

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

**LORRIS TOOLBOX**  
**Sada nástrojů pro vývoj**  
**a řízení robotů**

Vojtěch Boček

Brno 2013

# STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor SOČ: 18. Informatika

## LORRIS TOOLBOX

Sada nástrojů pro vývoj a řízení  
robotů

**Autor:** Vojtěch Boček

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

**Konzultant:** 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 příloženém 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:

## Poděkování

Děkuji Jakubu Streitovi za rady, obětavou pomoc, velkou trpělivost a podnětné připomínky poskytované během práce na tomto projektu, Martinu Vejnárovi za informace o programátoru Shupito, panu profesorovi Mgr. Miroslavu Burdovi za velkou pomoc s prací a v neposlední řadě Bc. Martinu Foučkovi za rady a pomoc při práci s Qt Frameworkem. Dále děkuji organizaci DDM Junior za poskytnutí podpory.

Tato práce byla vypracována za finanční podpory JMK a JCMM.



**Jihomoravský kraj**



## Anotace

Tato práce popisuje sadu nástrojů pro vývoj a ovládání libovolného zařízení schopného komunikovat po sériové lince nebo TCP socketu.

Aplikace zastřešuje několik modulů. Každý modul je určen pro specifickou činnost – parsování a zobrazování dat, programování mikročipů atd.

Hlavním přínosem tohoto softwaru je, že výrazně urychluje, zpřehledňuje a zjednodušuje vývoj a testování aplikací pro mikročipy, typicky programování a řízení různých druhů robotů.

**Klíčová slova:** analýza dat, počítačový program, robot, grafické zobrazení, vývoj a testování aplikací

## 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. Each module is designed for one particular function – parsing and displaying data, programming microcontrollers etc.

Main asset of this software is quick, well arranged and simple development and testing of applications for microcontrollers – typically programming and controlling various kinds of robots.

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

# Obsah

<b>Introduction</b>	<b>4</b>
Requirements for application	4
Present applications	4
Comparison of applications	5
<b>1 Lorris</b>	<b>6</b>
1.1 Website and repository	6
1.2 Application's structure	7
1.3 Session	9
1.4 Automatic updates	9
<b>2 Module: Analyzer</b>	<b>11</b>
2.1 Filters	14
2.2 Widget: number	15
2.3 Widget: bar	16
2.4 Widget: color	17
2.5 Widget: graph	18
2.6 Widget: script	19
2.7 Widget: circle	21
2.8 Widget: plátno	21
2.9 Widgety tlačítka a slider	22
2.10 Widget: vstup	23
2.11 Widget: status	24
2.12 Widget: terminál	25
<b>3 Modul: Proxy mezi sériovým portem a TCP socketem</b>	<b>27</b>
3.1 Proxy tunel	27
<b>4 Modul: Programátor</b>	<b>28</b>
4.1 Programátor Shupito	29
4.1.1 RS232 tunel	30
4.2 Bootloader avr232boot	30
4.3 Bootloader AVROSP	30
<b>5 Modul: Terminál</b>	<b>31</b>

<b>6</b>	<b>Příklady použití . . . . .</b>	<b>32</b>
6.1	Testování barevného senzoru . . . . .	32
6.2	Testování enkodérů . . . . .	33
6.3	Ladění PID regulátoru . . . . .	34
6.4	Vývoj robota pro soutěž Eurobot 2011 . . . . .	35
<b>7</b>	<b>Podpora joysticku . . . . .</b>	<b>38</b>
<b>8</b>	<b>Aplikace pro Android . . . . .</b>	<b>39</b>
8.1	Programátor . . . . .	41
8.2	Terminál . . . . .	42
	<b>Závěr . . . . .</b>	<b>43</b>
	<b>PŘÍLOHA A: Reference k widgetu <i>script</i> . . . . .</b>	<b>45</b>
	Základní script . . . . .	46
	Základní funkce . . . . .	47
	Vytvoření widgetu . . . . .	49
	Dostupné funkce widgetů . . . . .	50
	Widget číslo . . . . .	51
	Widget sloupcový bar . . . . .	51
	Widget barva . . . . .	52
	Widget graf . . . . .	52
	Widget vstup . . . . .	54
	Widget kolo . . . . .	57
	Widget plátno . . . . .	57
	Widget status . . . . .	58
	Widget tlačítko . . . . .	59
	Widget slider . . . . .	61
	Ukládání dat scriptu . . . . .	63
	Přístup k joysticku . . . . .	65
	Pár věcí na které je třeba myslet . . . . .	67
	<b>PŘÍLOHA B: Knihovny třetích stran . . . . .</b>	<b>68</b>
	<b>PŘÍLOHA C: Licence . . . . .</b>	<b>68</b>
	<b>PŘÍLOHA D: Reference . . . . .</b>	<b>69</b>

PŘÍLOHA E: Seznam obrázků . . . . .	73
-------------------------------------	----



# Úvod

Při stavbě robotů na robotické soutěže jsem se setkal s problémem zpracovávání dat z poměrně velkého množství senzorů (několik ultrazvukových měřáků vzdálenosti, enkodéry, které měří ujetou vzdálenost, tlačítka hlídající náraz do mantinelu, ...), které robot obsahuje, a jejich přehledného zobrazování.

## Požadavky na aplikaci

Od programu vyžaduji tyto vlastnosti:

1. Možnost zpracovávat data přicházející ze zařízení a přehledně je zobrazovat
2. Podpora co nejvíce formátů příchozích dat
3. Snadné a rychlé používání
4. Možnost běhu i na jiných systémech než je MS Windows
5. Co možná nejnížší cena
6. Snadná rozšiřitelnost (ideálně otevřený zdrojový kód)
7. Nezávislost na další aplikaci (např. MS Office Excel)

## Existující programy

Aplikací, které mají podobné určení (tj. vyčítání dat ze sériového portu a jejich zobrazování), jsem našel pouze několik. K dispozici jsou buď komerční aplikace, které stojí poměrně velké množství peněz (a přesto nesplňují všechny požadavky), anebo aplikace, které dokáží zobrazovat data pouze v jednom formátu – typicky graf.

- **SerialChart**[1] je open-source program<sup>1</sup> pro parsování a zobrazování dat přicházející ze sériového portu. Je jednoduchý a přehledný, dokáže však zobrazovat pouze graf a nastavení je třeba ručně napsat.
- **WinWedge**[2] je komerční program který dokáže zpracovávat data přicházející sériovým portem a zobrazovat je jako graf v MS Excel nebo ve webové stránce. Dokáže také posílat příkazy zpět do zařízení, má však horší ovládání a užší možnosti použití (hlavně kvůli nutnosti použít další program pro zobrazování). Je dostupný pouze pro MS Windows a základní verze stojí \$ 259.
- **Advanced Serial Data Logger**[3] je zaměřený primárně na export dat ze sériové linky do souboru, data dokáže zobrazovat pouze přeposláním do jiné aplikace (např. MS Office Excel), podobně jako WinWedge.
- **StampPlot Pro**[4] dokáže zobrazovat příchozí data ve widgetech zvolených uživatelem, má však komplikované ovládání, nemá otevřený zdrojový kód, je dostupný pouze pro MS Windows a pod verzí 7 nefunguje.

## Porovnání aplikací

Následující tabulka shrnuje funkce a vlastnosti jednotlivých programů. Číslování požadavků odpovídá seznamu v kapitole „Požadavky na aplikaci“.

Požadavky:	1	2	3	4	5	6	7
SerialChart	✓	✗	✓	✗	✓	✓	✓
WinWedge	✗	✓	✓	✗	✗	✗	✗
Advanced Serial Data Logger	✗	✓	✓	✗	✗	✗	✗
StampPlot Pro	✓	✓	✗	✗	✓	✗	✓

<sup>1</sup>Program s otevřeným zdrojovým kódem

Z těchto důvodů jsem se rozhodl napsat vlastní aplikaci, která bude všechny výše uvedené požadavky splňovat.

## 1 Popis rozhraní

Svůj program jsem pojmenoval „Lorris“, je vytvořený v C++ a využívá Qt Framework[5], což je multiplatformní framework, který mimo jiné umožňuje spustit aplikaci na více operačních systémech – testoval jsem na Debian Linux[6] (Wheezy, 64bit) a Windows 7.

### 1.1 Web a repozitář programu

GIT<sup>2</sup> repozitář programu jsem vytvořil na serveru GitHub[7], který kromě hostingu repozitáře poskytuje i několik dalších služeb, mezi nimi i hosting webu projektu. Na webu, který jsem vytvořil, jsou odkazy ke stažení spustitelných souborů pro Windows, popis programu, video s představením programu (6 min.), ukázky z programu (screenshoty) a návod ke zkompilování pro MS Windows a Linux.

- Repozitář: <https://github.com/Tassadar/Lorris>
- Web (česká verze):  
<http://tassadar.github.com/Lorris/cz/index.html>
- Web (anglická verze):  
<http://tassadar.github.com/Lorris/index.html>
- Prezentace práce:  
<http://www.sokolska.cz/soc-2012/bocek-vojtech-lorris-sada-nastroju-pro-robotiku/>

V repozitáři nadále probíhá aktivní vývoj.

---

<sup>2</sup> *GIT* – distribuovaný systém správy verzí

## 1.2 Struktura aplikace

Program je navrhnutý jako modulární aplikace, aby mohl zastřešit několik samostatných částí, které však mají podobnou oblast použití. Základní část programu poskytuje připojení k zařízení (např. robot, deska s čipem) a ukládání nastavení aplikace, samotné zpracování dat probíhá v modulech, které jsou otevírány v panelech – podobně jako stránky ve webovém prohlížeči. Lorris umí otevřít několik oken zároveň a dokáže také rozdělit okno na několik částí, na obrázku 2 je nalevo modul analyzátor a napravo terminál.

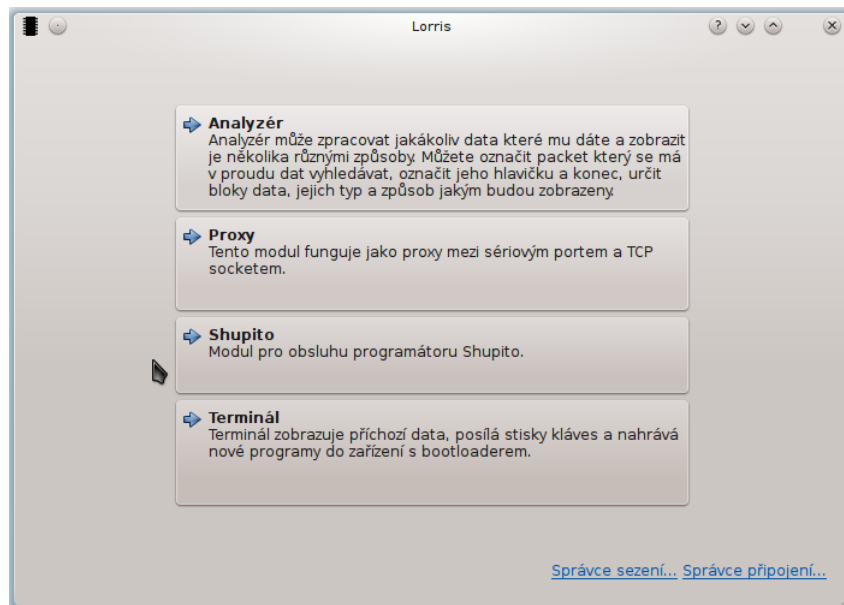
Možnosti připojení k zařízení:

- Sériový port
- Shupito Tunel (virtuální sériový port, viz kapitola 4.1.1)
- TCP socket<sup>3</sup>
- Načtení dat ze souboru

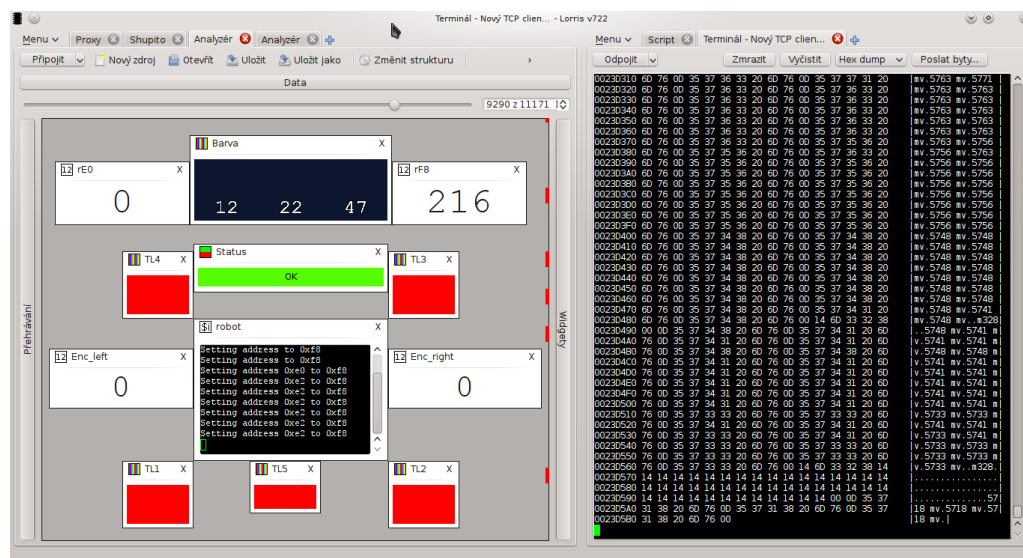
Je možné mít připojeno více různých modulů na jedno zařízení.

---

<sup>3</sup>*Transmission Control Protocol* – připojení přes internet.



Obrázek 1: Dialog vytvoření panelu



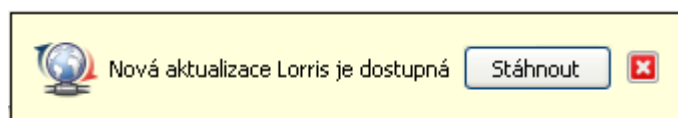
Obrázek 2: Ukázka rozdělení okna na více částí

### 1.3 Sezení

Lorris dokáže uložit vše, co má uživatel aktuálně otevřené (záložky, jejich uspořádání, informace o připojení, data jednotlivých záložek atd.), jako tzv. sezení (anglicky *session*). Sezení je možné později načíst a tímto se vrátit k předchozí práci. Lorris automaticky ukládá sezení před svým ukončením, takže když uživatel program znovu otevře, vše je ve stejném stavu jako když aplikaci opouštěl.

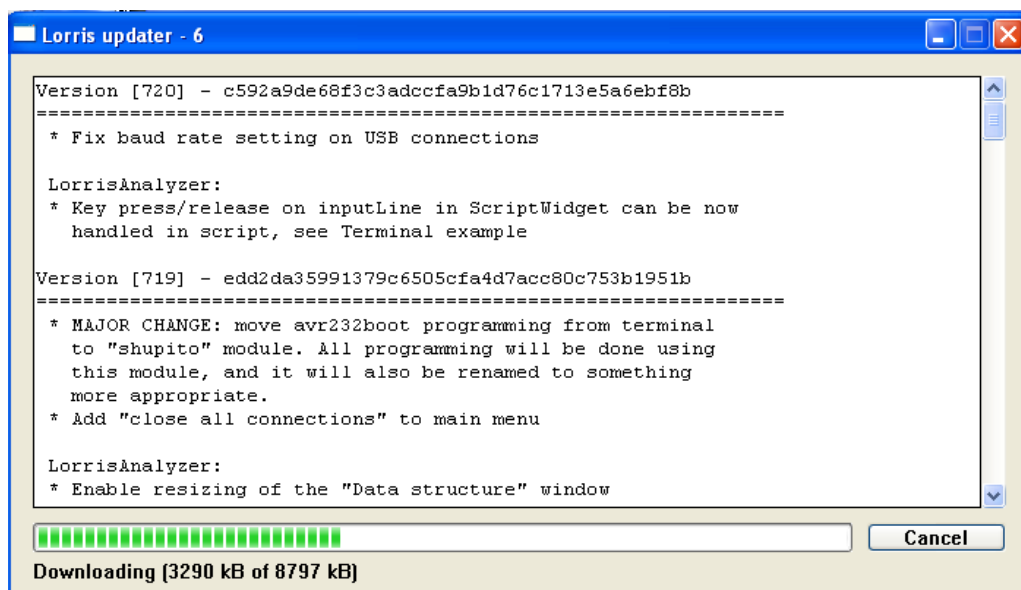
### 1.4 Automatická aktualizace

Lorris se pod Windows dokáže sama aktualizovat. Při spuštění kontroluje zda je dostupná nová verze a pokud ano, zobrazí uživateli malé upozornění:



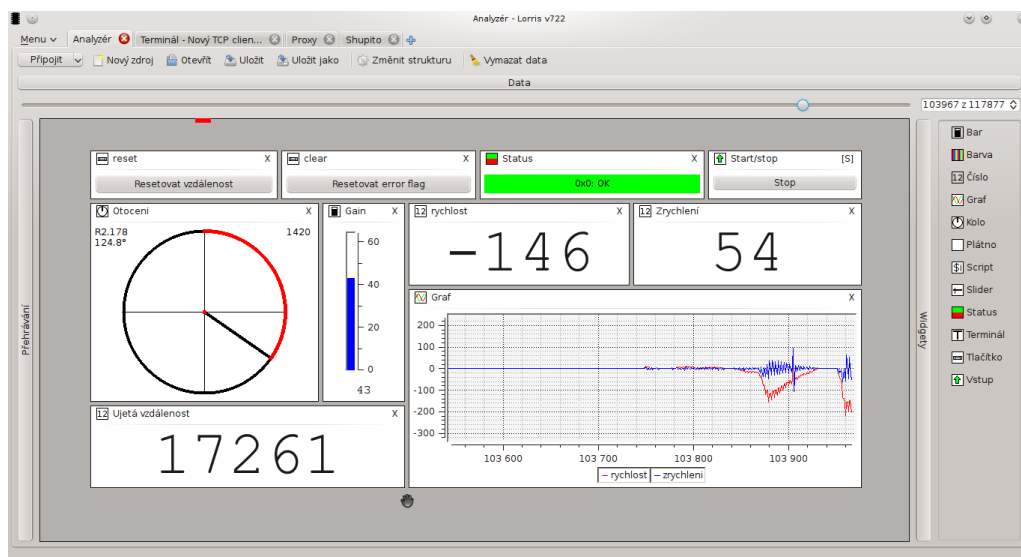
Obrázek 3: Upozornění o dostupné aktualizaci

V případě, že uživatel aktualizaci potvrdí, se Lorris ukončí a spustí se malý pomocný program, který stáhne novou verzi a nainstaluje ji. Zobrazuje při tom seznam změn oproti staré verzi.



Obrázek 4: Probíhající aktualizace

## 2 Modul: Analyzér



Obrázek 5: Modul analyzér

Tento modul parsuje data (strukturované do paketů) přicházející ze zařízení a zobrazuje je v grafických „widgetech“. Zpracovaná data si aplikace ukládá do paměti – listování pakety je možné pomocí posuvníku a boxu v horní části okna. Data (přijatá data, struktura paketů a rozestavení a nastavení widgetů) je také možné uložit do souboru a později zase v programu otevřít.

Struktura dat se nastavuje v samostatném dialogu (viz obrázek 7), kde je možno nastavit délku packetu, jeho endianness<sup>4</sup>, přítomnost hlavičky a její obsah – statická data („start bajt“), délka packetu (pokud je proměnná), příkaz a ID zařízení. Podle příkazu a ID zařízení je možno později data filtrovat.

<sup>4</sup>Endianness – pořadí uložení bajtů v paměti počítače



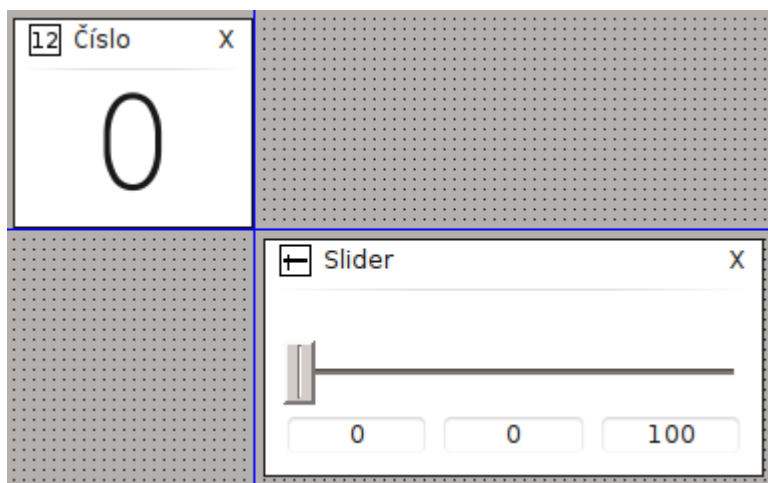
Po nastavení struktury se přijatá data začnou po packetech zobrazovat v horní části okna, a v pravé části se zobrazí sloupeček s dostupnými zobrazovacími widgety. Widgety se dají pomocí drag&drop principu „vytáhat“ na plochu v prostřední části okna. Data se k widgetu přiřadí taktéž pomocí drag&drop, tentokrát přetažení prvního bajtu dat na widget.

Poté widget zobrazuje data tohoto bajtu, nebo tento bajt bere jako první, pokud jsou data delší. Aby bylo možné zpětně poznat, který bajt je k widgetu přiřazen, je po najetí myši na widget červeně zvýrazněn.

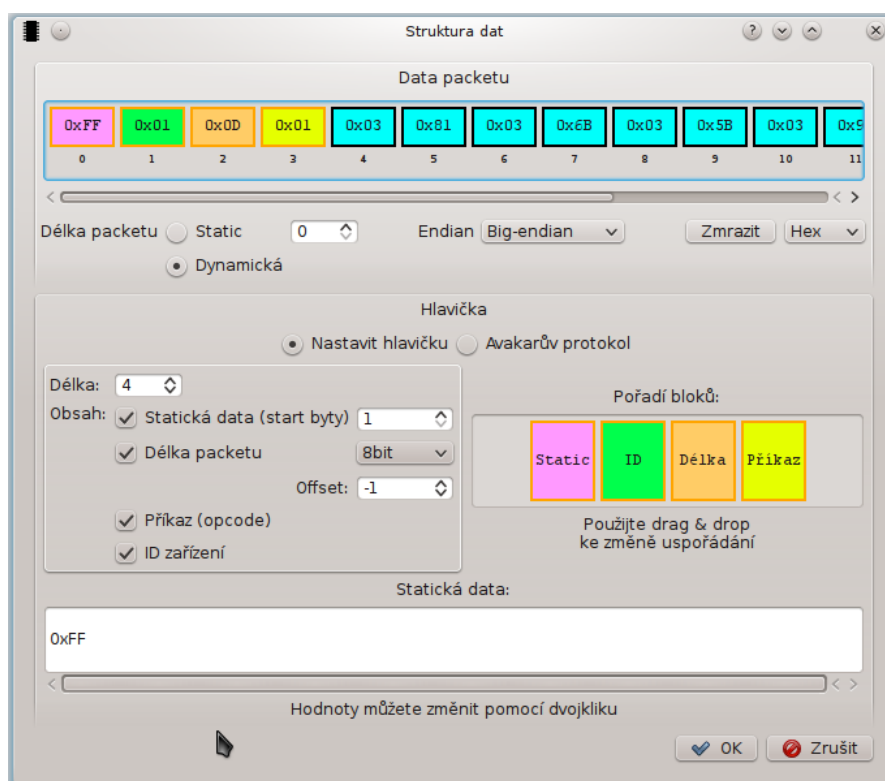
Nastavení widgetu jsou přístupná v kontextovém menu po pravém kliknutí myši na widget. Nastavit lze jméno a další parametry podle typu widgetu – podrobněji jsou možnosti nastavení popsány u jednotlivých widgetů. Widgety je taktéž možné „uzamknout“, aby nebylo možné je zavřít, měnit jejich pozici a velikost.

Widgety je možné přesně rozmisťovat pomocí „přichytávání“ k síti anebo k ostatním widgetům pomocí zarovnávacích čar (viz obrázek 6). Lze je také jednoduše a rychle duplikovat – stačí přemístit widget se stisknutou klávesou control.

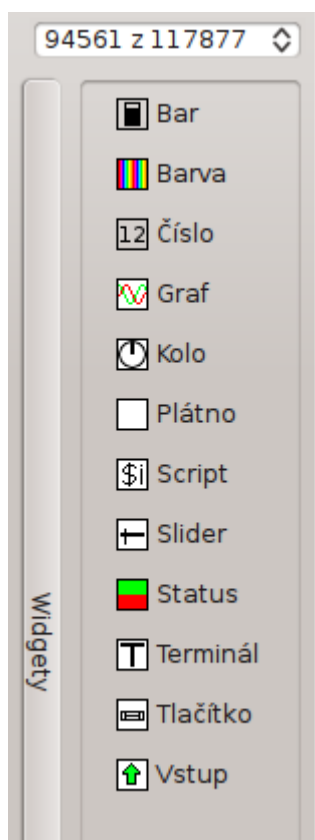
U některých widgetů se může hodit následující funkce: widgety je možné rychle zvětšit tak, aby zabraly celou viditelnou plochu pomocí gesta myši – stačí widget chytit jako při přesouvání a „zatřepat“ s ním zleva doprava. Při přesunutí se pak widget zmenší na svoji původní velikost.



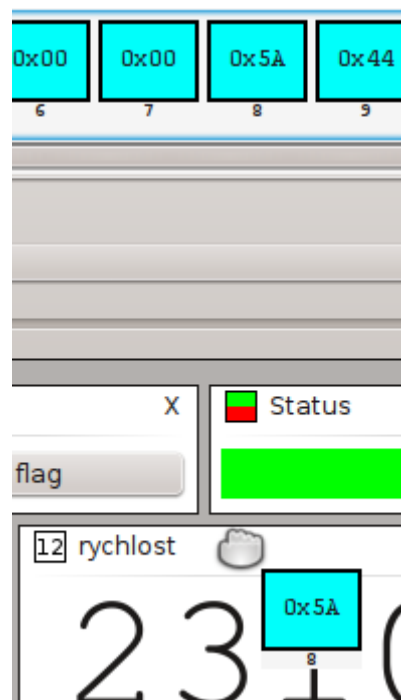
Obrázek 6: Zarovnávání widgetů pomocí sítě a čar



Obrázek 7: Dialog nastavení struktury dat



(a) Seznam widgetů

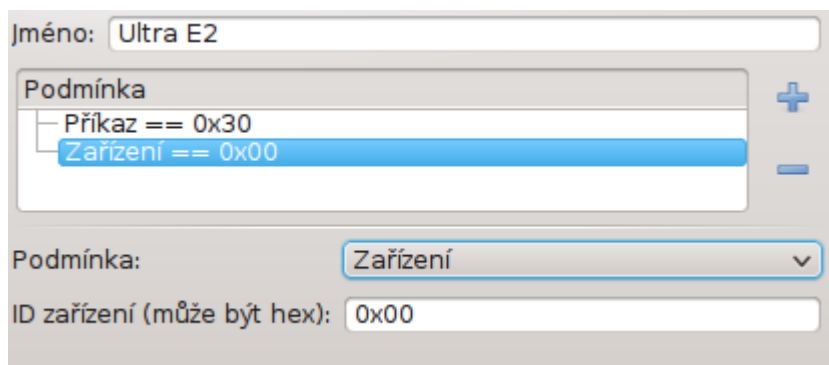


(b) Přiřazení dat pomocí drag&drop

Obrázek 8: Widgety

## 2.1 Filtrování dat

Analýzátor umí příchozí data filtrovat pomocí filtrů. Tyto filtry obsahují podmínky, podle kterých se určí, zda příchozí packet projde nebo ne.



Obrázek 9: Nastavení filtrů

Podmínka může kontrolovat buďto příkaz nebo zařízení z hlavičky packetu, hodnotu bajtu v packetu nebo může spustit jednoduchý uživatelský script. Díky scriptu je možné napsat takřka jakoukoliv podmínku pro filtrování.

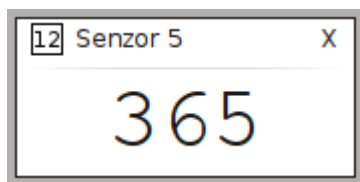
```

1 // Vrací true pokud má projít, false pokud ne
2 function dataPass(data, dev, cmd) {
3     return false;
4 }

```

Příklad 1: Script pro podmínku filtru

## 2.2 Widget: číslo



Obrázek 10: Widget: číslo

Tento widget dokáže zobrazovat celá čísla (se znaménkem i bez, 8 až 64

bitů dlouhé) a desetinná čísla (single-precision<sup>5</sup>, 32bit a 64bit).

Widget dále dokáže zarovnat číslo na maximální délku jeho datového typu a formátovat ho těmito způsoby:

- Desítkový – číslo v desítkové soustavě
- Desítkový s exponentem – použije exponent pro zapsání velkých čísel. Dostupné pouze pro desetinná čísla.
- Hexadecimální – výpis v šestnáctkové soustavě. Dostupné pouze pro přirozená čísla.
- Binární – zobrazí číslo ve dvojkové soustavě. Dostupné pouze pro přirozená čísla.

Další funkcí je přepočítávání hodnoty pomocí výrazu. Toto se hodí například u infračervených senzorů vzdálenosti, kdy se hodnota, kterou na senzoru naměří AD převodník, musí přepočítat pomocí určité rovnice abychom dostali hodnotu v centimetrech. Výraz může vypadat například takto:

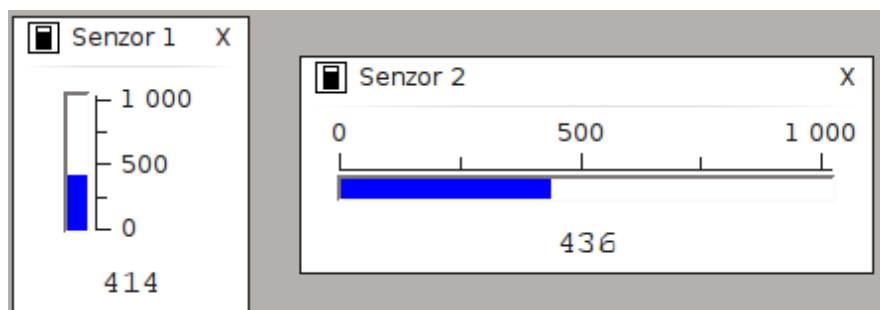
$$2914/(\%n+5)-1$$

kde `%n` je zástupná sekvence pro číslo, které by se jinak ve widgetu zobrazilo. Tento výraz je pro přepočítání vzdálenosti na centimetry podle hodnoty přečtené z infračerveného senzoru vzdálenosti Sharp GP2Y0A41.

---

<sup>5</sup>Standardní formát uložení desetinných čísel v jazyku C a dalších (standard IEEE 754-2008).

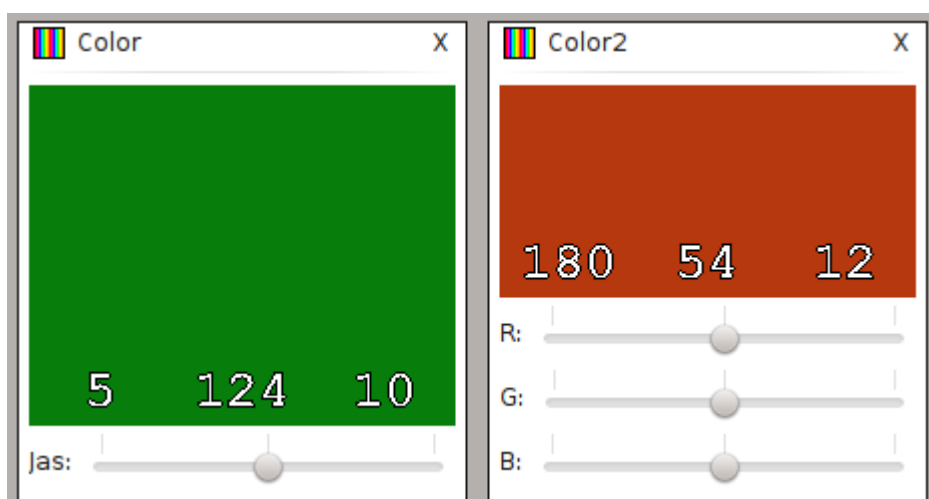
## 2.3 Widget: sloupcový bar



Obrázek 11: Widget: sloupcový bar

Widget zobrazuje hodnotu ve sloupcovém baru. Lze nastavit datový typ vstupních dat (stejně jako u čísla), orientaci (vertikální nebo horizontální) a rozmezí zobrazovaných hodnot. Stejně jako widget číslo také dokáže přepočítat hodnotu podle výrazu.

## 2.4 Widget: barva



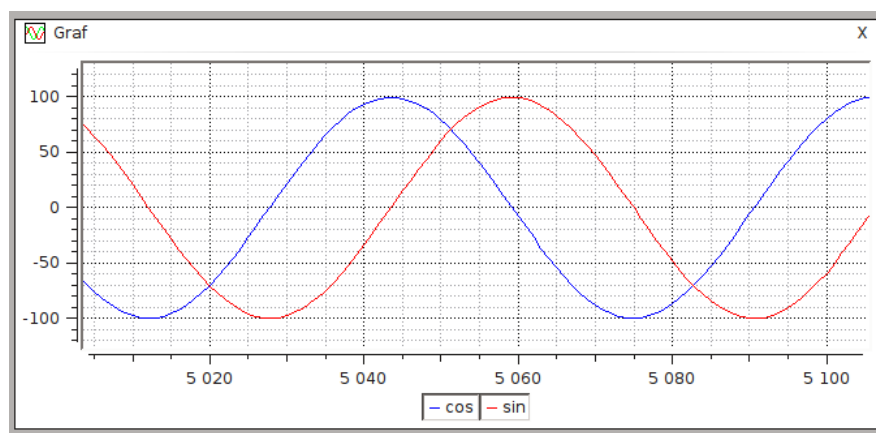
Obrázek 12: Widget: barva

Tento widget dokáže zobrazit příchozí hodnoty jako barevný obdélník. Podporované formáty:

- **RGB** (8b/kanál, 3x uint8)
- **RGB** (10b/kanál, 3x uint16)
- **RGB** (10b/kanál, 1x uint32)
- **Odstíny šedé** (8b/kanál, 1x uint8)
- **Odstíny šedé** (10b/kanál, 1x uint16)

Widget také dokáže provést korekci jasu všech barev zároveň nebo každé z barev RGB zvlášť.

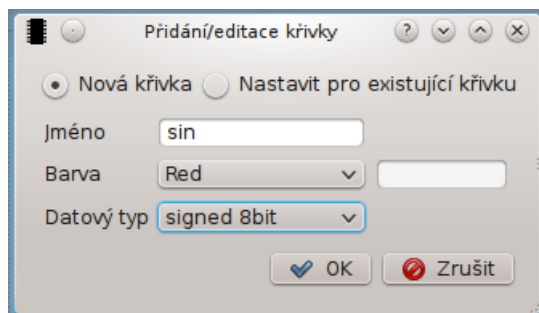
## 2.5 Widget: graf



Obrázek 13: Widget: graf

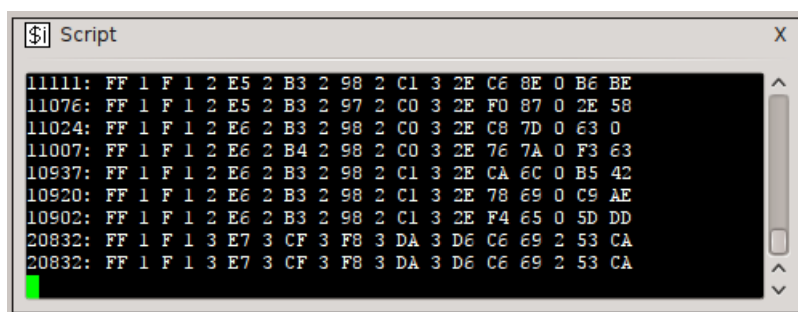
Widget graf zobrazuje hodnoty v grafu – na osu  $x$  se vynáší pořadí dat a na osu  $y$  hodnoty dat. Lze nastavovat jméno, barvu a datový typ křivky grafu, automatické posouvání grafu, velikost vzorku, měřítko os grafu a zobrazení legendy. Kliknutí na křivku grafu v legendě tuto křivku skryje.

Měřítka osy se ovládá otáčením kolečka myši po najetí kurzoru nad osu, po najetí do prostoru grafu se podobně ovládá měřítko celého grafu.



Obrázek 14: Dialog pro nastavení parametrů křivky grafu

## 2.6 Widget: script



Obrázek 15: Widget: script

Tento widget umožňuje zpracovávání dat pomocí scriptu, který si napíše sám uživatel. Může při tom použít buďto Python nebo QtScript[8] (jazyk založený na standardu ECMAScript<sup>6</sup>, stejně jako JavaScript<sup>7</sup>, díky tomu jsou tyto jazyky velmi podobné). Script může zpracovávat příchozí data, reagovat na stisky kláves a posílat data do zařízení. Základní výstup může

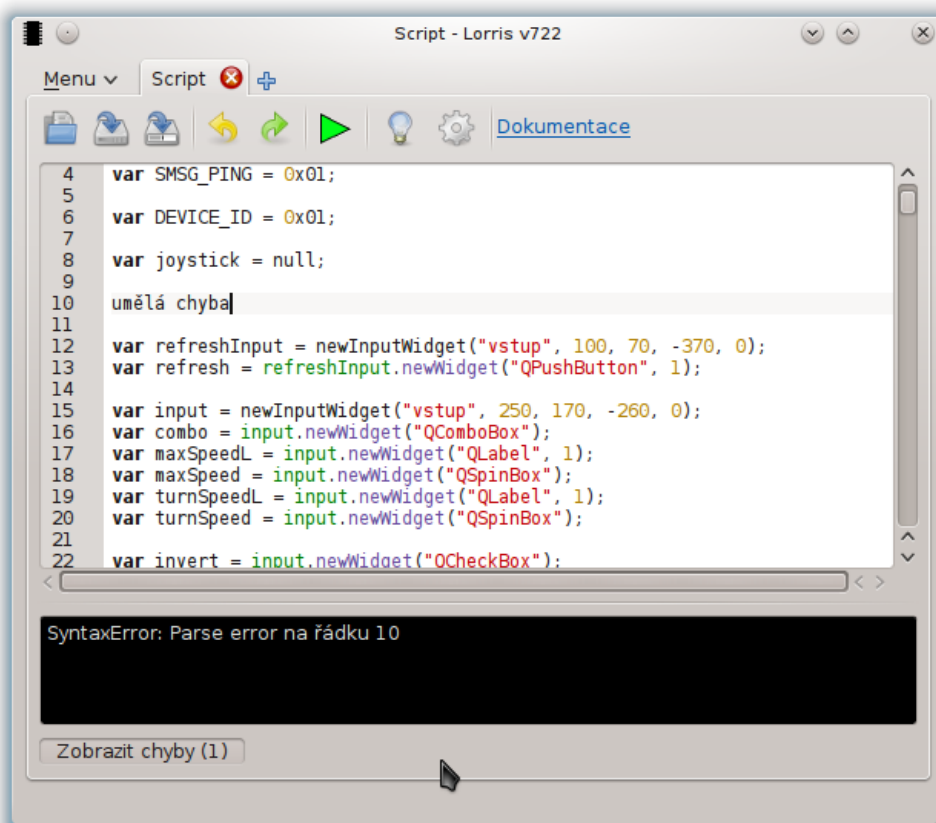
<sup>6</sup> *ECMAScript* – scriptovací jazyk standardu ECMA-262 a ISO/IEC 16262

<sup>7</sup> *JavaScript* – objektově orientovaný skriptovací jazyk, používaný hlavně na webu



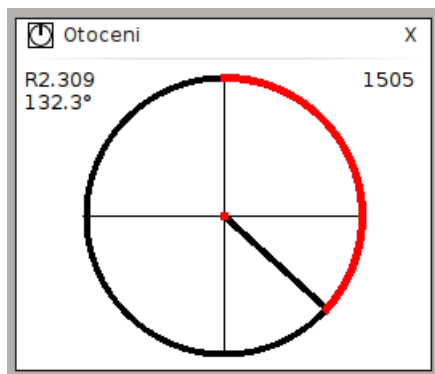
být zobrazen v terminálu (viz obrázek 15), je však možné využít ke zobrazování také ostatní widgety (číslo, bar, ...) – script si je vytvoří jako objekt a nastavuje do nich data. Reference k vestavěným funkcím, které lze použít ve scriptu, je v příloze A.

Editor scriptu má v sobě vestavěné ukázky kódu, například jak nastavit hodnotu existujícího widgetu *číslo*, jak odeslat data nebo jak reagovat na stisknutí klávesy (na obrázku 16 jsou skryté pod ikonkou žárovky). Je v něm také odkaz na automaticky generovanou dokumentaci, která je na adrese <http://technika.junior.cz/docs/Lorris/>.



Obrázek 16: Dialog pro nastavení zdrojového scriptu

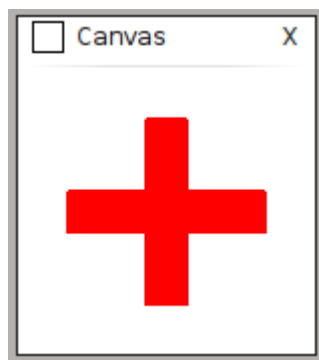
## 2.7 Widget: kolo



Obrázek 17: Widget: kolo

Widget kolo zobrazuje příchozí hodnotu jako úhel v kruhu, což se hodí například při zobrazování natočení kola robota. Dokáže zobrazit data přicházející jako úhel ve stupních, radiánech nebo jako číslo v určitém rozmezí (například enkodér s rozlišením 12 bitů vrací číslo od 0 do 4096).

## 2.8 Widget: plátno



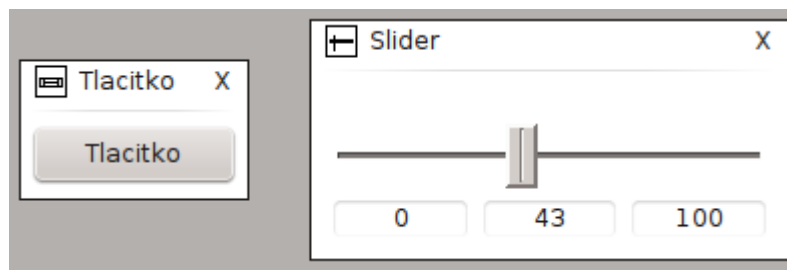
Obrázek 18: Widget: plátno

Plátno je widget který lze ovládat pouze ze scriptu a je určen ke kreslení 2D grafiky. Dokáže zobrazit čáry, obdélníky, kruhy a elipsy. V následujícím příkladu je kód pro nakreslení červeného kříže uprosřed widgetu.

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

Příklad 2: Ovládání widgetu plátno

## 2.9 Widgety tlačítko a slider



Obrázek 19: Widgety tlačítko a slider

Tyto dva widgety pouze umožňují interakci se scriptem – po stisknutí tlačítka se zavolá metoda ve scriptu, ve které může uživatel například poslat příkaz do robota, při posunutí slideru se ve scriptu zavolá metoda, ve které může uživatel například změnit rychlost robota a podobně. Tlačítku lze nastavit klávesovou zkratku a slideru zkratku pro „zaostření“, aby ho uživatel poté mohl posouvat pomocí šipek na klávesnici.

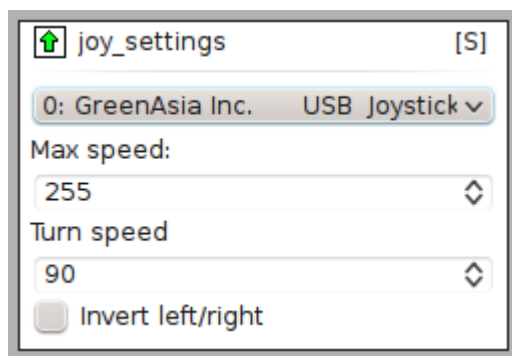
```

1  function Slider_valueChanged() {
2      appendTerm("Hodnota slideru " + Slider.getValue() + "\n");
3  }
4
5  function Tlacitko_clicked() {
6      appendTerm("Tlacitko stisknuto\n");
7  }

```

Příklad 3: Metody volané widgety *slider* a *tlačítko*

## 2.10 Widget: vstup



Obrázek 20: Widget *vstup* s nastavením joysticku

Tento widget je také určený k interakci se skriptem (tj. vstupu uživatele), přičemž skript také určuje prvky rozhraní – widget je v základu prázdný a až skript do něj přidá například tlačítko nebo textové pole. Tento widget je mírně složitější na obsluhu, ale lze díky němu použít všechny prvky UI, které Qt Framework obsahuje – tlačítka, posuvníky, textová pole, vysouvací seznamy a podobně. V příkladu 5 je skript pro vytvoření prvků jako v obrázku 20.

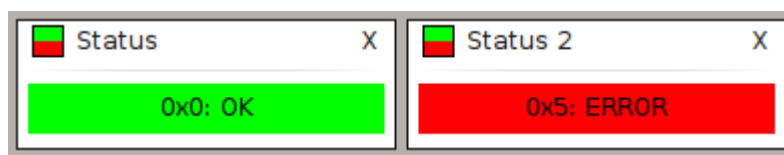
```

1 // parametry: jmeno Qt widgetu, "stretch" hodnota
2 var joyList = joy_settings.newWidget("QComboBox");
3 var maxSpdLabel = joy_settings.newWidget("QLabel", 1);
4 var maxSpd = joy_settings.newWidget("QSpinBox");
5 var turnSpdLabel = joy_settings.newWidget("QLabel", 1);
6 var turnSpd = joy_settings.newWidget("QSpinBox");
7 var invert = input.newWidget("QCheckBox");
8
9 // Nastaveni textu do QLabel
10 maxSpdLabel.text = "Max speed:";

```

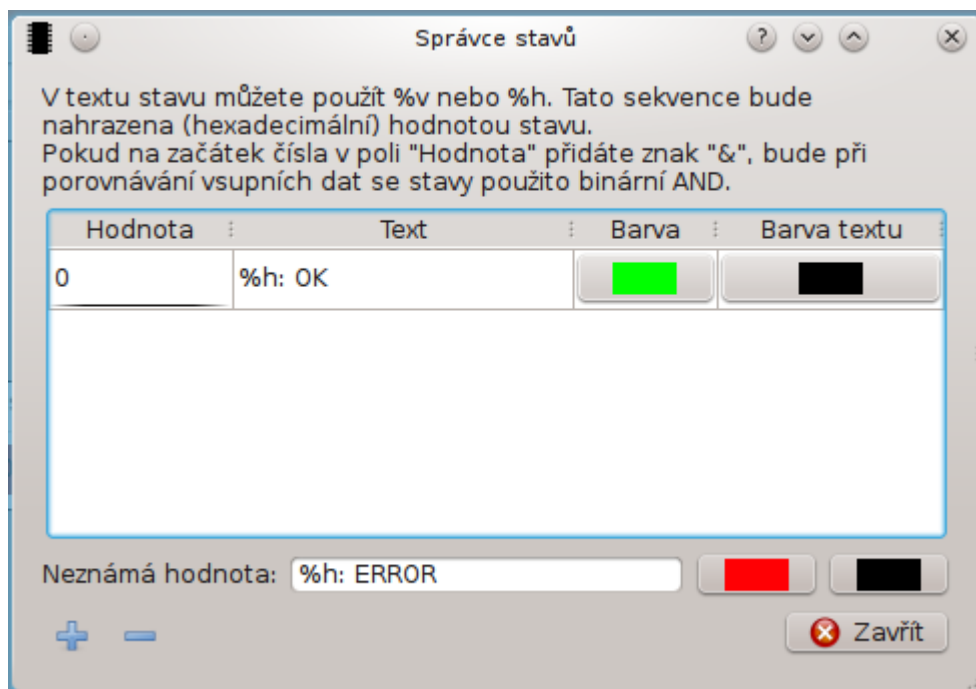
Příklad 4: Přidání prvků do widgetu *vstup*

## 2.11 Widget: status



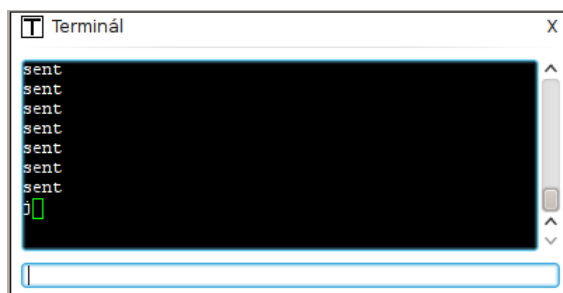
Obrázek 21: Widget status

Widget status je určený k zobrazování stavu například tlačítka (stisknuté/nestisknuté) nebo chybového stavu z enkodéru (0 = vše je v pořádku, ostatní čísla je možné dohledat v datasheetu enkodéru). Uživatel k příchozím hodnotám přiřadí jednotlivé stavy (text a barvu, viz obrázek 22) a widget je poté zobrazuje. Lze nastavit i „neznámou hodnotu“, která se zobrazí pokud žádný z nadefinovaných stavů neodpovídá příchozí hodnotě.



Obrázek 22: Nastavení stavů

## 2.12 Widget: terminál

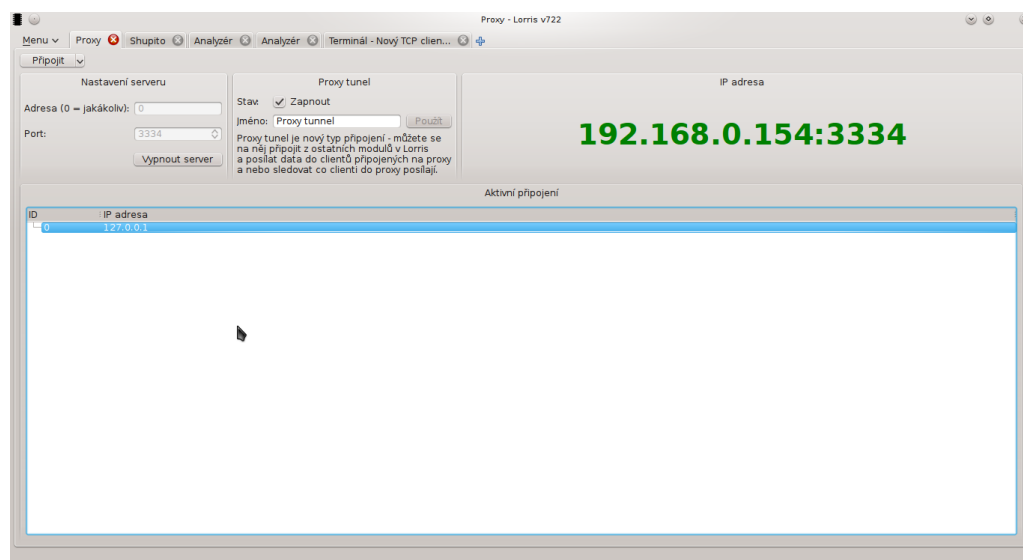


Obrázek 23: Widget terminál

Tento widget je zde pouze pro pohodlí uživatele, jedná se totiž o widget *script* ve kterém je přednastavený kód díky kterému se tento widget chová stejně jako terminál. Uživatel může script tohoto widgetu libovolně upravo-

vat.

### 3 Modul: Proxy mezi sériovým portem a TCP socketem



Obrázek 24: Proxy mezi sériovým portem a TCP socketem

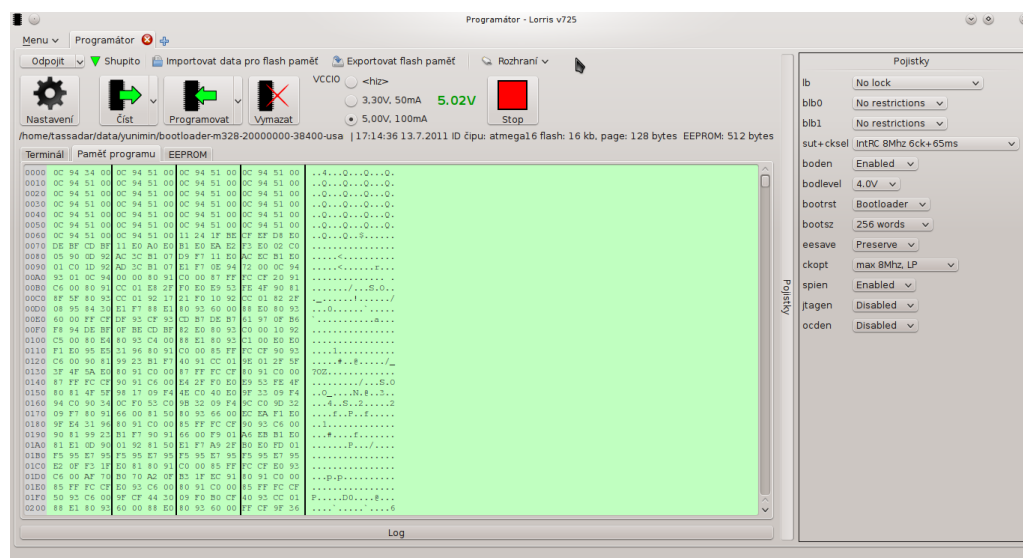
Jednoduchá proxy mezi sériovým portem a TCP socketem. Vytvoří server, na který je možné se připojit z Loris nebo jiného programu na jiném počítači. Po připojení se přeposílají data ze sériového portu připojeným klientům a naopak.

#### 3.1 Proxy tunel

Tento modul také přidává nové umělé připojení - „proxy tunel“. Pokud toto připojení použije nějaký z dalších modulů v Loris, tak může posílat a přijímat data od všech TCP klientů připojených na proxy. Toto je možné využít například tak, že v modulu analyzér je script, který generuje data a přes proxy tunel je pak odesílá všem klientům připojeným na proxy.

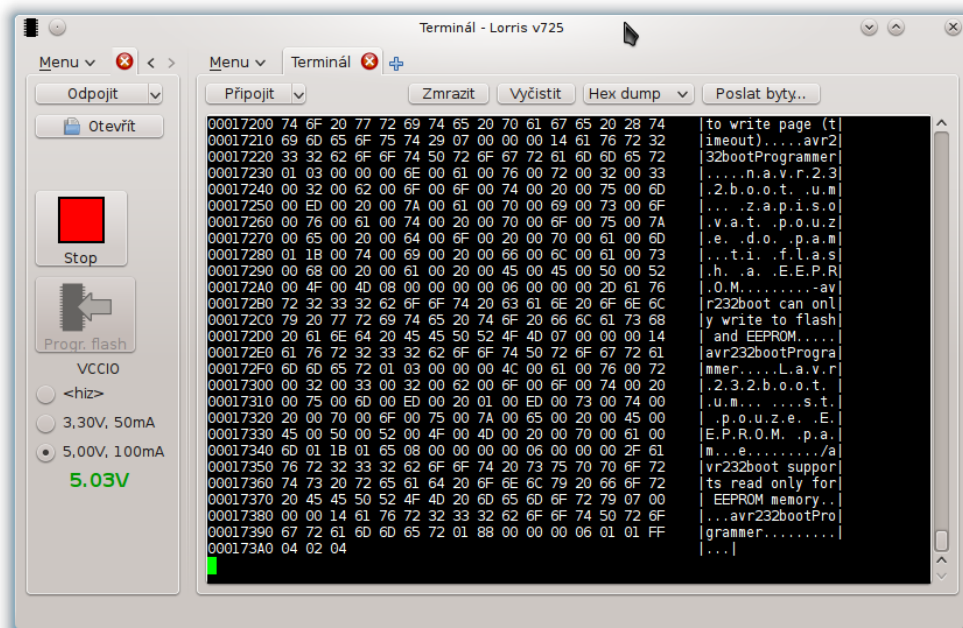


## 4 Modul: Programátor



Obrázek 25: Modul Programátor

Tento modul funguje jako grafické rozhraní pro několik typů programátorů a bootloaderů. Podporuje dva typy rozhraní – plné (obrázek 25) a zmenšené (obrázek 26). Plné rozhraní obsahuje tlačítka a nastavení pro programování všech pamětí čipu, zmenšené rozhraní pak obsahuje pouze tlačítka na programování hlavní paměti a zastavení čipu. Zmenšený mód je vhodný při rozdělení okna na více částí protože obsahuje pouze nejpoužívanější prvky a nezabírá zbytečně místo.



Obrázek 26: Zmenšené UI modulu *programátor* (nalevo) s otevřeným *terminálem*

## 4.1 Programátor Shupito

Shupito je programátor mikročipů vytvořený Martinem Vejnářem, který dokáže programovat mikrokontroléry pomocí ISP<sup>8</sup>, PDI<sup>9</sup> a JTAG<sup>10</sup> rozhraní.

Modul programátor v mojí práci slouží jako oficiální rozhraní pro Shupito. Převážná část komunikace s programátor je napsána samotným Martinem Vejnářem.

<sup>8</sup>*In-system programming* – rozhraní, které umožňuje programovat čipy bez dalšího zařízení přímo v desce plošného spoje.

<sup>9</sup>*Program and Debug Interface* – rozhraní firmy Atmel umožňující programování čipů přímo na desce, podobně jako ISP

<sup>10</sup>*Joint Test Action Group* – rozhraní podle standardu IEEE 1149.1 umožňující mimo jiné programování a debugování čipů

### 4.1.1 RS232 tunel

Shupito dokáže vytvořit tunel<sup>11</sup> pro RS232 linku z programovaného čipu do počítače. Lorris umí této funkci využít – aktivní tunel se zobrazí jako další typ připojení a je možné se na něj připojit v ostatních modulech.

## 4.2 Bootloader avr232boot

Autorem tohoto bootloderu je také Martin Vejnár. Avr232boot je pouze pro čipy Atmel ATmega a je inspirovaný referenčním bootloderem pro tyto mikrokontroléry, je však napsaný tak, aby zabíral v čipu co nejméně místa. Původně uměl pouze programovat flash paměť čipu (ta, ve které je uložen program), já jsem do něj přidal podporu čtení a zapisování paměti EEPROM<sup>12</sup>.

Lorris dokáže pomocí tohoto bootloderu programovat flash paměť a číst a programovat EEPROM.

## 4.3 Bootloader AVROSP

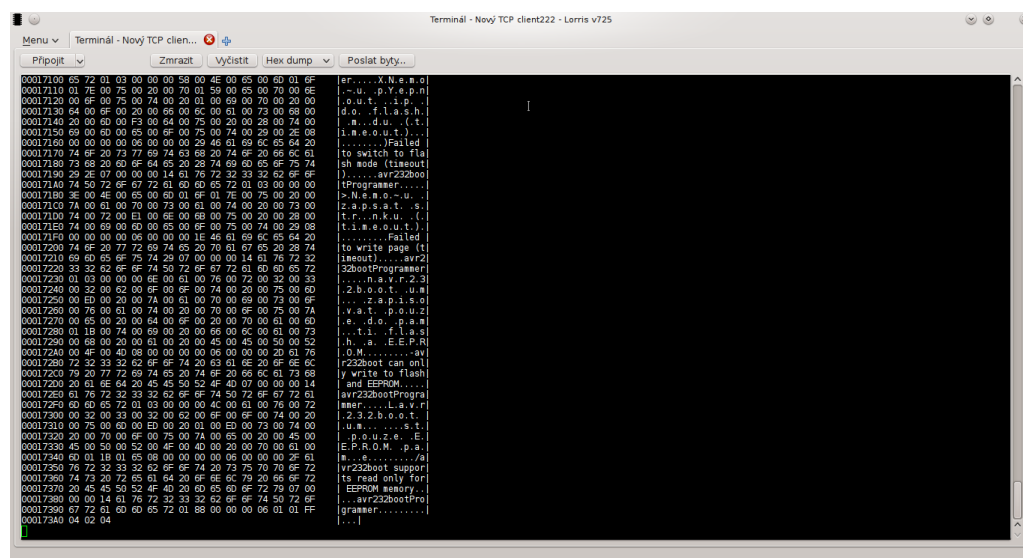
*AVR Open Source Programmer* je protokol používaný několika bootloderů firmy Atmel pro čipy ATmega a ATxmega. Lorris dokáže pomocí toho bootloderu programovat a číst flash i EEPROM paměť čipu.

---

<sup>11</sup>Přímé spojení programovaného čipu a počítače přes programátor.

<sup>12</sup>Typ flash paměti, která udrží data i bez proudu. V čípech se používá na uložení např. nastavení

## 5 Modul: Terminál



Obrázek 27: Modul terminál

Základní pomůcka při práci s mikrokontroléry, klasický terminál. Zobrazuje data přijatá přes sériový port a posílá stisky kláves a kromě klasického textového módu dokáže přichozí data zobrazovat jako hexadecimální hodnoty všech přichozích bajtů.

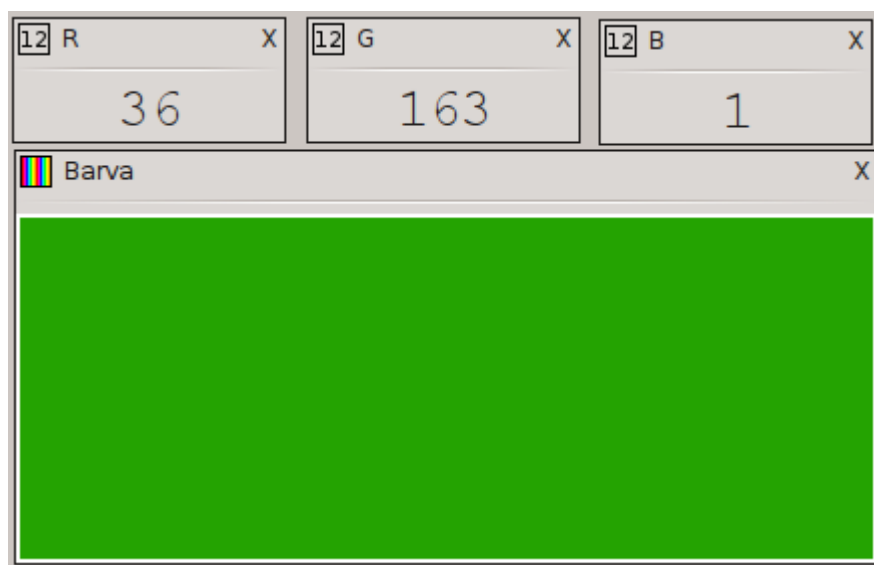
Terminálu je možné nastavit barevné schéma, velikost a font textu, jaká sekvence kontrolních znaků se má odeslat při stisknutí klávesy enter a chování některých kontrolních znaků (například jestli má znak `\n` vytvořit nový řádek nebo ne).

## 6 Příklady použití

### 6.1 Testování barevného senzoru

**Situace:** Stavím robota do soutěže (Eurobot, RobotChallenge, ...), ve které je možné se na herním poli orientovat podle barvy. Chci barevný senzor otestovat, proto jsem na nepájivém poli postavil jednoduchý obvod s čipem, na který je senzor připojený. Čip bude dávat senzoru pokyny k měření a vyčítat z něj RGB hodnoty, které následně pošle do RS232 linky.

**Řešení:** Program, který bude ze senzoru číst hodnoty naprogramuji do čipu pomocí programátoru Shupito, který také poskytne tunel pro RS232 linku. Na tento tunel se připojím modulem Analyzér, ve kterém díky widgetu „barva“ mohu vidět barvu, kterou senzor rozpoznal.

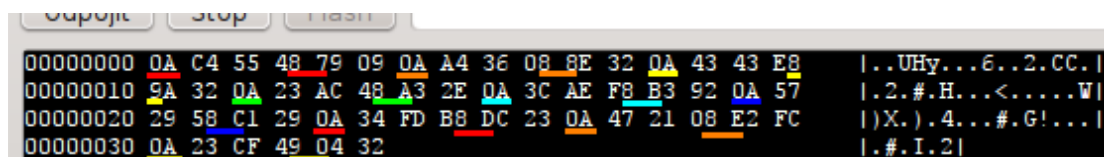


Obrázek 28: Barva v modulu Analyzér

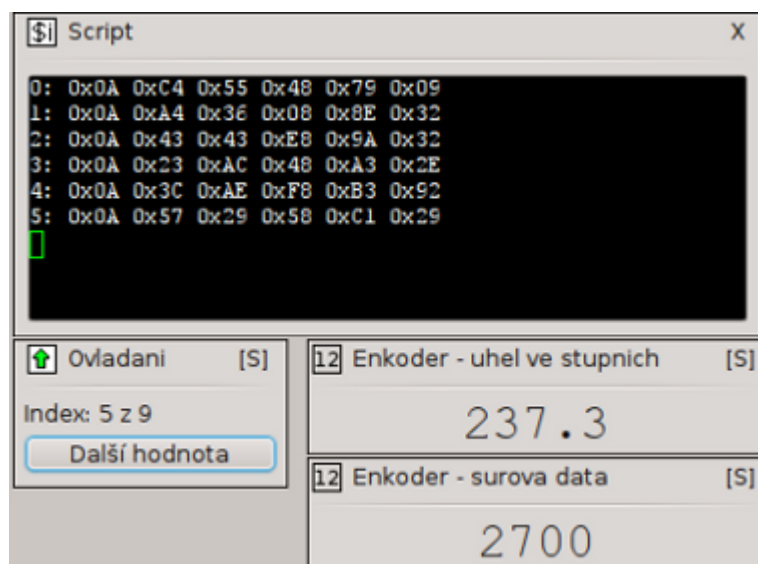
## 6.2 Testování enkodérů

**Situace:** Potřebuji otestovat přesnost magnetických enkodérů, které však odesílají data o úhlu natočení rozdělené v několika bajtech v takovém formátu, který znemožňuje použití např. terminálu.

**Řešení:** Nechci za tímto účelem stavět a programovat novou desku s dalším mikročipem, připojím tedy enkodér k počítači. V Lorrise otevřu modul analyzátor a ve widgetu „script“ napíši jednoduchý script, který složí úhel do jednoho čísla a zobrazí ho ve widgetu „číslo“.



Obrázek 29: Surová data z enkodérů s vyznačenými bajty, které obsahují úhel natočení



Obrázek 30: Data z enkodérů zpracovaná analyzérem

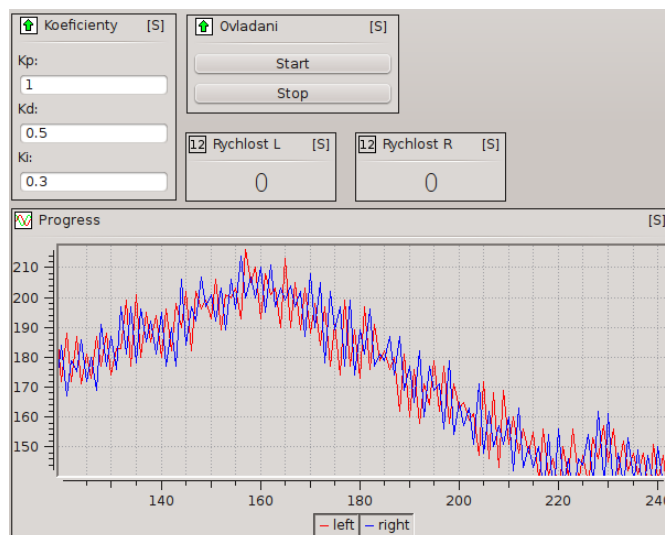
### 6.3 Ladění PID regulátoru

**Situace:** Robot kvůli rozdílnému výkonu motorů nejede rovně. Tento problém jsem se rozhodl řešit pomocí PID regulátoru, pro jehož správnou funkci je potřeba nastavit několik konstant.

**Řešení:** Program v robotovi mi posílá aktuální výkon motorů a nastavení konstant PID regulátoru a umožňuje přenastavení těchto konstant a ovládání robota. Tento program do robota nahrávám přes bluetooth pomocí modulu Terminál, protože čip má v sobě bootloader – díky tomu nemusím mít připojený programátor.

V modulu analyzátor si zobrazím aktuální hodnoty PID regulátoru (jako číslo) a výkon motorů (jako graf či číslo). Do widgetu script napíši jednoduchý script, který po stisku kláves změní nastavení konstant regulátoru nebo rozjede/zastaví robota.

Tento postup jsem použil při ladění PID regulátoru robota 3pi[10] během přípravy na soutěž *Line Follower Standard*, která je součástí Robotického dne 2012[11].

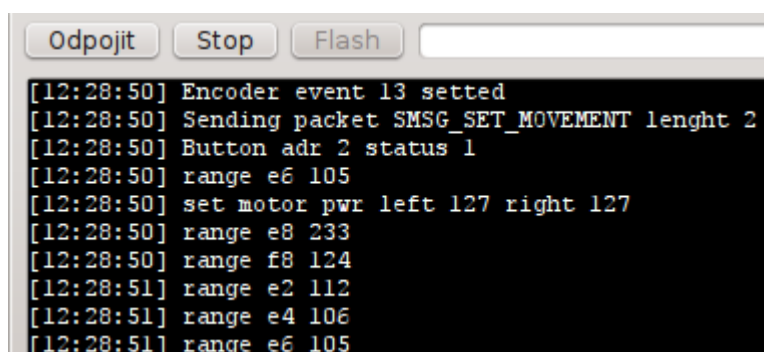


Obrázek 31: Ladění PID regulátoru

## 6.4 Vývoj robota pro soutěž Eurobot 2011

V minulém roce jsem se zúčastnil soutěže Eurobot[12]. Cíl soutěže je každý rok jiný, v minulém ročníku bylo cílem hrát něco jako zjednodušené šachy. Herní hřiště bylo rozděleno na barevnou šachovnici a leželi na něm „pěšci“ (žluté disky), které měli roboti posouvat na políčka svojí barvy, případně z nich stavět věže. Vyhrával robot s největším počtem bodů, které získával za pěšce na polích svojí barvy a postavené věže. Roboti navíc musí mít vyřešenou detekci soupeře, aby do sebe nenaráželi (např. pomocí ultrazvukových měřičů vzdálenosti). Kompletní pravidla, výsledková listina a další informace jsou na webu ročníku 2011[13].

Robot našeho týmu byl poměrně jednoduchý, přesto však obsahoval 5 ultrazvukových měřáků vzdálenosti, dva enkodéry a 5 tlačítek (detekce nárazu na mantinel a pěšce, kterého mohl robot převážet). Tyto senzory produkují poměrně značné množství dat, které se v terminálu zobrazuje nepřehledně.

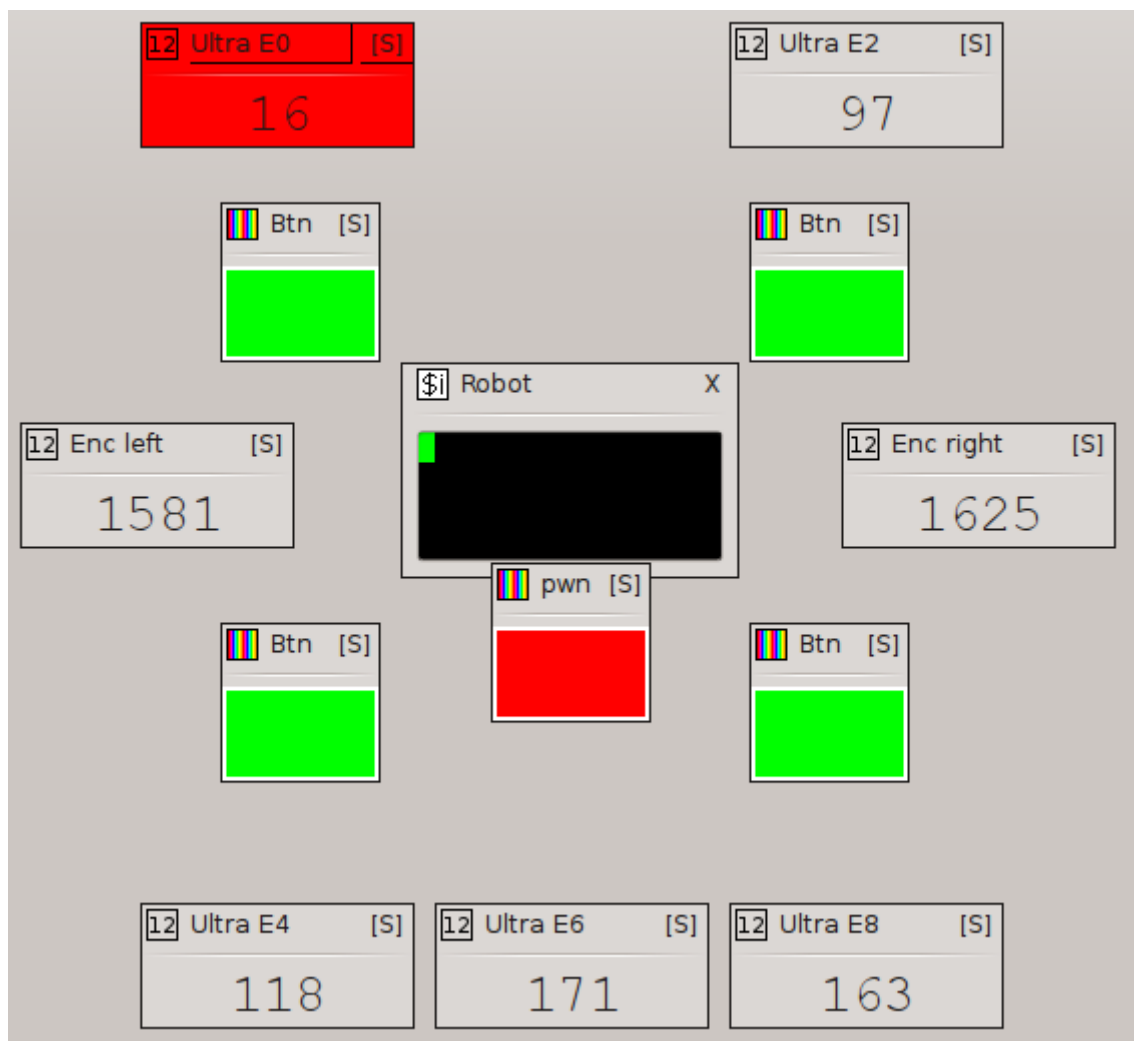


```
[12:28:50] Encoder event 13 setted
[12:28:50] Sending packet MSG_SET_MOVEMENT lenght 2
[12:28:50] Button adr 2 status 1
[12:28:50] range e6 105
[12:28:50] set motor pwr left 127 right 127
[12:28:50] range e8 233
[12:28:50] range f8 124
[12:28:51] range e2 112
[12:28:51] range e4 106
[12:28:51] range e6 105
```

Obrázek 32: Data z robota v terminálu

Zkušenost s programováním a laděním robota byla jedním z hlavních důvodů pro výběr zvoleného tématu práce. S použitím programu Lorris by se mohla tato data zobrazovat například takto:





Obrázek 33: Simulovaná data v analyzáru

Všechny widgety jsou rozmístěné stejně jako na robotovi – 2 ultrazvuky měří vzdálenost vpředu, 3 vzadu; tlačítka pro detekci mantinelu jsou na každém rohu, tlačítko uvnitř robota ověřuje, zda robot nabral pěšce a enkodéry na obou kolech měří ujetou vzdálenost.

Widget *script* s názvem „Robot“ uprostřed reprezentuje tělo robota. Widgety *číslo* s názvy „Ultra E0...8“ ukazují vzdálenost z ultrazvukových měřáků. Ve widgetu „Ultra E0“ je vzdálenost menší než 25 cm, což je hra-

niční vzdálenost, po které se robot zastaví, aby nevrazil do soupeře – aby widget na tuto skutečnost upozornil, má červené pozadí.

Widgety *barva* s názvy „btn“ jsou tlačítka značící náraz do mantinelu a „pwn“ je tlačítko, které se stiskne, pokud je v robotovi pěšec. Tlačítko, které má zelenou barvu, není stisklé, tlačítko s červenou barvou stisklé je.

Poslední widgety *číslo* s názvy „Enc left“ a „Enc right“ vypisují ujetou vzdálenost z enkodérů pravého a levého kola.



Obrázek 34: Náš robot *David* skončil na 4. místě v celostátním kole soutěže Eurobot 2011

## 7 Podpora joysticku

Lorris podporuje použití joysticku v modulu analyzér například k řízení robota. Nejdříve jsem k přístupu k joysticku používal knihovnu SDL[14], která však pro moje použití nebyla příliš vhodná – její předpokládané použití je v počítačových hrách, a podpora joysticku je jen jedna z mnoha částí které knihovna obsahuje. Její architektura se také nehodila ke zbytku Lorris.

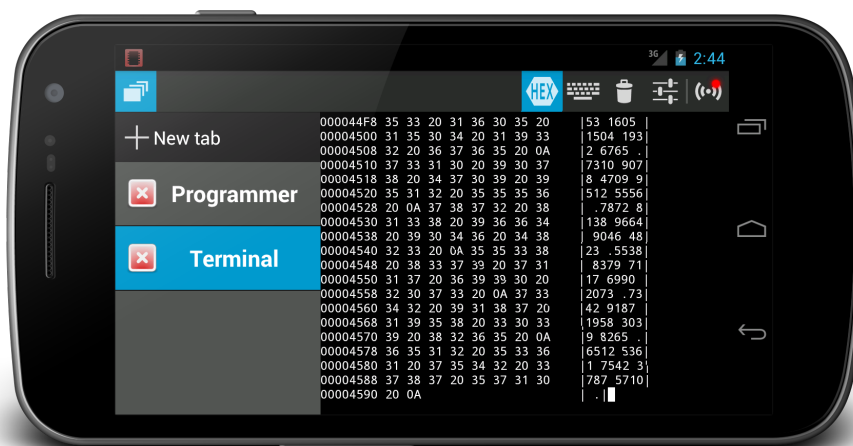
Při hledání náhrady jsem ale nenašel žádnou vhodnou knihovnu, jejíž funkce by byla pouze přistupování k joysticku a neměla žádné zbytečné funkce které bych v Lorris nepoužil, a tak jsem si napsal vlastní knihovnu.

Jmenuje se **libenjoy**, funguje pod Windows a Linuxem a je velmi malá a jednoduchá. Oproti SDL si dokáže zapamatovat připojené joysticky, a když joystick odpojíte a zase ho připojíte zpět (například kvůli přeskládání kabelů u počítače nebo se joystick odpojí sám kvůli špatnému kontaktu), tak není třeba ho znovu vybírat jako aktivní – sám se připojí bez jakékoliv interakce uživatele.

Libenjoy je vydána pod licencí GNU LGPLv2.1[22].

- GIT repozitář: <https://github.com/Tassadar/libenjoy>

## 8 Aplikace pro Android

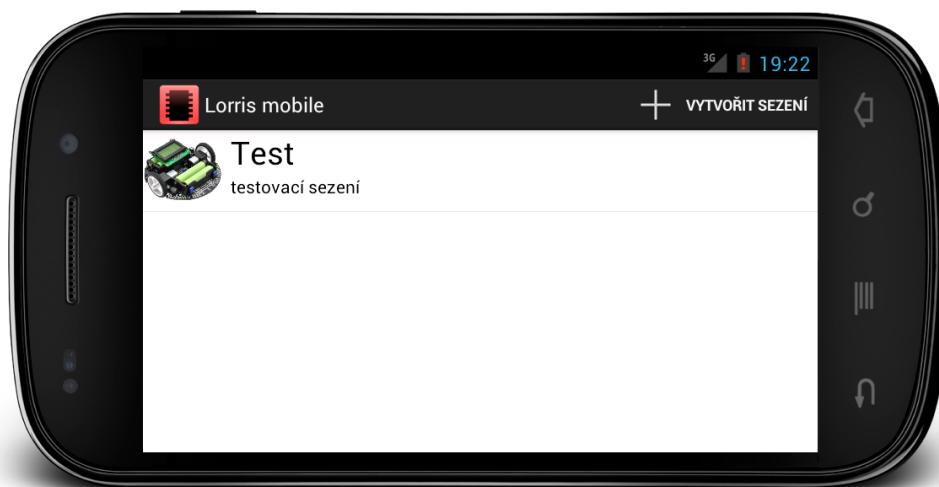


Obrázek 35: Lorris mobile

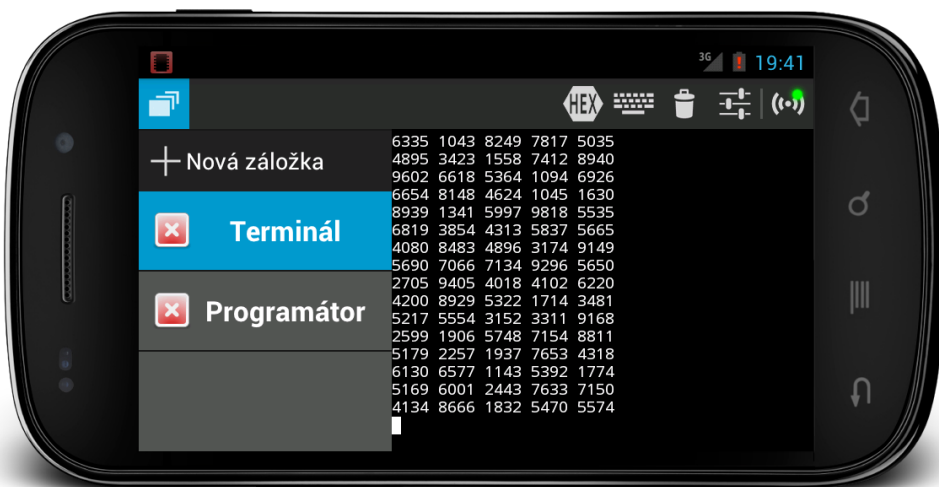
Aplikace **Lorris mobile** pro operační systém Google Android<sup>tm</sup> slouží jako přenosný doplněk k počítačové verzi Lorris – nemusí nutně obsahovat všechny funkce dekstopové aplikace ale má pomoci zejména když je v terénu potřeba rychle něco přenastavit či poupravit.

Aplikace funguje telefonech a tabletech s OS Android ve verzi 2.2 a vyšší, je optimalizována i pro větší obrazovky tabletů a je dostupná v oficiálním distribučním kanále Android aplikací – v obchodě Google Play[17], stačí hledat heslo „Lorris“.

Lorris mobile má podobnou architekturu jako desktopová verze Lorris. Začíná se vytvořením sezení, do kterého se ukládá vše, co uživatel v aplikaci otevře (obrázek 36). Po vytvoření a otevření sezení se uživatel dostane na hlavní obrazovku programu, kde si může otevřít jednotlivé moduly v záložkách podobně jako v desktopové aplikaci (obrázek 37).

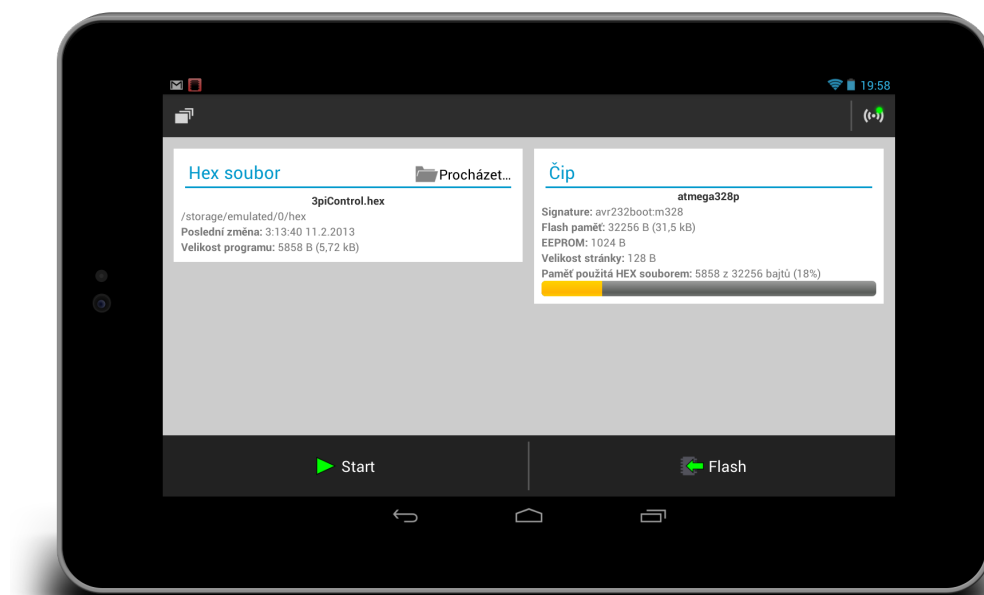


Obrázek 36: Lorriss mobile - výběr sezení



Obrázek 37: Lorriss mobile - přepínání záložek

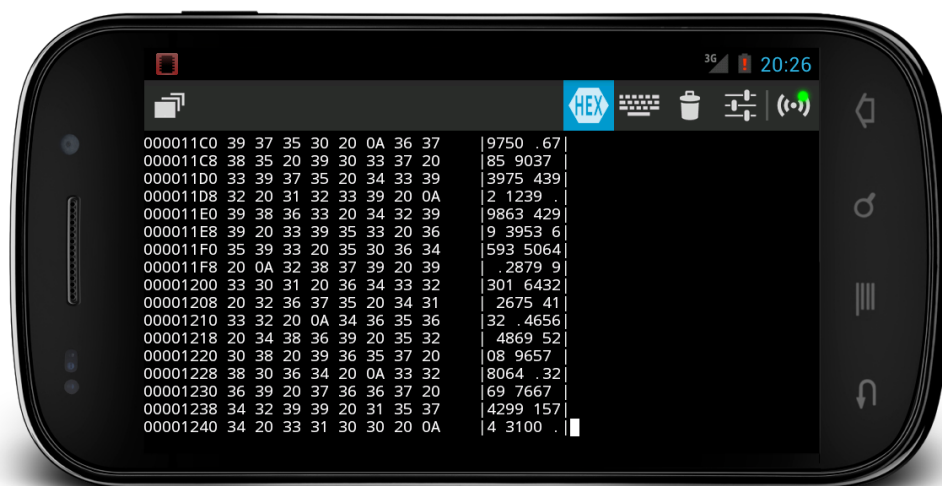
## 8.1 Programátor



Obrázek 38: Lorris mobile - programátor

Modul programátor dokáže programovat čipy pomocí bootloaderů **avr232boot** a **AVROSP** a také pomocí programátoru Shupito, pokud tablet nebo telefon na kterém aplikace běží podporuje připojení USB zařízení. Tato část Lorris mobile používá části nativního kódu z desktopové verze Lorris, díky tomu se kód lépe spravuje a je rychlejší.

## 8.2 Terminál



Obrázek 39: Lorris mobile - terminál

Klasický terminál. Umí většinu funkcí terminálu v desktopové verzi Lorris – zobrazuje data (jako text nebo hexadecimální hodnoty bajtů), odesílá stisky kláves, lze nastavit velikost a barva textu, barvu pozadí a jaké kontrolní znaky se mají odeslat při stisknutí klávesy enter.

## Závěr

Vytvořená aplikace splňuje všechny stanovené požadavky:

- ✓ 1. Možnost zpracovávat data přicházející ze zařízení a přehledně je zobrazovat
- ✓ 2. Podpora co nejvíce formátů příchozích dat
- ✓ 3. Snadné a rychlé používání
- ✓ 4. Možnost běhu i na jiných systémech než je MS Windows
- ✓ 5. Co možná nejnížší cena – program je dostupný zdarma
- ✓ 6. Snadná rozšiřitelnost (ideálně otevřený zdrojový kód)
- ✓ 7. Nezávislost na další aplikaci (např. MS Office Excel)

Program navíc výrazně přesáhl původní cíle – mimo zobrazování dat dokáže posílat data i zpět do zařízení, programovat mikročipy a vytvořit proxy mezi sériovým portem a TCP socketem. Ve srovnání s nalezenými programy s podobným zaměřením (viz úvod) je také jediný, který umožňuje uživateli napsat vlastní script pro parsování dat.

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 přidávat buďto 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
-----
```

Obrázek 40: Počet řádků spočítaný programem CLOC[15]

Kromě přidávání dalších vlastností do tohoto programu bych v budoucnu rád vytvořil podobný program (hlavně vlastnosti modulu Analyzář) 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 Qt-Scriptu, 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):

```
1  function onChanged(data, dev, cmd, index) {  
2      return "";  
3  }  
4  
5  function onKeyPress(key) {  
6  }  
7  
8  function onRawData(data) {  
9  }  
10  
11 function onWidgetAdd(widget, name) {  
12 }  
13  
14 function onWidgetRemove(widget, name) {  
15 }  
16  
17 function onScriptExit() {  
18 }  
19  
20 function onSave() {  
21 }
```

Příklad 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 **Integer**y 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 přijdou nějaká 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 analyzáru. 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.

```

1 function onKeyPress(key) {
2     appendTerm(key); // vypise _key_ do terminalu
3 }

```

Příklad 6: Vypsání stisknutých kláves do terminálu

- `clearTerm()` – vyčistí terminál.

```

1 function onKeyPress(key) {
2     if(key == "c")
3         clearTerm(); // vycisti terminal
4     else
5         appendTerm(key); // vypise _key_ do terminalu
6 }

```

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

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

```

1 function onKeyPress(key) {
2     sendData(key);
3 }

```

Příklad 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 QWidgetu. Může být 0, widget se poté vytvoří v levém horním rohu aktuálně viditelné plochy.

```

1 var cislo = newWidget(WIDGET_NUMBER,
2     "rychlost", 200, 100, -250, 0);
3
4 function onDataChanged(data, dev, cmd, index) {
5     cislo.setValue(data[0]);
6     return "";
7 }

```

Příklad 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<sup>13</sup> v kapitole „Properties“ a ve scriptu se používají takto:

```

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

```

Příklad 10: Vytvoření widgetu *číslo* a nastavení vlastnosti „visible“

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

<sup>13</sup><http://qt-project.org/doc/qt-4.7/qwidget.html#propertySection>

```

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

```

Příklad 11: Vytvoření widgetu *číslo* a použití slotu

Kromě těchto zděděných vlastností a funkcí má každý typ widgetu své 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

```

```

1 var cislo = newWidget(WIDGET_NUMBER,
2                       "test cislo", 200, 100, -250, 0);
3 cislo.setValue(40);
4 cislo.setFormula("%n-100");
5 ...
6 cislo.setValue(3.14);

```

Příklad 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 aktuální hodnotu

```

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

```

Příklad 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)` – Nastaví formát vstupu. Konstanty:  
  
`COLOR_RGB_8`, `COLOR_RGB_10`, `COLOR_RGB_10_UINT`,  
`COLOR_GRAY_8`, `COLOR_GRAY_10`

```

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

```

Příklad 14: Nastavení hodnot widgetu *barva*

## Widget graf

Tento widget se od ostatních poměrně výrazně liší – je třeba nejdříve vytvořit křivku až te 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

```

1  var graf = newWidget(WIDGET_GRAPH, "graf", 400, 250, -420, 0);
2  graf.setAxisScale(false, -105, 105); // meritko osy y
3  graf.setAxisScale(true, 0, 200); // meritko osy x
4
5  // vytvoreni krivky sin
6  var sin = graf.addCurve("sin", "blue");
7
8  // pridani bodu do krivky sin
9  var sinVal = 0;
10 for(var i = 0; i < 500; ++i) {
11     sin.addPoint(i, Math.sin(sinVal)*100);
12     sinVal += 0.1;
13 }
14 // presunuti na posledni hodnotu krivky
15 graf.updateVisibleArea();

```

Příklad 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<sup>14</sup>, například tlačítko (QPushButton), zaškrťovací 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.

<sup>14</sup><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).

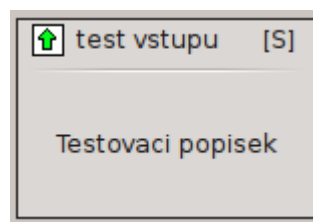
```

1  var vstup = newWidget(WIDGET_INPUT,
2      "test vstupu", 150, 100, -160, 0);
3  var label = vstup.newWidget("QLabel", 1);
4
5  // zarovnani textu na stred.
6  // 0x0080 a 0x0004 jsou konstanty Qt Frameworku
7  // Qt::AlignHCenter a Qt::AlignVCenter
8  label.alignment = 0x0080 | 0x0004;
9
10 // nastaveni textu
11 label.text = "Testovací popisek";

```

Příklad 16: Widget *vstup* – vytvoření QLabel

Widget vytvořený tímto příkladem vypadá takto:

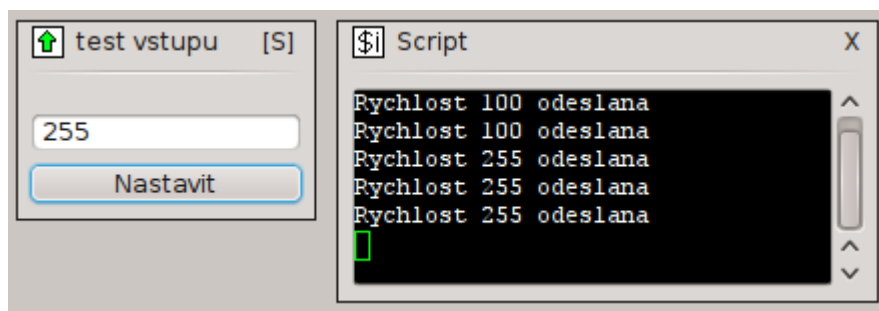


Obrázek 41: Widget *vstup* – vytvoření QLabel

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

```
1 var vstup = newWidget(WIDGET_INPUT,  
2     "test vstupu", 150, 100, -160, 0);  
3  
4 var rychlost = vstup.newWidget("QLineEdit");  
5 rychlost.text = "100";  
6  
7 var btn = vstup.newWidget("QPushButton", 1);  
8 btn.text = "Nastavit";  
9  
10 function posliRychlost() {  
11     var speed = parseInt(rychlost.text);  
12     sendData(new Array(speed));  
13     appendTerm("Rychlost " + speed + " odeslana\n");  
14 }  
15 // Pripojeni signalu "clicked" na slot posliRychlost()  
16 btn.clicked.connect(posliRychlost);
```

Příklad 17: Widget *vstup* – tlačítko



Obrázek 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)` – Nastaví vstupní formát. Konstanty: `ANG_RAD`, `ANG_DEG`, `ANG_RANGE`

```
1 var c = newWidget(WIDGET_CIRCLE, "kolo", 200, 200, -210, 0);
2
3 c.setAngType(ANG_DEG); // nastaveni vstupu na stupne
4 c.setValue(270);
```

Příklad 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.

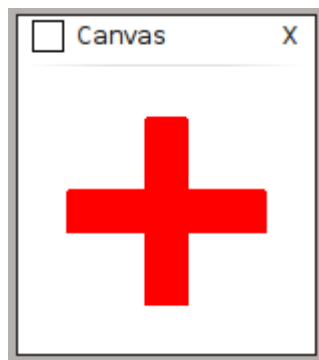
- `setLineColor(String barva)` – Barva čáry, kterou se prvky kreslí.
- `setFillColor(String barva)` – Barva výplně obdélníků, elips a kruhů

```

1 var c = newWidget(WIDGET_CANVAS, "Canvas", 140, 170, -150, 0);
2 c.setLineColor("red");
3 c.setFillColor("red");
4
5 c.drawRect(55, 10, 20, 110);
6 c.drawRect(10, 55, 110, 20);

```

Příklad 19: Nakreslení kříže ve widgetu *plátno*



Obrázek 43: Nakreslení kříže ve widgetu *plátno*

## Widget status

- `addStatus(int id, bool bitMaska, String text, String barvaPozadí, String k...`  
– Přidá nový status. `bitMaska` 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

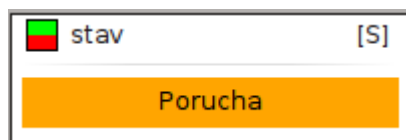
- `getValue()` – Vrábí aktuální hodnotu
- `showStatusManager()` – Vyvolá dialog pro správu stavů ve widgetu

```

1 var s = newWidget(WIDGET_STATUS, "stav", 0, 0, 200, 0);
2
3 s.addStatus(2, false, "Porucha", "orange", "black");
4 s.setValue(2);

```

Příklad 20: Ovládání widgetu *status* ze scriptu



Obrázek 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



```
1  var t = newWidget(WIDGET_BUTTON, "tlacitko", 0, 0, 200, 0);
2
3  t.setButtonName("Pokus");
4  t.setShortcut("Ctrl+H");
5  t.setColor("white");
6
7  function tlacitko_clicked() {
8      appendTerm("Tlacitko stisknuto!\n");
9  }
```

Příklad 21: Nastavení widgetu *tlacitko* 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`.

```
1  function Slider_valueChanged() {
2      appendTerm("hodnota: " + Slider.getValue() + "\n");
3  }
4
5  function Slider_minimumChanged() {
6      appendTerm("nove minimum: " + Slider.getMin() + "\n");
7  }
8
9  function Slider_maximumChanged() {
10     appendTerm("nove maximum: " + Slider.getMax() + "\n");
11 }
12
13 function Slider_typeChanged() {
14     appendTerm("Typ vstupu zmene na " +
15         (Slider.isInteger() ? "Integer" : "Double") + "\n");
16 }
17
18 function Slider_orientationChanged() {
19     appendTerm("orientace zmenena na " +
20         Slider.getOrientation() + "\n");
21 }
```

Příklad 22: Funkce, které 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()` – Vráťí minimální, maximální a aktuální hodnotu

```

1  var s = newWidget(WIDGET_SLIDER, "Slider", 0, 0, 300, 0);
2
3  s.setType(true); // desetinná čísla
4  s.setMax(6.28);
5  s.setValue(3.14);
6
7  function Slider_valueChanged() {
8      appendTerm("value changed: " + Slider.getValue() + "\n");
9  }

```

Příklad 23: Ovládání widgetu *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)` – Vrábí `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.

```

1  var s = newWidget(WIDGET_SLIDER, "Slider", 0, 0, 300, 0);
2  s.setType(true); // desetinná čísla
3  s.setMax(6.28);
4
5  // Nacte hodnotu, která byla před tím uložena v metodě save()
6  var ulozene = storage.getFloat("hodnotaPosuvniku", 3.14);
7  s.setValue(ulozene);
8
9  // Nacte pokusné pole čísel uložene v metodě save()
10 var pokus = storage.getInt32Array("pokusnePole", new Array());
11 appendTerm("Uložene pole: " + pokus + "\n");
12
13 function onSave() {
14     save();
15 }
16
17 function onScriptExit() {
18     save();
19 }
20
21 function save() {
22     storage.setFloat("hodnotaPosuvniku", Slider.getValue());
23
24     var pokus = new Array(4, 8, 15, 16, 23, 42);
25     storage.setInt32Array("pokusnePole", pokus);
26 }

```

Příklad 24: Ukládání dat scriptu

## Přístup k joysticku

Nejříve několik globálních metod pro práci s joysticky:

- `getJoystickNames()` – Vrátí pole Stringů se jmény připojených joysticků.
- `getJoystickIds()` – Vrátí 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 vrátí object Joystick nebo NULL pokud nebylo možné joystick otevřít.
- `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 změní stav tlačítka. Parametr `tlačítko` je index tlačítka a `stav` je číslo 0 nebo 1.

```

1  // Pokusi se otevrit prvni dostupny joystick
2  var ids = getJoystickIds();
3  var joy = getJoystick(ids[0]);
4
5  if(joy) {
6      // pripojeni na signaly
7      joy.axesChanged.connect(axesChanged);
8      joy.buttonChanged.connect(buttonChanged);
9
10     appendTerm("ID joysticku: " + joy.getId() + "\n");
11     appendTerm("Pocet os: " + joy.getNumAxes() + "\n");
12     appendTerm("Pocet tlacitek: " +
13         joy.getNumButtons() + "\n");
14 }
15
16 function axesChanged(axes) {
17     for(var i = 0; i < axes.length; ++i) {
18         var hodnota = joy.getAxisVal(axes[i]);
19         appendTerm("Osa " + axes[i] + ": " + hodnota + "\n");
20     }
21 }
22
23 function buttonChanged(id, state) {
24     appendTerm("Tlacitko " + id + ", stav: " + state + "\n");
25 }

```

Příklad 25: Otevření joysticku a čtení jeho hodnot

## **Pár věcí, na které je třeba při programování myslet**

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



## PŘÍLOHA B:

### Knihovny třetích stran

- **Qwt**[18] je knihovna pro Qt Framework obsahující tzv. widgety pro aplikace technického charakteru – grafy, sloupcové ukazatele, kompas a podobně. Ve svojí práci zatím z této knihovny používám pouze graf (v modulu analyzáru).
- **QExtSerialPort**[19] poskytuje připojení k sériovému portu a také dokáže vypsat seznam nalezených portů v počítači.
- **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:

### Reference

- [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:

### Seznam obrázků

1	Tab creation dialog . . . . .	8
2	Window divided to multiple parts . . . . .	8
3	New update notification . . . . .	9
4	Ongoing update . . . . .	10
5	Modul analyzer . . . . .	11
6	Widget alignment using grid and lines . . . . .	12
7	Packet structure dialog . . . . .	13
8	Widgety . . . . .	14
9	Filter settings . . . . .	15
10	Widget: number . . . . .	15
11	Widget: bar . . . . .	16
12	Widget: color . . . . .	17
13	Widget: graph . . . . .	18
14	Curve settings dialog . . . . .	18
15	Widget: script . . . . .	19
16	Script editor . . . . .	20
17	Widget: circle . . . . .	21
18	Widget: plátno . . . . .	21
19	Widgety tlačítko a slider . . . . .	22
20	Widget <i>vstup</i> s nastavením joysticku . . . . .	23
21	Widget status . . . . .	24
22	Nastavení stavů . . . . .	25
23	Widget terminál . . . . .	25
24	Proxy mezi sériovým portem a TCP socketem . . . . .	27
25	Modul Programátor . . . . .	28
26	Zmenšené UI modulu <i>programátor</i> (nalevo) s otevřeným <i>terminálem</i> . . . . .	29

27	Modul terminál . . . . .	31
28	Barva v modulu Analyzér . . . . .	32
29	Surová data z enkodérů s vyznačenými bajty, které obsahují úhel natočení . . . . .	33
30	Data z enkodérů zpracovaná analyzérem . . . . .	33
31	Ladění PID regulátoru . . . . .	34
32	Data z robota v terminálu . . . . .	35
33	Simulovaná data v analyzáru . . . . .	36
34	Náš robot <i>David</i> skončil na 4. místě v celostátním kole sou- těže Eurobot 2011 . . . . .	37
35	Lorris mobile . . . . .	39
36	Lorris mobile - výběr sezení . . . . .	40
37	Lorris mobile - přepínání záložek . . . . .	40
38	Lorris mobile - programátor . . . . .	41
39	Lorris mobile - terminál . . . . .	42
40	Počet řádků spočítaný programem CLOC[15] . . . . .	44
41	Widget <i>vstup</i> – vytvoření QLabel . . . . .	55
42	Widget <i>vstup</i> – tlačítko . . . . .	56
43	Nakreslení kříže ve widgetu <i>plátno</i> . . . . .	58
44	Ovládání widgetu <i>status</i> ze scriptu . . . . .	59