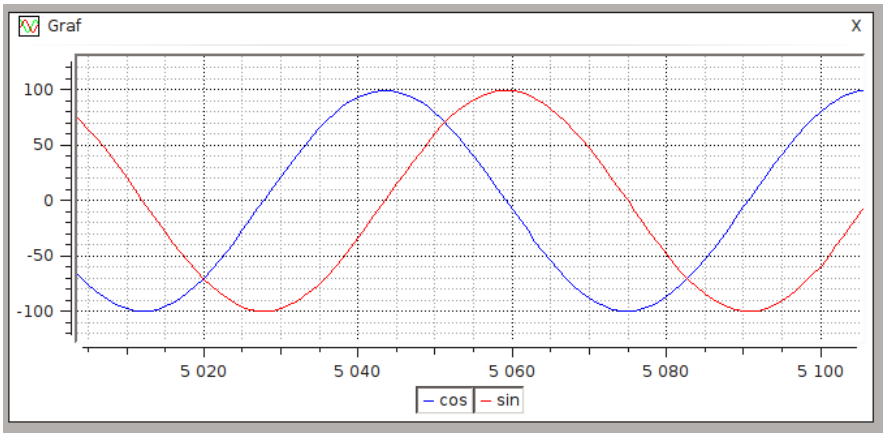


Figure 13: Widget: graph

curve and automatic scrolling, sample size and scale for graph. Graph also has legend which shows curve's names and colors, and curves can be hidden by clicking at their names in legend. Scale of each axis can be changed by scrolling the mouse wheel while hovering the cursor above axis. If the mouse is above graph area, mousewheel changes scale of both axes at once.

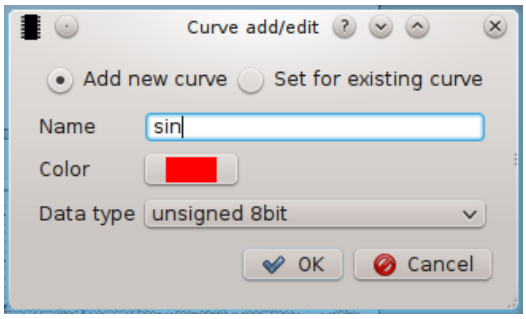


Figure 14: Curve settings dialog

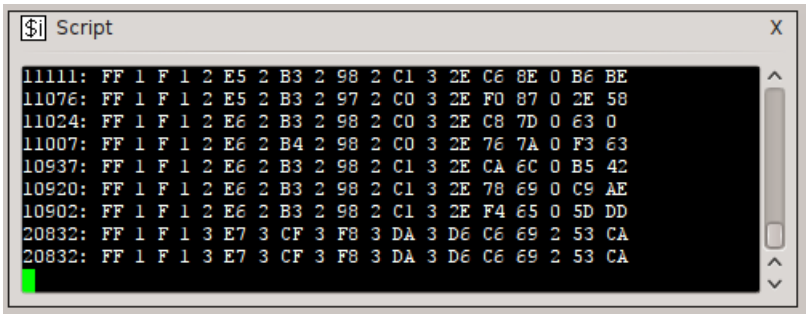


Figure 15: Widget: script

3.6 Widget: script

This widget uses user-written script to process data. Script can be written in Python or QtScript[9] (language based on ECMAScript[6], same as JavaScript[7], which means JavaScript and QtScript are very similar).

Script can process incoming data, react to keypresses and send data to device. Basic output can be displayed in terminal (image 15), but it is also possible to use other widget types to show data (number, bar, ...).

Script editor has built-in code samples, for example how to set value of existing *number* widget, how to send data to device or how to react to keypresses (on image 16 they are hidden under the lightbulb icon). Editor also has link to automatically generated documentation, which is available on http://technika.junior.cz/docs/Lorris/.

[6] *ECMAScript* – scripting language accoring to stadard ECMA-262 and ISO/IEC 16262

[7] *JavaScript* – scripting languge used primarily on web

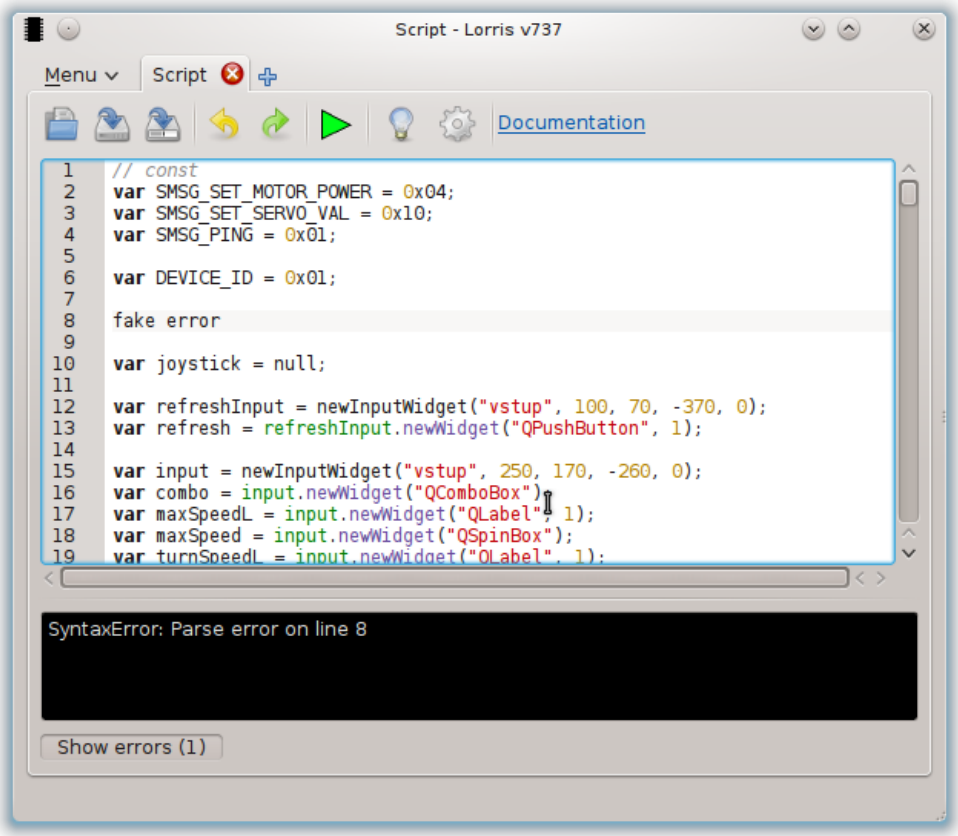


Figure 16: Script editor

3.7 Widget: circle

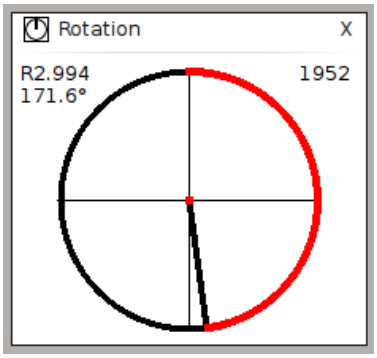


Figure 17: Widget: circle

Widget *circle* shows incoming data as angle in circle, which is useful for example when displaying rotation of robot's wheel. Incoming data can be in degrees, radians or just number in certain range (eg. data from 12bit encoder in range from 0 to 4095).

3.8 Widget: canvas

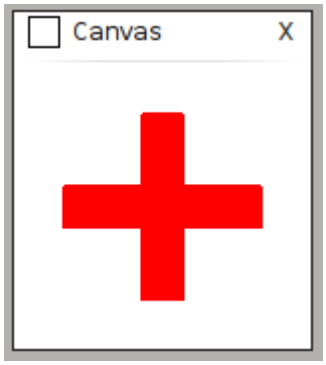


Figure 18: Widget: canvas

Canvas can be only controled from script and is supposed to be used to draw 2D graphics. It can draw lines, rectangles, circles and ellipses. Following code sample will draw red cross in the center of the widget.

```
1 Canvas.setLineColor("red");
2 Canvas.setFillColor("red");
3 // x, y, width, height
4 Canvas.drawRect(55, 10, 20, 110);
5 Canvas.drawRect(10, 55, 110, 20);
```

Example 2: Drawing to canvas

3.9 Widgets button and slider

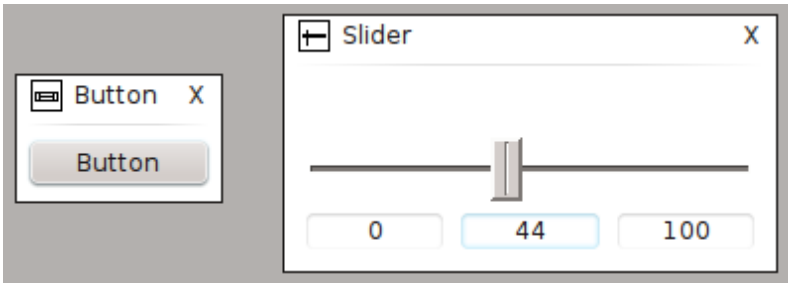


Figure 19: Widgets button and slider

These two widgets are used for interaction with script – callback method in script is invoked on button click. In this method user can for example send a command to robot. Similarly, callback method is invoked after moving slider, so that user can for example change robot's movement speed. Keyboard shortcut can be assigned to button "click" action and for slider to gain focus, so that user can move it using arrow keys.

```
function Slider_valueChanged() {
    appendTerm("Slider value: " + Slider.getValue() + "\n");
}

function Button_clicked() {
    appendTerm("Button clicked\n");
}
```

Example 3: *Slider* and *button* callbacks

3.10 Widget: input

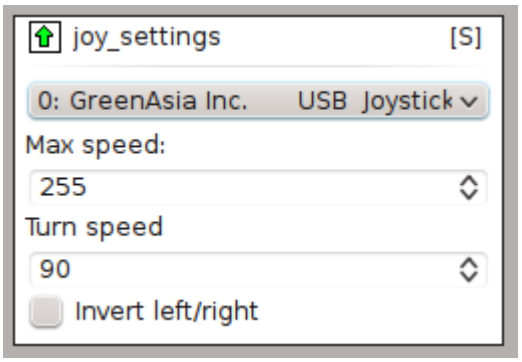


Figure 20: Joystick settings in widget *input*

This widget is also for interaction with script (user *input*), but script also defines interface itself – the widget is empty by default and script has to create UI components, for example button or text field. This widget is a bit more complex, but it can create any of the UI components Qt Framework offers – buttons, slider, text fields, combo boxes and so on. Code sample 4 creates UI from image 20.

```
// args: Qt widget name, stretch value
var joyList = joy_settings.newWidget("QComboBox");
var maxSpdLabel = joy_settings.newWidget("QLabel", 1);
var maxSpd = joy_settings.newWidget("QSpinBox");
var turnSpdLabel = joy_settings.newWidget("QLabel", 1);
var turnSpd = joy_settings.newWidget("QSpinBox");
var invert = input.newWidget("QCheckBox");

// set QLabel text
maxSpdLabel.text = "Max speed:";
```

Example 4: Adding UI components to widget *input*

3.11 Widget: status

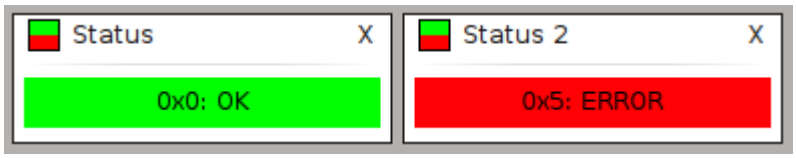


Figure 21: Widget status

Status is designed to show state of for example button (pressed/released) or error status from encoder (0 = okay, other values are error codes). User assigns states to incoming values (state consists of text and it's color, see image 22) and widget then shows active states. It supports "Unknown value", which is shown when incoming data don't match any defined status.

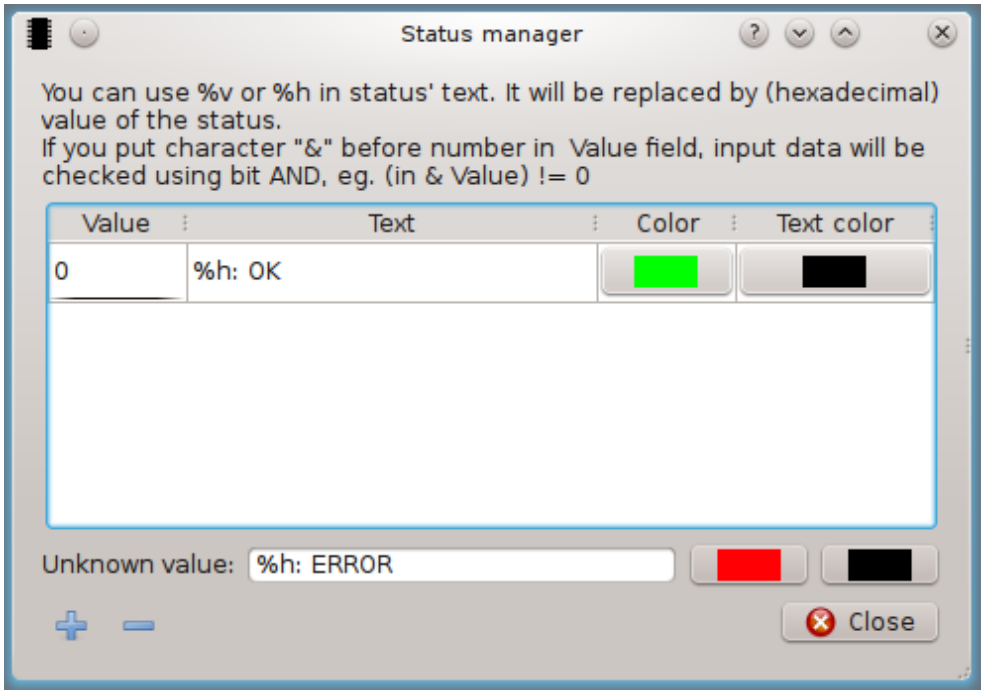


Figure 22: State definitions dialog

3.12 Widget: terminal

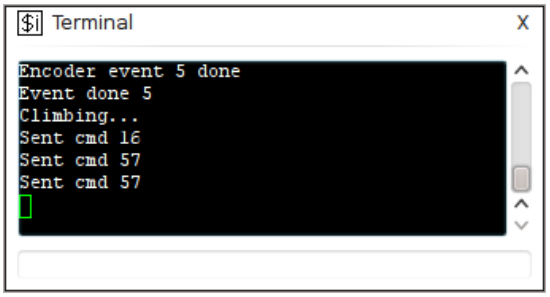


Figure 23: Widget terminal

This widget exists only for convenience of the user, it's widget *script* with preset code working exactly as terminal (sends keypresses, shows incoming data). User can edit predefined script, just like it was regular widget *script*.

4 Module: Proxy between serial port and TCP socket

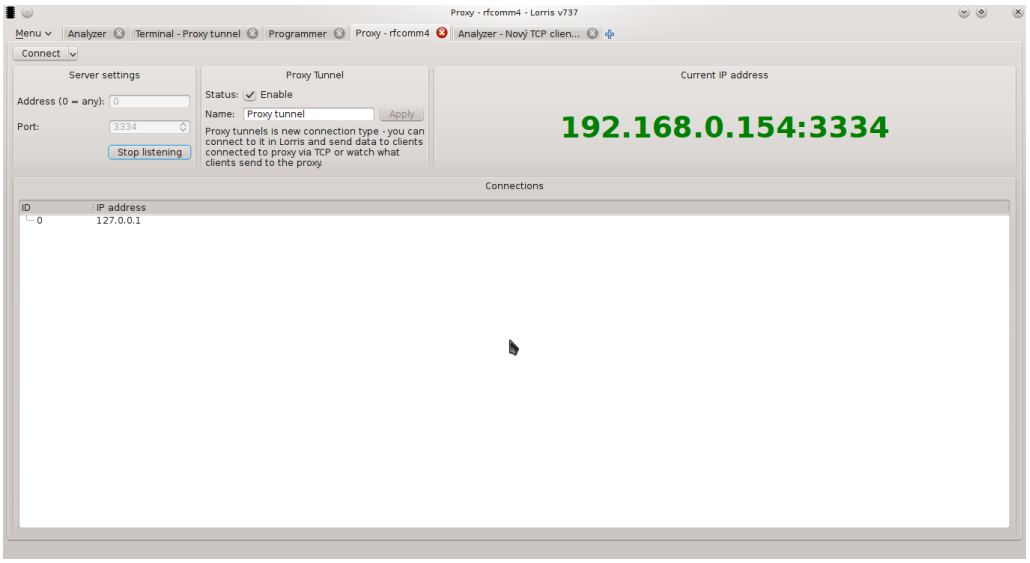


Figure 24: Proxy between serial port and TCP socket

Simple proxy which transfers data between serial port and TCP socket. It creates server the user can connect to from Lorris or other program on different computer. Data are transfered between serial port and connected clients.

4.1 Proxy tunnel

This module also adds new virtual connection – "proxy tunnel". If another Lorris module uses this connection, it can send and receive data from all clients connected to proxy. This can be used to for example generate data in analyzer and then send them to multiple TCP clients.

5 Module: programmer

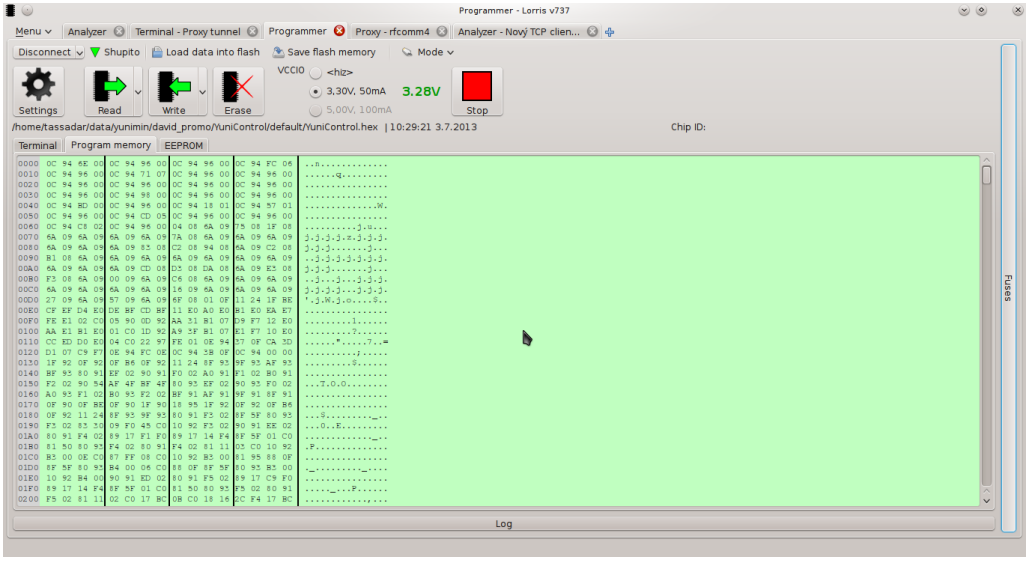


Figure 25: Module programmer

This module acts as graphical interface for several types of programmers and bootloaders. The interface has two modes – full (image 25) and minimal (image 26). Full interface contains all buttons and settings for programming all memories of the chip, minimal interface contains only button which flashes main memory and button to stop chip. Minimal interface is convenient when using the split feature as demonstrated in image 26, because it uses only a small amount of space.

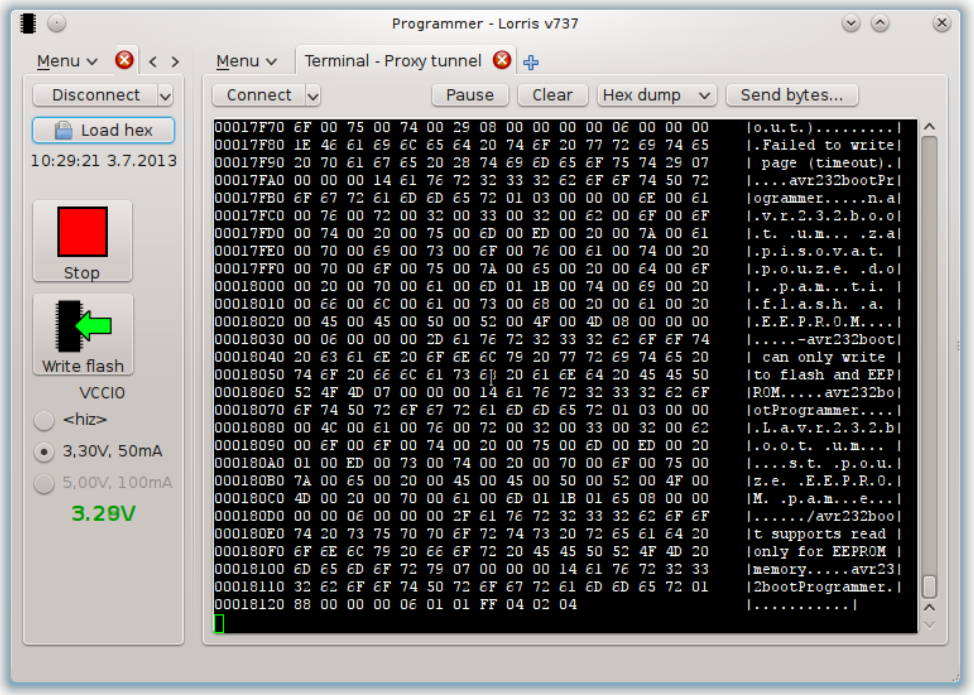


Figure 26: Minimal interface of module *programmer* (left) along with *terminal*

5.1 Shupito programmer

Shupito is microchip programmer created by Martin Vejnár. It can program microcontrollers using ISP[8], PDI[9] and JTAG[10] interfaces.

Module programmer in Lorris is official interface for Shupito programmer. Most of Shupito communication is written by Martin Vejnár.

[8] *In-system programming* – interface which can programm chips directly on their PCB

[9] *Program and Debug Interface* – interface by company Atmel with features similar to ISP

[10] *Joint Test Action Group* – interface standard IEEE 1149.1 which can be used to program and debug chips

5.1.1 UART tunnel

Shupito can create tunnel[11] for UART interface from programmed chip to computer. Lorris can use this feature – active tunnel creates new virtual connection and other modules can connect to it.

5.2 Bootloader avr232boot

Author of this bootloader is also Martin Vejnár. Avr232boot supports only Atmel ATmega chips and it is inspired by reference bootloader code for these chips, but it is designed to be as small as possible. Originally, it could only program flash memory of the chip (the one where program is stored), I added support for programming and reading of EEPROM[12] memory.

Lorris can use this bootloader to program flash memory and read and program EEPROM.

5.3 Bootloader AVROSP

AVR Open Source Programmer is protocol used by several bootloaders by Atmel for chips ATmega and ATxmega. Lorris can use this protocol to program and read both flash and EEPROM memory of the chip.

[11] Direct connection between programmed chip and the computer via programmer

[12] Flash memory which keeps data even without electricity. It is used to store for example program settings.

6 Module: terminal

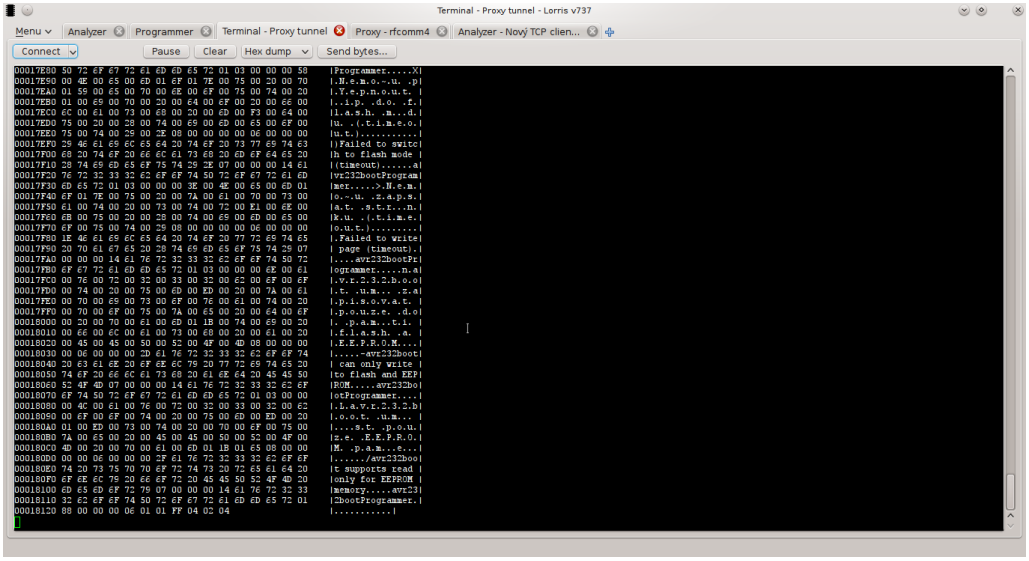


Figure 27: Module terminal

Fundamental tool for every developer, classic text terminal. It shows incoming data in either text mode or as hexadecimal values of each byte and sends keypresses.

User can set terminal's colors, font size, which sequence of control characters should be sent after return key press and behavior of several control characters (for example if character `\n` should create new line or not).

7 Joystick support

Lorris supports joystick in module analyzer to for example control robot. At first, I've used SDL[11] library to access joystick, but it was not really suitable for my use – SDL is video game library, joystick support is only one of many subsystems this library contains. It's architecture also wasn't ideal to use in Lorris.

I haven't found any suitable replacement of SDL, so I wrote my own library.

It is called **libenjoy**, it works under Windows and Linux and it is very small and simple. One major advantage over SDL is that it can remember connected joysticks – if you disconnect joystick and then plug it in again (because you want to reorganize cables on your desktop or because of bad USB connection), it will open the joystick again by itself – without any user interaction.

Libenjoy is released under GNU LGPLv2.1[26] license.

- GIT repository: https://github.com/Tasssadar/libenjoy

8 Usage examples

8.1 Color sensor testing

Situation: I'm builing robot for some competition (Eurobot, RobotChallange, ...) and I want to use color sensor to direct the robot. I also want to test the color sensor, so I've made simple circuit with chip and color sensor. Chip will instruct the sensor to measure the colors and send color values to computer via UART interface.

Solution: I use Shupito to program the chip and it's shupito tunnel to read data from UART interface. I connect analyzer module to shupito tunnel and then use widget *color* to show me color measured by the sensor.

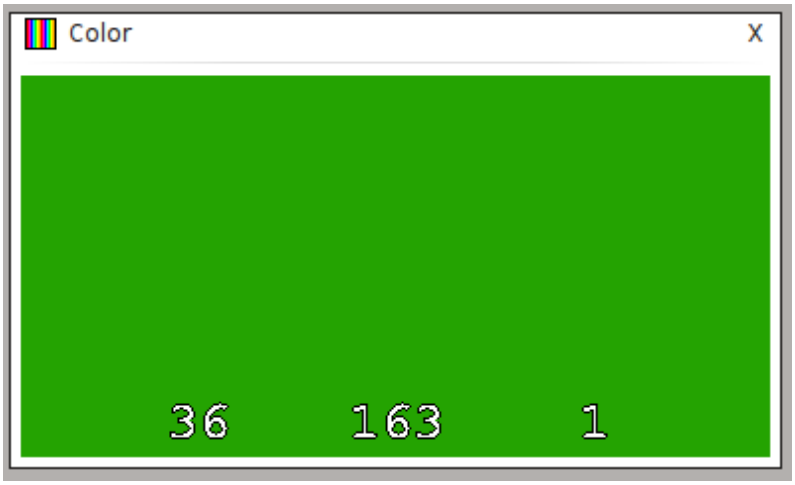


Figure 28: Color in analyzer module

8.2 Encoder testing

My schoolmate Marek Ortcikr made SOČ named *Modular building blocks for robots* (*Modulární stavba robota*). One of the blocks was magnetic encoder. This encoder looks like another wheel for the robot, but little magnet is placed in wheel's axis. Encoder chip placed directly in front of the magnet detects orientation of magnetic field generated by the magnet and therefore rotation of the wheel itself.

Encoder is able to calculate distance covered by the robot and it's speed and acceleration by monitoring changes in wheel's rotation.

Figure 29: Magnetic encoder

Lorris was used to demonsrate encoder's function on national tier of SOČ competition. Whole interface can be seen on image 42 on page 57, following text addresses each part individually.

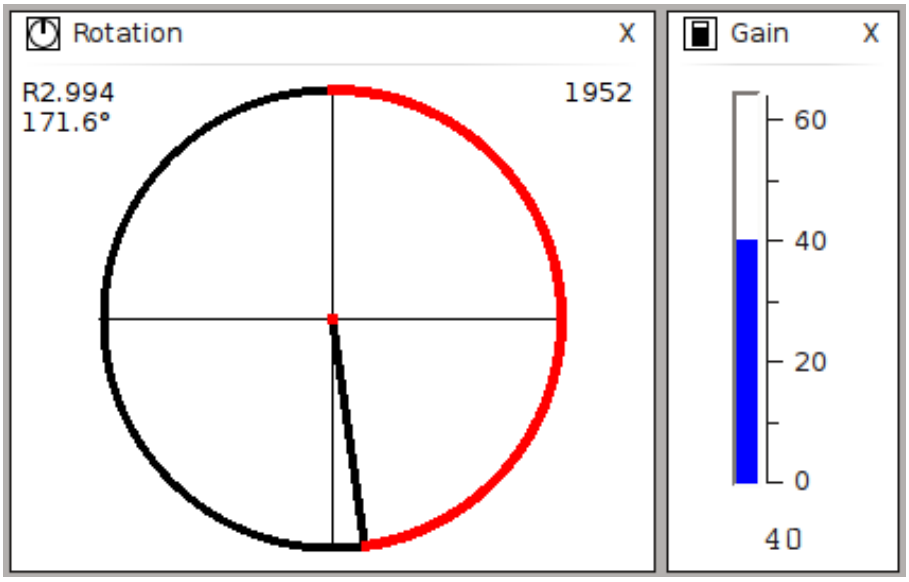


Figure 30: Rotation of the wheel

Widget *circle* is used to represent current rotation of encoder's wheel. Current value read from encoder (0 to 4095) is placed in top right corner, number in left corner are values converted to radians and degrees.

Values in widget *bar* named "Gain" represent strength of the magnetic field, thus how far is the magnet from the encoder's chip. Value is in range from 0 to 63, ideal is about the middle of this range.

Figure 31: Covered distance

This is widget *number* displaying covered distance in millimeters. Encoder sends this value in $\frac{1}{4096}$ of wheel's circumference, so formula `%n/32.5949` has to be used to convert it to millimeters.

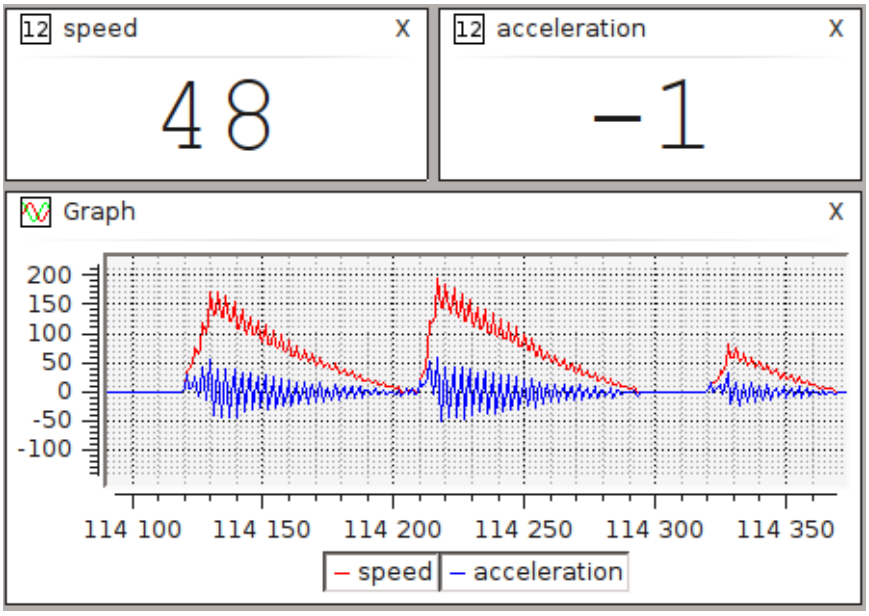


Figure 32: Speed and acceleration

Current speed and accelertaion are displayed in two widgets *number*, and widget *graph* underneath shows speed as red curve and acceleration as the blue one.

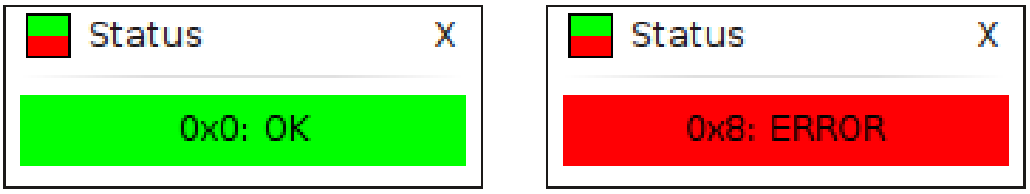


Figure 33: Encoder's status

Encoder's chip also sends status informations. If everything is okay, it sends number `0x0`, if some problem is encountered, it returns one of the error codes (e.g. `0x8` means that there is no magnet present). Widget *status* shows current error code and color according to informations from encoder.

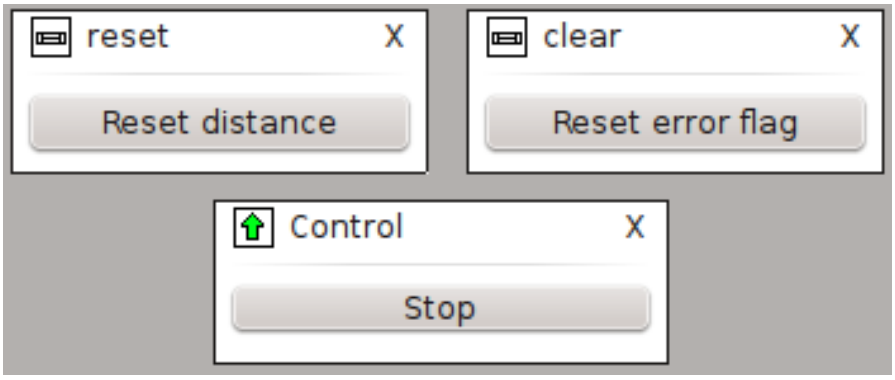


Figure 34: Encoder's controls

These *button* and *input* widgets are here to be used to control the encoder. Button "reset" resets distance counter to zero, button "clear" sends command to clear error code (it still stays even if cause of the error was fixed, it has to be manually cleared) and button "Control" starts/stops data stream from the encoder. All of these widgets are connected to script (image 42 does not contain *script* widget because it didn't fit the window) which reacts to button clicks by sending appropriate commands to encoder.

```
var run = true;
function reset_clicked() {
    sendData(new Array(0xFF, 0x01, 0x01, 0x00));
}
function clear_clicked() {
    sendData(new Array(0xFF, 0x01, 0x01, 0x01));
}
function startStop_clicked() {
    run = !run;
    startStopBtn.text = run ? "Stop" : "Start";
    sendData(new Array(0xFF, 0x01, 0x02, 0x02, run ? 1 : 0));
}
```

Example 5: This script sends commands to encoder

8.3 Tuning of PID regulator

Situation: Robot can't go straight because each motor has slightly different speed. I decided to solve this problem using PID regulator. But PID regulator needs several constants to be correctly set.

Solution: Robot's program is sending current motor speed and PID constants values to computer and also allows changing those constants via UART interface. This program is flashed into robot over bluetooth using avr232boot bootloader – I don't have to use any programmer, which would require cable connection.

I use widgets *number* and *graph* to show current PID constants and speed of both motors. Then I write simple script which will change PID constants after keypress and starts/stops robot.

I've used this process to tune PID regulator on my 3pi[12] robot. I've attened to *Line Follower Standard* competition on Robotic Day 2012 in Prague[13] with this robot and I've won the second place from total of 22 robots.

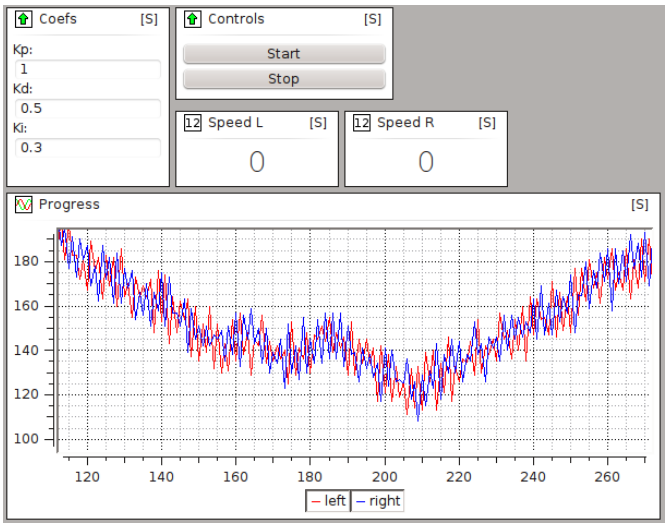


Figure 35: PID regulator tuning

8.4 Developement of robot for Eurobot 2011 competition

Usage of my Lorris program is exaplained here using example case of robot, which was developed on our school (SPŠ a VOŠ technická, Sokolská 1, Brno) in 2011 to compete in Eurobot contest.

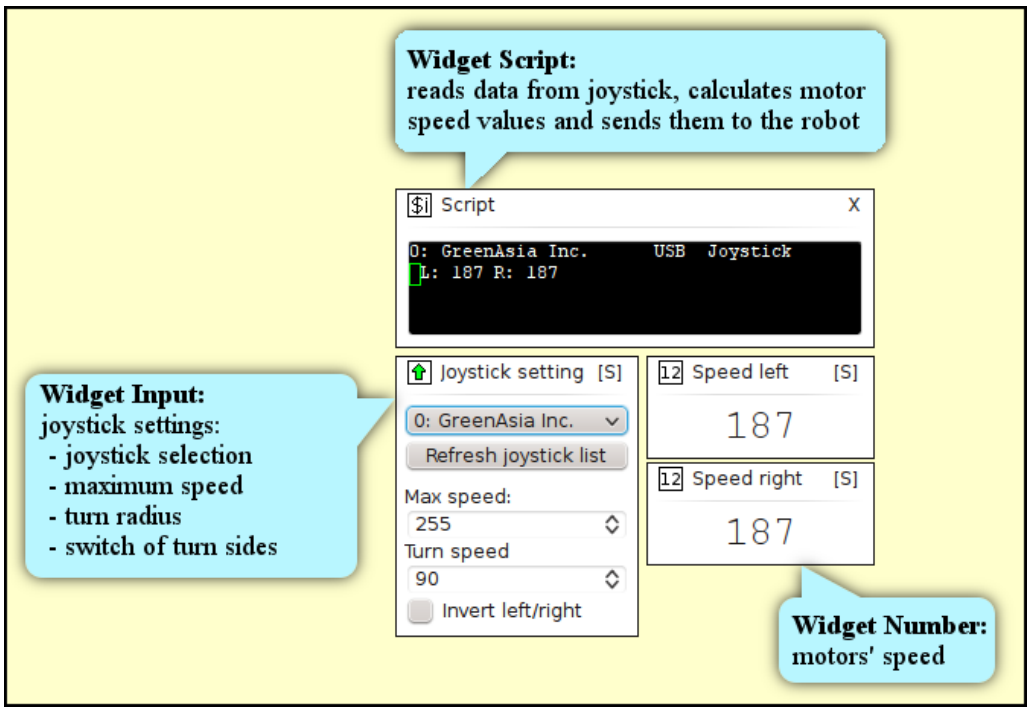
Goal and game mechanics are different each year, in 2011 the goal was to play something like simplyfied chess game. Game fields was divided to red and blue squares and upon it were "pawns" (yellow discs) and robots had to move the pawns to squares of their color or make "towers" by putting pawns top of each other. Winner was the robot with the most points, which were awared for each pawn on square of robot's color and for built towers. In addition to that, robot's had to detect each other in order not to collide (e.g. using ultrasound range finders). For complete rules, results and more informations, see the web page of Eurobot 2011[16].

Most pressing need for tool, which would let us test and debug all of the robot's functions and components quickly and easily, has arisen. Mainly usage of the Analyzer tool is presented here, due to the fact that it is the most visible part of the program. Other tools (Programmer, Terminal) were also used, for example for writing program into the robot's chip.

This example contains simple user interface for controling, testing and debugging of our robot, but this interface can also be used for other robots. You can also create new interface to fulfill your needs, for example when the robot is too atypic and requires different type of controls.

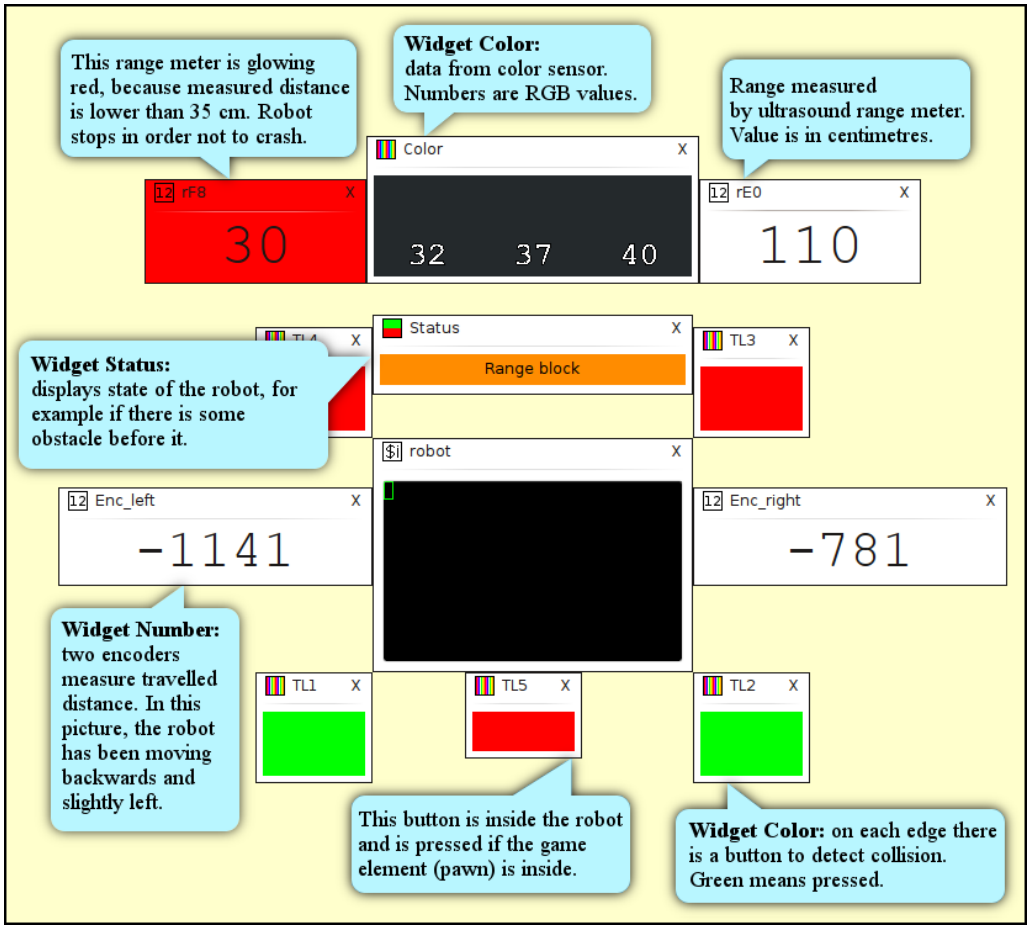
8.4.1 Robot's frame

Body of the robot was constructed first and even in this early stage, my Lorris program was already used. We needed to test if all the motors and servos work properly and how exactly they behave, so I assembled a small group of widgets in Lorris, which would allows to control the robot via joystick. Several widgets were used, namely Script, which was reading data from joystick, calculating the speed values for motors and sending them to the robot. Next, widget *Input*, which contained settings joystick parameters and lastly, 2 widgets *number* with motors' speeds.



8.4.2 Debugging and adjusting of sensors

When the mechanical frame was complete and tested, all sensors were added to the robot. After that, interface to actually see sensor values was needed, so I created interface for this purpose in Analyzer. It uses mainly *script*, *number color* and *status* widgets. Each and every one of these widgets can be moved around Analyzer's worspace and resized. That makes it possible to place widgets so that their positions coresponds with their real positions on the robot. Top view appears to be ideal for this task.



8.4.3 Programming of robot's reactive behavior

Peak of the developement was programming of its behavior on the game field. For this occasion, widget *script* in my Lorris program was used to large extent. Scripting enviroment which encapsulated robot's basic command sets was created in this widget. These command sets allows us to create more complicated behavioral patterns for the robot. It would be possible to write script for the robot directly, but this enviroment considerably simplified and sped up the developement. Another fact is also worth noticing – widget *Script* was used here not only to control the robot, but also to improve functionality of the Analyzer tool itself.

In this example, I use simple "actions", which are executed by the robot step by step. Each action has 3 main parameters – direction of movement, when the robot should stop and what should it do when it arrives at it's target destination. Each action can be changed directly in the scripting enviroment, bypassing the need to re-program robot after every change. All other parts of Lorris are still working, even when the robot is controlled by the script. That makes it possible to keep track of robot's state as well as all his sensors and quickly find the source of possible unexpected behavior.

You can find image no. 43 which belongs to this part of the text in attachments on page 58.

9 Android application

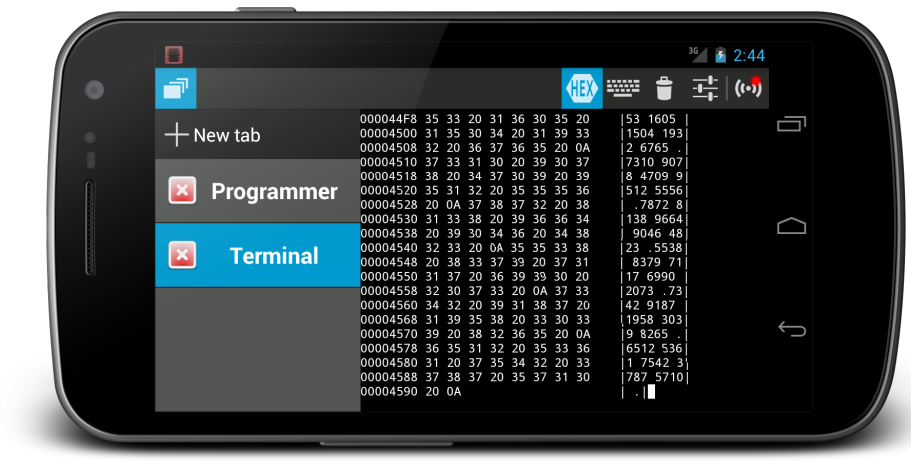


Figure 36: Lorris mobile

Application for Google Android[18] platform is the next step in Lorris' developement, because mobile devices with this operating systems are almost always at hand and are sufficient to quickly solve smaller problems.

Application **Lorris mobile** acts as portable addition to desktop version of Lorris – it may not have all the features of desktop version, but helps when you need to quickly correct or debug something out in the field.

App works on all tablets and phones with Android OS version 2.2 and higher, it is optimized also for bigger tablet screens and can be obtained in official distribution channel of Android application – in Google Play Store[19]. You can find it by searching for "Lorris".

Lorris mobile has similiar architecture as desktop Lorris. User has to create session first, so that everything he opens can be saved (image 37). After user loads the sessions, he gets to main screen of the application, where he can open modules in tabs, much like in desktop Lorris (image 38).

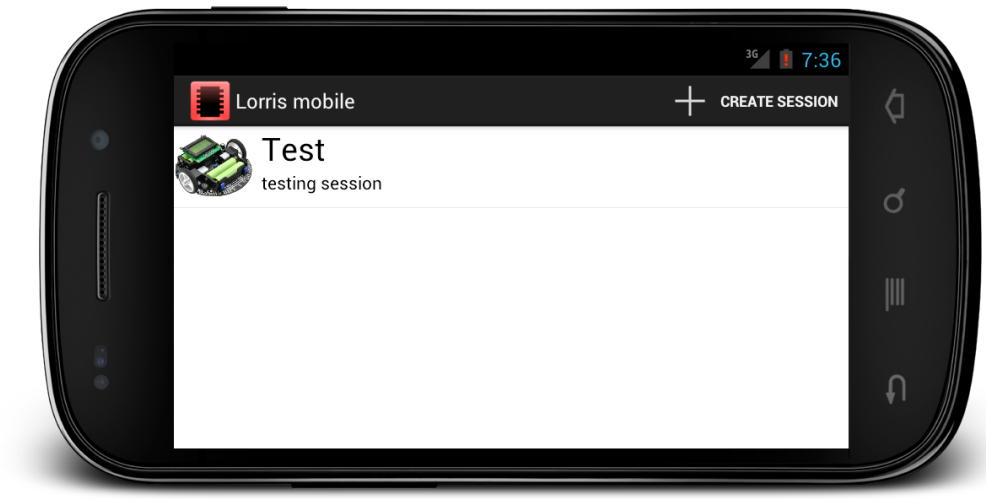


Figure 37: Lorris mobile – session selection

Figure 38: Lorris mobile – switching tabs

9.1 Programmer

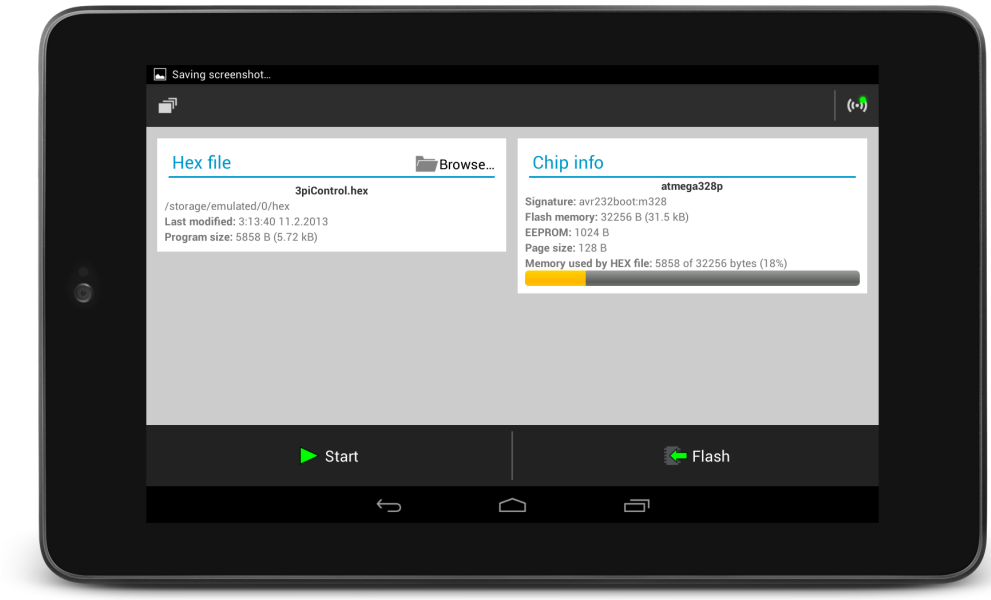


Figure 39: Lorris mobile – programmer

Module programmer can program chips using bootloaders **avr232boot** and **AVROSP** and also using Shupito programmer, if the device has USB host capabilities.

This part of Lorris mobile uses pieces of native code from desktop Lorris, which means the code is faster and easier to maintain.

9.2 Terminal

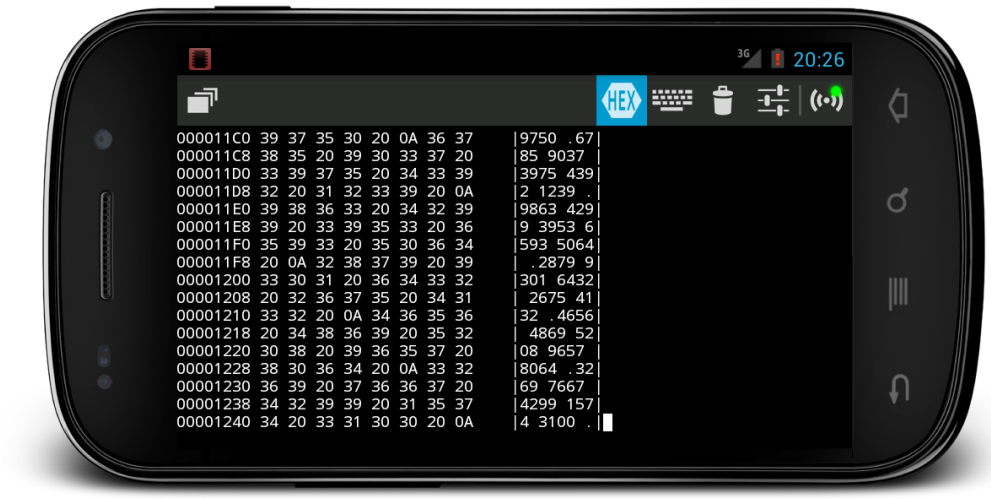


Figure 40: Lorris mobile – terminal

Classic terminal. Is has most features of the terminal in desktop version – it displays data (as text or hexadecimal values), sends keypresses and user can set terminal's colors, font size and which control characters are sent after return key press.

10 Real world usage

Members of some of the technical clubs on DDM[13] Junior[21] in Brno became the first users of Lorris toolbox on the very beginning of its development several years ago. Lorris helps there with developement of various devices, mainly robots, and kids who learn how to program microchips use *Shupito* programmer and thus also the Programmer module in Lorris. Modules terminal and analyzer are also useful during microchip programming lessons, terminal for simple communication with the chip and later analyzer for more advanced data processing.

Lorris is ideal for use in DDM Junior also because it is free – bigger company which makes microchip applications would probably get expensive commercial program similar to Lorris or develop its own single-purpose applications. However, solution used by large companies is somewhat unaccessible for state-funded institutions.

Nowaday Lorris has about 20 users on DDM Junior alone. Number of users however rising thanks to gradual spreading of *Shupito* programmer among users in whole Czech Republic.

I use Lorris whenever I work with robots and/or microcontrollers – to display data, program chips or control whole devices. Following list presents only some of the most significant applications made by other Lorris users:

- Development of several robots for this year's Robotic day[17]
- Programming of widge variety of microchips, using either *Shupito* programmer or bootloaders
- Development of *Shupito* itself
- Development of cheap logic probe
- Debugging of chips for control of three-phase motors (i.e. *drivers*)

[13] *Dům dětí a mládeže* – Organization which does clubs, camps or trips for kids

- Developmnet of system for controlling of up to 128 RGB LEDs for illumination of model plane
- Construction and programming of digital radio transmitter with ARM processor (semestral work)
- Developmnet of line following robot (graduation work)
- Tuning of PID controller

Conclusion

After several years of developement, I can certainly say the application meets all the requirements declared in chapter 1:

✔ 1. Ability to process data from device and show them clearly

✔ 2. Support for many formats of incoming data

✔ 3. Quick and simple to use

✔ 4. Support for other operating systems than MS Windows

✔ 5. Low price

✔ 6. Ability to easily expand program, ideally open-source

✔ 7. No dependencies on other applications (eg. MS Office Excel)

On top of that, the program greatly outdoes original goals – it can also send data to device, program microchips and create proxy between serial port and TCP socket. In comparison to other applications I've found (as described in introduction) Lorris is also the only one which allows user to write his own script to parse data.

Lorris has already been used in several real-world applications and it also has growing ranks of satisfied users, as described in chapter 10.

The application is continuously being enhaced, it is possible to virtually indefinitelly add either more widgets to Analyzer (compass, gauge meter, ...) or whole new modules (e.g. interface for cheap logic probe currently in developmet by Martin Vejnár). The program consist from about 32 thousand lines of code (without thir-party libraries) at this time (3.4.2013).

Enclosed CD contains source code, orientation video and promotional poster.

```
tassadar@nymeria:~/Lorris$ cloc --exclude-lang=make,IDL,Javascript,Python src
     313 text files.
     312 unique files.
      63 files ignored.

http://cloc.sourceforge.net v 1.56  T=4.0 s (62.8 files/s, 10558.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                            123           5755           1005          24643
C/C++ Header                   128           2386            751           7695
-------------------------------------------------------------------------------
SUM:                           251           8141           1756          32338
-------------------------------------------------------------------------------
```

Figure 41: Lines of code counted by program CLOC[20]

In the future, I would like to continue in adding new features to both computer and Android version of Lorris and in increasing of awarness about this useful piece of software.

ATTACHMENT A:

Third-party libraries

- **Qwt**[22] is library for Qt Framework which contains several widgets for techical applications – graphs, bars, compasses, gauge meters or similar.
- **QExtSerialPort**[23] provides connection to serial port and also enumerates serial ports found in the computer.
- **QHexEdit2**[24] is hex editor used to show content of chip's memory in module Programmer. I changed few mostly visual details in this library.
- **Tango Icon Library**[29] is set of icons released to the Public Doamain. Icons from this set on many places throughout the application.
- **EcWin7**[30] is library which provides API for progressbar built into task bar items in Windows 7.
- **QScintilla2**[31] is advanced text editor for Qt
- **PythonQt**[33] – python bindings for Qt.
- **Python**[34] is programming language, Lorris uses several parts of it's interpreter in combination with Python Qt.
- **Qt Solutions**[36] is collection of several addon classes for Qt.
- **libyb**[37] is library which provides communication with the *Shuputo* programmer

ATTACHMENT B:

Licenses

Lorris is released under the GNU GPLv3[25], licenses of used applications and libraries are as follows:

- **Qt Framework** is released under GNU LGPLv2.1[26]
- **Qwt** is distrubuted under the Qwt license[27], which is based of GNU LGPLv2.1
- **QExtSerialPort** is released under The New BSD License[28]
- **QHexEdit2** is released under the GNU LGPLv2.1
- **Tanto Icon Library**[29] is released to the Public Domain
- **EcWin7** is released under the GNU GPLv2
- **QScintilla2** is released under the GNU GPL v2 a v3
- **libenjoy**[32] is released under the GNU LGPLv2.1
- **PythonQt** is released under the GNU LGPLv2.1
- **Python** is released under the PSF License agreement[35]
- **Qt Solutions** is distrubuted under The New BSD License
- **libyb** is released under the Boost Software License[38]

All of these licenses allow free usage and spreading of the code.

ATTACHMENT C:

References

[1] *SerialChart* – Analyse and chart serial data from RS-232 COM ports
http://code.google.com/p/serialchart/
(Prior to 2. 25. 2013)

[2] *WinWedge* – RS232 data collection software
http://www.taltech.com/products/winwedge/
(Prior to 2. 25. 2013)

[3] *Advanced Serial Data Logger*
http://www.aggsoft.com/serial-data-logger.htm
(Prior to 2. 25. 2013)

[4] *StampPlot Pro* – Graphical Data Acquisition and Control
http://www.selmaware.com/stampplot/index.htm
(Prior to 2. 25. 2013)

[5] *LabVIEW* – Laboratory Virtual Instrumentation Engineering Workbench
http://sine.ni.com/np/app/main/p/docid/nav-104/lang/cs/
(Prior to 3. 6. 2013)

[6] *Qt* – Cross–platform application and UI framework
http://qt-project.org/
(Prior to 2. 25. 2013)

[7] *Debian Linux* – The Universal Operating System
http://www.debian.org/
(Prior to 2. 25. 2013)

[8] *GitHub* – Social Coding
https://github.com
(Prior to 2. 25. 2013)

[9] *Making Applications Scriptable*
http://qt-project.org/doc/qt-4.8/scripting.html
(Prior to 2. 25. 2013)

[10] *XBoot* – Extensible bootloader for ATMEL XMEGA microcontrollers
http://code.google.com/p/avr-xboot/
(Prior to 3. 6. 2013)

[11] *SDL* – Simple Directmedia Layer
http://www.libsdl.org/
(Prior to 2. 13. 2013)

[12] *Pololu 3pi Robot*
http://www.pololu.com/catalog/product/975
(Prior to 2. 25. 2013)

[13] *Robotic day 2012*
http://www.robotickyden.cz/2012/
(Prior to 2. 25. 2013)

[14] *Robotic day 2012* – results of Line Follower standard competition
http://www.robotickyden.cz/2012/results/lfs.php
(Prior to 2. 28. 2013)

[15] *Eurobot*
http://www.eurobot.org/
(Prior to 2. 25. 2013)

[16] *Eurobot 2011*
http://www.eurobot.cz/eurobot2011.php
(Prior to 2. 25. 2013)

[17] *Robotic day*
http://www.robotickyden.cz/
(Prior to 3. 4. 2013)

[18] *Google Android* – Operating system for smartphones
http://www.android.com/
(Prior to 2. 25. 2013)

[19] *Google Play Store* – Store with applications for Android OS
http://play.google.com/store
(Prior to 2. 14. 2013)

[20] *CLOC* – Count Lines of Code
http://cloc.sourceforge.net/
(Prior to 2. 25. 2013)

[21] *DDM Junior, Dornych 2, Brno, 656 20*
http://www.junior.cz
(Prior to 3. 4. 2013)

[22] *Qwt* – Qt Widgets for Technical Applications
http://qwt.sourceforge.net/
(Prior to 2. 25. 2013)

[23] *QExtSerialPort* – Qt interface class for old fashioned serial ports
http://code.google.com/p/qextserialport/
(Prior to 2. 25. 2013)

[24] *QHexEdit2* – Binary Editor for Qt
http://code.google.com/p/qhexedit2/
(Prior to 2. 25. 2013)

[25] *GNU General Public License v3*
http://gplv3.fsf.org/
(Prior to 2. 25. 2013)

[26] *GNU Lesser General Public License v2.1*
http://www.gnu.org/licenses/lgpl-2.1.html
(Prior to 2. 25. 2013)

[27] *Qwt license*
http://qwt.sourceforge.net/qwtlicense.html
(Prior to 2. 25. 2013)

[28] *The New BSD License*
http://www.opensource.org/licenses/bsd-license.php
(Prior to 2. 25. 2013)

[29] *Tango Icon Library*
http://tango.freedesktop.org/Tango_Icon_Library
(Prior to 2. 25. 2013)

[30] *EcWin7* – Windows 7 taskbar progress indicator
http://www.msec.it/blog/?p=118
(Prior to 2. 25. 2013)

[31] *QScintilla2* – Code editor
http://www.riverbankcomputing.co.uk/software/qscintilla/intro
(Prior to 2. 25. 2013)

[32] *libenjoy* – Small simple joystick library
https://github.com/Tasssadar/libenjoy
(Prior to 2. 25. 2013)

[33] *PythonQt* – Python bindings for Qt
http://pythonqt.sourceforge.net/
(Prior to 2. 25. 2013)

[34] *Python*, the programming language
http://www.python.org/
(Prior to 2. 25. 2013)

[35] *PSF License agreement*
http://docs.python.org/2/license.html
(Prior to 2. 25. 2013)

[36] *Qt Solutions*, a collection of minor Qt add-ons
http://qt.gitorious.org/qt-solutions
(Prior to 2. 25. 2013)

[37] *libyb*, a collection of minor Qt add-ons
https://github.com/avakar/libyb
(Prior to 2. 25. 2013)

[38] *The Boost Software License*
http://www.boost.org/users/license.html
(Prior to 2. 25. 2013)

ATTACHMENT D: Large images

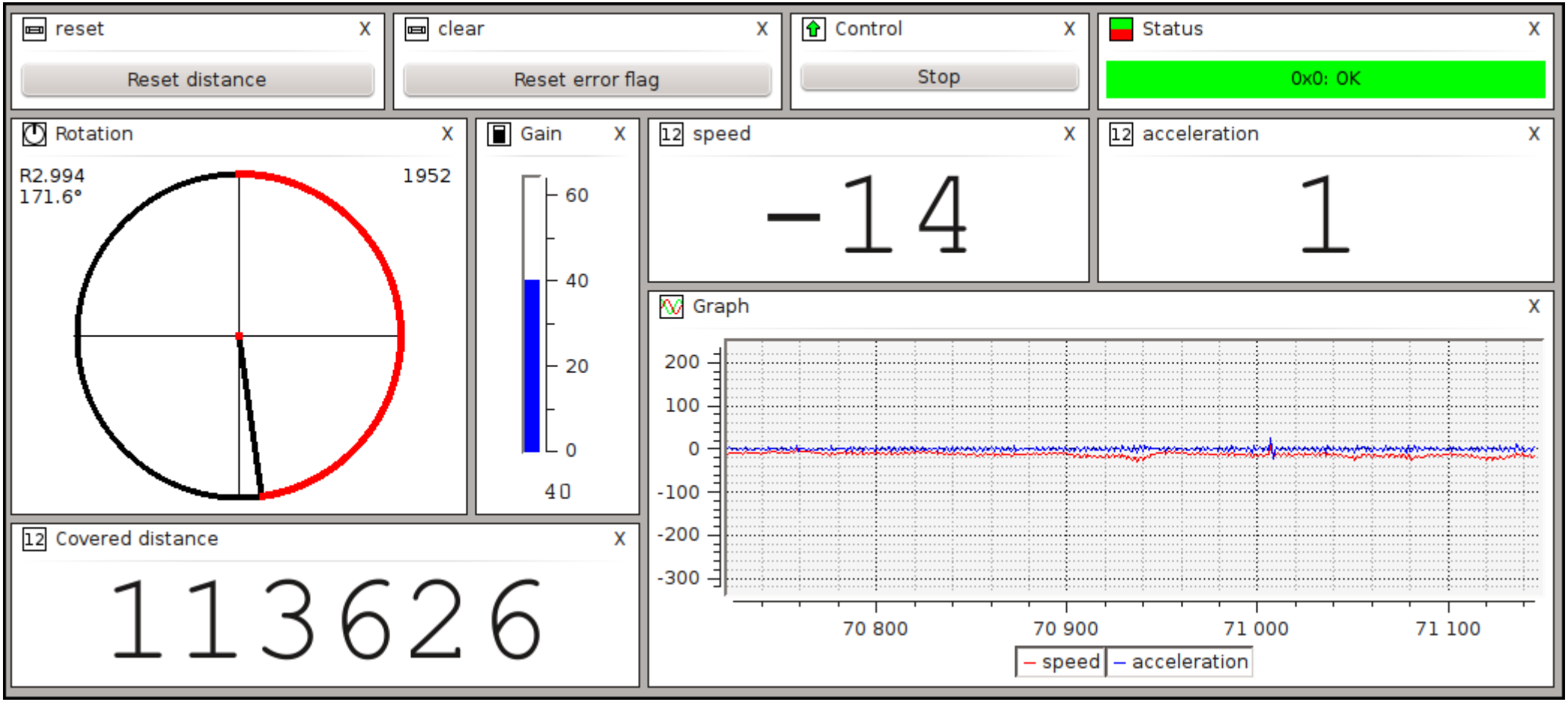


Figure 42: Data from the encoder processed by analyzer

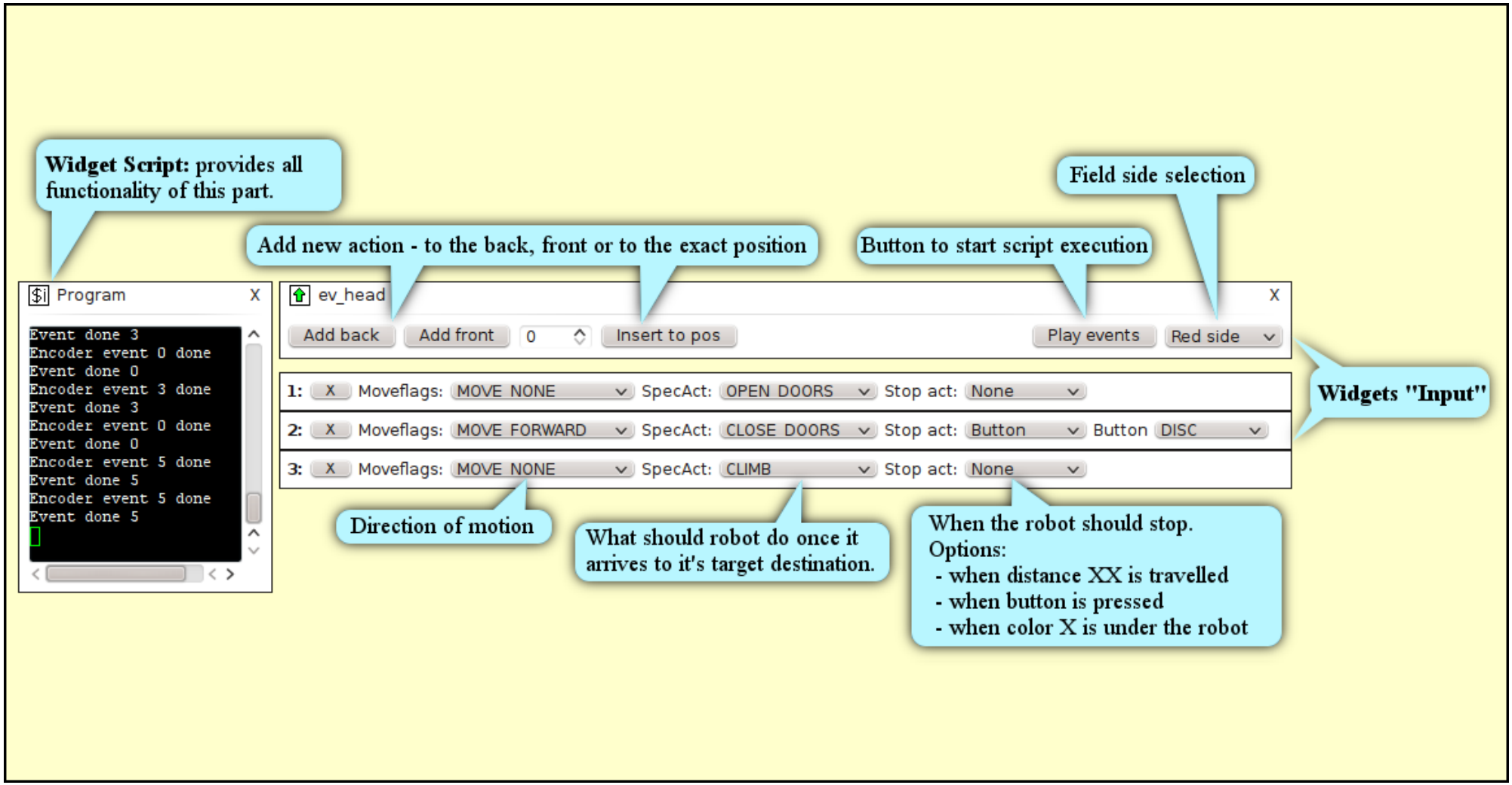


Figure 43: Programming of robot's behavior in module *Analyzer*

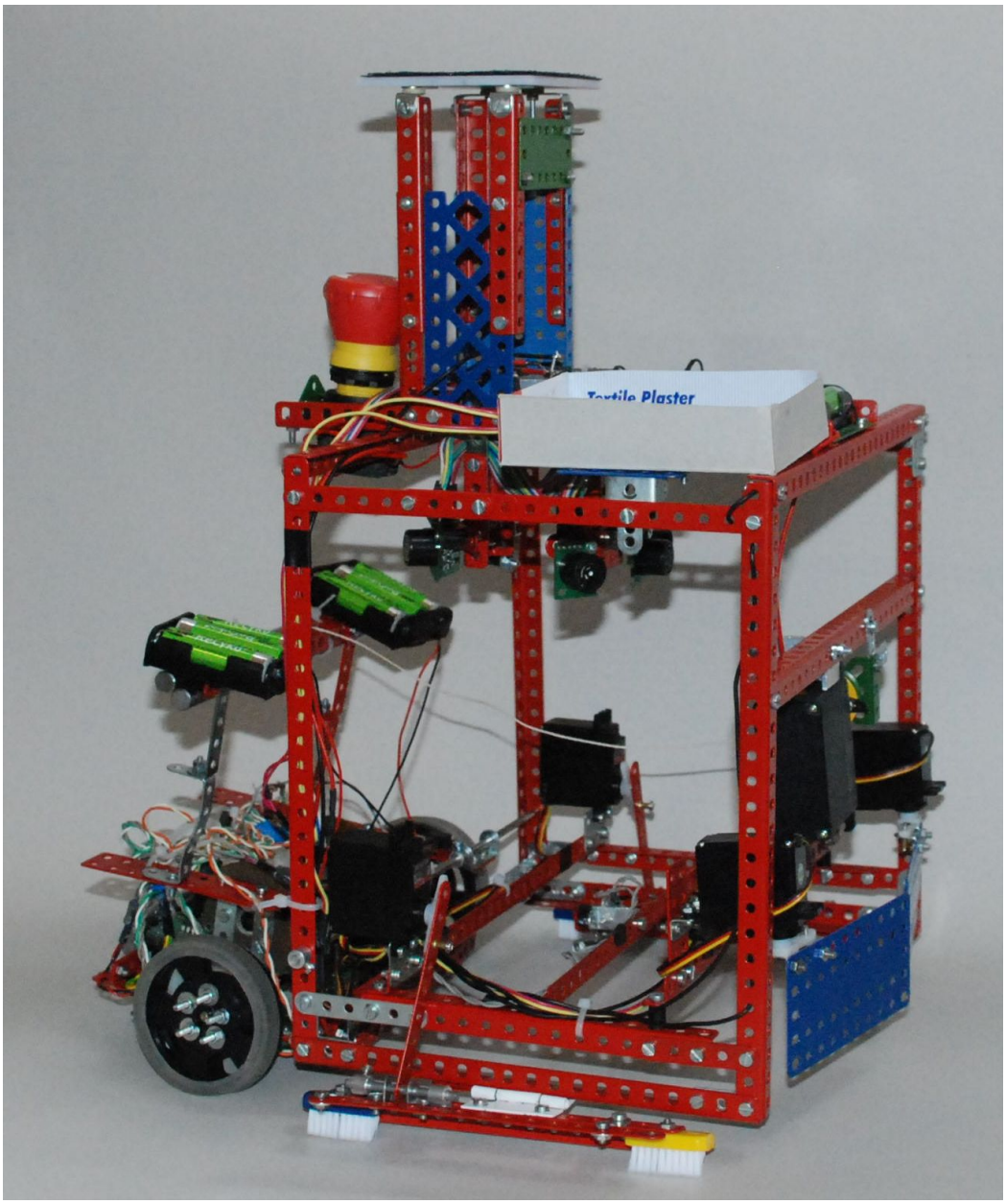


Figure 44: *David*, our robot, ended up on 4th place from the total of eleven robots in national round of Eurobot 2011 competition

ATTACHMENT E:

List of Figures