UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Krmiljenje mehanizma za usmerjanje spletne kamere s pametno mobilno napravo

(Control mechanism for directing webcams with mobile device)

Ime in priimek: Denis Subotić
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

Koper, december 2015

Ključna dokumentacijska informacija

Ime in PRIIMEK: Denis SUBOTIĆ

Naslov zaključne naloge: Krmiljenje mehanizma za usmerjanje spletne kamere s pametno mobilno napravo

Kraj: Koper

Leto: 2015

Število listov: 38 število slik: 17

Število prilog: 1 število strani prilog: 1 število referenc: 17

Mentor: doc. dr. Peter Rogelj

Ključne besede: Usmerjanje kamere, krmiljenje servo motorjev, magnetometer, Raspberry Pi, WebView

Izvleček:
V zaključni nalogi smo zgradili sistem za uporabniku prijazno krmiljenje mehanizma spletne kamere (pan-tilt). Običajno se tovrstne mehanizme upravlja z uporabniškimi vmesniki, temelječimi na standardnih vhodno-izhodnih napravah, kot sta miška in tipkovnica. Pomanjkljivost takšnega upravljanja je, da usmeritev kamere in smer pogleda uporabnika med seboj nista odvisni, kar otežuje naravno predstavo prostora. Ta problem želimo rešiti tako, da bomo kamero usmerjali s pametno mobilno napravo (mobilnim telefonom ali tablico), tako da bo usmeritev kamere enaka orientaciji mobilne naprave. Za upravljanje mehanizma bomo razvili aplikacijo za mobilne naprave z operacijskim sistemom Android. Podatke o usmeritvi naprave bomo pridobili iz vgrajenih senzorjev, kot sta magnetometer in žiroskop. Podatke o usmeritvi bomo preko spletne povezave pošiljali na nadzorni mini računalnik Raspberry PI, kjer bo nadzorni program krmilil mehanizem za usmerjanje spletne kamere. Mehanizem, ki ga bomo izdelali sami, bosta poganjala dva analogna servomotorja.

Key words documentation

Name and SURNAME: Denis SUBOTIĆ

Title of final project paper: Control mechanism for directing webcams with mobile device

Place: Koper

Year: 2015

Number of pages: 38 Number of figures: 17

Number of appendices: 1 Number of appendix pages: 1 Number of references: 17

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: Camera directing, servo motor control, magnetometer, Raspberry Pi, WebView

Abstract:
In the article we have built a system for user-friendly control mechanism for directing a web camera on a pan-tilt gimbal with an intuitive user friendly interface. Typically, that mechanisms are managed with user interfaces based on standard input-output devices such as a mouse and keyboard. The disadvantage of that is between direction of gaze and direction of cameras, where directions are not a same and conditionally to show the natural area is difficult. This problem we want to solve with mobile devices (tablet or smartphone), where web-cameras are managed with mobile device, the orientation of camera is equal to the orientation of the mobile device. For management mechanism, we have developed an application for mobile devices with the Android operating system. The data of orientation from mobile device we give from the embedded sensors such as magnetometer and gyroscope. That data we send via online connection to minicomputer Raspberry Pi, where the control program controls mechanism for directing webcams. The mechanism that we developed ourselves, is driven by two analog servo motors.

Zahvala

Zahvaljujem se mentorju, dr. Petru Roglju, za strokovno pomoč pri izdelavi zaključne naloge.

Zahvaljujem se Deanu Kendiču, ki mi je pomagal pri izdelavi mehanizma za krmiljenje spletne kamere (pan-tilt).

Zahvaljujem se tudi družini, ki mi je omogočila študij, in ženi ter prijateljem za podporo in pomoč med študijem.

Kazalo vsebine

Kazalo slik

Kazalo prilog

Seznam kratic

RPI	Raspberry Pi
GPIO	General-purpose input/output
PWM	Pulse-width modulation
JS	JavaScript
*i*2*c*	Inter Integrated Circuit
VR	Virtual Reality
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IP	Internet Protocol
CPU	Central Processing unit
SDA	Serial Data
SCL	Serial Clock

1 Uvod

Zaključna projektna naloga predstavlja sistem za krmiljenje spletne kamere s pomočjo pametne mobilne naprave. Običajno se tovrstne mehanizme upravlja z uporabniškimi vmesniki, temelječimi na standardnih vhodno-izhodnih napravah, kot sta miška in tipkovnica. Pomanjkljivost takšnega upravljanja je, da usmeritev kamere in smer pogleda uporabnika med seboj nista odvisni, kar otežuje naravno predstavo prostora. Cilj zaključne projektne naloge je razviti tak sistem, da je med mehanizmom za upravljanje spletnih kamer in spletno kamero intuitivna interakcija. V drugem poglavju je predstavljen sistem po enotah. V tretjem poglavju je predstavljen načrt sistema, kjer je za vsako enoto/komponento predstavljena izbira tehnologije, način njenega delovanja in povedano, čemu bi natančno služila tovrstna tehnologija v našem projektu. V četrtem poglavju je prikazana izvedba, kjer predstavimo razvojno okolje, opravljeno testiranje in pridobljeni rezultati take rešitve. Na koncu zaključne naloge je predstavljen končni izdelek, njegove slabe in dobre lastnosti ter mogoče izboljšave.

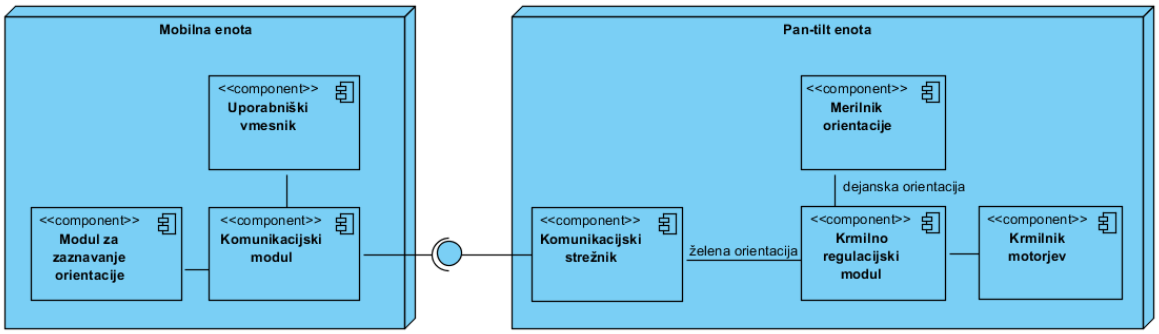
1.1 Motivacija

Danes krmiljenje spletnih kamer večinoma deluje preko standardnih vhodnih naprav, kot so tipkovnica, igralna palica ali pa miška. Dojemanje naravnega prostora, kjer usmeritev kamere in pogled uporabnika med seboj nista odvisna, pa je zelo težko. Enega boljših primerov predstavljajo igrice, v katerih igramo v prvi osebi. Igra se izvaja z uporabo tipkovnic in mišk. Predstavljajmo si uporabo naše predlagane rešitve tako, da bi imeli pred sabo (pred očmi) neki ekran, ki ga lahko usmerjamo v različne smeri (v našem primeru tablični računalnik ali Googlov Cardboard). Igra bi bila zelo napeta, odziv igralcev bi bil hitrejši in motivacije bi bile zelo večje. Drugi primer pa je lahko nadzorovanje nekega prostora z oddaljene lokacije: tako bi bili manj omejeni glede izbire smeri pogleda in ogled bi lahko bil hitrejši. Torej, glavna ideja zaključne naloge je v tem, da uporabniku ponudimo prijazen sistem tako, da mu je dovolj naravno premikati pametno mobilno napravo in naknadno si ogledati okolje, v katerem je kamera postavljena, kar pomeni, da je mogoča komunikacija na daljavo med uporabnikom (pametno mobilno napravo) in kamero (krmiljenje na daljavo). Naknadno se lahko tak sistem nadgradi v navidezno resničnost (v angl. Virtual Reality). Navidezna

resničnost je neka vrsta računalniške simulacije in uporabniku daje občutek, da je v nekem umetnem okolju. V našem primeru je to umetno okolje ponazoritev realnega okolja. Zaključna projektna naloga rešuje problem v delu, kjer pravilno usmerjamo kamero glede na aktivnost uporabnika, ne pa tudi prenosa slik iz kamer v predstavitev v okolju navidezne resničnosti.

2 Predstavitev sistema

Najprej si poglejmo, kako lahko naš problem razgradimo na več enot, kjer vsaki enoti oziroma komponenti določimo njeno funkcijo in povezave do drugih enot. Našo razgradnjo sistema prikazuje Slika 1.



Slika 1: Komponentni diagram - Razdelitev sistema po enotah in po komponentah

Sistem sestoji iz dveh enot, vsaka od njiju pa vključuje več komponent sistema. Prva enota je mobilna enota in nudi uporabniku uporabniški vmesnik za boljšo in bolj intuitivno interakcijo z mobilno enoto (Uporabniški vmesnik). Ko mobilno napravo (v našem primeru pametni telefon) premikamo, mobilna enota že meri usmerjenosti (Modul za zaznavanje orientacije).

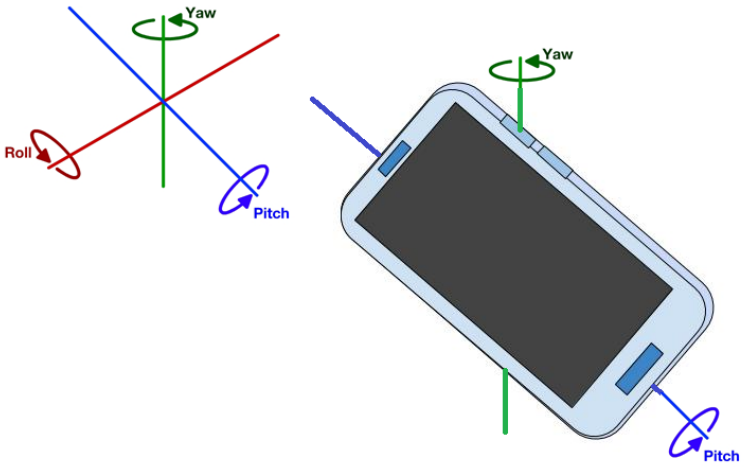
Druga enota je pan-tilt enota. Sestavljena je iz treh modulov, med katerimi so merilnik orientacije, krmilno-regulacijski modul in krmilnik motorjev. Merilnik orientacije meri usmerjenost spletne kamere. Krmilno-regulacijski modul je modul, ki regulira usmerjenost spletne kamere. Krmilnik motorjev je elektronsko vezje, ki z generiranjem ustreznega krmilnega signala za motorje, omogoča njihovo upravljanje.

Obe enoti povezuje komunikacijski vmesnik, preko katerega mobilna enota pan-tilt enoti posreduje podatke o usmerjenosti mobilne naprave. Oglejmo si na grobo delovanje oz. namen vsake enote in komponente.

2.1 Mobilna enota

Mobilna enota združuje vse komponente, ki so vezane na pametno mobilno napravo (tablični računalnik ali pa pametni telefon) (glej Sliko 2). Uporabniški vmesnik mobilne

enote uporabniku omogoča vnos naslova (kjer bo posredoval podatke) in se odloči, kdaj začeti meritve. Druga komponenta je pridobivanje orientacije. Danes je vsaka pametna mobilna naprava opremljena s številnimi in raznovrstnimi vgrajenimi senzorji, ki imajo različne funkcije. En takih senzorjev (električnih vezij) je magnetometer. Ta nam da informacijo o trenutni usmerjenosti mobilne naprave s tem, da je usmerjenost določena s tremi koti zasuka okrog treh koordinatnih osi trirazsežnega prostora [2]. Z drugimi besedami bi lahko to interpretirali kot orientacijo glede na vsako os. Ogledamo si lahko Sliko 2.



Slika 2: Komponente rotacij

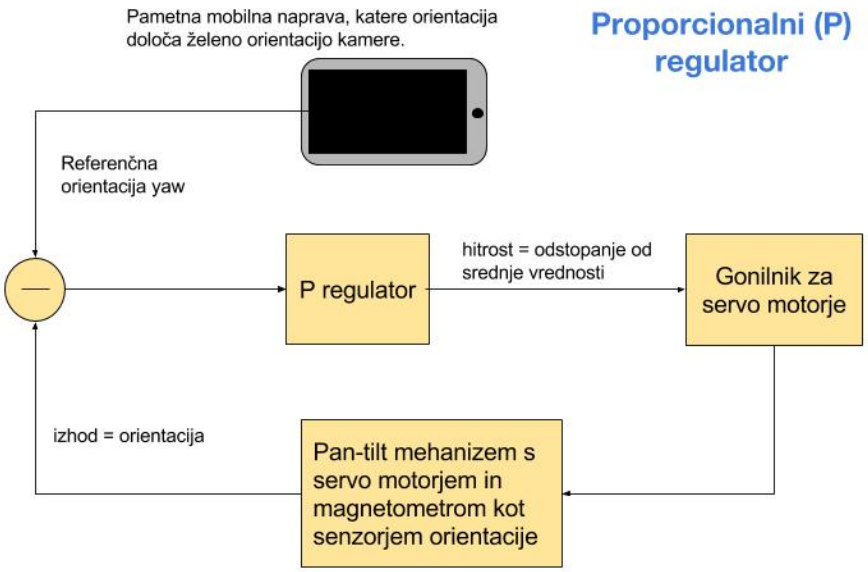
Tretja in zadnja komponenta mobilne enote, komunikacijski modul, skrbi za posredovanje podatkov, pridobljenih od vgrajenih senzorjev pametnih naprav pan-tilt enot.

2.2 Pan-tilt enota

Pan-tilt enota vključuje vse komponente, ki regulirajo oz. krmilijo našo spletno kamero. Glavna naloga pan-tilt enote je usmerjanje pan-tilt mehanizma, kar zahteva prejemanje podatkov o želeni usmerjenosti od mobilne enote ter merjenje dejanske orientacije, na osnovi tega pa ustrezno krmiljenje oziroma regulacijo servomotorjev.

3 Načrt sistema

Zastavili smo si cilj, da na daljavo upravljamo neki sistem v nekem realnem času. Za upravljanje usmerjenosti lahko uporabimo servo motorje. Najprej začnimo upravljati Pitch komponento usmerjenosti, torej komponento, ki govori o višini pogleda kamere. Ustrezno Pitch komponento usmerjenosti kamere lahko dosežemo tako, da glede na želeno usmerjenost, izračunamo potreben krmilni signal za motorje, ki le te pravilno usmeri. Pri servomotorju, ki pa je namenjen Yaw rotaciji, pa taka rešitev ni mogoča. Motorji so namreč omejeni z obsegom mogoče rotacije, ki je običajno 180 °, medtem ko se uporabnik lahko poljubno suče v prostoru. Zato smo se odločili, da za to komponento uporabimo neprekinjeno rotacijo na motorju, kjer ne moremo neposredno krmiliti usmerjenosti, pač pa motorju lahko določamo hitrost in smer vrtenja. Ker torej motor sam ne omogoča enolične preslikave med vhodnim signalom in dejanskim kotom zasuka, potrebujemo za usmerjanje dodatno informacijo o zasuku motorja – usmerjenosti enote, ki jo pridobimo z uporabo magnetometra. Na osnovi teh informacij lahko motor upravljamo z regulacijskim postopkom. Glej Sliko 3.



Slika 3: Proporcionalni (P) regulator

3.1 Mobilna enota

Mobilni operacijski sistemi so programske platforme, s katerimi lahko dostopamo do vseh komponent neke pametne naprave ter uporabljamo dodatne programe, ki jih lahko neka pametna naprava izvaja (to so kalkulator, igrice itd.). Mobilni operacijski sistemi danes tekmujejo z računalniškimi operacijskimi sistemi, ker pametne mobilne naprave pridobivajo vedno boljše performanse. Danes najbolj uporabljeni mobilni operacijski sistemi so: Android (82,8 %), iOS (13,9 %) in Windows Phone (2,6 %) [8].

3.1.1 Android

Mobilni operacijski sistem Android je razvilo podjetje Google na osnovi odprtokodnega operacijskega sistema Linux (na ravni distribucije GNU/LINUX za vgrajene sisteme) [3]. Več kot milijarda ljudi ga danes uporablja na pametnih telefonih in tabličnih računalnikih, pa tudi na televizijskih sprejemnikih, pametnih urah, v pametnih očalih in v avtomobilih. Pravijo, da nam Android posladka vsakdanje življenje, zato ima vsaka nova verzija ime po nečem sladkem. Zadnja verzija je Android 6.0 Marshmallow (sladka penica). Grafični vmesnik temelji na elementih živahnih barv in odzivni funkcionalnosti z imenom Material Design.

3.1.2 Windows Phone 8.1

Mobilni operacijski sistem Windows Phone je razvilo podjetje Microsoft. Prvič je bil predstavljen na Mobile World kongresu leta 2010. Windows Phone je lahko nameščen samo na Microsoftovih mobilnih napravah [17]. V decembru bo izšla nova verzija z imenom Windows Phone 10, ki bo omogočala uporabo istih aplikacij na pametnih telefonih kot tudi na računalnikih. To pomeni, da bomo imeli možnost razviti samo eno aplikacijo, uporabno za obe platformi. Grafični vmesnik je tu zelo drugačen kot Androidov, in sicer želi, da so na ekranih samo prikazana okenca, ki ponazarjajo aplikacije. Odzivnost je zelo visoka, saj ni veliko zahtevnih elementov. Microsoft je svoj grafični vmesnik poimenoval Modern UI oz. grafični jezik Microsoft. Modern UI izhaja iz tako imenovanega grafičnega vmesnika Metro UI.

3.1.3 Android ali Windows Phone?

Danes se je težko odločiti, za kateri mobilni operacijski sitem razviti aplikacijo, saj bi radi imeli takšno aplikacijo, ki bi koristila vsem uporabnikom. Da bi imeli možnost uporabljati isto aplikacijo na različnih platformah, lahko uporabimo spletno rešitev (v angl. WebView solution) [7], ki je le ena izmed mogočih rešitev hkratnega razvoja za

več platform. Takšna aplikacija, je zgrajena z uporabo spletnih tehnologij, ki delujejo v spletnih brskalnikih, vendar je lahko prilagojena na specifično mobilno platformo tako, da v njej deluje kot samostojna aplikacija. Aplikacija torej opravlja iste funkcije kot spletni brskalnik, ampak mogoče je odpraviti naslovno vrstico in vse nepotrebna orodja, ki jih nudi sam brskalnik. V naslednjih poglavjih je opisana rešitev, ki je lahko implementirana na različnih mobilnih operacijskih sistemih, testirana pa je bila samo na dveh, tj. na sistemih Android 4.3 in Windows mobile 8.1.

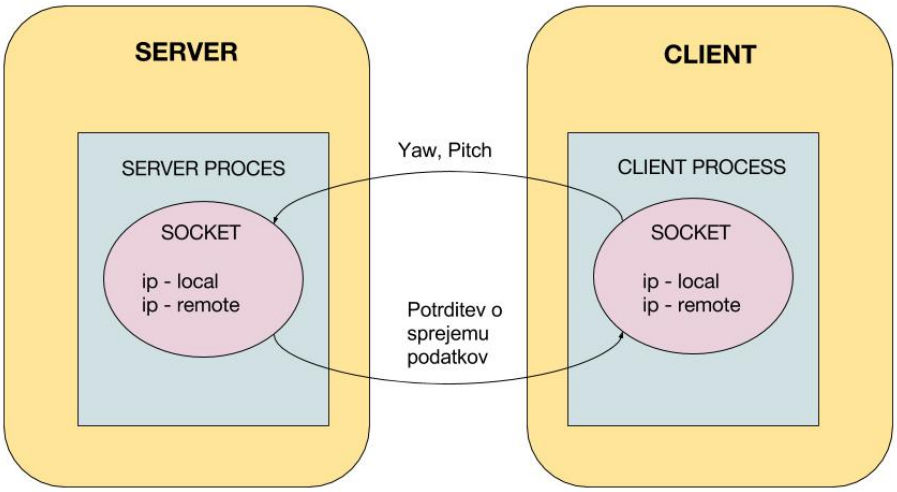
3.1.4 Pridobivanje orientacije mobilne naprave

Kot smo že omenili, bomo mobilno aplikacijo razvili kot WebView (spletna stran), ki jo zaženemo preko spleta. Programski jezik, točneje skriptni jezik, ki smo ga uporabili za našo rešitev, je JavaScript. Naloga mobilne aplikacije je branje vrednosti integriranih senzorjev mobilne naprave, kot je magnetometer, določitev orientacije ter pošiljanje pripadajočih kotov pan-tilt enoti.

3.1.5 Posredovanje podatkov pan-tilt enoti

Vtičnica (ang. socket) v informatiki, v modernih operacijskih sistemih, ponazarja abstraktno programsko opremo, ki omogoča lažjo (z uporabo knjižnic API) komunikacijo oz. izmenjavo podatkov preko omrežja (žično ali brezžično) [4]. Do kanala dostopa preko vrat, pri čemer komunikacija med procesi teče na dveh fizičnih, med sabo ločenih napravah. Z vidika programiranja je vtičnica objekt, s katere je mogoče brati podatke za njihovo prejemanje od oddaljene naprave ter vanjo vpisovati podatke, da jih pošljemo oddaljeni napravi [4]. Pri socketu je implementiran odjemalec-strežnik model (glej Sliko 4). Za realizacijo naše rešitve sem uporabil odprtokodno knjižnico socket.io. Mobilna enota se obnaša kot odjemalec (client) v našem projektu, in kar je tudi razvidno iz naslednje slike, pošiljamo dva podatka: Yaw in Pitch. Protokol, ki smo ga uporabili, je TCP protokol. TCP protokol je varnejši od UDP protokola, ker TCP nadzoruje napake in to, kdo se poveže. Skrbi, da vsi paketi pridejo do cilja brez nobene izgube. Če pride do napake ali izgube podatkov zahteva ponoven prenos. UDP protokol ne nadzoruje pravilnosti prenosa podatkov niti ne ustvarja povezave med odjemalcem in strežnikom [12]. Zdaj bi lahko rekli, zakaj ne uporabljati UDP-ja v primeru, ko je bistvo samo posredovanje vrednosti, kar tudi pohitri komunikacijo med pametno napravo in samim strežnikom. Lahko bi upoštevali tudi to možnost, ampak če vsakdo lahko pošilja vrednosti na IP naslov strežnika, potem lahko neka druga naprava pošilja svoje kote in spletna kamera ne bi več sledila suku ključnega uporabnika. Temu bi se raje izognili in skušali omejiti samo na enega uporabnika, s tem pa torej nadzorovati, kdo se poveže. S tem izgubimo na hitrosti komunikacije, ampak se izognemo anomalijam,

kot so nepredvidljive rotacije spletne kamere glede na suk uporabnika.



Slika 4: Socket

Glede na to, da se mobilna enota obnaša kot odjemalec, lahko vidimo spodaj, kako lahko implementiramo rešitev:

```
<script>
  function startWatch() {
        //povezemo se na server ki ima naslov
        // ip = 192.168.1.12 na vrata 8080
    var socket = io.connect("http://192.168.1.12:8080");
    //on metoda s parametrom connect vrne rezultat ce je
    //povezava bila uspesna
    socket.on("connect", function () {
      document.getElementById("zapis").innerHTML="Connected!";
    });

        //klicemo metodo, ki bo prejemala iz naprave
        //orientacijo s tremi osmi
    getDataFromSensors(socket);

  }
  function getDataFromSensors(socket) {
        if (window.DeviceOrientationEvent) {
              //event se sprozi ob vsaki meritvi
        window.addEventListener('deviceorientation',function(eventData){
              // gamma
        var tiltLR = eventData.gamma;
              // beta
        var tiltFB = eventData.beta;
```

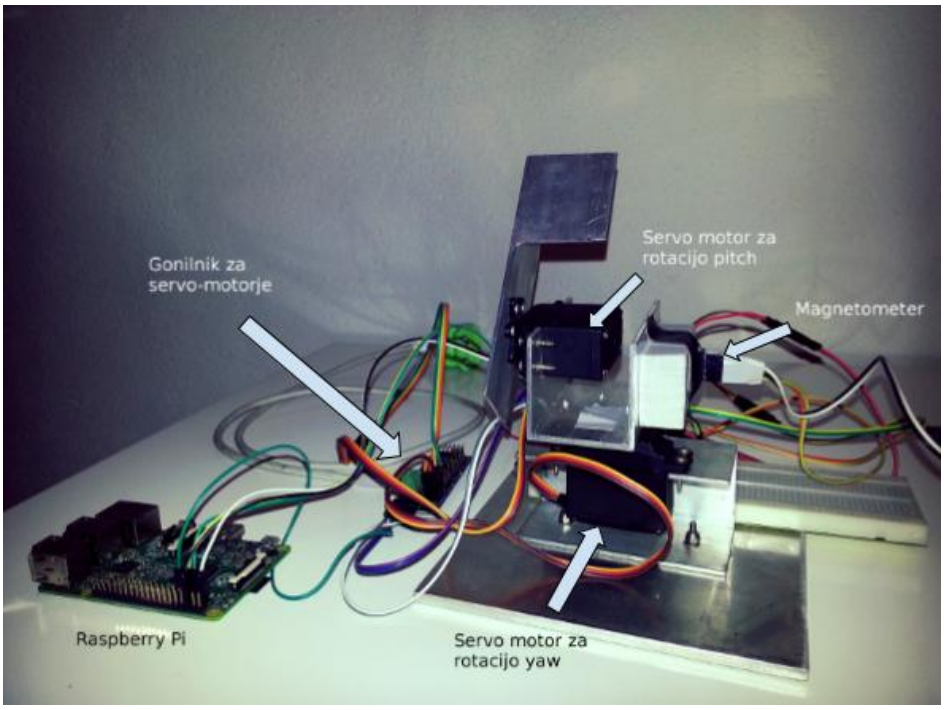
```
                // alpha
            var dir = eventData.alpha
            //socketu pisemo celo stevilo orientacije dir
            socket.send({pan: parseInt(dir), tilt: tiltLR})
            }, false);
        } else {
        }
    }
</script>
```

Komentar: Na začetku vidimo IP naslov, ki je bil uporabljen v lokalnem omrežju. Trenutno je torej fiksirana na 192.168.1.12. Da bi lahko naslov menjali vsakič, ko si to želimo, bi bilo najbolje dodati neki input field, kjer uporabnik ročno vnese naslov, na katerem se poveže pametna naprava.

3.1.6 Uporabniški vmesnik

Uporabniku smo dali možnost, da si lahko sam nastavlja IP naslov za povezavo na pan-tilt enoto in ob povezavi mu izpisuje vse tri kote usmerjenosti mobilne naprave.

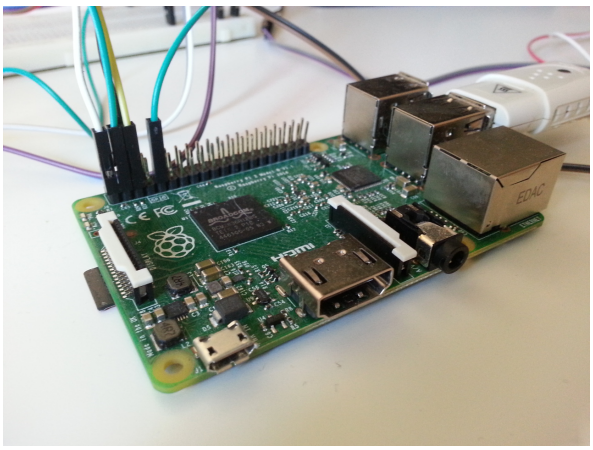
3.2 Pan-tilt enota



Slika 5: Pan-tilt enota

Pan-tilt enota sestoji iz računalniškega dela, ki temelji na mini računalniku Ra-

spberry PI, ter strojnega dela, ki združuje mehanske komponente, kot so servomotorji in (aluminijasti) nosilci. Raspberry Pi (RPi Slika 6) je zelo majhen (mini) računalnik, velikosti kreditne kartice, ki se lahko poveže na monitor ali pa televizijo. RPi se lahko uporablja z navadnimi standardnimi napravami, kot sta tipkovnica in miška. Gre za zelo kompaktno napravo, s katero lahko nadomestimo navaden računalnik; ima vse lastnosti običajnih računalnikov, le da ima nekoliko manjšo zmogljivost. Z njim torej lahko opravljamo vse naloge, ki jih sicer lahko opravimo na običajnih računalnikih [14].



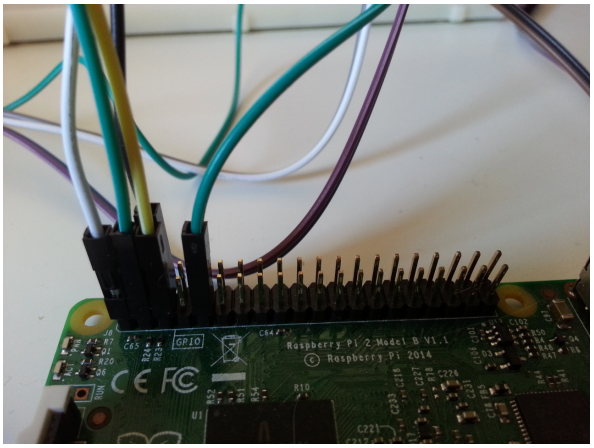
Slika 6: Raspberry Pi

3.2.1 GPIO vmesnik

Na Raspberry Pi lahko preko GPIO (General Purpose Input/Output) vmesnika razmeroma preprosto povežemo najrazličnejše senzorje, na primer za merjenje temperature, vlažnosti, pritiska in za detekcijo premika. GPIO vmesnik sestoji iz 40 sponk (pinov)(glej Sliko 8), med katerimi je 28 vhodno-izhodnih, nekateri pa imajo še posebne dodatne lastnosti. Tako na primer dve sponki lahko uporabimo za I2C vodilo.

3.2.2 I2C vodilo

Med sponkami, ki so namenjene izmenjavi podatkov, imamo še eno vgrajeno I2c vodilo (Inter Integrated Circuit). I2C vodilo je razvil Philips leta 1982. Namen tega vodila je bil na začetku preprosta povezava procesorja (CPU) s perifernimi enotami v TV-sprejemniku. Po veliki proizvodnji takih vodil so leta 1992 uvedli prvo verzijo protokola. I2C vodilo (Glej Sliko 8) je sestavljeno iz dveh signalnih linij. Omogoča nam, da povežemo več naprav skupaj samo na dveh signalnih linijah. S tem zmanjšamo število vodnikov, ampak imamo še eno posebnost, in sicer: en pin je namenjen prenosu podatkov in ukazov (SDA), drugi pa sinhronizaciji in določanju hitrosti komunikacije



Slika 7: GPIO vmesnik [13]

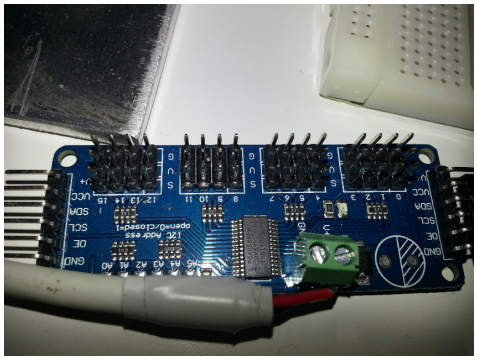
(SCL) (torej urni signal, v angl. clock). Če bi hoteli povezati kakšno komponento na I2c vodilo, potem ta komponenta pridobi unikaten naslov [11].

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Slika 8: GPIO vmesnik mini računalnika Raspberry PI z I2C vmesnikom na sponkah 3 in 5 (označena z modro barvo)

3.2.3 Krmilnik za pulzno širinsko modulacijo

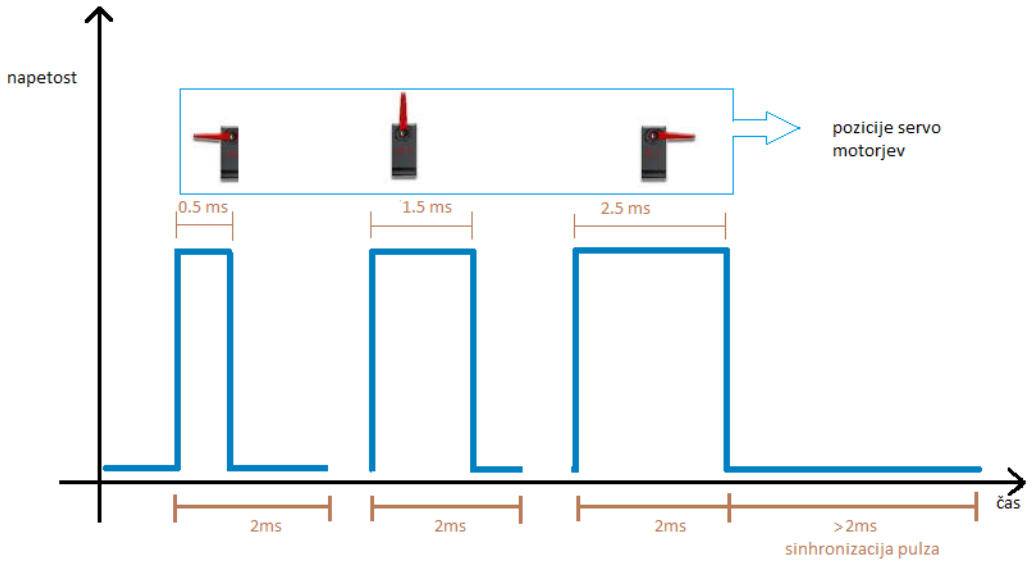
Krmilnik za pulzno širinsko modulacijo (PWM, v angleščini Pulse Width Modulation) je vezje, ki nam omogoča povezati hkrati 16 naprav [1]. Krmilnik je sicer prvenstveno namenjen krmiljenju LED svetil, uporabimo pa ga lahko tudi v druge namene, med katerimi je tudi krmiljenje servomotorjev. Naprava je povezana na I2c vodilo in generira PWM signal vsakemu servomotorju posebej. Od računalninka preko I2C vodila prejme 12-bitne podatke, ki določajo časovni zamik začetka in konca pulza v periodi PWM signala. Ta model smo izbrali, ker imamo na razpolago knjižnico, potrebno za upravljanje pwm signala za določen servomotor.



Slika 9: Krmilnik PWM

3.2.4 Krmiljenje motorjev s pulzno širinsko modulacijo

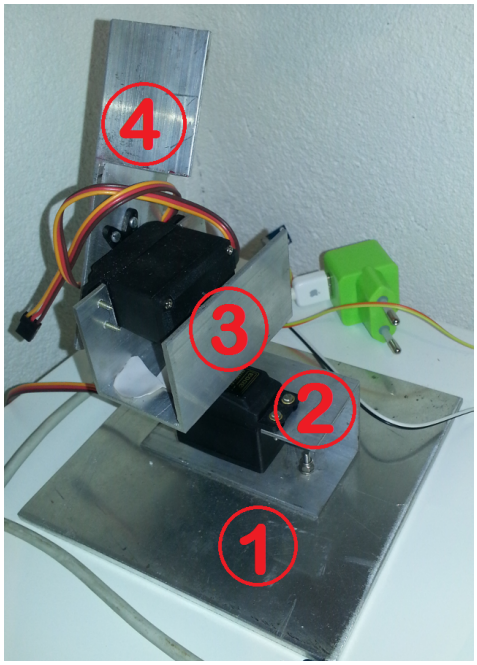
Pulzni širinski signal je krmilni signal, ki nam služi pri upravljanju servo ukazov in pridobitvi takega odziva motorja, kot si ga želimo (glej Sliko 10). Osnovna frekvenca PWM signala za krmiljenje servomotorjev mora biti 50 Hz z impulzno širino med 1 ms in 2 ms, kar ustreza zasukom med 0 in 180 °. Vedeti pa moramo, da so te vrednosti standardne, za katere ima vsak motor lahko svojo toleranco [15]. Predlagano je, da testirate svoje servomotorje in upoštevati morebitna odstopanja.



Slika 10: Upravljanje servo-motorja s pulznim širinskim signalom

3.2.5 Strojni del - profili

1. Plošča aluminija, ki ima nalogo za stabiliziranje gibanja mehanizma, neke vrste podpore. Plošča je večjega profila kakor druge komponente. Dimenzije: 14 x 14 cm.



Slika 11: Pan-tilt strojni del

2. Stojalo iz aluminija L-profil, ki drži servomotor vodoravno. Podpora je nastavljena na sredini glavne osnovne plošče. Pritrjena je z dvema vijakoma in na vrhu je odprtina za servomotor. Naloga tega pritrjenega servomotorja je obračanje spletne kamere v levo in desno smer vertikalne osi. Dimenzije: 8,5 x 4 x 4 cm.

3. Isto stojalo iz aluminija, kot je opisano v 2. točki. Razlika je samo v tem, da je to stojalo pritrjeno na majhnem kolesu servomotorja in drži pritrjen drugi servomotor, ki bo vrtel spletno kamero.

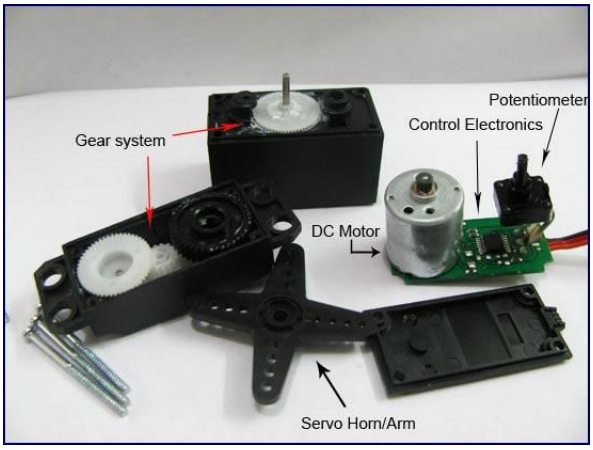
4. Zadnji kos aluminija je namenjen pritrditvi spletne kamere. Višina: 12 cm in širina: 4 cm.

3.2.6 Servo-motor in njegovo delovanje

Servomotor je električna naprava, ki na vhodu prejme diskreten signal in na izhodu aktivira analogno vrtenje. Smer zasuka motorja je sorazmerna trajanju impulzov vhodnega signala. Na splošno povedano, s pulzno širinskim signalom krmilimo servomotor tako, da določamo kot zasuka motorja. Po navadi so taki motorji omejeni s kotom zasuka med 0 in 180 stopinj. Če si želimo imeti servo motor, ki suče v neskončnosti, se lahko običajni servo motor predela (ali kupimo že takega, ki ima nastavek). Za motor, ki se lahko vrti nad 360 °, pravimo, da ima neprekinjeno vrtenje (v angl. continuous rotation). Namesto kota zasuka pri takšnem motorju dolžina vhodnega impulza določa hitrost (in smer) vrtenja motorja. Ustrezno nastavljen motor pri srednje dol-

gih vhodnih impulzih miruje, pri odstopanju od pa se vrti v eno ali drugo smer. Več odstopanja imamo, hitrejše vrtenje bomo imeli.

Znotraj servomotorja imamo različne mehanske in elektronske dele. Glej Sliko 12.



Slika 12: Servo motor in njegove komponente

1. DC motor
2. Potenciometer
3. Krmilna elektronika - v angleščini feedback system
4. Motorno krmilo
5. Krmilni mehanizem

Potenciometer je priključen na izhodno gred tako, da pridobimo vrednost upornosti proporcionirano kotu trenutnega položaja. Krmilna elektronika skrbi za smer vrtenja. Krmilni mehanizem (kolesa) služi natančnejšemu pozicioniranju [5].

Videli smo, da servomotor lahko prejme dve vlogi, določanje zasuka z omejitvijo na največji mogoči zasuk 180 stopinj ter določanje hitrosti neprekinjenega vrtenja, kjer zasuk motorja ni omejen. Neprekinjeno vrtenje pomeni, da motor vrti levo ali pa desno nad 360 ° (če to uporabnik krmili z mobilno napravo, pomeni, da se vrti okoli sebe). To pomeni, da je servomotor presegel tudi maksimalen kot 180 °. V našem projektu rabimo oba servomotorja, kar pomeni, da je enega treba modificirati. Modificirati je treba motor, ki bo vrtel na vertikalni osi, ker se uporabnik lahko v prostoru suče v nedogled. Kako predelati servomotor? Predelava motorjev (glej Sliko 12) je preprosta. Vse, kar je treba narediti, je, da iz motorja odstranimo aluminijast nastavek, ki omejuje zasuk motorja, in njegovo gred, ki povezuje z vgrajenim potenciometrom.

3.2.7 Magnetometer in njegovo delovanje

Vezje HMC5883L (glej Sliko 13), je 3-osni digitalni magnetometer. Vezje ima širok spekter uporabe, najpogostejša pa je kot digitalni kompas za zaznavanje smeri ali pa za detekcijo feromagnetnih kovin. Vgrajeni so trije magnetni senzorji (magnetoupor), razporejeni na treh oseh (kartezijski koordinatni sistem z x, y, z osmi). Komponenta HMC5883L je zgrajena tako, da se jo uporablja preko I2c vodila [9].

Slika 13: Magnetometer vezje

Magnetno polje in električni tok sta zelo tesno povezana oz. odvisna. Medtem ko se električni tok pomika skozi žico, se tvori magnetno polje. Smer magnetnega polja (npr. zemlje) vpliva na pretok elektronov v senzorju in s pomočjo spremembe toka se lahko izračuna smer magnetnega polja. Nastavitve magnetometra in prevzemanje vrednosti se lahko opravlja preko odprtokodne knjižnice hmc5883l, ki je napisana v skriptnem jeziku Python. V naši nalogi smo si sami uredili nastavitve magnetometra ter izračunali koordinato preko Raspberryjevih knjižnic uporabe I2c vodila SMBUS. Navodila (rešitev) so prikazana v nadaljevanju.

Raspberry Pi igra glavno vlogo v našem projektu. Sprejema vrednosti s strani mobilnih naprav, nato nadzoruje dejansko vrednost magnetometra na pan-tiltu in na koncu upravlja servomotorje tako, da se obrne spletna kamera na želeno orientacijo.

3.2.8 Merjenje orientacije pan-tilt naprave

Preden začnemo meriti, je treba določiti naslov magnetometra na I2c vodilu. Če odpremo konzolo, lahko to preverimo z ukazom: **i2cdetect -y 1**. V našem primeru je to naslov **0x1E**. Pripravimo nastavitve:

```
bus = smbus.SMBus(1)
address = 0x1E
```

Za magnetometer smo pripravili nekaj funkcij za branje in pisanje zlogov na i2c vodilo. Uporabili smo knjižnico SMBUS.

```
def read_byte(adr):
    return bus.read_byte_data(address, adr)

def read_word(adr):
    high = bus.read_byte_data(address, adr)
    low = bus.read_byte_data(address, adr+1)
    val = (high << 8) + low
    return val

def read_word_2c(adr):
    val = read_word(adr)
    if (val >= 0x8000):
        return -((65535 - val) + 1)
    else:
        return val

def write_byte(adr, value):
    bus.write_byte_data(address, adr, value)
```

Komentar: funkcija math.atan2(num1,num2) - parametri/koordinate so lahko v drugih primerih različni, odvisno od tega, kakšno lego ima magnetometer.

3.2.9 Krmiljenje pitch komponente usmerjenosti

Krmiliti pitch komponente usmerjenosti je zelo preprosto. Naslavlja se PWM krmilnik, ki vsebuje registre za krmiljenje vsakega od motorjev. Torej moramo za krmiljenje motorja le vpisovati podatke na pravi register PWM krmilnika.

```
pwm = PWM(0x40)
```

Nato je treba nastaviti frekevenco impulznih signalov. Običajno 50Hz.

```
pwm.setPWMFreq(50)
```

Če imamo 50 Hz, pomeni, da je ena perioda:

$$T = 1/f$$

iz tega sledi, da imamo pulz širine 20 ms. Omenili smo, da je servomotor namenjen vrtenju med 0 in 180 °, kar je tudi dovolj za krmiljenje pitch komponente usmerjenosti.

Po nasvetih [14] na videu vidimo, da se polovica, 90 °, nahaja na 1,5 ms. Zato da bi krmilili servo motor, je treba določiti čas pulza kot relativni delež časa celotne periode. Vrednosti pošiljamo preko I2c vodila z velikostjo 12 bit. Da pridobimo sredino, je dovolj izračunati:

$$\frac{1,5ms}{20ms} * 2^{12} = \frac{1,5}{20} * 4096 = 307$$

(zaokrožimo na 307, ker lahko krmilimo samo s celimi števili). Če preverimo, ali je to res polovica za naš servomotor, lahko opazimo, da ni, ker je naš motor verjetno zamaknjen za nekoliko stopinj. V našem primeru smo dobili polovico na vrednosti 320. Glede na to, kako smo postavili servomotor na strojnem delu, je zdaj treba to samo prilagoditi. To pomeni, da če imamo 0 °, mora spletna kamera gledati navzdol, če pa imamo 90 °, potem mora spletna kamera gledati naravnost. Želene vrednosti pitch in yaw komponente pridobimo iz podatkovne baze, kamor jih vpisuje komunikacijski strežnik.

```
db = MySQLdb.connect(host="localhost",user="root",passwd="fmnt",db="or")
cur = db.cursor()
cur.execute("SELECT * FROM degree WHERE id = 1")
ver = cur.fetchone()
db.close()
```

Za krmiljenje lahko uporabimo naslednji ukaz:

```
pwm.setPWM(3,0,int(ver[2]*0.8)+230)
```

Komentar: prvi parameter (vrednost 3) pomeni, da smo motor povezali na vhod krmilnika 3. ver[2] je druga vrednost, ki jo pridobimo iz podatkovne baze in predstavlja želeno vrednost kota Pitch. Konstanta 0.8 nam določa razmerje med doseženega kota rotacije motorja v stopinjah in relativno dolžino pulza določenega z 12 bitno ločljivostjo.

3.2.10 Regulacija yaw komponente usmerjenosti

V prejšnjih poglavjih smo videli, da je za to komponento treba servo motor predelati tako, da pridobimo neprekinjeno vrtenje. Za doseganje želenega kota takšnega motorja ne moremo krmiliti, pač pa njegovo usmerjenost lahko upravljamo z uporabo regulacijske zanke. Zato temu ne rečemo več krmiljenje motorja, ampak regulacija, ki jo lahko rešimo s proporcionalnim regulatorjem. Želeno orientacijo že imamo na razpolago, ker smo v prejšnji točki prebrali celo vrstico v podatkovni bazi. Zdaj moramo pridobiti dejansko orientacijo magnetometra oz. spletne kamere. Zato sledi klic funkcije magneto():

```
magneto = magneto()
```

In takoj zatem je treba izračunati referenčno vrednost. Želeni vrednosti – orientaciji – odštejemo kot magnetometra. Rezultat pa je treba obdržati v intervalu 0–360 °, zato referenco delimo po modulu 360:

```
ref = (int(mag)-ver[1])%360
```

Če je referenčna vrednost večja od 180 °, moramo obdržati pravilno smer vrtenja motorja in zato odštejemo referenci vrednost 360:

```
if(ref>180):
        ref=ref-360
```

Ostalo nam je še upravljanje servomotorja. Preveriti je treba, ob kateri dolžini krmilnega pulza PWM signala motor miruje. Če smo frekvenco PWM signala nastavili na 50 Hz, pomeni, da bomo na polovici območja (90 °) z uporabo prejšnjih formul dobili vrednost 307. Ni nujno, da je naš servomotor ravno na polovici, ko dolžino pulza nastavimo na vrednost 307, ker na pozicijo vpliva nastavitev potenciometra. Če je slednji nastavljen točno na ničlo, potem se lahko držimo formule, drugače je treba testirati, kdaj je neki servomotor na njegovi polovici oz. miruje. V našem projektu smo dobili stanje mirovanja na vrednosti 322 (zato smo v naslednji kodi prišteli 322 referenčni vrednosti).

```
pwm.setPWM(0, 0, 322-int(ref*0.5));
```

Komentar: Motor za komponento yaw je na krmilniku PWM povezan na ničelnem vhodu (0 – prvi parameter v funkciji setPWM()). Konstanta 0.5 je določena tako, da preprečimo preveliko ojačanje povratno-zančnega sistema, ki bi povzročilo oscilacije.

3.2.11 Komunikacijski strežnik

Kot smo že omenili, pan-tilt enota mora sprejemati podatke od mobilne enote. Videli smo, da odjemalec (mobilna naprava) pošilja podatka yaw in pitch preko vmesnika socket. Tako mora tudi pan-tilt enota sprejemati podatke preko socketa. Delo strežnika opravlja mini računalnik Raspberry Pi. Komunikacijski strežnik tudi zažene dodaten proces, v katerem poteka upravljanje servomotorjev.

Najprej vidimo, kako ustvarimo proces. Proces se lahko ustvari znotraj drugega ali pa sprožimo vsakega v svoji konzoli. Če hočemo ustvariti proces znotraj drugega procesa, lahko to naredimo s pomočjo knjižnice node.js z imenom **child_ process** in metodo **spawn()** na naslednji način:

```
var spawn = require("child_process").spawn;
spawn('python',["hmc.py"]);
```

Pred zagonom strežnika, se lahko najprej povežemo na podatkovno bazo.

```
var mysql      = require('mysql');
var connection = mysql.createConnection({
        host     : 'localhost',
        user     : 'root',
        password : 'fmnt',
        port: '3306',
        database : 'or'
});
connection.connect();
```

Če vzamemo najprej proces, ki bo pričakoval zahtevo po povezavi odjemalca, lahko vidimo, kako to implementirati.

Naslednji sklop programske kode prikazuje, kako ustvariti poslušalca po povezavi druge naprave:

```
var http = require('http').Server(app);
http.listen(8080, function(){
        console.log('listening on 8080...');
});
```

V primeru povezave med odjemalcem in strežnikom je potem treba nadzorovati to, kar se piše in bere iz objekta socket. Najprej se ustvari socket za vhodne in izhodne operacije (pisanje in branje):

```
var io = require('socket.io')(http);
```

Z metodo on podamo parameter 'connection' in funkcijo, za katero hočemo, da se izvede med povezavo. Funkcija, za katero želimo, da se izvaja med povezavo, je poslušanje sporočil, ki pridejo od trenutnega povezanega odjemalca. Če je bilo sprejeto kakšno sporočilo, potem ga lahko z osebno funkcijo interpretiramo. To sporočilo vsebuje koordinate magnetometra pametnih naprav. Medtem ko smo sprejeli sporočilo, ga lahko preko podatkovne baze posredujemo drugemu procesu. Omenjen postopek se naredi na naslednji način:

```
io.on('connection', function(socket){
  console.log('a user connected');
  socket.on('message',function(msg){
        var post = {id:1,kot:msg.dir,tilt:msg.tilt}
        connection.query('UPDATE degree SET ?',post,function(e,r,f){
  });
});
```

Parameter **msg** vsebuje podatek tipa objekt, kjer imamo dve vrednosti: dir(yaw), tilt(pitch). Vsako to vrednost pa posredujemo podatkovni bazi tako, da vrstico z identifikatorjem 1 posodobimo vrednosti. Poglejmo si v celoti rešitev:

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

var mysql      = require('mysql');
var connection = mysql.createConnection({
        host     : 'localhost',
        user     : 'root',
        password : 'fmnt',
        port: '3306',
        database : 'or'
});
connection.connect();

http.listen(8080, function(){
    console.log('listening on 8080...');
});

io.on('connection', function(socket){
  console.log('a user connected');
  socket.on('message',function(msg){
        var post = {id:1,kot:msg}
    connection.query('UPDATE degree SET ?',post,function(e,r,f){
  });
});
```

Obvestilo: Strežnik se zažene z ukazom **node**.

4 Izvedba

4.1 Razvojno okolje

Če si želimo ustvariti aplikacijo za mobilne naprave z operacijskim sistemom Android, potem je treba uporabljati razvojno okolje z imenom Android Studio. V primeru za Windows Phone pa je treba uporabiti razvojno okolje Visual Studio (dovolj odprta verzija Community Edition). V naslednjih dveh podpoglavjih bomo videli, kako realizirati aplikacijo, ki temelji na spletu (WebView).

4.1.1 Android Studio

1. Najprej ustvarimo nov projekt z activity layout prazen.

2. Trenutna aplikacija ima svoj layout in moramo ga spremeniti v WebView tako, da odpremo activity layout, ki se nahaja v mapi res, in zamenjamo vsebino z:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/webview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
/>
```

3. Zdaj odpremo MainActivity.java in v metodi onCreate na koncu dodamo inicializacijo in povezavo na layout od WebView ter povezavo web strani, na katero se hočemo povezati. Dodati je treba:

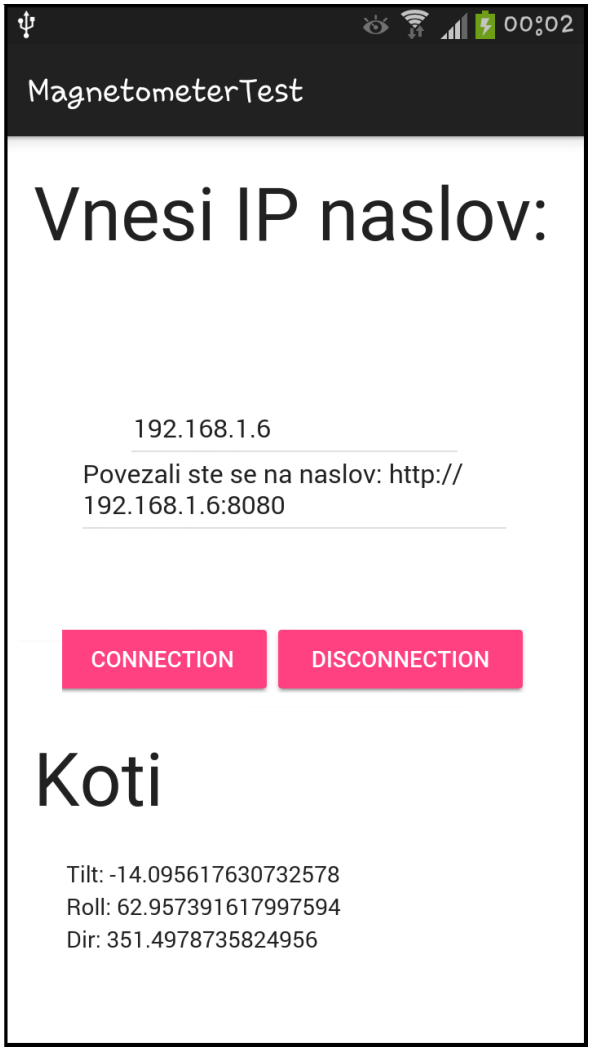
```
WebView myWebView = (WebView) findViewById(R.id.webview);
WebSettings webSettings = myWebView.getSettings();
myWebView.loadUrl("file:///android_asset/www/index.html");
```

4. Naša rešitev bo v skriptnem jeziku Javascript, zato je treba še to aktivirati z naslednjim ukazom:

```
webSettings.setJavaScriptEnabled(true);
```

5. Na zadnje je treba še ustvariti html datoteko (koda v prilogi z imenom app/src/main/assets/www/index.html)

Izgled android mobilne aplikacije (glej Sliko 14):



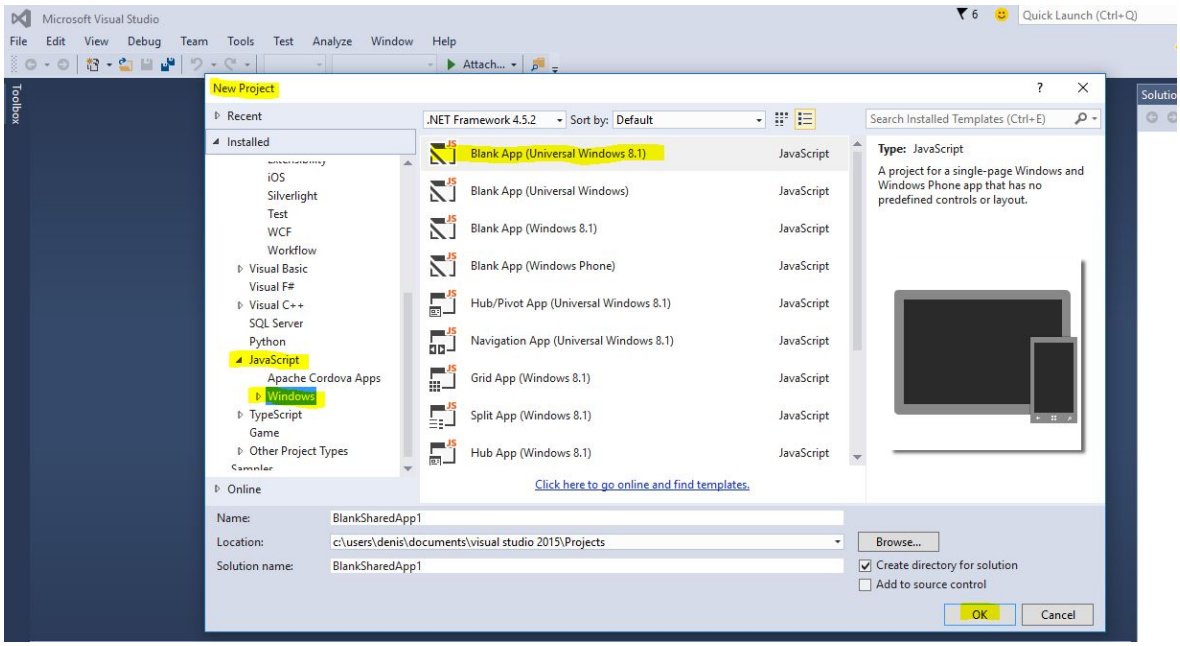
Slika 14: Android aplikacija - uporabniški vmesnik

4.1.2 Visual Studio

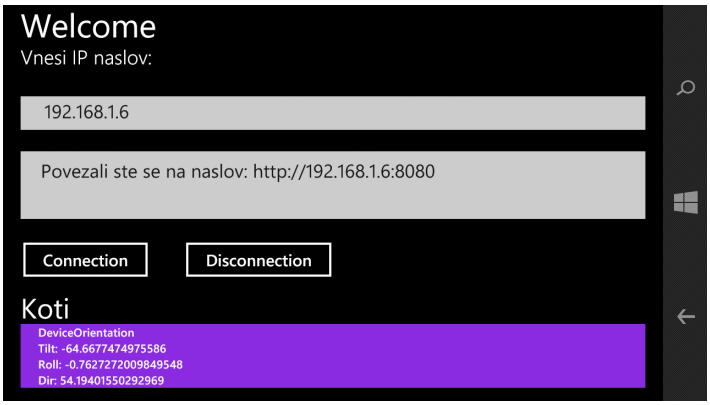
Na razpolago moramo imeti Visual Studio Community, ki vključuje paket za mobilne aplikacije JS (to opcijo imamo tudi med nalaganjem samega programa). Ustvarimo nov projekt: file → new → New project. Na Sliki 15 pa je evidentirano v rumenem to, kar je treba izbrati, in gumb OK, s katerim končamo.

Visual Studio ustvari sam WebView za pametne naprave z operacijskim mobilnim sistemom Windows Phone. V glavni mapi projekta je datoteka z imenom default.html. Dodamo še našo skripto in zamenjamo body značko z našo prejšnjo (koda v prilogi z imenom default.html).

Za videz Windows mobilne aplikacije glej Sliko 16.



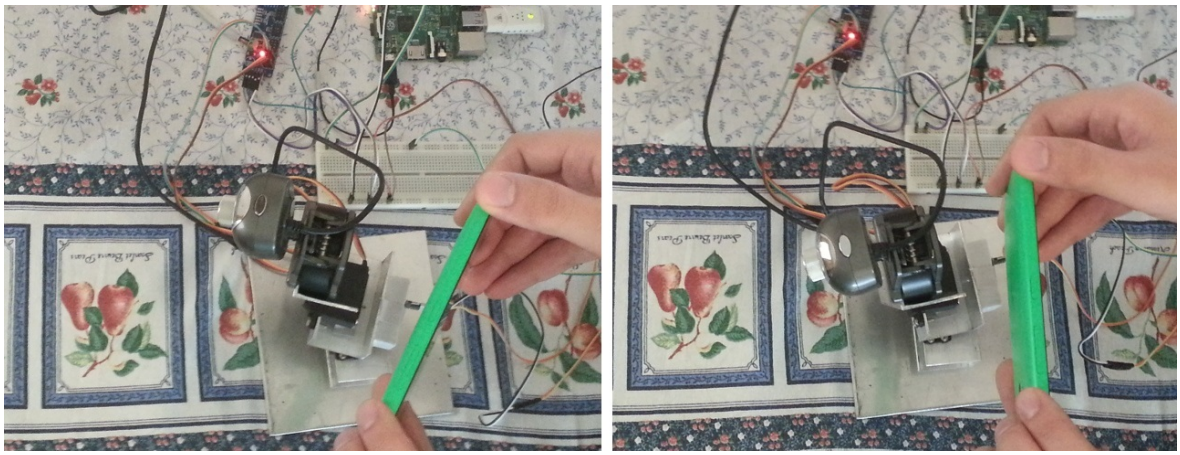
Slika 15: Visual studio - razvojno okolje



Slika 16: Windows Phone aplikacija - uporabniški vmesnik

5 Zaključek

Že med razvojem smo imeli veliko nepredvidenih težav. Med prvimi težavami, ki smo jih zasledili, je pozicioniranje magnetometra. Nismo predvideli, da servomotor oddaja magnetno polje, ki ob prisotnosti magnetometra zaustavi njegovo delovanje. Kako smo to ugotovili? Ko smo testirali delovanje samega magnetometra, smo ga imeli v roki. Delovalo je, ampak ko smo ga pozicionirali na tretji L-profil mehanizma, je ta prenehal delovati. Tu je pomembno, da je magnetometer odmaknjen od motorja in kovinskih delov. V naši izvedbi je bilo to tako, da smo oddaljenost od kovine dosegli s pritrditvijo magnetometra na stirodur. Zaradi lažje pritrditve pa smo vezje obrnili tako, da nam vrtenje okrog navpične osi (yaw) določata z in y komponenti magnetometra. Druga težava je bila v zakasnitvi rotacije mehanizma glede na mobilno napravo s katero mehanizem upravljamo. Opazili smo, da ima velik vpliv na to funkcija spawn(), ki ustvarja proces znotraj procesa strežnika. Poskusili smo zagnati oba procesa, ločeno vsakega v svoji konzoli, in rezultati so bili precej boljši. Na težavo smo naleteli tudi, ko smo se odločali za posredovanje podatkov. Preden smo se odločili za uporabo podatkovne baze, smo podatke zapisovali v eno datoteko. Problem nastane, ko oba procesa hkrati želita zapisovati oz. brati vrednost. Izbrali smo podatkovno bazo, ker ta že skrbi za dostop do vrednosti. V zaključni nalogi smo si najprej ogledali na grobo sistem oz. rešitev za naš sistem, ki krmili spletno kamero s pomočjo pametnih naprav. Nato smo tik za tem opisali tehnologijo in naprave, ki smo jih uporabljali, da bi se to vse realiziralo. Predstavljeni so bili postopki in kako je bila izpeljana programska koda, ki smo uporabljali. Rezultati so bili več kot dobri, kjer imamo pri krmiljenju majhno relativno napako +/– 2 ° (glej Sliko 17). Napaka je delno odvisna od magnetometra (po specifikacijah imamo toleranco +/– 1 °) in delno do nje pride zaradi servomotorja, ker ga nismo korigirali z nastavitvijo ničelnega kota. Lahko bi bile še kakšne izboljšave, če bi programsko kodo namesto v programskem jeziku Python izvedli v programskem jeziku C. Lahko bi tudi iz pametne naprave takoj zapisovali v podatkovno bazo in bi odpravili proces, ki sprejema vrednosti in jih zapisuje v podatkovno bazo. S tem bi lahko pridobili v času reakcije spletne kamere na premik uporabnika. Tudi tu ostane komunikacija preko TCP protokola. V prihodnosti bi lahko dodali tudi rotacijo Roll. Dodali bi še en servomotor, po možnosti manjši, kakor sta za yaw in pitch. Dovolj bi bilo krmiliti na isti način, kot smo rešili za Pitch rotacijo.



Slika 17: Rezultati

6 Literatura

[1] Adafruit, *ADAFRUIT 16-CHANNEL 12-BIT PWM/SERVO DRIVER - I2C INTERFACE*,
https://www.adafruit.com/product/815.

[2] Alex, *The quadcopter : control the orientation*,
http://theboredengineers.com/2012/05/the-quadcopter-basics/.

[3] *Android (operating system) - Wikipedia*,
https://en.wikipedia.org/wiki/Android_(operating_system).

[4] Circa Davide, *Socket: cosa sono e come funzionano - FortyZone*,
http://fortyzone.it/socket-cosa-sono/.

[5] Frances Reed, *How Servo Motors Work - Jameco ELECTRONICS*,
http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html.

[6] Android API, *Sensor types — Android Open Source Project*,
https://source.android.com/devices/sensors/sensor-types.html.

[7] Android API, *Building Web Apps in WebView — Android Developers*,
http://developer.android.com/guide/webapps/webview.html.

[8] *IDC: Smartphone OS Market Share 2015, 2014, 2013, and 2012*, IDC Research, Inc..
http://www.idc.com/prodserv/smartphone-os-market-share.jsp.

[9] Michael Horne, *Using a HMC5883L magnetometer/compass with the #RaspberryPi — Raspberry Pi Pod*,
http://www.recantha.co.uk/blog/?p=2547.

[10] Node JS, *Node.js v5.1.0 Documentation*,
https://nodejs.org/api/child_process.html.

[11] Peter Krkoč, *Programiranje z Arduino (6) - I2C vodilo*,
http://www.svet-el.si/o-reviji/programiranje/2716-216-43.

[12] Claudio Pomes, *Differenza tra TCP e UDP nella trasmissione di dati su internet*,
http://developer.android.com/guide/webapps/webview.html.

[13] Raspberry Pi Foundation, *GPIO: Raspberry Pi Models A and B - Raspberry Pi Documentation*,
https://www.raspberrypi.org/documentation/usage/gpio/README.md.

[14] Raspberry Pi Foundation, *What is a Raspberry Pi*,
https://www.raspberrypi.org/help/what-is-a-raspberry-pi/.

[15] Kevin Townsend, *Adafruit 16 Channel Servo Driver with Raspberry Pi*,
https://learn.adafruit.com/adafruit-16-channel-servo-driver-with-raspberry-pi.

[16] *Servo Motor — Servo Mechanism — Theory and Working Principle*,
http://www.electrical4u.com/servo-motor-servo-mechanism-theory-and-working-principle/.

[17] *Windows Phone - Wikipedia*,
https://en.wikipedia.org/wiki/Windows_Phone.

Priloge

Priloga A: Izvorna koda mobilne in pan-tilt enote (zgoščenka)

- Windows Phone aplikacija
  - TestMagnetometerJS/default.html
- Android aplikacija
  - MagnetometerTest/app/src/main/assets/www/index.html
  - Knjižnica uporabljena - socket.io.js
- Pan-tilt enota
  - Upravljanje servo motorjev in merjenje orientacije iz magnetometra - hmc.py
  - Strežnik - server.js
  - Knjižnica (upravljanje servo motorja) - Adafruit_I2C.py
  - Knjižnica (upravljanje servo motorja) - Adafruit_PWM_Servo_Driver.py
  - Knjižnica za sprejem podatkov - socket.io.js