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

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně, použil jsem pouze podklady (literaturu, SW atd.) citované v práci a uvedené v přiloženém seznamu a postup při zpracování práce je v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Brně dne: 6.3.2013 podpis:

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 toolbox designed for developement and control of any device capable of connecting to serial port or TCP socket.

The application contains several modules and each module is designed for one particular function: parsing and displaying data, programming microcontrollers, etc.

This software is designed to speed-up and simplify developement and testing of applications for microcontrollers, typically programming and controling various kinds of robots.

Key words: data analysis, computer program, robot, graphical interface, developement and testing of applications

Contents

Introduction

I'm 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 detects collision with borders of the game field ...), and there was no way to comfortably and clearly show data from those sensors.

Requirements for application

I require following features from the application:

1. Ability to process data from device and cleary show them
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)

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-soruce 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 graph in MS Excel or as 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.

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	✔	✔	✗	✗	✔	✗	✔

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

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

1 Lorris

Lorris is program written in C++ with use of Qt Framework[5]. Qt is multiplatform framework, which (among other things) makes it possible to run Lorris on multiple operating systems – I'm using Debian Linux[6] (Wheezy, 64bit) and Windows 7 for testing.

1.1 Website and repository

Lorris' GIT[2] repository is hosted on GitHub[7]. 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
- 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] *GIT* – distributed version control system

1.2 Application's structure

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 (eg. 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 windows to multiple parts like presented in image 2 – windows is divided in the middle, you can see two tabs at once. The one on the left is analyser and the other one is terminal.

Connection options:

- Serial port
- Shupito Tunel (virtual serial port, more in chapter 4.1.1)
- TCP socket[3]
- Loading data from file

It is possible to connect multiple modules to one device.

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

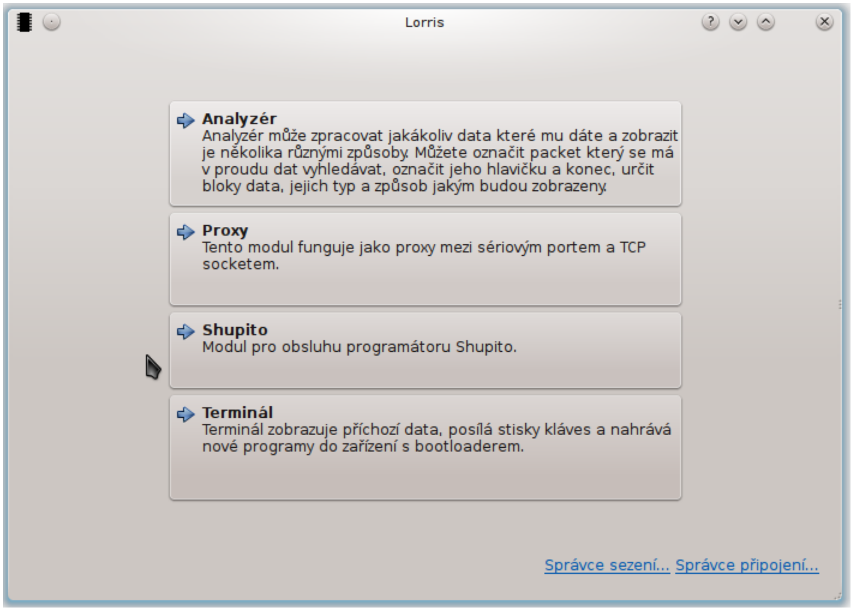


Figure 1: Tab creation dialog

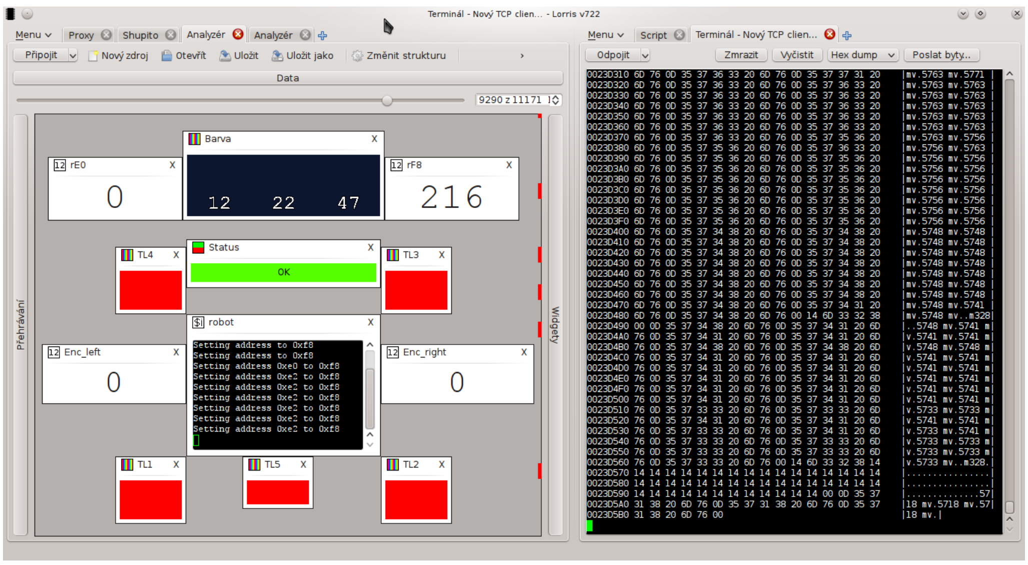


Figure 2: Window divided to multiple parts

1.3 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.

1.4 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.

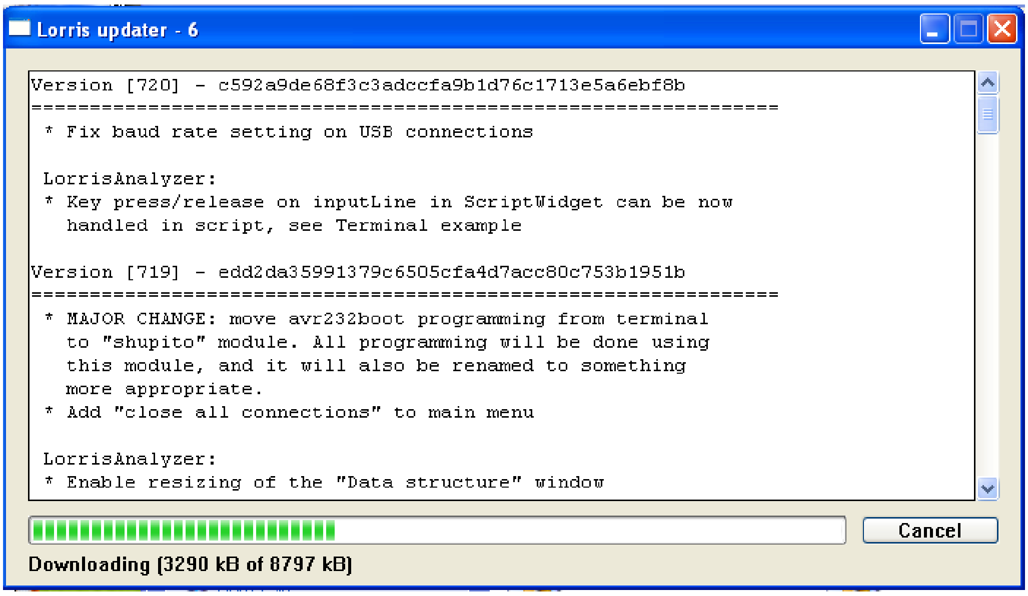


Figure 4: Ongoing update

2 Module: Analyzer

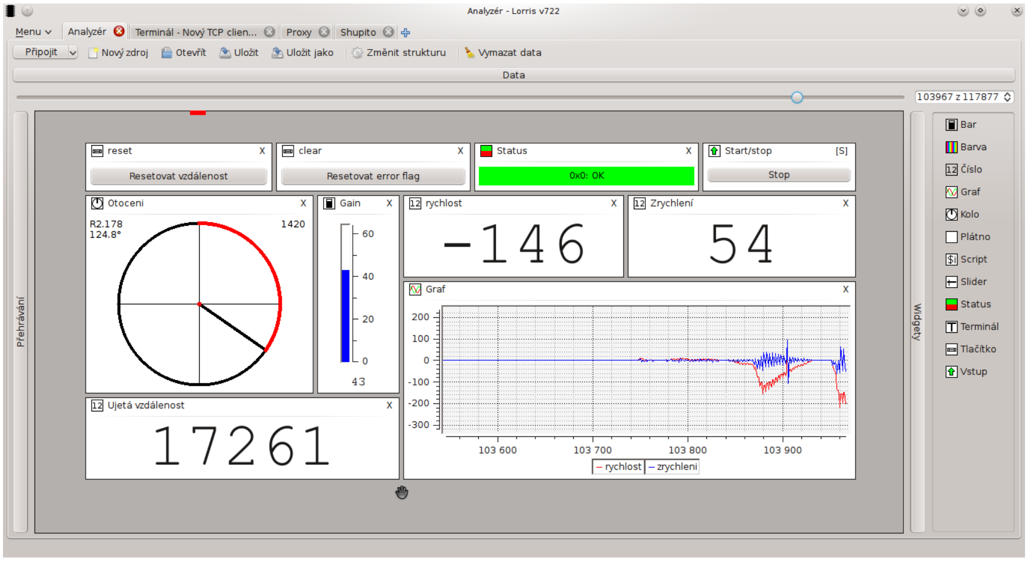


Figure 5: Modul analyzer

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 strcuture is set and user can then ”drag” widgets from 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 know 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 ”alignment 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.

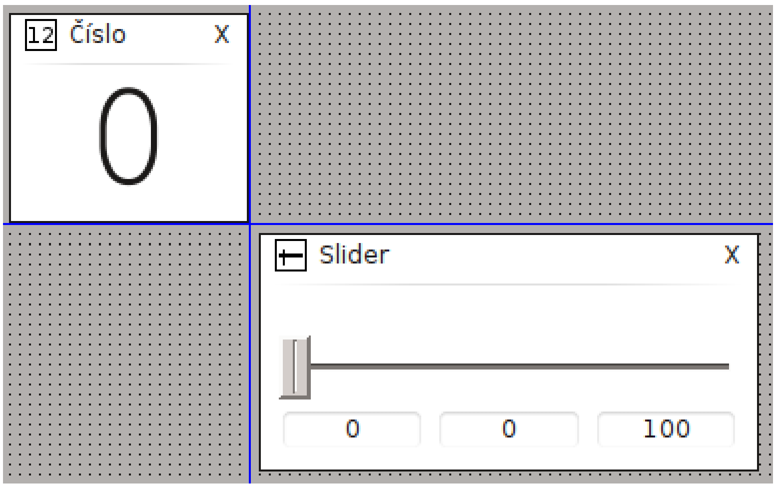


Figure 6: Widget aligment using grid and lines

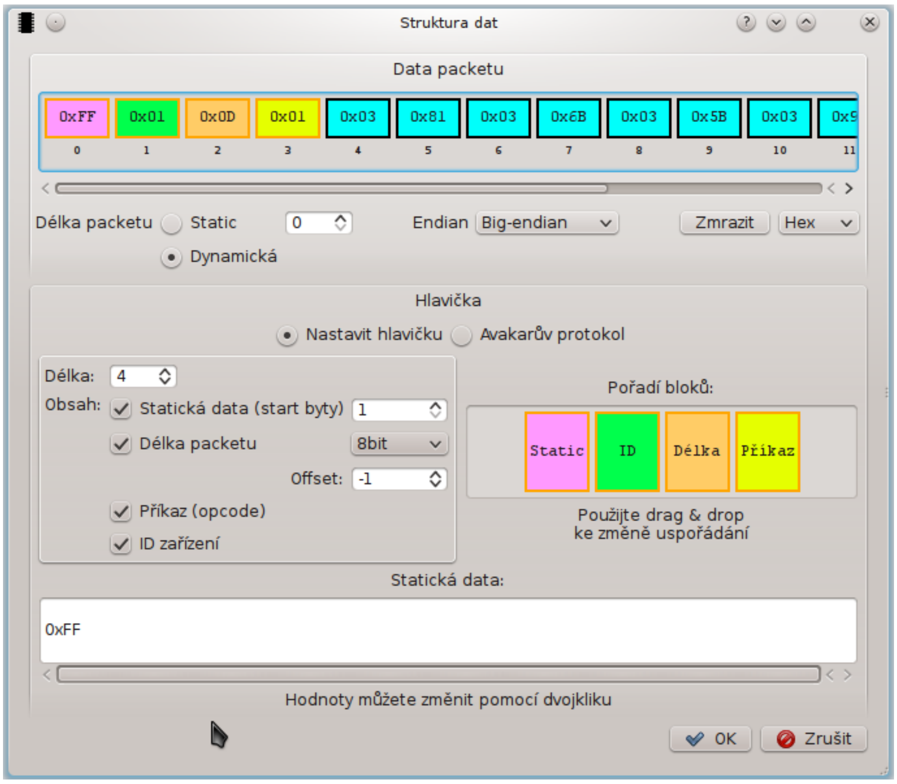
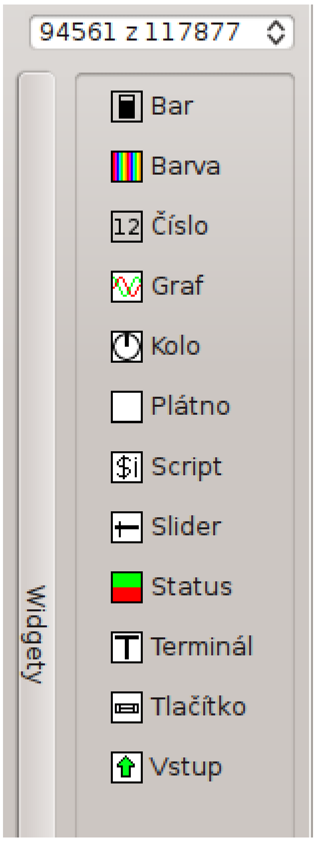
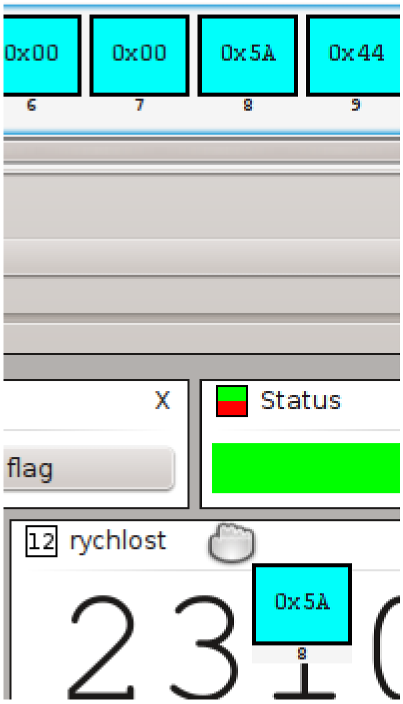


Figure 7: Packet structure dialog



(a) List of widgets



(b) Assigning data using drag&drop

Figure 8: Widgety

2.1 Filters

Analyzer can filter data using multiple filters at once. Each filter contains conditions, which determine if packet is filtered out or not.

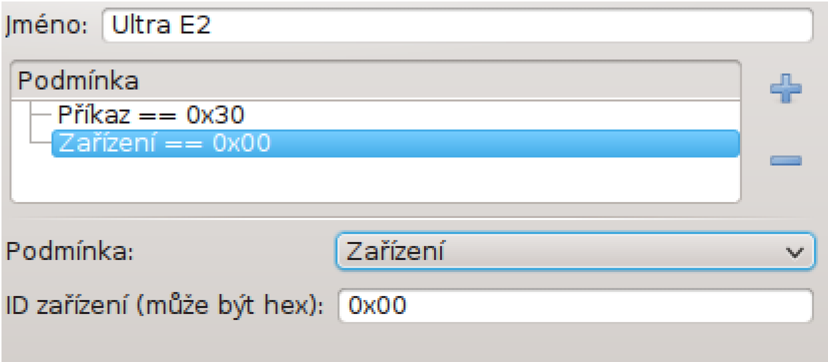


Figure 9: Filter settings

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

2.2 Widget: number

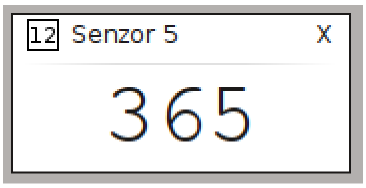


Figure 10: Widget: number

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

Another feature is formula to re-calculate widget's value. This is useful for example while showing data from infrared range meters, 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 is for converting distance measured by Sharp GP2Y0A41 infrared range meter to centimeters.

2.3 Widget: bar

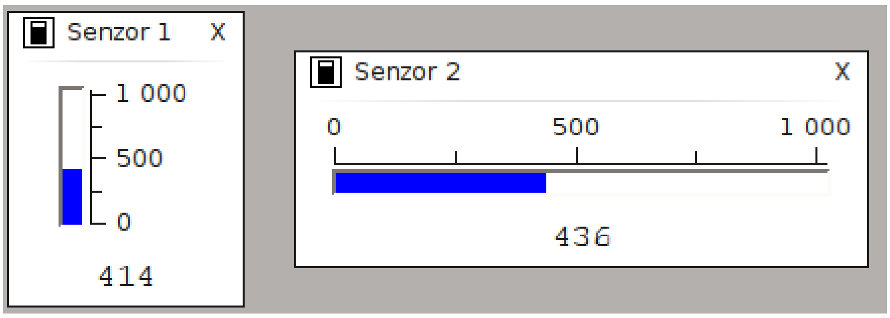


Figure 11: Widget: bar

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

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, same as widget *number*.

2.4 Widget: color

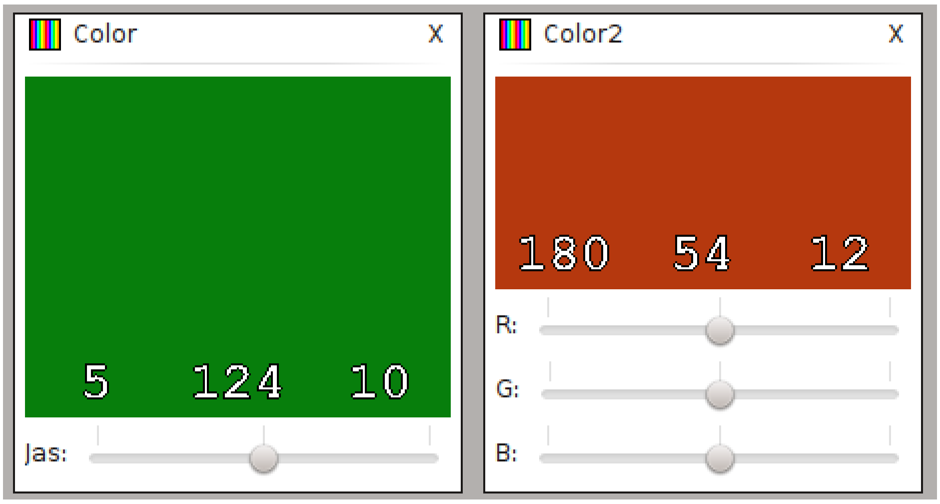


Figure 12: Widget: color

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.

2.5 Widget: graph

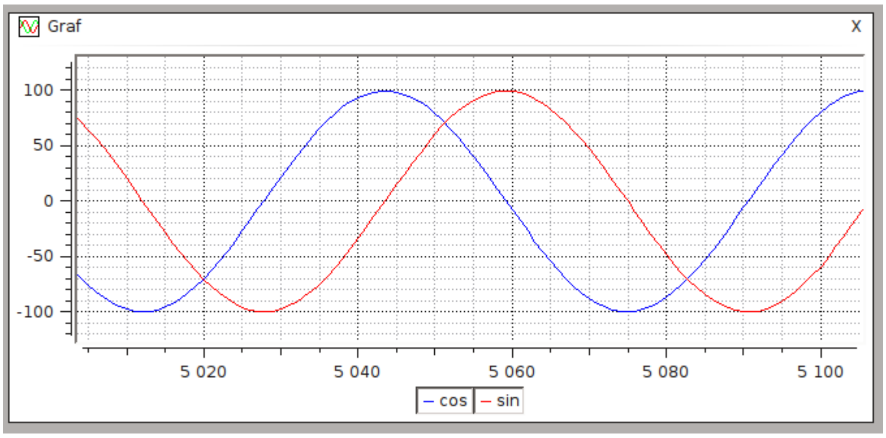


Figure 13: Widget: graph

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

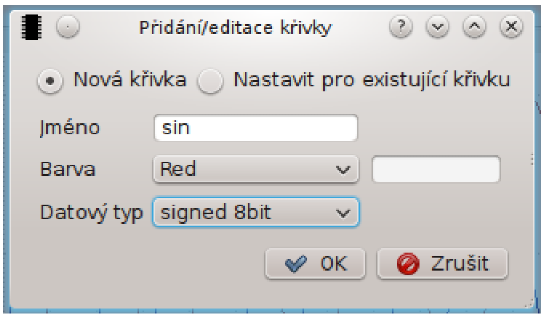


Figure 14: Curve settings dialog

2.6 Widget: script

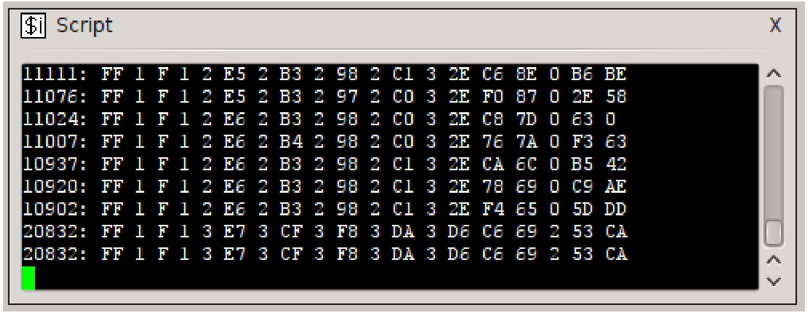


Figure 15: Widget: script

This widget uses user-written script to process data. Script can be written in Python or QtScript[8] (language based on ECMAScript[6], same as JavaScript[7], so JavaScript and QtScript are very similar).

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

Script editor has buil-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 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

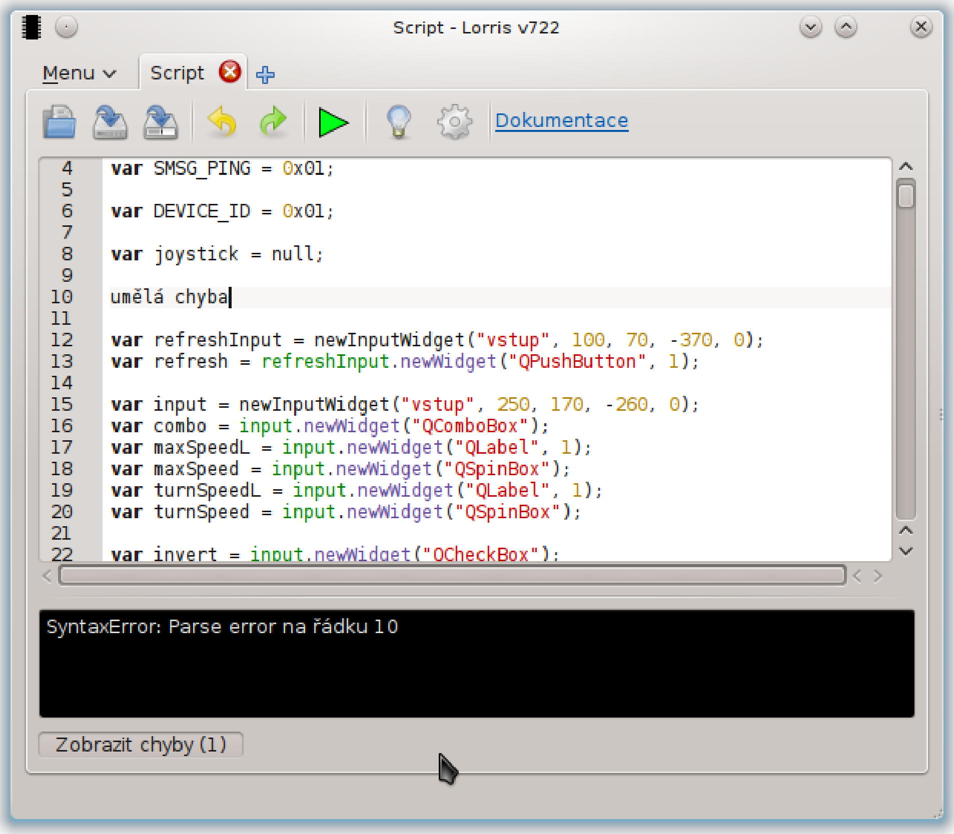


Figure 16: Script editor

2.7 Widget: circle

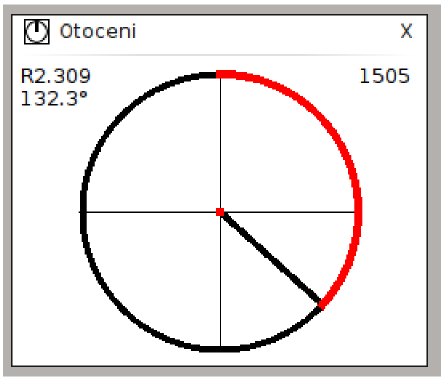


Figure 17: Widget: circle

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 4096).

2.8 Widget: canvas

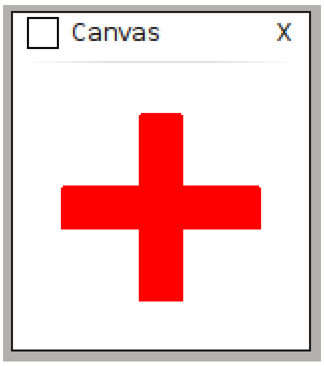


Figure 18: Widget: canvas

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.

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

Example 2: Drawing to canvas

2.9 Widgets button and slider

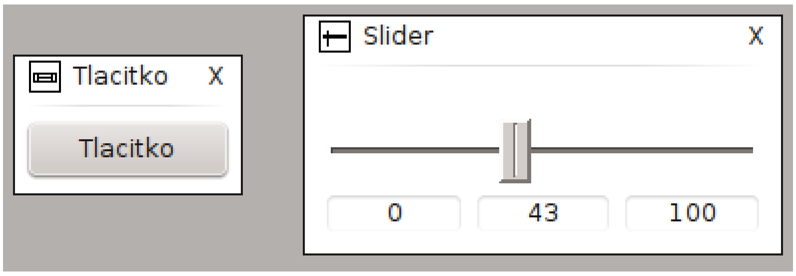


Figure 19: Widgets button and slider

These two widgets are used for interaction with script – callback method in script is invoked in script on button click. In this method user can for example send 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 useing arrow keys.

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

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

Example 3: *Slider* and *button* callbacks

2.10 Widget: input

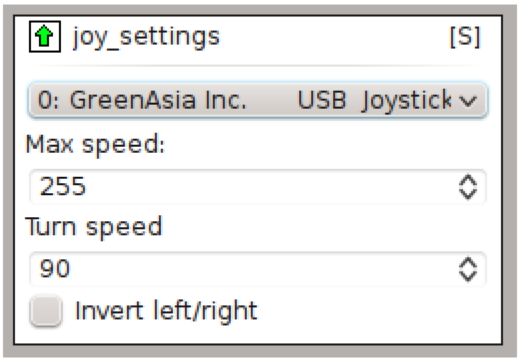


Figure 20: Joystick settings in widget *input*

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 Framerk contains – buttons, slider, text fields, combo boxes and so on. Code sample 5 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*

2.11 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 suports "Unknown value", which is shown when incoming data don't match any defined status.

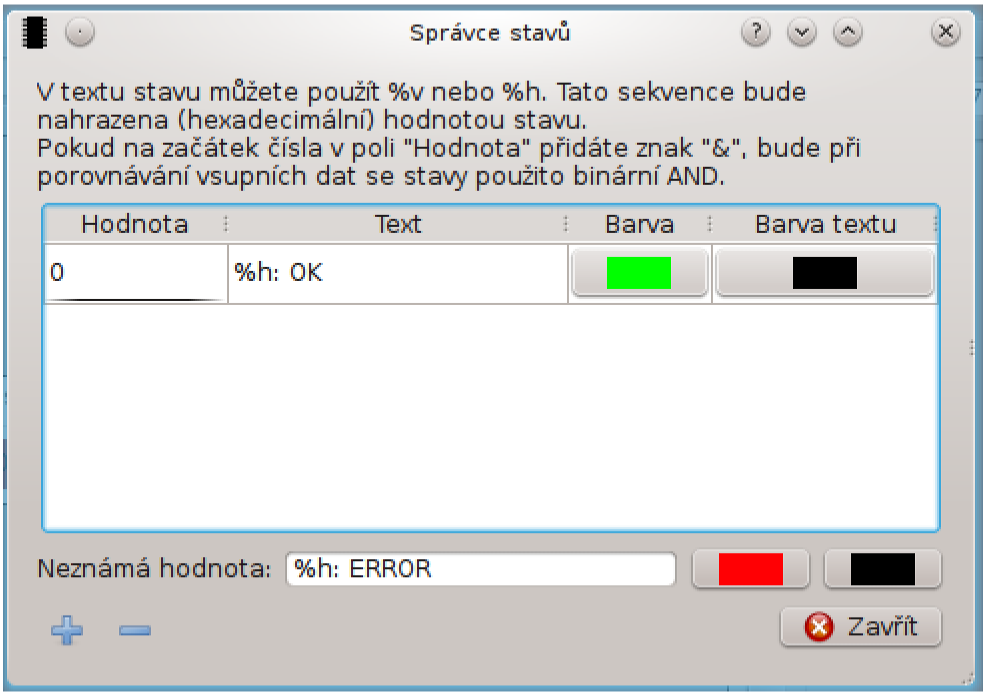


Figure 22: State definitions dialog

2.12 Widget: terminal

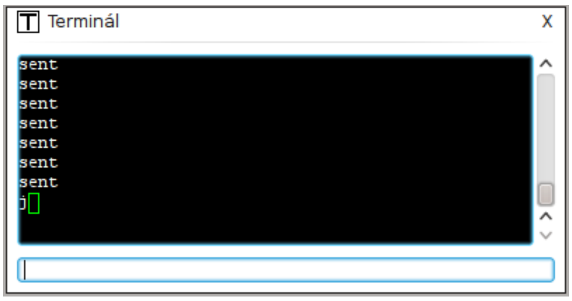


Figure 23: Widget terminál

This widget exists only for convenience of the user, it is widget *script* with preset code which works exactly as terminal (sending keypresses, showing incoming data). User can edit this predefined script, just like it was regular widget *script*.

3 Module: Proxy between serial port and TCP socket

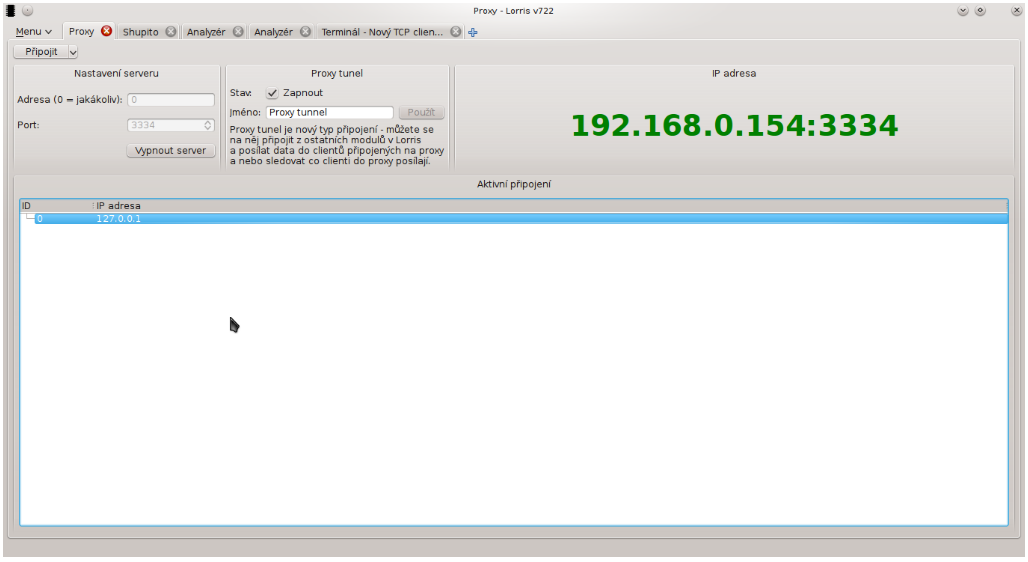


Figure 24: Proxy between serial port and TCP socket

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

3.1 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.

4 Module: programmer

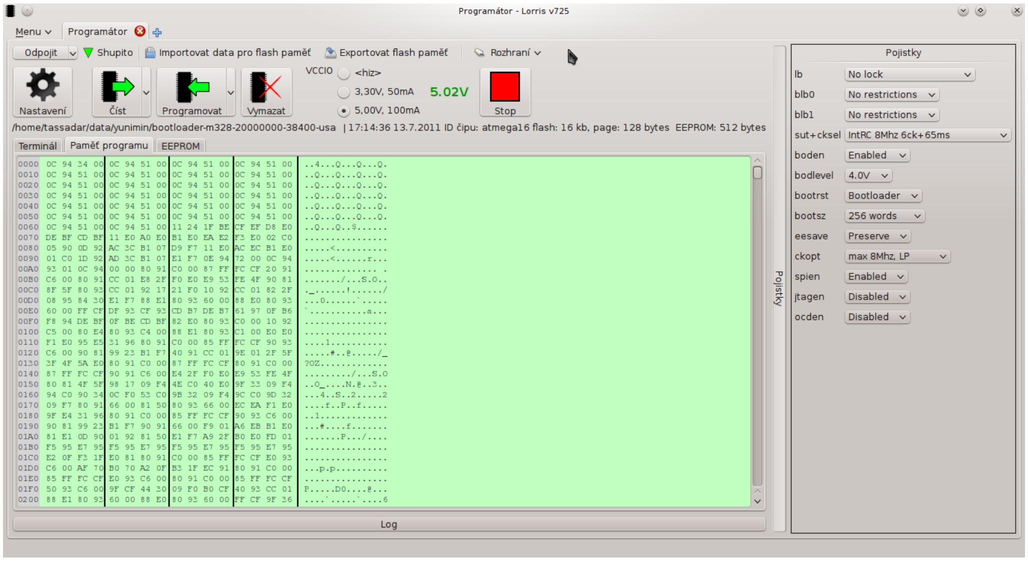


Figure 25: Module programmer

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.

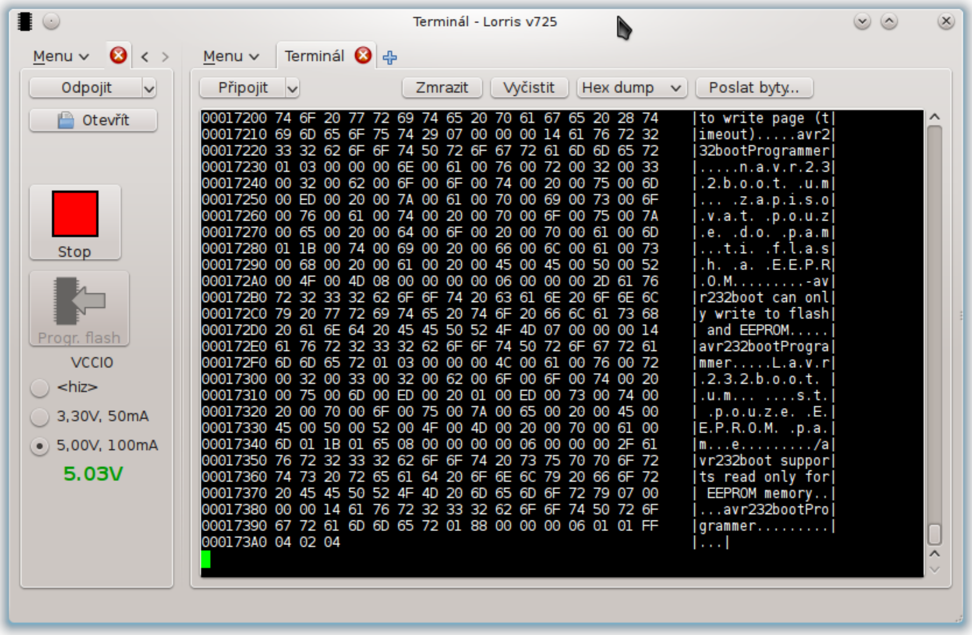


Figure 26: Minimal interface of module *programmer* (left) along with *terminal*

4.1 Shupito programmer

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

4.1.1 RS232 tunnel

Shupito can create tunnel[11] for RS232 interface from programmed chip to computer. Lorris can use this feature – active tunnel creates new virtual connection and other modules can connect to it.

4.2 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. It originally could only program flash memory of the chip (the one where program is stored) and 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.

4.3 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 computer via programmer

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

5 Module: terminal

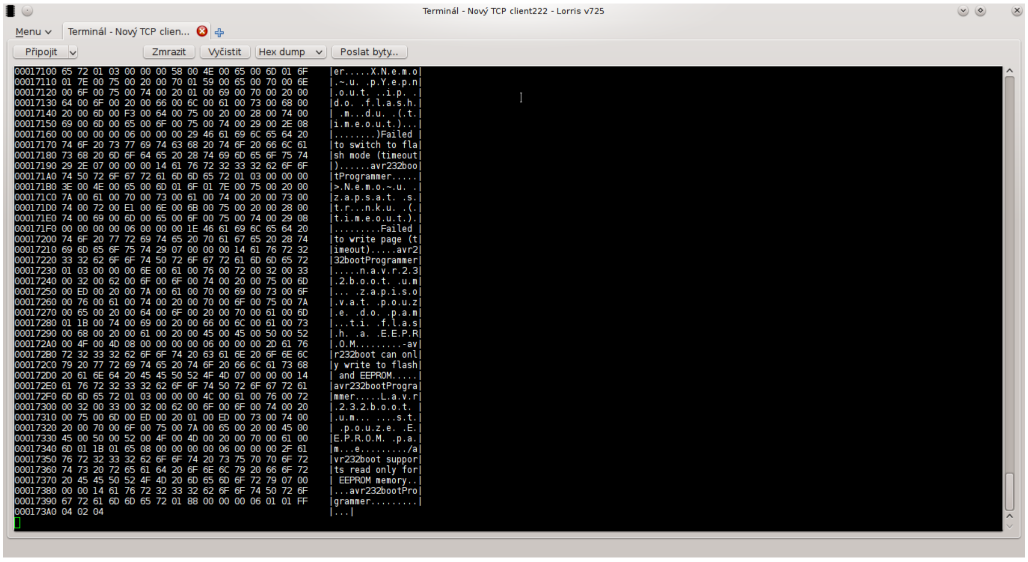


Figure 27: Module terminal

Fundamental tool for every developer, classic 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 sequnce of control characters should be send after return key press and behavior of several control characters (for example if character `\n` should create new line or not).

6 Usage examples

6.1 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 sending color values to computer via RS232 interface.

Solution: I use Shupito to program the chip and it's shupito tunnel to read data from RS232 interface. I connect analyzer module to shupito tunnel and then use widget *color* to show me color measured by the sensor.

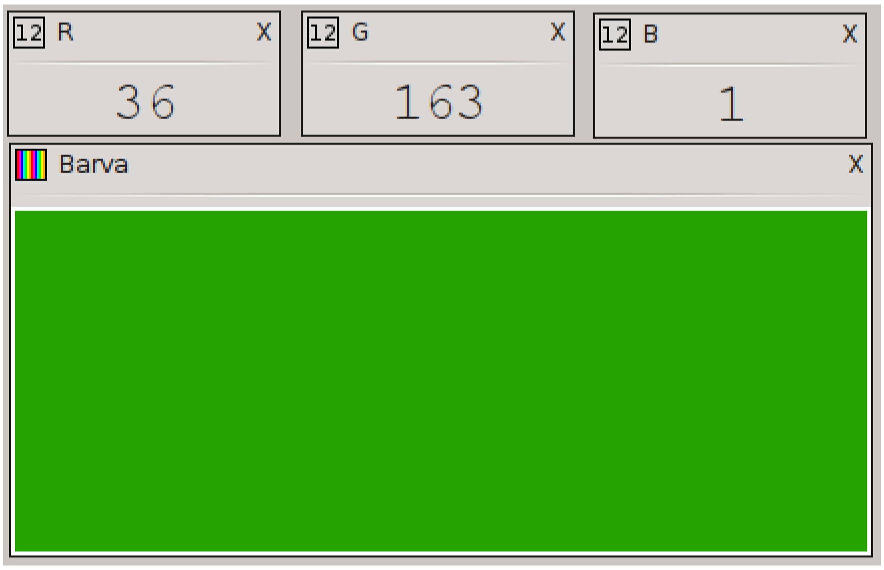


Figure 28: Color in analyzer module

6.2 Encoder testing

Situation: I need to test the precision of magnetic encoders, but they are sending the angle in several bytes which are not in one sequence, so I can't use terminal.

Solution: I don't want to make and program another PCB with chip to process data from encoder, so I connect the encoder to computer and to analyzer module in Lorris. Then I use widget *script* to assemble the angle value and to show it in widget *number*.

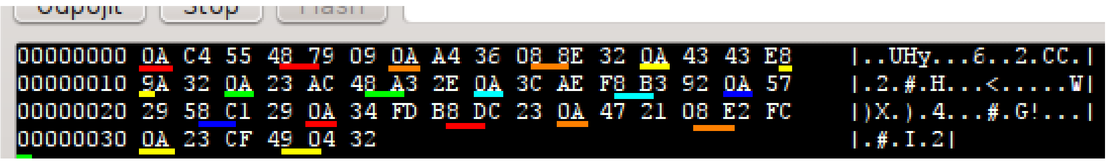


Figure 29: Raw encoder data. Highlighted bytes are parts of the angle.

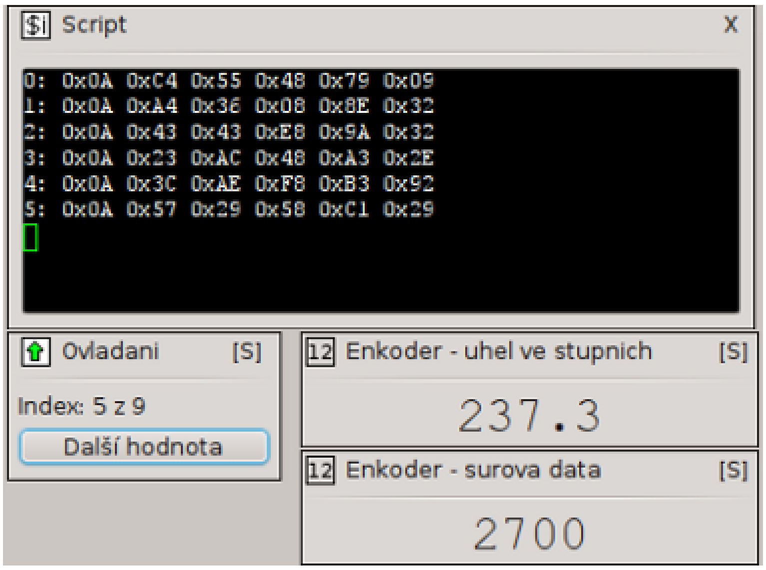


Figure 30: Encoder data processed by analyzer

6.3 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 RS232 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[10] robot. I've attened to *Line Follower Standard* competition on Robotic Day 2012 in Prague[11] with this robot and I've won second place from total of 22 robots.

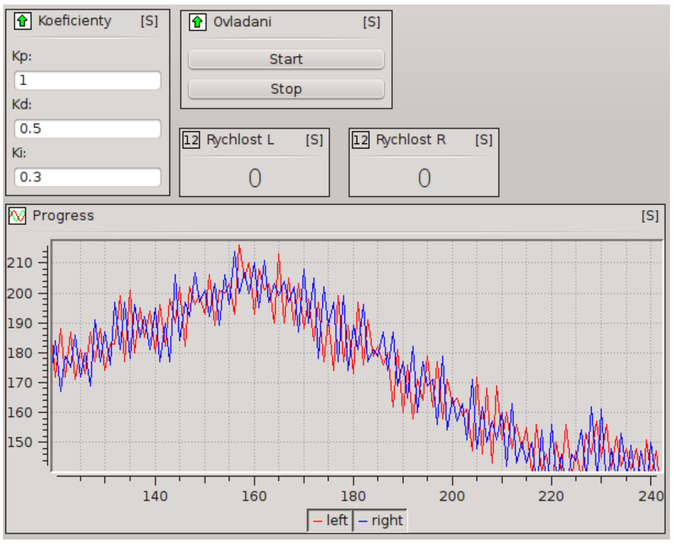


Figure 31: PID regulator tuning

6.4 Developement of robot for Eurobot 2011 competition

I've participated in Eurobot[12] contest in 2011. Goal and game mechanics are different each year, in 2011, the goal was to play something like simplified chess game. Game field was colored chessboard and robots were supposed to move "pawns" (yellow discs) to fields of their color and optionally to build towers from said pawns. Winner was the robot with the highest score. Score was calculated by number of built towers and pawns on fields of correct color. In addition to that, robots must not crash into each other, so they must have some means to detect the opponent (eg. ultrasound range meters). You can find complete rules and result list on Eurobot's website[13].

Our robot was quite simple, but it had 5 ultrasound range meters, two encoders and 5 buttons nevertheless. These sensors make a lot of data, and terminal is not ideal to show all of them.

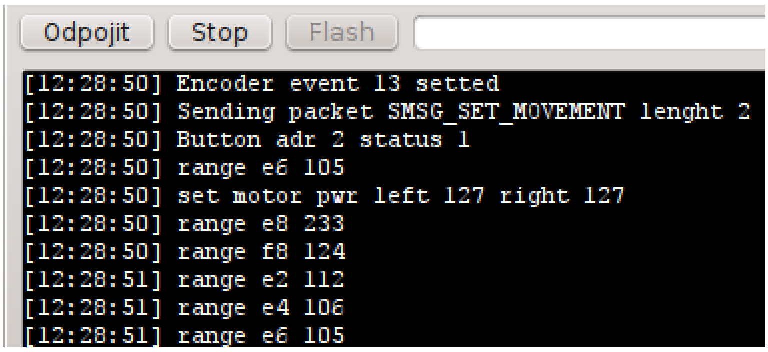


Figure 32: Data from robot in terminal

This experience with programming and debugging of robot was one of the main reasons to make this work. If I'd use Lorris to show data from our robot, it would look like this:

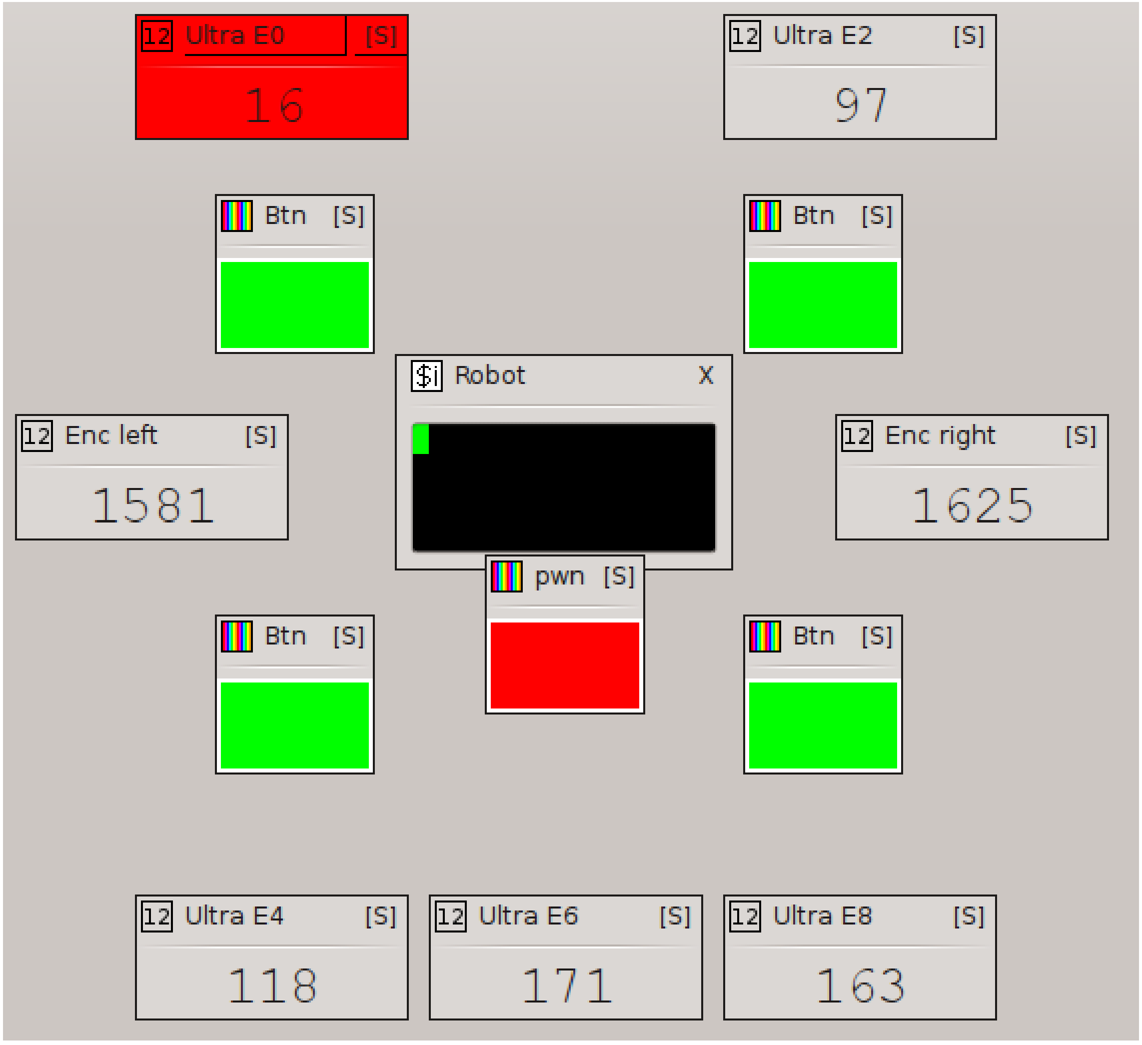


Figure 33: Data simulation in Lorris

All the widgets are positioned same as sensors on robot – 2 ultrasound range meters measure the distance in front and 3 in back; buttons for detecting collisiton with the border are on each corner, button inside the robot checks if robot has pawn inside or not and encoders on both wheels measure the covered distance.

Widget *script* named "Robot" in the center represents robot's body. Widgets *number* named "Ultra E0..8" display distance measured by the ultrasound meters. Widget "Ultra E0" has value lesser than 25 cm, which means robot has to stop in order not to crash into opponent, so it is colored red.

Widgets *color* named "btn" are buttons which detect collision with game field border and "pwn" is button inside robot which is pressed if robot is carrying a pawn. Green means the button is released, red means it is pressed.

And finally, widgets *number* named "Enc left" and "Enc right" display distance measured by encoders on right and left wheel.

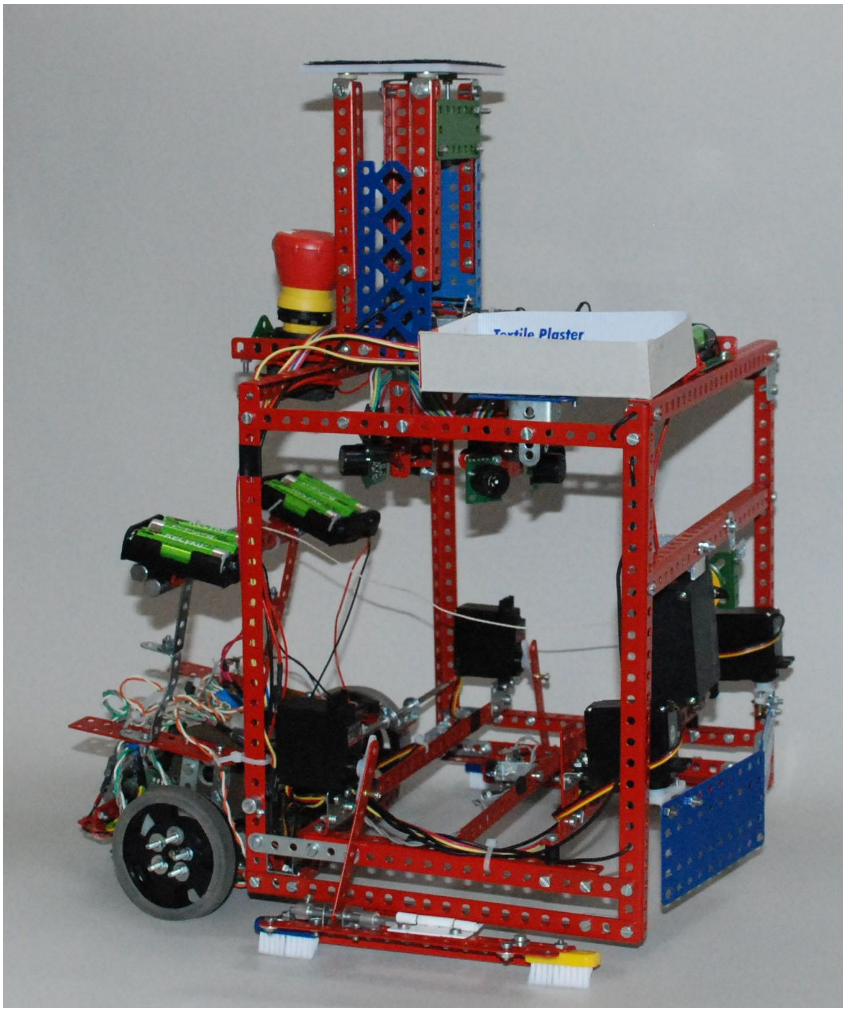


Figure 34: *David* – our robot ended up on the 4th place from total of 11 robots in national Eurobot 2011 competition

7 Joystick support

Lorris supports joystick in module analyzer to for example control robot. At first, I've used SDL[14] 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 replacemnt 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[22] license.

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

8 Android application

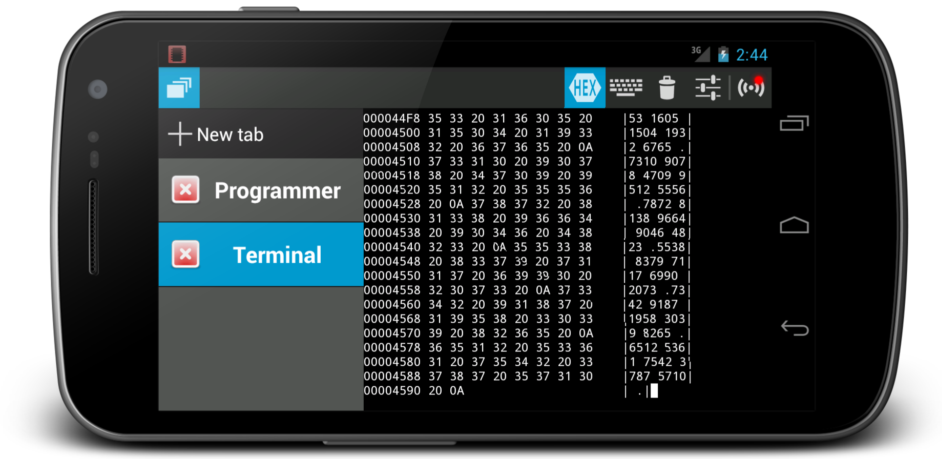


Figure 35: Lorris mobile

Lorris mobile is application for Google Android^{tm} OS and it acts as mobile addition to desktop version of Lorris – it may not have all the features of desktop versio, but helps when you need to quickly correct or debug something 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[17]. 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 36). 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 37).

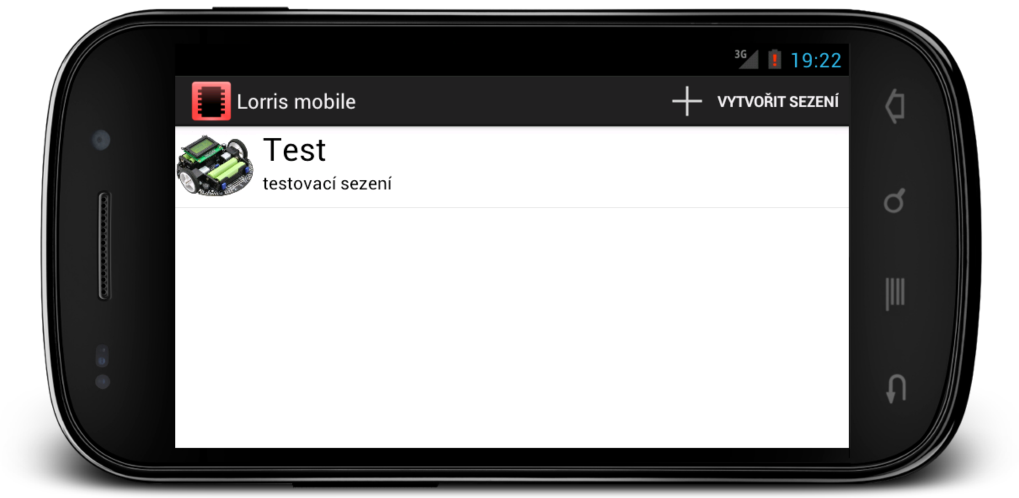


Figure 36: Lorris mobile – session selection

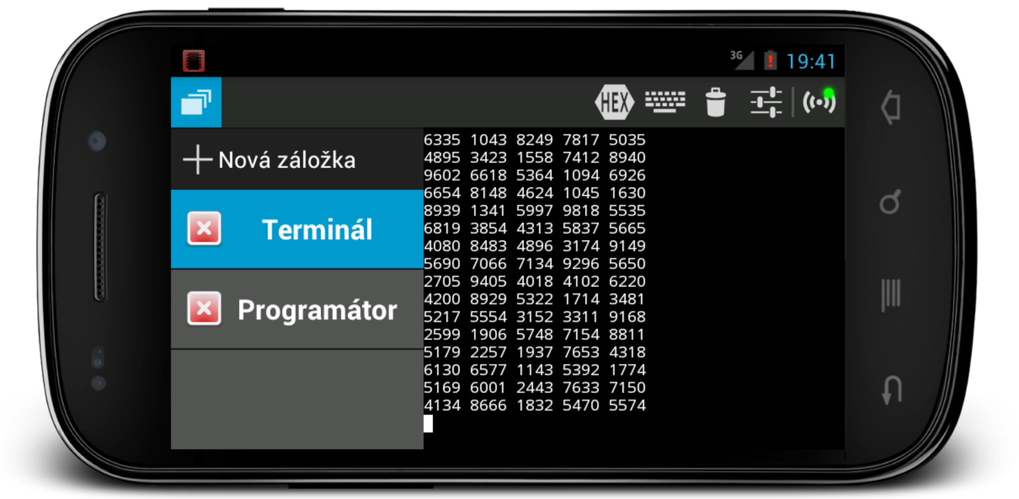


Figure 37: Lorris mobile – switching tabs

8.1 Programmer

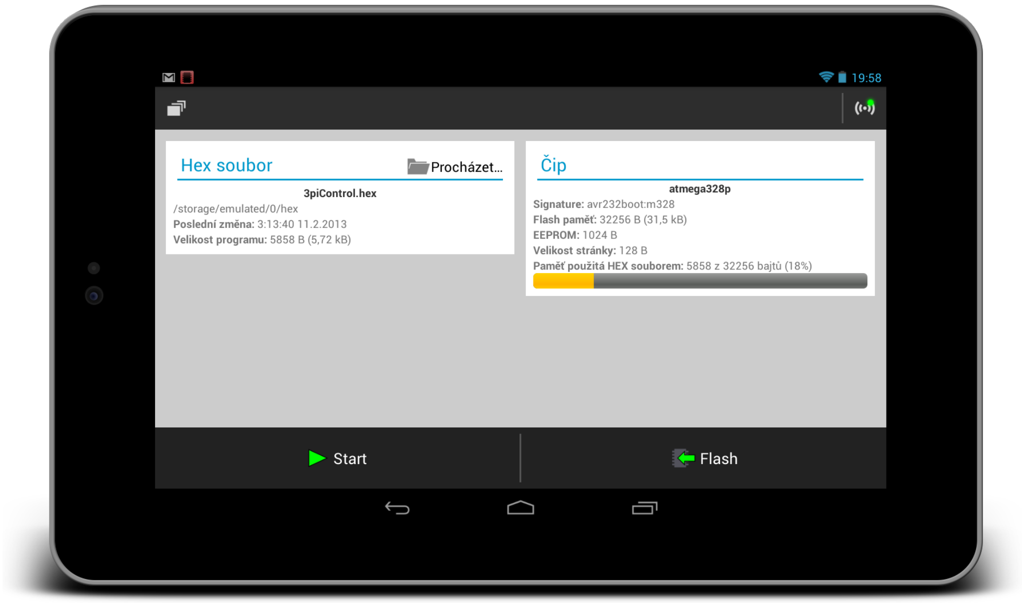


Figure 38: Lorris mobile – programmer

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 parts of native code from desktop Lorris, which means the code is faster and easier to maintain.

8.2 Terminal

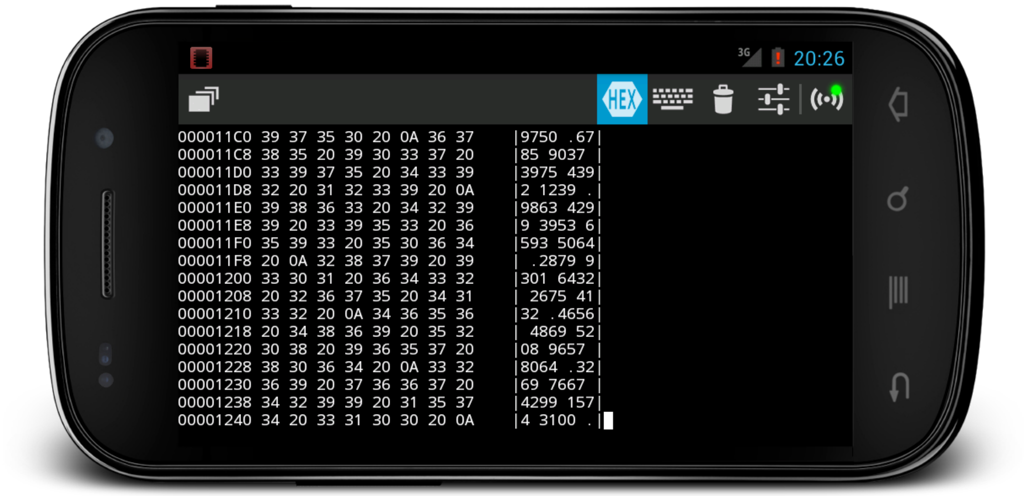


Figure 39: Lorris mobile – terminal

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

Conclusion

Lorris meets all the requirements which have been set:

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

✔ 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

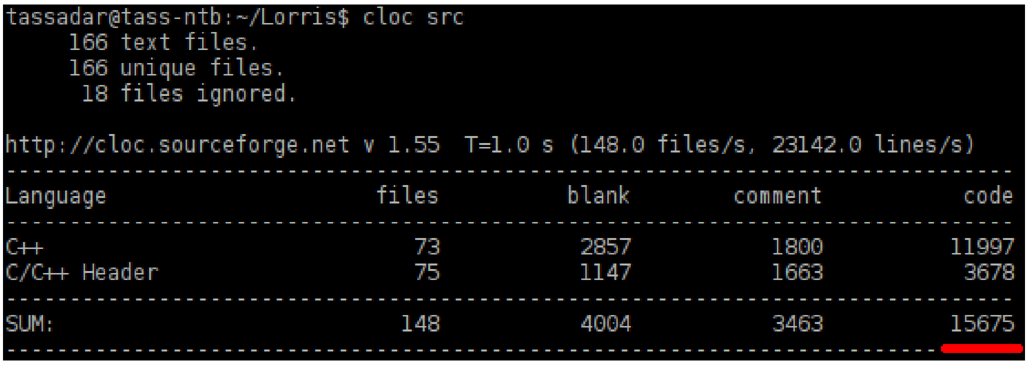
✔ 6. Ability to easily expand program, ideally open-source

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

On top of that, the program greatly outdoes original goals – it 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.

Přestože se jedná o zcela nový software, byl již použit při testování barevného senzoru, ladění PID regulátoru pro robota na sledování čáry a je používán pro ovládání programátoru Shupito. Další možnosti použití jsou uvedeny v kapitole 6.

Aplikace je nadále vyvíjena, mohu prakticky donekonečna přídávat budťo další typy widgetů do modulu Analyzér (například kompas, směrový kříž, ...) nebo celé nové moduly (například ovládání robota pomocí joysticku). Program má v současné době (19.4.2012) asi 15 a půl tisíce řádků kódu (bez knihoven třetích stran). Zdrojové kódy a instruktážní video jsou přiloženy na CD.



```
tassadar@tass-ntb:~/Lorris$ cloc src
     166 text files.
     166 unique files.
      18 files ignored.

http://cloc.sourceforge.net v 1.55  T=1.0 s (148.0 files/s, 23142.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C++                             73           2857           1800          11997
C/C++ Header                    75           1147           1663           3678
-------------------------------------------------------------------------------
SUM:                           148           4004           3463          15675
-------------------------------------------------------------------------------
```

Figure 40: Počet řádků spočítaný programem CLOC[15]

Kromě přidávání dalších vlastností do tohoto programu bych v budoucnu rád vytvořil podobný program (hlavně vlastnosti modulu Analyzér) pro přenosná zařízení (chytrý mobilní telefon či tablet), protože pro tato zařízení žádná taková aplikace v současné době neexistuje a chtěl bych vyzkoušet programování pro tyto platformy (zejména pro Google Android[16]).

PŘÍLOHA A:

Reference k widgetu *script*

Widget *script* umožňuje parsování dat pomocí scriptu, který se píše v QtScriptu, který je založený na standardu ECMAScript, na kterém je založený JavaScript. Jazyk je hodně podobný JavaScriptu a většinou můžete použít jeho referenci. Tento text předpokládá alespoň základní znalost JavaScriptu nebo podobného programovacího jazyku.

- http://en.wikipedia.org/wiki/ECMAScript
- https://qt-project.org/doc/qt-4.8/scripting.html
- http://www.w3schools.com/jsref/default.asp – JS reference

Online dokumentace

Ke scriptu je dostupná automaticky generovaná dokumentace, který obsahuje všechny dostupné metody a příklady scriptů:

- http://technika.junior.cz/docs/Lorris

Základní script

Script by může obsahovat následující funkce (ale nemusí, pokud je nepoužívá):

```
function onDataChanged(data, dev, cmd, index) {
    return "";
}

function onKeyPress(key) {
}

function onRawData(data) {
}

function onWidgetAdd(widget, name) {
}

function onWidgetRemove(widget, name) {
}

function onScriptExit() {
}

function onSave() {
}
```

Example 5: Základní script

`onDataChanged(data, dev, cmd, index)` je volána při změně pozice v datech (tj. když přijdou nová data nebo uživatel pohne posuvníkem historie). Může vracet `string`, který se přidá do terminálu.

- `data` – **pole s Integery** obsahující příchozí data

- `dev` – **Integer** s ID zařízení (může být definováno v hlavičce packetu – pokud není, `dev` se rovná -1)
- `cmd` – **Integer** s ID příkazu (může být definováno v hlavičce packetu – pokud není, `cmd` se rovná -1)
- `index` – **Integer** s indexem packetu v příchozích datech.

`onKeyPress(key)` je volána po stisku klávesy v terminálu.

- `key` – **String** se stisknutou klávesou

`onRawData(data)` je volána kdykoliv příjdou nějáká data.

- `data` – **pole s bajty** obsahují nenaparsovaná data

`onWidgetAdd(widget, name)`
`onWidgetRemove(widget, name)`
jsou volány při přidání/odebrání widgetu z plochy

- `widget` – **objekt** widgetu
- `name` – **String** se jménem widgetu

`onScriptExit()` – tato funkce je volána při ukončení scriptu. Je určena pro ukládání nastavení scriptu.
`onSave()` – tato funkce je volána těsně před uložením dat analyzéru. Je určena pro ukládání nastavení scriptu.

Základní funkce

Jsou dostupné základní javascriptové knihovny (`Math`, `Date`, ...) a samotná Lorris poskytuje další rozšiřující funkce.

- `appendTerm(string)` – přidá do terminálu text.

```
function onKeyPress(key) {
    appendTerm(key); // vypise _key_ do terminalu
}
```

Example 6: Vypsání stisknutých kláves do terminálu

- `clearTerm()` – vyčistí terminál.

```
function onKeyPress(key) {
    if(key == "c")
        clearTerm(); // vycisti terminal
    else
        appendTerm(key); // vypise _key_ do terminalu
}
```

Example 7: Vypsání stisknutých kláves do terminálu a jeho vyčištění po stisku klávesy C

- `sendData(pole Integerů)`
  `sendData(String)` – pošle data do zařizení

```
function onKeyPress(key) {
    sendData(key);
}
```

Example 8: Poslání ASCII kódu stisknuté klávesy

- `throwException(String)` – zobrazí vyskakovací okno s hláškou
- `moveWidget(widget, int x, int y)`
  `resizeWidget(widget, int sirka, int vyska)`
  Tyto funkce přesunou/změní velikost widgetu. `X` a `Y` jsou absolutní

hodnoty na ploše widgetů.

- `newWidget()` – tato funkce potřebuje o něco obsáhlejší popis, který je v následující kapitole

Vytvoření widgetu

Script může vytvořit všechny ostatní typy widgetů a posílat do nich data.

```
newWidget(typ, "jméno");
newWidget(typ, "jméno", šířka, výška);
newWidget(typ, "jméno", šířka, výška, Xoffset, Yoffset);
```

- `typ` – **konstanta**, typ widgetu. Používají se tyto konstanty:

  `WIDGET_NUMBER, WIDGET_BAR, WIDGET_COLOR, WIDGET_GRAPH, WIDGET_SCRIPT, WIDGET_INPUT, WIDGET_TERMINAL, WIDGET_BUTTON, WIDGET_CIRCLE, WIDGET_SLIDER, WIDGET_CANVAS, WIDGET_STATUS`

- `jméno` – **String**, jméno widgetu, zobrazí se v titulku
- `šířka` – **Integer**, šířka widgetu v pixelech. Může být 0, poté se zvolí minimální velikost.
- `výška` – **Integer**, výška widgetu v pixelech. Může být 0, poté se zvolí minimální velikost.
- `Xoffset` – **Integer**, vodorovná vzdálenost v pixelech od levého horního rohu mateřského ScriptWidgetu. Může být 0, widget se poté vytvoří v levém horním rohu aktuálně viditelné plochy.
- `Yoffset` – **Integer**, svislá vzdálenost v pixelech od levého horního rohu mateřského ScriptWidgetu. Může být 0, widget se poté vytvoří v levém horním rohu aktuálně viditelné plochy.

```
var cislo = newWidget(WIDGET_NUMBER,
                "rychlost", 200, 100, -250, 0);

function onDataChanged(data, dev, cmd, index) {
    cislo.setValue(data[0]);
    return "";
}
```

Example 9: Vytvoření widgetu *číslo* a nastavení jeho hodnoty z příchozích dat

Dostupné funkce widgetů

Objekt widgetu je podtřídou třídy z Qt Frameworku QWidget – díky tomu může používat jeho vlastnosti a sloty. Popis vlastností najdete v Qt referenci[13] v kapitole Properties a ve scriptu se používají takto:

```
var cislo = newWidget(WIDGET_NUMBER,
                "rychlost", 200, 100, -250, 0);
cislo.visible = false; // skryti widgetu
```

Example 10: Vytvoření widgetu *číslo* a nastaveni vlastnosti visible

Popis slotů je taktéž v Qt referenci, tentokrát pod kapitolou Public slots. Používají se jako metody:

```
var cislo = newWidget(WIDGET_NUMBER,
                "rychlost", 200, 100, -250, 0);
cislo.setDisabled(true); // znemozneni interakce s widgetem
```

Example 11: Vytvoření widgetu *číslo* a použití slotu

Kromě těchto zděděných vlastností a funkcí má každý typ widgetu své

[13] http://qt-project.org/doc/qt-4.7/qwidget.html#propertySection

vlastní.

Widget číslo

- `setValue(Integer nebo double)` – Nastaví hodnotu widgetu
- `setFormula(String)` – nastaví výraz pro přepočítávání hodnoty
- `setDataType(konstanta)` – Nastaví typ vstupu. Konstanty:

  `NUM_UINT8, NUM_UINT16, NUM_UINT32, NUM_UINT64,`
  `NUM_INT8, NUM_INT16, NUM_INT32, NUM_INT64,`
  `NUM_FLOAT, NUM_DOUBLE`

```
var cislo = newWidget(WIDGET_NUMBER,
                "test cislo", 200, 100, -250, 0);
cislo.setValue(40);
cislo.setFormula("%n-100");
...
cislo.setValue(3.14);
```

Example 12: Nastavení hodnoty widgetu *číslo*

Widget sloupcový bar

- `setValue(Integer)` – Nastaví hodnotu widgetu
- `setRange(Integer min, Integer max)` – Nastaví minimální a maximální hodnotu widgetu
- `setRotation(Integer)` – Nastaví rotaci sloupce. 0 pro svislou, 1 pro vodorovnou
- `setFormula(String)` – nastaví výraz pro přepočítávání hodnoty

- getMin(), getMax(), getValue() – vrací minimální, maximální a aktualní hodnotu

```
var bar = newWidget(WIDGET_BAR, "test bar");
bar.setRange(0, 100); // rozmezi hodnot 0 az 100
bar.setValue(45); // nastaveni hodnoty na 45
bar.setRotation(1); // otoceni na vodorovno
```

Example 13: Nastavení hodnot widgetu *sloupcový bar*

Widget barva

- setValue(Integer r, Integer g, Integer b)
  setValue(String barva)
  setValue(Integer rgb)
  setValueAr(pole integerů)
  – Nastaví barvu ve widgetu.
- setColorType(konstanta) – Nastavý formát vstupu. Konstanty:

  COLOR_RGB_8, COLOR_RGB_10, COLOR_RGB_10_UINT,
  COLOR_GRAY_8, COLOR_GRAY_10

```
var clr = newWidget(WIDGET_COLOR, "test barva");
clr.setValue(255, 255, 0);
clr.setColorType(COLOR_RGB_10);
clr.setValue(543, 1023, 200);
```

Example 14: Nastavení hodnot widgetu *barva*

Widget graf

Tento widget se od ostatních poměrně výrazně liší – je třeba nejdříve vytvořit křivku až té nastavovat hodnoty. Funkce samotného widgetu graf jsou tyto:

- `addCurve(String jméno, String barva)` – Vytvoří a vrátí novou křivku. `barva` může být buďto html název (např. red, blue) nebo HTML hex zápis (např. #FF0000)

- `removeCurve(String jméno)`
  `removeAllCurves()`
  Odebrání jedné nebo všech křivek

- `setAxisScale(bool proX, double min, double max)` – Nastaví měřítko os. `proX` je **true** pokud nastavujete měřítko osy x

- `updateVisibleArea()` – Přesune pohled na nejvyšší hodnotu osy x

`addCurve(String jméno, String barva)` vrátí křivku, která má tyto funkce:

- `addPoint(Integer index, double hodnota)` – Vloží bod křivky. `index` určuje pořadí bodů (bod s indexem 0 bude vždy před bodem s indexem 50, i když bude vložen až po něm). Pokud bod se stejným indexem už existuje, je jeho hodnota změněna

- `clear()` – Smaže všechny body křivky

```
var graf = newWidget(WIDGET_GRAPH, "graf", 400, 250, -420, 0);
graf.setAxisScale(false, -105, 105); // meritko osy y
graf.setAxisScale(true, 0, 200); // meritko osy x

// vytvoreni krivky sin
var sin = graf.addCurve("sin", "blue");

// pridani bodu do krivky sin
var sinVal = 0;
for(var i = 0; i < 500; ++i) {
    sin.addPoint(i, Math.sin(sinVal)*100);
    sinVal += 0.1;
}
// presunuti na posledni hodnotu krivky
graf.updateVisibleArea();
```

Example 15: Zobrazení křivky funkce sinus ve widgetu *graf*

Widget vstup

Tento widget lze vytvořit pouze ze scriptu a umí zobrazit a ovládat většinu Qt widgetů[14], například tlačítko (`QPushButton`), zaškrtávací políčko (`QCheckBox`) či textové políčko (`QLineEdit`). Dokumentace k těmto widgetům je v Qt referenci, opět můžete používat vlastnosti (Properties) a funkce (Public slots). Funkce widgetu *vstup*:

- `newWidget(String jméno, Integer roztahování = 0)` – Vytvoří a vrátí nový QWidget. `jméno` musí být jméno třídy widgetu, například QPushButton, QCheckBox nebo QLineEdit. `roztahování` značí jak moc se bude widget roztahovat oproti ostatním.

[14] http://qt-project.org/doc/qt-4.7/widgets-and-layouts.html

- `removeWidget(Objekt widget)` – Odstraní widget vrácený voláním `newWidget`.
- `clear()` – Odstraní všechny widgety.
- `setHorizontal(bool horizontal)` – Nastaví způsob uspořádání widgetů (vedle sebe nebo pod sebou).

```
var vstup = newWidget(WIDGET_INPUT,
                "test vstupu", 150, 100, -160, 0);
var label = vstup.newWidget("QLabel", 1);

// zarovnani textu na stred.
// 0x0080 a 0x0004 jsou konstanty Qt Frameworku
// Qt::AlignHCenter a Qt::AlignVCenter
label.alignment = 0x0080 | 0x0004;

// nastaveni textu
label.text = "Testovaci popisek";
```

Example 16: Widget *vstup* – vytvoření QLabel

Widget vytvořený tímto příkladem vypadá takto:

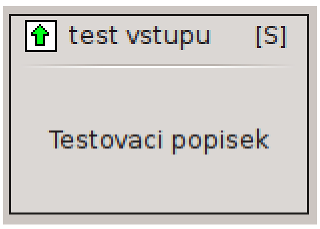


Figure 41: Widget *vstup* – vytvoření QLabel

QtScript podporuje i využití principu signálů a slotů, díky tomu lze ve scriptu reagovat například na stisknutí tlačítka.

```
var vstup = newWidget(WIDGET_INPUT,
                "test vstupu", 150, 100, -160, 0);

var rychlost = vstup.newWidget("QLineEdit");
rychlost.text = "100";

var btn = vstup.newWidget("QPushButton", 1);
btn.text = "Nastavit";

function posliRychlost() {
    var speed = parseInt(rychlost.text);
    sendData(new Array(speed));
    appendTerm("Rychlost " + speed + "odeslana\n");
}
// Pripojeni signalu "clicked" na slot posliRychlost()
btn.clicked.connect(posliRychlost);
```

Example 17: Widget *vstup* – tlačítko

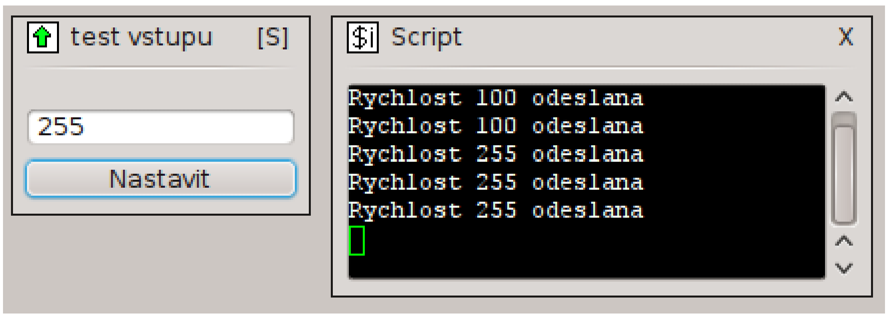


Figure 42: Widget *vstup* – tlačítko

Widget kolo

- `setValue(číslo)` – Nastaví zobrazený úhel
- `setClockwise(bool clockwise)` – Nastaví jestli se úhel počítá po nebo proti směru hodinových ručiček
- `setAngType(konstanta, min, max` – Nastavý vstupní formát. Konstanty: `ANG_RAD`, `ANG_DEG`, `ANG_RANGE`

```
var c = newWidget(WIDGET_CIRCLE, "kolo", 200, 200, -210, 0);

c.setAngType(ANG_DEG); // nastaveni vstupu na stupne
c.setValue(270);
```

Example 18: Nastavení hodnot widgetu *kolo*

Widget plátno

- `clear()` – Vymaže vše, co je ve widgetu namalované
- `setBackground(String barva)` – Nastaví barvu pozadí
- `drawLine(int x1, int y1, int x2, int y2)` – Nakreslí čáru.
- `drawLine(int x, int y)` – Nakreslí čáru. Začátek je v bodě, kde končí předchozí nakreslená čára (nebo v [0,0] pokud ještě nebyla žádná nakreslená).
- `drawRect(int x, int y, int sirka, int vyska)` – Nakreslí obdélník.
- `drawEllipse(int x, int y, int sirka, int vyska)` – Nakreslí elipsu
- `drawEllipse(int x, int y, int polomer)` – Nakreslí kruh
- `setLineSize(int tloušťka)` – Tloušťka čáry, kterou se prvky kreslí.
- `setLineColor(String barva)` – Barva čáry, kterou se prvky kreslí.

- `setFillColor(String barva)` – Barva výplně obdélníků, elips a kruhů

```
var c = newWidget(WIDGET_CANVAS, "Canvas", 140, 170, -150, 0);
c.setLineColor("red");
c.setFillColor("red");

c.drawRect(55, 10, 20, 110);
c.drawRect(10, 55, 110, 20);
```

Example 19: Nakreslení kříže ve widgetu *plátno*

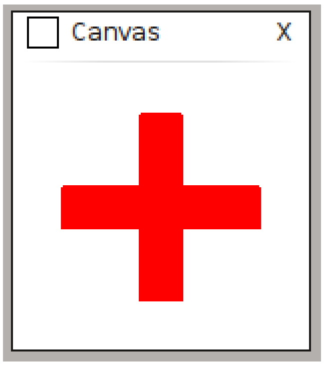


Figure 43: Nakreslení kříže ve widgetu *plátno*

Widget status

- `addStatus(int id, bool bitMaska, String text, String barvaPozadí, String b`
  – Přidá nový status. `bitMask` určuje, zda se má použít přímé porovnání s hodnotou `id` nebo bitový operátor `AND`.
- `removeStatus(int id, bool bitMaska` – Odebere status
- `setValue(Integer)` – Nastaví vstupní hodnotu
- `getValue()` – Vrátí aktuální hodnotu

- `showStatusManager()` – Vyvolá dialog pro správu stavů ve widgetu

```
var s = newWidget(WIDGET_STATUS, "stav", 0, 0, 200, 0);

s.addStatus(2, false, "Porucha", "orange", "black");
s.setValue(2);
```

Example 20: Ovládání widgetu *status* ze scriptu

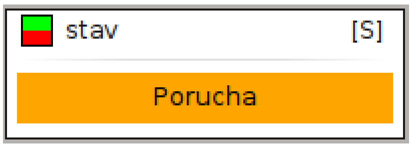


Figure 44: Ovládání widgetu *status* ze scriptu

Widget tlačítko

Kromě funkcí, které tento widget má je také možné nastavit metodu která se provede po kliknutí, stačí ve scriptu vytvořit metodu `JménoWidgetu_clicked()`.

- `setButtonName(String text)` – Nastaví text na tlačítku
- `setShortcut(String zkratka)` – Nastaví klávesovou zkratku pro tlačítko
- `setColor(String barva)` – Nastaví barvu tlačítka
- `setTextColor(String barva)` – Nastaví barvu textu na tlačítku

```
var t = newWidget(WIDGET_BUTTON, "tlacitko", 0, 0, 200, 0);

t.setButtonName("Pokus");
t.setShortcut("Ctrl+H");
t.setColor("white");

function tlacitko_clicked() {
    appendTerm("Tlacitko stisknuto!\n");
}
```

Example 21: Nastavení widgetu *tlačítko* ze scriptu

Widget slider

Kromě funkcí, které tento widget má je také možné nastavit metodu která se provedou při různých změnách stavu slideru. Mají tvar `JménoWidgetu_jmenoZmeny()`, v následujícím příkladě má widget jméno Slider.

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

function Slider_minimumChanged() {
    appendTerm("nove minimum: " + Slider.getMin() + "\n");
}

function Slider_maximumChanged() {
    appendTerm("nove maximum: " + Slider.getMax() + "\n");
}

function Slider_typeChanged() {
    appendTerm("Typ vstupu zmene na " +
        (Slider.isInteger() ? "Integer" : "Double") + "\n");
}

function Slider_orientationChanged() {
    appendTerm("orientace zmenena na " +
        Slider.getOrientation() + "\n");
}
```

Example 22: Funkce, ktere jsou volány při změně stavu widgetu *slider*

- `setType(bool double)` – Nastaví typ hodnot které widget nastavuje (celá nebo desetinná čísla).

- `setMin, setMax, setValue (číslo)` – Nastaví minimální, maximální a aktuální hodnotu
- `getMin(), getMax(), getValue()` – Vrátí minimální, maximální a aktální hodnotu

```
var s = newWidget(WIDGET_SLIDER, "Slider", 0, 0, 300, 0);

s.setType(true); // desetinna cisla
s.setMax(6.28);
s.setValue(3.14);

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

Example 23: Ovládání wigetu *slider* ze scriptu

Ukládání dat scriptu

Na uložení hodnot použitých ve scriptu (například nastavení) je připravena třída `ScriptStorage`. Ve scriptu je dostupná jako objekt `storage` a má tyto funkce:

- `clear()` – Vymaže všechna uložená dataPass
- `exists(String klíč)` – Vrátí `true` pokud hodnota s tímto klíčem existuje.
- `setXXXX(String klíč, XXX hodnota)`
  `getXXXX(String klíč, XXX pokudKlíčNeexistuje)`
  `setYYYYArray(String klíč, PoleYYY hodnota)`
  `getYYYYArray(String klíč, PoleYYY pokudKlíčNeexistuje)`
  Funkce pro uložení a načtení hodnoty.

  `XXX` typy mohou být `Bool, Uint32, Int32, Float` nebo `String`. Pole `YYYY` může být s prvky typu `UInt32, Int32` nebo `Float`. Druhý parametr u `getXXX` metod je výchozí hodnota, která se vrátí pokud klíč neexistuje.

```
var s = newWidget(WIDGET_SLIDER, "Slider", 0, 0, 300, 0);
s.setType(true); // desetinna cisla
s.setMax(6.28);

// Nacte hodnotu, ktera byla pred tim ulozena v metode save()
var ulozene = storage.getFloat("hodnotaPosuvniku", 3.14);
s.setValue(ulozene);

// Nacte pokusne pole cisel ulozene v metode save()
var pokus = storage.getInt32Array("pokusnePole", new Array());
appendTerm("Ulozene pole: " + pokus + "\n");

function onSave() {
    save();
}

function onScriptExit() {
    save();
}

function save() {
    storage.setFloat("hodnotaPosuvniku", Slider.getValue());

    var pokus = new Array(4, 8, 15, 16, 23, 42);
    storage.setInt32Array("pokusnePole", pokus);
}
```

Example 24: Ukládání dat scriptu

Přístup k joysticku

Nejříve několik globálních metod pro práci s joysticky:

- `getJoystickNames()` – Vráti pole Stringů se jmény přípojených joysticků.
- `getJoystickIds()` – Vráti pole Integerů s ID připojených joysticků. Indexy v tomto poli korespondují s polem z funkce `getJoystickNames()`, tj. ID na pozici 0 patří ke jménu na pozici 0.
- `getJoystick(int id)` – Otevře joystick s daným ID a vrací object `Joystick` neno NULL pokud nebylo možné joystick otevřit.
- `closeJoystick(Joystick)` – Zavře a uvolní objekt joysticku

Objekt `joystick` pak má následující metody:

- `getId()` – Vrátí ID joysticku
- `getNumAxes()`
  `getNumButtons()` – Vrátí počet os nebo tlačítek
- `getAxisVal(int osa)` – Vrátí aktuální hodnotu osy joysticku jako číslo mezi -32768 a 32767. Parametr `osa` je číslo od 0 do `getNumAxes()-1`.
- `getButtonVal(int tlačítko)` – Vrátí aktuální hodnotu tlačítka jako číslo 0 (uvolněno) nebo 1 (stisknuto). Parametr `tlačítko` je číslo od 0 do `getNumButtons()-1`.

Kromě toho má joystick také dva signály, na které se můžete ve scriptu napojit:

- `axesChanged(Pole integerů)` – Volá se když se hodnota některé z os změní. V poli jsou indexy os které se změnily.
- `buttonChanged(int tlačítko, int stav)` – Volá se když se zmení stav tlačítka. Parametr `tlačítko` je index tlačítka a `stav` je číslo 0 nebo 1.

```
// Pokusi se otevrit prvni dostupny joystick
var ids = getJoystickIds();
var joy = getJoystick(ids[0]);

if(joy) {
    // pripojeni na signaly
    joy.axesChanged.connect(axesChanged);
    joy.buttonChanged.connect(buttonChanged);

    appendTerm("ID joysticku: " + joy.getId() + "\n");
    appendTerm("Pocet os: " + joy.getNumAxes() + "\n");
    appendTerm("Pocet tlacitek: " +
            joy.getNumButtons() + "\n");
}

function axesChanged(axes) {
    for(var i = 0; i < axes.length; ++i) {
        var hodnota = joy.getAxisVal(axes[i]);
        appendTerm("Osa " + axes[i] + ": " + hodnota + "\n");
    }
}

function buttonChanged(id, state) {
    appendTerm("Tlacitko " + id + ", stav: " + state + "\n");
}
```

Example 25: Otevření joysticku a čtení jeho hodnot

Pár věcí, na které je třeba při programování myslet

- Widgety vytvořené ze scriptu se neukládájí do datového souboru – po načtení se vytvoří znovu, bez dat.
- Stav proměnných ve scritptu se zatím neukládá do souboru.
- Po stisknutí Ok nebo Použít v dialogu nastavení scritpu se script načte znovu – staré widgety se smažou a vytvoří nové, bez dat.
- Jazyk nemá žádné pojistky proti špatnému kódu – pokud ve scritpu bude nekonečná smyčka, Lorris prostě zamrzne

PŘÍLOHA B:

Knihovny třetích stran

- **Qwt**[18] je knihovna pro Qt Framework obsahující tzv. widgety pro aplikace technického charakteru – grafy, sloucové ukazatele, kompasy a podobně. Ve svojí práci zatím z této knihovny používám pouze graf (v modulu analyzéru).

- **QExtSerialPort**[19] poskytuje připojení k sériovému portu a také dokáže vypsat seznam nalezených portů v počítačí.

- **QHexEdit2**[20] je hex editor použitý v modulu programátoru Shupito na zobrazování obsahu paměti. V této knihovně jsem upravoval několik málo drobností, týkajících se především vzhledu.

PŘÍLOHA C:

Licence

Lorris je dostupný pod licencí GNU GPLv3[21], licence použitých programů a knihoven jsou následující:

- **Qt Framework** je distribuován pod licencí GNU LGPLv2.1[22]

- **Qwt** je distribuováno pod Qwt license[23], která je založená na GNU LGPLv2.1

- **QExtSerialPort** je distribuován pod The New BSD License[24]

- **QHexEdit2** je distribuován pod licencí GNU LGPLv2.1

- **avr232client** je distribuován pod licencí Boost Software License v1.0[25]

Všechny tyto licence umožňují svobodné používání a šíření kódu.

PŘÍLOHA D:

References

[1] *SerialChart* – Analyse and chart serial data from RS-232 COM ports
http://code.google.com/p/serialchart/
(Stav ke dni 6.3.2012)

[2] *WinWedge* – RS232 data collection software
http://www.taltech.com/products/winwedge/
(Stav ke dni 6.3.2012)

[3] *Advanced Serial Data Logger*
http://www.aggsoft.com/serial-data-logger.htm
(Stav ke dni 6.3.2012)

[4] *StampPlot Pro* – Graphical Data Acquisition and Control
http://www.selmaware.com/stampplot/index.htm
(Stav ke dni 6.3.2012)

[5] *Qt* – Cross–platform application and UI framework
http://qt.nokia.com/
(Stav ke dni 6.3.2012)

[6] *Debian Linux* – The Universal Operating System
http://www.debian.org/
(Stav ke dni 6.3.2012)

[7] *GitHub* – Social Coding
https://github.com
(Stav ke dni 6.3.2012)

[8] *Making Applications Scriptable*
http://qt-project.org/doc/qt-4.8/scripting.html
(Stav ke dni 13.3.2012)

[9] *avr232client*
http://technika.junior.cz/trac/wiki/avr232client
(Stav ke dni 6.3.2012)

[10] *Pololu 3pi Robot*
http://www.pololu.com/catalog/product/975
(Stav ke dni 18.4.2012)

[11] *Robotický den 2012*
http://www.roboticday.org/cz/
(Stav ke dni 18.4.2012)

[12] *Eurobot*
http://www.eurobot.cz/
(Stav ke dni 6.3.2012)

[13] *Eurobot 2011*
http://www.eurobot.cz/eurobot2011.php
(Stav ke dni 6.3.2012)

[14] *SDL* – Simple Directmedia Layer
http://www.libsdl.org/
(Stav ke dni 13.2.2013)

[15] *CLOC* – Count Lines of Code
http://cloc.sourceforge.net/
(Stav ke dni 8.3.2012)

[16] *Google Android* – Operační systém pro chytré telefony
http://www.android.com/
(Stav ke dni 8.3.2012)

[17] *Google Play Store* – Obchod s aplikacemi pro OS Android
http://play.google.com/store
(Stav ke dni 14.2.2013)

[18] *Qwt* – Qt Widgets for Technical Applications
http://qwt.sourceforge.net/
(Stav ke dni 6.3.2012)

[19] *QExtSerialPort* – Qt interface class for old fashioned serial ports
http://code.google.com/p/qextserialport/
(Stav ke dni 6.3.2012)

[20] *QHexEdit2* – Binary Editor for Qt
http://code.google.com/p/qhexedit2/
(Stav ke dni 6.3.2012)

[21] *GNU General Public License v3*
http://gplv3.fsf.org/
(Stav ke dni 6.3.2012)

[22] *GNU Lesser General Public License v2.1*
http://www.gnu.org/licenses/lgpl-2.1.html
(Stav ke dni 6.3.2012)

[23] *Qwt license*
http://qwt.sourceforge.net/qwtlicense.html
(Stav ke dni 6.3.2012)

[24] *The New BSD License*
http://www.opensource.org/licenses/bsd-license.php
(Stav ke dni 6.3.2012)

[25] *The Boost Software License*
http://www.boost.org/users/license.html
(Stav ke dni 6.3.2012)

PŘÍLOHA E:

List of Figures