

**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 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:

## **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.



## Anotace

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

Protože se nejedná o jednoduchou aplikaci ani o jednostranně zaměřenou sadu nástrojů, protože se navíc celá sada průběžně rozrůstá a protože je rozsah a záběr použití všech funkcí a modulů této sady příliš velký, nelze ji stručně popsat v omezeném prostoru anotace.

Pro získání ucelenější představy o tom, co vše lze pomocí sady Lorris dosáhnout, prosím nahlédněte do úvodu práce.

Hlavním přínosem tohoto softwarového balíku je, že urychluje, zpřehledňuje a hlavně výrazně zjednoduší vývoj a testování aplikací pro mikročipy, typicky programování a řízení různých druhů robotů.

**Klíčová slova:** analýza binárních dat, programování a řízení robotů, vývoj pro mikrokontroléry, programování mikročipů

## Annotation

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

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

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

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

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

# Obsah

<b>Úvod</b>	4
Modul: analyzér	4
Modul: programátor	5
Modul: terminál	5
Modul: proxy mezi sériovým portem a TCP socketem	5
<b>1 Motivace</b>	6
1.1 Požadavky na aplikaci	6
1.2 Existující programy	7
1.3 Porovnání aplikací	8
<b>2 Popis rozhraní</b>	8
2.1 Web a repozitář programu	8
2.2 Struktura aplikace	9
2.3 Sezení	11
2.4 Automatická aktualizace	11
<b>3 Modul: Analyzér</b>	12
3.1 Filtrování dat	16
3.2 Widget: číslo	17
3.3 Widget: sloupcový bar	18
3.4 Widget: barva	19
3.5 Widget: graf	20
3.6 Widget: script	21
3.7 Widget: kolo	23
3.8 Widget: plátno	23
3.9 Widgety tlačítka a slider	24
3.10 Widget: vstup	25
3.11 Widget: status	26
3.12 Widget: terminál	27
<b>4 Modul: Proxy mezi sériovým portem a TCP socketem</b>	28
4.1 Proxy tunel	28
<b>5 Modul: Programátor</b>	29
5.1 Programátor Shupito	30

5.1.1	UART tunel . . . . .	31
5.2	Bootloader avr232boot . . . . .	31
5.3	Bootloader AVROSP . . . . .	32
<b>6</b>	<b>Modul: Terminál . . . . .</b>	<b>33</b>
<b>7</b>	<b>Podpora joysticku . . . . .</b>	<b>34</b>
<b>8</b>	<b>Příklady použití . . . . .</b>	<b>35</b>
8.1	Testování barevného senzoru . . . . .	35
8.2	Testování enkodéru . . . . .	36
8.3	Ladění PID regulátoru . . . . .	40
8.4	Stavba robota pro soutěž Eurobot 2011 . . . . .	41
8.4.1	Mechanická kostra robota . . . . .	42
8.4.2	Ladění a nastavení senzorů . . . . .	43
8.4.3	Programování reaktivního chování robota . . . . .	44
<b>9</b>	<b>Aplikace pro Android . . . . .</b>	<b>45</b>
9.1	Programátor . . . . .	47
9.2	Terminál . . . . .	48
<b>10</b>	<b>Reálné nasazení . . . . .</b>	<b>49</b>
<b>Závěr</b>	<b>51</b>	
<b>PŘÍLOHA A: Reference k widgetu <i>script</i></b>	<b>53</b>	
Základní script . . . . .	54	
Základní funkce . . . . .	55	
Vytvoření widgetu . . . . .	57	
Dostupné funkce widgetů . . . . .	58	
Widget číslo . . . . .	59	
Widget sloupcový bar . . . . .	59	
Widget barva . . . . .	60	
Widget graf . . . . .	61	
Widget vstup . . . . .	62	
Widget kolo . . . . .	65	
Widget plátno . . . . .	65	
Widget status . . . . .	66	

Widget tlačítko . . . . .	67
Widget slider . . . . .	69
Ukládání dat scriptu . . . . .	71
Přístup k joysticku . . . . .	73
<b>PŘÍLOHA B: Knihovny třetích stran . . . . .</b>	<b>75</b>
<b>PŘÍLOHA C: Licence . . . . .</b>	<b>76</b>
<b>PŘÍLOHA D: Reference . . . . .</b>	<b>77</b>
<b>PŘÍLOHA E: Velké obrázky . . . . .</b>	<b>82</b>
<b>PŘÍLOHA F: Seznam obrázků . . . . .</b>	<b>85</b>

# Úvod

Lorris je rozsáhlá sada nástrojů, které mají společný cíl – pomáhat při vývoji, ladění a řízení zejména robotů, ale i jiných elektronických zařízení. V této kapitole najdete stručně popsány nejdůležitější vlastnosti jednotlivých modulů a dále v práci jsou pak všechny moduly důkladně popsány v jednotlivých kapitolách.

## Modul: analyzér

- Soustřeďuje se na zobrazování dat ze zařízení v grafické podobě.
- Analyzér pro zobrazování používá tzv. widgety – malá „okna“, která zobrazují určitou část dat.
- Widgety mají individuální nastavení a uživatel si je může umístit na libovolné místo na pracovní ploše.
- Lorris obsahuje několik typů widgetů, například *Číslo*, *Barva*, *Sloupcový bar*, *Kolo* (zobrazení úhlu v kružnici) či *Graf*.
- Pomocí widgetů lze sestavit rozhraní vyhovující prakticky jakémukoliv zařízení.
- Analyzér je ideální i pro snadné zobrazování dat z prvků, u kterých není vhodné jako výstup použít čísla – například barevný senzor.
- Některé widgety mohou posílat data i směrem do zařízení. Díky tomu je možné kromě zobrazování dat jejich prostřednictvím zařízení i ovládat.
- Pozornost si zaslouží widget „script“. Uživatel v něm může napsat vlastní script, který zpracovává příchozí data. Script může využít ostatní widgety a další části Lorris, díky tomu lze zobrazit i jinými způsoby interpretovat takřka jakémukoliv data.

- Pomocí scriptu lze upravovat i samotnou sadu Lorris.

## Modul: programátor

- Grafické rozhraní pro několik typů bootloaderů a programátorů pro mikročipy.
- Dokáže do čipu zapistovat program, číst a mazat paměť čipu nebo programovat pojistky.
- Oficiální GUI k programátoru Shupito.
- Shupito je programátor mikrokontrolérů. Na jeden konec programátoru se připojí čip, na druhý počítač – bez programátoru nelze do některých mikrokontrolérů nahrát program.

## Modul: terminál

- Klasický terminál - zobrazuje příchozí data jako text nebo vypisuje byty jako hexadecimální čísla.

## Modul: proxy mezi sériovým portem a TCP socketem

- Vytvoří server připojený na sériový port - k tomuto portu se pak lze připojit odkudkoliv z internetu.
- Umožňuje vzdáleně ladit, řídit, či jinak komunikovat se zařízením pomocí sítě Internet.

Anglická verze tohoto textu a propagační poster jsou dostupné na přiloženém CD ve formátu PDF.

*Enclosed CD contains English version of this work and promotional poster as PDF files.*

# **1 Motivace**

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

V zájmu zjednodušení a zrychlení vývoje robota jsem se rozhodl najít nějakou aplikaci, která by přehledně zobrazovala data z robota. Seznam požadavků, které jsem na tuto aplikaci stanovil, je sepsaný v kapitole 1.1.

## **1.1 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 nejnižší cena
6. Snadná rozšířitelnost (ideálně otevřený zdrojový kód)
7. Nezávislost na další aplikaci (např. MS Office Excel)

## 1.2 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.
- **LabVIEW**[5] je obrovský softwarový balík s dlouhou historií, který lze použít k velké škále operací – různá laboratorní měření, analýzu signálů, ovládání robotů, ovládání celých systémů pro laboratorní měření a mnoho dalšího. LabVIEW je ale uzavřený software a díky svému zaměření je velmi drahý, ceny základní verze začínají asi na 20 000 kč.

---

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

### 1.3 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	✓	✓	✗	✗	✓	✗	✓
LabVIEW	✓	✓	✗	✓	✗	✗	✓

Žádný z dostupných programů nesplňuje všechny požadavky, proto jsem se rozhodl napsat vlastní aplikaci, která bude obsahovat všechny stanovené vlastnosti.

## 2 Popis rozhraní

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

### 2.1 Web a repozitář programu

GIT<sup>2</sup> repozitář programu jsem vytvořil na serveru GitHub[8], 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.

---

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

- Repozitář: <https://github.com/Tasssadar/Lorris>
- Web (česká verze):  
<http://tasssadar.github.com/Lorris/cz/index.html>
- Web (anglická verze):  
<http://tasssadar.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.

## 2.2 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 zaráz a dokáže také rozdělit okno na několik částí, na obrázku 2 je nalevo modul analyzér a napravo terminál.

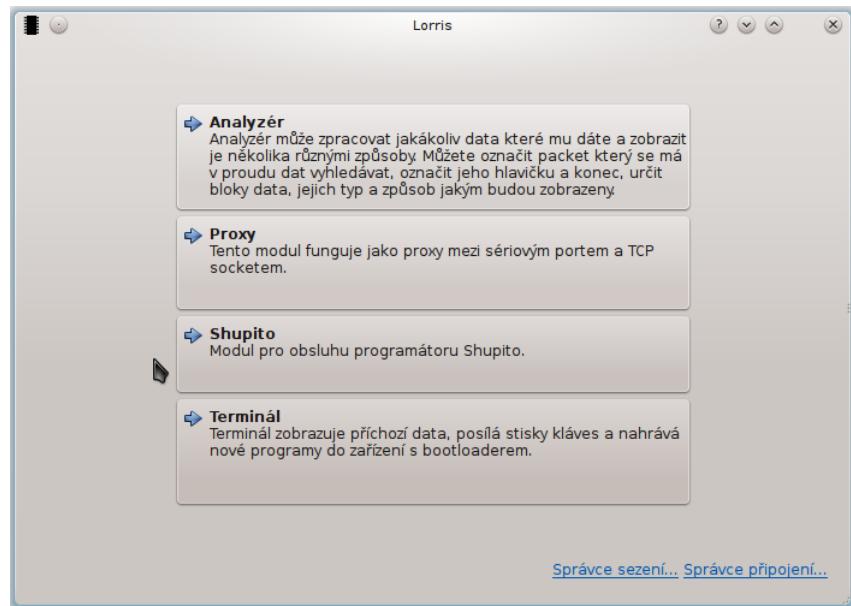
Možnosti připojení k zařízení:

- Sériový port
- Shupito Tunel (virtuální sériový port, viz kapitola 5.1.1)
- TCP socket<sup>3</sup>
- Načtení dat ze souboru

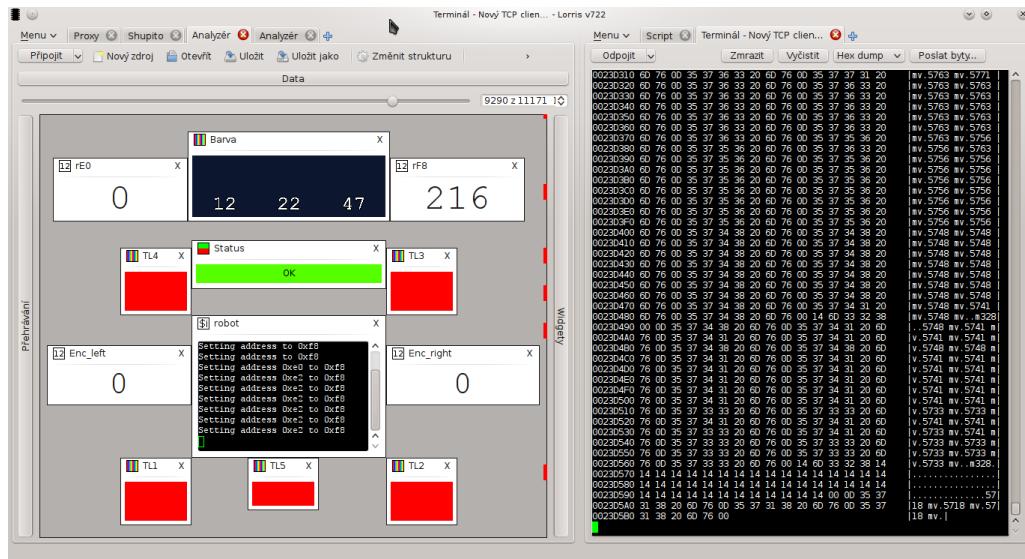
Je možné mít připojeno více různých modulů na jedno zařízení.

---

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



Obrázek 1: Dialog vytvoření panelu



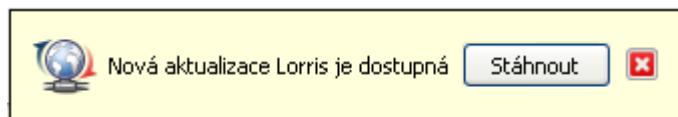
Obrázek 2: Ukázka rozdělení okna na více částí

## 2.3 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.

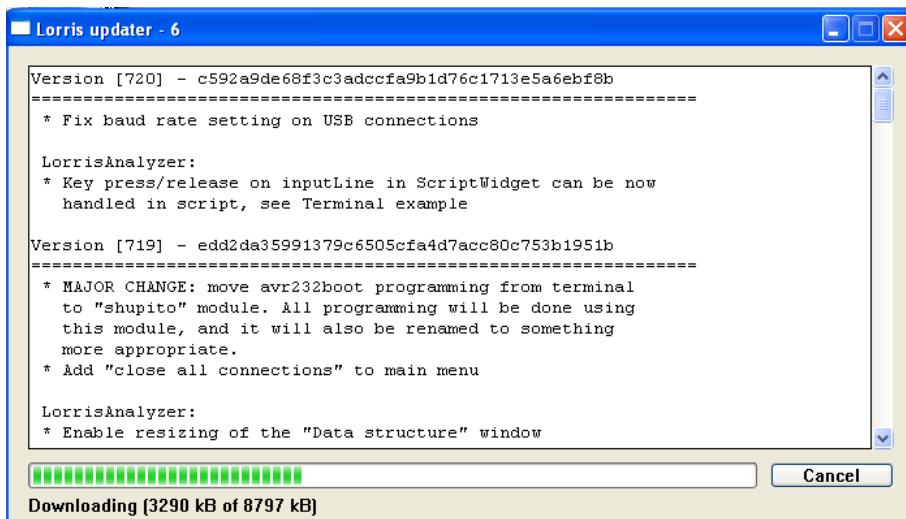
## 2.4 Automatická aktualizace

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



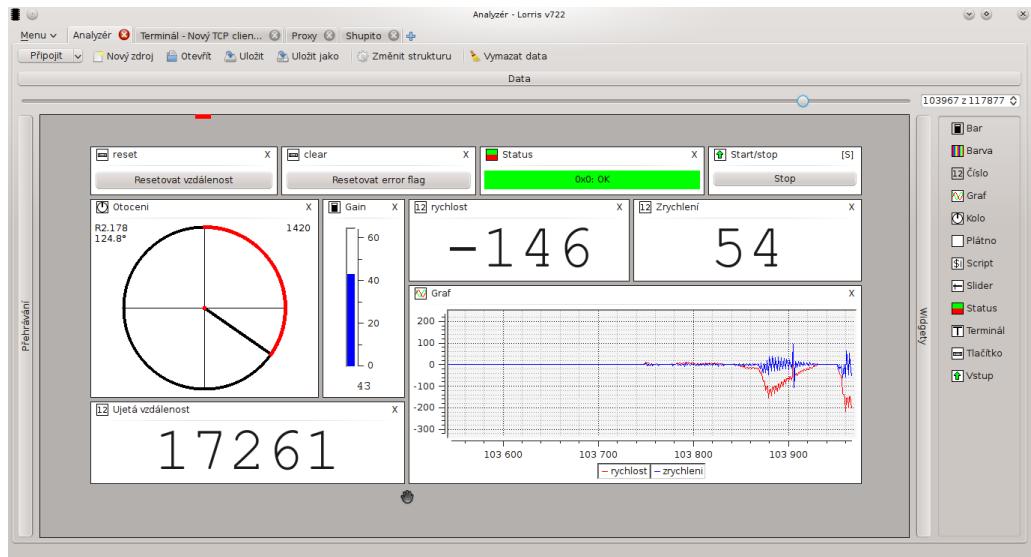
Obrázek 3: Upozornění o dostupné aktualizaci

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

### 3 Modul: Analyzér



Obrázek 5: Modul analyzér

Tento modul parsuje data (strukturované do packetů) přicházející ze zařízení a zobrazuje je v grafických „widgetech“. Zpracovaná data si aplikace ukládá do paměti – listování packety je možné pomocí posuvníku a boxu v horní části okna. Data (přijatá data, struktura packetů 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

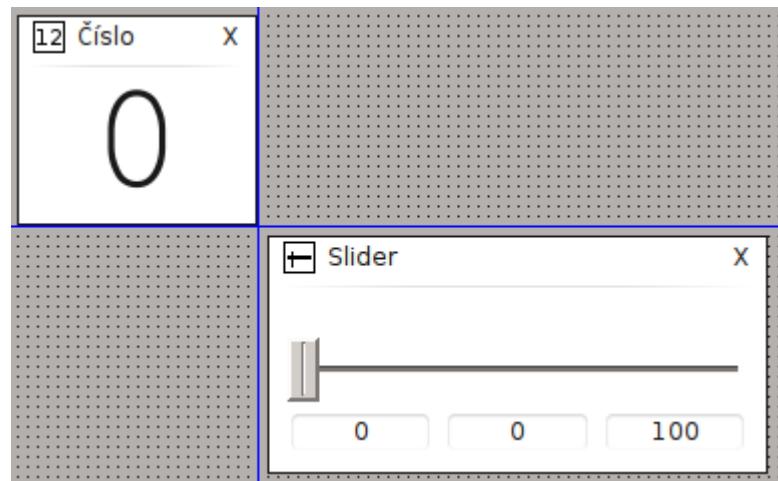
Po nastavení struktury se přijatá data začnou po packetech zobrazovat v horní části okna, a v pravé části se zobrazí sloupeček s dostupnými zobrazovacími widgety. Widgety se dají pomocí drag&drop principu „vytahat“ na plochu v prostřední části okna. Data se k widgetu přiřadí taktéž pomocí drag&drop,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

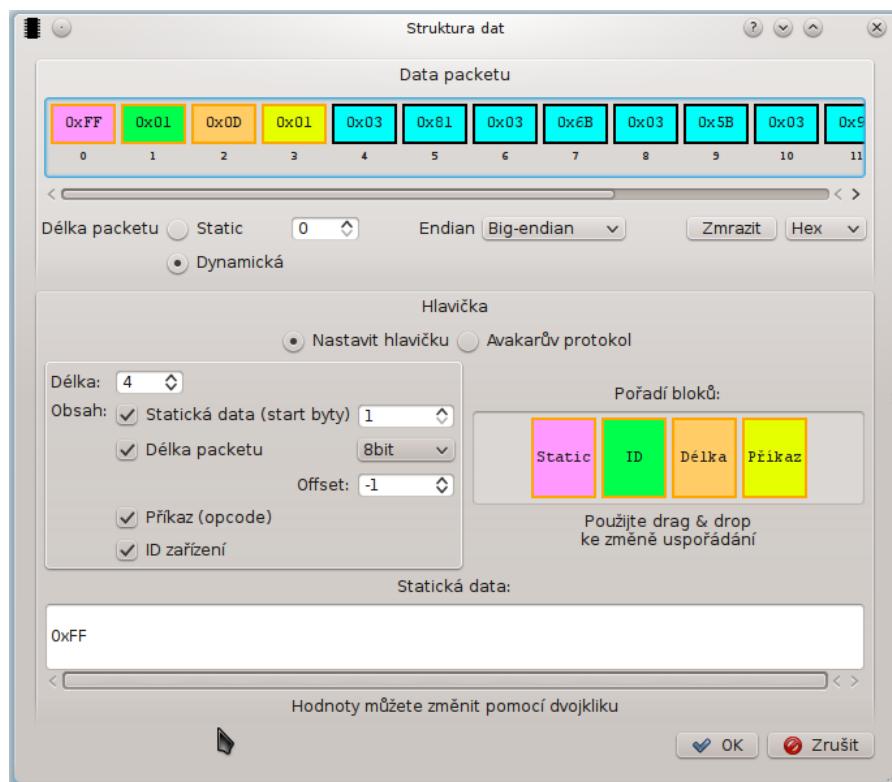
Nastavení widgetu jsou přistupná v kontextovém menu po pravém kliknutí myši na widget. Nastavit lze jméno a další parametry podle typu widgetu – podrobněji jsou možnosti nastavení popsány u jednotlivých widgetů. Widgety je taktéž možné „uzamknout“, aby nebylo možné je zavřít, měnit jejich pozici a velikost.

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.

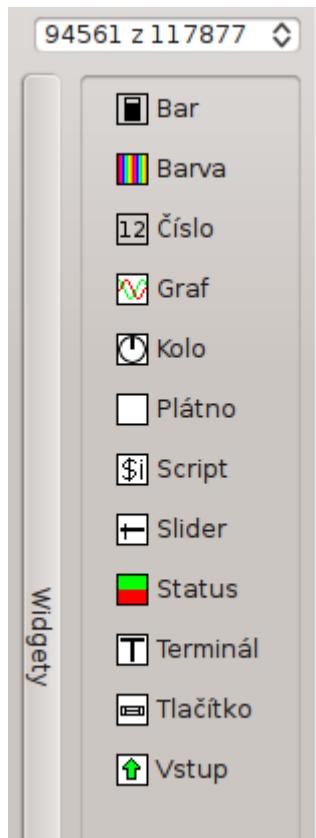
U některých widgetů se může hodit následující funkce: widgety je možné rychle zvětšit tak, aby zabraly celou viditelnou plochu pomocí gesta myši – stačí widget chytit jako při přesouvání a „zatřepat“ s ním zleva doprava. Při přesunutí se pak widget změní na svoji původní velikost.



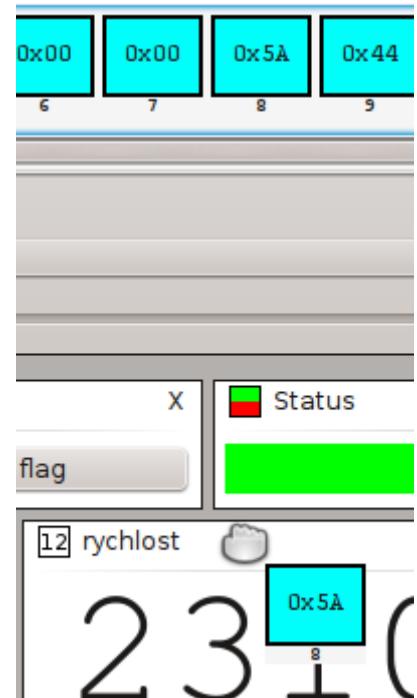
Obrázek 6: Zarovnávání widgetů pomocí sítě a čar



Obrázek 7: Dialog nastavení struktury dat



(a) Seznam widgetů

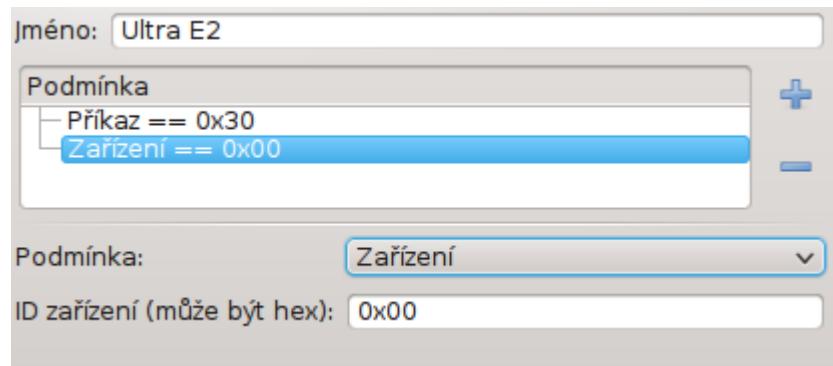


(b) Přiřazení dat pomocí  
drag&drop

Obrázek 8: Widgety

### 3.1 Filtrování dat

Analyzér umí příchozí data filtrovat, přičemž každý filtr může obsahovat několik podmínek podle kterých se určí, zda příchozí packet projde nebo ne.



Obrázek 9: Nastavení filtrování

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 podmínu pro filtrování.

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

Příklad 1: Script pro podmínu filtrování

### 3.2 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:

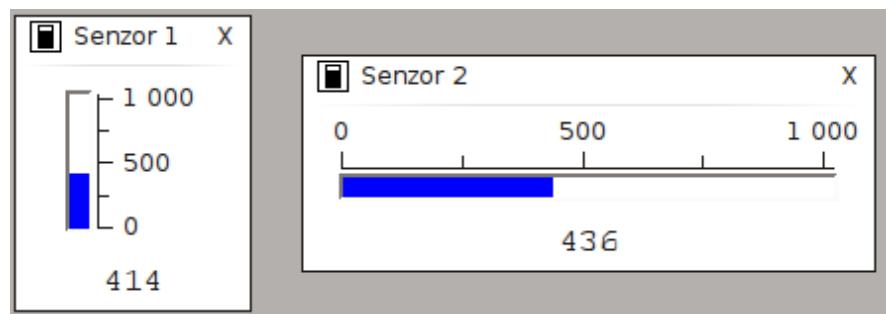
$$2914 / (%n+5) - 1$$

---

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

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.

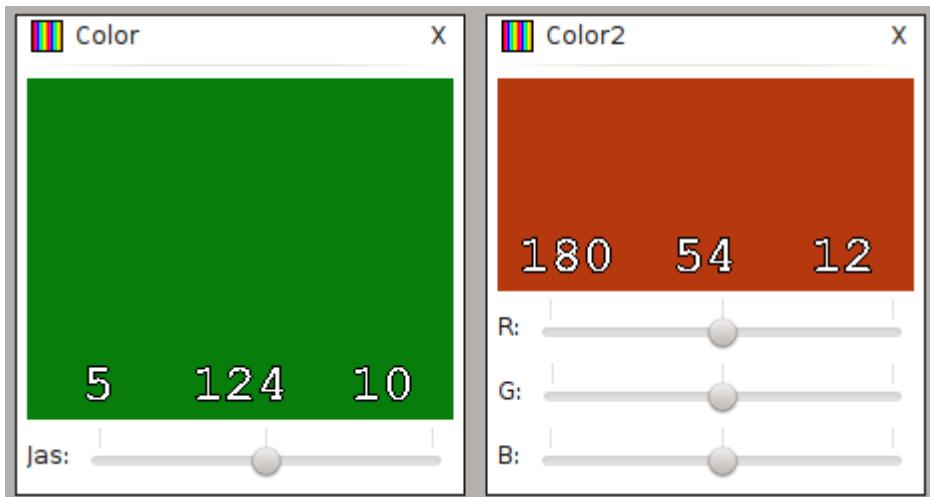
### 3.3 Widget: sloupcový bar



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 přepočívat hodnotu podle výrazu.

### 3.4 Widget: barva



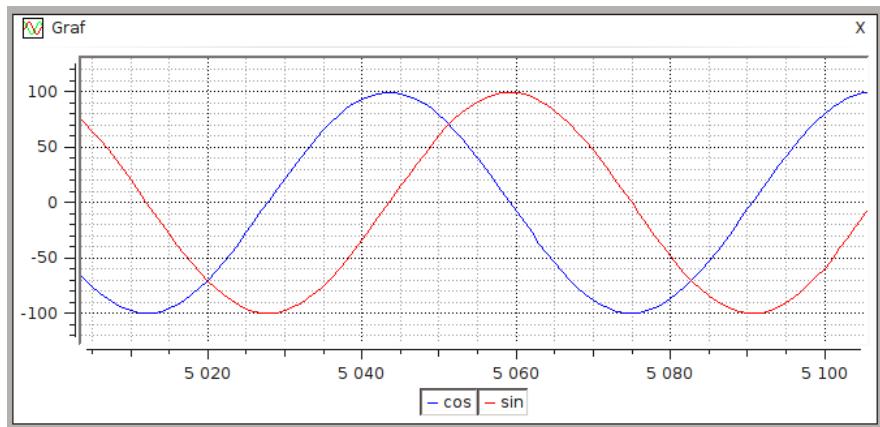
Obrázek 12: Widget: barva

Tento widget dokáže zobrazit příchozí hodnoty jako barevný obdélník. Porované 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)

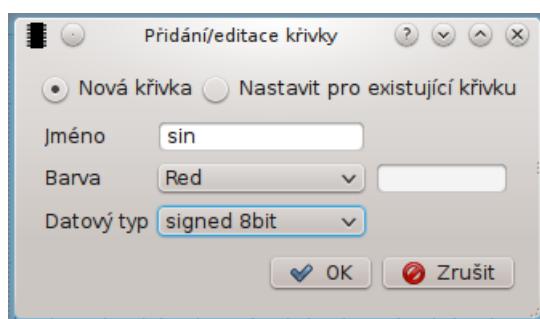
Widget také dokáže provést korekci jasu všech barev zaráz nebo každé z barev RGB zvlášť.

### 3.5 Widget: graf



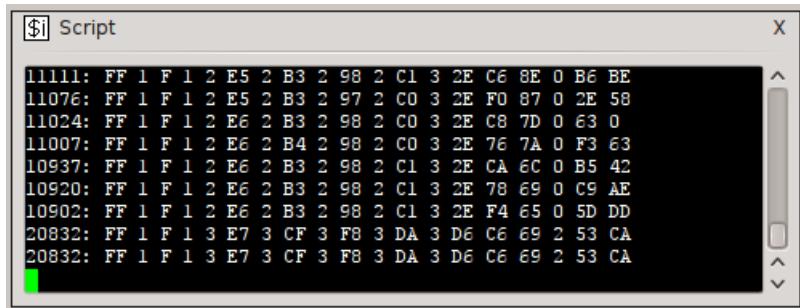
Obrázek 13: Widget: graf

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 osy grafu a zobrazení legendy. Kliknutí na křivku grafu v legendě tuto křivku skryje. Měřítko 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

### 3.6 Widget: script



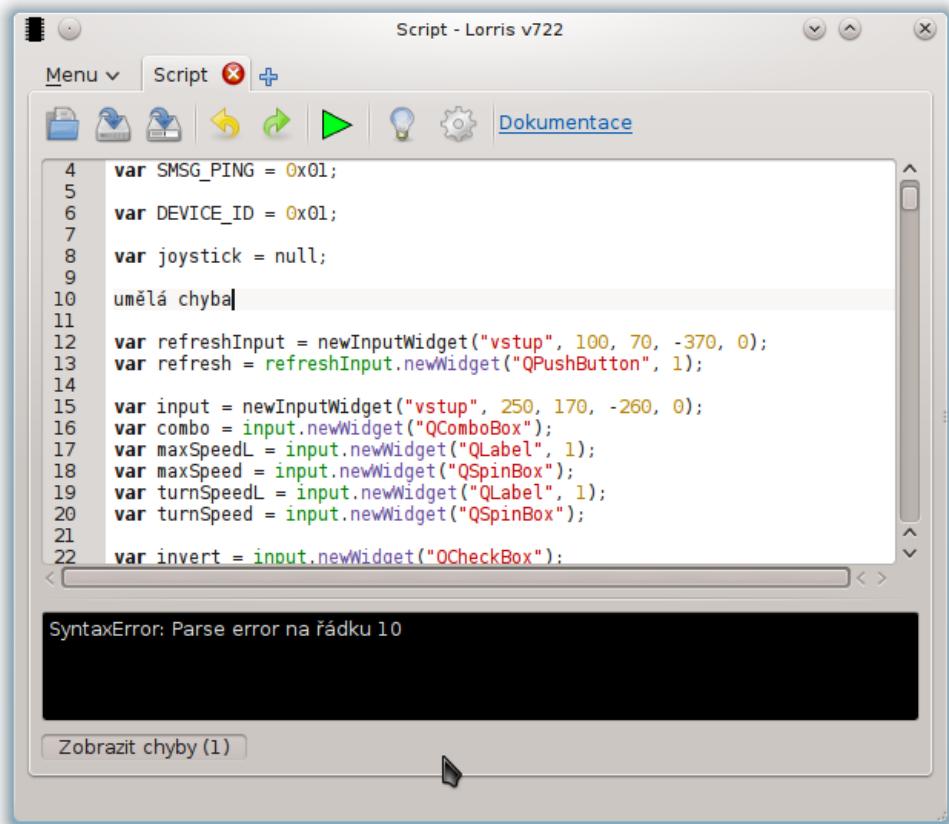
Obrázek 15: Widget: script

Tento widget umožňuje zpracovávání dat pomocí scriptu, který si napíše sám uživatel. Může při tom použít buďto Python nebo QtScript[9] (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 být zobrazen v terminálu (viz obrázek 15), je však možné využít ke zobrazení 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/>.

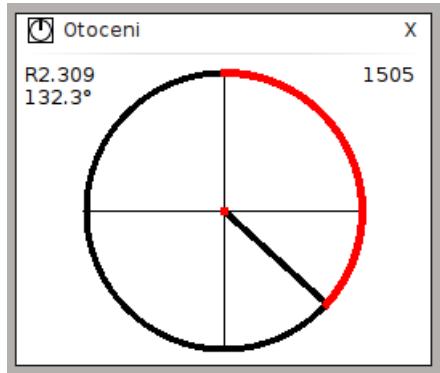
<sup>6</sup>ECMAScript – scriptovací jazyk standardu ECMA-262 a ISO/IEC 16262

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



Obrázek 16: Dialog pro nastavení zdrojového scriptu

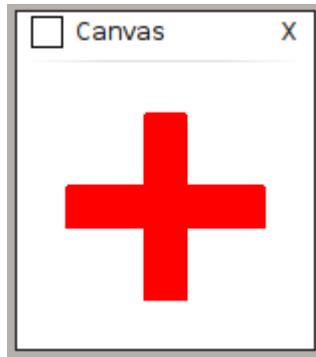
### 3.7 Widget: kolo



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 4095).

### 3.8 Widget: plátno



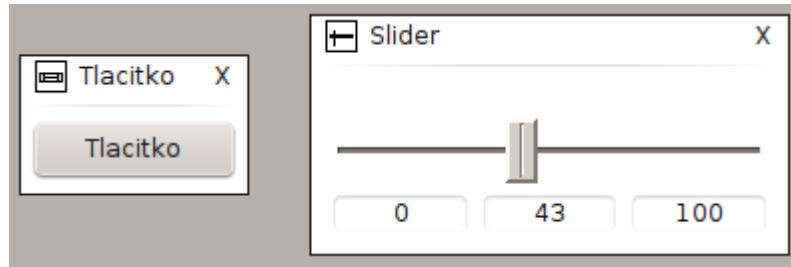
Obrázek 18: Widget: plátno

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

### 3.9 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 rychlosť robota a podobně. Tlačítku lze nastavit klávesovou zkratku a slideru zkratku pro „zaostření“, aby ho uživatel poté mohl posouvat pomocí šipek na klávesnici.

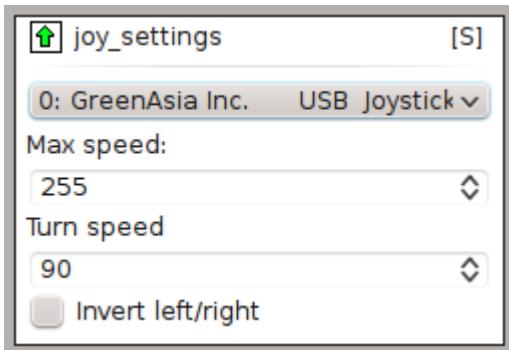
```

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*

### 3.10 Widget: vstup



Obrázek 20: Widget *vstup* s nastavením joysticku

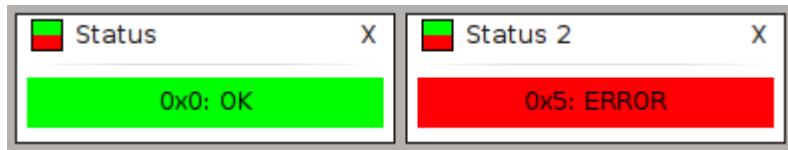
Tento widget je také určený k interakci se scriptem (tj. vstupu uživatele), přičemž script také určuje prvky rozhraní – widget je v základu prázdný a až script do něj přidá například tlačítko nebo textové pole. Tento widget je mírně složitější na obsluhu, ale lze díky němu použít všechny prvky UI, které Qt Framework obsahuje – tlačítka, posuvníky, textová pole, vysouvací seznamy a podobně. V příkladu 6 je script pro vytvoření prvků jako v obrázku 20.

```

1 // parametry: jméno 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 // Nastavení textu do QLabel
10 maxSpdLabel.text = "Max speed:";
```

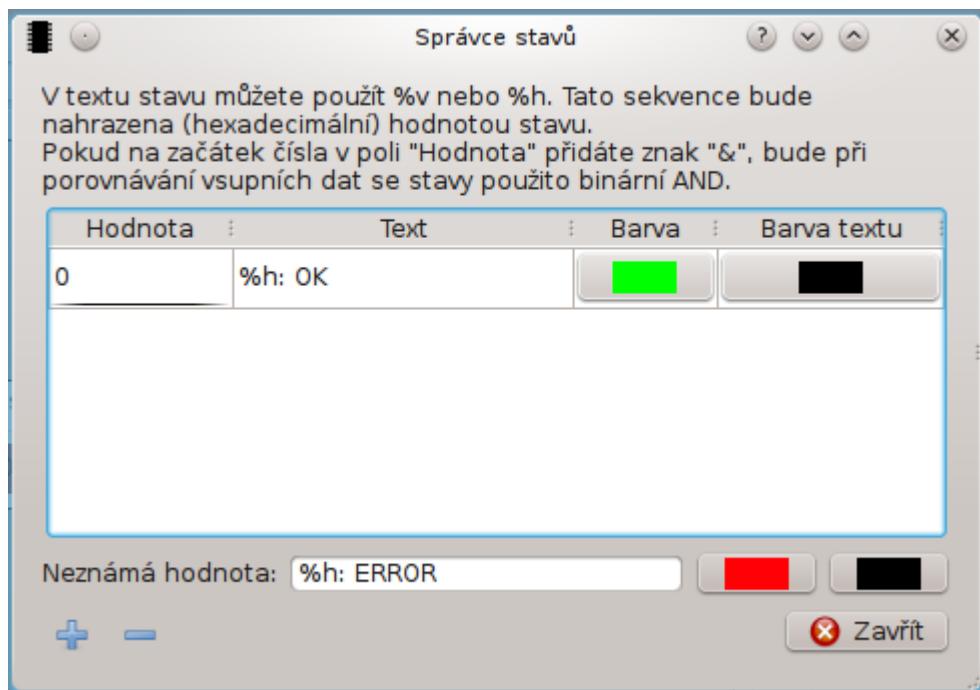
Příklad 4: Přidání prvků do widgetu *vstup*

### 3.11 Widget: status



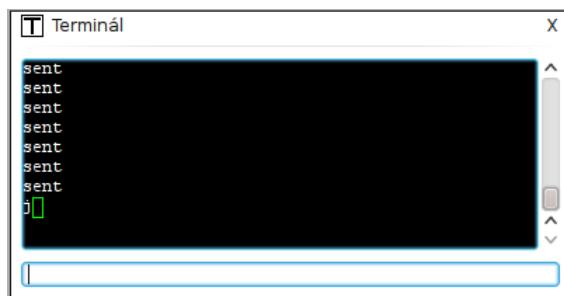
Obrázek 21: Widget status

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

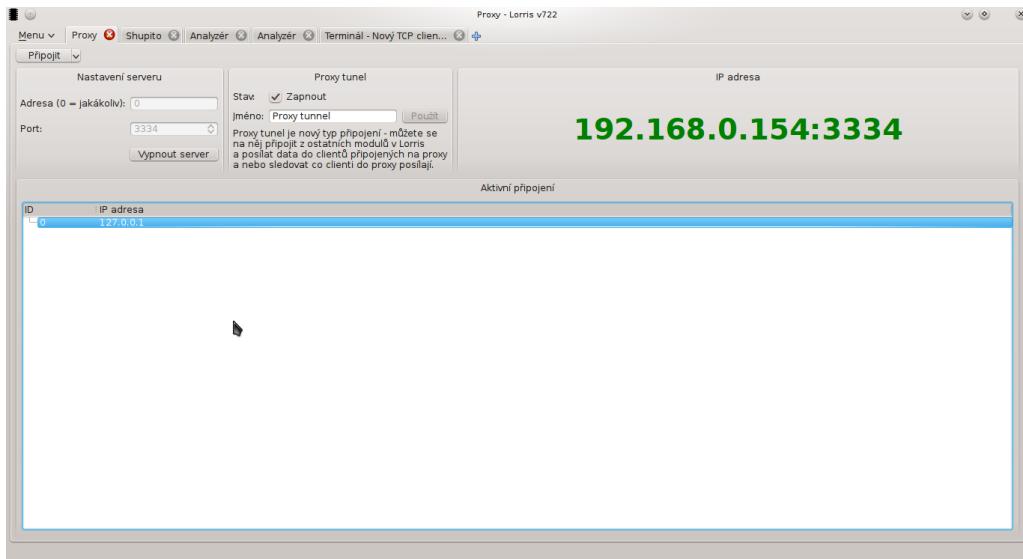
### 3.12 Widget: terminál



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 widget chová stejně jako terminál. Uživatel může script tohoto widgetu libovolně upravovat.

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



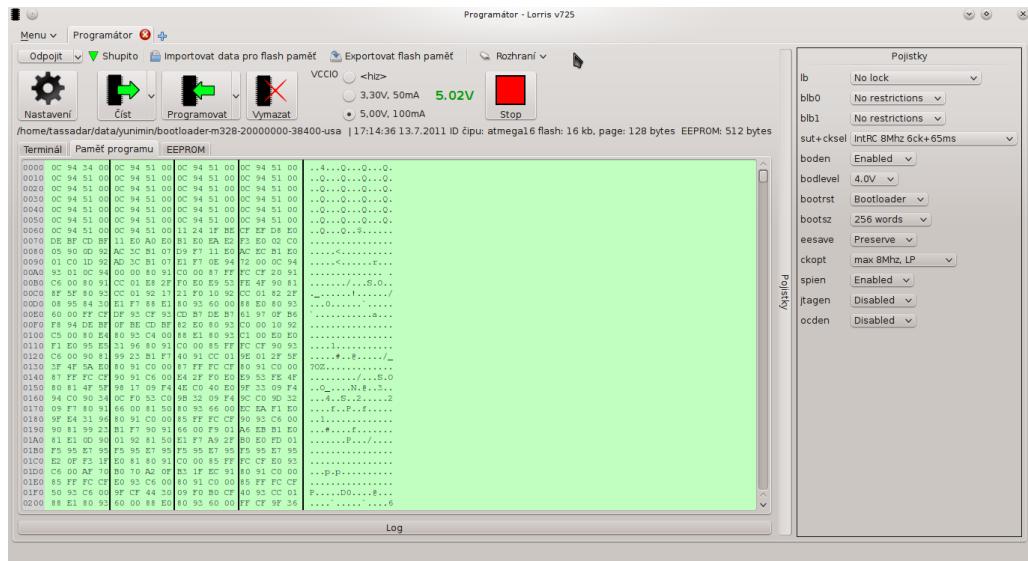
Obrázek 24: Proxy mezi sériovým portem a TCP socketem

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

### 4.1 Proxy tunel

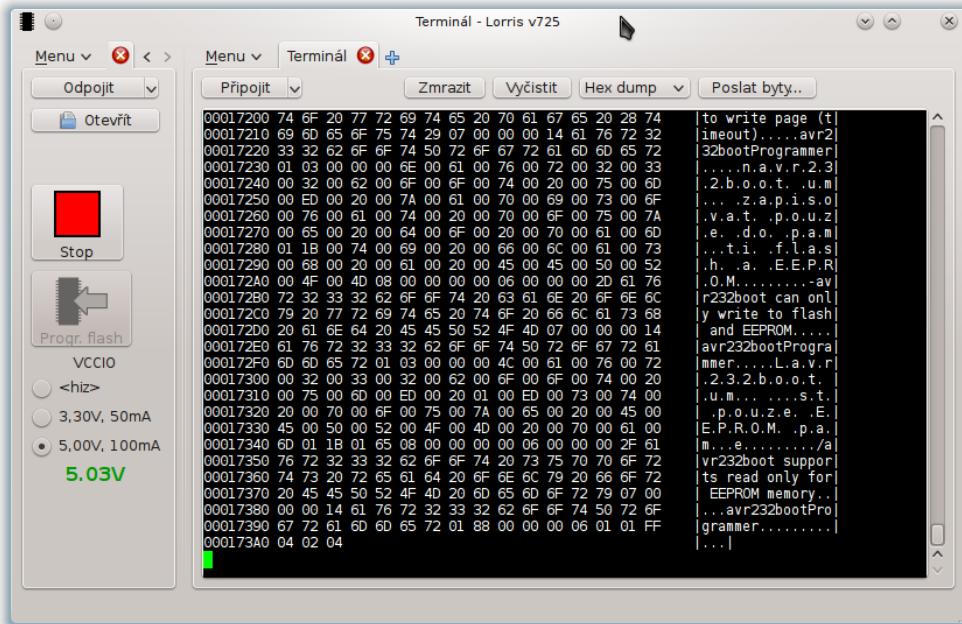
Tento modul také přidává nové virtuální připojení - „proxy tunel“. Pokud toto připojení použije nějaký z dalších modulů v Lorris, tak může posílat a přijímat data od všech TCP klientů připojených na proxy. Toto je možné využít například tak, že v modulu analyzátor je script, který generuje data a přes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

## 5 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 27). Plné rozhraní obsahuje tlačítka a nastavení pro programování všech pamětí čipu, zmenšené rozhraní pak obsahuje pouze tlačítko 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*

## 5.1 Programátor Shupito

Shupito je programátor mikročipů vytvořený Martinem Vejnárem, 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 programátorem je napsána samotným Martinem Vejnárem.

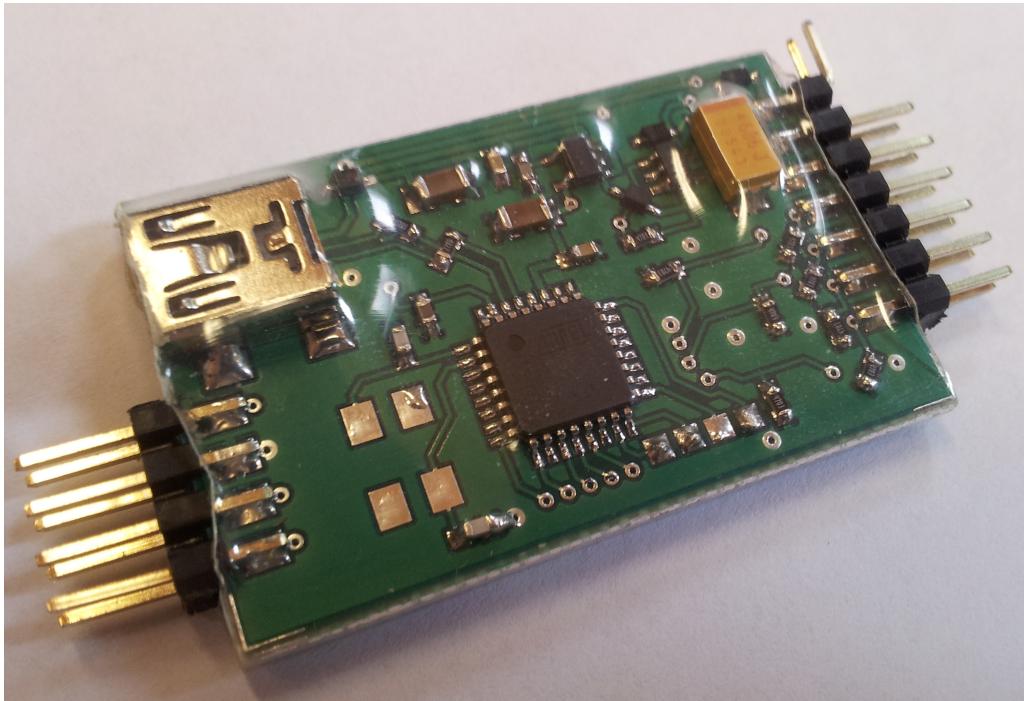
<sup>8</sup> *In-system programming* – rozhraní, které umožňuje programovat čipy přímo na desce plošného spoje.

<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

### 5.1.1 UART tunel

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



Obrázek 27: Programátor *Shupito*

## 5.2 Bootloader avr232boot

Autorem tohoto bootloaderu je také Martin Vejnár. Avr232boot je pouze pro čipy Atmel ATmega a je inspirováný referenčním bootloaderem 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

---

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

EEPROM<sup>12</sup>.

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

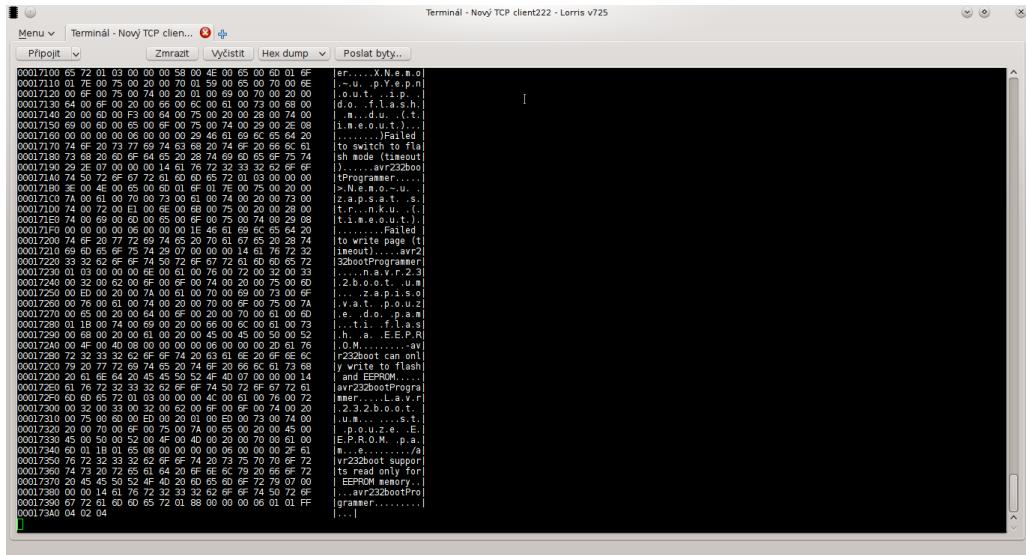
### 5.3 Bootloader AVROSP

*AVR Open Source Programmer* je protokol používaný poměrně velkým množstvím bootloaderů, například referenčními bootloadery firmy Atmel pro čipy ATmega a ATxmega a rozšířitelným a modulárním bootloaderem XBoot[10]. Lorris dokáže pomocí tohoto protokolu programovat a číst flash i EEPROM paměť čipu.

---

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

## 6 Modul: Terminál



Obrázek 28: Modul terminál

Základní pomůcka při práci s mikrokontroléry, běžný textový terminál – zobrazuje data přijatá přes sériový port a posílá stisky kláves. Kromě klasického textového módu dokáže příchozí data zobrazovat jako hexadecimální hodnoty všech příchozích 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).

## 7 Podpora joysticku

Lorris podporuje použití joysticku v modulu analyzér například k řízení robota. Nejdříve jsem k přístupu k joysticku používal knihovnu SDL[11], ta však pro moje použití nebyla příliš vhodná – tuto knihovnu autoři vytvořili jako základ pro tvorbu počítačových her a podpora joysticku je tedy 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 přeskladání kabelů u počítače nebo se joystick odpojí sám kvůli špatnému kontaktu), není třeba ho znova vybírat jako aktivní, sám se připojí bez jakékoliv interakce uživatele.

Libenjoy je vydána pod licencí GNU GPLv2.1[26].

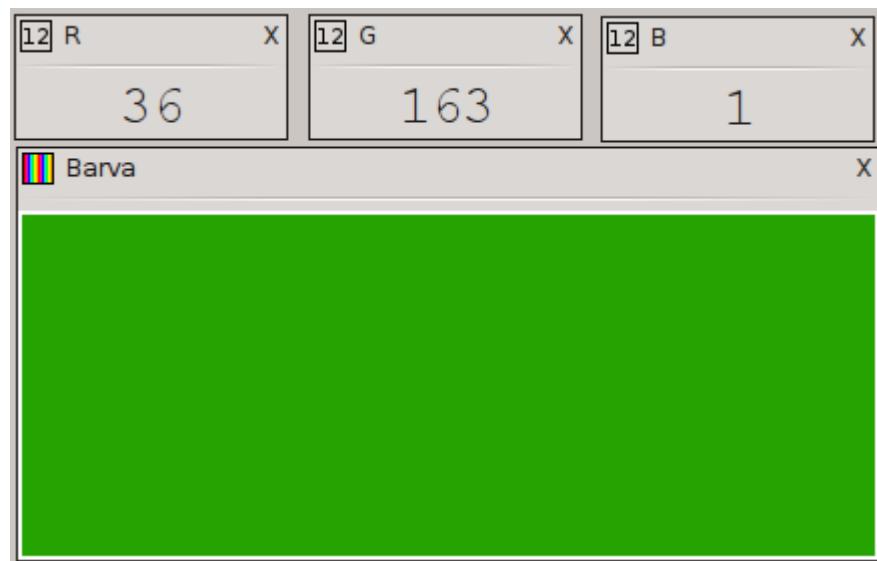
- GIT repozitář: <https://github.com/Tassadar/libenjoy>

## 8 Příklady použití

### 8.1 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 vycítit 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 29: Barva v modulu Analyzér

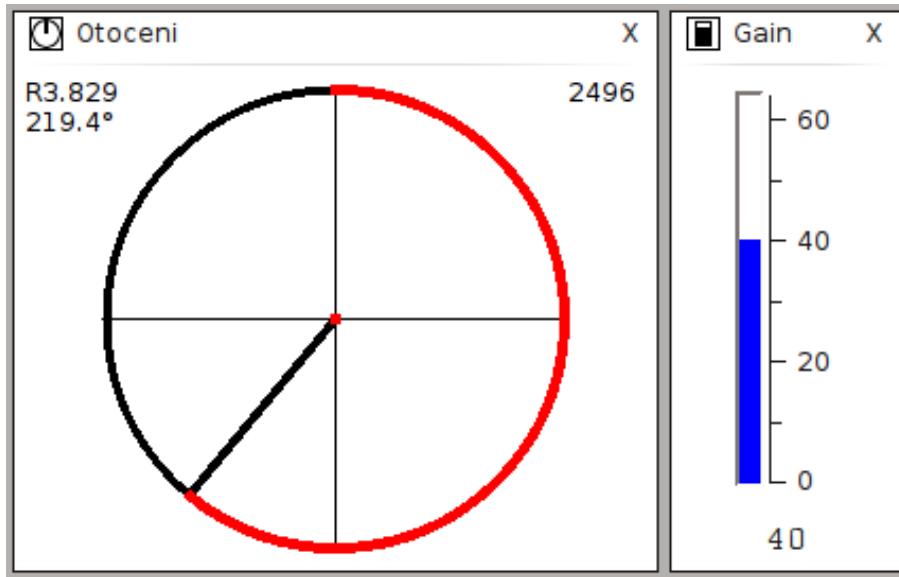
## 8.2 Testování enkodéru

V roce 2012 vypracoval můj spolužák Marek Ortcikr práci SOČ *Modulární stavba robota* ve které byl jedním z modulů magnetický vlečný enkodér. Tento modul vypadá jako další kolečko, které se na robota připevní, uvniř osy kola je však magnet a naproti němu napevno uchycený čip enkodéru. Čip snímá orientaci magnetického pole vytvořeného magnetem v ose kola a podle toho dokáže určit jeho natočení. Enkodér poté sleduje změny v natočení kola podle kterých je možné zjistit ujetou vzdálenost, rychlosť a zrychlení robota.



Obrázek 30: Magnetický enkodér

Pro demonstraci enkodéru na celostatním kole soutěže SOČ byla použita moje aplikace Lorris. Celé rozhraní je vidět na obrázku 47 na straně 82, zde jsou popsány jednotlivé části zvlášť.



Obrázek 31: Natočení kola

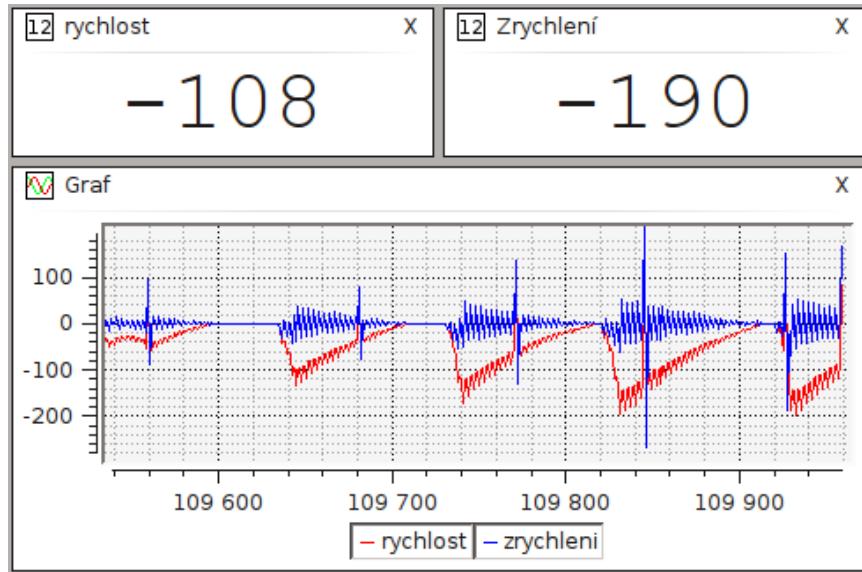
Widget *kolo* zde znázorňuje aktuální natočení kolečka enkodéru. V pravém horním rohu je aktuální hodnota z enkodéru (0 až 4095), vlevo je hodnota přepočítaná na radiány a stupně.

Widget *bar* pojmenovaný „gain“ ukazuje sílu magnetického pole, tedy jak daleko je magnet v ose kolečka od čipu enkodéru. Hodnota má rozmezí od 0 do 63, ideální je zhruba střed tohoto rozmezí.



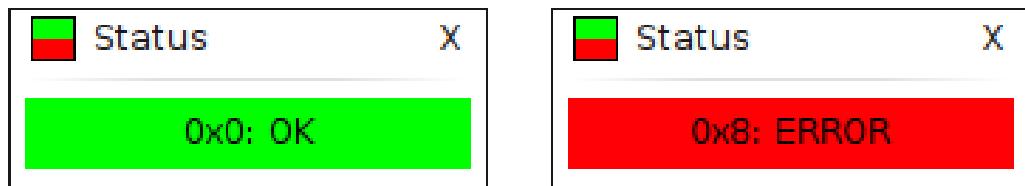
Obrázek 32: Ujetá vzdálenost

Tento widget *číslo* zobrazuje naměřenou vzdálenost v milimetrech. Enkodér odesílá tuto informaci v  $\frac{1}{4096}$  obvodu kolečka, takže je třeba ji přepočítat pomocí výrazu  $%n/32.5949$ .



Obrázek 33: Rychlosť a zrychlení

Dva widgety *číslo* jsou zde použity k zobrazení aktuální rychlosťi a zrychlení. Pod nimi je widget *graf* ve kterém je rychlosť jako červená křivka a zrychlení jako modrá.



Obrázek 34: Stav enkodéru

Čip enkodéru také informuje o svém stavu, pokud je vše v pořádku vrací číslo 0x0, narazí-li na nějaký problém, vrátí některý z chybových kódů (například 0x8 při nepřítomnosti magnetu v ose kola). Widget *status* ukazuje chybový kód a barvu podle informací z enkodéru.



Obrázek 35: Ovládání enkodéru

Tyto widgety *tlačítko* a *vstup* obstarávají ovládání enkodéru. Tlačítko „reset“ vrátí počítadlo ujeté vzdálenosti na nulu, tlačítko „clear“ pošle enkodéru příkaz k resetu chybového kódu (kód po chybě zůstává, i když přičina již byla napravena, je třeba ho resetovat) a tlačítko „Ovladani“ zastavuje a spouští posílání dat z enkodéru. Všechny tyto widgety jsou připojené na script (do obrázku 47 se už widget script nevešel), který po kliknutí na tlačíka odešle do enkodéru příslušné příkazy.

```

1 var run = true;
2 function reset_clicked() {
3     sendData(new Array(0xFF, 0x01, 0x01, 0x00));
4 }
5 function clear_clicked() {
6     sendData(new Array(0xFF, 0x01, 0x01, 0x01));
7 }
8 function startStop_clicked() {
9     run = !run;
10    startStopBtn.text = run ? "Stop" : "Start";
11    sendData(new Array(0xFF, 0x01, 0x02, 0x02, run ? 1 : 0));
12 }
```

Příklad 5: Script, který odesílá příkazy do enkodéru

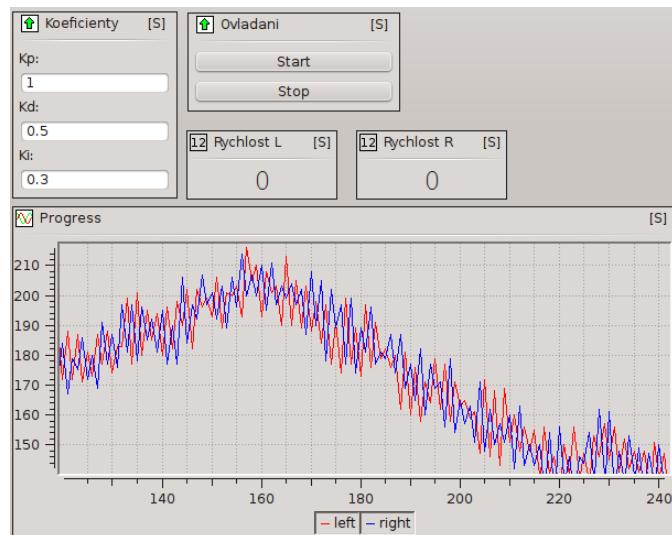
### 8.3 Ladění PID regulátoru

**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 analyzér 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[12] během přípravy na soutěž *Line Follower Standard*, která je součástí Robotického dne 2012[13]. Na soutěži jsem skončil na 2. místě z 22 robotů[14].



Obrázek 36: Ladění PID regulátoru

## 8.4 Stavba robota pro soutěž Eurobot 2011

Použití mého programu Lorris je zde prezentováno na příkladu stavby robota, který vznikal na naší škole (SPŠ a VOŠ technická, Sokolská 1, Brno) v roce 2011 pro soutěž Eurobot[15] (fotografi robotu najdete v přílohách na straně 84, obrázek č. 49).

Ačkoliv je robot již 2 roky starý a celá sada Lorris se od té doby značně posunula vpřed, robot *David* představuje velmi dobrý příklad využití velké části funkcí celého balíku nástrojů.

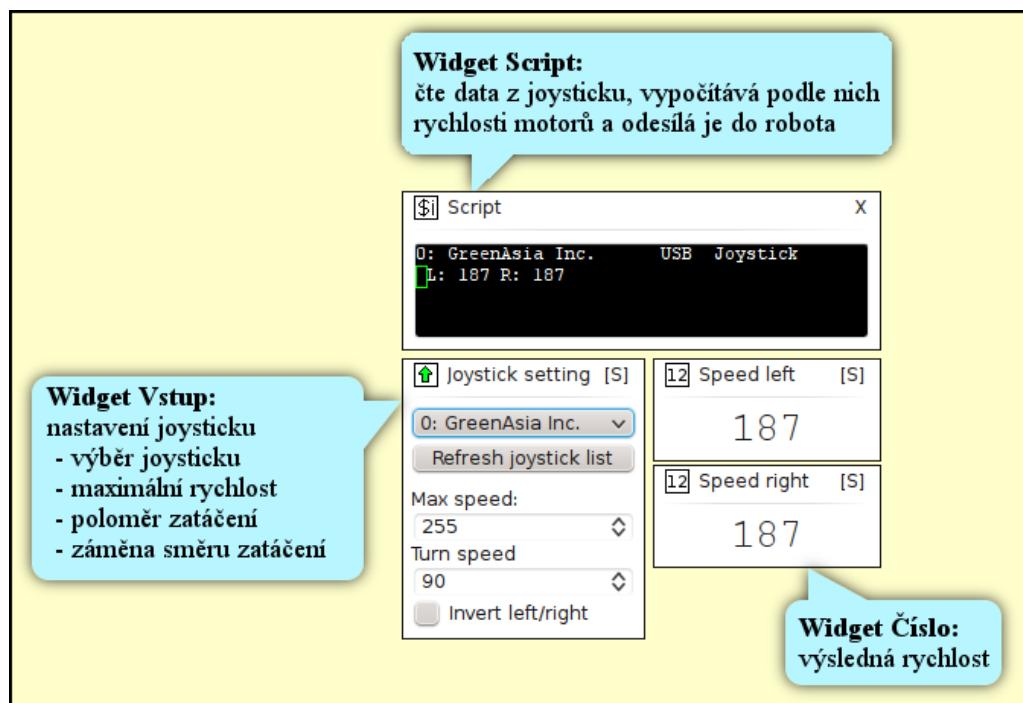
Cíl soutěže je každý rok jiný, v roce 2011 měli roboti za úkol hrát něco jako zjednodušené šachy. Herní hřiště bylo rozděleno na barevnou šachovnici a leželi na něm „pěšci“ (žluté disky), které měli roboti posouvat na políčka svojí barvy, případně z nich stavět věže. Vyhrával robot s největším počtem 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 dálkoměrů). Kompletní pravidla, výsledková listina a další informace jsou na webu ročníku 2011[16].

Právě při vývoji tohoto robota vyvstala palčivá potřeba mít k dispozici nástroj, který by umožňoval ve všech fázích jeho vývoje snadné a rychlé testování a ladění všech funkcí a komponent robota. Vzhledem k tomu, že nejviditelnější částí programu Lorris je nástroj Analyzér, je v této ukázce prezentováno především jeho použití, ostatní nástroje (Programátor, Terminál) však byly také použity, například při programování mikročipu v robotovi.

V příkladu je vytvořeno jednoduché uživatelské prostředí pro ovládání, testování a programování pro jednoho robota. Toto prostředí však lze znova použít i pro jiného robota anebo vytvořit nové, pokud je robot příliš atypický a vyžaduje jiný typ ovládání.

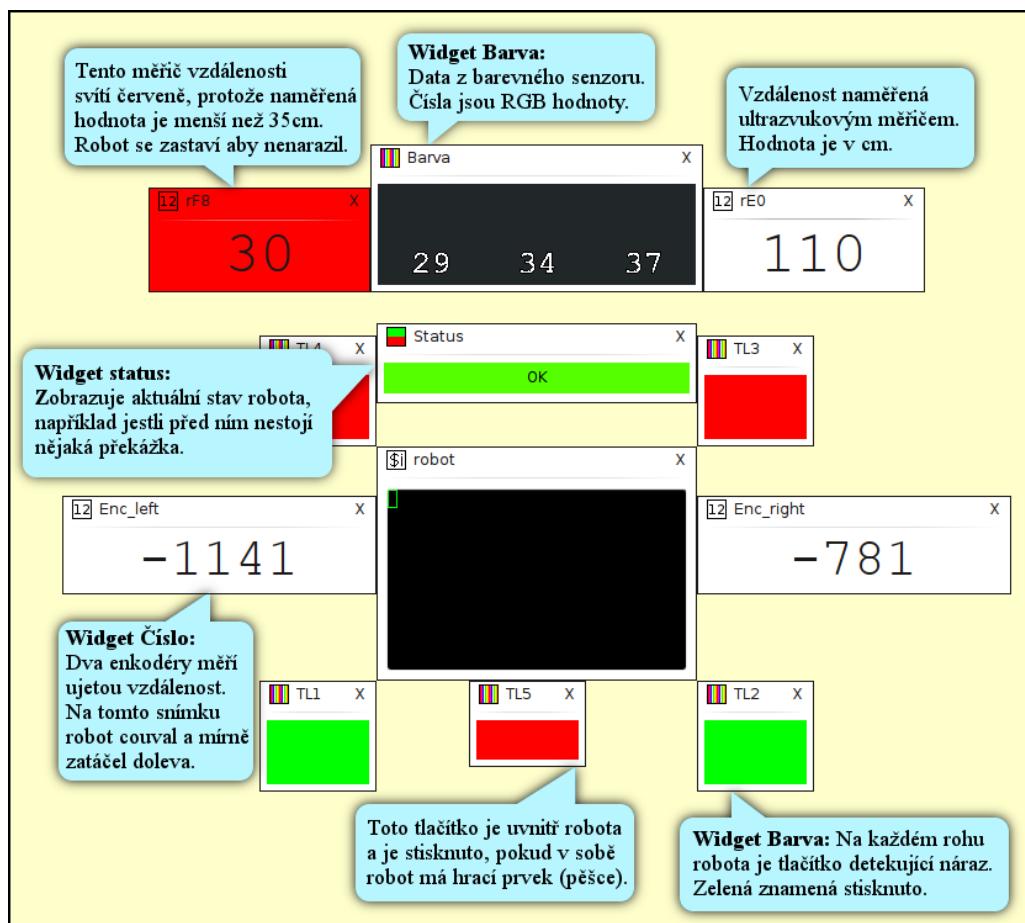
#### 8.4.1 Mechanická kostra robota

Jako první byla navržena mechanická konstrukce robota. Již v této fázi byla využita moje aplikace Lorris. Pro otestování funkčnosti a chování motorů a servomotorů bylo použito ovládání pomocí joysticku. V Lorris jsem sestavil menší skupinu widgetů: *Script*, který čte data z joysticku, přepočítává je na rychlosti, které je třeba nastavit motorům a odesílá je do robota. Dále widget *Vstup*, ve kterém je nastavení ovládání pomocí joysticku a 2 widgety *Číslo*, které zobrazují aktuální rychlosti motorů.



### 8.4.2 Ladění a nastavení senzorů

Po vyladění mechanické části robota byl osazen senzory. Po jejich umístění jsem v nástroji Analyzér vytvořil rozhraní, které využívá zejména widgetů *Script*, *Cílo*, *Barva* a *Status*. Každý z těchto widgetů je možné na pracovní ploše Analyzéru přesouvat, zmenšovat nebo zvětšovat, díky čemuž je možné jejich rozmístění tak, aby odpovídalo skutečným pozicím senzorů na robotu. Jako optimální se jeví zobrazení jako při pohledu shora.



### **8.4.3 Programování reaktivního chování robota**

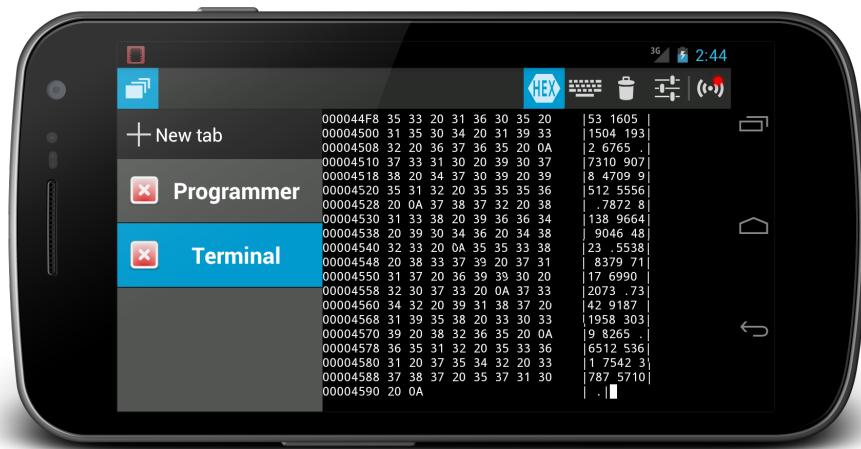
Vrcholem vývoje robota bylo programování jeho chování na herní ploše. Při této příležitosti se v plné míře uplatnil widget *Script* programu Lorris. V tomto widgetu bylo vytvořeno scriptovací prostředí, které zapouzdřilo nejtypičtější povelové sady, pomocí kterých lze s výhodou konstruovat složitější vzorce chování robota. Widget *Script* by umožnil i přímé psaní scriptu pro řízení robota, ale zmíněné prostředí tuto práci výrazně zjednodušilo.

Za povšimnutí stojí také to, že zde byl widget *script* využit nejen pro řízení robota, ale i pro vylepšení fungování samotného nástroje Analyzér.

V tomto příkladu používám jednoduché „akce“, které robot postupně provádí. Každá akce má 3 hlavní parametry - směr jízdy, kdy se má robot zastavit a co má vykonat, když se zastaví na cílovém místě. Všechny akce je možné ve scriptovacím prostředí rovnou měnit, bez nutnosti přeprogramovávat robota. Všechny ostatní části prostředí Lorris stále fungují, i když robot je právě řízen nastaveným scriptem. Díky tomu lze sledovat stav robota i všech jeho senzorů a rychle zjistit zdroj případného neočekávaného chování.

*Obrázek č. 48, který patří této části textu, najdete v přílohách na straně 83.*

## 9 Aplikace pro Android



Obrázek 37: Lorris mobile

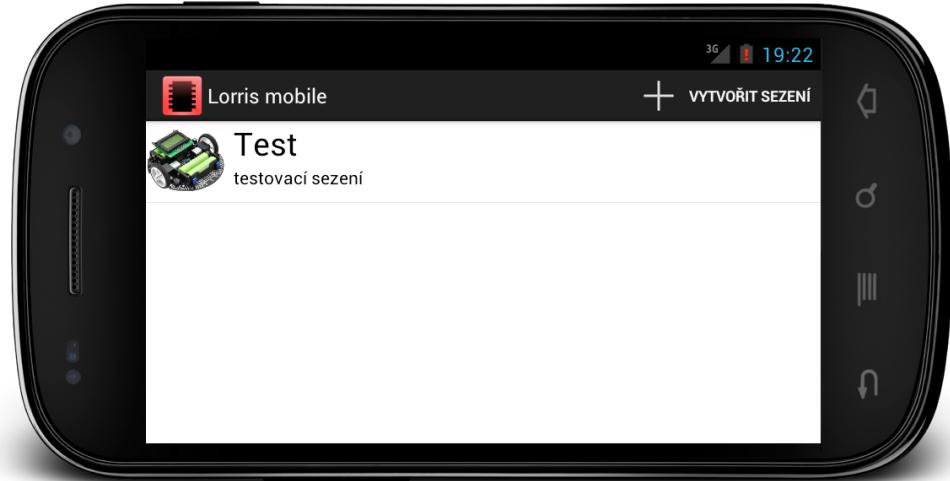
Jako další vývojový stupeň jsem začal s vývojem Lorris pro platformu Google Android[18], protože chytrá zařízení s tímto operačním systémem jsou často při ruce a stačí k rychlému vyřešení menšího problému.

Aplikace **Lorris mobile** slouží jako přenosný doplněk k počítačové verzi Lorris – nemusí nutně obsahovat všechny funkce desktopové aplikace ale má pomocí zejména když je v terénu potřeba rychle něco přenastavit či poupravit.

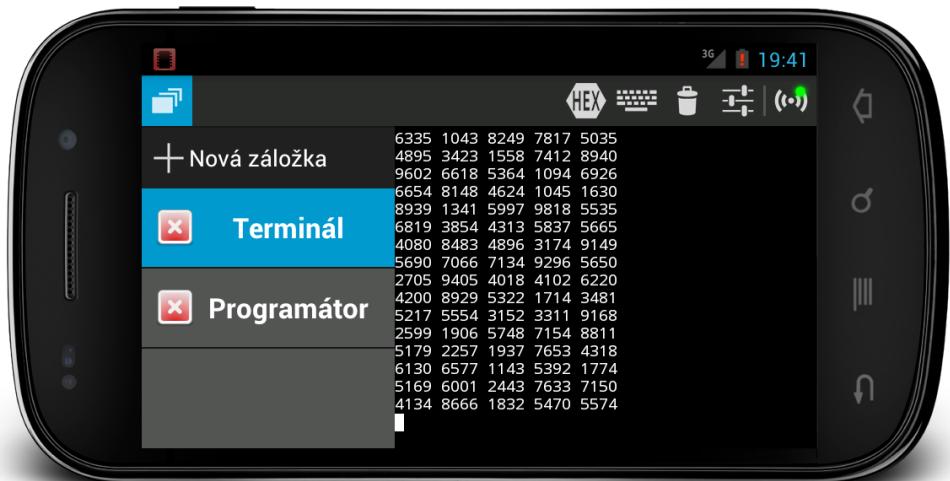
Aplikace funguje na telefonech a tablettech s OS Android ve verzi 2.2 a vyšší, je optimalizována i pro větší obrazovky tabletů a je dostupná v oficiálním distribučním kanále Android aplikací – v obchodě Google Play[19], stačí hledat heslo „Lorris“.

Lorris mobile má podobnou architekturu jako desktopová verze Lorris. Začíná se vytvořením sezení, do kterého se ukládá vše, co uživatel v aplikaci otevře (obrázek 38). 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 39).



Obrázek 38: Lorris mobile - výběr sezení



Obrázek 39: Lorris mobile - přepínání záložek

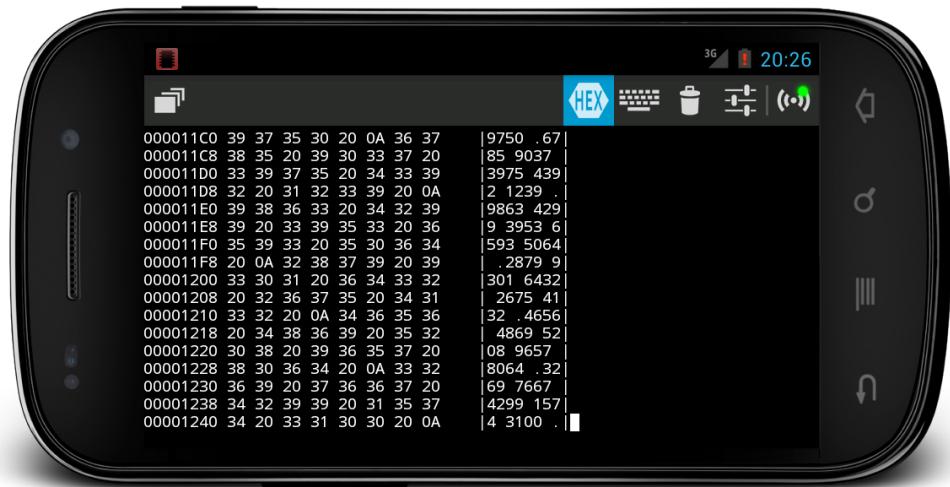
## 9.1 Programátor



Obrázek 40: Lorris mobile - programátor

Modul programátor dokáže programovat čipy pomocí bootloaderů **avr232boot** a **AVROSP** a také pomocí programátoru Shupito. Tato část Lorris mobile používá části nativního kódu z desktopové verze Lorris, díky tomu se kód lépe spravuje a je rychlejší.

## 9.2 Terminál



Obrázek 41: Lorris mobile - terminál

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

## 10 Reálné nasazení

Prvními uživateli sady Lorris se stali od samého začátku vývoje před několika lety členové některých technicky zaměřených kroužků na DDM Junior[21] v Brně. Lorris zde pomáhá při stavbě různých zařízení, zejména robotů, a žáci, kteří se učí programovat mikrokontroléry, používají programátor *Shupito* a tím pádem i modul Programátor z balíku Lorris. V průběhu výuky programování mikrokontrolérů pomáhá také modul Terminál (jednoduchá komunikace s čipem) a později i Analyzér (pokročilé zpracovávání dat z čipu).

Lorris je pro použití v DDM Junior vhodná také díky nulové pořizovací ceně – větší firma, zabývající se vývojem aplikací pro mikrokontroléry, by pravděpodobně pořídila drahý komerční program s podobnými funkcemi jako má Lorris nebo vyvinula vlastní jednoúčelové aplikace. Pro příspěvkové organizace typu DDM je však řešení používané velkými firmami poněkud nedostupné.

K dnešnímu dni má Lorris přibližně 20 uživatelů jen z řad DDM Junior. Díky postupnému rozšiřování programátoru *Shupito* mezi uživatele po celé ČR se ale okruh uživatelů Lorris utěšeně rozrůstá.

Já Lorris používám kdykoliv pracuji s roboty a mikrokontroléry – pro zobrazení dat, programování mikročipů či ovládání celých zařízení. Následující seznam představuje pouze několik významnějších aplikací, pro které již byla sada Lorris použita ostatními uživateli:

- Vývoj několika robotů pro soutěže letošního Robotického dne[17]
- Programování velkého počtu typů mikročipů, ať už pomocí programátoru *Shupito* nebo pomocí bootloaderů
- Vývoj samotného programátoru *Shupito*
- Já používám desktopovou Lorris ke generování testovacích dat pro mobilní verzi Lorris

- Vývoj levné logické sondy
- Ladění čipů pro ovládání třífazového motoru (tzv. *driver*)
- Vývoj systému pro řízení až 128 RGB diod pro osvětlení modelu letadla
- Stavba a programování digitální vysílačky používající ARM procesor (semestrální práce)
- Vývoj robota pro sledování čáry (maturitní práce)

## Závěr

Po několikaletém vývoji mohu konstatovat, že aplikace splňuje všechny požadavky požadavky stanovené v kapitole 1:

- ✓ 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 nejnižší cena – program je dostupný zdarma
- ✓ 6. Snadná rozšířitelnost (ideálně otevřený zdrojový kód)
- ✓ 7. Nezávislost na další aplikaci (např. MS Office Excel)

Program navíc výrazně přesáhl původní cíle – kromě 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.

Lorris již má za sebou řadu ostrých nasazení a také rozšiřující se okruh věrných uživatelů, jak je podrobněji popsáno v kapitole 10.

Aplikace je nadále vyvíjena, mohu prakticky donekonečna přidávat buďto další typy widgetů do modulu Analyzér (například kompas, směrový kříž, ...) nebo celé nové moduly (například rozhraní pro novou levnou logicou sondu, kterou vyvíjí Martin Vejnár). Program má v současné době (4.3.2013) asi 32 tisíc řádků kódu (bez knihoven třetích stran).

Přiložené CD obsahuje zdrojový kód, instruktážní video, propagační poster a anglickou verzi textu.

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

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

Obrázek 42: Počet řádků spočítaný programem CLOC[20]

V budoucnu bych rád pokračoval v přidávání dalších funkcí do počítacové i mobilní verze Lorris a případně také v rozšiřování povědomí o celém balíku.

## 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 onDataChanged(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 6: Základní script

**onDataChanged(data, dev, cmd, index)** je volána při změně pozice v datech (tj. když přijdou nová data nebo uživatel pohně 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 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 7: 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 8: 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 9: 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 obsáhlější 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. Pokud není tento paramter zadán, vytvoří se nový widget v levém horním rohu aktuálně viditelné plochy.

- **Yoffset** – **Integer**, svislá vzdálenost v pixelech od levého horního rohu mateřského ScriptWidgetu. Pokud není tento paramter zadán, vytvoří se nový widget 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 10: 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 11: 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 12: 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 13: 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 14: 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

```
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 15: 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

```

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 16: 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 tlačítka (QPushButton), zaškrťávací 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).

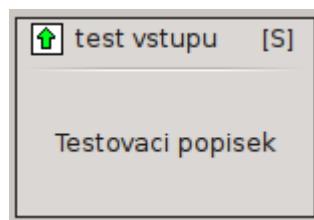
```

1 var vstop = newWidget(WIDGET_INPUT,
2                         "test vstupu", 150, 100, -160, 0);
3 var label = vstop.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 = "Testovaci popisek";

```

Příklad 17: Widget *vstop* – vytvoření QLabel

Widget vytvořený tímto příkladem vypadá takto:

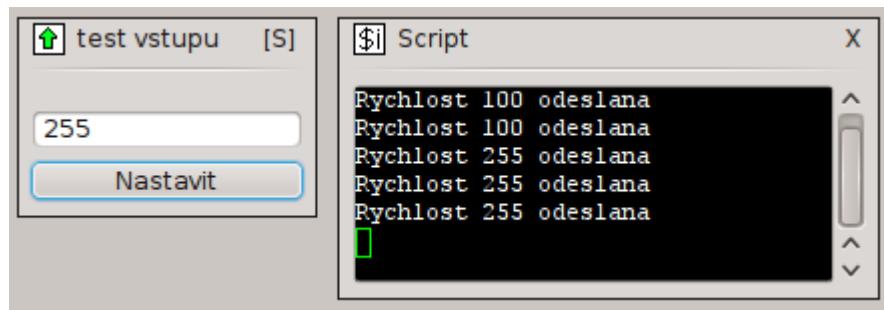


Obrázek 43: Widget *vstop* – vytvoření QLabel

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 18: Widget *vstup* – tlačítko



Obrázek 44: 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); // nastavení vstupu na stupne
4 c.setValue(270);
```

Příklad 19: 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.

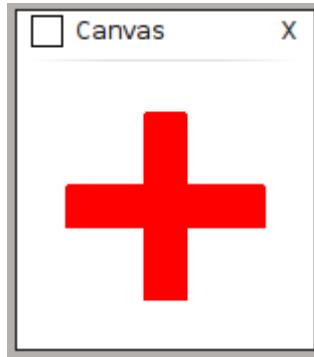
- `setLineColor(String barva)` – Barva čáry, kterou se prvky kreslí.
- `setFillColor(String barva)` – Barva výplně obdélníků, elips a kruhů

```

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 20: Nakreslení kříže ve widgetu *plátno*



Obrázek 45: Nakreslení kříže ve widgetu *plátno*

## Widget status

- `addStatus(int id, bool bitMaska, String text  
String barvaPozadí, String barvaTextu)`

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()` – Vrátí 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 21: Ovládání widgetu *status* ze scriptu



Obrázek 46: 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 22: Nastavení widgetu *tlačítka* 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 23: 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()` – Vrátí minimální, maximální a aktuální hodnotu

```

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

Příklad 24: 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 data
- `exists(String klíč)` – Vrátí `true` pokud hodnota s tímto klíčem existuje.
- `setXXXX(String klíč, XXXX hodnota)`  
`getXXXX(String klíč, XXXX pokudKlíčNeexistuje)`  
`setYYYYArray(String klíč, PoleYYYY hodnota)`  
`getYYYYArray(String klíč, PoleYYYY 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 `getXXXX` 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); // desetinna cisla
3  s.setMax(6.28);
4
5  // Nacte hodnotu, ktera byla pred tim ulozena v metode save()
6  var ulozene = storage.getFloat("hodnotaPosuvniku", 3.14);
7  s.setValue(ulozene);
8
9  // Nacte pokusne pole cisel ulozene v metode save()
10 var pokus = storage.getInt32Array("pokusnePole", new Array());
11 appendTerm("Ulozene 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 25: Ukládání dat scriptu

## Přístup k joysticku

Nejříve několik globálních metod pro práci s joysticky:

- `getJoystickNames()` – Vráti pole Stringů se jmény připojených 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 objekt `Joystick` neno 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
16function axesChanged(axes) {
17    for(var i = 0; i < axes.length; ++i) {
18        var hodnota = joy.getAxisVal(axes[i]);
19        appendTerm("Os " + axes[i] + ": " + hodnota + "\n");
20    }
21}
22
23function buttonChanged(id, state) {
24    appendTerm("Tlacitko " + id + ", stav: " + state + "\n");
25}

```

Příklad 26: Otevření joysticku a čtení jeho hodnot

## PŘÍLOHA B:

### Knihovny třetích stran

- **Qwt**[22] je knihovna pro Qt Framework obsahující tzv. widgety pro aplikace technického charakteru – grafy, sloucové ukazatele, kompasy a podobně.
- **QExtSerialPort**[23] poskytuje připojení k sériovému portu a také dokáže vypsat seznam nalezených portů v počítači.
- **QHexEdit2**[24] je hex editor použitý v modulu programátor na zobrazení obsahu paměti. V této knihovně jsem upravoval několik málo drobností, týkajících se především vzhledu.
- **Tanto Icon Library**[29] je sada ikon vydaných jako volné dílo. Lorris používá ikonky z této sady na mnoha místech napříč celým programem.
- **EcWin7**[30] je knihovna, která poskytuje API k progressbaru v hlavním panelu Windows 7.
- **QScintilla2**[31] je pokročilý editor textu pro Qt.
- **PythonQt**[33] jsou Python bindings pro Qt.
- **Python**[34] je programovací jazyk, používají se některé části jeho interpreteru v kombinaci s PythonQt.
- **Qt Solutions**[36] je soubor několika přídavných modulů pro Qt
- **libyb**[37] je knihovna která obstarává komunikaci s programátorem Shupito

## PŘÍLOHA C:

### **Licence**

Lorris je dostupný pod licencí GNU GPLv3[25], licence použitých programů a knihoven jsou následující:

- **Qt Framework** je distribuován pod licencí GNU LGPLv2.1[26]
- **Qwt** je distribuováno pod Qwt license[27], která je založená na GNU LGPLv2.1
- **QExtSerialPort** je distribuován pod The New BSD License[28]
- **QHexEdit2** je distribuován pod licencí GNU LGPLv2.1
- **Tanto Icon Library**[29] je vydána jako volné dílo (*Public domain*)
- **EcWin7** je distribuováno pod GNU GPLv2
- **QScintilla2** je distribuována pod GNU GPL v2 a v3
- **libenjoy**[32] je distribuována pod GNU GPLv2.1
- **PythontQt** je distribuována pod GNU LGPLv2.1
- **Python** je distribuován pod PSF License agreement[35]
- **Qt Solutions** je distribuován pod The New BSD License
- **libyb** je distribuován pod Boost Software License[38]

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 25.2.2013)
- [2] *WinWedge* – RS232 data collection software  
<http://www.taltech.com/products/winwedge/>  
(Stav ke dni 25.2.2013)
- [3] *Advanced Serial Data Logger*  
<http://www.aggsoft.com/serial-data-logger.htm>  
(Stav ke dni 25.2.2013)
- [4] *StampPlot Pro* – Graphical Data Acquisition and Control  
<http://www.selmaware.com/stampplot/index.htm>  
(Stav ke dni 25.2.2013)
- [5] *LabVIEW* – Laboratory Virtual Instrumentation Engineering Workbench  
<http://sine.ni.com/np/app/main/p/docid/nav-104/lang/cs/>  
(Stav ke dni 6.3.2013)
- [6] *Qt* – Cross–platform application and UI framework  
<http://qt-project.org/>  
(Stav ke dni 25.2.2013)
- [7] *Debian Linux* – The Universal Operating System  
<http://www.debian.org/>  
(Stav ke dni 25.2.2013)

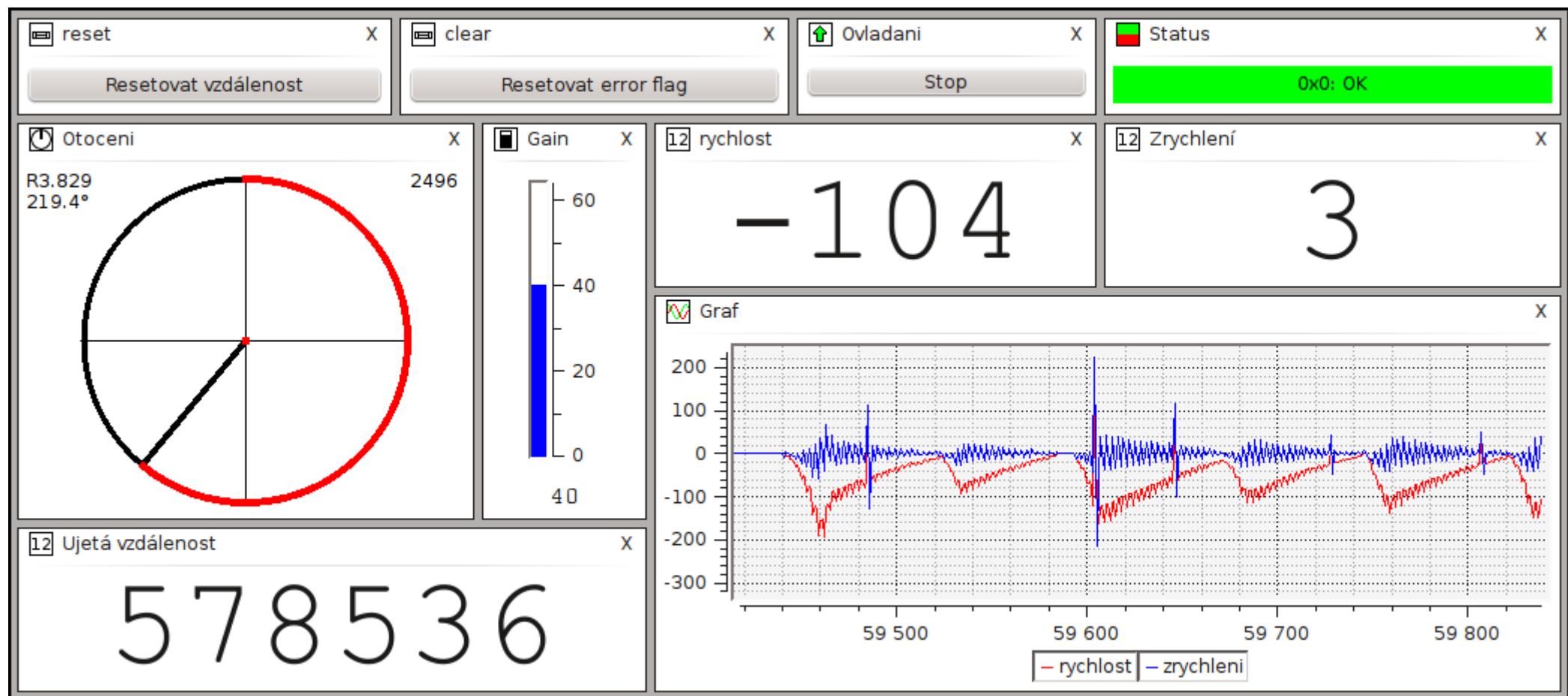
- [8] *GitHub* – Social Coding  
<https://github.com>  
(Stav ke dni 25.2.2013)
- [9] *Making Applications Scriptable*  
<http://qt-project.org/doc/qt-4.8/scripting.html>  
(Stav ke dni 25.2.2013)
- [10] *XBoot* – Extensible bootloader for ATMEL XMEGA microcontrollers  
<http://code.google.com/p/avr-xboot/>  
(Stav ke dni 6.3.2013)
- [11] *SDL* – Simple Directmedia Layer  
<http://www.libsdl.org/>  
(Stav ke dni 13.2.2013)
- [12] *Pololu 3pi Robot*  
<http://www.pololu.com/catalog/product/975>  
(Stav ke dni 25.2.2013)
- [13] *Robotický den 2012*  
<http://www.robotickyden.cz/2012/>  
(Stav ke dni 25.2.2013)
- [14] *Robotický den 2012* – výsledky soutěže Line Follower standard  
<http://www.robotickyden.cz/2012/results/lfs.php>  
(Stav ke dni 28.2.2013)
- [15] *Eurobot*  
<http://www.eurobot.org/>  
(Stav ke dni 25.2.2013)
- [16] *Eurobot 2011*  
<http://www.eurobot.cz/eurobot2011.php>  
(Stav ke dni 25.2.2013)

- [17] *Robotický den*  
<http://www.robotickyden.cz/>  
(Stav ke dni 4.3.2013)
- [18] *Google Android* – Operační systém pro chytré telefony  
<http://www.android.com/>  
(Stav ke dni 25.2.2013)
- [19] *Google Play Store* – Obchod s aplikacemi pro OS Android  
<http://play.google.com/store>  
(Stav ke dni 14.2.2013)
- [20] *CLOC* – Count Lines of Code  
<http://cloc.sourceforge.net/>  
(Stav ke dni 25.2.2013)
- [21] *DDM Junior, Dornych 2, Brno, 656 20*  
<http://www.junior.cz>  
(Stav ke dni 4.3.2013)
- [22] *Qwt* – Qt Widgets for Technical Applications  
<http://qwt.sourceforge.net/>  
(Stav ke dni 25.2.2013)
- [23] *QExtSerialPort* – Qt interface class for old fashioned serial ports  
<http://code.google.com/p/qextserialport/>  
(Stav ke dni 25.2.2013)
- [24] *QHexEdit2* – Binary Editor for Qt  
<http://code.google.com/p/qhexedit2/>  
(Stav ke dni 25.2.2013)
- [25] *GNU General Public License v3*  
<http://gplv3.fsf.org/>  
(Stav ke dni 25.2.2013)

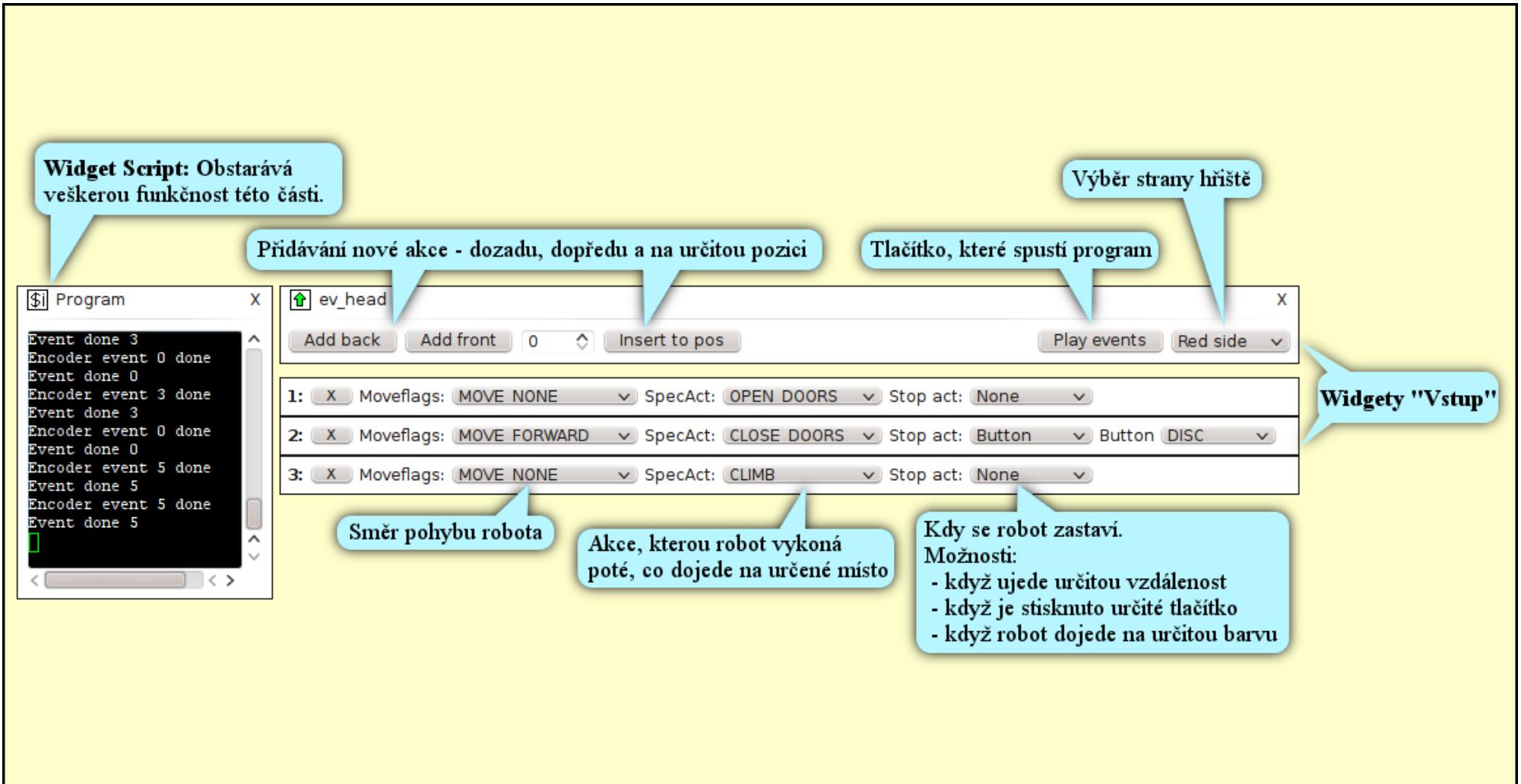
- [26] *GNU Lesser General Public License v2.1*  
<http://www.gnu.org/licenses/lgpl-2.1.html>  
(Stav ke dni 25.2.2013)
- [27] *Qwt license*  
<http://qwt.sourceforge.net/qwtlicense.html>  
(Stav ke dni 25.2.2013)
- [28] *The New BSD License*  
<http://www.opensource.org/licenses/bsd-license.php>  
(Stav ke dni 25.2.2013)
- [29] *Tango Icon Library*  
<http://tango.freedesktop.org/Tango\Icon\Library>  
(Stav ke dni 25.2.2013)
- [30] *EcWin7* – Windows 7 taskbar progress indicator  
<http://www.msec.it/blog/?p=118>  
(Stav ke dni 25.2.2013)
- [31] *QScintilla2* – Code editor  
<http://www.riverbankcomputing.co.uk/software/qscintilla/intro>  
(Stav ke dni 25.2.2013)
- [32] *libenjoy* – Small simple joystick library  
<https://github.com/Tassadar/libenjoy>  
(Stav ke dni 25.2.2013)
- [33] *PythonQt* – Python bindings for Qt  
<http://pythonqt.sourceforge.net/>  
(Stav ke dni 25.2.2013)
- [34] *Python*, the programming language  
<http://www.python.org/>  
(Stav ke dni 25.2.2013)

- [35] *PSF License agreement*  
<http://docs.python.org/2/license.html>  
(Stav ke dni 25.2.2013)
- [36] *Qt Solutions*, a collection of minor Qt add-ons  
<http://qt.gitorious.org/qt-solutions>  
(Stav ke dni 25.2.2013)
- [37] *libyb*, a collection of minor Qt add-ons  
<https://github.com/avakar/libyb>  
(Stav ke dni 25.2.2013)
- [38] *The Boost Software License*  
<http://www.boost.org/users/license.html>  
(Stav ke dni 25.2.2013)

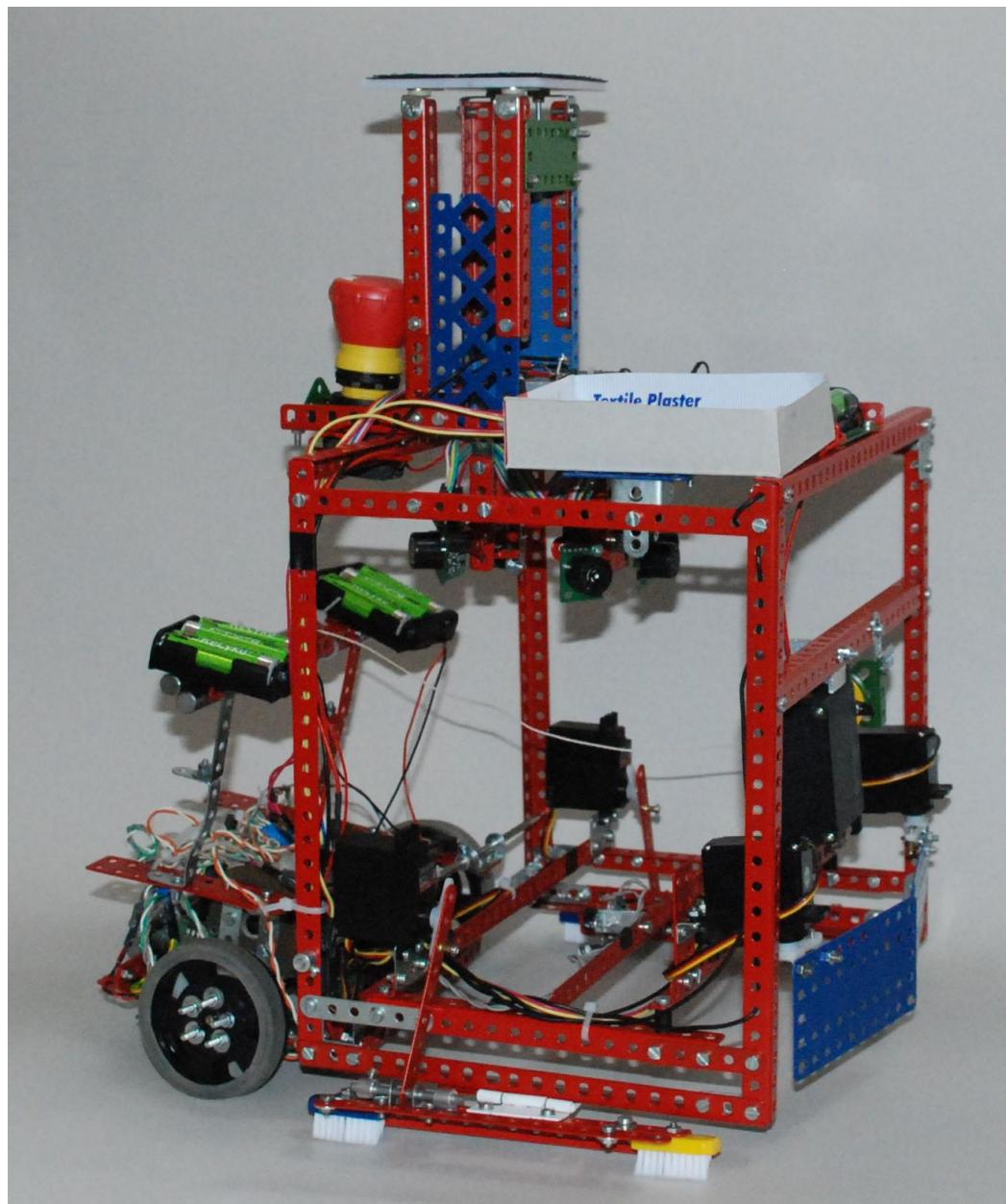
## PŘÍLOHA E: Velké obrázky



Obrázek 47: Data z enkodéru zpracovaná analyzárem



Obrázek 48: Programování chování robota v modulu *Analyzér*



Obrázek 49: Náš robot *David* skončil na 4. místě z 11 v celostátním kole soutěže Eurobot 2011

## PŘÍLOHA F:

### Seznam obrázků

1	Dialog vytvoření panelu . . . . .	10
2	Ukázka rozdělení okna na více částí . . . . .	10
3	Upozornění o dostupné aktualizaci . . . . .	11
4	Probíhající aktualizace . . . . .	11
5	Modul analyzér . . . . .	12
6	Zarovnávání widgetů pomocí sítě a čar . . . . .	14
7	Dialog nastavení struktury dat . . . . .	14
8	Widgety . . . . .	15
9	Nastavení filtrů . . . . .	16
10	Widget: číslo . . . . .	17
11	Widget: sloupcový bar . . . . .	18
12	Widget: barva . . . . .	19
13	Widget: graf . . . . .	20
14	Dialog pro nastavení parametrů křivky grafu . . . . .	20
15	Widget: script . . . . .	21
16	Dialog pro nastavení zdrojového scriptu . . . . .	22
17	Widget: kolo . . . . .	23
18	Widget: plátno . . . . .	23
19	Widgety tlačítko a slider . . . . .	24
20	Widget <i>vstup</i> s nastavením joysticku . . . . .	25
21	Widget status . . . . .	26
22	Nastavení stavů . . . . .	27
23	Widget terminál . . . . .	27
24	Proxy mezi sériovým portem a TCP socketem . . . . .	28
25	Modul Programátor . . . . .	29

26	Zmenšené UI modulu <i>programátor</i> (nalevo) s otevřeným <i>terminálem</i> . . . . .	30
27	Programátor <i>Shupito</i> . . . . .	31
28	Modul terminál . . . . .	33
29	Barva v modulu Analyzér . . . . .	35
30	Magnetický enkodér . . . . .	36
31	Natočení kola . . . . .	37
32	Ujetá vzdálenost . . . . .	37
33	Rychlosť a zrychlení . . . . .	38
34	Stav enkodéru . . . . .	38
35	Ovládání enkodéru . . . . .	39
36	Ladění PID regulátoru . . . . .	40
37	Lorris mobile . . . . .	45
38	Lorris mobile - výběr sezení . . . . .	46
39	Lorris mobile - přepínání záložek . . . . .	46
40	Lorris mobile - programátor . . . . .	47
41	Lorris mobile - terminál . . . . .	48
42	Počet řádků spočítaný programem CLOC[20] . . . . .	52
43	Widget <i>vstup</i> – vytvoření QLabel . . . . .	63
44	Widget <i>vstup</i> – tlačítko . . . . .	64
45	Nakreslení kříže ve widgetu <i>plátno</i> . . . . .	66
46	Ovládání widgetu <i>status</i> ze scriptu . . . . .	67
47	Data z enkodéru zpracovaná analyzárem . . . . .	82
48	Programování chování robota v modulu <i>Analyzér</i> . . . . .	83
49	Náš robot <i>David</i> skončil na 4. místě z 11 v celostátním kole soutěže Eurobot 2011 . . . . .	84