UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

RIADENIE RÁDIOVO OVLÁDANÉHO MODELU POČÍTAČOM A JEHO INTERAKCIA S VONKAJŠÍM SVETOM NA PLATFORME RASPBERRY PI

Diplomová práca

89f960c2-54b7-4f8a-8716-341f07949814

2015 **Bc. Peter Otto**

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

RIADENIE RÁDIOVO OVLÁDANÉHO MODELU POČÍTAČOM A JEHO INTERAKCIA S VONKAJŠÍM SVETOM NA PLATFORME RASPBERRY PI

Diplomová práca

89f960c2-54b7-4f8a-8716-341f07949814

Študijný program: Aplikovaná informatika
Študijný odbor: 9.2.9 Aplikovaná informatika
Pracovisko: Katedra informatiky FPV UMB Banská Bystrica
Vedúci diplomovej práce: Mgr. Peter Trhan, PhD.

Banská Bystrica, 2015 **Bc. Peter Otto**

Univerzita Mateja Bela v Banskej Bystrici
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Peter Otto
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	9.2.9. aplikovaná informatika
Typ záverečnej práce:	Magisterská záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Riadenie rádiovo ovládaného modelu počítačom a jeho interakcia s vonkajším svetom na platforme Raspberry PI

Anotácia: Zaoberajte sa prenosom videa cez WiFi sieť s nízkou latenciou. Vyriešte ovládanie RC auta prostredníctvo počítača, porovnajte spôsob odosielania riadiacich príkazov technológiami AJAX a Web Sockets. Získané poznatky využite pri riadení RC auta podľa rozpoznaného obrazu knižnicou OpenCV. Prácu realizujte na mini počítači Raspberry Pi s Raspberry Pi kamerou.

Vedúci:	Mgr. Peter Trhan, PhD.
Katedra:	KIN FPV - Katedra informatiky
Vedúci katedry:	PaedDr. Mgr. Vladimír Siládi, PhD.
Dátum zadania:	11.04.2014
Dátum schválenia:	19.03.2015

prof. Ing. Miroslav Svítek, Dr.
garant študijného programu

ABSTRAKT

OTTO, Peter;
RIADENIE RÁDIOVO OVLÁDANÉHO MODELU POČÍTAČOM A JEHO INTERAKCIA S VONKAJŠÍM SVETOM NA PLATFORME RASPBERRY PI :[Diplomová práca];
Univerzita Mateja Bela v Banskej Bystrici, Fakulta prírodných vied,
Katedra Informatiky; Vedúci diplomovej práce: Mgr. Peter Trhan, PhD.,
Banská Bystrica 2015, 82 s.

Práca sa zaoberá riadením rádiovo ovládaného modelu prostredníctvom počítača Raspberry Pi. Skúma spôsob prenosu videa v reálnom čase cez WiFi sieť. Porovnáva metódy odosielania riadiacich príkazov. Získané poznatky sú uplatnené pri implementácii aplikácie autonómneho riadenia. Výsledkom práce je mobilný, počítačom aj človekom ovládateľný model auta, schopný interakcie s vonkajším svetom, riadenia cez internet a programovania trasy jazdy. Nadobudnuté znalosti môžu byť zúžitkované v pokročilejších robotických projektoch, ktoré sa venujú algoritmom počítačového videnia, riadeniu motorov a prenosu obrazu.

Cieľom práce je vyriešiť riadenie rádiovo ovládaného auta počítačom Raspberry Pi a prenos videa cez WiFi sieť s nízkou latenciou pomocou Raspberry Pi kamery. Ďalším cieľom je porovnať spôsoby odosielania riadiacich príkazov technológiami AJAX a Web Sockets. Cieľom je využiť získané poznatky pri riadení RC auta podľa rozpoznaného obrazu knižnicou OpenCV.

Kľúčové slová: Raspberry Pi, OpenCV, ServoBlaster, AJAX, Web Sockets.

ABSTRACT

OTTO, Peter;
DRIVING RADIO CONTROLLED MODEL THROUGH A COMPUTER AND ITS INTERACTION WITH THE OUTSIDE WORLD ON THE RASPBERRY PI PLATFORM :[Diploma thesis];
Matej Bel University in Banská Bystrica, Faculty of natural sciences,
Department of informatics; Supervisor: Mgr. Peter Trhan, PhD.,
Banská Bystrica 2015, 82 p.

The thesis focuses on the driving of radio controlled model through the Raspberry Pi computer. The paper also explores the method of real-time video streaming over WiFi network. This paper further compares methods of sending controlling commands. Gained knowledge is applied in the implementation of application for autonomous control of a model. The result of this thesis is a vehicle that can be controlled by computer or by human. This vehicle is capable of interaction with the outside world through the Internet. Gained knowledge can be used in more advanced robotic projects that deal with algorithms of computer vision, motor control and video transfer.

The main objective of the thesis is to solve a radio controlled car driving through the Raspberry Pi computer and transferring video through WiFi network with low latency using the Raspberry Pi camera. Another objective is to compare sending methods for controlling commands that are executed through AJAX and Web Sockets technologies. The objective is to apply gained knowledge in driving a model of the car using a picture recognized by OpenCV library.

Keywords: Raspberry Pi, OpenCV, ServoBlaster, AJAX, Web Sockets.

Predhovor

Uvedenie miniatúrneho počítača Raspberry Pi na trh zaznamenalo vo svete veľký úspech. Účelom jeho vzniku bola podpora výučby programovania na školách. Domácim používateľom ponúkol nízkorozpočtovú alternatívu k multimediálnym centrám. Progresívnejšie orientovaným subjektom poskytol priestor na vznik inovatívnych projektov v oblasti robotiky a automatizácie.

Rádiovo ovládané modely sú spravidla riadené pomocou pákových ovládačov. Ovládač má pod kontrolou človek, teda ide o manuálne riadenie. Riadenie vyžaduje priamu viditeľnosť modelu. Modely vyšších cenových kategórií sú často vybavené kamerami na prenos obrazu z pohľadu prvej osoby. Ovládače bývajú nahrádzané mobilnými telefónmi, tabletmi, či PC, ktoré poskytujú dodatočnú funkcionalitu.

Cieľom tejto práce je prestavba rádiovo ovládaného modelu auta na model riadený počítačom Raspberry Pi. Práca analyzuje metódy prenosu obrazu s Raspberry Pi kamerou. Porovnáva spôsoby zasielania riadiacich príkazov. Riadenie počítačom a prenos videa umožňujú aplikovať algoritmy počítačového videnia, vďaka ktorým model nadobudne znaky autonómneho správania. Prínosom práce je ukázať čitateľovi možnosti tejto platformy, ktoré môže využiť pri tvorbe vlastných projektov.

Algoritmy počítačového videnia uvedené v práci sú modifikáciou a doplnením existujúcich algoritmov, pochádzajúcich z internetových zdrojov, prevažne diskusných fór a ukážok zdrojových kódov knižnice OpenCV. Hlavným zdrojom informácií pri tvorbe práce bolo Raspberry Pi fórum. Práca je realizovaná na rádiovo ovládanom modeli auta s Raspberry Pi kamerou a počítačom Raspberry Pi s príslušenstvom. Z softvéru boli použité knižnice OpenCV a PeerJS, operačný systém Raspbian, aplikácia ServoBlaster, framework UV4L, framework Tornado a vývojové prostredie Microsoft Visual Studio.

Obsah

Zoznam skratiek a značiek

AC	Alternating current
Ah	Ampérhodina
AJAX	Asynchronous JavaScript and XML
AP	Access point
API	Application programming interface
B&W	Black and white
BGR	Blue-green-red
CCD	Charge-coupled device
CMD	Command Prompt
CPU	Central processing unit
CSI	Camera serial interface
DC	Direct current
DHCP	Dynamic Host Configuration Protocol
DMA	Direct memory access
ESC	Electronic speed control
fps	frames per second
FPV	First-person view
FS	File system
FTP	File Transfer Protocol
GPIO	General-purpose input/output
GPS	Global Positioning System
GST	GStreamer
GUI	Graphical user interface
HSV	Hue-saturation-value
HTML	HyperText Markup Language
HTPC	Home theater PC
HTTP	Hypertext Transfer Protocol
HUD	Head-up display
IP	Internet Protocol
JSON	JavaScript Object Notation
LED	Light-emitting diode
Li-Ion	Lithium-ion
Li-Pol	Lithium polymer

mAh	miliampérhodina
MMAL	Multi-Media Abstraction Layer
ms	milisekunda
NiMH	Nickel–metal hydride
OS	Operačný systém
P2P	Peer-to-peer
PC	Personal computer
PCB	Printed circuit board
PCM	Pulse-code modulation
PHP	PHP: Hypertext Preprocessor
PWM	Pulse-width modulation
px	pixel
RC	Radio controlled
ROI	Region of interest
RPi	Raspberry Pi
SB	ServoBlaster
SIFT	Scale-Invariant Feature Transform
SOC	System on a chip
SSH	Secure shell
SSID	Service Set Identifier
SURF	Speeded-Up Robust Features
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform resource locator
UV4L	User space Video4Linux
V4L	Video4Linux
WebRTC	Web Real-Time Communication
Wh	Watthodina
WPA	Wi-Fi Protected Access
µs	mikrosekunda

Úvod

Počítač Raspberry Pi podnietil vznik masívnej komunity vývojárov a nových elektronických projektov. Dokazujú to aj mnohé úspešné kampane davového financovania (anglicky crowdfunding) na portáloch, akým je napríklad Kickstarter. Či už ide o tenké terminály nahradzujúce pracovné stanice, vlastné multimediálne centrá s ovládačom, kávovary, IP kamery, herné konzoly alebo robotické stavebnice, všetky sú po technickej stránke založené práve na tomto miniatúrnom zariadení. Rozmach Raspberry Pi sa pripisuje predovšetkým jeho nízke cene, dostupnosti, popularizácii na školách a existencii veľkého počtu prídavných modulov.

V diplomovej práci som sa rozhodol využiť Raspberry Pi na prestavbu rádiovo ovládaného (ďalej RC) modelu auta na model riadený počítačom. Transformácia spôsobu riadenia umožní implementovať funkcie, ktorými disponujú skôr prémiové modely. Hovoríme napríklad o prenose videa (anglicky stream) z pohľadu prvej osoby (ďalej len FPV), interakcii s okolitými predmetmi alebo programovaní trasy. V práci sa tiež zameriam na minimalizáciu nákladov spojených s prestavbou a využitím čo najmenšieho počtu prídavných komponentov.

Mojím prvým cieľom je nahradiť pôvodný (páčkový) ovládač počítačovou myšou a klávesnicou. Zmena spôsobu ovládania bude vyžadovať zmenu spôsobu komunikácie a odosielania riadiacich príkazov. V tejto súvislosti porovnám spôsoby odosielania riadiacich príkazov technológiami AJAX a Web Sockets. Druhým cieľom je zabezpečiť plynulý prenos obrazu z pohľadu RC auta s minimálnou odozvou. Na tento účel použijem RPi kamerový modul. Streamovanie obrazu na PC obrazovku a ovládanie modelu klávesnicou a myšou poskytnú zážitok z riadenia na diaľku. Tretím cieľom je vyvinúť aplikáciu, ktorá umožní modelu interagovať s vonkajším prostredím. Na rozpoznávanie obrazu použijem knižnicu OpenCV. Všetka komunikácia medzi RPi a riadiacim počítačom bude prebiehať prostredníctvom WiFi siete.

Práca sa skladá z piatich častí. V prvej z nich opisujem použité komponenty a postup zostavenia modelu. Druhá časť sa zaoberá režimami bezdrôtového adaptéra. Tretia časť práce analyzuje spôsob zasielania riadiacich príkazov. Štvrtá časť porovnáva metódy streamovania obrazu. V poslednej časti sa venujem tvorbe

aplikácie autonómneho riadenia a doplnkovým funkciám. Súčasťou práce je CD obsahujúce zdrojové kódy a demonštračné videá k vytvoreným programom.

1 Zostavenie RC modelu a konfigurácia komponentov

V prvej časti práce si predstavíme jednotlivé komponenty RC modelu a ich základné nastavenia. Model bude prestavbou už existujúceho rádiovo ovládaného modelu auta na model ovládaný počítačom Raspberry Pi prostredníctvom WiFi siete. Naša prestavba zahŕňa inštaláciu RPi, kamerového modulu, bezdrôtového adaptéra a mobilného napájania. Kamerový modul použijeme na prenos FPV videa v reálnom čase, neskôr tiež na rozpoznávanie obrazu a interakciu s okolitými objektmi.

Tab. 1.1: Prehľad komponentov finálneho zostavenia modelu

	Max. spotreba (A)	**Obstarávacia cena (€)**
HPI Brama 10B RTR Buggy	65 (motor)	129
Raspberry Pi model A	0,5	19,76
RPi camera board 5 MP	0,25	22,79
TP-LINK TL-WN725N	0,25	8,69
CONNECT IT Power Bank 5200	2,1 (výstup)	15,54
ORION NiMH 5100mAh 7,2V	—	58
Spolu	**—**	**253,78**

1.1 Výber RC auta

Základnú časť zostavy tvorí RC auto. Riadenie počítačom RPi som testoval na dvoch modeloch áut. Prvý model (Obr. 1.1) s motorom na jednosmerný prúd (ďalej DC), druhý model (Obr. 1.2) s motorom na striedavý prúd (ďalej AC). V oboch autách nájdeme okrem motora aj elektronický rýchlostný regulátor (ďalej ESC), rádiový prijímač a servo. Rádiový prijímač používať nebudeme. Je potrebný iba vtedy, ak ovládame model ovládačom, ktorý býva bežnou súčasťou RC súpravy.

Obr. 1.1: Brama 10B 1:10 RTR Buggy

Obr. 1.2: Buggy Ranger Brushless 4WD RtR

Po overení charakteristík oboch motorov sa na naše účely ukázal ako vhodnejší model Brama 10B s DC motorom. Oproti modelu Buggy Ranger s AC motorom má motor Bramy vyššiu ťažnú silu, jemnejší nábeh a zvládne i náročnejší terén. Na rovné povrchy je naopak lepší rýchlejší model Ranger. Opisu fungovania motorov sa podrobnejšie venujem na strane 43 v kapitole Konfigurácia ESC a typy motorov.

1.2 Počítač Raspberry Pi

Počítač RPi je mini počítač veľkosti platobnej karty združujúci všetky komponenty na spoločnom PCB. Dôvodom vzniku bola potreba lacného PC určeného deťom, na účely výučby programovania, ktorý by bol k dispozícii celosvetovo. Počítač RPi sa hodí na stavbu HTPC, elektronické projekty a tiež bežnú kancelársku prácu. Vďaka architektúre SOC a hardvérovému Full HD enkodéru/dekodéru zvláda spracovať objemné toky dát. Je dostupný v niekoľkých verziách, ktoré sa líšia v konfigurácii a spotrebe. Prvými uvedenými na trh boli modely A a B. Z nich neskôr vznikli vylepšené verzie A+ a B+.

Obr. 1.3: Počítač RPi model A

Model B sa vyznačuje bohatšou výbavou, vyššou spotrebou aj cenou. Spoločnými prvkami základných verzií A a B sú: kompozitný video výstup, stereo audio výstup, skupina LED signalizačných diód, HDMI výstup, napájací konektor (micro USB), čítačka SD pamäťových kariet, GPIO piny a SOC čip BCM2835 architektúry ARM združujúci pamäť RAM, GPU, CPU (700 Mhz) a radič USB. Model B má oproti modelu A dvojnásobok pamäte RAM (512 MB), USB portov (2) a obsahuje LAN port pre Ethernet [1].

Spotreba modelu A je zhruba 0,5 A a modelu B 0,7 – 1 A, v závislosti od pripojených periférií. Prvá revízia RPi bola produkovaná v Ázii. Revízia 2.0 a novšie sa už vyrábajú priamo vo Veľkej Británii, odkiaľ celý projekt aj pochádza.

1.2.1 Inštalácia OS a práca s SD kartou

Diskom RPi je SD karta s odporúčanou kapacitou aspoň 8 GB. Pre ARM existuje niekoľko linuxových distribúcií OS. My sme zvolili Raspbian – komunitou tvorený port systému Debian Wheezy. Inštaláciu vykonáme v prostredí OS Windows. Distribúciu je možné stiahnuť v sekcii *Downloads* na oficiálnej stránke[1] RPi. Ďalej zo stránky stiahneme nástroj Win32 Disk Imager, ktorým zapíšeme obraz OS na SD kartu. Rovnakým nástrojom si z karty neskôr môžeme vytvoriť záložný obraz so všetkými vykonanými nastaveniami a nainštalovaným softvérom.

Súborový systém (ďalej FS) SD karty sa po nahratí distribúcie zmení na linuxový. Linuxové FS Windows štandardne nepodporuje. Pre prácu s nimi v tomto OS je

[1] http://www.raspberrypi.org

potrebné nainštalovať špeciálny ovládač – napríklad Paragon ExtFS[2]. Program nám umožňuje pracovať s FS typu Ext2/3/4 v režime čítania aj zápisu.

Keď máme OS nahratý, nasleduje rozšírenie súborového systému a aktivácia SSH prístupu. Uistime sa, či je k RPi zapojený TV a klávesnica. Počítač spustíme zapojením micro USB kábla do napájacieho konektora. Predvolené používateľské meno je *pi* a heslo *raspberry*. Po načítaní systému do terminálu zadáme príkaz:

```
sudo raspi-config
```

Najskôr expandujeme FS, aby využil plnú kapacitu SD karty (voľba *Expand Filesystem*). Podobne zapneme v menu podporu SSH konzoly pre vzdialený prístup cez sieť (*Advanced Options – SSH*). Na záver zvolíme *Finish* a reštartujeme RPi.

1.3 Kamerový modul pre RPi

Kamerový modul je malý PCB pripojiteľný do CSI konektora. Modul použijeme na záznam videa z pohľadu RC auta, pre účely vzdialeného riadenia a rozpoznávania obrazu. Kamera podporuje nahrávanie Full HD videa (1080p) so snímkovou frekvenciou 30 obrázkov za sekundu, ako aj bežné fotografovanie. Ponúka množstvo nastavení, efektov a režimov, napríklad time-lapse (zrýchlená fotografia) či slow-motion (spomalené video). Bola navrhnutá špeciálne pre RPi, aby využila jeho plný potenciál. K doske sa pripája pomocou 15cm pruhovaného kábla. K hardvéru sa dá pristupovať aplikačnými rozhraniami MMAL a V4L, pre ktoré existuje množstvo knižníc tretích strán [2].

[2] http://www.paragon-software.com/home/extfs-windows/index.html

Obr. 1.4: Kamerový modul pre RPi

Počítač RPi podporuje aj mnoho bežných webových kamier pripojiteľných cez USB port, avšak čo u kamerového modulu spracováva obraz GPU s hardvérovou akceleráciou, to u webovej kamery pripadá vo veľkej miere na CPU.

Parametre kamerového modulu:

- kompatibilný s oboma modelmi,
- senzor s rozlíšením 5 MPix,
- nahrávanie videa,
- rozlíšenie 1080p pri 30 fps,
- rozlíšenie 720p pri 60 fps,
- rozlíšenie 640x480p pri 60/90 fps,
- veľkosť 20 mm × 25 mm × 9 mm,
- hmotnosť 3 g,
- spotreba 250 mA.

Okrem základnej verzie kamerového modulu existuje tiež verzia NOIR (anglicky No Infra Red), ktorá nemá infračervený filter. Je vhodná na snímanie obrazu v infračervenom spektre, v podmienkach s nízkym osvetlením.

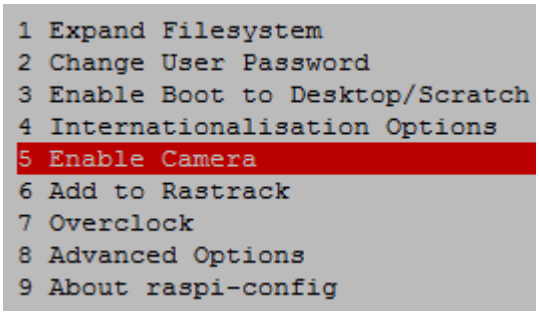
Konektor CSI, ku ktorému sa pripája kamera, sa nachádza vedľa HDMI portu. Pred zasunutím kábla je potrebné po stranách konektora uvoľniť plastové poistky vydvihnutím smerom nahor. Modrý pruh na kábli má smerovať k USB portu. Po zasunutí kábla poistky zasunieme späť, čím sa kábel zaistí. Na kamere môže byť po

zakúpení prilepená tenká priesvitná fólia, ktorá má iba ochrannú funkciu a pred použitím by mala byť odstránená.

Po zapojení je potrebné kameru aktivovať. Do príkazového riadka zadáme:

```
sudo raspi-config
```

V ponuke vyberieme *Enable Camera* a výber potvrdíme tlačidlom *Enter*.



Obr. 1.5: Aktivácia kamery v aplikácii raspi-config

Aby sa zabezpečil plynulý chod videa v najvyššom rozlíšení, prejdeme do časti *Advanced Options – Memory Split*, kde pridelíme grafickému čipu minimálne 128 MB z pamäte RAM. Následne prejdeme na *Finish*. Po potvrdení výzvy a reštartovaní RPi by mal byť modul inicializovaný, čo môžeme overiť príkazom *raspivid*. Ak máme k RPi pripojený monitor, na obrazovke uvidíme počas 5 s obraz [3].

1.4 Bezdrôtový adaptér

Kľúčovú požiadavku komunikácie medzi RPi a PC predstavuje bezdrôtový prístup. Náš RPi model A má iba jeden USB port a neobsahuje sieťový port. Preto s ním vieme pracovať len po pripojení klávesnice a displeja, alebo využitím WiFi adaptéra a SSH prístupu. Pripojenie ďalších zariadení (napríklad myši) vyžaduje rozbočovač – u zariadení s vyššou spotrebou aj s externým napájaním. Port USB modelu A dokáže dodať najviac 250 mA. Aby sme sa vyhli použitiu rozbočovača, potrebujeme vybrať vhodný WiFi adaptér.

Obr. 1.6: WiFi USB adaptér TP-LINK TL-WN725N

Adaptér TP-LINK WN725N túto požiadavku spĺňa. Spotrebúva hraničných 250 mA, podporuje rýchlosti do 150 Mbps a dokáže pracovať v režime klienta aj AP (viac o režimoch na strane 26). Ovládače adaptéra v distribúcii OS Debian pre RPi predvolene nie sú obsiahnuté, preto ich nainštalujeme dodatočne. Najskôr zistíme verziu nášho OS príkazom *uname -a*. Ovládač nainštalujeme vykonaním nasledovných príkazov:

```
sudo su
wget https://dl.dropboxusercontent.com/u/80256631/8188eu-201xyyzz.tar.gz
tar -zxvf 8188eu-201xyyzz.tar.gz
install -p -m 644 8188eu.ko /lib/modules/$(uname -r)/kernel/drivers/net/wireless
insmod /lib/modules/$(uname -r)/kernel/drivers/net/wireless/8188eu.ko
depmod –a
reboot
```

Reťazec „xyyzz“ sa nahrádza v závislosti od verzie jadra OS. Kompletné názvy archívov sú dostupné na stránke fóra[3]. Ak sa po vykonaní príkazov a reštartovaní RPi rozsvieti na WiFi adaptéri modrá dióda, inštalácia prebehla úspešne.

Čitateľovi sa zrejme podsúva otázka, ako má nainštalovať ovládače bez fungujúceho adaptéra. V prvom rade potrebujeme skopírovať archív s ovládačom na SD kartu. Na ostatné kroky postačí pripojená klávesnica s obrazovkou. Prístupu k linuxovej partícii SD karty na platforme Windows sa venujeme vyššie – v kapitole Inštalácia OS a práca s SD kartou. Inou možnosťou je použiť adaptér, ktorý nevyžaduje inštaláciu ovládačov, spojiť sa s RPi cez SSH a súbory skopírovať napríklad pomocou FTP klienta (viď Obr. 3.2).

[3] http://www.raspberrypi.org/forums/viewtopic.php?p=462982

1.5 Batérie pre napájanie RPi a RC modelu

Pri riadení RC modelu očakávame, že jeho dojazd nebude obmedzovaný dĺžkou napájacieho kábla. Mobilné napájanie RPi môžeme zrealizovať viacerými spôsobmi. Vyskúšal som a porovnal dva z nich. Najskôr išlo o napájanie pomocou bežných batérií. Neskôr, po zistení ich nedostatkov, boli nahradené hotovým produktom. Na napájanie RC modelu používame NiMH batériu s vysokým vybíjateľným prúdom.

1.5.1 Napájanie RPi sústavou nabíjateľných batérií

Na napájanie RPi potrebujeme dosiahnuť napätie medzi 5 – 6 V. Ak sa rozhodneme použiť batérie, je vhodné zapojenie do série, pri ktorom sa napätie sčítava a kapacita zostáva nezmenená. Aby sme dosiahli požadované napätie s nabíjateľnými NiMH tužkovými 1,2V AA článkami, potrebujeme zapojiť do série 5 batérií. Ak by sme použili články s napätím 1,5 V, postačovali by aj 4 kusy. V oboch prípadoch však treba počítať s poklesom napätia. U prvého príkladu síce máme istú rezervu, ale vzhľadom na rýchlosť poklesu napätia takéto zapojenie neodporúčam.

Vhodnejším riešením je použiť viac článkov a napätie regulovať. Zapojením 8 článkov s napätím 1,2 V do série dôjdeme k hodnote 9,6 V. Ďalej túto hodnotu budeme znižovať pomocou automobilového USB DC-DC regulátora napätia na výsledných 5 V.

Obr. 1.7: Napäťový regulátor CONNECT IT CI-176 Car Charger Mini

Regulátor má dva USB nabíjacie porty. Prvý port vie dodať 2,1 A, druhý 1 A. Batérie sú umiestené v plastovom držiaku, ktorý je vybavený vodivými kontaktmi na vytvorenie sériového zapojenia. Pri zapájaní vodičov medzi držiakom a regulátorom berme zreteľ na správnu polaritu. Napäťový vodič sa nachádza v strede a uzemňovací

na bočnej strane regulátora. Napäťový vodič pripájame k kladnému terminálu akumulátorového boxu. Nesprávnym zapojením dôjde k skratu a okamžitému zničeniu regulátora.

Obr. 1.8: Napájanie pomocou 8 AA batérií

Aby sme nemuseli obvod zopínať zasúvaním batérií do držiaka, pridáme medzi terminály držiaka a regulátora mechanický spínač. Ak uvažujeme celkovú spotrebu RPi 1 A a batérie s kapacitou 2000 mAh, v ideálnom prípade by prevádzková doba bola 3,84 h ((9,6 × 2) ÷ (5 × 1)). Vzhľadom na energetické straty, vplyvom regulácie napätia a poklesu napätia na článkoch, treba počítať so skrátením tejto doby.

Výhodou napájania RPi s bežnými AA článkami je ich všestrannosť a ľahká dostupnosť. Držiak batérií výrazne napomáha v priestorovom usporiadaní a spôsobe zapojenia. Ako nevýhoda sa ukázal byť nedostatočný prítlak batérií k kontaktom držiaka, respektíve ich náchylnosť k uvoľneniu pri otrasoch. K uvoľneniu dochádzalo pomerne často, a to najmä pri nárazoch alebo prechodom cez hrboľaté prekážky. Bez pridaného spínača bolo takisto nutné zapínať RPi, buď zasúvaním batérie alebo kábla do USB portu regulátora. Z týchto dôvodov bolo vhodné vyhľadať nejakú kompaktnú alternatívu, ktorá by zahŕňala všetky uvedené súčasti zároveň.

1.5.2 Napájanie RPi externou batériou

Lepším riešením sa ukázalo použitie externej batérie pre tablety a mobilné telefóny. Zvolil som batériu CONNECT IT Power Bank 5200, ktorá v sebe integruje

Li-Ion akumulátor s kapacitou 5200 mAh, 5V DC regulátor napätia, štvoricu LED diód indikujúcich stav nabitia a tlačidlo na rozsvietenie diód.

Obr. 1.9: Batéria CONNECT IT CI-247 Power Bank 5200

Akumulátor typu Li-Ion má z dlhodobého hľadiska v porovnaní s NiMH článkami minimálny pokles kapacity pri nabíjaní. Dokáže uchovať až 26 Wh (5 V × 5,2 Ah) energie, čo je viditeľný rozdiel voči 19,2 Wh u AA článkov (9,6 V × 2 Ah). Na nabíjanie slúži micro USB port. Výstupný (klasický) USB port vie dodať maximálne 2,1 A. V pomere cena/výkon ide o podstatne výhodnejšiu voľbu ako kupovanie tužkových batérií. Pri cene vyššej len o zhruba 1 € sú náklady porovnateľné so sústavou nabíjateľných batérií bez regulátora, s ktorým sa celková suma ešte navýši. Dôležitým faktorom je aj nižšia hmotnosť – 120 g (batéria) vs. 225 g (sústava AA článkov). Prevádzková doba batérie vychádza s uvedenými parametrami na 5,2 hod.

1.5.3 Napájanie RC modelu

Na napájanie RC modelov sa používajú pohonné akumulátory s vysokou zaťažiteľnosťou. Medzi najlacnejší typ patria NiMH batérie. Okrem nich sa môžeme stretnúť s drahšími a druhými najpoužívanejšími – Li-Pol batériami, ktoré sú využívané pre svoju nižšiu hmotnosť najmä v lietajúcich modeloch.

Pre naše potreby som zvolil NiMH akumulátor s nasledovnými parametrami:

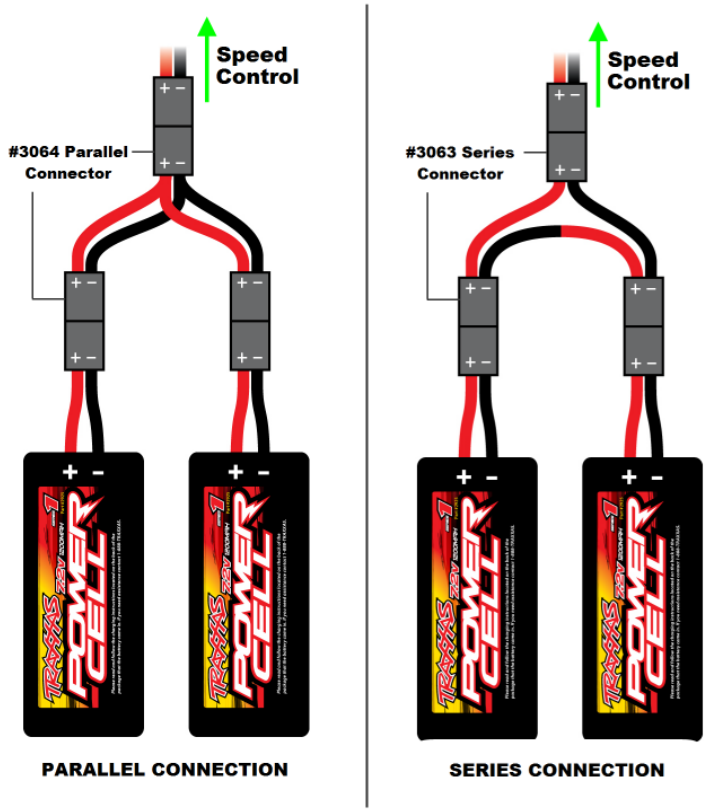
- hmotnosť: 431 g,
- počet článkov: 6,
- napätie: 7,2 V (1,2 V na článok),
- kapacita: 5100 mAh,

- krátkodobá zaťažiteľnosť: do 150 A,
- rozmery: 140 × 47 × 24 mm.

Obr. 1.10: Akumulátor 5100 mAh 7,2V NiMH

Spôsob nabíjania sa u oboch uvedených typov batérií odlišuje. Batérie Li-Pol, na rozdiel od batérií NiMH, obsahujú okrem napájacích vodičov aj vodiče pre takzvaný balancer. Funkciou balancera je sledovanie nabíjacieho procesu u jednotlivých článkov, aby nedošlo k ich prebitiu. V prípade prebitia môže u Li-Pol batérií dôjsť k požiaru. Balancer býva súčasťou lepších nabíjačiek, ale predáva sa aj samostatne [4].

Ak máme viacero (rovnakých) batérií a chceme ich nabíjať súčasne, sú dve možnosti zapojenia – sériové alebo paralelné. Pri sériovom zapojení sa sčítava napätie a pri paralelnom kapacita batérií.

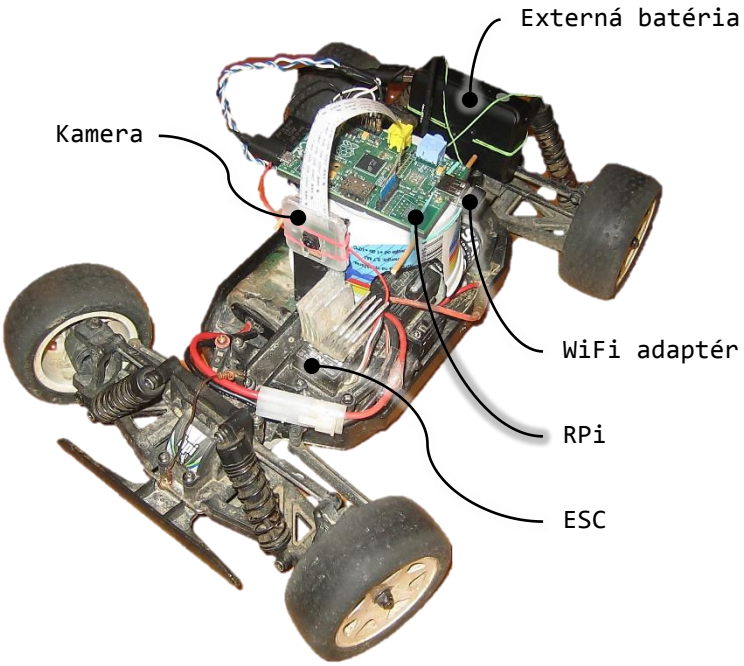


Obr. 1.11: Konektory pre paralelné (vľavo) a sériové (vpravo) zapojenie batérií [5]

Pri nabíjaní s paralelným zapojením sa výsledný nabíjací prúd vypočíta ako súčin prúdu, ktorým bežne napájame jednu batériu a počtu paralelne zapojených batérií. Batérie NiMH pri nabíjaní zapájame výlučne do série, pretože u paralelného zapojenia dochádza kvôli prelievaniu elektrického náboja k ich postupnému vybíjaniu.

1.6 Uchytenie komponentov

V predošlých kapitolách sme sa venovali jednotlivým komponentom, z ktorých bude RC model zložený. V ďalšom kroku na auto uchytíme RPi, kameru a externú batériu. Výsledné zostavenie znázorňuje Obr. 1.12. Obrázok zahŕňa aj zapojenie vodičov serva a ESC, ktorému sa venujeme zvlášť v kapitole Prepojenie RC modelu a RPi, na strane 41.



Obr. 1.12: Výsledné zostavenie RC modelu Brama 10B

Počítač RPi sme uchytili pomocou plastového téglika prilepeného tavnou pištoľou k šasi modelu. Spájacím materiálom je plast s nízkou teplotou tavenia. Do téglika bol vystrihnutý otvor na vývod káblov. V hornej časti téglika boli hrotom spájkovačky vytvorené kruhové otvory, do ktorých sa zasunuli špajdle. Počítač RPi sa k tégliku pripevnil obopnutím gumičiek medzi vrchom PCB a vyčnievajúcich koncov špajdlí.

Na uchytenie kamery bol použitý kus CD obalu, ktorý sme prilepili priesvitným silikónom k šasi a ručne vyrobenému chladiču. Chladič je uchytený navrchu ESC. Bol vyrobený z chladiča PC procesora. Týmto dodatočným chladením vieme počas letných dní predísť prehriatiu a vypínaniu ESC. Kameru sme následne zachytili dvoma gumičkami umiestnenými nad a pod CCD snímačom.

Batéria je tvaru kvádra so zaoblenými rohmi. Nenájdeme na nej žiadne miesta, o ktoré by sa dala prichytiť k šasi. Riešením je použitie viacerých gumičiek a plastových uťahovačov, ktorými obopneme batériu vodorovne a zvislo po stranách, čím sa dostatočne stabilizuje.

1.7 Zhrnutie

V tomto okamihu máme model zo stránky základných komponentov kompletne zostavený. Z dvoch dostupných RC áut sme uprednostnili model vybavený DC motorom. Predstavili sme si verzie RPi, nainštalovali OS na SD kartu a expandovali

systém súborov, aktivovali kamerový modul a priradili mu dostatočnú pamäť. Bezdrôtový adaptér vyžadoval dodatočnú inštaláciu ovládačov, pretože ho OS štandardne nepodporoval. Porovnali sme možnosti mobilného napájania a z nich vybrali napájanie externou batériou, ktoré sa ukázalo ako najvhodnejšie nielen z hľadiska pomeru cena/výkon, ale aj kvôli výbave zahrňujúcej reguláciu napätia. V ďalšej kapitole budeme konfigurovať bezdrôtovú sieť, čo nám umožní spojiť sa s RPi vzdialene cez SSH.

2 Komunikácia medzi PC a RPi

Po zostavení RC modelu a nainštalovaní ovládača bezdrôtového adaptéra môžeme prejsť ku konfigurácii sieťového pripojenia. K počítaču RPi sa budeme pripájať z operačného systému Windows programom PuTTY [1] – klientom pre sieťové protokoly SSH, Telnet a Rlogin. Konfigurácia pripojenia bude spočívať v úprave parametrov systémových súborov, respektíve inštalácie dodatočného softvéru – v závislosti od vybraného pracovného režimu adaptéra.

Adaptér TP-LINK TL-WN725N podporuje režim klienta aj režim prístupového bodu (AP). V režime klienta si uvedieme postup pripojenia k bežnému bezdrôtovému smerovaču – WiFi router, a rovnako aj k mobilnému telefónu Lumia 520. Režim AP nám naopak umožní vytvoriť prístupový bod priamo z RPi, čím sa odstráni potreba prostredníka v podobe smerovača.

2.1 Počítač RPi ako klient

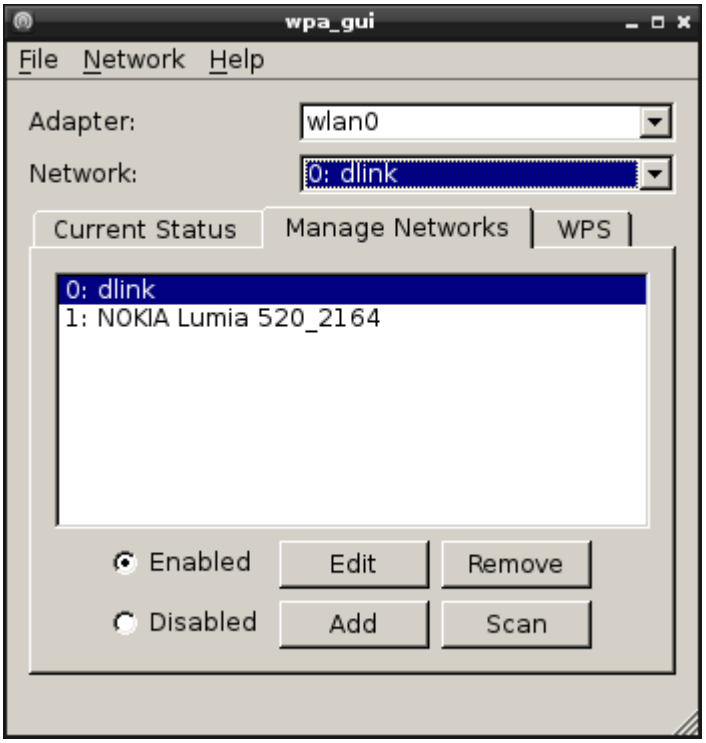
Režim klienta je predvoleným režimom WiFi adaptéra. Aj keď má RPi štandardne nastavené dynamické prideľovanie IP adresy, potrebujeme explicitne určiť, k ktorému AP sa má pripájať. Sú dva spôsoby, ako to urobiť. Jednoduchší spôsob – cez grafické rozhranie (ďalej GUI), alebo zložitejší spôsob – ručne v konfiguračnom súbore.

2.1.1 Konfigurácia siete cez GUI

Aktivujeme GUI príkazom *startx*. Automatické aktivovanie GUI pri štarte RPi nastavíme po zadaní príkazu *sudo raspi-config* voľbou *Enable Boot to Desktop*. Ďalej otvorme z pracovnej plochy utilitu WiFi Config. Pomocou tohto nástroja vieme jednoduchým spôsobom konfigurovať bezdrôtové pripojenia. Práca s programom vyžaduje okrem klávesnice aj pripojenie myši, preto sa u RPi modelu A zrejme nevyhneme použitiu USB rozbočovača.

V otvorenom okne vyberieme adaptér *wlan0* a klikneme na tlačidlo *Scan*. Otvorí sa ďalšie okno, kde opäť klikneme na tlačidlo *Scan*. V zozname sa zobrazia dostupné bezdrôtové siete. Zvolenú sieť vyberieme dvojklikom. V ďalšom okne skontrolujeme parametre AP a klikneme na tlačidlo *Add*.

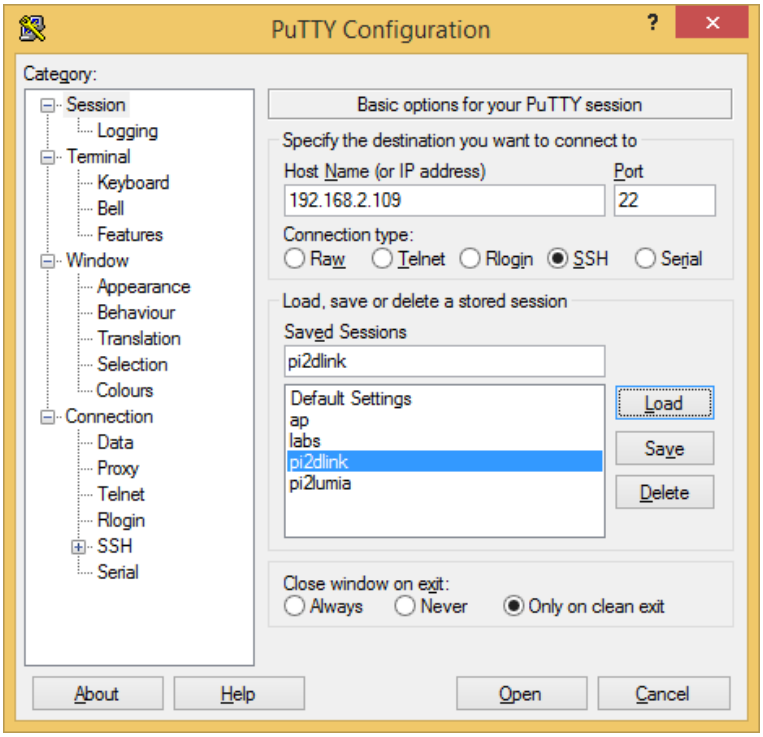
[1] http://www.chiark.greenend.org.uk/~sgtatham/putty



Obr. 2.1: Konfigurácia siete v aplikácii WiFi Config

Po pridaní sieťového pripojenia uvidíme na karte *Manage Networks* zoznam uložených AP. Vyberme tie, ktoré budeme používať a každý z nich prepnime do stavu *Enabled*. Sieť je v prípade potreby manuálnej konfigurácie možné pridať tlačidlom *Add*. Následne vyberieme z výsuvného zoznamu *Network* zvolený AP, čím automaticky dôjde k pripojeniu. Z karty *Current Status* si poznačme IP adresu.

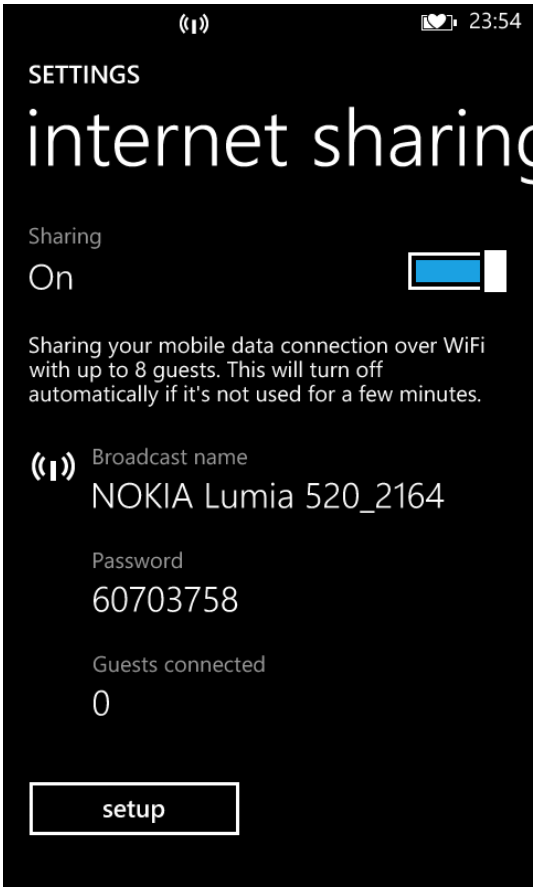
Od tohto okamihu vieme z bežného PC/notebooku pomocou SSH ovládať RPi bezdrôtovo, napríklad programom PuTTY. Do poľa *host* vložíme získanú IP adresu RPi, port *22* a typ pripojenia *SSH*. V prípade, že sa nám s RPi nepodarí spojiť, môžeme skúsiť dočasne vypnúť firewall, eventuálne antivírusový softvér. Po pripojení uvidíme okno terminálu, s ktorým sa pracuje rovnako ako pri fyzickom zapojení s TV.



Obr. 2.2: PuTTY – pripojenie k RPi

V zozname AP (Obr. 2.1) si môžeme všimnúť dve dostupné siete. Sieť *dlink* je vytvorená bezdrôtovým smerovačom a sieť *NOKIA Lumia 520_2164* mobilným telefónom Nokia Lumia 520. V telefóne nie je možné vytvoriť WiFi sieť samostatne bez toho, aby sme zdieľali internet a mali aktivované mobilné dáta.

Mobilné predplatené dáta aktivujeme v sekcii *nastavenia – mobilná sieť*, časť *dátové pripojenie*. Funkciu zdieľania internetu aktivujeme v sekcii *nastavenia – zdieľanie internetu*. Parametrami siete sú meno siete (SSID) a heslo. K telefónu môže byť súčasne pripojených najviac 8 klientov. Vytvorená sieť používa automaticky šifrovanie typu WPA2.



Obr. 2.3: Nokia Lumia 520 – zdieľanie internetu

2.1.2 Manuálna konfigurácia siete

Hoci nám GUI dovoľuje pohodlne spravovať bezdrôtové siete, poskytuje iba obmedzené možnosti konfigurácie. Povedzme, že budeme chcieť, aby bola IP adresa prideľovaná RPi staticky. To nám GUI neumožňuje, a teda sa nevyhneme ručnej úprave konfiguračných súborov. Nimi sú:

```
/etc/network/interfaces
/etc/wpa_supplicant/wpa_supplicant.conf
```

Súbor *interfaces* obsahuje zoznam WiFi pripojení a nastavenia IP adries:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0

iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf

iface lumia inet static
address 192.168.137.73
netmask 255.255.255.0
gateway 192.168.137.1
```

```
iface dlink inet static
address 192.168.2.109
netmask 255.255.255.0
gateway 192.168.2.222

iface default inet dhcp
```

V zozname sme definovali dve sieťové pripojenia (*lumia*, *dlink*) so statickým pridelením IP adresy. Ak sa napríklad budeme chcieť pomocou PuTTY spojiť s RPi cez AP *lumia*, do poľa *host* zadáme adresu 192.168.137.73 a pre *dlink* zadáme adresu 192.168.2.109.

Súbor *wpa_supplicant.conf* bližšie definuje jednotlivé parametre pripájania:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
        ssid="dlink"
        key_mgmt=NONE
        auth_alg=OPEN
        id_str="dlink"
}

network={
        ssid="NOKIA Lumia 520_2164"
        psk="60703758"
        proto=RSN
        key_mgmt=WPA-PSK
        pairwise=CCMP
        auth_alg=OPEN
        priority=10
        id_str="lumia"
}
```

Parametre pripájania je možné plne konfigurovať aj v GUI na karte *Manage Networks* tlačidlom *Edit*. Všimnime si vzťah medzi hodnotou *id_str* a názvom siete v súbore *interfaces*. Ďalej, hodnota *priority* (predvolene 0) uprednostní ten SSID, v ktorom je vyššie číslo. Čiže, ak by vysielali oba AP (*dlink* aj *lumia*) súčasne, RPi by sa pripojil k Lumii.

2.2 Počítač RPi ako prístupový bod

Druhým režimom WiFi adaptéra je režim AP. V tomto režime dokáže RPi suplovať funkciu bezdrôtového smerovača. Nie je potrebné žiadne ďalšie zariadenie. Tým sa znateľne zredukuje hmotnosť RC zostavy a spotreba energie. Nevýhodou môže byť chýbajúca internetová konektivita, keďže RPi model A neobsahuje sieťový port. Ani u modelu B nemá otázka smerovania internetových paketov svoje

opodstatnenie. Museli by sme totiž použiť sieťový kábel, čo by nebolo v súlade s požiadavkou mobility.

Vytvorenie AP z RPi pozostáva z troch častí:

1. Inštalácia softvéru.
2. Konfigurácia DHCP servera.
3. Konfigurácia AP.

2.2.1 Inštalácia softvéru

V prvom kroku budeme na RPi inštalovať softvér potrebný na vytvorenie AP. Táto operácia vyžaduje pripojenie na internet – pozri režim klienta. Do terminálu zadajme tieto príkazy:

```
sudo apt-get update
sudo apt-get install isc-dhcp-server
```

Prvý príkaz aktualizuje repozitár a druhý nainštaluje DHCP server. Na vytvorenie hosťujúceho AP použijeme program *hostapd*. Hoci aplikáciu nájdeme aj v repozitári, nemáme zaručené, že bude pracovať so zvoleným WiFi adaptérom. Preto si skompilujeme svoju vlastnú verziu.

Zo stránky[2] výrobcu adaptéra stiahneme a nainštalujeme ovládače (zip archív) modelu RTL8188CUS pre Unix (Linux):

```
sudo unzip RTL8188C_8192C_USB_linux_v4.0.2_9000.20130911.zip
cd RTL8188C_8192C_USB_linux_v4.0.2_9000.20130911
cd wpa_supplicant_hostapd
sudo tar -xvf wpa_supplicant_hostapd-0.8_rtw_r7475.20130812.tar.gz
cd wpa_supplicant_hostapd-0.8_rtw_r7475.20130812
cd hostapd
sudo make
sudo make install

sudo mv hostapd /usr/sbin/hostapd
sudo chown root.root /usr/sbin/hostapd
sudo chmod 755 /usr/sbin/hostapd
```

Keď sme skončili, reštartujeme RPi (*sudo reboot*) [6].

2.2.2 Konfigurácia DHCP servera

V ďalšom kroku upravíme konfiguračný súbor DHCP servera, ktorý zabezpečí automatické prideľovanie IP adries pripojeným klientom [6].

[2] http://www.realtek.com.tw/search/default.aspx?keyword=RTL8188CUS

Súbor otvoríme príkazom:

```
sudo nano /etc/dhcp/dhcpd.conf
```

Vyhľadáme v ňom tieto riadky:

```
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;
```

Na začiatok nájdených riadkov pridáme mriežku (znak komentára):

```
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;
```

Z ďalšieho riadka s textom „#authoritative;“ naopak mriežku odstránime.

Potom prejdime na koniec súboru, kde pridáme riadky:

```
subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.10 192.168.42.50;
option broadcast-address 192.168.42.255;
option routers 192.168.42.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

Adresy IP budú prideľované z rozsahu 192.168.42.10 až 192.168.42.50 a adresa RPi bude 192.168.42.1. Nakoniec súbor uložíme – CTRL+X, potom Y a ENTER.

Ďalej otvoríme súbor *isc-dhcp-server*:

```
sudo nano /etc/default/isc-dhcp-server
```

V spodnej časti nájdeme riadok s textom „INTERFACES=""“, ktorý zmeníme na:

```
INTERFACES="wlan0"
```

Pre pridelenie statickej IP k RPi upravíme obsah súboru */etc/network/interfaces* na:

```
auto lo

iface lo inet loopback
iface eth0 inet dhcp

allow-hotplug wlan0

iface wlan0 inet static
address 192.168.42.1
netmask 255.255.255.0
```

2.2.3 Konfigurácia AP

Posledným krokom je konfigurácia prístupového bodu. Ukážeme si, ako nastaviť otvorenú (nešifrovanú) sieť, k ktorej sa môže pripojiť ľubovoľný klient [6].

Začneme vytvorením konfiguračného súboru:

```
sudo nano /etc/hostapd/hostapd.conf
```

Do súboru vložíme:

```
interface=wlan0
driver=rtl871xdrv
ssid=rpi
hw_mode=g
ieee80211n=1
wmm_enabled=1
channel=6
auth_algs=1
```

Hodnota *driver* sa môže u jednotlivých WiFi adaptérov líšiť. Najčastejšie sa stretneme s *rtl871xdrv* a *nl80211*. Názov siete je SSID a uvidíme ho v zozname AP.

Ak by sme chceli nastaviť šifrovanú sieť, doplníme nasledovné riadky:

```
macaddr_acl=0
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=heslo
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

Pri ukladaní súboru sa uistime, že sa za riadkami nenachádzajú žiadne prázdne znaky navyše. Mohlo by to spôsobiť neplatnosť nastavení.

Po vytvorení konfiguračného súboru zadajme príkaz:

```
sudo nano /etc/default/hostapd
```

V súbore *hostapd* potrebujeme definovať cestu k konfiguračnému súboru. Nájdime reťazec „#DAEMON_CONF=""“ a nahraďme ho riadkom:

```
DAEMON_CONF="/etc/hostapd/hostapd.conf"
```

Nezabudnime pritom na odstránenie mriežky. Po úprave súbor uložíme. Na záver konfigurácie AP aktivujme štartovanie aplikácie *hostapd* a DHCP servera pri štarte OS:

```
sudo update-rc.d hostapd enable
sudo update-rc.d isc-dhcp-server enable
```

Pozn.: Pre deaktiváciu automatického štartovania namiesto *enable* zadáme *disable*.

Keď sú nastavenia kompletné, reštartujeme RPi (*sudo reboot*). Po opätovnom štarte by mal WiFi adaptér konečne pracovať v režime AP a my by sme mali byť schopní spojiť sa s RPi cez vyššie uvedenú IP adresu (192.168.42.1).

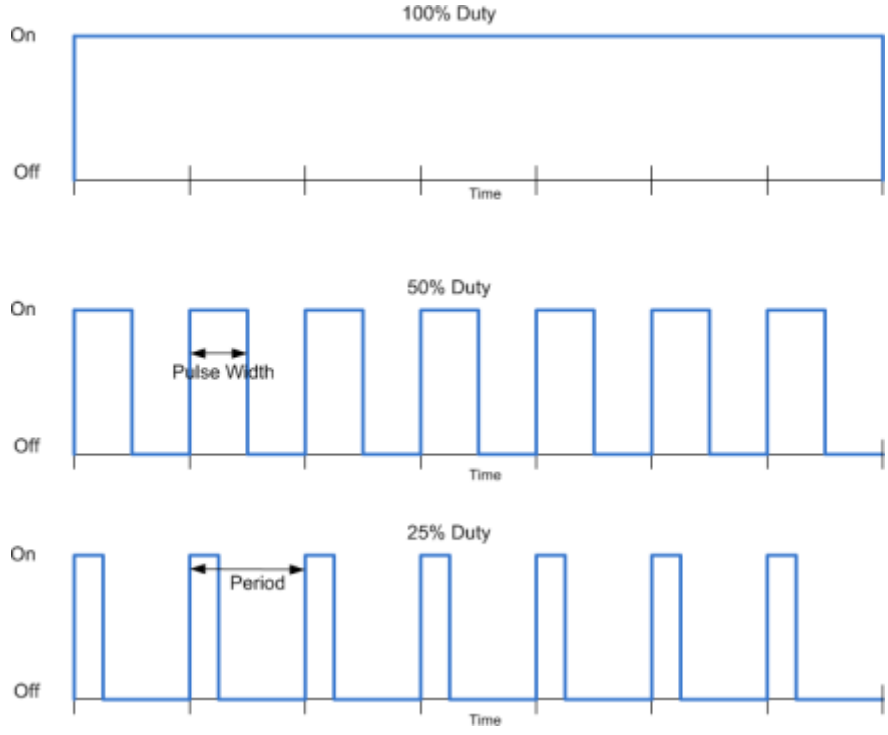
2.3 Zhrnutie

Touto kapitolou sme uzatvorili prípravnú fázu práce a nadobudli všetky predpoklady pre plnenie jej hlavných cieľov. Sme schopní bezdrôtovo komunikovať s RPi. Pripájame sa k nemu buď cez prostredníka (router, telefón) alebo priamo – vtedy RPi pracuje ako prístupový bod. Sieť WiFi sme konfigurovali manuálne aj pomocou grafického rozhrania. Odteraz budeme používať RPi temer stále v režime AP, čím eliminujeme závislosť od pokrytia signálu externého AP a maximalizujeme rýchlosť komunikácie. V nasledovnej kapitole analyzujeme program ServoBlaster a jeho použitie na riadenie RC modelu pomocou RPi.

3 Riadenie serv a ESC aplikáciou ServoBlaster

ServoBlaster (ďalej SB) je ovládač jadra, ktorý umožňuje na RPi ovládať servá/ESC prostredníctvom GPIO pinov. Pozícia serva je kontrolovaná odosielaním príkazov ovládaču, určujúcich šírku pulzu, ktorú má servo použiť. Ovládač uchováva nastavenú šírku pulzu dovtedy, kým pošleme nový príkaz s inou šírkou.

Predvolene môžeme kontrolovať 8 pinov. Maximálny počet konfigurovateľných pinov je 21. Servá bežne potrebujú na aktiváciu vysoký pulz so šírkou v rozpätí 0,5 ms – 2,5 ms, kde šírka pulzu riadi pozíciu serva. Pulzy by mali byť opakované približne každých 20 ms, hoci frekvencia pulzu nie je rozhodujúca. Dôležitá je šírka pulzu, keďže sa priamo prejaví v natočení serva. Intervalom obnovovania rozumieme dĺžku cyklu (anglicky duty cycle). Na nasledovnom obrázku vidíme 3 priebehy impulzovej šírkovej modulácie (ďalej len PWM) signálu:



Obr. 3.1: Impulzová šírková modulácia riadenia serva [7]

Prvý priebeh znázorňuje pravú krajnú polohu serva, dĺžka pulzu je rovná dĺžke duty cycle. V druhom priebehu je servo vycentrované, dĺžka pulzu je polovičná. V poslednom priebehu má servo polovičné natočenie vľavo. Podobným spôsobom vieme ovládať ESC. Ak motor nepodporuje spätný chod (anglicky reverse), intuitívne

očakávame, že pri polovičnej šírke pulzu (1,5 ms) budú polovičné otáčky motora, pri 100% (2,5 ms) šírke pôjde motor na plný výkon a pri 0% (0,5 ms) šírke bude stáť.

ServoBlaster môže byť konfigurovaný na generovanie pulzov so šírkou 0 – 100 % celkovej dĺžky cyklu, čo nám umožňuje použiť ho napríklad aj na riadenie jasu LED diód. Program po štarte vytvorí súbor zariadenia */dev/servoblaster*, na ktorý budeme zasielať príkazy. Príkaz môže mať tvar:

```
<cislo-serva>=<pozicia-serva>
```

alebo

```
P<hlavicka>-<pin>=<pozicia-serva>
```

V prvom prípade *<cislo-serva>* je číslo serva, ktoré má predvolenú hodnotu 0 – 7. V druhom prípade *<hlavicka>* je číslo 1 alebo 5 – podľa toho, ku ktorej hlavičke je servo pripojené, *<pin>* je číslo pinu z príslušnej hlavičky. Položka *<pozicia-serva>* je šírka pulzu v násobkoch 10 µs (0,01 ms). Hodnota násobku (alebo tiež veľkosť kroku) môže byť v prípade potreby zmenená cez príkazový riadok. Môže byť špecifikovaná aj v jednotkách µs alebo v percentách maximálnej povolenej šírky pulzu.

Ak chceme napríklad nastaviť servo s číslom 3 na šírku pulzu 1,2 ms, vykonáme to nasledovným príkazom:

```
echo 3=120 > /dev/servoblaster
```

Číslo 120 vyjadruje počet krokov, teda výsledná šírka pulzu je 1200 µs (120 × 10), respektíve 1,2 ms. Podľa druhého spôsobu zápisu by príkaz vyzeral takto:

```
echo P1-12=120 > /dev/servoblaster
```

Alternatívou k absolútnemu určovaniu šírky pulzu je relatívny zápis – vzhľadom k súčasnej pozícii serva, pomocou predpony „+“ alebo „-“. Nasledovným príkazom by sme pripočítali 10 krokov k aktuálnej polohe serva:

```
echo P1-12=+10 > /dev/servoblaster
```

Nastavenie inej šírky pulzu ako je násobok predvolenej veľkosti kroku vedie k zaokrúhleniu nadol k najbližšiemu násobku. Ak by sme napríklad odoslali príkaz „0=+1us“, pozícia serva sa nezmení a príkaz „0=-19us“ sa bude považovať za „0=-10us“. U relatívneho nastavovania sú tiež kontrolované hranice minimálnej a maximálnej možnej pozície serva. V prípade prekročenia šírky je táto automaticky

obmedzená podľa príslušnej hranice. Ak nastavíme servo na hodnotu 0, zastavíme odosielanie signálov, pričom aktuálna pozícia serva sa nezmení.

Štandardne je podporovaných 8 serv/ESC, ktoré sú mapované nasledovne:

Tab. 3.1: ServoBlaster – mapovanie pinov

Číslo serva	**GPIO pin**	**Pin v hlavičke P1**
0	4	P1-7
1	17	P1-11
2	18	P1-12
3	21/27	P1-13
4	22	P1-15
5	23	P1-16
6	24	P1-18
7	25	P1-22

P1-13 pripájame buď k GPIO-21 alebo 27, v závislosti od verzie dosky RPi.

Aktivovaním SB sa GPIO piny nastavia ako výstupy a ich šírky pulzov sa nastavia na 0 (neutrálna poloha). Akonáhle chceme pozíciu serva zmeniť, na aktiváciu príslušného výstupu nám stačí odoslať príkaz na */dev/servoblaster*. Ak sa ovládač nepodarí načítať alebo pri jeho ukončení, výstupy sa nakonfigurujú do pôvodného stavu, v akom boli pred aktivovaním ovládača.

Ovládač neodosiela riadiace príkazy na všetky servá súčasne, ale s určitým posunom v rámci celkovej dĺžky cyklu, čím sa minimalizuje spotreba potrebná na ich riadenie. Pracuje na princípe spájaného zoznamu DMA kontrolných blokov, pričom posledný blok ukazuje na prvý blok. Po inicializovaní riadiaci DMA obvod neustále prechádza týmto „kruhovým“ zoznamom. Ovládač pritom do ničoho nezasahuje, s výnimkou situácie, kedy je potrebné zmeniť šírku pulzu. V danom časovom okamihu existujú dva DMA kontrolné bloky. Prvý blok odosiela do registra slovo „clear output“, čím sa vyčistí výstup. Druhý blok odosiela niekoľko slov PWM frontu na vygenerovanie požadovaného časového úseku šírky pulzu. Podľa meraní autora SB (ďalej len autor) ovládač udržiava šírky pulzov a frekvenciu opakovania na stabilnej úrovni, a to aj pri vysokom vyťažení SD karty. Dalo sa to očakávať, keďže generovanie pulzov prebieha na strane hardvéru a nie je ovplyvnené odozvou pri prerušovaní ani plánovačom.

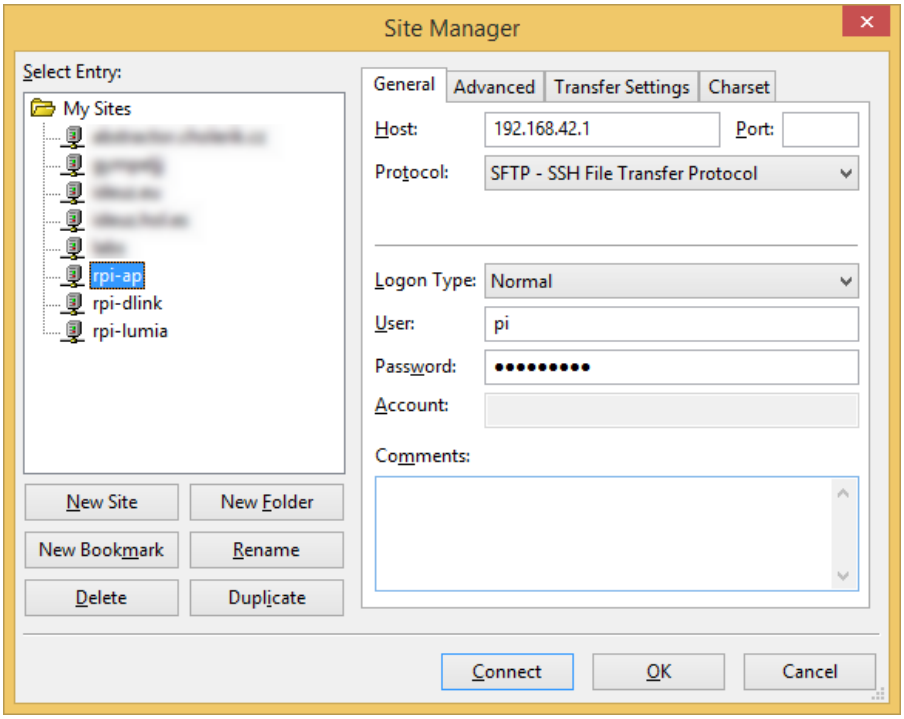
Ovládač používa DMA kanál 14 a PWM kanál 1. Nijakým spôsobom nekontroluje, či sú piny využívané inými procesmi. Pri aktivovaní konvertuje piny na výstupy, preto by sme ich v žiadnom prípade nemali použiť na opačný účel. Autor odporúča napájať

servá samostatným obvodom, hoci RPi má jeden 5V a jeden 3V3[1] pin. Zapojenie serv k týmto pinom môže viesť k nadmernej spotrebe, čo povedie k zrúteniu systému a následnému reštartu.

Existujú dve implementácie SB – modul jadra a používateľský modul. Modul jadra vznikol ako prvý. Na jeho skompilovanie potrebujeme mať prislúchajúcu verziu jadra OS. Používateľský modul to nevyžaduje a má niektoré funkcie navyše. Autor z týchto dôvodov odporúča používať používateľský modul [8].

3.1 Používateľský modul

Používateľský modul skompilujeme zo súborov *servod.c* a *Makefile*. Nachádzajú sa na webovej stránke[2] SB v adresári *user*, odkiaľ ich skopírujeme do RPi, napríklad programom FileZilla[3].



Obr. 3.2: Správca lokalít v programe FileZilla

Pripojiť sa môžeme priamo z hlavného okna programu, alebo si najskôr vytvoríme profil v ponuke *File – Site Manager* (Obr. 3.2). Potrebujeme vyplniť polia *Host* (vložíme IP adresu RPi) a *Protocol* (vyberieme *SFTP – SSH*). Ak sa pripájame

[1] Označenie 3V3 sa používa kvôli lepšej čitateľnosti na PCB a je rovnocenné zápisu 3,3V.

[2] https://github.com/richardghirst/PiBits/tree/master/ServoBlaster

[3] https://filezilla-project.org

z hlavného okna, kde výber protokolu nie je, zadáme port 22. Po prekopírovaní súborov do spoločného adresára ServoBlaster skompilujeme príkazom:

```
make servod
```

Automatické aktivovanie SB pri štarte OS nakonfigurujeme príkazom:

```
sudo make install
```

V adresári */etc/init.d*, ktorý obsahuje súbory automaticky aktivované pri štarte systému, pribudne nový skript s názvom *servoblaster*. Skript predvolene obsahuje konfiguračný parameter *idle-timeout* nastavený na 2 sekundy.

Na konfiguráciu programov aktivovaných pri štarte môžeme použiť program *update-rc.d*, ktorého štandardný zápis má tvar:

```
update-rc.d <nazov-skriptu> [<enable|disable|status>]
```

Programom teda vieme podľa potreby zapnúť/vypnúť automatické aktivovanie alebo zobraziť aktuálny stav daného skriptu, napríklad pre deaktiváciu aktivovania SB zadáme:

```
update-rc.d servoblaster disable
```

Základné informácie a zoznam parametrov SB zobrazíme príkazom:

```
./servod --help
```

SB konfigurujeme v tvare:

```
./servod [<parametre>]
```

Parametre sú voliteľné. Ak nezadáme žiadne parametre, použijú sa predvolené hodnoty. Zoznam parametrov opisuje Tabuľka 1 v prílohe A. Ak nie je zadaný parameter *idle-timeout*, riadiace príkazy SB zasiela pravidelne s časovým odstupom definovaným parametrom *cycle-time*. Parameter *p5pins* je platný len pre PCB 2. revízie, predvolenou hodnotou je prázdny reťazec.

Nasledovné príkazy sú v základnej konfigurácii SB navzájom ekvivalentné:

```
echo 0=150 > /dev/servoblaster
echo 0=50% > /dev/servoblaster
echo 0=1500us > /dev/servoblaster
echo P1-7=150 > /dev/servoblaster
echo P1-7=50% > /dev/servoblaster
echo P1-7=1500us > /dev/servoblaster
```

Pripomeňme si, že šírka pulzu môže byť zadaná aj relatívne vzhľadom k aktuálnej hodnote v krokoch, napríklad:

```
echo 0=+10 > /dev/servoblaster
echo 0=-20 > /dev/servoblaster
```

Po aktivovaní SB sa na výstupe zobrazí aktuálna konfigurácia parametrov a mapovanie pinov:

```
sudo ./servod

Board revision:                  1
Using hardware:                PWM
Using DMA channel:              14
Idle timeout:             Disabled
Number of servos:                8
Servo cycle time:            20000us
Pulse increment step size:     10us
Minimum width value:     50 (500us)
Maximum width value:   250 (2500us)
Output levels:               Normal

Using P1 pins:           7,11,12,13,15,16,18,22

Servo mapping:
     0 on P1-7           GPIO-4
     1 on P1-11          GPIO-17
     2 on P1-12          GPIO-18
     3 on P1-13          GPIO-21
     4 on P1-15          GPIO-22
     5 on P1-16          GPIO-23
     6 on P1-18          GPIO-24
     7 on P1-22          GPIO-25
```

Následne proces *servod* pracuje na pozadí, čo si môžeme overiť pomocou príkazu:

```
ps -ax
```

ServoBlaster ukončíme zápisom:

```
sudo killall servod
```

Ak chceme dosiahnuť pri riadení väčšiu plynulosť, môžeme nastaviť menšiu hodnotu parametra *step-size*, ako je predvolená hodnota. Poznamenajme, že zmena veľkosti kroku vyžaduje aj jemu zodpovedajúcu úpravu parametrov *min*, *max* a šírky pulzu. Riadenie LED diód môže vyžadovať kratší interval opakovania pulzov (parameter *cycle-time*), aby sa zabránilo blikaniu [8].

Ak chceme použiť napríklad iba dve servá a tieto majú byť označené ako 4 a 5, identifikátory nepoužitých serv nahradíme zástupným symbolom 0:

```
sudo ./servod --p1pins=0,0,0,0,15,16
...
Using P1 pins:           0,0,0,0,15,16

Servo mapping:
     4 on P1-15          GPIO-22
     5 on P1-16          GPIO-23
```

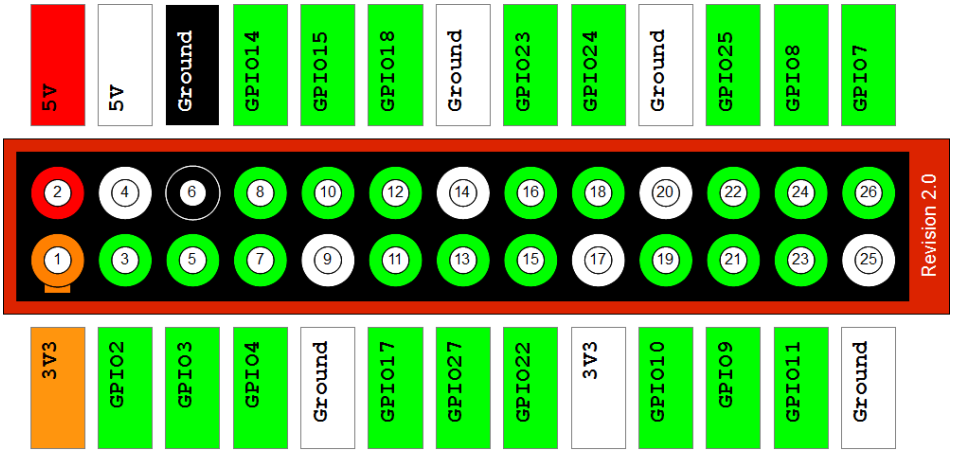
Ak chceme odkazovať na servá podľa pinov hlavičky P1, napríklad pod označením 7, 11, 12, 15 a prislúchajúcim k pinom 7, 11, 12, 15, parameter bude mať tvar:

```
--p1pins=0,0,0,0,0,0,0,7,0,0,0,11,12,0,0,15
```

V prípade, že určíme oba parametre *p1pins* a *p5pins*, čísla serv budú priradené podľa poradia, v akom sa parametre vyskytli. Úplný prehľad pinov hlavičiek P1 a P5 je možné nájsť na webovej stránke[4].

3.2 Prepojenie RC modelu a RPi

V predošlej kapitole sme uviedli postup inštalácie SB a spôsob používania. Teraz prejdeme k zapojeniu serva a ESC k RPi. Obr. 3.3 znázorňuje schému rozloženia GPIO pinov druhej revízie PCB, na ktorej nájdeme 2 × 3V3, 2 × 5V, 5 × uzemnenie (anglicky ground) a 17 signálnych pinov. Revíziu dosky a číselné mapovanie pinov, na ktoré budeme odosielať signály zistíme pri štarte SB.



Obr. 3.3: RPi – rozloženie GPIO pinov [9]

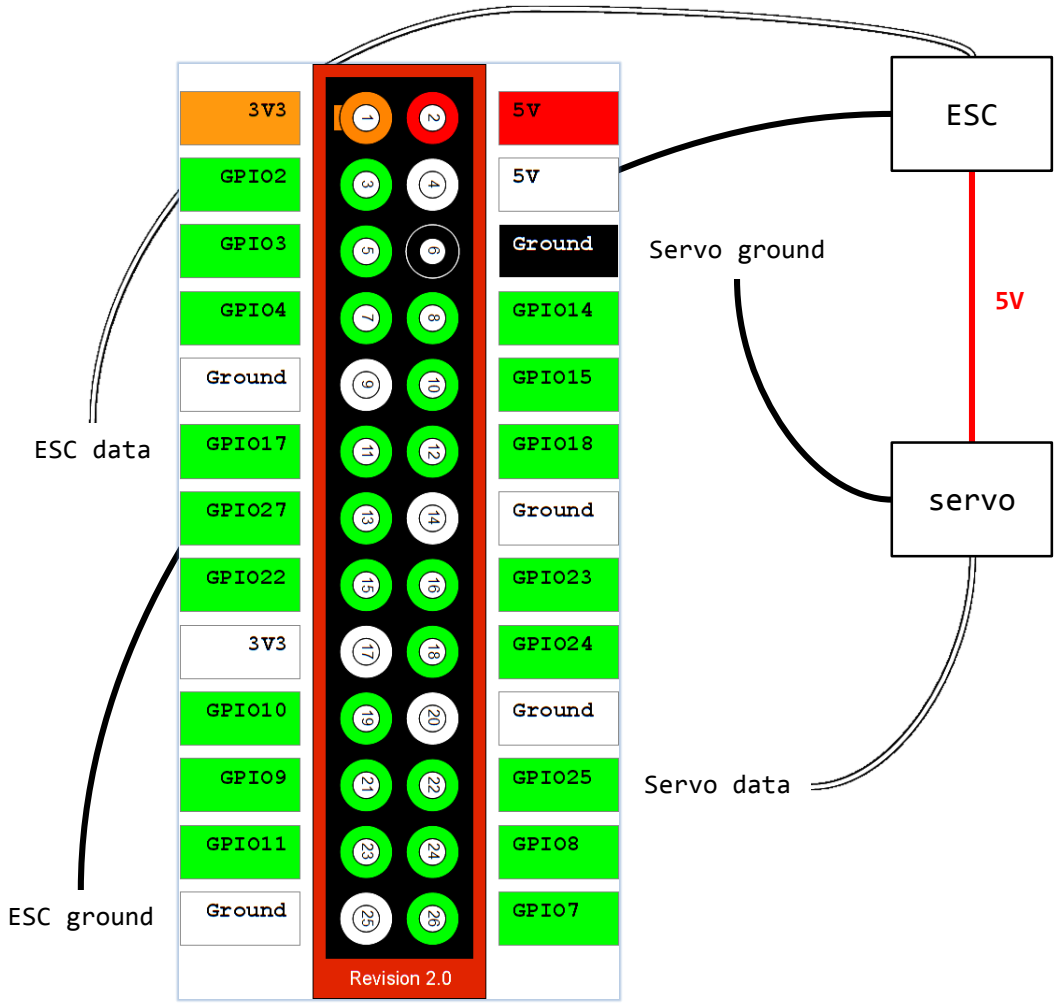
Zo serva typicky vedie trojica vodičov (ďalej len servo kábel) označených bielou, čiernou a červenou farbou. Červený kábel označujeme ako napájací, čierny ako uzemňovací a biely kábel ako dátový. Batériou je napájané ESC. Väčšina ESC dokáže napájať aspoň jedno servo, teda okrem dátového a uzemňovacieho vodiča majú aj napäťový vodič, ktorý sa pripája k servu.

V našom prípade zapojíme dátové vodiče servo káblov k signálnym pinom, uzemňovacie k ground a napájacie k 5V pinom. Regulátor ESC podporuje aj napájanie serva – postačuje spojiť napäťový vodič ESC s napäťovým vodičom serva. Ak by sme tieto napäťové vodiče pripojili na RPi k 5V pinom, ako sme pôvodne zamýšľali, napájali by sme tým zároveň aj RPi, ktoré by potom nebolo potrebné

[4] http://elinux.org/Rpi_Low-level_peripherals

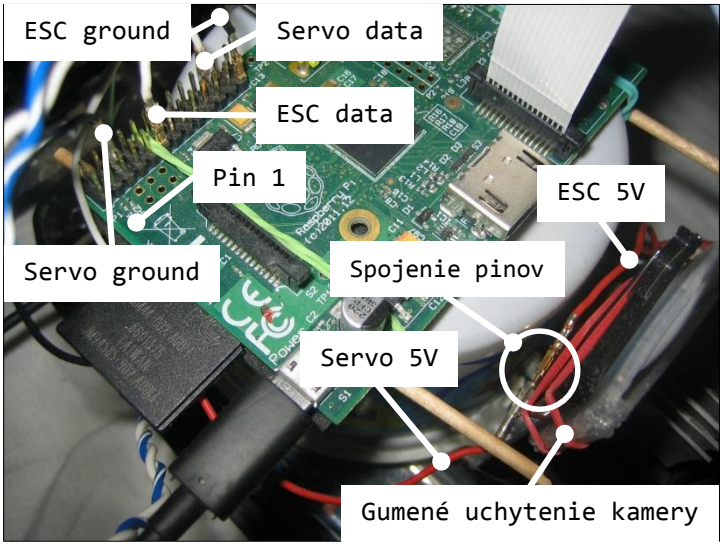
napájať štandardným spôsobom cez USB kábel. Treba však zdôrazniť, že zapojenie cez štandardný napájací konektor zahŕňa prepäťovú ochranu (do 6 V). Napájanie RPi cez GPIO piny túto ochranu nemá a nesprávnym zapojením môže ľahko dôjsť k jeho nenávratnému poškodeniu [1].

Na riadenie ESC (riadi motor) si zvoľme pin 11 (GPIO17) a na riadenie serva pin 22 (GPIO25). Pin 11 má v mapovaní SB číslo 1 a pin 22 číslo 7. K pinu 11 pripojíme dátový vodič vedúci z ESC a k pinu 22 dátový vodič vedúci zo serva. Uzemňovacie vodiče pripojíme ľubovoľne k pinom označeným ako ground. Napäťové vodiče (5V) spojíme mechanicky alebo letovaným spojom.



Obr. 3.4: Schéma zapojenia pinov RC modelu k RPi

Výsledné zapojenie kabeláže znázorňuje nasledovný obrázok:



Obr. 3.5: Zapojenie pinov RC modelu k RPi

Na spojenie napäťových vodičov ESC a serva bol použitý kúsok jednožilového odizolovaného medeného drôtu. Správnosť zapojenia overíme vyskúšaním serva. K regulátoru ESC pripojme akumulátor a do príkazového riadka zadajme:

```
echo 7=150 > /dev/servoblaster
```

Rameno serva sa nastaví do neutrálnej polohy. Pripočítaním alebo odpočítaním hodnoty sa vychýli vľavo, respektíve vpravo. Na riadenie ESC nahradíme vo vyššie uvedenom príkaze číslo 7 číslom 1.

3.3 Konfigurácia ESC a typy motorov

Prakticky každé ESC vyžaduje pri štarte počiatočnú inicializáciu. Ak sme doposiaľ RC model ovládali ovládačom a teraz ho chceme riadiť počítačom, bude pravdepodobne nutné ESC preprogramovať. Postup programovania je uvedený v manuáli ESC a spočíva v nastavení nových hodnôt krajných polôh a strednej (neutrálnej) polohy.

Krajné, a teda aj stredné hodnoty môžu byť pre konkrétne servo či ESC rozdielne. Podľa typu motora sa u oboch testovaných modelov (pozri stranu 12) líšia aj príslušné ESC. Motor DC obsahuje komutátor, ktorý v AC motore (označovaný ako brushless) nenájdeme. Motor AC má 3 drôtené cievky (vinutia) v statore – nepohybujúcej sa časti a trojicu magnetov na rotore – otáčajúcej sa časti. Motor poháňa trojfázový signál z troch vodičov. Otáčavý pohyb je dôsledkom ich sekvenčného napájania. Regulátor ESC potom konvertuje jednu fázu digitálnych pulzov do troch fáz napájajúcich cievky. Ak by však signály napájajúce cievky mali sínusový priebeh,

výkon by bol premenený na teplo všade tam, kde cievky nie sú plne nabudené, čo by bolo prinajlepšom plytvaním, prinajhoršom nebezpečné.

Aby sa takému správaniu predišlo, ESC obsahuje mosfety[5] (anglicky metal–oxide–semiconductor field-effect transistor) zostavené do takzvaného zjednodušeného H mostíka, ktorými prepína výstupný signál do dvoch stavov – zapnutý/vypnutý. Ak sa požaduje aj reverzný chod motora, mostík je v plnej konfigurácii. Na striedanie stavov mosfety nespotrebúvajú takmer žiaden prúd a vyžadujú len minimálne napätie. Rýchlosť motora je potom určená frekvenciou týchto pulzov, počtom trojíc magnetov, z ktorých motor pozostáva a priemerným výkonom určeným šírkou pulzov. Regulátor ESC používa indukčnú spätnú väzbu (alebo niekedy 4. vodič), aby sa zabezpečilo, že výkon určený šírkou pulzu je dostatočný na chod motora pri danej frekvencii. Prepínanie stavov prebieha prostredníctvom ESC na každej fáze motora a odvodzuje sa z jediného vstupného PWM signálu, ktorý privádzame na GPIO pin [10].

Nevýhodou konfigurácie SB je, že parametre sú totožné pre všetky GPIO piny. Ak sú napríklad prípustné hodnoty serva v rozmedzí 100 – 200 krokov, ESC by mal byť naprogramovaný rovnako. Pri motore DC mal ESC tieto hodnoty nastavené predvolene. Boli totožné s hodnotami serva, preto ho nebolo nutné programovať. V prípade AC motora bol interval prípustných hodnôt príslušného ESC väčší. Nulové otáčky dosahoval pri 150 krokoch, plné spätné pri 50 krokoch a plné dopredné s hodnotou 250 krokov. Ako vidíme, interval ESC u AC motora je o 100 krokov väčší ako u ESC s DC motorom. Ak chceme používať oba regulátory ESC s rovnakými prípustnými hodnotami ako majú servá, potrebujeme ESC AC motora preprogramovať.

Postup programovania spočíva v zasielaní pulzov s požadovanou šírkou do ESC. Ubezpečme sa, že SB nie je aktívny s parametrom *idle-timeout* (pri automatickom aktivovaní SB je tento parameter nastavený), inak dôjde k predčasnému ukončeniu opakovania pulzov. Keď sme programovanie ukončili, prejdeme k inicializácii. Až po nej bude možné riadiť otáčky motora. Inicializáciu ESC vykonávame nastavením základných polôh motora/serva. Presný postup nastavenia požadovaných polôh a času, počas ktorého majú v nich zotrvať, nájdeme v dokumentácii zariadení. Väčšinou postačuje na určitú dobu nastaviť neutrálnu polohu, potom minimálnu

[5] http://en.wikipedia.org/wiki/MOSFET

a maximálnu polohu. Úspešná inicializácia býva sprevádzaná zvukovým, respektíve svetelným signálom.

3.4 Zhrnutie

ServoBlaster nám veľmi zjednodušil prácu v tom, že sme nepotrebovali obstarávať žiadny dodatočný hardvér. Umožnil nám prestaviť všetky dostupné GPIO piny RPi na výstupy a zasielať nimi riadiace PWM signály servu a ESC. Vysvetlili sme si princíp riadenia pomocou PWM, skompilovali používateľský modul SB, prepojili vodiče serva a ESC s RPi a poukázali na rôzne úskalia, s ktorými sa môžeme pri riadení ESC stretnúť. ServoBlaster sa bude pri štarte OS aktivovať automaticky s parametrom *idle-timeout* nastaveným na 200 ms. Tým zamedzíme potenciálnym kolíziám pri strate kontroly nad riadením. Skôr než začneme s tvorbou aplikácie využívajúcej SB na riadenie modelu, je potrebné nájsť vhodný spôsob na streamovanie obrazu z kamerového modulu. Stream bude nevyhnutnou súčasťou aplikácie a umožní riadiť model v režime FPV.

4 Metódy streamovania obrazu

V nasledovnej časti práce sa zaoberám možnosťami bezdrôtového, kontinuálneho prenosu obrazu s nízkou latenciou. Zdrojom obrazu je RPi kamerový modul a príjemcom (klientom) PC s OS Windows. Cieľom je nájsť taký spôsob streamovania a takú konfiguráciu, vďaka ktorým bude možné ovládať RC model v reálnom čase a s čo najlepšou kvalitou a rozlíšením obrazu. Najvyššiu prioritu pritom bude mať požiadavka minimálnej odozvy a implementovateľnosti v ďalšej kapitole vytváranej aplikácie. Odozvu zistíme zo snímky obrazovky, na ktorej sa z polovice bude nachádzať aplikácia stopiek a z druhej polovice okno so streamom, pričom RPi kameru nasmerujeme k oknu stopiek.

4.1 Raspivid v spolupráci s Netcat a MPlayer

Prvý spôsob streamovania videa z RPi zamýšľa použitie programov Raspivid, Netcat a MPlayer. Raspivid je hlavným programom kamerového modulu pre nahrávanie videa. Detailné informácie o dostupných parametroch aplikácie je možné

nájsť v oficiálnej dokumentácii[1]. Netcat[2] je pomocný počítačový program na čítanie a zapisovanie údajov cez sieťové pripojenia používajúce TCP/IP protokol. Tento nástroj môže byť použitý samostatne alebo v spolupráci s inými programami. My ním budeme odosielať obraz z aplikácie *raspivid* a na strane klienta ho naopak čítať – v súčinnosti s multimediálnym prehrávačom MPlayer[3].

Pred samotným štartom streamovania je potrebné u klienta nastaviť načúvanie. Načúvanie na danom porte aktivujeme cez príkazový riadok (ďalej CMD) príkazom:

```
nc\nc.exe -u -L -p 5001 | mp\mplayer.exe -framedrop -nosound -fps 60 -
demuxer h264es -
```

Predpokladáme pritom, že priečinky oboch programov sú umiestnené do súčasného pracovného adresára. Znakom „|“ sme zreťazili oba procesy. To znamená, že výstup prvého procesu sa použije ako vstup druhého. Význam parametrov ozrejmuje Tabuľka 2 v prílohe A. Následne na RPi aktivujeme streamovanie príkazom:

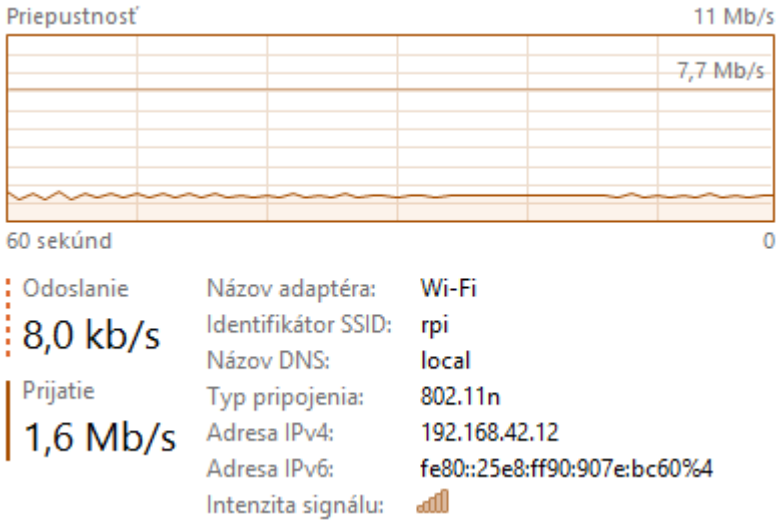
```
raspivid -n -t 0 -w 1280 -h 720 -fps 30 -vf -hf -b 1500000 -o - | nc -u
192.168.42.12 5001
```

Výstup z *raspivid* je vstupom pre Netcat (*nc*), ktorý ďalej odosiela stream na IP adresu klienta (192.168.42.12) cez port 5001. Opis parametrov – pozri Tabuľka 3, príloha A. Po aktivovaní streamovania sa na obrazovke klienta otvorí nové okno s prijímaným videom. Vyťaženie siete (Obr. 4.1) preukázalo stabilný príjem obrazu bez výkyvov a latencia prenosu bola približne 160 ms.

[1] http://www.raspberrypi.org/documentation/raspbian/applications/camera.md

[2] http://netcat.sourceforge.net

[3] http://www.mplayerhq.hu



Obr. 4.1: Vyťaženosť WiFi siete (Raspivid + Netcat + MPlayer)

4.2 Raspivid v spolupráci s GStreamer

GStreamer (ďalej GST) je knižnica určená na prácu s mediálnym obsahom, počnúc prehrávaním Ogg/Vorbis súborov, streamovaním, až po komplexné spracovanie audia a videa [11]. Podobne ako v predošlom prípade, aj teraz využijeme na záznam obrazu program *raspivid*, ktorý zreťazíme s knižnicou GST.

Začneme inštaláciou GST na RPi. Najprv editujeme repozitár:

```
sudo nano /etc/apt/sources.list
```

Na koniec súboru pridajme nový zdroj:

```
deb http://vontaene.de/raspbian-updates/ . main
```

Po uložení aktualizujme RPi a nainštalujme GST:

```
sudo apt-get update
sudo apt-get install gstreamer1.0-tools gstreamer1.0-plugins-base gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly -y
```

Ďalej zo stránky GST [4] skopírujeme inštalátor pre Windows. V sprievodcovi zvoľme plnú inštaláciu a cieľový adresár, napríklad *C:\gstreamer*. Ak použijeme rovnaké parametre pre *raspivid*, aké sme použili v predošlej kapitole a výstup zreťazíme s GST, príkaz na aktiváciu streamovania bude mať tvar:

```
raspivid -n -t 0 -w 1280 -h 720 -fps 30 -vf -hf -b 1500000 -o - | gst-launch-1.0 -e -vvvv fdsrc ! h264parse ! rtph264pay pt=96 config-interval=5 ! udpsink host=192.168.42.12 port=5001
```

[4] http://gstreamer.freedesktop.org/data/pkg/windows/1.3.91/gstreamer-1.0-x86_64-1.3.91.msi

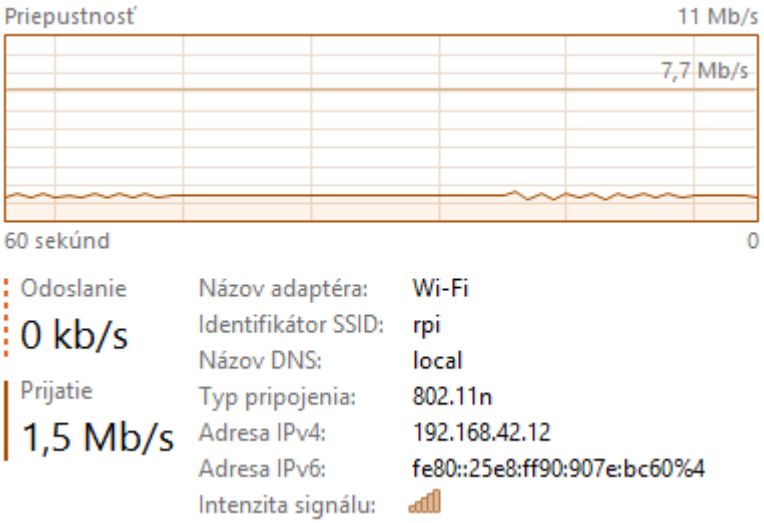
Význam použitých parametrov pre GST je možné dohľadať na webovej stránke projektu. Prehrávanie streamu aktivujeme u klienta v príkazovom riadku. Súčasným pracovným adresárom musí byť priečinok s vykonateľnými súbormi. Pre vstup doň zadajme:

```
C:
cd gstreamer\1.0\x86_64\bin
```

Príjem obrazu zahájime príkazom:

```
gst-launch-1.0 -e -v udpsrc port=5001 ! application/x-rtp, payload=96 !
rtpjitterbuffer ! rtph264depay ! avdec_h264 ! fpsdisplaysink sync=false text-
overlay=false
```

Vzápätí na obrazovke klienta uvidíme nové okno s prijímaným videom. Vyťaženie siete (Obr. 4.2) preukázalo rovnocenný príjem streamu ako v predošlom prípade a odozva činila 110 ms.



Obr. 4.2: Vyťaženosť WiFi siete (Raspivid + GStreamer)

4.3 Streamovací server HTTP pre UV4L

Jednoduchý framework UV4L poskytuje ovládače tretej strany pre video zariadenia kompatibilné s oficiálnym V4L2 [5]. Streamovací server je prídavným modulom (anglicky plugin) pre UV4L. Umožňuje streamovať video v reálnom čase prostredníctvom HTTP protokolu. Konfigurácia parametrov a zobrazenie streamu sa realizuje v prostredí internetového prehliadača. Streamovateľné formáty v prehliadači sú MJPEG, JPEG a H264.

Inštaláciu UV4L pre OS Raspbian zahájime nasledovnými príkazmi [12]:

[5] http://en.wikipedia.org/wiki/Video4Linux

```
wget http://www.linux-projects.org/listing/uv4l_repo/lrkey.asc && sudo apt-key add ./lrkey.asc
```

Do súboru */etc/apt/sources.list* pridáme riadok:

```
deb http://www.linux-projects.org/listing/uv4l_repo/raspbian/ wheezy main
```

Ďalej aktualizujeme OS a nainštalujeme UV4L spolu s prídavným modulom ovládača kamery:

```
sudo apt-get update
sudo apt-get install uv4l uv4l-raspicam
```

Na záver nainštalujeme voliteľné moduly:

```
sudo apt-get install uv4l-server
sudo apt-get install uv4l-uvc
sudo apt-get install uv4l-xscreen
sudo apt-get install uv4l-mjpegstream
```

Po ukončení inštalácie reštartujeme RPi (*sudo reboot*).

Server HTTP aktivujeme príkazom:

```
uv4l --auto-video_nr --driver raspicam --server-option '--port=9000'
```

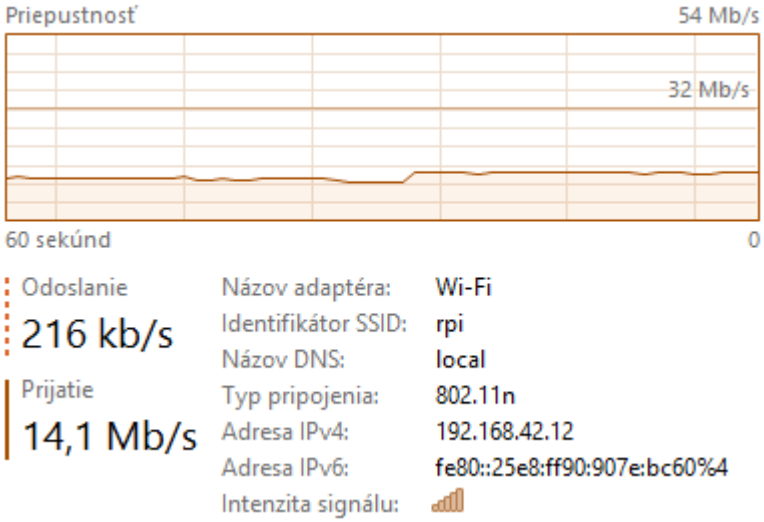
Ak chceme aktivovať server pri štarte OS, pridajme uvedený riadok do súboru */etc/rc.local*. Proces ovládača môžeme kedykoľvek ukončiť príkazom *pkill uv4l*. Vysvetlenie parametrov príkazu:

Tab. 4.1: UV4L HTTP server – inicializačné parametre

--auto-video_nr	Ak je špecifikovaný, použije sa prvé dostupné video zariadenie. V opačnom prípade sa automaticky predpokladá zariadenie s indexom 0.
--driver	Zoznam ovládačov kamery, ktoré sa majú použiť.
--server-option	Parameter servera. Môže byť opakovaný viackrát za sebou, ak chceme vložiť viac nastavení.
--port=9000	Port HTTP servera.

Detailnejšie nastavenia servera budeme konfigurovať cez webové rozhranie. To je v prípade zvoleného portu (9000) a WiFi adaptéra v režime AP dostupné na adrese *http://192.168.42.1:9000*. Konfiguráciu streamovania nájdeme v časti *control panel*. Následne v časti *Resolution & Format* zvolíme rozlíšenie 1280 × 720 px a formát MJPEG. V časti *Control settings* aktivujeme *horizontal mirror* a *vertical mirror*. Ostatné prepínacie tlačidlá deaktivujeme. Parameter *jpeg quality* nastavíme na hodnotu 10. Zmeny aplikujeme tlačidlom *Apply*.

Stream zobrazíme na adrese *http://192.168.42.1:9000/stream*. Vyťaženie siete silne závisí od nastavenej kvality obrazu, čo sa prejavilo aj na dátovom toku (Obr. 4.3). Odozva dosahovala zhruba 320 ms.



Obr. 4.3: Vyťaženosť WiFi siete (UV4L + HTTP server)

4.4 Zhrnutie

Preskúmali sme tri spôsoby streamovania videa. V prvých dvoch bolo základom použitie programu *raspivid*, ktorý je súčasťou distribúcie, v kombinácii s aplikáciami Netcat, MPlayer a frameworkom GStreamer. V treťom spôsobe sme použili neoficiálne ovládače kamery UV4L s prídavným modulom HTTP servera, umožňujúcom konfiguráciu a prehrávanie streamu prostredníctvom webového prehliadača. Vo všetkých testoch bolo použité rovnaké rozlíšenie (1280 × 720 px) a rovnaký režim WiFi adaptéra (AP). Najlepšia odozva bola dosiahnutá druhým uvedeným spôsobom – kombináciou *raspivid* a GStreamer (110 ms). Hoci sú prvé dva uvedené spôsoby vyhovujúcejšie z hľadiska odozvy, rozhodol som sa pre tvorbu aplikácie (spomínanej v závere predošlej kapitoly) zvoliť HTTP server. Argumentom pre jeho výber je fakt, že knižnica OpenCV (pozri ďalšiu kapitolu) dokáže predvolene pracovať rovnako s MJPEG, ako aj s H264 streamom.

5 Aplikácia autonómneho riadenia s využitím PC videnia

V záverečnej časti práce sa zaoberám samostatným riadením RC modelu, nezávislým od riadenia človekom. To počíta so streamovaním videa z pohľadu prvej osoby a následnou identifikáciou objektov, reguláciou rýchlosti a smeru jazdy. Vývojovým prostredím pre tvorbu aplikácie autonómneho riadenia je *Microsoft Visual Studio Express 2013 for Windows Desktop* (ďalej VS). Na rozpoznávanie obrazu bude slúžiť knižnica OpenCV. Do vývojového prostredia VS okrem OpenCV integrujeme tiež knižnicu libcurl, ktorú neskôr využijeme ako jednu z možností zasielania riadiacich príkazov do RPi. Ďalej sa budeme venovať zasielaniu príkazov technológiami AJAX a Web Sockets.

5.1 Knižnice OpenCV, cURL a libcurl

Knižnica OpenCV je slobodná a otvorená multiplatformová knižnica pre manipuláciu s obrazom. Je zameraná predovšetkým na počítačové videnie a spracovanie obrazu v reálnom čase. Pôvodne ju vyvíjala spoločnosť Intel. Jej beh je možné urýchliť v spolupráci s knižnicou Integrated Performance Primitives (Intel IPP). Knižnicu je možné využívať z prostredia jazykov C, C++ a s generátorom rozhrania SWIG, tiež z prostredí jazykov Python a Octave [13].

Softvérový projekt cURL poskytuje knižnicu a nástroj príkazového riadka na prenos údajov cez rozličné protokoly. Projekt zahŕňa dva produkty – cURL a libcurl. Prvýkrát bol projekt vydaný v roku 1997. Projekt cURL sa používa v príkazovom riadku alebo v skriptoch na prenos dát. Používa sa tiež v autách, televíznych boxoch, smerovačoch, audio vybavení, mobilných telefónoch, tabletoch, settop boxoch, mediálnych prehrávačoch a je chrbticou internetového prenosu pre tisíce softvérových aplikácií ovplyvňujúcich viac ako miliardu používateľov [14][16].

Jednoduchá, bezplatná knižnica Libcurl slúži na prenos údajov a pracuje s URL. Táto knižnica podporuje množstvo protokolov, SSL certifikáty, nahrávanie cez HTTP POST, HTTP PUT, FTP, odosielanie formulárov cez HTTP, proxy servery, cookies, autentifikáciu s menom a heslom, údaje o prenose súboru HTTP, či proxy tunelovanie. Knižnicu je možné zostaviť a pracovať s ňou identicky na veľkom počte platforiem vrátane Solaris, NetBSD, FreeBSD, OpenBSD, Darwin, HPUX, IRIX,

AIX, Tru64, Linux, UnixWare, HURD, Windows, Amiga, OS/2, BeOs, Mac OS X, Ultrix, QNX, OpenVMS, RISC OS, Novell NetWare, DOS a iných [15][16].

Knižnice OpenCV a libcurl integrujeme do VS kliknutím pravým tlačidlom myši na aktuálny projekt a po výbere položky *Vlastnosti* úpravíme parametre podľa – pozri Tabuľka 4, príloha A. Všimnime si hodnoty a konfigurácie pre *Linker – Vstup – Ďalšie závislosti*. Knižnicu *libcurl.lib* zadávame pre obe konfigurácie (*Debug* a *Release*) rovnakú a knižnice pre OpenCV zvlášť pre každú konfiguráciu. Knižnice OpenCV pre konfiguráciu *Debug* majú na konci názvu príponu *d*. Ak sme postupovali správne, uvidíme v režime všetkých konfigurácií kombinovaný zápis závislostí v tvare *libcurl.lib;<iné možnosti>*.

Uvedený postup platí pre *VS Express 2013 for Desktop* (verzia 12.0), OpenCV verzie 2.4.9, cURL a libcurl verzie 7.38.0 pre 32 bitový Windows. Knižnicu OpenCV sme inštalovali do *C:\opencv*, cURL a libcurl do *C:\curl*. V premenných prostrediach vo vlastnostiach systému (WIN+R, *SystemPropertiesAdvanced*) sme do systémovej premennej *Path* pridali cestu:

```
C:\opencv\build\x86\vc12\bin;
```

Tým sa zaručí správna funkčnosť OpenCV vo VS. Knižnicu libcurl budeme kvôli prenositeľnosti kopírovať priamo do adresára projektu so skompilovanými súbormi – viac v kapitole 5.3.

5.2 Režimy aplikácie a spôsob implementácie

Naša aplikácia bude podporovať štyri hlavné režimy:

- manuálne riadenie,
- sledovanie lopty,
- sledovanie čiary,
- rozpoznávanie symbolov.

Manuálne riadenie umožňuje ovládať RC model z PC pomocou myši a klávesnice. Horizontálnym pohybom myši meníme smer a vertikálnym pohybom rýchlosť jazdy. Neutrálom považujeme strednú polohu kurzora vzhľadom k obom osiam. Aktivácia motora je podmienená bezpečnostnou poistkou – držaním klávesy CTRL, W alebo ľavého tlačidla myši počas regulácie rýchlosti.

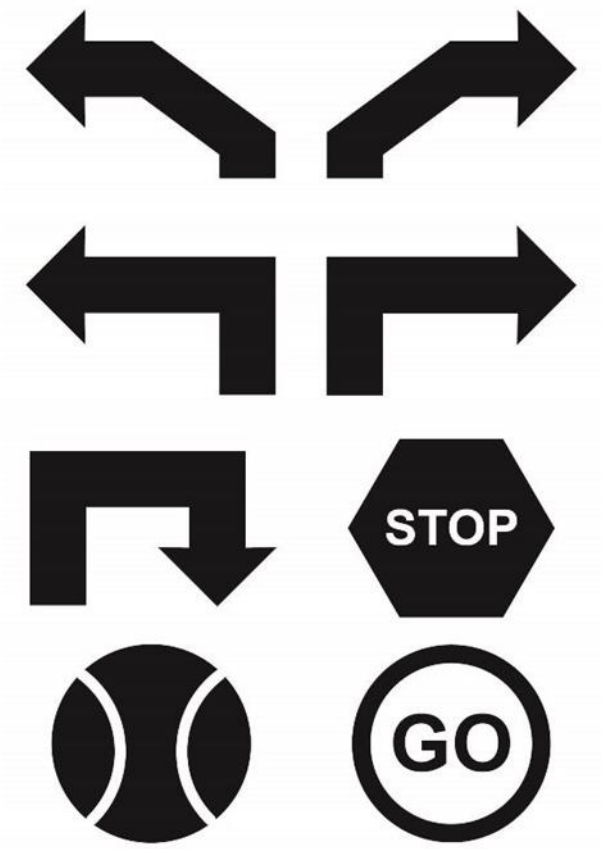
Sledovanie lopty, čiary a rozpoznávanie symbolov využíva knižnicu OpenCV a pracuje na princípe prahovania farby (anglicky thresholding). U sledovania lopty na scénu umiestnime homogénny objekt – v našom prípade ním bude červená fitlopta. Potom pomocou metódy prahovania odfiltrujeme farbu lopty. Vznikne binárny (čiernobiely) obraz, ktorý bude obsahovať biele body (pixely) na tých miestach, kde sa nachádza lopta. Polohu lopty určíme funkciou hľadania súvislých plôch z binárneho obrázka. Auto rozpozná loptu iba do určitej vzdialenosti. Ak je táto vzdialenosť väčšia ako stanovená v konfiguračnom súbore, nasledovanie sa pozastaví. Rýchlosť modelu narastá so zväčšovaním vzdialenosti od lopty a klesá pri jej približovaní. Ak je lopta veľmi blízko, model začne cúvať.

V režime sledovania čiary je objektom prahovania čiara bielej farby. Na rozdiel od predošlého režimu nám stačí sledovať iba výrez obrazu – takzvanú oblasť záujmu (anglicky region of interest; skratka ROI), a v ňom zmenu horizontálnej pozície čiary. Rýchlosť motora nastavujeme na konštantnú, mení sa len pozícia serva, teda zmena smeru jazdy.

Pri rozpoznávaní symbolov sa hľadajú v obraze vopred definované tvary. Najskôr sa do pamäte načíta zoznam podporovaných symbolov. Symboly sú čierne vzory na bielom podklade, s čiernym orámovaním v tvare obdĺžnika. Obr. 5.1 znázorňuje všetkých osem podporovaných vzorov (bez orámovania): zatočenie vľavo/vpravo o 45°, zatočenie vľavo/vpravo o 90°, otočenie o 180°, zastavenie, hľadanie lopty a jazda vpred. Rozpoznávanie začína nájdením orámovania – hľadajú sa objekty v tvare štvoruholníka. Potom sa porovnáva vzor v obdĺžniku so symbolmi načítanými v pamäti. Porovnávanie je založené na funkcii bitového exkluzívneho súčtu[1] (XOR) medzi vzorovým a porovnávaným symbolom. Prekrývajúcim bodom s rovnakou hodnotou sa priradí 0 (čierna), ostatným 255 (biela). Výstupom funkcie je binárna matica (ďalej diferenčná). Čím je v diferenčnej matici väčší počet nulových bodov, tým väčšia je pravdepodobnosť identifikácie hľadaného symbolu. Na základe rozpoznaného symbolu sa rozhodne, akú akciu má model vykonať. Metóda rozpoznávania pomocou funkcie XOR bola zvolená najmä kvôli jej rýchlosti. Použitie

[1] http://cs.wikipedia.org/wiki/Bitový_operátor

iných známych metód ako SURF[2] a SIFT[3] nie je z hľadiska zvýšených výpočtových nárokov pre RPi vhodné.



Obr. 5.1: Rozpoznávanie symbolov – podporované vzory [17]

Aplikáciu implementujeme dvoma spôsobmi – v prvom ako aplikáciu príkazového riadka, v druhom ako webovú stránku. Aplikáciu príkazového riadka rozdeľujeme na dve verzie – offboard a onboard. Offboard verzia deleguje rozpoznávanie obrazu na CPU klienta, kým onboard verzia spracúva stream priamo na RPi. Implementácia prostredníctvom webovej stránky podporuje z vyššie uvedených režimov iba prvý z nich – manuálne ovládanie. Obsahuje však doplnkovú funkciu – memorizáciu trasy. Zapamätávanie pracuje na princípe uchovávania aktuálnych hodnôt riadiacich príkazov v pravidelných časových intervaloch. To nám umožní spätne zrekonštruovať celú trasu jazdy.

Konzolová verzia aplikácie bude na zasielanie riadiacich príkazov do RPi využívať HTTP požiadavky prostredníctvom knižnice *libcurl* alebo TCP/IP sokety na báze

[2] http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html

[3] http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html

modelu klient-server. V prípade webovej stránky (HTML verzie) používame webové sokety[4] (WS) alebo menej efektívne AJAX[5] požiadavky. Ukážeme si, že RC model je možné ovládať aj cez internet, a to pomocou knižnice PeerJS jazyka JavaScript.

5.3 Offboard verzia

Prvým spôsobom implementácie, ktorý si opíšeme, bude offboard verzia. Táto verzia je kľúčovou a poslúži nám na prezentáciu všetkých podporovaných režimov aplikácie uvedených vyššie. Offboard verzia beží na PC klienta. Klient vykonáva analýzu streamu z RPi a odosiela riadiace príkazy pre RC model. Offboard aplikácia je interaktívna, zachytáva stlačenia klávesov a v režime manuálneho riadenia reaguje na pohyb myši.

Kód aplikácie spolu s konfiguráciou tvoria tri súbory:

- Source.cpp – hlavný kód s metódou *main*,
- fcie.h – implementácia podporovaných režimov a zahrnutia hlavičiek,
- settings.txt – konfiguračný súbor.

V adresári skompilovaného programu sa celkovo nachádzajú tieto súbory: arrowB.jpg, arrowGo.jpg, arrowL.jpg, arrowL45.jpg, arrowR.jpg, arrowR45.jpg, arrowStop.jpg, arrowT.jpg, libcurl.dll, libeay32.dll, libssh2.dll, msvcr120.dll, rpi.exe, settings.txt, ssleay32.dll a zlib1.dll. Súbory s príponou *jpg* predstavujú podporované symboly, súbory s príponou *dll* patria ku knižnici libcurl a aplikácia *rpi.exe* je spúšťacím súborom offboard verzie. Spôsob použitia aplikácie pri jej štarte:

```
usage:  rpi.exe [<rpi_ip> <h264|mjpeg>]
```

Ak nezadáme žiadne argumenty, predvolenou IP adresou RPi (argument *rpi_ip*) bude IP adresa RPi v režime AP (192.168.42.1) a predvoleným formátom video streamu (argument *h264* alebo *mjpeg*) bude MJPEG. Povolený počet argumentov je 0 alebo práve 2 (IP adresa a formát streamu). Stream konfigurujeme podľa kapitoly 4.3.

5.3.1 Grafické rozhranie aplikácie v prostredí OS Windows

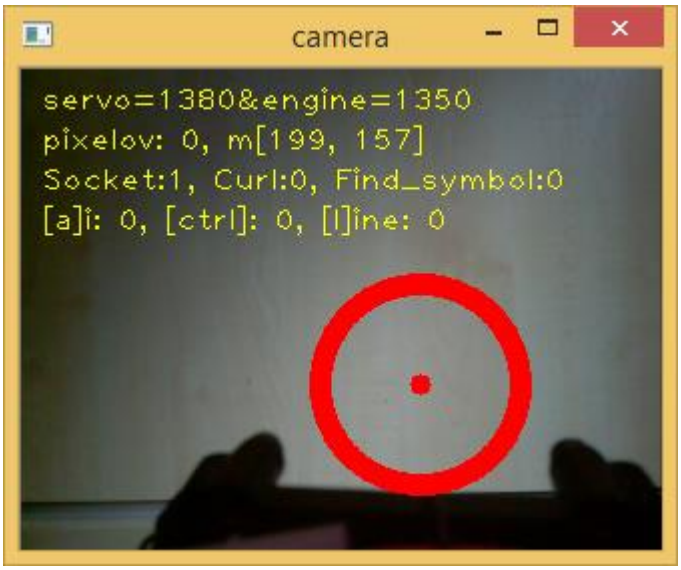
Nami vytvorená aplikácia sa skladá z niekoľkých okien, využívaných v závislosti od aktívneho režimu. Hlavným oknom je okno streamu, ktoré obsahuje okrem

[4] http://en.wikipedia.org/wiki/WebSocket

[5] http://en.wikipedia.org/wiki/Ajax_(programming)

prijímaného obrazu aj HUD (anglicky head-up display) s informáciami o pozícii kurzora, aktívnom režime či hodnotách zasielaných parametrov. Zoznam ovládacích kláves:

- S – komunikácia cez sokety,
- C – komunikácia cez knižnicu libcurl,
- F – režim rozpoznávania symbolov,
- A – režim sledovania lopty,
- L – režim sledovania čiary,
- CTRL – aktivácia motora (v režime manuálneho riadenia).

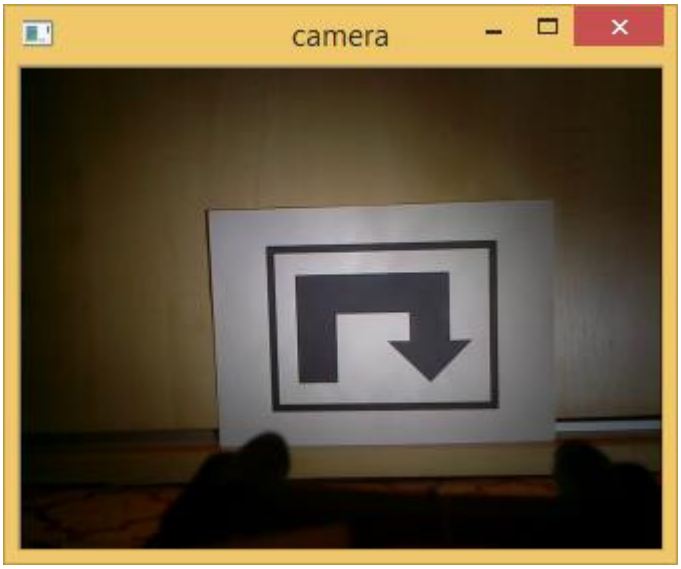


Obr. 5.2: Offboard verzia – okno so streamom (režim manuálneho riadenia)

V režime manuálneho riadenia (Obr. 5.2) okno reaguje aj na pohyb myši. Poloha kurzora je vyznačená červeným kruhom. Vodorovným pohybom kurzora sa reguluje zatáčanie serva, zvislým pohybom smer otáčania a rýchlosť motora. Neutrálom rozumieme polohu kurzora v strede okna.

Ďalšie okná slúžia na zobrazovanie prahovaného obrazu, hrán objektov a diferenčnej matice. S oknom prahovania (Obr. 5.4) pracujeme vo všetkých režimoch s výnimkou režimu manuálneho riadenia. Okno hrán (Obr. 5.5) a diferenčnej matice (Obr. 5.6) používame v režime rozpoznávania symbolov, na ktorom si aj ukážeme ich využitie.

Nasledovný obrázok zachytáva okamih rozpoznania symbolu (otočenie o 180°):

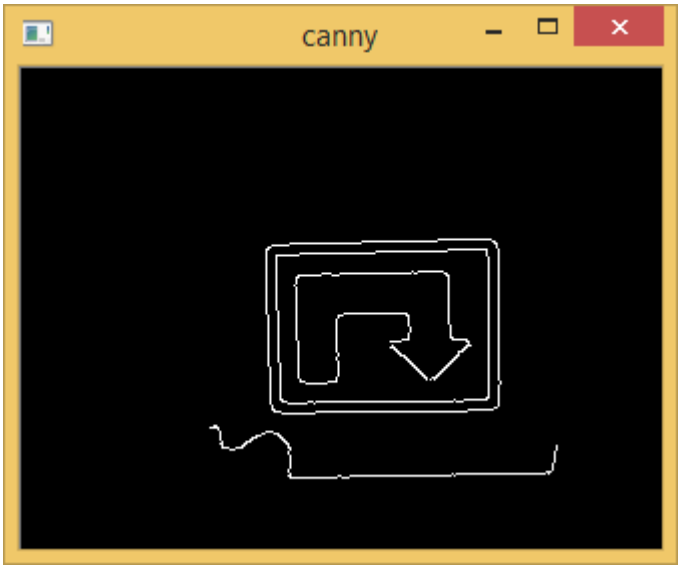


Obr. 5.3: Offboard verzia – rozpoznanie symbolu

Displej HUD je skrytý a riadenie programu preberá metóda *drive* (pozri kapitolu 5.3.6). Na prahovanom obraze môžeme vidieť tvar rozpoznávaného symbolu:

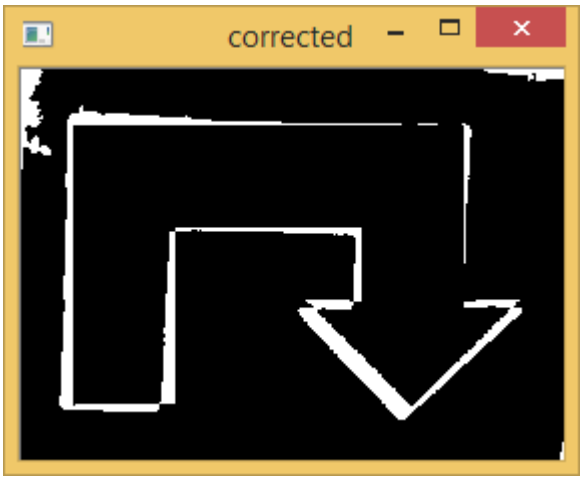
Obr. 5.4: Offboard verzia – prahovanie obrazu

Hranice symbolu a orámovania zobrazujeme v okne *canny*:



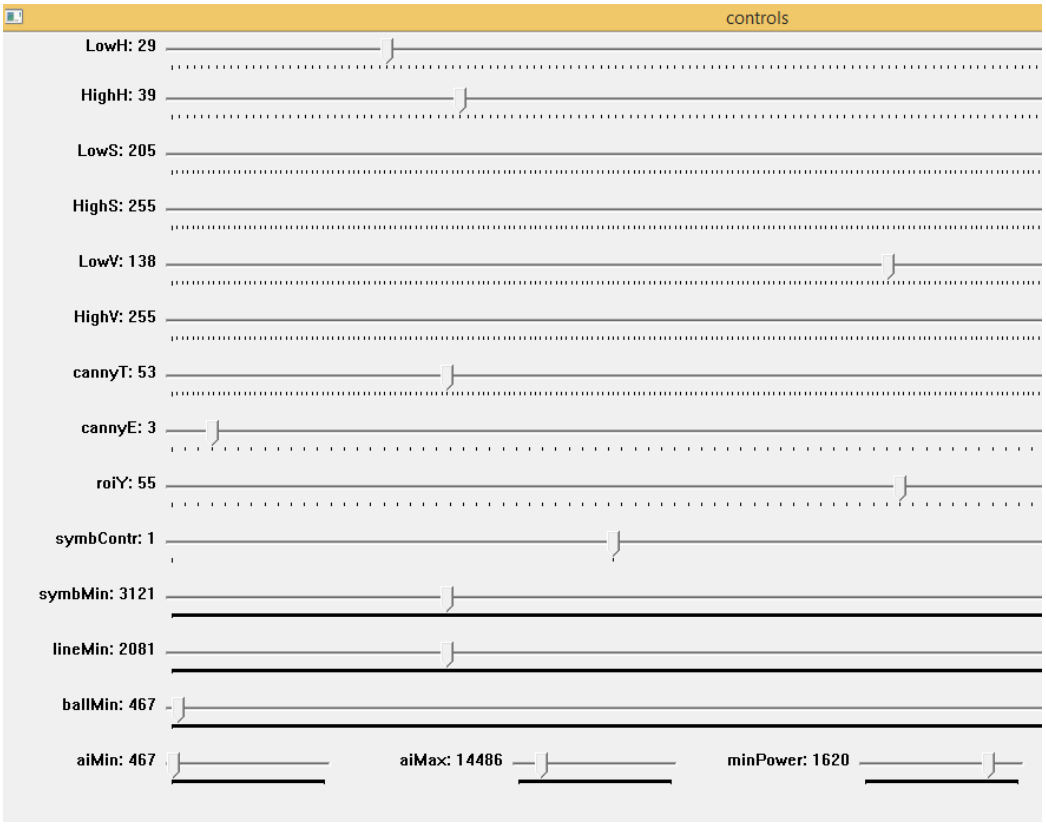
Obr. 5.5: Offboard verzia – detekcia hrán

Detekcia hrán sa vyžaduje na identifikáciu geometrických tvarov a stanovenie polohy vrcholov orámovania pri korekcii perspektívy. Diferenčná matica znázorňuje úroveň prekrytia vzorového a porovnávaného symbolu:



Obr. 5.6: Offboard verzia – diferenčná matica

Parametre programu konfigurujeme v okne *controls* (Obr. 5.7). Predvolené hodnoty parametrov sa pri štarte načítavajú z konfiguračného súboru *settings.txt*.



Obr. 5.7: Offboard verzia – konfigurácia parametrov

Význam konfiguračných parametrov objasňuje Tabuľka 5 v prílohe A.

5.3.2 Analýza hlavného algoritmu

V tejto kapitole si rozoberieme hlavnú metódu *main* zo súboru *Source.cpp* – pozri príloha B. Hlavný algoritmus začína prípravou a vytvorením objektu *cap* so streamovaným videom. Ak sa program štartuje s argumentom *h264*, stream bude komprimovaný vo formáte H.264. Atribúty streamu nastavujeme v tvare *cap.set(atribút, hodnota)*. Po úspešnom otvorení streamu sa do pamäte načítajú podporované vzory pre režim rozpoznávania symbolov a na obrazovku sa vypíše aktuálne rozlíšenie prijímaného videa získané cez *cap.get(atribút)*. Ďalej sa vytvoria komunikačné objekty – *curl* pre libcurl a *ConnectSocket* pre sokety. Dokumentácia k libcurl API je dostupná online[6]. Implementácia metódy *vytvor_soket* sa nachádza v súbore *fcie.h*. Ide o klientsku časť kódu[7]. Serverová časť[8] – súbor *myServer.c*, beží na RPi.

[6] http://curl.haxx.se/libcurl/c

[7] http://msdn.microsoft.com/en-us/library/windows/desktop/ms737591%28v=vs.85%29.aspx

Okná vytvárame metódou *namedWindow(názov, príznak)*. Príznak *WINDOW_NORMAL* umožňuje meniť rozmery okna, u *CV_WINDOW_AUTOSIZE* sa prispôsobí veľkosť automaticky, bez možnosti meniť rozmery. Konfiguračné parametre načítavame zo súboru *settings.txt* po riadkoch funkciou *fgets*. Riadky rozdeľujeme na slová funkciou *strtok*. Slová reprezentujúce hodnotu parametrov sa ukladajú do kontajnera typu *vector*. Po naplnení kontajnera inicializujeme premenné parametrov a v okne *controls* vytvoríme grafické bežce na zmenu ich hodnôt v tvare:

```
createTrackbar(názov_parametra, názov_okna, adresa_parametra, max_hodnota);
```

V cykle *for* prebieha načítavanie snímok zo streamu – *cap.read()*, výpočet riadiacich príkazov (pozri nasledovné kapitoly) a ich odosielanie cez libcurl alebo sokety. Riadiace príkazy odosielame z metódy *main,* iba ak nie je aktivovaný režim rozpoznávania symbolov. V ňom sú riadiace príkazy zasielané priamo z metódy *drive* (súbor *fcie.h*) – pozri koniec kapitoly 5.3.6.

Asynchrónne stlačenie kláves/tlačidiel zisťujeme funkciou *GetAsyncKeyState*, ktorej parametrom je číselný kód klávesy. Pre kláves CTRL použijeme konštantu *VK_CONTROL* (kód 0x11) a pre ľavé tlačidlo myši konštantu *VK_LBUTTON* (kód 0x01). V prípade, že nie je stlačené nič z uvedených kláves/tlačidiel a zároveň nie je aktívny režim sledovania lopty, aktivuje sa neutrál, to znamená nastavenie riadiaceho príkazu motora na hodnotu 1500 µs.

Ak sme aktivovali odosielanie príkazov cez libcurl, hodnoty sa odošlú metódou POST v tvare *servo=[hodnota]&engine=[hodnota]* na spracovanie skriptom *rpi.php*, na ktorý sa odkazujeme vyššie pri inicializácii objektu *curl*. Skriptu *rpi.php* sa venujeme v kapitole 5.5.2. K odoslaniu hodnôt dochádza v každej tretej iterácii cyklu *for* (riadiaca premenná *i*), čím zamedzíme preťaženiu PHP servera.

Tvar hodnôt pri komunikácii cez sokety je podobný: *7=[hodnota]us,1=[hodnota]us* (7 – servo, 1 – motor; viac v kapitole 3.2). Hodnoty sú odosielané aplikácii *myServer*, tvoriacej serverovú časť (pozri vyššie). Na ich extrakciu použijeme opäť funkciu *strtok*. Oddeľovačom bude čiarka. Tým získame samostatné príkazy pre riadenie serva a motora, ktoré zapíšeme postupne do deskriptora SB.

[8] http://cs.smith.edu/dftwiki/index.php/Tutorial:_Client/Server_on_the_Raspberry_Pi#Server_code

Po odoslaní riadiacich príkazov, vykreslení HUD a okna streamu (funkcia *imshow*) nasleduje funkcia *waitKey* na (synchrónne) zachytávanie stlačenia kláves, ktorej parametrom je dĺžka čakania na stlačenie klávesu (v ms). Funkcia je blokujúca, to znamená, že vykonávanie algoritmu je počas čakania na stlačenie klávesu pozastavené. Predvolenou hodnotou je 30 ms (približne 33 fps). Nižšou hodnotou vieme docieliť stabilnejšie vyťaženie siete a tým nižšiu odozvu, no zároveň dôjde aj k zvýšeniu vyťaženia CPU. Ideálna doba čakania je čo najnižšia (minimálne 1 ms), pri ktorej nie je CPU klienta plne vyťažený. Funkcia vracia číselnú hodnotu stlačeného klávesu. Na ukončenie procesu použijeme kláves ESC (číselná hodnota 27) a na pozastavenie kláves medzerník. V ostatných prípadoch sledujeme aktiváciu/deaktiváciu podporovaných režimov (*f* – rozpoznávanie symbolu, *a* – sledovanie lopty, *l* – sledovanie čiary) a spôsobu odosielania riadiacich príkazov (*c* – curl, *s* – sokety).

Na záver ukončíme spojenie so soket serverom zaslaním hodnoty „-1“ a dealokujeme pamäť objektu *curl*.

5.3.3 Režim manuálneho riadenia

Aktiváciou režimu manuálneho riadenia sa pri každom načítaní snímky z RPi kamery vykonáva iba výpočet riadiacich parametrov motora a serva:

```
engine = 1000 + round((1 - coords.y / dHeight) * 1000 / 10) * 10;
servo = 1000 + round((1 - coords.x / dWidth) * 1000 / 10) * 10;
```

Objekt *coords* uchováva polohu kurzora relatívne k streamu, premenné *dHeight* a *dWidth* sú rozmermi streamu. Model pôvodne reagoval na riadiace príkazy presne opačne, teda zatáčal vľavo namiesto vpravo a pohyboval sa dopredu namiesto dozadu. Hodnoty sme preto invertovali cez doplnok k jednotke a zároveň upravili na násobok desiatky, aby vyhovovali konfigurácii SB (pozri parameter *step-size* v kapitole 3.1). Podobným spôsobom budeme riadiace parametre odvádzať aj v ďalších kapitolách.

5.3.4 Režim sledovania lopty

V režime sledovania lopty sa zavolá funkcia *trackBall*, ktorej úlohou je zistiť plochu lopty (*area*) a súradnice jej stredu (*roi_center*):

```
trackBall(frame, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV),
              roi_center, area, ballMin);

if (area >= aiMax)
```

```
        engine = 1300;
else if (area <= aiMin)
        engine = 1500;
else {
        diff = (area - aiMin) / (aiMax - aiMin);
        engine = minPower + round((1 - diff) * (100) / 10) * 10;
}
servo = 1000 + round((1 - roi_center.x / dWidth) * 1000 / 10) * 10;
```

Ak je plocha väčšia alebo rovná parametru *aiMax*, model začne cúvať. Ak je plocha menšia alebo rovná parametru *aiMin*, model sa nebude pohybovať. V ostatných prípadoch sa rýchlosť motora určí ako súčet hodnoty parametra *minPower* a pomeru plochy lopty vzhľadom k hraniciam stanoveným parametrami *aiMin* a *aiMax*. Parameter *minPower* zaručí, že sa model bude pohybovať istou (minimálnou) rýchlosťou. Druhý sčítanec ovplyvňuje zrýchlenie. Čím ďalej bude lopta k modelu, tým bude zrýchlenie vyššie. Naopak, čím bude lopta bližšie, tým bude zrýchlenie nižšie. Návratová hodnota funkcie *trackBall* je typu void, hľadané hodnoty premenných vkladáme ako referencie (znak „&“). Kód funkcie – pozri príloha B.

V prvom kroku konvertujeme snímku streamu z farebného modelu BGR do HSV[9], ktorý je vhodnejší na segmentáciu farieb. Parametrami funkcie *cvtColor*[10] sú vstupný obrázok, výstupný obrázok a konverzný kód farebného modelu. Nový obrázok v HSV farebnom priestore sa skladá z troch vrstiev, značiacich farebný odtieň (anglicky hue), sýtosť (anglicky saturation) a jas (anglicky value). V knižnici OpenCV sú prípustné hodnoty z intervalu 0 – 179, 0 – 255 a 0 – 255, v uvedenom poradí pre každú zložku.

V druhom kroku použijeme funkciu *inRange*[11] na získanie prahovaného obrázka. Parametrami funkcie sú vstupný obrázok, dolná hranica, horná hranica a výstupný obrázok. Hranicou rozumieme pole alebo skalár hodnôt reprezentujúcich zložky HSV obrázka. Hranice vymedzujú záujmové body. Bodom ležiacim medzi hranicami sa vo výstupnom (čiernobielom) obrázku priradí hodnota 255 (biela) a ostatným bodom hodnota 0 (čierna). Prahovaný obrázok môže obsahovať diskrétne, redundantné body, ktoré môžeme vylúčiť aplikovaním morfologických operácií – eróziou (podmytím) a dilatáciou (rozšírením)[12]. Kombináciou operácií (funkcie *erode* a *dilate*) dosiahneme

[9] http://cs.wikipedia.org/wiki/HSV

[10] http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor

[11] http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#inrange

[12] http://docs.opencv.org/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

aj celistvejšie vyplnenie tvaru lopty v obrázku. Použije sa pritom rovnaký štruktúrny element[13], získaný funkciou *getStructuringElement*, definovanou parametrami tvar elementu a veľkosť elementu. Vstupným a zároveň aj výstupným obrázkom funkcií (prvé dva parametre) erózie a dilatácie je prahovaný obrázok. Tretím parametrom je štruktúrny element. Po aplikácii uvedených funkcií zobrazíme výsledný prahovaný obrázok v okne *treshold*.

V ďalšom kroku vykonávame rozpoznanie lopty. Najskôr sa funkciou *findContours*[14] z obrázka extrahujú kontúry. Parametrami funkcie sú vstupný obrázok, výstupné pole kontúr, výstupné pole hierarchie, spôsob získania kontúr, metóda aproximácie kontúr a posun kontúr. Spomedzi kontúr nás zaujímajú tie, ktoré majú viac ako 10 vrcholov a ktorých plocha je väčšia, ako minimálna plocha obrysu pre rozpoznanie lopty (parameter *min_area*). Postupne prechádzame pole kontúr a funkciou *contourArea* zisťujeme ich plochu (návratová premenná *area*). Ak je uvedená podmienka splnená, vykreslíme (funkcia *drawContours*) príslušnú kontúru (s indexom *i*) do pôvodnej snímky (premenná *frame*). Ďalej z kontúry aproximujeme elipsu (funkcia *fitEllipse*) a tú vykreslíme tiež do pôvodnej snímky. Druhú návratovú hodnotu – premennú *roi_center*, získame z atribútu *center* aproximovanej elipsy. Analýzu zvyšných obrysov ďalej nevykonávame (príkaz *break*).

Demonštračné video offboard verzie v režime sledovania lopty je dostupné na stránke *http://youtu.be/KlCeNIPWv9A*. Klient v rozlíšení 640 × 480 px dokázal spracovať približne 7 obrázkov za sekundu. Video znázorňuje všetky tri opisované akcie – cúvanie (lopta je veľmi blízko), státie (lopta je ďaleko) a pohyb vpred (plocha obrysu lopty je v stanovenom intervale). Sledovaným objektom je červená fitlopta s cirka metrovým priemerom.

5.3.5 Režim sledovania čiary

Sledovanie čiary začína výberom oblasti záujmu (objekt *roiR*). Oblasť záujmu je časť snímky, v ktorej budeme rozpoznávať čiaru. Analýza celej snímky by bola zbytočná, pretože sa zameriavame len na horizontálnu zmenu polohy čiary. Náš výrez

[13] http://en.wikipedia.org/wiki/Structuring_element

[14] http://goo.gl/ri5q0k

bude mať 100% šírku a 19% výšku snímky. Zvislá poloha začiatku výrezu (v %) je určená parametrom *roiY*. Kód vyzerá nasledovne:

```
Rect roiR(0, roiY * (dHeight / 100), dWidth, 0.19*dHeight);
lineFollow(frame, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV),
              roiR, roi_center, area, lineMin);
circle(frame, roi_center, 0, Scalar(0, 0, 255), hrubka);
line(frame, Point(roi_center.x, 0), Point(roi_center.x, frame.rows),
       Scalar::all(255), 3);

if (area < lineMin)
       engine = 1500;
else
       engine = minPower;

servo = 1000 + round((1 - roi_center.x / frame.cols) * 1000 / 10) * 10;
```

Funkcia *lineFollow* vo výreze hľadá pozíciu (*roi_center*) a plochu čiary (*area*). Pozícia čiary sa následne znázorní v pôvodnej snímke kružnicou a zvislou čiarou (funkcie *circle* a *line*). Ak je plocha obrysu čiary vo výreze menšia ako minimálna plocha obrysu pre rozpoznanie čiary (parameter *lineMin*), model sa nebude pohybovať. V opačnom prípade pôjde model rýchlosťou určenou parametrom *minPower*. Pozícia serva sa vypočíta z x súradnice stredu obrysu čiary. Kód funkcie *lineFollow* – pozri príloha B.

Funkcia *lineFollow* najskôr vytvorí výrez z pôvodného obrázka (*frame*). Potom nasleduje konverzia farebného modelu RGB do farebného modelu HSV, prahovanie, séria erózie, dilatácie a nájdenie obrysov čiary. Týmto metódam sme sa bližšie venovali v predošlej kapitole. Na výpočet polohy tvaru čiary tentokrát[15] využijeme momenty obrázka[16] (funkcia *moments*[17]). Zvislú súradnicu (*centerY*) definujeme nastálo ako polovicu výšky snímky. Horizontálnu súradnicu vypočítame z podielu prvého priestorového momentu okolo osi X (atribút *m10*) a nultého centrálneho momentu (atribút *m00*) [18].

Cyklom *for* iterujeme pole kontúr, pričom zisťujeme ich plochu (návratová premenná *area*). Ak je plocha väčšia, ako minimálna plocha obrysu pre rozpoznanie čiary (premenná *min_area*), vypočítame z momentov vodorovnú súradnicu obrysu a spolu so zvislou súradnicou ich uložíme do objektu *roi_center* (druhá návratová

[15] V režime sledovania lopty polohu kontúry (atribút *center*) určujeme funkciou *fitEllipse*.

[16] http://en.wikipedia.org/wiki/Image_moment

[17] http://goo.gl/NnAJrQ

premenná). Kontúru čiary vykreslíme do výrezu. Zvyšné obrysy ignorujeme (príkaz *break*).

Demonštračné video offboard verzie v režime sledovania čiary je dostupné na stránke *http://youtu.be/AI5VxRjfE44*. Riadenie serva je automatické. Motor sa aktivuje klávesom CTRL alebo ľavým tlačidlom myši (pozri kapitolu 5.3.2). Video znázorňuje prechod RC modelu ponad čiaru nakreslenú bielou kriedou na bežnej asfaltovej ceste.

5.3.6 Režim rozpoznávania symbolov

Posledným režimom je režim rozpoznávania symbolov. Kód algoritmu začína hľadaním symbolu (funkcia *findSymbol*). Návratovou hodnotou funkcie je index nájdeného symbolu. Ak je index nezáporný (prebehlo rozpoznanie) a symbol reprezentuje loptu (index = 3), aktivuje sa režim hľadania lopty (premenná *ai*). Pre ostatné symboly sa zavolá funkcia *drive* (pozri nižšie). Príkazom *continue* začne ďalšia iterácia nadradeného cyklu. Nastavením premennej *find_symbol* na hodnotu 0 umožníme algoritmu vo vetvení dôjsť na blok režimu rozpoznávania lopty:

```
symbol_id = findSymbol(frame, symbols, cannyT, symbContr, symbMin);
if (symbol_id >= 0) {
	if (symbol_id == 3) { //symbol lopty
		find_symbol = 0;
		ai = 1;
	}
	else {
		drive(symbol_id, ConnectSocket, cap, minPower, frame);
	}
	continue;
}
```

Prvým krokom algoritmu funkcie *findSymbol* je konverzia snímky (funkcia *cvtColor* – pozri kapitolu 5.3.4) do odtieňov šedej (anglicky grayscale). Obraz sa následne rozostrí (funkcia *GaussianBlur*[18]) a použije ako vstup pre hľadanie hrán (funkcia *Canny*[19]). Kód funkcie *findSymbol* – pozri príloha B.

Parametrami funkcie *GaussianBlur* sú: vstupný obrázok, výstupný obrázok, rozmery jadra (uvažované okolie pixelov), štandardná odchýlka v smere osi X a štandardná odchýlka v smere osi Y. Účelom funkcie je zjemniť obrázok pred aplikáciou funkcie *Canny*. Parametre funkcie *Canny*: vstupný obrázok (8 bitový

[18] http://docs.opencv.org/modules/imgproc/doc/filtering.html#gaussianblur

[19] http://docs.opencv.org/modules/imgproc/doc/feature_detection.html#canny

obrázok v formáte B&W), výstupný obrázok, dolná hranica prahovania, horná hranica prahovania a veľkosť apertúry. Vzhľad snímky po detekcii hrán zachytáva Obr. 5.5.

V upravenom obrázku sa ďalej nájdu kontúry (funkcia *findContours* – pozri kapitolu 5.3.4). Každú kontúru v iterácii cyklu aproximujeme funkciou *approxPolyDP*[20], čím sa zníži počet jej vrcholov. Parametre funkcie: vstupná kontúra (pole bodov), výstupná (aproximovaná) kontúra, presnosť aproximácie (pozri Tabuľka 5, príloha A – parameter *cannyT*) a parameter uzavretosti kontúry. Presnosť aproximácie sme definovali ako 5 % z obvodu kontúry (funkcia *arcLength*[21]). Parameter uzavretosti nastavený na hodnotu *true* znamená, že prvý a posledný vrchol výstupnej kontúry budú spojené [19].

Spomedzi kontúr nás budú zaujímať práve tie kontúry, ktoré majú štyri vrcholy a plochu (funkcia *contourArea* – pozri kapitolu 5.3.4) väčšiu, ako je minimálna plocha obrysu pre rozpoznanie symbolu (premenná *contour_area*). Po splnení oboch podmienok môže začať analýza symbolu. Najskôr je potrebné upraviť perspektívu obrazu tak, aby symbol nebol nijak pootočený alebo skosený. Z aproximovaného obrysu určíme pozíciu vrcholov. Pomocou priestorových momentov (funkcia *moments* – pozri kapitolu 5.3.5) určíme stred (objekt *center*) pôvodnej kontúry (neaproximovanej). Stred nám poslúži pri stanovení poradia vrcholov (funkcia *sortCorners*). Správne poradie vrcholov je rozhodujúce pre úpravu perspektívy. Ak je pozícia vrcholu vľavo hore od stredu kontúry, je zrejmé, že ide o ľavý horný roh. Poradie ostatných vrcholov sa zistí analogicky. Definíciu funkcie *sortCorners* je možné nájsť v súbore *fcie.h*.

Pred samotnou úpravou perspektívy (funkcia *warpPerspective*[22]) potrebujeme vypočítať (funkcia *getPerspectiveTransform*[23]) transformačnú maticu (premenná *transmtx*). Parametrami funkcie *getPerspectiveTransform* sú súradnice vrcholov kontúry rozpoznávaného symbolu (pole *corners*) a súradnice vrcholov prislúchajúceho výstupného obrázka (pole *quad_pts*). Priradenie vrcholov ilustruje nasledovný obrázok:

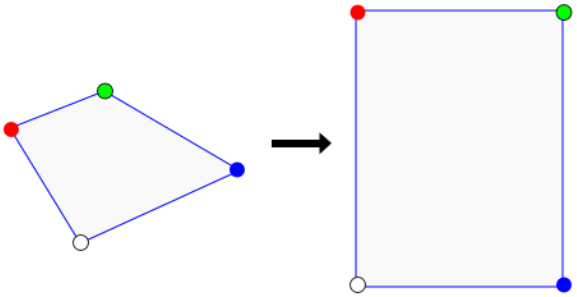
---

[20] http://goo.gl/m4o7TG

[21] http://goo.gl/kl3Znz

[22] http://goo.gl/hmDBPZ

[23] http://goo.gl/XfQW4R



Obr. 5.8: Zobrazenie prislúchajúcich vrcholov – vľavo pole *corners*, vpravo pole *quad_pts* [20]

Parametre funkcie *warpPerspective* sú: vstupný obrázok, výstupný obrázok, transformačná matica a rozmery výstupného obrázka. Výstupný obrázok (objekt *correctedImg*) má tri kanály (24 bitová farebná hĺbka) a rovnaké rozmery ako podporované vzory (pozri Obr. 5.1). Vzhľad obrázka pred a po korekcii perspektívy je pozorovateľný na Obr. 5.5 a Obr. 5.6.

Po úprave perspektívy nasleduje prahovanie, ktorým sa získa binárna snímka. Transformovaný obraz (premenná *correctedImg*) vychádza z pôvodnej snímky (premenná *frame*), preto ho najskôr potrebujeme skonvertovať do odtieňov sivej. Metódou *convertTo*[24] meníme kontrast obrázka. Parameter *kontrast* vyjadruje činiteľ zväčšenia/zmenšenia číselnej hodnoty každého pixela. Zmenou kontrastu môžeme zlepšiť rozpoznávaciu schopnosť v zhoršených svetelných podmienkach. Prahovanie vykonáme v dvoch etapách pomocou funkcie *threshold*[25]. Parametre funkcie sú: vstupný obrázok, výstupný obrázok, prahová hodnota, maximálna návratová hodnota pixelov a typ prahovania. Hodnoty pixelov sa určia podľa vzťahu:

$$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases} \quad [21]$$

Pixelom s hodnotou vyššou, ako je prahová (tresh) sa priradí maximálna návratová hodnota (maxval), ostatným hodnota nula. V prvej etape priradíme bielu farbu pixelom s hodnotou vyššou ako 140. V druhej etape bude prahovou hodnotou priemer (premenná *medVal*) najvyššej a najnižšej hodnoty bodov v obrázku. Najvyššiu a najnižšiu hodnotu bodov nájdeme funkciou *minMaxLoc*[26].

[24] http://docs.opencv.org/modules/core/doc/basic_structures.html#mat-convertto

[25] http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#threshold

[26] http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#minmaxloc

Princíp porovnávania symbolov sme spomenuli už v kapitole 5.2. Parametrami funkcie bitového exkluzívneho súčtu (*bitwise_xor* [27]) sú: porovnávaný obrázok, vzorový obrázok, výstupný obrázok (diferenčná matica). Ukážku diferenčnej matice môžeme vidieť na Obr. 5.6. Čím menej bielych bodov (premenná *non_zero*) sa v matici nachádza, tým presnejšia zhoda nastala. Počet bielych bodov zisťujeme funkciou *countNonZero* [28]. Pred začatím porovnávania definujeme premennú *min_non_zero* (maximálny prípustný počet bielych bodov) a index (premenná *match*) hľadaného symbolu (na začiatku rovný hodnote -1). Hranicu maximálneho počtu nenulových bodov znižujeme pri každom priblížení k rozpoznávanému symbolu. Ak dôjde k rozoznaniu symbolu (index je nezáporný), v okne *corrected* vykreslíme diferenčnú maticu a v okne streamu názov rozoznaného symbolu. V okne streamu vykresľujeme (bez ohľadu na stav rozpoznania) takisto orámovanie aproximovanej kontúry spolu s vyznačením vrcholov (funkcie *line* a *circle*). Zvyšnými kontúrami sa nezaoberáme (príkaz *break*).

Nasledovný kód je fragmentom tela funkcie *drive*. Ilustruje implementáciu riadenia pri rozpoznaní prvého symbolu – zatočenia vľavo o 90° (index symbolu je 0):

```
switch (symbol_id)
{
case 0: //Left 90
	sprintf_s(buffer, "7=%dus,1=%dus", 2000, minPower);
	for (i = 0; i < 30; i++) {
		send(socket, buffer, (int)strlen(buffer), 0);
		cap.read(frame);
		imshow("camera", frame);
		waitKey(1);
	}
	break;
	...
}
```

Požadovaný efekt dosiahneme kombináciou iterácie (cyklus *for*) a čakania (funkcia *waitKey*). Príkazy musia byť odosielané (funkcia *send*) čo najrýchlejšie, preto čakáme 1 ms (minimálna prípustná hodnota). Podobným spôsobom sú riešené aj ostatné prípady. Líšia sa iba riadiacimi hodnotami motora, serva a počtom iterácií cyklu.

Objekt *cap* je inštanciou triedy *VideoCapture*, metódou *read* načítavame snímky zo streamu. Funkciou *imshow* následne zobrazíme získaný obrázok v okne streamu.

[27] http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#bitwise-xor

[28] http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#countnonzero

Rýchly sled oboch operácií zabezpečí rovnomerné vyťaženie komunikácie a schopnosť rýchlo zareagovať na nové symboly. Za týmto účelom prebieha tiež komunikácia, výhradne cez TCP/IP sokety (nie *curl*). V onboard verzii aplikácie by sme sa vyťažením siete nemuseli zaoberať vôbec, pretože interval načítavania snímok zo streamu by nemal žiaden vplyv na oneskorenie zasielania riadiacich príkazov.

Demonštračné video offboard verzie v režime rozpoznávania symbolov je dostupné na stránke *http://youtu.be/XUaZGbw4X2o*. Video znázorňuje reakciu RC modelu na vybrané symboly. Symboly sú vytlačené čiernou farbou na bielom papieri veľkosti A4.

5.4 Onboard verzia

Onboard verzia aplikácie pracuje priamo na RPi. Neodosiela sa žiaden stream, celý výkon sa využije na dosiahnutie čo najvyššieho počtu fps. Zdrojový kód napísaný vo VS sa prekladá priamo na RPi, teda potrebujeme zabezpečiť, aby distribúcia obsahovala aj knižnicu OpenCV. Pre tento účel používame obraz SD karty zo stránky fóra [29] RPi, na ktorom sa knižnica nachádza v rovnakej verzii (2.4.9), s akou pracujeme v predošlej kapitole. Operačný systém pochopiteľne zahŕňa iba základný softvér, preto je potrebné opäť nainštalovať všetky chýbajúce ovládače a programy. Knižnica libcurl ani ostatné spôsoby zasielania príkazov (sokety, AJAX) v onboard verzii nemajú opodstatnenie. Klient iba aktivuje vybraný režim rozpoznávania obrazu cez SSH a sleduje pohyb RC modelu. Formát rozpoznávaného obrazu je typu MJPEG.

Nespornou výhodou spracovania obrazu bezprostredne po zosnímaní je zredukovanie odozvy systému. Obraz sa vyhodnotí čo najskôr, ako je to možné – neodosiela sa cez sieť. Kód aplikácie vychádza z offboard verzie a je mierne upravený a optimalizovaný na šetrné využívanie zdrojov. Na skompilovanie použijeme skript *build_all.sh* z adresára *opencv-2.4.9/samples/c*, ktorý priradí hlavičkové súbory knižnice automaticky. Obsah skriptu je nasledovný:

```
#!/bin/sh

if [ $# -gt 0 ] ; then
    base=`basename $1 .c`
    echo "compiling $base"
    gcc -ggdb `pkg-config opencv --cflags --libs` $base.c -o $base
else
```

[29] http://www.raspberrypi.org/forums/viewtopic.php?f=43&t=80628&start=25

```
       for i in *.c; do
           echo "compiling $i"
           gcc -ggdb `pkg-config --cflags opencv` -o `basename $i .c` $i `pkg-
config --libs opencv`;
       done
       for i in *.cpp; do
           echo "compiling $i"
           g++ -ggdb `pkg-config --cflags opencv` -o `basename $i .cpp` $i
`pkg-config --libs opencv`;
       done
   fi
```

Ak nezadáme žiaden parameter, skript skompiluje všetky súbory typu *c* a *cpp* v aktuálnom adresári. V adresári projektu sa celkovo nachádzajú tieto súbory: arrowB.jpg, arrowGo.jpg, arrowL45.jpg, arrowL.jpg, arrowR45.jpg, arrowR.jpg, arrowStop.jpg, arrowT.jpg, build_all.sh, onboard, onboard.cpp, settings_line.txt a settings.txt.

Aplikáciu štartujeme súborom *onboard*. Nastavenia parametrov uchovávame v súbore *settings.txt*. Súbor *settings_line.txt* je úpravou súboru *settings.txt* pre režim sledovania čiary. Po štarte aplikácie sa na obrazovku vypíše spôsob jej použitia:

```
usage: ./onboard <line|ai|symbol> <device nr.> [<debug>]
```

Povinným argumentom je výber režimu (sledovanie čiary, sledovanie lopty, rozpoznávanie symbolu) a číslo video zariadenia (štandardne 0). Argument *debug* je voliteľný, zadávame ho vtedy, ak chceme vidieť výpisy aplikácie zahrňujúce počet fps a hodnoty riadiacich príkazov. Riadiace príkazy odosielame štandardným spôsobom cez deskriptor SB, napríklad:

```
FILE *f = fopen("/dev/servoblaster", "w");
fprintf(f, "7=%dus\n", 1750); fflush(f);
```

Z hľadiska optimalizácie kódu sú takmer všetky lokálne premenné, ktoré pri deklarácii zároveň aj inicializujeme, označené ako statické premenné. Nachádzajú sa v najčastejšie volaných funkciách.

Onboard verzia je najvhodnejšia pre funkciu sledovania čiary. Aby model nevyšiel z dráhy, potrebujeme dosiahnuť dostatočný počet fps a čo najnižšiu odozvu systému. V rozlíšení 320 × 240 px dokázal RPi spracovať približne 17 fps, pri ktorých auto nestratilo čiaru zo zorného poľa a prešlo až na jej koniec. Demonštračné video je dostupné na stránke *http://youtu.be/S738DGEAwE0*. Znázorňuje prechod RC modelu ponad bielu pásku nalepenú na bežnej asfaltovej ceste. V ostatných režimoch je z dôvodu vyšších výpočtových nárokov vhodnejšie použiť offboard verziu. S offboard

aplikáciou model vzhľadom na vyššiu odozvu vplyvom streamovania obrazu nedokázal trvalo kopírovať líniu a jeho rýchlosť bolo nutné regulovať manuálne.

Obr. 5.9: Onboard verzia – sledovanie čiary

5.5 Verzia HTML

Webová implementácia je rozdelená do nasledovných modulov:

- index.html – hlavná stránka s odkazmi na ostatné moduly,
- ws.html – modul riadenia cez webové sokety s video streamom,
- ws_video_off.html – rovnaký ako predošlý, bez video streamu,
- server.html – JavaScript server pre knižnicu *peer.js*,
- client.html – JavaScript klient pre knižnicu *peer.js*,
- ajax.html – modul riadenia cez AJAX s video streamom.

Funkciu memorizácie trasy obsahujú moduly *ws.html* a *ws_video_off.html*. Vo všetkých moduloch pracujeme s JavaScript knižnicou jQuery verzie 2.1.1. Moduly sme umiestnili na SD kartu. Na ich otvorenie cez prehliadač potrebujeme mať na RPi nainštalovaný webový server a pre funkčnosť modulu *ajax.html* aj podporu jazyka PHP:

```
sudo apt-get -y install lighttpd
sudo apt-get -y install php5-common php5-cgi php5
sudo lighty-enable-mod fastcgi-php
sudo service lighttpd force-reload
sudo chown www-data:www-data /var/www
sudo chmod 775 /var/www
sudo usermod -a -G www-data pi
sudo reboot
```

Po reštartovaní RPi umiestníme všetky súbory našej HTML aplikácie do adresára */var/www*. Úvodnú stránku potom nájdeme na adrese *http://192.168.42.1*.

5.5.1 Modul ws.html

Na komunikáciu cez webové sokety (ďalej v texte WS) použijeme prehliadač Google Chrome. Chrome bude vystupovať ako klient a serverovú časť implementujeme na RPi v jazyku Python, ktorý je súčasťou distribúcie. Podporu WS pre server zaistíme frameworkom Tornado:

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
sudo apt-get install python-pip
sudo pip install tornado
```

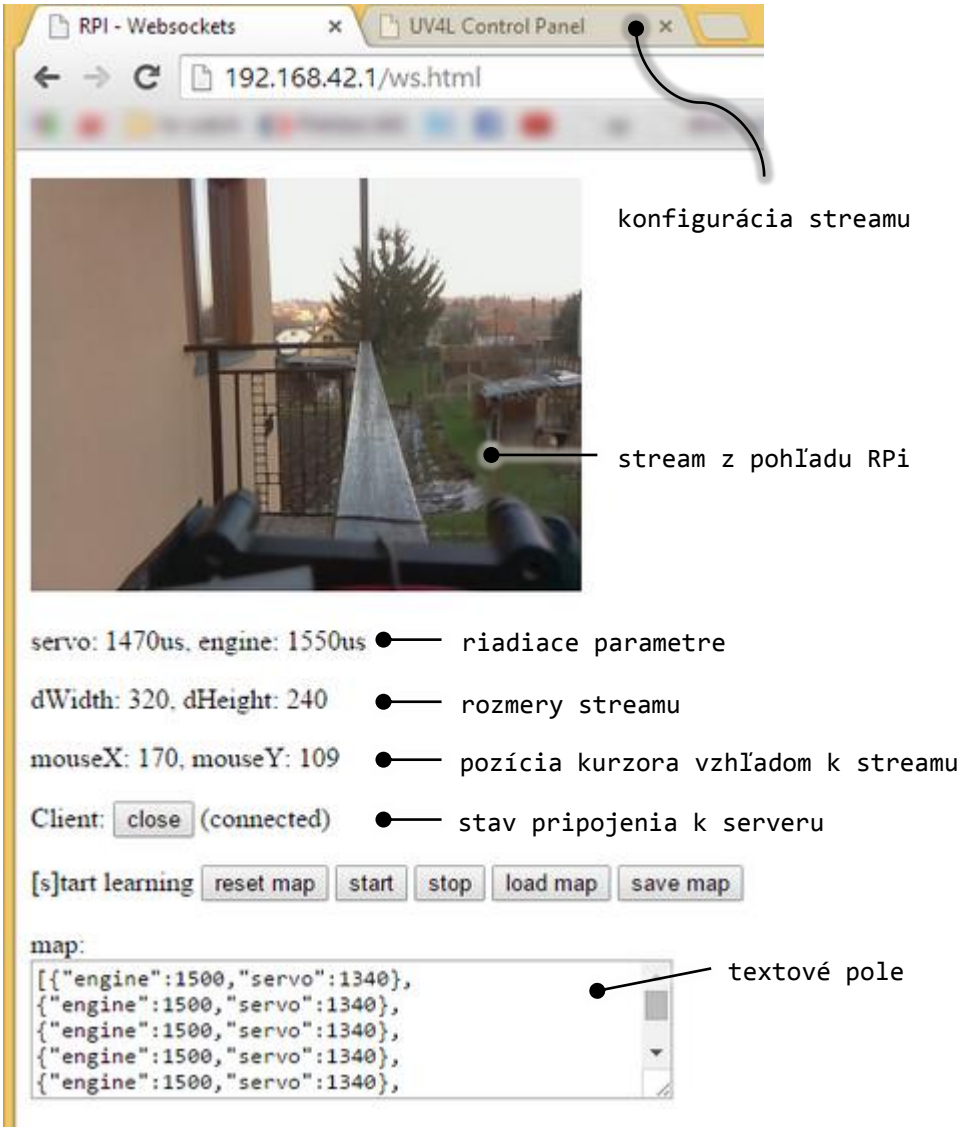
Kód serverovej časti (vytvoríme súbor *server.py*) – pozri príloha B. Server po štarte príkazom *python server.py* načúva na porte 8888. Po prijatí správy v JSON formáte odošle deskriptoru SB riadiaci príkaz, osobitne pre ESC a servo. Príkaz odosielame procedúrou *sendpwm*. Volaním metódy *flush* [30] zabezpečíme zápis všetkých dát do deskriptora. Ak by sme túto metódu nezavolali, SB by na žiadne ďalšie odoslané príkazy nereagoval.

Kód klienta (*ws.html*) – pozri príloha B. Klient sa po štarte pripojí cez adresu RPi (*location.hostname*) a port 8888 k serveru a vytvorí webový soket. Soket môžeme neskôr ukončiť okrem zatvorenia servera/klienta kliknutím na tlačidlo *close* (príkaz *ws.close()*) – pozri Obr. 5.10. Obraz z kamery (element *#stream*) zobrazujeme vo formáte MJPEG (viac o konfigurácii streamu v kapitole 4.3). Inicializujeme rozmery videa a určíme jeho stred. Z rozmerov streamu a aktuálnej polohy myši pri pohybe (udalosť *mousemove*) nad ním vypočítame hodnoty riadiacich príkazov. Postup výpočtu je podobný ako v kapitole 5.3.2.

Zavolaním metódy *drive* sa aktivuje pravidelné odosielanie riadiacich príkazov v intervale 50 ms, teda dvadsaťkrát za sekundu. Príkazy sú odosielané vo formáte JSON. Udalosťami *keydown* a *keyup* zisťujeme stlačenie kláves. Držaním klávesu „A“ odblokujeme zasielanie hodnôt pre ESC iných ako 1500 µs, pri ktorej sa motor neotáča (neutrál). Funkcia memorizácie trasy sa aktivuje/deaktivuje klávesom „S“.

[30] http://www.cplusplus.com/reference/cstdio/fflush

Aktivovaním funkcie sa začnú riadiace príkazy ukladať do jednorozmerného poľa. Trasu je tiež možné načítať z textového poľa, alebo vygenerovať do textového poľa.



Obr. 5.10: HTML verzia – modul ws.html

5.5.2 Modul ajax.html

Ajax modul využíva na zasielanie riadiacich príkazov asynchrónne požiadavky jazyka JavaScript. Metóda odosielania je typu POST[31]. Spôsob načítania streamu, zachytávania stlačenia klávesov a pohybu myši nad streamom zostáva nezmenený. Odlišnosť implementácie voči predchádzajúcemu modulu tkvie v procedúre *drive*:

```
function drive() {
    clearInterval(int)
```

[31] http://cs.wikipedia.org/wiki/POST

```
    int = setInterval(function() {
           $.post('rpi.php', { engine: keys[87]? engine : 1500, servo: servo })
    }, 50)
}
```

Údaje sú odosielané na spracovanie PHP skriptom *rpi.php*, ktorý sa nachádza v rovnakom adresári ako ostatné moduly. Kód skriptu je prostý:

```
<?
    echo shell_exec("
           echo 1=$_POST[engine]us > /dev/servoblaster;
           echo 7=$_POST[servo]us > /dev/servoblaster;"
    );
?>
```

Metóda *shell_exec* [32] vykoná príkaz nachádzajúci sa medzi úvodzovkami v termináli RPi a vráti kompletný výstup ako reťazec.

5.5.3 Moduly client.html a server.html

Tieto moduly nám umožňujú ovládať model cez internet. Princíp komunikácie je ukrytý v premostení webového soketu a knižnici PeerJS[33], určenej na vytváranie P2P spojení prostredníctvom WebRTC[34] API. Modul *server.html* vytvorí webový soket a P2P spojenie. Modul *client.html* vytvorí iba P2P spojenie. Klient odosiela riadiace hodnoty serveru cez P2P a server ich následne preposiela cez webový soket v rámci WiFi siete do RPi. Video prenos môže byť realizovaný napríklad aplikáciou Skype nainštalovanou na smartfóne. Telefón slúži ako AP pre RPi a je uchytený na modeli.

Modul *client.html* je opäť upravenou verziou predošlých modulov:

```
function drive() {
    clearInterval(int)
    int = setInterval(function() {
           if (socket.open)
                  socket.send({ engine:keys[87]? engine:1500, servo:servo })
    }, 50)
}
```

Objekt *socket* v tomto prípade reprezentuje P2P spojenie so serverom:

```
var client = new Peer({key: '72rwkk1xno7xpqfr'});

client.on('open', function(id) {
    console.log('client id: ' + id);
});
client.on('close', function() {
    console.log('client closed')
```

[32] http://php.net/manual/en/function.shell-exec.php

[33] http://peerjs.com

[34] http://cs.wikipedia.org/wiki/WebRTC

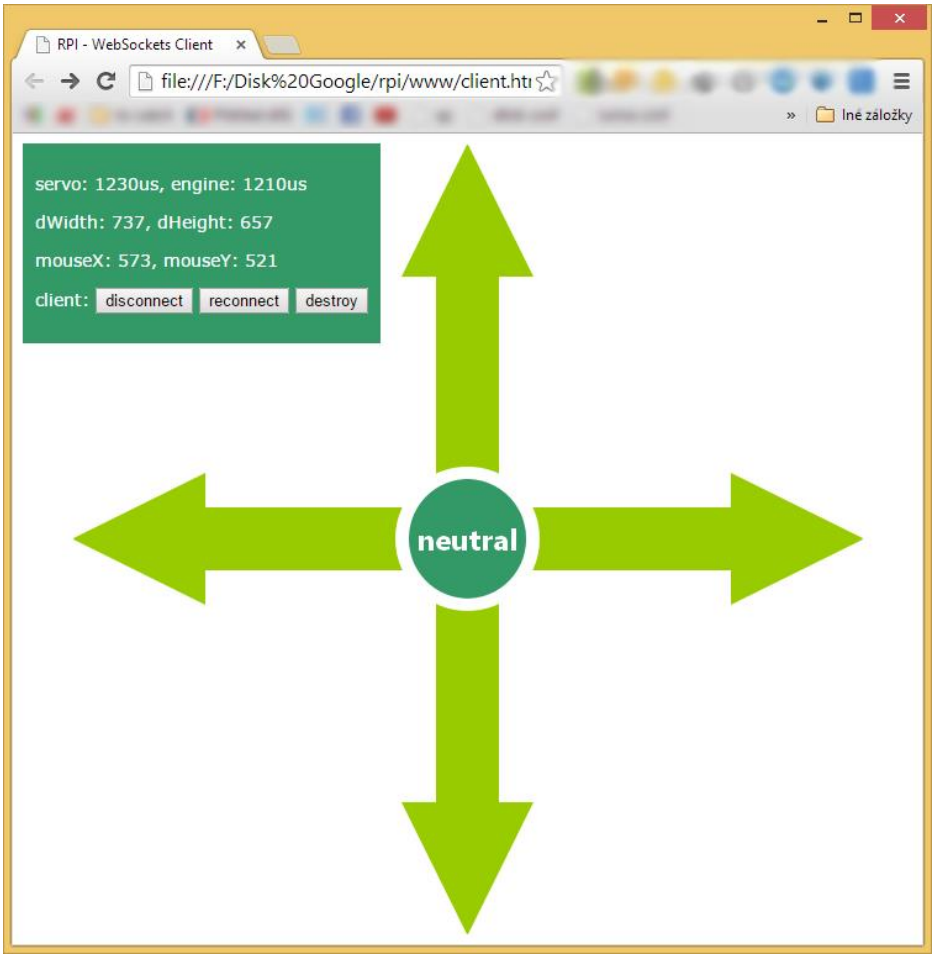
```
});
client.on('disconnected', function() {
    console.log('client disconnected')
});

$('#client-disconnect').click(function(){
    client.disconnect();
});
$('#client-reconnect').click(function(){
    client.reconnect();
});
$('#client-destroy').click(function(){
    client.destroy();
});

var socket;
var server = 'l3ve8x1wai82gldi';

socket = client.connect(server)
socket.on('open', function() {
    console.log('pripojeny na server');
});
socket.on('data', function(data) {
    console.log(data, '(server)')
});
socket.on('close', function() {
    console.log('odpojeny zo servera')
});
socket.on('error', function(err) { console.log(err.message) });
```

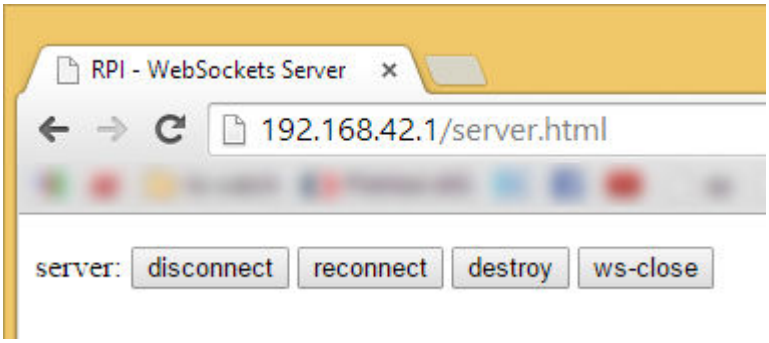
Na pripojenie k serveru je potrebné vytvoriť osobitne spojenie *client* s rovnakým kľúčom skupiny (*key*), aký má server. Každé spojenie má svoj automaticky generovaný identifikátor. Identifikátor môže byť špecifikovaný tiež ručne, čo sme urobili. Tým simulujeme model klient-server vo vzťahu 1:1 na sieti typu P2P.



Obr. 5.11: HTML verzia – modul client.html

Kód modulu *server.html* – pozri príloha B. Štartu klienta musí predchádzať štart servera. Server vytvorí P2P spojenie s ručne špecifikovaným identifikátorom a kľúčom skupiny. Ak by sme nezadali identifikátor manuálne, bol by vygenerovaný náhodne a klientovi by sa tak znemožnilo spojiť sa so serverom. Kľúč skupiny sme získali po zaregistrovaní z webovej stránky knižnice PeerJS. Kľúč sa generuje na základe nami stanoveného maximálneho počtu spojení (1 – 50) v rámci vytváranej P2P siete. V našom prípade sú dve spojenia – 1 spojenie pre server a 1 spojenie pre klienta.

Server po vytvorení webového soketu a P2P spojenia čaká na príjem dát od klienta. Keď klient dáta odošle, server ich následne prepošle cez webový soket do RPi.



Obr. 5.12: HTML verzia – modul server.html

5.6 Zhrnutie

Zámerom poslednej časti práce bolo vytvoriť aplikáciu automatického riadenia RC modelu. Na začiatku kapitoly sme si predstavili použité softvérové knižnice a uviedli postup integrácie knižníc OpenCV a libcurl do vývojového prostredia Microsoft Visual Studio. Ďalej sme opísali podporované režimy aplikácie. Náš program bol implementovaný v podobe konzolovej aplikácie a webovej stránky.

Konzolová aplikácia má dve variácie – offboard a onboard. Offboard verzia beží na klientskom PC. Na zasielane riadiacich príkazov využíva TCP/IP sokety alebo HTTP POST požiadavky (knižnica libcurl). Onboard verzia pracuje priamo na RPi. Výhodou offboard verzie je streamovanie obrazu a potenciálne vyšší výkon (závisí od PC klienta) pri analýze obrazu. Nevýhodou offboard verzie je potreba aktívnej komunikácie cez bezdrôtovú sieť a z toho vyplývajúce nároky na nízku odozvu, stabilitu a rovnomerné vyťaženie spojenia. Onboard verzia pracuje lokálne, preto nemá takmer žiadnu odozvu, respektíve odozva závisí iba od rýchlosti samotného RPi. Testovanie ukázalo, že na sledovanie čiary je vhodnejšia onboard verzia a pre ostatné režimy offboard verzia aplikácie. Dôvodom lepšieho výkonu onboard verzie v režime sledovania čiary sú zrejme znížené výpočtové nároky na analýzu výrezu obrazu, v porovnaní s analýzou celého obrazu u ostatných režimov.

Implementácia formou webovej stránky – označovaná ako HTML verzia, je určená výlučne na manuálne ovládanie modelu. Ostatné režimy nie sú podporované. Pozostáva z niekoľkých modulov, ktoré môžeme podľa spôsobu odosielania riadiacich príkazov rozdeliť do troch kategórií. Prvým spôsobom odosielania príkazov sú webové sokety. Umožňujú obojsmerný prenos dát cez samostatné TCP spojenie. Moduly HTML aplikácie komunikujúce touto cestou sú navyše vybavené funkciou memorizácie trasy. Druhým spôsobom odosielania príkazov sú HTTP POST

požiadavky (cez AJAX), založené na modeli požiadavka-odpoveď. Tretí spôsob odosielania príkazov využíva webové sokety v kombinácii s aplikačným rozhraním WebRTC, ktoré nám umožňuje prostredníctvom spojenia typu peer-to-peer ovládať RC model aj cez internet. Prezentované algoritmy zasielania príkazov cez webové sokety a AJAX sa príliš nelíšia. Rozdiel spočíva predovšetkým v metóde odosielania a implementácii serverovej časti.

Záver

V práci sme sa zaoberali prestavbou RC modelu auta na model riadený počítačom. Na zasielanie riadiacich príkazov bol použitý mini počítač Raspberry Pi. Streamovanie obrazu zaistila Raspberry Pi kamera. Pri prestavbe bolo snahou minimalizovať náklady na realizáciu, počet použitých komponentov a eventuálne aj spotrebu zostavy.

Prvým cieľom práce bolo nahradiť pôvodný (páčkový) ovládač RC modelu počítačovou myšou a klávesnicou. Zasielanie riadiacich príkazov pomocou ovládača nahradil program ServoBlaster, ktorý umožnil generovať PWM signál a ním prostredníctvom vstavaných GPIO pinov priamo riadiť servo a ESC modelu auta. Spomedzi spôsobov odosielania riadiacich príkazov medzi RPi a PC sme porovnali technológie AJAX a Web Sockets. Výsledok vyznel v prospech webových soketov, u ktorých bola komunikácia súvislejšia a za rovnakých podmienok dátovo efektívnejšia.

Druhým cieľom bolo zabezpečiť plynulý prenos obrazu z pohľadu RC auta s minimálnou odozvou. Opísali sme tri metódy streamovania. Najnižšie oneskorenie bolo zaznamenané súčinnosťou aplikácií raspivid a GStreamer. Na účely neskoršej tvorby aplikácie využívajúcej počítačové videnie sme nasadili metódu streamovania pomocou HTTP servera, pretože podporoval video formáty kompatibilné s knižnicou pre manipuláciu s obrazom.

Tretím cieľom bolo vyvinúť aplikáciu, ktorá by RC modelu umožnila interagovať s vonkajším prostredím. Naša vytvorená aplikácia používa na rozoznávanie obrazu knižnicu OpenCV a má tri verzie. Prvá verzia je určená pre beh na PC, druhá pre beh na RPi. Tieto verzie podporujú funkcie autonómneho riadenia – sledovanie čiary, sledovanie lopty a rozpoznávanie symbolov. Tretia verzia pracuje v webovom prehliadači. Zahŕňa doplnkové funkcie a implementuje metódy zasielania riadiacich príkazov cez AJAX/websokety.

Z hľadiska naplnenia uvedených cieľov môžeme konštatovať, že sa nám podarilo všetky ciele úspešne uskutočniť. Prestavba RC modelu je finančne nenáročná, vyžaduje iba základné komponenty. Voľbou vhodného RPi modelu, externej batérie a WiFi adaptéra sa dosiahli výborné prevádzkové parametre. Získané poznatky vypovedajú o mimoriadnej využiteľnosti počítača Raspberry Pi, a to najmä

v domácich robotických projektoch a multimediálnych aplikáciách. S ďalším rastom výkonu, miniaturizáciou a znižovaním spotreby sa môžeme v budúcnosti stretnúť s novými, sofistikovanými zariadeniami, ktoré v mnohých prípadoch dokážu nahradiť tie dnešné zariadenia za zlomok ich ceny.

V práci som zhrnul svoje doposiaľ nadobudnuté skúsenosti s platformou Raspberry Pi. Okrem základných realizovaných cieľov práca prináša aj niektoré doplnkové funkcie, najmä ovládanie RC modelu cez internet a programovanie trasy jazdy. K ďalším funkciám by v budúcnosti mohla pribudnúť napríklad GPS navigácia alebo rameno na manipuláciu s objektmi. Výmenou RPi za novší model s vyšším výkonom možno očakávať aj lepšie výsledky v podporovaných režimoch, a tým spoľahlivejšie autonómne riadenie.

Zoznam bibliografických odkazov

[1] FAQs | Raspberry Pi [online]. [cit. 2014-02-20]. Dostupné na internete: <http://www.raspberrypi.org/faqs>.

[2] Camera | Raspberry Pi [online]. [cit: 2014-02-18]. Dostupné na internete: <http://www.raspberrypi.org/camera>.

[3] CAMERA [online]. [cit. 2015-01-18]. Dostupné na internete: <http://www.raspberrypi.org/documentation/hardware/camera.md>.

[4] BALANCER Li-Po [online]. [cit. 2015-01-18]. Dostupné na internete: <http://www.forum.tamiyaforum.cz/viewtopic.php?f=116&t=6193#p107237>.

[5] Dual Batteries in 1/16 Models [online]. [cit. 2015-01-18]. Dostupné na internete: <http://traxxas.com/support/Dual-Batteries-116-Models>.

[6] FRIED, L. Setting up a Raspberry Pi as a WiFi access point [online]. [cit. 2015-01-18]. Dostupné na internete: <https://learn.adafruit.com/setting-up-a-raspberry-pi-as-a-wifi-access-point?view=all>.

[7] WILLIAMS, M. Software PWM on a Raspberry Pi [online]. [cit. 2015-01-18]. Dostupné na internete: <http://marks-space.com/2013/09/23/software-pwm-on-a-raspberry-pi>.

[8] HIRST, R. ServoBlaster [online]. [cit. 2015-01-18]. Dostupné na internete: <https://github.com/richardghirst/PiBits/tree/master/ServoBlaster>.

[9] Le Raspberry Pi : Installer WiringPi & Utiliser le GPIO [online]. [cit. 2015-01-18]. Dostupné na internete: <http://domologis.fr/2014/12/07/le-raspberry-pi-installer-wiringpi-utiliser-le-gpio>.

[10] What is an ESC [online]. [cit. 2015-01-18]. Dostupné na internete: <http://blog.pistuffing.co.uk/?p=1128>.

[11] GStreamer: open source multimedia framework [online]. [cit. 2015-01-18]. Dostupné na internete: <http://gstreamer.freedesktop.org>.

[12] RISOLIA, L. Linux Projects – Documentation [online]. [cit. 2015-01-18]. Dostupné na internete: <http://linux-projects.org/modules/sections/?op=viewarticle&artid=14>.

[13] OpenCV [online]. [cit. 2015-01-18]. Dostupné na internete: <http://en.wikipedia.org/wiki/OpenCV>.

[14] STENBERG, D. Curl and libcurl [online]. [cit. 2015-01-18]. Dostupné na internete: <http://curl.haxx.se>.

[15] STENBERG, D. Libcurl – the multiprotocol file transfer library [online]. [cit. 2015-01-18]. Dostupné na internete: <http://curl.haxx.se/libcurl>.

[16] cURL [online]. [cit. 2015-01-18]. Dostupné na internete: <http://en.wikipedia.org/wiki/CURL>.

[17] MATOS, S. Signs reading with OpenCV [online]. [cit. 2015-01-18]. Dostupné na internete: <http://roboticssamy.blogspot.pt/2014/02/signs-reading-with-opencv-code.html>.

[18] FERNANDO, S. Color Detection & Object Tracking [online]. [cit. 2015-01-18]. Dostupné na internete: <http://opencv-srf.blogspot.sk/2010/09/object-detection-using-color-seperation.html>.

[19] MATOS, S. Signs reading with OpenCV [online]. [cit. 2015-01-18]. Dostupné na internete: <http://roboticssamy.blogspot.pt/2013/11/rs4-robot-update-new-features.html>.

[20] AMIN, N. Automatic perspective correction for quadrilateral objects [online]. [cit. 2015-01-18]. Dostupné na internete: <http://opencv-code.com/tutorials/automatic-perspective-correction-for-quadrilateral-objects>.

[21] Miscellaneous Image Transformations [online]. [cit. 2015-01-18]. Dostupné na internete: <http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#threshold>.

Príloha A: Konfiguračné parametre

Tabuľka 1: Parametre programu ServoBlaster

--pcm	Ak je nastavený, na vytvorenie oneskorení sa použije PCM namiesto PWM hardvéru.
--idle-timeout=Nms	Doba, počas ktorej majú byť odosielané pulzy po poslednom príkaze.
--cycle-time=Nus	Dĺžka cyklu v µs (predvolene 20 000 µs).
--step-size=Nus	Veľkosť kroku v µs (predvolene 10 µs).
--min={N|Nus|N%}	Minimálna povolená šírka pulzu definovaná počtom krokov, µs alebo v percentách. Predvolene 50 krokov (500 µs).
--max={N|Nus|N%}	Maximálna povolená šírka pulzu definovaná počtom krokov, µs alebo v percentách. Predvolene 250 krokov (2 500 µs).
--invert	Invertuje zadané hodnoty.
--dma-chan=N	Nastavuje číslo DMA kanálu, predvolene 14.
--p1pins=<zoznam>	Nastavuje zoznam pinov, ktoré sa majú použiť na hlavičke P1 (predvolene „7,11,12,13,15,16,18,22“).
--p5pins=<zoznam>	Nastavuje zoznam pinov, ktoré sa majú použiť na hlavičke P5 (predvolene „“).

Pozn.: ms – milisekunda, µs – mikrosekunda; 1 s = 1000 ms, 1 ms = 1000 µs.

Tabuľka 2: Netcat a MPlayer – parametre príjmu obrazu

u	Režim UDP[1].
L	Aktivuje režim načúvania. Veľké písmeno znamená „tvrdší“ režim (podporovaný len v OS Windows), pri ktorom načúvanie začne opäť aj po odpojení aktuálneho klienta.
p	Naslúchací port. Rovnaký (5001) nastavíme aj pri streamovaní.
framedrop	Povolí odhadzovanie oneskorených snímok.
nosound	Aktivuje režim bez prijímania zvuku.
fps	Počet snímok za sekundu. Ak je vyšší ako počet fps nastavený pri streamovaní, prehrávanie bude mať nižšiu odozvu.
demuxer	Formát prijímaného videa (FFmpeg H.264).
-	Použitý na konci príkazu. Vyjadruje, že MPlayer bude čítať zo štandardného vstupu (stdin) a nie zo súboru.

Tabuľka 3: Raspivid – parametre streamovania obrazu

n	Nezobrazí sa okno s náhľadom (ak je pripojený TV).
t	Dĺžka záznamu (v ms). Predvolene 5000. Ak je rovná 0, záznam bude prebiehať neustále.
w	Šírka obrazu (px). Predvolene 1920.
h	Výška obrazu (px). Predvolene 1080.
fps	Počet snímok za sekundu.
vf	Vertikálne preklopenie obrazu.
hf	Horizontálne preklopenie obrazu.
b	Dátový tok v bitoch za sekundu. Predvolene 10 000 000 (10 Mb/s).
o	Výstupný súbor. Ak je „-“, bude sa zapisovať na štandardný výstup (stdout).

[1] http://en.wikipedia.org/wiki/User_Datagram_Protocol

Tabuľka 4: Visual Studio – integrácia knižníc OpenCV a libcurl

Sekcia	**Parameter**	**Hodnota**	**Konfigurácia**
Ladenie	Prostredie	PATH=$(ExecutablePath)$(LocalDebuggerEnvironment)	Všetky
Adresáre VC++	Adresáre spustiteľných súborov	C:\curl\dlls;$(ExecutablePath)	Všetky
	Adresáre súborov k zahrnutiu	C:\curl\include;$(IncludePath)	
	Adresáre knižníc	C:\curl\lib;$(LibraryPath)	
C/C++			
Všeobecné	Ďalšie adresáre súborov k zahrnutiu	C:\opencv\build\include;%(AdditionalIncludeDirectories)	Všetky
Preprocesor	Definície preprocesora	WIN32;_CRT_SECURE_NO_WARNINGS;NDEBUG;_CONSOLE;_LIB	
Linker			
Všeobecné	Ďalšie adresáre knižníc	C:\opencv\build\x86\vc12\lib;%(AdditionalLibraryDirectories)	Všetky
Vstup	Ďalšie závislosti	libcurl.lib	Všetky
		opencv_contrib249d.lib opencv_stitching249d.lib opencv_videostab249d.lib opencv_superres249d.lib opencv_nonfree249d.lib opencv_gpu249d.lib opencv_ocl249d.lib opencv_legacy249d.lib opencv_ts249d.lib opencv_calib3d249d.lib opencv_objdetect249d.lib opencv_features2d249d.lib opencv_highgui249d.lib opencv_photo249d.lib opencv_video249d.lib opencv_imgproc249d.lib opencv_ml249d.lib opencv_flann249d.lib	Debug

		opencv_core249d.lib	
		opencv_calib3d249.lib opencv_contrib249.lib opencv_core249.lib opencv_features2d249.lib opencv_flann249.lib opencv_gpu249.lib opencv_highgui249.lib opencv_imgproc249.lib opencv_legacy249.lib opencv_ml249.lib opencv_nonfree249.lib opencv_objdetect249.lib opencv_ocl249.lib opencv_photo249.lib opencv_stitching249.lib opencv_superres249.lib opencv_ts249.lib opencv_video249.lib opencv_videostab249.lib	Release

Tabuľka 5: Parametre aplikácie autonómneho riadenia

LowH	Minimálny farebný odtieň. (0 – 179)
HighH	Maximálny farebný odtieň. (0 – 179)
LowS	Minimálna sýtosť farby. (0 – 255)
HighS	Maximálna sýtosť farby. (0 – 255)
LowV	Minimálny jas. (0 – 255)
HighV	Maximálny jas. (0 – 255)
cannyT	Hodnota prahovania pri hľadaní hrán v režime rozpoznávania symbolov. (0 – 255)
cannyE	Presnosť aproximácie. Určuje maximálnu vzdialenosť medzi pôvodnou krivkou a jej aproximáciou v režime rozpoznávania symbolov. (0 – 100)
roiY	Zvislá pozícia začiatku oblasti záujmu v režime sledovania čiary. (0 – 100 %)
symbContr	Kontrast B&W snímky pred prahovaním v režime rozpoznávania symbolov. (0 – 3)
symbMin	Minimálna plocha obrysu pre rozpoznanie symbolu. (0 – 15 000 px)
lineMin	Minimálna plocha obrysu pre rozpoznanie čiary. (0 – 10 000 px)
ballMin	Minimálna plocha obrysu pre rozpoznanie lopty. (0 – 100 000 px)
aiMin	Maximálna plocha obrysu lopty, pri ktorej sa model nepohybuje. (0 – 50 000 px)
aiMax	Minimálna plocha obrysu lopty, pri ktorej model cúva. (0 – 100 000 px)
minPower	Minimálna šírka pulzu ovplyvňujúca rýchlosť motora. (0 – 2 000 μs)

Príloha B: Ukážky zdrojových kódov

Funkcia main

```
int main(int argc, const char** argv) {

	char buffer[DEFAULT_BUFLEN];

	const char *ip = "192.168.42.1";
	sprintf_s(buffer, "http://%s:9000/stream/video.mjpeg", ip);

	if (argc > 1) {
		if (argc != 3) {
			cout << "usage:  " << argv[0] << "[<rpi_ip> <h264|mjpeg>]"
				<< endl << endl;
			return 1;
		}
		ip = argv[1];
		sprintf_s(buffer, "http://%s:9000/stream/video.%s", ip, argv[2]);
	}

	cout << buffer << endl;

	VideoCapture cap(buffer);
	if (argc == 3 && strcmp(argv[2], "h264") == 0)
		cap.set(CV_CAP_PROP_FOURCC, CV_FOURCC('X', '2', '6', '4'));

	if (!cap.isOpened()) {
		cout << "Cannot open the video cam" << endl;
		system("pause");
		return -1;
	}

	Symbol symbols[8];
	if (readRefImages(symbols) == -1) {
		printf("Error reading reference symbols\n");
		system("pause");
		return -1;
	}

	double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH);
	double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT);
	double area = 0;

	cout << "Frame size : " << dWidth << " x " << dHeight << endl;

	bool	curl_send = 0,
		socket_send = 1,
		ai = 0,
		line_follow = 0,
		find_symbol = 1;

	int	c,
		servo = 0,
		engine = 0,
		symbol_id = -1,
		last_symbol = -1;

	/* vytvorenie CURL */
	CURL *curl;
	CURLcode res;
```

```
sprintf_s(buffer, "http://%s/rpi.php", ip);
curl_global_init(CURL_GLOBAL_ALL);
curl = curl_easy_init();
curl_easy_setopt(curl, CURLOPT_URL, buffer);

/* vytvorenie soketu */
int iResult;
SOCKET ConnectSocket;
if (vytvor_soket(ip, ConnectSocket) == 1) {
	system("pause");
	return 1;
}

/* opencv vars */
Point2f
	center(dWidth / 2, dHeight / 2),
	roi_center;
Point coords;
Mat frame;

/* Windows */
namedWindow("controls", CV_WINDOW_NORMAL);
namedWindow("camera", CV_WINDOW_AUTOSIZE);
namedWindow("treshold", CV_WINDOW_AUTOSIZE);
namedWindow("corrected", CV_WINDOW_AUTOSIZE);
namedWindow("canny", CV_WINDOW_AUTOSIZE);

/* Trackbars */
FILE *fr = fopen("settings.txt", "r");
if (fr == NULL) {
	cout << "settings.txt not opened" << endl;
	return 1;
}

vector<int> settings;
while (fgets(buffer, 100, fr) != NULL) {
	if (strlen(buffer) < 2) continue;
	vector<string> tokens;
	char * pch;
	pch = strtok(buffer, " ;=");
	while (pch != NULL) {
		tokens.push_back(pch);
		pch = strtok(NULL, " ;=");
	}
	settings.push_back(atoi(tokens.at(2).c_str()));
}
fclose(fr);

int i = 0;
// settings.txt
int iLowH = settings[i++];
int iHighH = settings[i++];
...

i = 0;

createTrackbar("LowH", "controls", &iLowH, 179); //Hue (0 - 179)
createTrackbar("HighH", "controls", &iHighH, 179);
...

setMouseCallback("camera", CallBackFunc, &coords);
```

```
double diff;

for (;;) {

	if (!cap.read(frame)) {
		cout << "Cannot read a frame from video stream" << endl;
		break;
	}

	if (find_symbol) {...} //viď kapitola 5.3.6
	else if (line_follow) {...} //viď kapitola 5.3.5
	else if (ai) {...} //viď kapitola 5.3.4
	else {...} //viď kapitola 5.3.3

	if (!find_symbol) {
		sprintf_s(buffer, "servo=%d&engine=%d", servo, engine);
		putText(frame, buffer, Point(10, 20), 1, 1, Scalar(0, 255, 255));
		if (!ai && !GetAsyncKeyState(VK_CONTROL)
			&& !GetAsyncKeyState(VK_LBUTTON))
			engine = 1500;

		if (socket_send) {
			sprintf_s(buffer, "7=%dus,1=%dus", servo, engine);
			iResult = send(ConnectSocket, buffer,
					(int)strlen(buffer), 0);

			if (iResult == SOCKET_ERROR) {
				printf("send failed with error: %d\n",
					WSAGetLastError());
				closesocket(ConnectSocket);
				WSACleanup();
				system("pause");
				return 1;
			}
		}
		else if (++i >= 3) {
			i = 0;
			if (curl_send) {
				curl_easy_setopt(curl, CURLOPT_POSTFIELDS, buffer);
				res = curl_easy_perform(curl);
				if (res != CURLE_OK)
					fprintf(stderr, "curl_easy_perform()
						failed: %s\n",
						curl_easy_strerror(res));
			}
		}
	}

	sprintf_s(buffer, "pixelov: %.0f, m[%d, %d]", area, coords.x, coords.y);
	putText(frame, buffer, Point(10, 40), 1, 1, Scalar(0, 255, 255));

	sprintf_s(buffer, "Socket:%d, Curl:%d, Find_symbol:%d",
			socket_send, curl_send, find_symbol);
	putText(frame, buffer, Point(10, 60), 1, 1, Scalar(0, 255, 255));
	sprintf_s(buffer, "[a]i: %d, [ctrl]: %d, [l]ine: %d", ai,
			GetAsyncKeyState(VK_CONTROL) != 0, line_follow);
	putText(frame, buffer, Point(10, 80), 1, 1, Scalar(0, 255, 255));

	imshow("camera", frame);

	c = waitKey(delay);
```

```
		if (c == 27)
			break;
		else if (c == 'c')
			curl_send = !curl_send;
		else if (c == 's')
			socket_send = !socket_send;
		else if (c == 'f')
			find_symbol = !find_symbol;
		else if (c == 'a')
			ai = !ai;
		else if (c == ' ')
			waitKey();
		else if (c == 'l')
			line_follow = !line_follow;
	}

	send(ConnectSocket, "-1", 2, 0);

	curl_easy_cleanup(curl);
	curl_global_cleanup();

	closesocket(ConnectSocket);
	WSACleanup();

	return 0;
}
```

Funkcia trackBall

```
void trackBall(Mat frame, Scalar LowHSV, Scalar HighHSV, Point2f& roi_center,
                    double &area, int min_area)
{
	Mat hsv;
	cvtColor(frame, hsv, COLOR_BGR2HSV);

	Mat imgThresholded;
	inRange(hsv, LowHSV, HighHSV, imgThresholded);

	static Mat structuringElement = getStructuringElement(MORPH_ELLIPSE,
                                                              Size(5, 5));
	erode(imgThresholded, imgThresholded, structuringElement);
	dilate(imgThresholded, imgThresholded, structuringElement);

	dilate(imgThresholded, imgThresholded, structuringElement);
	erode(imgThresholded, imgThresholded, structuringElement);

	imshow("treshold", imgThresholded);

	Mat kontury;
	imgThresholded.copyTo(kontury);

	vector<vector<Point>> contours;
	vector<Vec4i> hierarchy;
	findContours(kontury, contours, hierarchy, CV_RETR_LIST,
                CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

	int contours_size = contours.size();
	contours.resize(contours_size);

	vector<RotatedRect> minEllipse(contours_size);

	static Scalar color(255, 255, 255);

	for (int i = 0; i < contours_size; i++)
	{
		area = contourArea(contours[i]);
		if (contours[i].size() > 10 && area > min_area)
		{
			drawContours(frame, contours, i, color, 1, 8,
                                vector<Vec4i>(), 0, Point());
			minEllipse[i] = fitEllipse(contours[i]);

			ellipse(frame, minEllipse[i], color, 2, 8);
			roi_center = minEllipse[i].center;

			break; //len 1 obj
		}
	}
}
```

Funkcia lineFollow

```
void lineFollow(Mat &frame, Scalar LowHSV, Scalar HighHSV, Rect roiR, Point2f&
                    roi_center, double &area, int min_area)
{
	frame = frame(roiR);

	Mat hsv;
	cvtColor(frame, hsv, COLOR_BGR2HSV);

	Mat imgThresholded;
	inRange(hsv, LowHSV, HighHSV, imgThresholded);

	static Mat structuringElement = getStructuringElement(MORPH_ELLIPSE,
                                                              Size(5, 5));
	erode(imgThresholded, imgThresholded, structuringElement);
	dilate(imgThresholded, imgThresholded, structuringElement);

	dilate(imgThresholded, imgThresholded, structuringElement);
	erode(imgThresholded, imgThresholded, structuringElement);

	imshow("treshold", imgThresholded);

	vector<vector<Point>> contours;
	vector<Vec4i> hierarchy;
	findContours(imgThresholded, contours, hierarchy, CV_RETR_LIST,
                CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

	int contours_size = contours.size();
	contours.resize(contours_size);

	Moments mu;
	static Scalar color(255, 255, 255);
	static float centerY = frame.rows / 2;

	for (int i = 0; i < contours_size; i++) {
		area = contourArea(contours[i]);
		if (area > min_area) {
			mu = moments(contours[i]);
			roi_center = Point2f(mu.m10 / mu.m00, centerY);
			drawContours(frame, contours, i, color, 1, 8,
                                vector<Vec4i>(), 0, Point());
			break; //len 1 obj
		}
	}
}
```

Funkcia findSymbol

```
int findSymbol(Mat frame, Symbol *symbols, int edgeThresh, int kontrast, int
contour_area)
{
	cvtColor(frame, imgGrayScale, CV_BGR2GRAY);
	GaussianBlur(imgGrayScale, imgGrayScale, Size(9, 9), 2, 2);

	Canny(imgGrayScale, canny_output, edgeThresh, edgeThresh * 3, 3);
	imshow("canny", canny_output);

	findContours(canny_output, contours, hierarchy, CV_RETR_LIST,
			CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

	contours_size = contours.size();
	contours.resize(contours_size);

	static bool first_run = true;
	if (first_run) {
		first_run = false;
		quad_pts.clear();
		quad_pts.push_back(Point2f(0, 0));
		quad_pts.push_back(Point2f(correctedImg.cols, 0));
		quad_pts.push_back(Point2f(correctedImg.cols, correctedImg.rows));
		quad_pts.push_back(Point2f(0, correctedImg.rows));
	}

	int match = -1,
		i, j;

	static Point symbol_position(frame.cols / 2, frame.rows - 50);
	static Scalar color(Scalar::all(255));

	for (i = 0; i < contours_size; i++) {
		approxPolyDP(contours[i], approx_polygon,
				arcLength(contours[i], true) *0.05, true);

		if (approx_polygon.size() == 4
			&& contourArea(contours[i]) > contour_area) {

			corners.clear();
			for (j = 0; j < 4; j++)	{
				corners.push_back(approx_polygon[j]);
			}

			mu = moments(contours[i], false);
			center = Point2f(mu.m10 / mu.m00, mu.m01 / mu.m00);

			sortCorners(corners, center);

			transmtx = getPerspectiveTransform(corners, quad_pts);
			warpPerspective(frame, correctedImg, transmtx,
					correctedImg.size());

			cvtColor(correctedImg, correctedImgBin, CV_RGB2GRAY);
			correctedImgBin.convertTo(correctedImgBin, -1, kontrast, 0);
			threshold(correctedImgBin, correctedImgBin, 140, 255, 0);
			minMaxLoc(correctedImgBin, &minVal, &maxVal);
			medVal = (maxVal - minVal) / 2;
			threshold(correctedImgBin, correctedImgBin, medVal, 255, 0);

			imshow("corrected", correctedImgBin);
```

```
                        min_non_zero = 12000;
                        match = -1;

                        for (j = 0; j < 8; j++) {
                                bitwise_xor(correctedImgBin, symbols[j].img, diffImg);
                                non_zero = countNonZero(diffImg);
                                if (non_zero < min_non_zero) {
                                        min_non_zero = non_zero;
                                        match = j;
                                }
                        }

                        if (match != -1) {
                                bitwise_xor(correctedImgBin, symbols[match].img, diffImg);
                                imshow("corrected", diffImg);
                                putText(frame, symbols[match].name, symbol_position,
                                         1, 2, color, 2);
                        }

                        for (j = 0; j < 4; j++) {
                                circle(frame, corners[j], 10, color, -1, 8, 0);
                                line(frame, corners[j], corners[(j + 1) % 4], color,
                                        1, 8);
                        }

                        break; //len 1 obj
                }
        }

        return match;
}
```

Modul ws.html – kód servera (súbor server.py)

```
import tornado.httpserver
import tornado.websocket
import tornado.ioloop
import tornado.web

import os
from subprocess import call
from time import sleep

import json

print '[websocket server]\n'
sb = open('/dev/servoblaster', 'w')

def sendpwm(pwm, pin):
        sb.write(str(pin) +"="+ str(pwm) + 'us\n')
        sb.flush()

class WSHandler(tornado.websocket.WebSocketHandler):
        def check_origin(self, origin):
                return True

        def open(self):
                print 'klient pripojeny\n'

        def on_message(self, message):
                pwm = json.loads(message)
                if pwm['servo']:
                        sendpwm(pwm['servo'], 7)
                if pwm['engine']:
                        sendpwm(pwm['engine'], 1)

        def on_close(self):
                print 'klient odpojeny\n'
                tornado.ioloop.IOLoop.instance().stop()

application = tornado.web.Application([(r'/ws', WSHandler),])

if __name__ == "__main__":
    http_server = tornado.httpserver.HTTPServer(application)
    http_server.listen(8888)
    tornado.ioloop.IOLoop.instance().start()
```

Modul ws.html – kód klienta (súbor ws.html)

```
var keys = {};
var learn_map = false
$(document).keydown(function (e) {
    keys[e.which] = true
    if (e.which == 83) { //klavesa S
           learn_map = !learn_map
           $("#learn_map").html(learn_map ? '[s]top learning' : '[s]tart learning')
    }
}).keyup(function (e) {
    delete keys[e.which];
});

var engine=1500, servo=1500
$("#engine").html(engine);
$("#servo").html(servo);

var int, i
var map_params = []
var params
function drive() {
    clearInterval(int)
    int = setInterval(function() {
           if (ws.readyState === 1) {
                  params = { engine: keys[87]? engine : 1500, servo: servo }
                  if (learn_map)
                         map_params.push(params)
                  else if (map_params[i] != undefined) {
                         params = map_params[i]
                         ++i
                  }
                  ws.send(JSON.stringify(params))
           }
    }, 50)
}

$("#stream").attr('src',"http://"+location.hostname+":9000/stream/video.mjpeg"
).load(function() {
    var dWidth = Number($("#stream").width());
    var dHeight = Number($("#stream").height());

    $("#dWidth").html(dWidth);
    $("#dHeight").html(dHeight);

    var mouseX = dWidth/2;
    var mouseY = dHeight/2;

    $("#stream").mousemove(function(e) {
           mouseX= Number(e.pageX);
           mouseY= Number(e.pageY);

           $("#mouseX").html(mouseX);
           $("#mouseY").html(mouseY);

           engine = 1000 + (1 - (mouseY / dHeight)) * 1000
           servo  = 1000 + (1 - (mouseX / dWidth )) * 1000
           engine = Math.ceil(engine / 10) * 10
           servo  = Math.ceil(servo  / 10) * 10

           if (engine>2000) engine=2000
           else if (engine<1000) engine=1000
```

```
            if (servo>2000) servo=2000
            else if (servo<1000) servo=1000

            $("#engine").html(engine);
            $("#servo").html(servo);
    });

    drive()
});

var ws = new WebSocket("ws://"+location.hostname+":8888/ws")
ws.onopen = function(evt) {
    console.log('pripojene')
}
ws.onmessage = function(evt) {
    //console.log('sprava:'+ evt.data)
}
ws.onclose = function(evt) {
    console.log('nepripojene')
    $('#client-status').html('disconnected')
}

$('#ws-close').click(function(){
    ws.close();
});
```

Kód modulu server.html

```
var server = new Peer('l3ve8x1wai82gldi', {key: '72rwkk1xno7xpqfr'});

server.on('open', function(id) {
    console.log('server id: ' + id);
});
server.on('close', function() {
    console.log('server closed')
});
server.on('disconnected', function() {
    console.log('server disconnected')
});

server.on('connection', function(connection) {
    connection.on('open', function() {
           console.log('client '+connection.peer+ ' pripojeny')
           connection.send('vitaj na serveri')
    });
    connection.on('data', function(data) {
           if (ws.readyState === 1)
                  ws.send(JSON.stringify({ engine: data.engine, servo: data.servo }))
    });
    connection.on('close', function() {
           console.log('client '+connection.peer+ ' odpojeny')
    });
    connection.on('error', function(err) {
           console.log(err.message)
    });
})

$('#server-disconnect').click(function(){
    server.disconnect();
});
$('#server-reconnect').click(function(){
    server.reconnect();
});
$('#server-destroy').click(function(){
    server.destroy();
});

var ws = new WebSocket("ws://"+location.hostname+":8888/ws")
ws.onopen = function(evt) {
    console.log('ws pripojene')
}
ws.onmessage = function(evt) {
    //console.log('sprava:'+ evt.data)
}
ws.onclose = function(evt) {
    console.log('ws nepripojene')
}

$('#ws-close').click(function(){
    ws.close();
});
```

Príloha C: Sprievodný materiál

Súčasťou práce je CD s kompletnými zdrojovými kódmi, vytvorenými aplikáciami a demonštračnými videami. Štruktúra CD:

```
+---videa
|       offboard - rozpoznavanie symbolov.mp4
|       offboard - sledovanie ciary.mp4
|       offboard - sledovanie lopty.mp4
|       onboard - sledovanie ciary.mp4
|
\---zdrojove kody
    +---html
    |   |   ajax.html
    |   |   client.html
    |   |   index.html
    |   |   ipcam.php
    |   |   ipcam.txt
    |   |   jquery-2.0.3.min.js
    |   |   jquery-2.1.1.min.js
    |   |   myServer.c
    |   |   offline.html
    |   |   peer.min.js
    |   |   rpi.php
    |   |   server.html
    |   |   server.py
    |   |   ws.html
    |   |   ws_video_off.html
    |   |
    |   \---img
    |           controls.png
    |
    +---offboard
    |   |   ConsoleApplication1.sdf
    |   |   rpi.sdf
    |   |   rpi.sln
    |   |
    |   +---ConsoleApplication1
    |   |       arrowB.jpg
    |   |       arrowGo.jpg
    |   |       arrowL.jpg
    |   |       arrowL45.jpg
    |   |       arrowR.jpg
    |   |       arrowR45.jpg
    |   |       arrowStop.jpg
    |   |       arrowT.jpg
    |   |       ConsoleApplication1.vcxproj
    |   |       ConsoleApplication1.vcxproj.filters
    |   |       ConsoleApplication1.vcxproj.user
    |   |       fcie.h
    |   |       minni.png
    |   |       Mslogo.avi
    |   |       onboard.cpp
    |   |       result.png
    |   |       settings.txt
    |   |       Source.cpp
    |   |
    |   \---Release
    |           arrowB.jpg
    |           arrowGo.jpg
```

```
|           arrowL.jpg
|           arrowL45.jpg
|           arrowR.jpg
|           arrowR45.jpg
|           arrowStop.jpg
|           arrowT.jpg
|           libcurl.dll
|           libeay32.dll
|           libssh2.dll
|           msvcr120.dll
|           rpi.exe
|           rpi.pdb
|           settings.txt
|           ssleay32.dll
|           zlib1.dll
|
\---onboard
    |   onboard.sdf
    |   onboard.sln
    |
    +---onboard
    |       arrowB.jpg
    |       arrowGo.jpg
    |       arrowL.jpg
    |       arrowL45.jpg
    |       arrowR.jpg
    |       arrowR45.jpg
    |       arrowStop.jpg
    |       arrowT.jpg
    |       fcie.h
    |       onboard.cpp
    |       onboard.vcxproj
    |       onboard.vcxproj.filters
    |       onboard.vcxproj.user
    |       result.png
    |       settings.txt
    |
    \---Release
            arrowB.jpg
            arrowGo.jpg
            arrowL.jpg
            arrowL45.jpg
            arrowR.jpg
            arrowR45.jpg
            arrowStop.jpg
            arrowT.jpg
            libcurl.dll
            libeay32.dll
            libssh2.dll
            msvcr120.dll
            onboard.exe
            onboard.pdb
            rpi.exe
            rpi.pdb
            ssleay32.dll
            zlib1.dll
```

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

RIADENIE RÁDIOVO OVLÁDANÉHO MODELU POČÍTAČOM A JEHO INTERAKCIA S VONKAJŠÍM SVETOM NA PLATFORME RASPBERRY PI

Používateľská príručka

2015 **Bc. Peter Otto**

Funkcia programu

Aplikácia je určená na riadenie RC modelu auta pomocou bežného stolového počítača alebo notebooku. Aplikácia umožňuje riadiť model manuálne alebo automaticky v režime sledovania čiary, lopty, alebo rozpoznávania symbolov. Používateľ má počas celej jazdy modelu možnosť sledovať na svojej obrazovke video prenos v reálnom čase z pohľadu RC auta. Základným predpokladom pre funkčnosť programu je vlastnenie počítača Raspberry Pi a Raspberry Pi kamery. Aplikácia má tri verzie: html, onboard a offboard. Verzia HTML je určená pre internetový prehliadač, verzia onboard pre beh na počítači Raspberry Pi a verzia offboard pre beh na klasickom počítači.

Inštalácia programu

1. Prekopírovanie adresárov html a offboard do adresára s právom na zápis.
2. Prekopírovanie adresára onboard na SD kartu počítača Raspberry Pi. Tento krok je voliteľný a vyžaduje inštaláciu knižnice OpenCV na počítači Raspberry Pi. Pre funkčnosť je potrebné program zostaviť. Na zostavenie môže byť použitý skript build_all.sh z adresára opencv-2.4.9/samples/c, ktorý je súčasťou knižnice (testovaná verzia 2.4.9). Skript skopírujeme do spoločného adresára so zdrojovými kódmi onboard verzie.
3. Spustenie aplikácie. Spúšťacím súborom HTML verzie je html/index.html. Pre aktiváciu offboard verzie použijeme súbor offboard/Release/rpi.exe. Onboard verzia sa aktivuje príkazom ./onboard.

Spôsob použitia aplikácie pri spustení:

- offboard verzia – rpi.exe [<rpi_ip> <h264|mjpeg>].
- onboard verzia – ./onboard <line|ai|symbol> <device nr.> [<debug>]

Opis parametrov:

- rpi_ip – adresa IP počítača Raspberry Pi.
- h264/mjpeg – voľba komprimačného algoritmu pre príjem streamu.
- line/ai/symbol – voľba funkcie automatického režimu; line – sledovanie čiary, ai – sledovanie lopty, symbol – rozpoznávanie symbolu.
- device nr. – index video zariadenia, predvolene 0.
- debug – režim s výpisom o činnosti.

Opis pracovných súborov

Konzolová aplikácia (onboard a offboard verzia) používa jeden konfiguračný súbor (txt), osem obrázkov (jpg) a šesť dynamicky odkazovaných knižníc (dll). Offboard verzia navyše používa jeden skript php a aplikáciu soket servera. Webová aplikácia (HTML verzia) používa šesť modulov (html), dve JavaScript knižnice (js), jeden skript php, jeden skript python (py) a jeden obrázok (png).

Konzolová aplikácia

- rpi.exe – spúšťací súbor offboard verzie.
- onboard – spúšťací súbor onboard verzie.
- settings.txt – konfiguračný súbor.
- arrowB.jpg, arrowGo.jpg, arrowL.jpg, arrowL45.jpg, arrowR.jpg, arrowR45.jpg, arrowStop.jpg, arrowT.jpg – obrázky podporovaných symbolov.
- libcurl.dll, libeay32.dll, libssh2.dll, msvcr120.dll, ssleay32.dll a zlib1.dll – dynamicky odkazované knižnice patriace knižnici Libcurl.
- myServer – soket server (beží na Raspberry Pi).
- rpi.php – spracúva AJAX a POST požiadavky (beží na Raspberry Pi).

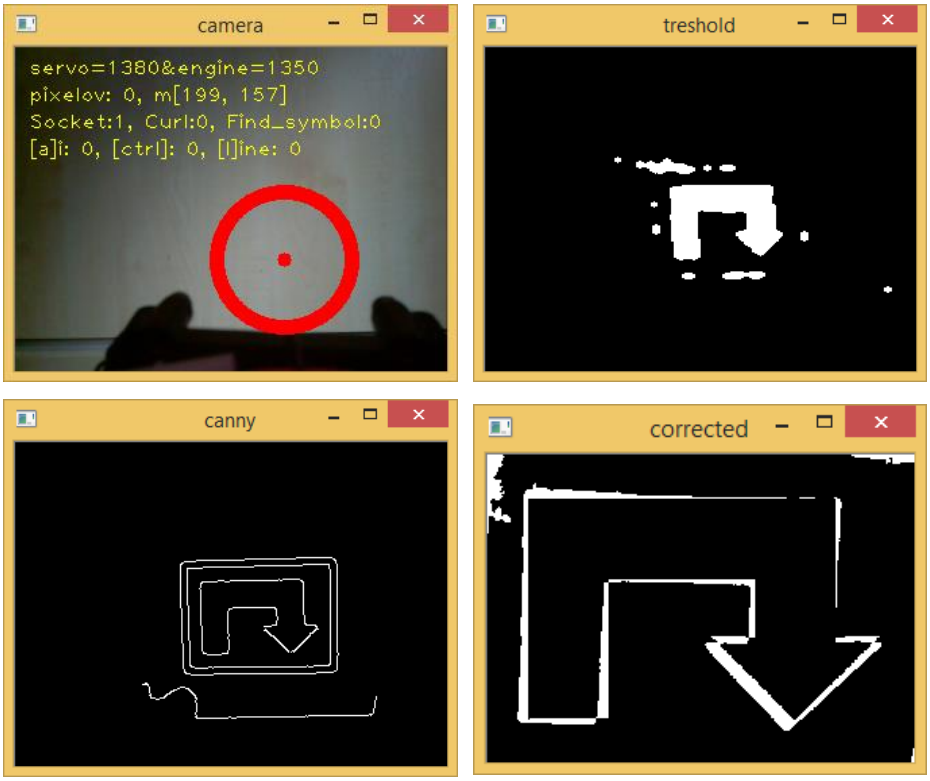
Webová aplikácia

- index.html – hlavná stránka s odkazmi na ostatné moduly.
- ws.html – modul riadenia cez webové sokety s video streamom.
- ws_video_off.html – rovnaký ako predošlý, bez video streamu.
- server.html – JavaScript server pre knižnicu PeerJS.
- client.html – JavaScript klient pre knižnicu PeerJS.
- ajax.html – modul riadenia cez AJAX s video streamom.
- jquery-2.1.1.min.js – knižnica jQuery.
- peer.min.js – knižnica PeerJS.
- rpi.php – spracúva AJAX a POST požiadavky (beží na Raspberry Pi).
- server.py – websoket server (beží na Raspberry Pi).
- img\controls.png – obrázok znázorňujúci spôsob ovládania myšou.

Grafické používateľské rozhranie programu

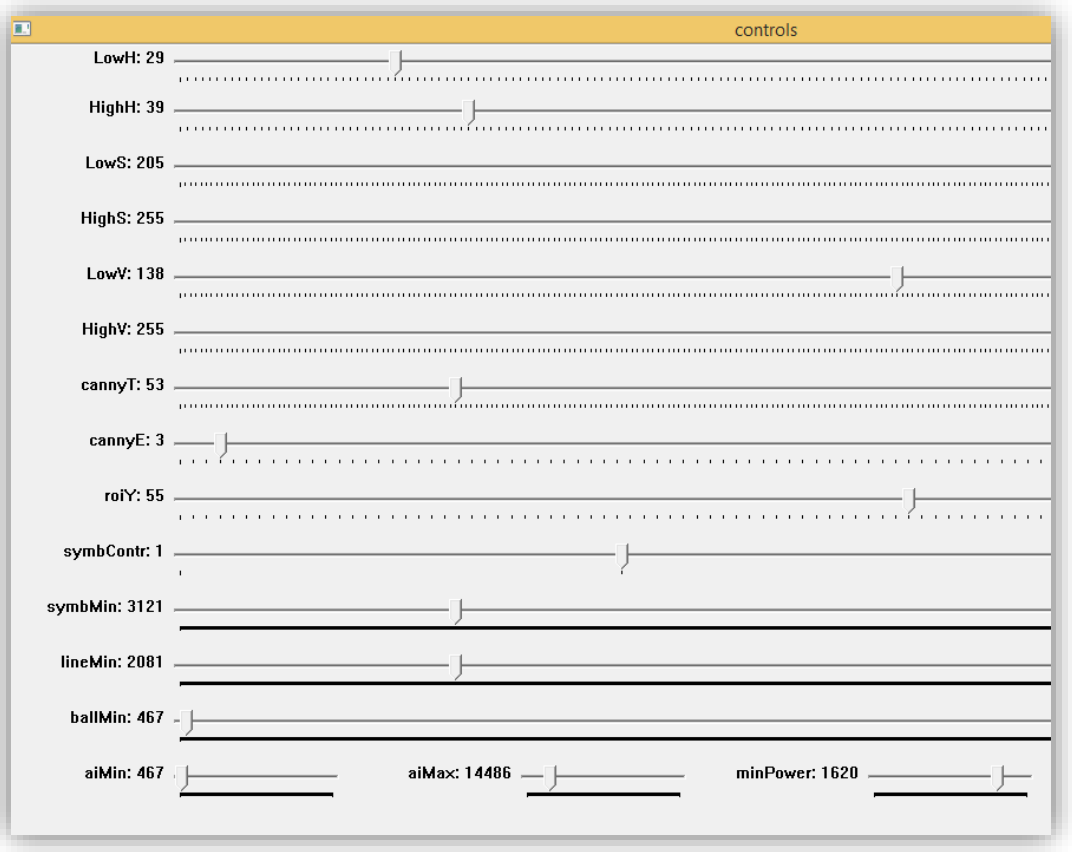
Grafickým rozhraním disponuje offboard verzia a webová (HTML) verzia aplikácie. V offboard verzii môžeme pracovať najviac s piatimi oknami, v závislosti od zvoleného režimu. Prvým oknom je okno streamu, ktoré zobrazuje pôvodný obraz z RC auta a informácie o aktuálnych parametroch. K informáciám patrí:

- servo/engine – hodnoty riadiacich príkazov serva a motora,
- pixelov – počet bielych bodov v okne prahovania (neaktívne),
- m[x,y] – súradnice kurzora vzhľadom k oknu streamu; poloha kurzora v okne je znázornená červeným kruhom,
- Socket – stav komunikácie cez sokety; ovládame klávesom S,
- Curl – stav komunikácie cez knižnicu Libcurl; ovládame klávesom C,
- Find_symbol – stav režimu rozpoznávania symbolov; ovládame klávesom F,
- ai – stav režimu sledovania lopty; ovládame klávesom A,
- ctrl – stav aktivácie pohonu; ovládame klávesom CTRL alebo ľavým tlačidlom myši,
- line – stav režimu sledovania čiary; ovládame klávesom L.



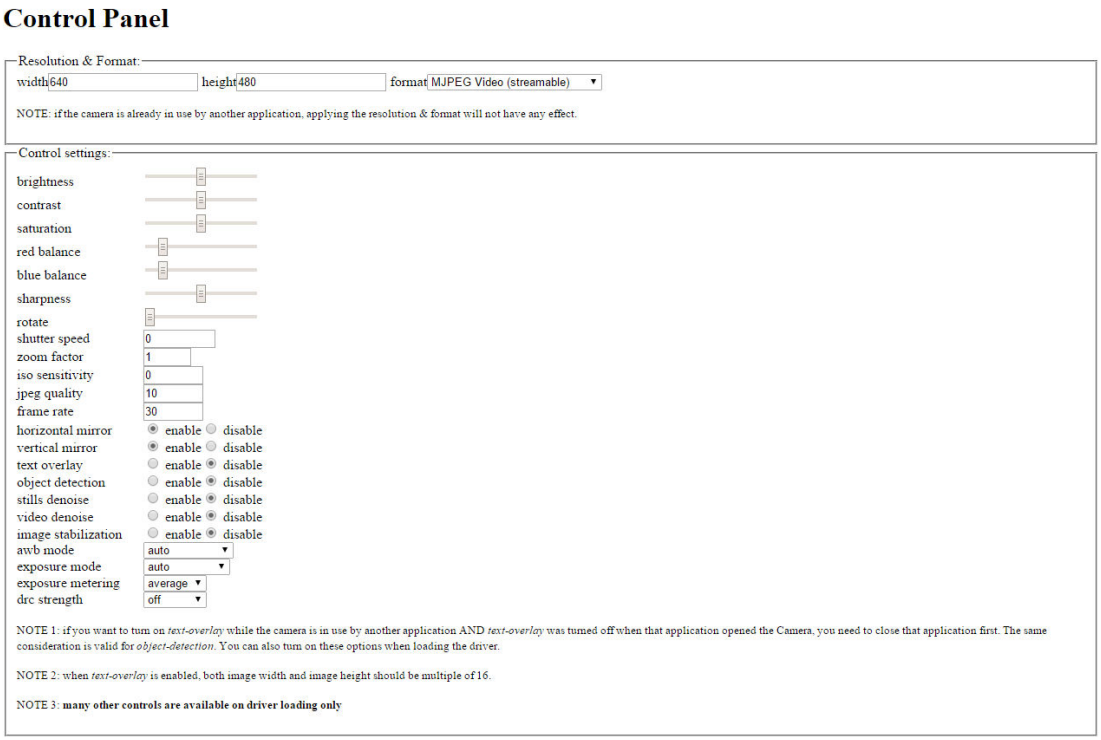
Obr. 1: Offboard verzia – vľavo hore okno streamu, vpravo hore okno prahovania, vľavo dole okno hrán, vpravo dole okno prekrytia

Druhým oknom je okno prahovania. Toto okno zobrazuje objekty segmentované na základe ich farby. Tretie okno – okno hrán, zobrazuje obrysy objektu z okna prahovania. Štvrtým oknom je okno prekrytia, ktoré zobrazuje úroveň prekrytia vzorového a porovnávaného symbolu. Posledným oknom je okno s konfiguračnými parametrami. Parametre je možné meniť myšou pomocou bežcov. Význam parametrov ozrejmuje Tabuľka 5 v prílohe A.



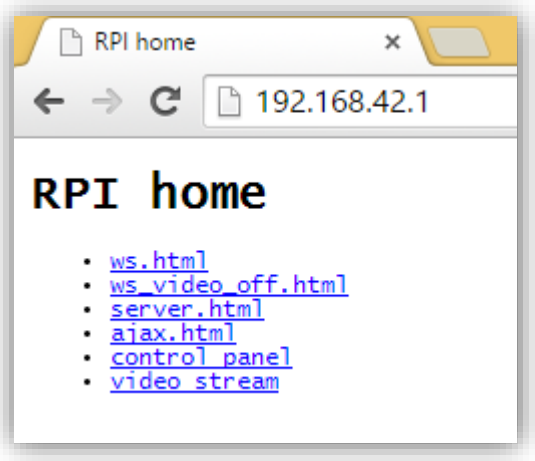
Obr. 2: Offboard verzia – okno s konfiguračnými parametrami

Konfigurácia video streamu je spoločná pre všetky varianty aplikácie (offboard, onboard aj HTML). Súčasťou HTTP servera UV4L, použitého na streamovanie z počítača Raspberry Pi, je ovládací panel (HTML stránka), pomocou ktorého je možné meniť parametre obrazu. K parametrom obrazu použiteľným v aplikácii patrí: rozlíšenie, výber komprimačného algoritmu, svetlosť, kontrast, sýtosť, vyváženie červenej, vyváženie modrej, ostrosť, rotácia, miera zväčšenia, citlivosť iso, úroveň kvality (jpeg), počet obrázkov za sekundu, vodorovné preklopenie, zvislé preklopenie, zjemnenie snímky, zjemnenie videa, stabilizácia obrazu, režim vyváženia bielej, režim expozície, režim merania expozície a úroveň dynamického rozsahu.



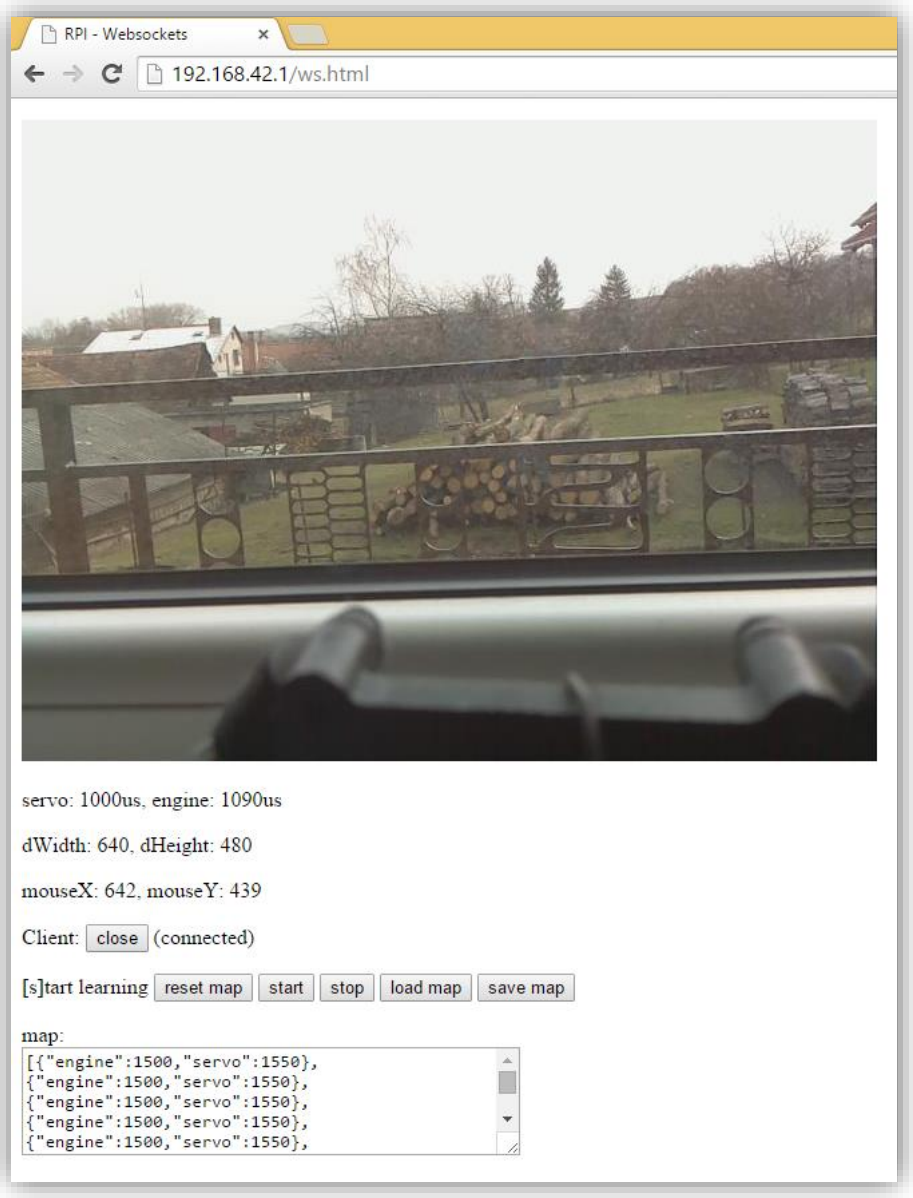
Obr. 3: Ovládací panel (modul *control panel*) – konfigurácia video streamu

V HTML verzii sa na úvodnej stránke (index.html) nachádza niekoľko odkazov pre prístup k jednotlivým modulom aplikácie. Stránka tiež obsahuje odkaz pre prístup do ovládacieho panela (control panel) a samostatného okna so streamom (video stream). V zozname odkazov nie je uvedený modul client.html, pretože môže byť umiestnený nezávisle od ostatných modulov.



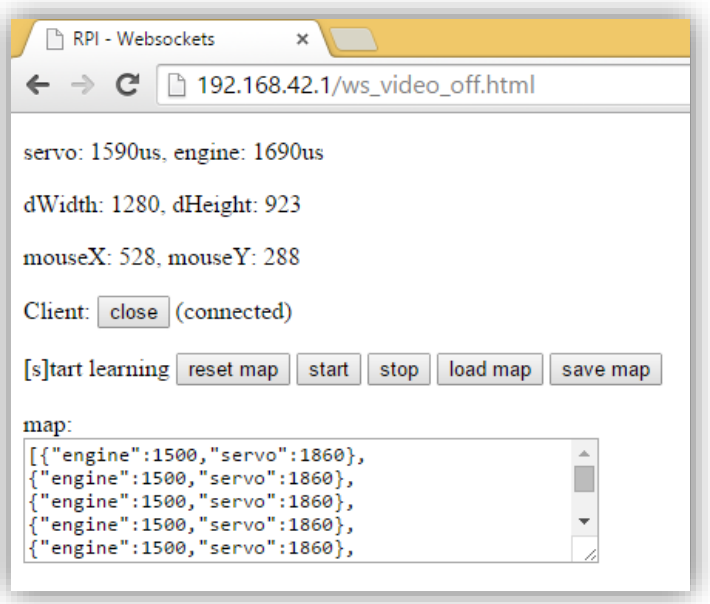
Obr. 4: Verzia HTML – hlavná stránka (modul index.html)

Prvým modulom v zozname odkazov je modul riadenia cez webové sokety s video streamom (obr. 5). Modul obsahuje tieto prvky: streamované video, riadiace parametre serva a motora, rozlíšenie streamu, súradnice kurzora vzhľadom k video streamu, stav spojenia k serveru a tlačidlo *close* na ukončenie spojenia, stav aktivácie memorizácie trasy, tlačidlo *reset map* na zmazanie uloženej trasy, tlačidlo *start* na vykonanie zaznamenanej trasy, tlačidlo *stop* na zastavenie vykonávania zaznamenanej trasy, tlačidlo *load map* na načítanie trasy z textového poľa, tlačidlo *save map* na uloženie zaznamenanej trasy do textového poľa, textové pole pre zobrazovanie riadiacich príkazov (v formáte JSON). Funkcia memorizácie trasy sa aktivuje/deaktivuje klávesom S. Na aktiváciu pohonu slúži kláves W.



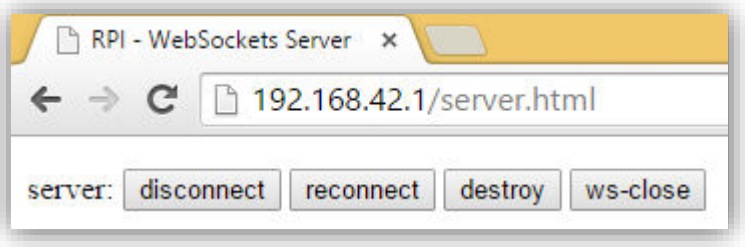
Obr. 5: Verzia HTML – modul ws.html

Druhým modulom je modul riadenia cez webové sokety bez video streamu (obr. 6). Modul obsahuje (s výnimkou streamovaného videa) rovnaké prvky ako predchádzajúci modul a pracuje sa s ním rovnakým spôsobom. Súradnice kurzora sa v tomto prípade vzťahujú k rozmerom okna internetového prehliadača.



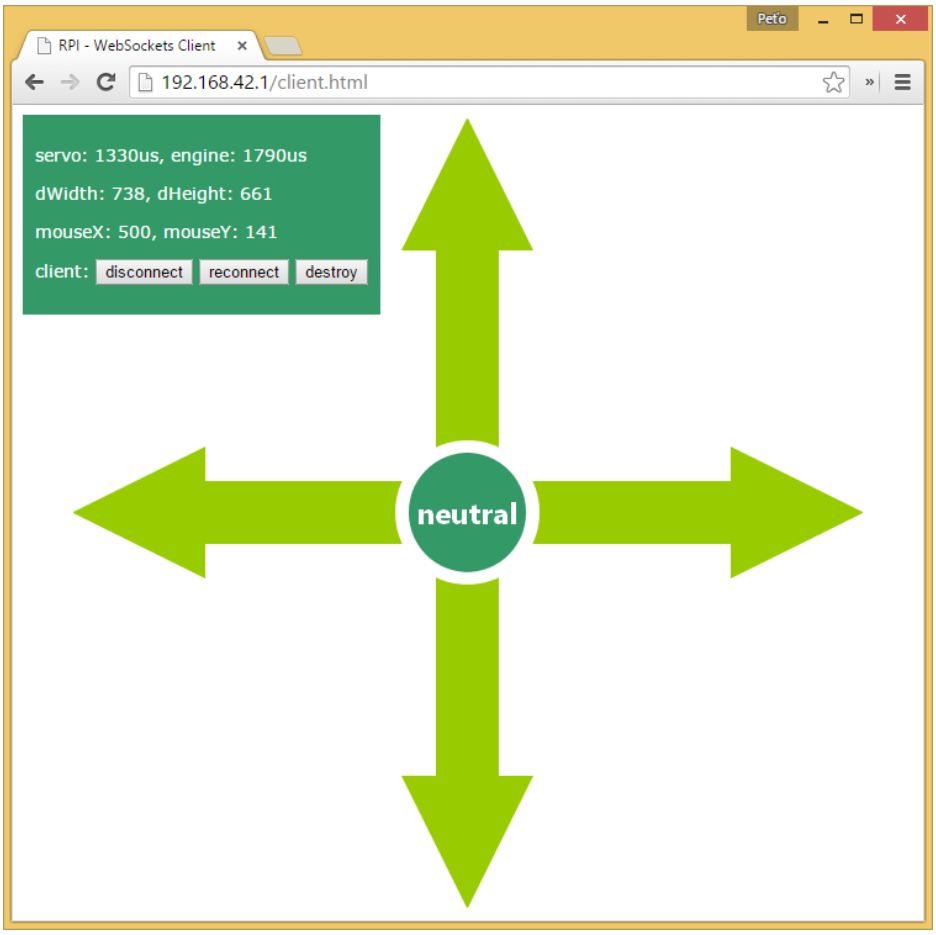
Obr. 6: Verzia HTML – modul ws_video_off.html

Tretí modul – server pre knižnicu PeerJS (obr. 7), obsahuje štyri ovládacie prvky: tlačidlo *disconnect* na odpojenie z PeerJS servera, tlačidlo *reconnect* na obnovenie spojenia s PeerJS serverom, tlačidlo *destroy* na ukončenia spojenia s PeerJS serverom, tlačidlo *ws-close* na ukončenie spojenia s websoket serverom. Modul server.html je potrebné aktivovať pred aktivovaním modulu client.html. Pre správnu funkciu modulu server.html je potrebné pred zatvorením okna prehliadača ukončiť spojenie s PeerJS serverom (tlačidlo *destroy*).



Obr. 7: Verzia HTML – modul server.html

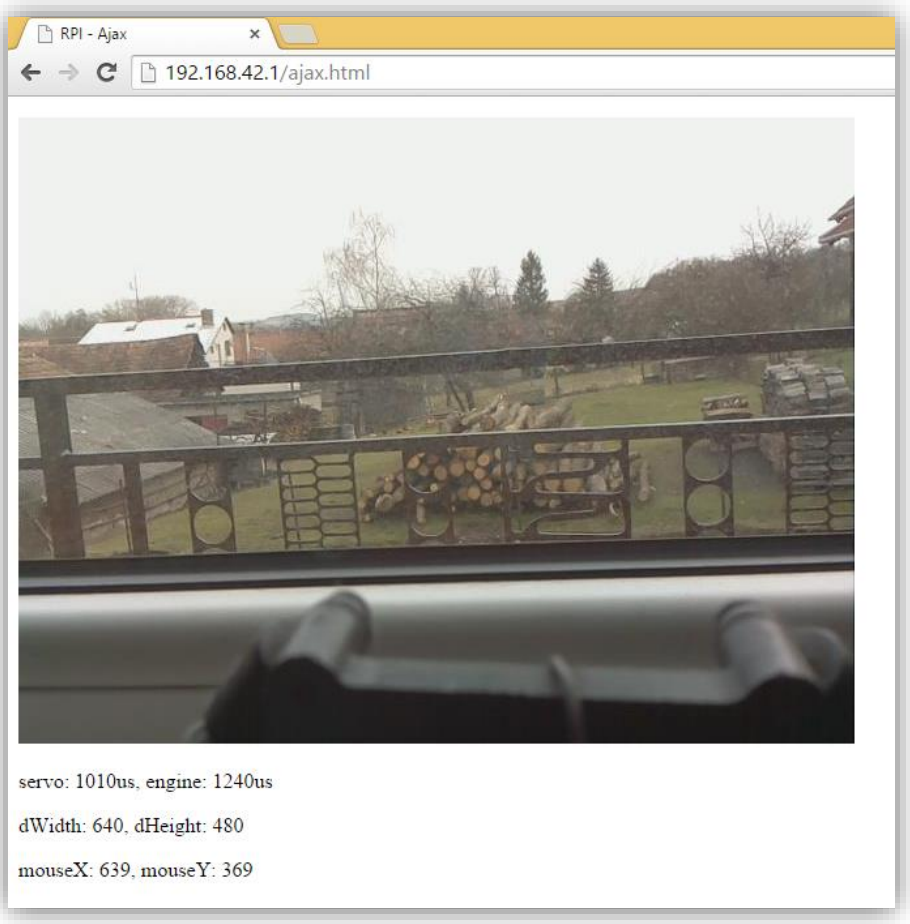
Klient pre knižnicu PeerJS (obr. 8), obsahuje tri ovládacie prvky: tlačidlo *disconnect* na odpojenie z PeerJS servera, tlačidlo *reconnect* na obnovenie spojenia s PeerJS serverom, tlačidlo *destroy* na ukončenia spojenia s PeerJS serverom. Súčasťou modulu sú opäť informácie o riadiacich príkazoch serva a motora, rozmery okna a pozícia kurzora vzhľadom k oknu prehliadača. V pozadí stránky sa nachádza obrázok znázorňujúci spôsob riadenia RC modelu. Pohon aktivujeme klávesom W.



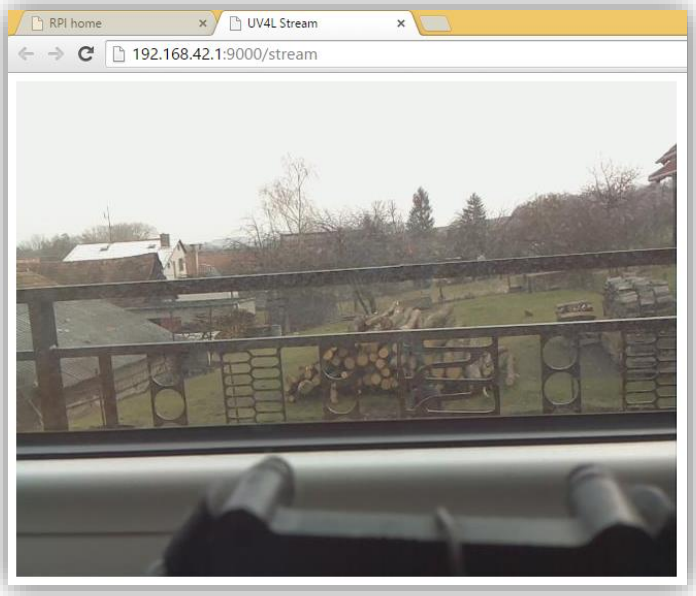
Obr. 8: Verzia HTML – modul client.html

Posledným modulom je modul riadenia cez AJAX s video streamom (obr. 9). Modul okrem streamu obsahuje iba základné informácie – hodnoty riadiacich príkazov, rozmery streamu a súradnice kurzora vzhľadom k streamu.

Obrázok 10 znázorňuje samostatné okno so streamom, ktoré nemá žiadnu špecifickú funkciu. Okno sa používa iba na overenie nastavení konfiguračných parametrov a funkčnosti streamovania z počítača Raspberry Pi do PC klienta.



Obr. 9: Verzia HTML – modul ajax.html



Obr.10: Verzia HTML – samostatné okno so streamom

Príklady použitia

V nasledovných príkladoch pracujeme s IP adresou 192.168.42.1, ktorá v našom prípade prislúcha počítaču Raspberry Pi s WiFi adaptérom v režime prístupového bodu. Príklady offboard verzie nezahŕňajú príkaz na vytvorenie soket servera (./myServer). Server sa aktivuje automaticky pri štarte počítača Raspberry. Výpis činnosti konzolovej aplikácie (onboard aj offboard verzia) je dostupný v termináli. Výpis činnosti webovej stránky (HTML verzia) je dostupný v sekcii Nástroje pre vývojarov – Konzola (v prehliadači Google Chrome stlačme kláves F12).

Príklad 1: Manuálne riadenie cez AJAX – verzia HTML

1. Otvorenie úvodnej stránky na adrese http://192.168.42.1.
2. Konfigurácia streamu v sekcii control panel.
3. Aktivácia modulu ajax.html (z úvodnej stránky).
4. Kláves W – aktivácia pohonu.
5. Zatvorenie prehliadača – koniec programu.

Príklad 2: Manuálne riadenie cez internet – verzia HTML

Príklad vyžaduje pripojenie k internetu u PC i Raspberry Pi.

1. Otvorenie úvodnej stránky na adrese http://192.168.42.1.
2. Konfigurácia streamu v sekcii control panel.
3. Aktivácia websoket servera v termináli RPi – príkaz *python server.py*.
4. Aktivácia modulu server.html (z úvodnej stránky).
5. Aktivácia modulu client.html (odkiaľkoľvek).
6. Kláves W – aktivácia pohonu.
7. Ukončenie spojenia klienta s serverom (server.html) – tlačidlo destroy.
8. Ukončenie spojenia (server.html) s P2P serverom – tlačidlo destroy.
9. Ukončenie spojenia (server.html) s websoket serverom – tlačidlo ws-close.
10. Zatvorenie prehliadača – koniec programu.

Príklad 3: Memorizácia trasy – verzia HTML

1. Otvorenie úvodnej stránky na adrese http://192.168.42.1.
2. Konfigurácia streamu v sekcii control panel.
3. Aktivácia websoket servera v termináli RPi – príkaz *python server.py*.
4. Aktivácia modulu ws.html (z úvodnej stránky).

5. Vyčistenie trasy (voliteľné) – tlačidlo reset map.
6. Kláves S – aktivácia memorizácie trasy.
7. Kláves W – aktivácia pohonu.
8. Kláves S – deaktivácia memorizácie trasy.
9. Zobrazenie uložených príkazov trasy (voliteľné) – tlačidlo save map.
10. Vykonanie zaznamenanej trasy (aktivácia pohonu) – tlačidlo start.
11. Zastavenie vykonávania zaznamenanej trasy – tlačidlo stop.
12. Ukončenie spojenia s websoket serverom – tlačidlo close.
13. Zatvorenie prehliadača – koniec programu.

Príklad 4: Sledovanie čiary – verzia offboard

1. Konfigurácia streamu na adrese http://192.168.42.1:9000/panel.
2. Aktivácia aplikácie v tvare: rpi.exe 192.168.42.1 h264.
3. Kláves F – vypnutie predvoleného režimu (rozpoznávanie symbolov).
4. Kláves L – zapnutie režimu sledovania čiary.
5. Nastavenie parametrov v okne controls.
6. Je čiara správne rozpoznaná?
    a. Áno – prejsť na krok č. 7.
    b. Nie – prejsť na krok č. 5.
7. Kláves CTRL alebo ľavé tlačidlo myši – aktivácia pohonu.
8. Kláves ESC – koniec programu.

Príklad 5: Sledovanie čiary – verzia onboard

1. Konfigurácia streamu na adrese http://192.168.42.1:9000/panel.
2. Konfigurácia parametrov – editácia súboru settings.txt. Vyhovujúce hodnoty parametrov je možné zistiť z grafického rozhrania offboard verzie.
3. Aktivovanie aplikácie v tvare: ./onboard line 0 debug.
4. Sleduje model čiaru?
    a. Áno – prejsť na krok č. 5.
    b. Nie – prejsť na krok č. 5 a následne na krok č. 2.
5. Kláves CTRL+C – koniec programu.

Príklad 6: Rozpoznávanie symbolov – verzia offboard

1. Konfigurácia streamu na adrese http://192.168.42.1:9000/panel.
2. Aktivovanie aplikácie v tvare: rpi.exe 192.168.42.1 h264.

3. Kláves S – deaktivácia odosielania riadiacich príkazov cez sokety.
4. Nastavenie parametrov v okne controls.
5. Je symbol správne rozpoznaný?
    a. Áno – prejsť na krok č. 6.
    b. Nie – prejsť na krok č. 4.
6. Kláves S – aktivácia odosielania riadiacich príkazov cez sokety.
7. Kláves ESC – koniec programu.

Príklad 7: Zmena spôsobu zasielania riadiacich príkazov – offboard verzia

1. Konfigurácia streamu na adrese http://192.168.42.1:9000/panel.
2. Aktivovanie aplikácie v tvare: rpi.exe 192.168.42.1 h264.
3. Kláves S – deaktivácia odosielania riadiacich príkazov cez sokety.
4. Kláves C – aktivácia odosielania riadiacich príkazov cez HTTP požiadavky (knižnica Libcurl).
5. Kláves ESC – koniec programu.

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

RIADENIE RÁDIOVO OVLÁDANÉHO MODELU POČÍTAČOM A JEHO INTERAKCIA S VONKAJŠÍM SVETOM NA PLATFORME RASPBERRY PI

Systémová príručka

2015 **Bc. Peter Otto**

Funkcia programu

Navrhovaný program umožňuje riadiť RC model auta pomocou počítača. Riadenie môže byť manuálne alebo automatické. V režime manuálneho riadenia počítač obsluhuje človek. V režime automatického riadenia sú využité algoritmy počítačového videnia, vďaka ktorým program poskytuje nasledovné funkcie:

1. sledovanie lopty,
2. sledovanie čiary,
3. rozpoznávanie symbolov.

Analýza riešenia

Úlohou navrhovanej aplikácie je poskytnúť používateľovi rozhranie medzi bežným počítačom a počítačom Raspberry Pi – na riadenie rádiovo ovládaného modelu RC auta. Aplikácia je implementovaná dvoma spôsobmi – ako konzolová aplikácia a webová stránka. Konzolová aplikácia má dve verzie – offboard a onboard. Verzia offboard je určená pre beh na PC klienta, onboard verzia je určená pre beh na počítači Raspberry Pi. Verzia onboard podporuje iba režimy automatického riadenia. Webová stránka podporuje iba režim manuálneho ovládania, ktorý rozširuje o funkciu zapamätávania trasy.

Aplikácia sa v oboch vyhotoveniach ovláda počítačovou myšou a klávesnicou. Myš sa používa na riadenie smeru jazdy, klávesnica slúži na aktiváciu pojazdu. Súčasťou oboch verzií aplikácie je okno s video streamom z pohľadu RC auta.

Konzolová aplikácia je vytvorená pomocou vývojového prostredia Microsoft Visual Studio, v programovacom jazyku C++. Webová aplikácia je vytvorená pomocou programu Adobe Dreamweaver, v značkovacom jazyku HTML a skriptovacom jazyku JavaScript.

Návrh riešenia

Konzolová aplikácia používa grafické používateľské rozhranie, ktoré pozostáva z niekoľkých okien. V oknách sú zobrazované jednotlivé fázy rozpoznávania objektov a konfiguračné parametre. Konfiguračné parametre sú načítavané z textového súboru umiestneného v spoločnom adresári aplikácie. Navrhovanú aplikáciu je možné ovládať klávesmi a myšou. Klávesy slúžia na zmenu aktívneho režimu, spôsobu

komunikácie s počítačom Raspberry Pi a aktiváciu pohonu RC modelu. Myš sa používa na zmenu smeru jazdy a interaktívnu editáciu hodnôt konfiguračných parametrov. Všetky podporované funkcie automatického riadenia sú založené na rovnakom princípe, ktorý spočíva v segmentácii farieb a následnej separácii vybraných objektov podľa vopred definovaných kritérií. Konzolová aplikácia využíva na zasielanie riadiacich príkazov cez bezdrôtovú sieť TCP/IP sokety (knižnica Winsock) alebo HTTP požiadavky (knižnica Libcurl). Požiadavky HTTP spracováva na strane servera skript PHP, ktorý pre svoju činnosť vyžaduje inštaláciu PHP servera. Onboard variant konzolovej aplikácie odosiela príkazy priamo, pretože beží lokálne na počítači Raspberry Pi. Onboard verzia tiež neobsahuje grafické používateľské rozhranie, ovláda sa výlučne cez príkazový riadok.

Webová implementácia je realizovaná sústavou statických HTML stránok, ktoré predstavujú jednotlivé moduly. Moduly sú diferencované podľa podpory streamovania videa, technológie odosielania riadiacich príkazov a dodatočnej funkcionality. Verzia HTML podporuje režim manuálneho riadenia a dve doplnkové funkcie – memorizáciu trasy a ovládanie cez internet. Na zasielanie riadiacich príkazov cez WiFi sieť sú použité webové sokety alebo AJAX požiadavky. Príjem správ odoslaných webovými soketmi na strane servera (Raspberry Pi) zabezpečuje serverová aplikácia implementovaná vo frameworku Tornado. Správy odoslané prostredníctvom AJAX požiadaviek sú spracované rovnakým spôsobom ako v prípade konzolovej aplikácie, teda PHP skriptom.

Opis funkcií a procedúr

int vytvor_soket()

Funkcia vytvorí TCP/IP soket pre komunikáciu so serverovou aplikáciou. V prípade neúspešného vytvorenia soketu funkcia vypíše chybové hlásenie a skončí s návratovou hodnotou 1.

Parametre:

- const char*ip – IP adresa cieľového uzla.
- SOCKET &novy_soket – vytvorený TCP/IP soket (výstup).

int readRefImages()

Funkcia načíta do pamäte zoznam podporovaných symbolov, prevedie ich do odtieňov šedej, aplikuje prvotné prahovanie a nastaví načítaným symbolom názov. Návratovou hodnotou je 1 v prípade úspechu, inak 0.

Parametre:

- Symbol *symbols – výstupné pole podporovaných symbolov.

void sortCorners()

Úlohou funkcie je identifikovať polohu vrcholov vstupnej kontúry a zoradiť ich v správnom poradí. Poloha vrcholov sa určuje podľa známeho stredu kontúry.

Parametre:

- vector<Point2f>& corners – vstupný/výstupný vektor vrcholov kontúry.
- Point2f center – súradnice stredu kontúry.

int findSymbol()

Funkcia hľadá symbol v snímke. Návratovou hodnotou je index nájdeného symbolu vo vektore symbolov načítaných funkciou readRefImages. V prípade neúspešného hľadania funkcia vracia hodnotu -1.

Parametre:

- Mat frame – vstupná snímka.
- Symbol *symbols – vektor načítaných symbolov.
- int edgeThresh – hodnota prahovania pri hľadaní hrán.
- int kontrast – kontrast snímky pred prahovaním.
- int contour_area – minimálna plocha obrysu pre rozpoznanie symbolu.

void lineFollow()

Funkcia hľadá v snímke čiaru. Oblasť hľadania je vymedzená parametrom roiR. Ak funkcia nájde kontúru zodpovedajúcu čiare, nastaví výstupné parametre (súradnice stredu kontúry, plochu kontúry) a vykreslí kontúru do snímky.

Parametre:

- Mat &frame – vstupná/výstupná snímka.
- Scalar LowHSV – skalár hodnôt pre dolnú hranicu prahovania.
- Scalar HighHSV – skalár hodnôt pre hornú hranicu prahovania.
- Rect roiR – oblasť záujmu v snímke.
- Point2f& roi_center – súradnice stredu nájdenej kontúry (výstup).
- double &area – plocha obrysu čiary (výstup).
- int min_area – minimálna plocha obrysu pre rozpoznanie čiary.

void trackBall()

Funkcia hľadá v snímke loptu. Ak funkcia nájde kontúru zodpovedajúcu lopte, nastaví výstupné parametre (súradnice stredu kontúry, plochu kontúry) a vykreslí kontúru do snímky.

Parametre:

- Mat frame – vstupná/výstupná snímka.
- Scalar LowHSV – skalár hodnôt pre dolnú hranicu prahovania.
- Scalar HighHSV – skalár hodnôt pre hornú hranicu prahovania.
- Point2f& roi_center – súradnice stredu nájdenej kontúry (výstup).
- double &area – plocha obrysu lopty (výstup).
- int min_area – minimálna plocha obrysu pre rozpoznanie lopty.

void CallBackFunc()

Funkcia zachytáva udalosti myši. Pri pohybe myši nastaví aktuálnu polohu kurzora do globálnej premennej.

Parametre:

- int event – typ vyvolanej udalosti (kliknutie, pohyb).
- int x – aktuálna horizontálna pozícia kurzora.
- int y – aktuálna vertikálna pozícia kurzora.
- int flags – konfiguračný parameter (príznak).
- void* userdata – ukazovateľ na premennú so súradnicami kurzora (výstup).

void drive()

Funkcia na základe indexu rozpoznaného symbolu vyvolá požadovaný manéver, napríklad zatočenie modelu, jazda vpred, zastavanie.

Parametre:

- int symbol_id – index rozpoznaného symbolu.
- SOCKET socket – inštancia TCP/IP soketu.
- VideoCapture cap – inštancia triedy pre manipuláciu s obrazom.
- int minPower – minimálna šírka pulzu ovplyvňujúca rýchlosť motora.
- Mat &frame – aktuálna snímka z kamerového modulu.

Technické a programové požiadavky pri preklade

Tab. 1: Technické požiadavky pri preklade

	Minimálne požiadavky	*Odporúčané požiadavky*
Procesor	Intel Atom Quad Core Z3775 @ 1,46 Ghz	Intel Pentium G3258 @ 3,20 GHz
Pamäť	2 GB	8 GB
Grafická karta	Integrovaná Intel® HD Graphics	MSI R9 270X
Pevný disk	154 MB voľného miesta	300 MB voľného miesta

Tab. 2: Programové požiadavky pri preklade

Operačný systém	Windows 8.1 64-Bit
Vývojové prostredie	Microsoft Visual Studio Express 2013 for Desktop
Konfigurácia prostredia	Pozri Tabuľka 4, príloha A

Nadväznosť na iné programové produkty

Vytvorená aplikácia využíva knižnicu OpenCV (manipulácia s obrazom), knižnicu Libcurl (prenos údajov), knižnicu PeerJS (vytváranie P2P spojení), aplikáciu ServoBlaster (riadenie serv a elektronických rýchlostných regulátorov), framework UV4L (streamovanie videa), framework Tornado (podpora webových soketov) a knižnicu Winsock (podpora klasických TCP/IP soketov).