UNIVERZITA MATEJA BELA
Fakulta prírodných vied

PRÍRODOVEDEC 2015

Zborník príspevkov zo ŠVK 2015
konanej 23. apríla 2015

BELIANUM
2015

Prírodovedec 2015
Zborník príspevkov zo ŠVK 2015 konanej 23. apríla 2015

Odborný garant:
prof. RNDr. Ján Spišiak, DrSc.

Editori:
Ing. Slavka Račáková, PhD.
prof. RNDr. Ján Spišiak, DrSc.

Zborník neprešiel jazykovou úpravou

ISBN 978-80-557-0943-7

OBSAH

Fakulta prírodných vied Univerzity Mateja Bela

ŠVK 2015

SEKCIA INFORMATIKA

Recenzenti:
Mgr. Peter TRHAN, PhD.
Ing. Jana JACKOVÁ, PhD.
RNDr. Miroslav MELICHERČÍK, PhD.
Mgr. Michal VAGAČ, PhD.

Riadenie rádiovo ovládaného modelu počítačom a jeho interakcia s vonkajším svetom na platforme Raspberry Pi

Peter OTTO[*], Peter TRHAN[**]

Katedra informatiky FPV UMB, Tajovského 40, 974 01 Banská Bystrica,
[*]e-mail: potto@azet.sk, [**]e-mail: peter.trhan@umb.sk

Abstract: Driving radio controlled model through a computer and its interaction with the outside world on the Raspberry Pi platform: *The paper 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 paper 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.*

Keywords: *Raspberry Pi, OpenCV, ServoBlaster, Web Sockets*

Úvod

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.

Cieľom tejto práce je prestavba rádiovo ovládaného modelu auta na model riadený počítačom Raspberry Pi. Práca analyzuje spôsob prenosu obrazu s Raspberry Pi kamerou a metódy 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 (ďalej OS) Raspbian, aplikácia ServoBlaster, framework UV4L, framework Tornado a vývojové prostredie Microsoft Visual Studio.

Zostavenie RC modelu

Model bude prestavbou už existujúceho rádiovo ovládaného modelu auta na model ovládaný počítačom Raspberry Pi (RPi) prostredníctvom WiFi siete. Naša prestavba zahŕňa inštaláciu RPi, kamerového modulu, bezdrôtového adaptéra a mobilného napájania. Kamerový modul použijeme na prenos obrazu z pohľadu prvej osoby (FPV) a v reálnom čase, neskôr tiež na rozpoznávanie obrazu a interakciu s okolitými objektmi.

Základnú časť zostavy tvorí rádiovo ovládané (ďalej RC) auto HPI Brama 10B RTR Buggy s motorom na jednosmerný prúd (ďalej DC). V aute 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.

Počítač RPi sa hodí na stavbu multimediálneho PC (HTPC), elektronické projekty a tiež bežnú kancelársku prácu. Vďaka architektúre SOC (angl. system on chip) 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+. Model B sa vyznačuje bohatšou výbavou, vyššou spotrebou aj cenou.

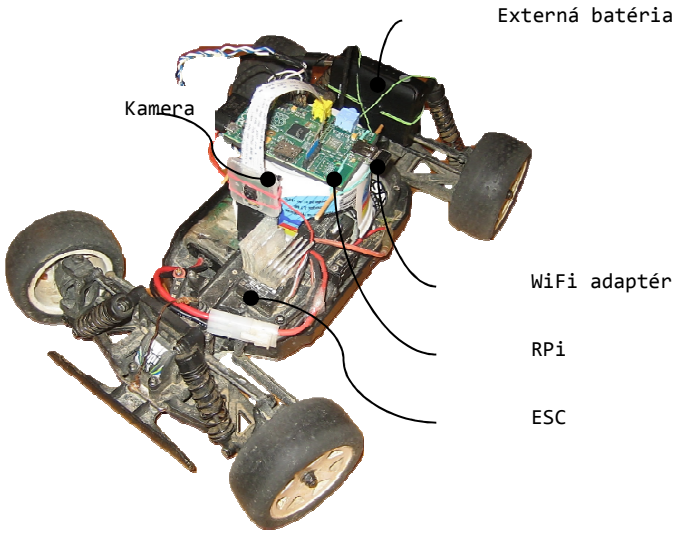
Spoločnými prvkami základných verzií A a B sú: kompozitný video výstup, stereo audio výstup, skupina LED signalizačných diód, HDMI výstup, napájací konektor (micro USB), čítačka SD pamäťových kariet, GPIO (angl. General-purpose input/output) piny a SOC čip BCM2835 architektúry ARM združujúci pamäť RAM, 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 0.

Pri riadení RC modelu očakávame, že jeho dojazd nebude obmedzovaný dĺžkou napájacieho kábla. Na napájanie RC modelu používame NiMH batériu Orion s vysokým vybíjateľným prúdom. Mobilné napájanie RPi môžeme zrealizovať napríklad pomocou 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. Akumulátor dokáže uchovať 26 Wh (5 V × 5,2 Ah) energie. Na nabíjanie slúži micro USB port. Výstupný (klasický) USB port vie dodať maximálne 2,1 A. Prevádzková doba batérie vychádza s uvedenými parametrami na 5,2 hod.

V tejto práci z dôvodu redukcie textu neuvádzam postup inštalácie OS na RPi, inštalácie bezdrôtového adaptéra TP-LINK TL-WN725N a konfigurácie bezdrôtovej komunikácie, ani postup aktivácie RPi kamery. Tieto informácie je možné nájsť v mojich starších článkoch 00.

Uchytenie komponentov

V ďalšom kroku na auto uchytíme RPi, kameru a externú batériu. Výsledné zostavenie znázorňuje obr. 1. Obrázok zahŕňa aj zapojenie vodičov serva a ESC, ktorému sa budeme venovať neskôr.



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.

ServoBlaster a prepojenie RC modelu s RPi

ServoBlaster (ďalej SB) je ovládač jadra OS, ktorý nám 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. Postup inštalácie SB a doplňujúce informácie je možné nájsť v staršej práci 0.

Revíziu RPi dosky (PCB) a číselné mapovanie pinov, na ktoré budeme odosielať signály, zistíme pri štarte SB. Naša doska je 2. revízie a má 2 × 3V3, 2 × 5V, 5 × uzemnenie (angl. ground) a 17 signálnych pinov. 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 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 0.

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.

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. Pre riadenie ESC nahradíme vo vyššie uvedenom príkaze číslo 7 číslom 1.

Streamovací server HTTP pre UV4L

Jednoduchý framework UV4L poskytuje ovládače tretej strany pre video zariadenia kompatibilné s oficiálnym V4L2[1]. Streamovací server je prídavným modulom pre UV4L. Server umožňuje streamovať video z RPi 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 0:

```
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*.

Detailnejšie nastavenia servera budeme konfigurovať cez webové rozhranie. Ak má napríklad RPi adresu IP v tvare *192.168.42.1*, rozhranie je 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*

[1] [1] http://en.wikipedia.org/wiki/Video4Linux

mirror. Ostatné prepínacie tlačidlá deaktivujeme. Parameter *jpeg quality* nastavíme na hodnotu 10. Zmeny aplikujeme tlačidlom *Apply*.

Adresa video streamu je *http://192.168.42.1:9000/stream*. Vyťaženie siete silne závisí od nastavenej kvality obrazu, čo sa prejavilo aj na dátovom toku. S uvedeným nastavením odozva dosahovala zhruba 320 ms.

Aplikácia autonómneho riadenia

Naša aplikácia bude podporovať štyri hlavné režimy – manuálne riadenie, sledovanie lopty, sledovanie čiary a 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 klávesa CTRL alebo W, alebo ľavého tlačidla myši počas regulácie rýchlosti.

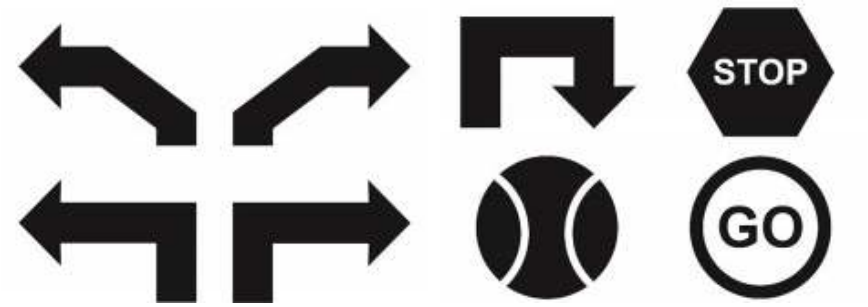
Sledovanie lopty, čiary a rozpoznávanie symbolov využíva knižnicu OpenCV a pracuje na princípe prahovania farby (angl. thresholding). U sledovania lopty na scénu umiestnime homogénny objekt – v našom prípade ním bude červená fitlopta. Potom pomocou metódy prahovania odfiltrujeme farbu lopty. Vznikne 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 (angl. 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. Obrázok 2 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[2] (XOR) medzi vzorovým a porovnávaným symbolom. Prekrývajúcim bodom s rovnakou hodnotou sa priradí 0 (čierna), ostatným bodom hodnota 255 (biela). Výstupom funkcie je binárna matica (ďalej diferenčná matica). 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 iných známych metód ako SURF[3] a SIFT[4] nie je z hľadiska zvýšených výpočtových nárokov pre RPi vhodné.



Rozpoznávanie symbolov – podporované vzory 0

[2] [2] http://cs.wikipedia.org/wiki/Bitový_operátor

[3] [3] http://goo.gl/4gKXRP

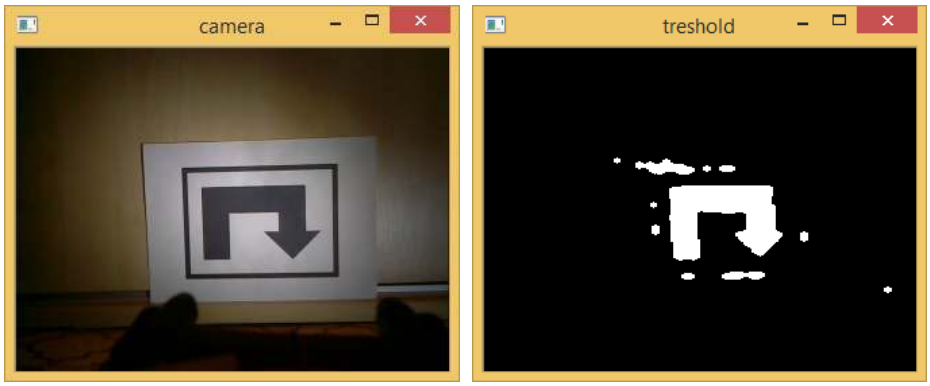
[4] [4] http://goo.gl/4cUuYP

Aplikácia je implementovaná ako konzolová aplikácia a webová stránka. Implementácia prostredníctvom webovej stránky (HTML verzia) podporuje z vyššie uvedených režimov iba prvý z nich – manuálne ovládanie. Obsahuje však doplnkovú funkciu – memorizáciu trasy. Zapamätávanie trasy pracuje na princípe uchovávania aktuálnych hodnôt riadiacich príkazov v pravidelných časových intervaloch. To umožňuje spätne zrekonštruovať trasu jazdy.

Konzolová aplikácia na zasielanie riadiacich príkazov do RPi využíva TCP/IP sokety na báze modelu klient-server. Verzia HTML používa webové sokety[5] (WS). Neskôr si ukážeme, že RC model je možné ovládať aj cez internet, a to pomocou knižnice PeerJS jazyka JavaScript.

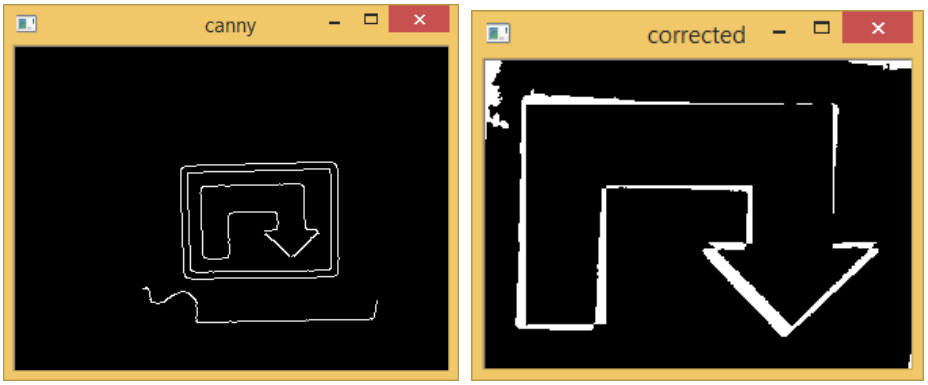
Konzolová verzia

Konzolová aplikácia sa skladá z niekoľkých okien, využívaných v závislosti od aktívneho režimu. Hlavným oknom je okno streamu, ktoré obsahuje okrem prijímaného obrazu aj HUD (angl. head-up display) s informáciami o pozícii kurzora, aktívnom režime a hodnotách zasielaných parametrov. Ďalšie okná slúžia na zobrazovanie prahovaného obrazu, hrán objektov a diferenčnej matice. S oknom prahovania (obr. 3 vpravo) pracujeme vo všetkých režimoch s výnimkou režimu manuálneho riadenia. Okno hrán (obr. 4 vľavo) a diferenčnej matice (obr. 4 vpravo) používame v režime rozpoznávania symbolov, na ktorom si aj ukážeme ich využitie.



Offboard verzia – rozpoznanie symbolu (vľavo), prahovanie obrazu (vpravo)

Obrázok 3 vľavo zachytáva okamih rozpoznania symbolu (otočenie o 180°). Na prahovanom obraze (obr. 3 vpravo) môžeme vidieť tvar rozpoznávaného symbolu. Hranice symbolu a orámovania zobrazujeme v okne *canny* (obr. 4 vľavo). Detekcia hrán sa vyžaduje na identifikáciu geometrických tvarov a stanovenie polohy vrcholov orámovania pri korekcii perspektívy. Diferenčná matica (obr. 4 vpravo) znázorňuje úroveň prekrytia vzorového a porovnávaného symbolu.



Offboard verzia – detekcia hrán (vľavo), diferenčná matica (vpravo)

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;
```

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

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 dokumentácii SB). Podobným spôsobom budeme riadiace parametre odvádzať aj v ďalších kapitolách.

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 od modelu, tým bude zrýchlenie vyššie. Naopak, čím bude lopta bližšie k modelu, tým bude zrýchlenie nižšie.

V prvom kroku (funkcia *trackBall*) konvertujeme snímku streamu z farebného modelu BGR do HSV[6], ktorý je vhodnejší na segmentáciu farieb. Nový obrázok v HSV farebnom priestore sa skladá z troch vrstiev, značiacich farebný odtieň (angl. hue), sýtosť (angl. saturation) a jas (angl. 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*[7] na získanie prahovaného obrázka. Prahovaný obrázok môže obsahovať diskrétne, redundantné body, ktoré môžeme vylúčiť aplikovaním morfologických operácií – eróziou (podmytím) a dilatáciou (rozšírením)[8]. Kombináciou operácií dosiahneme aj celistvejšie vyplnenie tvaru lopty v obrázku.

V ďalšom kroku vykonávame rozpoznanie lopty. Najskôr sa funkciou *findContours*[9] z obrázka extrahujú kontúry. Spomedzi kontúr nás zaujímajú tie, ktoré majú viac ako 10 vrcholov a ktorých plocha je väčšia, ako minimálna plocha obrysu pre rozpoznanie lopty. Postupne prechádzame pole kontúr a zisťujeme ich plochu. Ak je uvedená podmienka splnená, vykreslíme príslušnú kontúru do pôvodnej snímky. Ďalej z kontúry aproximujeme elipsu a tú vykreslíme tiež do pôvodnej snímky.

Demonštračné video režimu sledovania lopty je dostupné na stránke *http://youtu.be/KlCeNIPWv9A*. Klient[10] 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.

Režim sledovania čiary

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

[6] [6] http://cs.wikipedia.org/wiki/HSV

[7] [7] http://docs.opencv.org/modules/core/doc/operations_on_arrays.html#inrange

[8] [8] http://goo.gl/Znr9mk

[9] [9] http://goo.gl/ri5q0k

[10] [10] http://goo.gl/I2PAJK

zameriavame len na horizontálnu zmenu polohy čiary. Náš výrez 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 (*lineMin*) pre rozpoznanie čiary, model sa nebude pohybovať. V opačnom prípade pôjde model vopred definovanou rýchlosťou. Pozícia serva sa vypočíta z x súradnice stredu obrysu čiary.

Funkcia *lineFollow* najskôr vytvorí výrez z pôvodného obrázka. Potom nasleduje konverzia farebného modelu RGB do farebného modelu HSV, prahovanie, séria erózie, dilatácie a nájdenie obrysov čiary. Na výpočet polohy tvaru čiary využijeme momenty obrázka[11]. Zvislú súradnicu definujeme nastálo ako polovicu výšky snímky. Horizontálnu súradnicu vypočítame z podielu prvého priestorového momentu okolo osi X a nultého centrálneho momentu 0.

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

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

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*. 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 do odtieňov šedej (angl. grayscale). Obraz sa následne rozostrí funkciou *GaussianBlur*[12] a použije ako vstup pre hľadanie

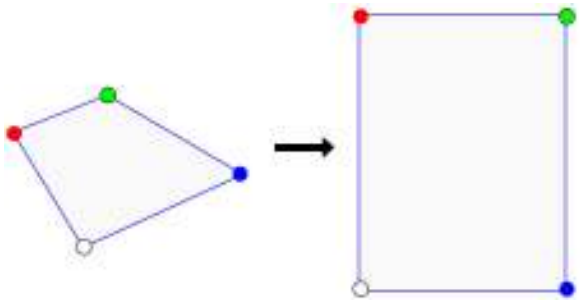
[11] [11] http://en.wikipedia.org/wiki/Image_moment

[12] [12] http://docs.opencv.org/modules/imgproc/doc/filtering.html#gaussianblur

hrán. Vzhľad snímky po detekcii hrán zachytáva obr. 4 vľavo. V upravenom obrázku sa ďalej nájdu kontúry. Každú kontúru aproximujeme funkciou *approxPolyDP*[13], čím znížime počet jej vrcholov.

Spomedzi kontúr nás budú zaujímať práve tie, ktoré majú štyri vrcholy a plochu väčšiu, ako je minimálna plocha obrysu pre rozpoznanie symbolu (*symbMin*). 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 určíme stred pôvodnej kontúry (neaproximovanej). Stred nám poslúži pri stanovení poradia vrcholov. 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.

Pred úpravou perspektívy vypočítame funkciou *getPerspectiveTransform*[14] transformačnú maticu. Na samotnú úpravu perspektívy potom slúži funkcia w*arpPerspective*[15]. Priradenie vrcholov ilustruje nasledovný obrázok:



Zobrazenie prislúchajúcich vrcholov – vľavo vstupné, vpravo výstupné pole vrcholov 0

Vzhľad obrázka pred a po korekcii perspektívy je pozorovateľný na obr. 4. Po úprave perspektívy nasleduje prahovanie, ktorým sa získa binárna snímka. Transformovaný obraz vychádza z pôvodnej snímky, preto ho najskôr potrebujeme skonvertovať do odtieňov sivej. Prahovanie vykonáme v dvoch etapách pomocou funkcie *threshold*[16]. Hodnoty pixelov výstupného obrázka sa určia podľa vzťahu:

$$\mathtt{dst}(x,y) = \begin{cases} \mathtt{maxval} & \text{if } \mathtt{src}(x,y) > \mathtt{thresh} \\ 0 & \text{otherwise} \end{cases}$$

0

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 najvyššej a najnižšej hodnoty bodov v obrázku. Najvyššiu a najnižšiu hodnotu bodov nájdeme funkciou *minMaxLoc*[17].

Princíp porovnávania symbolov sme spomenuli už vyššie. Výstupom funkcie bitového exkluzívneho súčtu (*bitwise_xor*[18]) je diferenčná matica. Ukážku diferenčnej matice môžeme vidieť na obr. 4 vpravo. Čím menej bielych bodov sa v matici nachádza, tým presnejšia zhoda nastala. Počet bielych bodov zisťujeme funkciou *countNonZero*[19]. Pred začatím porovnávania definujeme maximálny prípustný počet bielych bodov a index 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* (obr. 4 vpravo) 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.

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
```

[13] http://goo.gl/m4o7TG
[14] http://goo.gl/XfQW4R
[15] http://goo.gl/hmDBPZ
[16] http://goo.gl/JUF8N0
[17] http://goo.gl/dRKc99
[18] http://goo.gl/rA5LcB
[19] http://goo.gl/DwBhV8

```
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. Rýchly sled oboch operácií zabezpečí rovnomerné vyťaženie komunikácie a schopnosť rýchlo zareagovať na nové symboly.

Demonštračné video režimu 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.

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,
- server.html – JavaScript server pre knižnicu peer.js,
- client.html – JavaScript klient pre knižnicu peer.js.

Funkciu memorizácie trasy obsahuje modul *ws.html*. Vo všetkých moduloch pracujeme s JavaScript knižnicou jQuery. Moduly sme umiestnili na SD kartu. Na ich otvorenie cez prehliadač potrebujeme mať na RPi nainštalovaný webový server:

```
sudo apt-get -y install lighttpd
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 umiestnime všetky súbory našej HTML aplikácie do adresára */var/www*. Úvodnú stránku potom nájdeme na adrese *http://192.168.42.1*.

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*):

```
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()
```

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*[20] zabezpečíme zápis všetkých dát do deskriptora. Ak by sme túto metódu nezavolali, SB by na žiadne ďalšie odoslané príkazy nereagoval.

Klient sa po štarte pripojí cez adresu RPi a port 8888 k serveru a vytvorí webový soket. Hodnoty riadiacich príkazov sa odvádzajú z polohy myši vzhľadom k streamu. Postup výpočtu je podobný výpočtu u manuálneho riadenia v konzolovej verzii aplikácie.

Riadiace príkazy sú odosielané v intervale 50 ms, teda dvadsaťkrát za sekundu. Udalosťami *keydown* a *keyup* jazyka JavaScript zisťujeme stlačenie kláves. Držaním klávesa „A“ odblokujeme zasielanie hodnôt pre ESC iných ako 1500 μs, pri ktorej sa motor neotáča (neutrál). Funkcia memorizácie trasy sa aktivuje/deaktivuje klávesom „S“. Aktivovaním funkcie sa začnú riadiace príkazy ukladať do jednorozmerného poľa. Trasu je tiež možné načítať z textového poľa, alebo vygenerovať do textového poľa.

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[21], určenej na vytváranie P2P spojení prostredníctvom WebRTC[22] 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 prístupový bod pre RPi a je uchytený na modeli.

[20] [20] http://www.cplusplus.com/reference/cstdio/fflush

[21] [21] http://peerjs.com

[22] [22] http://cs.wikipedia.org/wiki/WebRTC

Na pripojenie k serveru je potrebné vytvoriť osobitne spojenie s rovnakým kľúčom skupiny, aký má server. Každé spojenie má buď automaticky generovaný, alebo ručne špecifikovaný identifikátor. Manuálnym určením identifikátora simulujeme model klient-server vo vzťahu 1:1 na sieti typu P2P.

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.

Záver

V práci som sa zaoberal prestavbou RC modelu auta na model riadený počítačom. Na zasielanie riadiacich príkazov bol použitý mini počítač Raspberry Pi. Streamovanie obrazu zaistila Raspberry Pi kamera. Zasielanie riadiacich príkazov pomocou páčkového ovládača nahradil program ServoBlaster, ktorý umožnil generovať PWM signál a ním prostredníctvom vstavaných GPIO pinov priamo riadiť servo a ESC modelu auta.

Na účely streamovania videa z pohľadu RC auta s minimálnou odozvou a tvorby aplikácie využívajúcej počítačové videnie sme nasadili metódu streamovania pomocou HTTP servera, ktorý podporoval video formáty kompatibilné s knižnicou pre manipuláciu s obrazom. Naša vytvorená aplikácia používa na rozoznávanie obrazu knižnicu OpenCV. Podporuje funkcie autonómneho riadenia – sledovanie čiary, sledovanie lopty a rozpoznávanie symbolov. Aplikácia HTML zahŕňa doplnkové funkcie a implementuje metódy zasielania riadiacich príkazov cez websokety.

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.

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, za zlomok ich ceny.

Literatúra

FAQs | Raspberry Pi [online]. [cit. 2015-03-05]. Dostupné na internete: <http://www.raspberrypi.org/faqs>.

OTTO, P. IP kamera na platforme Raspberry Pi. Banská Bystrica, 2014

OTTO, P. Bezdrôtové riadenie elektronických rýchlostných regulátorov a serv počítačom Raspberry Pi. Banská Bystrica, 2014

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

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>.

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>.

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>.

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