Vođenje robota primjenom interneta

Hrnjić, Nino

Undergraduate thesis / Završni rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:690195

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-26**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

Sveučilište u Zagrebu

Faklutet strojarstva i brodogradnje

ZAVRŠNI RAD

Nino Hrnjić

Zagreb, 2015.

Sveučilište u Zagrebu

Faklutet strojarstva i brodogradnje

ZAVRŠNI RAD

Mentor:

prof. dr. sc. Mladen Crneković

Nino Hrnjić

Zagreb, 2015.

Izjava

Izjavljujem da sam ovaj rad izradio samostalno koristeći se znanjem stečenim tijekom studija i navedenom literaturom.

Zahvala

Srdačno se zahvaljujem voditelju rada prof. dr. sc. Mladenu Crnekoviću na predloženoj temi, pristupačnosti, te na mnogim korisnim savjetima. Također se zahvaljujem svojim prijateljima i obitelji koji su mi neizmjerno pomogli svojim razumijevanjem i podrškom.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum 07 -09- 2015	Prilog
Klasa: 602-04/15-6/3	
Ur.broj: 15-1703-15-287	

ZAVRŠNI ZADATAK

Student: **NINO HRNJIĆ** Mat. br.: 0035175169

Naslov rada na hrvatskom jeziku: **VOĐENJE ROBOTA PRIMJENOM INTERNETA**

Naslov rada na engleskom jeziku: **INTERNET REMOTE CONTROLLED ROBOT**

Opis zadatka:

Zbog visoke cijene većini ljudi industrijski robot još uvijek nije dostupan. Kako bi širili znanje i vještine vezane uz robotiku moguće je zainteresiranima dati pristup robotu preko interneta. Tako bi stekli neka indirektna iskustva, a jedini trošak bio bi trošak vremena.

Potrebno je projektirati grafičko sučelje koje će internetskom posjetitelju omogućiti pokretanje robota u realnom vremenu uz odgovarajući video nadzor.

Tražena rješenja:

- opisati naredbe odabranog robota kojima će korisnik pokretati robota,
- dizajnirati grafičko sučelje za korisnika koje će uključiti i "živu" sliku,
- definirati vezu između serverske aplikacije prema korisniku i veze prema robotu.

Zadatak zadan:
25. studenog 2014.

Zadatak zadao:
Prof. dr. sc. Mladen Crneković

Rok predaje rada:
1. rok: 26. veljače 2015.
2. rok: 17. rujna 2015.

Predviđeni datumi obrane:
1. rok: 2., 3., i 4 . ožujka 2015.
2. rok: 21., 22., i 23. rujna 2015.

Predsjednik Povjerenstva:
Prof. dr. sc. Zoran Kunica

Sadržaj

Sadržaj ii

Sažetak iii

Popis oznaka iv

Popis slika v

Popis tablica vi

1 Uvod 1

2 Robot Mitsubishi RM501 2

2.1 Naredbe za upravljanje robotom 5

3 Veza između web preglednika i servera, te servera i kinematičkog kontrolera robotske ruke 6

3.1 HTTP 6

3.2 WebSocket 7

3.3 RFCOMM 7

4 Web Server i programski jezik serverskih aplikacija 8

4.1 Python 8

4.2 Motion 9

5 Web preglednik i programski jezici za izradu web sučelja 10

5.1 HTML 10

5.2 CSS 11

5.3 JavaScript i jQuery 11

6 Realizacija Web aplikacije 12

6.1 Skripta korisničke strane - login.html 12

6.2 Skripta korisničke strane - site.html 13

6.3 Python Tornado web server aplikacija 16

6.4 Python Tornado WebSocket aplikacija 19
6.5 Motion webcam serverka aplikacija 21

7 Zaključak **23**

Literatura **24**

Prilozi **25**

Skripta login.html 25
Skripta site.html 27
Skripta MainA.py 35
Skripta wshandler.py 35

Sažetak

Tema ovog završnog rada je izrada web aplikacije sa živom slikom za upravljanje robotskom rukom Mitsubishi RM501. Za poslužitelja aplikacije odabrano je računalo sa Ubuntu Linux operativnim sustavom. Serverska strana aplikacije je napravljena u Python programskom jeziku uz upotrebu Tornado knjižnice koja služi kao server i razvojni okvir za izradu web aplikacija. Također je na serverskoj strani, upotrebom gotove Motion aplikacije i web kamere, preko računalnog kanala dostavljena živa slika napravljenoj aplikaciji. Programski kod korisničke strane kojim je izrađeno cijelo grafičko sučelje i dio aplikacijske logike, a koji se izvršava u korisničkom web pregledniku, napisan je upotrebom JavaScript, CSS, HTML standardnih jezika za izradu web stranica.

Ključne riječi : Web; Mitsubishi RM501; RFCOMM; WebSocket; HTTP; Python; Motion; JavaScript; HTML; CSS;

Popis oznaka

Oznaka	Opis
p_x	Pozicija prihvatnice
p_y	Pozicija prihvatnice
p_z	Pozicija prihvatnice
φ	Orijentacija prihvatnice (nagib)
ψ	Orijentacija prihvatnice (valjanje)
q_1	Upravljane koordinate robota (struk)
q_2	Upravljane koordinate robota (rame)
q_3	Upravljane koordinate robota (lakat)
q_4	Upravljane koordinate robota (nagib)
q_5	Upravljane koordinate robota (valjanje)

Popis slika

2.1 Mitsubishi RM501 robotski sustav 2
2.2 Stupnjevi slobode gibanja robota Mitsubishi RM501 3
2.3 Dimenzije pojedinih članaka robota Mitsubishi RM501 3
6.1 Početna stranica aplikacije 12
6.2 Dio koda login.html 13
6.3 Sučelje prije spajanja s robotom 14
6.4 HTML kod tipke connect 14
6.5 JavaScript kod tipke connect 15
6.6 JavaScript kod tipke open 15
6.7 Sučelje nakon spajanja s robotom 16

Popis tablica

2.1 Osnovne karakteristike robota Mitsubishi RM501 4
2.2 Tablica ili popis ili lista naredbi 5
2.3 Popis pogrešaka Ex 5
6.1 Parametri Motion aplikacije 22

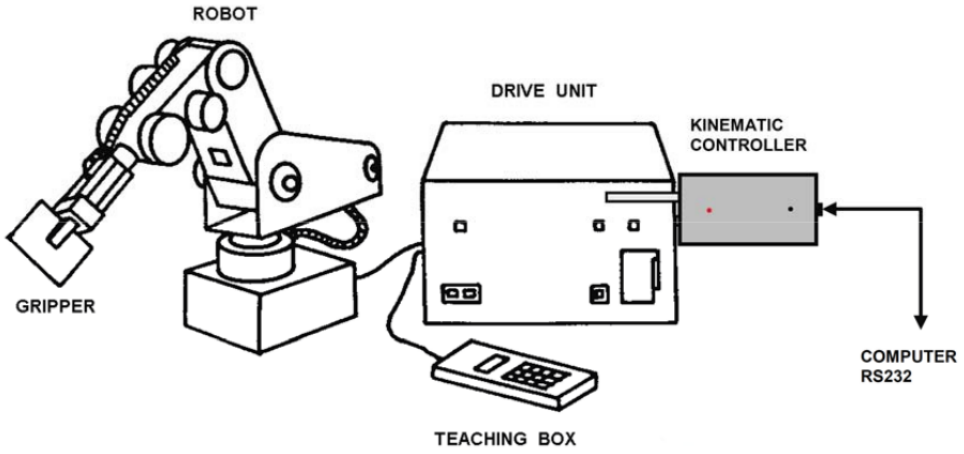
1 Uvod

Za ovaj rad bilo je potrebno osmisliti i izraditi web aplikaciju sa živom slikom za upravljanje robotske ruke Mitsubishi RM501. Web aplikacija je računalni program koji koristi web preglednik kao korisničko sučelje. Rastući broj web aplikacija koji proizlazi iz današnje sveprisutnosti interneta i praktičnosti upotrebe web preglednika kao korisničkog sučelja, a zbog lakoće nadopunjavanja i održavanja aplikacije bez potrebe distribuiranja i instaliranja programske podrške kod potecijalnih mnogobrojnih korisnika kao i zbog svoje inherentne više platformske podrške, predstavlja trend tvrtkama koje se bave aplikacijskom programskom podrškom da omoguće dostupniju prisutnost programa koji su dosad bili dostupni samo kao lokalne aplikacije. Njen kod korisničke strane se piše u programskim jezicima za preglednike. Za potrebe ovog rada korišteni su HTML, CSS i JavaScript sa njegovom knjižnicom jQuery, a njima je kodirana sva prezentacija (GUI) aplikacije te dio aplikacijske logike. Ostali dio aplikacijske logike izvršava se na serveru koji i omogućava dostupnost aplikacije web pregledniku i neizostavan je dio strukture web aplikacije. Za ovu aplikaciju kao server je odabran Python Tornado web server koji bi se pokretao na Ubuntu Linux operativnom sustavu, te je u skladu s time sva aplikacijska logika servera pisana u Python programskom jeziku. Na serverskoj strani korištenjem Motion aplikacije putem računalnih portova dostavljena je i ziva slika samog robota korisničkom agentu.

U nastavku ovog rada bit će riječi o robotu Mitsubishi RM501 i njegovom kinematičkom kontroleru. Objasnit će se komunikacijski protokoli koji povezuju korisničko web sučelje i server, te server i kinematički kontroler robota. Potom će se objasniti sastavni djelovi web aplikacije kao što su web preglednik i web server skupa sa njihovim programskim jezicima, te će u zadnjem poglavlju biti objašnjen kod skripti izrađene web aplikacije i podešavanje gotove Motion aplikacije.

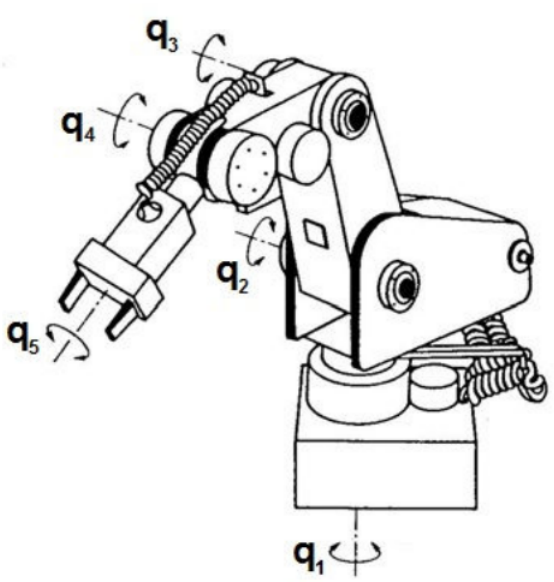
2 | Robot Mitsubishi RM501

Mitsubishi RM501 je robot starije tehnologije, nastao u drugoj polovini 20. stoljeća. Zbog problema u serijskoj komunikaciji između robota i novijih računala za njega je razvijen kinematički kontroler koji služi kao posrednik u komunikaciji između računala i robota. U razvoju kinematičkog kontrolera primjenjen je mikrokontroler proizvođača Atmel baziran na procesoru serije 8051. U njemu je pohranjen kinematički model robota koji omogućava upravljanje robotom odnosno njegovo pozicioniranje u prostoru unutarnjih ili vanjskih koordinata. Kinematički kontroler također brine da nebi došlo do sudara robota i podloge, mjeri vrijeme izvođenja zadatka, te se pomoću njega mogu definirat i neke dodatne opcije poput brzine, otvorenosti hvataljke, dužine alata. Sve naredbe za upravljanje dolaze putem serijske veze od nadređenog računala, a zadaje ih korisnik putem korisničkog sučelja na računalu. Od kontrolera se paralelnom vezom instrukcije za upravljanje prenose do robota.



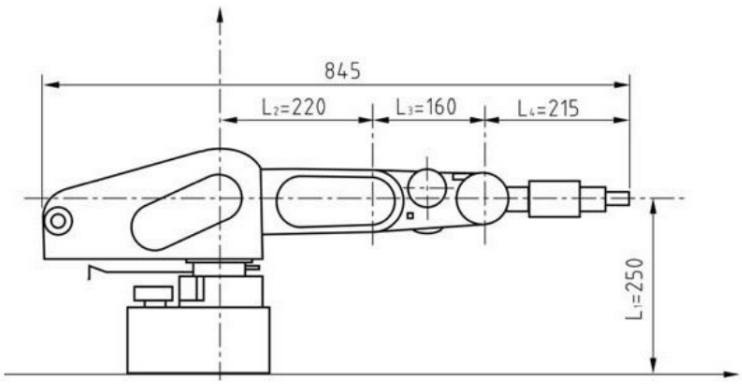
Sl. 2.1: Mitsubishi RM501 robotski sustav

Robot ima 5 stupnjeva slobode gibanja slika i svi su ostvareni rotacijskim zglobovima, svaki sa pripadajućim elektromotorom kojim vrši zakret određenog članka. Robot je opremljen prihvatnicom sa dva prsta koja mu daje mogućnost hvatanja i držanja predmeta. Shematski prikaz robota i njegovih stupnjeva slobode gibanja dan je Sl. 2.2, a pregled osnovnih karakteristika robota prikazan je u Tab. 2.1.



Sl. 2.2: Stupnjevi slobode gibanja robota Mitsubishi RM501

Položaj prihvatnice može se opisati pomoću vektora vanjskih $r = [\ p_x\ p_y\ p_z\ \varphi\ \psi\ ]^T$ i vektora unutarašnjih koordinata $q = [\ q_1\ q_2\ q_3\ q_4\ q_5\ ]^T$. Na Sl. 2.3 su prikazane osnovne dimenzije robota iz kojih se upotrebom matrica homogenih transformacija izrađuje kinematički model.



Sl. 2.3: Dimenzije pojedinih članaka robota Mitsubishi RM501

Masa sustava	27 kg robot i 23 kg upravljačka jedinica
Priključci	paralelno i serijski (RS232)
Pogonski sustav	DC motori (24 V)
Mjerenje kuta zakreta zgloba	Optički enkoder
Rotacija struka q_1	300°
Rotacija ramena q_2	130°
Rotacija lakta q_3	90°
Nagib q_4	± 90°
Valjanje q_5	± 180°
Dopuštena nosivost	1, 2 kg (uključujući masu hvataljke)
Max. brzina prihvatnice	400 mm/s
Ponovljivost	± 0, 5 mm

Tab. 2.1: Osnovne karakteristike robota Mitsubishi RM501

2.1 Naredbe za upravljanje robotom

Upravljanje robotom ostvareno je posredovanjem kinematičkog kontrolera koji sadrži kinematički model robota i omogućava jednostavno upravljanje robotom pomoću unutarnjih ili vanjskih koordinata uzimajući u obzir fizičke ograničenosti robota i brineći da ne dođe do sudara robota sa podlogom. Upravljanje se svodi na slanje niza naredbi za koje s obzirom na mogućnost izvođenja kinematički kontroler vraća odgovor ili signalizira pogrešku. Naredbe korištene u ovom radu prikazane su i kratko objašnjene u Tab. 2.2 i predstavljaju samo neke od mogućih naredbi, a Tab. 2.3 daje prikaz mogućih pogrešaka.

Naredba	**Odgovor**	**Opis**
x127.3y-215.7z64.1f-32.9p20.1	Ex	Zadavanje apsolutne pozicije robotu
Dx10.5y20z-10f0p10	Ex	Zadavanje relativne pozicije robotu
SPEED=7	Ex	Postavljanje brzine
OPEN	E0	Otvaranje hvataljke
CLOSE	E0	Zatvaranje hvataljke
INIT	E0	Inicijalizacija robota
HOME	Ex	Postavljanje hvataljke u poziciju HOME
?R	#R=(127.3,-215.7,64.1,-32.9,20.1)	Vrijednost vektora R
?INIT	#INIT=NO ili YES	Status inicijalizacije

Tab. 2.2: Tablica ili popis ili lista naredbi

x	**Opis**
0	Nema pogreške
1	Parametar a1 izvan dosega
2	Parametar a2 izvan dosega
3	Parametar a3 izvan dosega
4	Parametri a4,a5 izvan dosega
5	Ulazni parametar izvan dosega
6	Točka izvan dosega robota
7	Robot nije inicijaliziran
8	Opasnost od sudara
9	Nepoznata naredba

Tab. 2.3: Popis pogrešaka Ex

3 Veza između web preglednika i servera, te servera i kinematičkog kontrolera robotske ruke

Web server i web preglednik povezani su preko internetske mreže njenim TCP/IP skupom protokola uz upotrebu HTTP (Hypertext Transfer Protocol) i WebSocket protokola aplikacijskog sloja, a veza web servera i kinematičkog kontrolera robota je ostvarena preko bluetooth bežične mreže upotrebom RFCOMM (Radio frequency communication) protokola transportnog sloja. U sljedećim podpoglavljima opisati će se samo neposredno korišteni protokoli, te se zbog opširnosti neće ulaziti u detalje mrežnih modela i njenih protokola koji se hijerarhijski nalaze u nižem sloju.

3.1 HTTP

Hypertext Transfer Protocol (HTTP) je protokol aplikacijskog sloja koji omogućava prijenos datoteka koje u sebi sadrže veze na druge dokumente. Takvi dokumenti označavaju se kao hipertekst, a veze koje sadrže nazivaju se hiper-tekstualne veze. Primjenjuje se od 1990. godine pojavom usluge interneta World Wide Web. Temelji se na modelu klijent-poslužitelj, a za prijenos podataka najčešće se oslanja na pouzdani TCP protokol. Standardno je priključna točka (engl. port) na kojoj sluša web poslužitelj 80, ali se može definirati drugačije.

HTTP protokolom definiraju se:

- forma komunikacije između klijenta i poslužitelja,
- kodiranje znakova karakterističnih za brojne jezike (engl. character set),
- kodiranje sadržaja (engl. content coding),
- pristup dokumentima za različite tipove protokola,
- pristup dokumentima uz provjeru identiteta (autorizacija i autentikacija),
- pohrana dokumenata u privremenu memoriju (engl. caching),
- sigurnosni aspekti osjetljivih točka u komunikaciji između klijenta i poslužitelja.

3.2 WebSocket

WebSocket je protokol aplikacijskog sloja koji omogućava dvosmjernu, istodobnu (engl. full-duplex), dugotrajnu komunikaciju poslužitelja i klijenta preko jednog TCP (Transmission Control Protocol) kanala. Standardiziran 2011. godine napravljen je sa idejom da zamijeni već postojeće dvosmjerne komunikacijske tehnologije koje koriste HTTP kao transportni sloj. Dotadašnje tehnologije napravljene su kao kompromis između efikasnosti i pouzadnosti zbog toga što HTTP incijalno nije zamišljen da bude korišten kao dvosmjerna komunikacija. WebSocket je neovisan TCP-baziran protokol čija jedina veza sa HTTP-om je to što je njegova inicijalizirajuća pozdravna poruka (engl. handshake) interpretirana od HTTP servera kao zahtjev za prebacivanje protokola (engl. upgrade request). Zbog trajno otvorenog TCP kanala koje mu je prethodio samo jedan inicijalizirajući pozdrav, za razliku od dotadašnjih tehnologija sa više TCP kanala čija izmjena informacija od klijenta prema serveru je uvjek iznova prenosila HTTP zaglavlje (engl. header), WebSocket smanjuje kašnjenje te je pogodan za aplikacije čiji se rad treba odvijati u realnom vremenu.

3.3 RFCOMM

Radio frequency communication (RFCOMM) je Bluetooth protokol transportnog sloja napravljen nad L2CAP protokolom kako bi oponašao kanal RS-232 serijske veze. Slično kao TCP pruža mogućnost jednostavnog i pouzdanog transfera podataka korisniku. Njegovom upotrebom aplikacije koje koriste komunikaciju preko serijske veze mogu se jednostavno prebaciti na bežičnu Bluetooth komunikaciju.

4 Web Server i programski jezik serverskih aplikacija

Web server je računalo koje posjeduje serversku programsku podršku i primarno služi dostavljanju web stranica web pregledniku na način da zatraženi resurs koji se nalazi u njegovoj memoriji prenosi putem globalne internetske mreže upotrebom HTTP protokola. Na serverskoj strani se mogu nalaziti i skripte koje omogućuju dodatne funkcije i dinamičku web stranicu. Za potrebe ovoga rada osmišljen je web server koji bi se nalazio na računalu sa Ubuntu Linux operativnim sustavom. Osmišljeni server se sastoji od tri serverske aplikacije. Jedna od aplikacija služi za prosljeđivanje žive slike putem računalnog porta sa web kamere prema web aplikaciji. To je gotova i besplatna Motion webcam server aplikacija dostupna samo za Linux operativne sustave. Nakon instalacije Motion aplikacije potrebno je samo podesiti konfiguracijsku tekstualnu datoteku te je pokrenuti paralelno sa ostalim serverskim aplikacijama. Preostale dvije aplikacije, pisane Python programskim jezikom uz upotrebu Tornado knjižnice koja predstavlja razvojini okvir za izradu web i asinkronih komunikacijskih mreža, služe ostvarivanju veze između web preglednika i servera, te servera i kinematičkog kontrolera robotske ruke. Primarnu funkciju web servera, da na zahtjev korisnika upotrebom HTTP protokola dostavi HTML i s njim povezane resurse web pregledniku, obavlja Tornado web server aplikacija. Ona također sadrži i aplikacijsku logiku za autentifikaciju korisnika. Posljedna Tornado WebSocket server aplikacija služi implementaciji RFCOMM i WebSocket komunikacijskih protokola na serverskoj strani. WebSocket protokol služi dvosmjernoj komunikaciji između servera i korisničkog preglednika, a upotrebom RFCOMM protokola ostvarena je veza između serverskog računala i kinematičkog kontrolera robotske ruke. U sljedećim poglavljima bit će objašnjene napisane serverske skripte i podešavanje konfiguracijske datoteke Motion aplikacije, a u narednim podpoglavljima pokazat će se osnovne značajke Python programskog jezika i Motion aplikacije.

4.1 Python

Python je interpreterski, interaktivni, objektu orjentirani programski jezik, koji na optimalan način ujedinjuje najbolje ideje i načela rada drugih programskih jezika. Spoj

je tradicionalnih skriptnih jezika i sistemskih jezika, jer nudi jednostavnost skriptnih jezika i napredne programske alate koji se tipično nalaze u sistemskim razvojnim jezicima. Njegova jednostavnost omogućuje programeru više razmišljanja o problemu nego o jeziku. Python je besplatan (za akademske ustanove i neprofitnu upotrbu), otvorenog koda (engl. open-source), s jako dobrom potporom, literaturom i dokumentacijom. Njegova proširiva knjižnica pokriva sve od funkcija operacijskog sustava do struktura podataka potrebnih za gradnju web servera. Izvođenje Python programa može se izvoditi interaktivno unutar Python interpretera, moguće ga je izvoditi umetnutog unutar drugih programa, te može biti spremljen u skripte koje se pozivaju iz sistemske linije. Spremanje u skripte izvodi se iz običnog tekstualnog ili nekog drugog podržanog editora na način da se nakon željenog imena tekstualna datoteka završi sa .py. Za izradu Python skripta serverskih aplikacija korištena je dodatno instalirana Tornado knjižnica. Tornado knjižnica služi kao web server i kao razvojni okvir za izradu web aplikacija.

4.2 Motion

Motion je besplatan, otvorenog koda (engl. open-source) program koji služi praćenju video signala iz jedne ili više kamera uz mogućnost detekcije promjene slike. Također ima ugrađen jednostavni webcam server koji je u mogućnosti dati video mjpeg formata. Program je napisan u C programskom jeziku i napravljen je za Linux operativne sustave. Ne posjeduje grafičko sučelje za upravljanje, već se u potpunosti podešava preko terminala ili upotrebom konfiguracijskih datoteka.

5 Web preglednik i programski jezici za izradu web sučelja

Web preglednik je programska podrška za dohvaćanje, prikazivanje i razmijenu informacija preko globalane mreže. U klijent server modelu preglednik je klijent koji se pokreće na računalu i stupa u vezu sa web serverom te od njega zahtjeva informacije, uključujući HTML, CSS, JavaScript, slike, video i druge resurse. Web server dostavlja zatražene informacije pregledniku koji ih kao rezultat iscrtava na korisničkom računalu. Proces dohvaćanja informacija započinje unosom URL adrese u preglednik. URL je akronim za Uniform Resource Locator, u prijevodu ujednačeni lokator sadržaja i predstavlja standard kojim se definira lokacija informacijskog resursa i metoda kojom ju je potrebno dohvatiti. Njegova najčešća sintaksa izgleda u obliku metoda://IP_adresa:broj_kanala/dodatni_put. Metoda koja je najčešće i ime protokola koji se koristi a može biti http, https, ftp, ws i mnoge druge.

Internet Protocol address je numerička oznaka dodijeljena svim uređajima koji sudjeluju u mreži i koji koriste internet protokol za razmijenu informacija. Uz plaćenu uslugu ona može biti povezana sa imenom domene koja se pomoću zato namjenjenog sustava (Domain Name System) prevodi u numeričku IP adresu. Broj kanala nije obvezan, jer ukoliko se izostavi bit ce korišten standardni kanal protokola, npr. 80 za http ili 443 za https. Dodatni put predstavlja različite resurse na istom web serveru. Najpoznatiji web preglednici su Internet Explorer, Firefox, Google Chrome, Opera i Apple Safari. Za potrebe ovog rada korištenjem HTML, CSS, i JavaScript jezika su napisana dva HTML dokumenta koja predstavljaju početnu autentifikacijsku web stranicu i glavnu web stranicu koja služi za upravljanje robotom Mitsubishi RM501. Detalji spomenutih dokumenata obradit će se u sljedećim poglavljima, a u narednim podpoglavljima će se iznijeti osnovne značajke jezika korištenih za njihovu izradu.

5.1 HTML

HTML je akronim za HyperText Markup Language, a predstavlja jezik koji se koristi za izradu web stranica. HTML definira strukturu i organizaciju web dokumenta uz pomoć različitih HTML znakova (engl. tag) i njihovih atributa. Svaki HTML dokument počinje

<html> znakom i završava </html> znakom, te se sastoji od zaglavlja i tjela. Zaglavlje se odvaja znakovima <head> i </head> i unutar njega se stavljaju sve informacije o stranici koje se ne prikazuju u prozoru preglednika. Tijelo se odvaja znakovima <body> i </body> i unutar njega se stavljaju sve informacije koje se žele prikazati korisnicima. Atributima unutar znakova se određuju dodatna svojstva znaka kao na primjer ime ili klasa kojima se referenciraju pojedinačni znakovi ili grupa znakova. Unutar HTML dokumenta mogu se ugraditi skripe jezika kao JavaScript koje utječu na ponašanje web stranica i CSS skripte koje definiraju izgled HTML elemenata. HTML znakovi također mogu hipervezama (engl. hyperlink) dostavljati druge informacijske resure. Od 2014 je u upotrebi HTML5 standard čija glavna novina se odnosi na način kojim HTML podržava web aplikacije.

5.2 CSS

CSS je akronim od Cascading Style Sheets, stilski jezik koji predstavlja novu značajku HTML-a koja olakšava i pruža bolju kontrolu nad prikazom web stranica. Prije njega prikazivanje sadržaja određivalo se HTML znakovima. CSS se unutar HTML dokumenta može pisati na dva načina. Prvi način je da se umetne u zaglavlju dokumenta između znakova <style> i </style>, a drugi način je umetanje unutar samih HTML znakova, npr. TEKST što kao rezultat daje riječ TEKST u plavoj boji.

5.3 JavaScript i jQuery

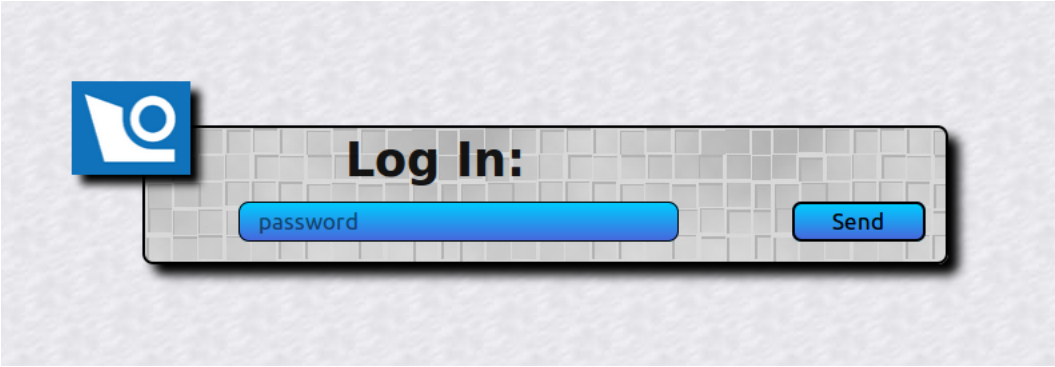
JavaScript je skriptni programski jezik koji se prvenstveno koristi za definiranje funkcionalnosti web stranica. Jednostavno se poveziva sa HTML elementima i izvršava se na klijentskoj strani. Također se može koristiti i za neke druge svrhe poput programiranja serverske strane u okruženjima kao Node.js te kao skripta unutar pdf dokumenta. Unutar HTML dokumenta stavlja se u zaglavlje sa početnim znakom <script> i završnm znakom </script>. Upotrebom njegove najpopularnije knjižnice jQuery olakšava se sintaksa i izvođenje najčešćih manipulacija nad HTML elementima.

6 Realizacija Web aplikacije

Za rad napravljene web aplikacije potrebno je serversko računalo sa Linux operativnim sustavom na kojem su pokrenute serverske aplikacije, web kamera, korisničkog računalo sa web preglenikom i pristupom komunikacijskoj mreži web servera, te serversko računalo treba biti povezano sa kinematičkim kntrolerom robota Mitsubishi RM501 pomoću bluetooth veze. Web aplikacija se sastoji od dvije skripte napisane u jezicima za web preglednike čije izvršavanje se izvodi od strane korisničkog agenta, te tri paralelno pokrenute serverske aplikacije od kojih su dvije napisane u Python programskom jeziku, a jedna je gotova aplikacija za koju je samo potrebno prije pokretanja podesiti konfiguracijsku datoteku. U sljedećim podpoglavljima objasnit će se napisane skripte i podešavanje konfiguracijske datoteke Motion serverske aplikacije.

6.1 Skripta korisničke strane - login.html

Skripta login.html predstavlja početnu stranicu za korisnika aplikacije s pomoću koje se izvršava autentifikacija nakon koje se pristupa grafičkom sučelju za upravljane robotom. Funkcionalnio najbitniji dio koda login.html prikazan je Sl. 6.2, a ostali dio koda napisan u CSS-u, koji se odnosi na graficki prikaz, nije bitan za funkcionalnost same aplikacije i neće se zbog opsežnosti posebno objašnjavati već će se dati u prilogu. Ovdje će se samo pokazati njegov krajnji efekt u obliku Sl. 6.1.



Sl. 6.1: Početna stranica aplikacije

Kao što je već rečeno početna stranica login.html služi za autentifikaciju korisnika preko zadane lozinke. Sama provjera ispravnosti lozinke pripada serverskoj aplikaciji, tako

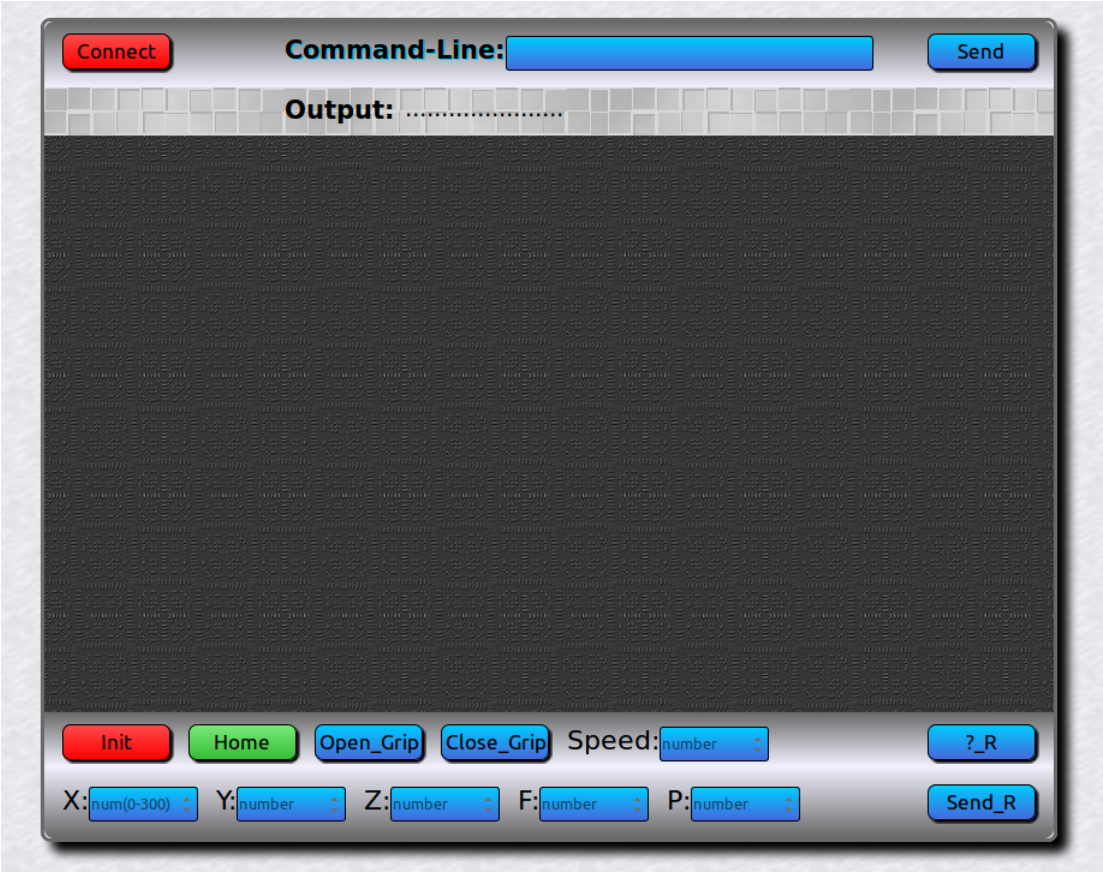
da je funkcija login.html stranice da omogući upis i slanje lozinke prema serveru. Upis i slanje lozinke ostvareno je pomoću dva HTML `<input>` znaka koja se nalaze unutar HTML `<form>` znaka. Prvi `<input>` znak je vidljiv na liniji 102, Sl. 6.2 i stvara polje za unos lozinke, dok drugi predstavlja tipku pomoću koje se obrazac šalje. Slanje lozinke pokreće akciju koja je definirana kao atribut HTML `<form>` znaka. U našem slučaju za akciju je zadan checkme.html koji se ostvaruje metodom get, što zapravo predstavlja slanje lozinke na server preko pod adrese naše stranice tipa adresa_stranice/checkme.html, koja je definirana unutar servera pomoću aplikacijkog objekta Tornado web server aplikacije.

```
<form action="checkme.html" method="get">
    <input type="text" name="pass"  class="inputlogin" id="inpass" placeholder="    password">
    <input type="submit" value="Send" class="inputlogin" id="inbtn">
</form>
```

Sl. 6.2: Dio koda login.html

6.2 Skripta korisničke strane - site.html

Nakon uspješne autentifikacije korisniku se unutar preglednika iscrtava site.html koji predstavlja grafičko sučelje za upravljane robotskom rukom, prikazano Sl. 6.3. U nastavku će se, zbog opsežnosti, pokazati samo dio koda iz kojeg se može vidijeti funkcionalnost, te veza sa serverom, a cijeloviti kod bit će dan u prilogu. Živa slika aplikaciji je dostavljena preko računalnog kanala uz pomoć Motion web cam serverske aplikacije, a naredbe se nakon slanja putem WebSocket protokola prosljeđuju WebSocket serverskoj aplikaciji koja ih dalje prosljeđuje robotu. Kao što je spomenuto, site.html sadrži dio aplikacijske logike same aplikacije. Taj dio je pisan pomoću JavaScript programskog jezika koji se izvršava unutar korisničkog preglednika, a jednostavno se poveziva sa html znakovima kojima je definirana sama stranica.



Sl. 6.3: Sučelje prije spajanja s robotom

Da bi se spojio sa robotom i njegovim video nadzorom korisnik mora pritisnuti tipku connect, koja je unutar site.html predstavljena html znakom `<button>` čiji atribut `id` je nazvan takoder connect, a prikazan je Sl. 6.4.

```
294                <button id="connect" class="btn" type="submit">Connect</button>
```

Sl. 6.4: HTML kod tipke connect

Pritiskom tipke connect korisnik će aktivirati JavaScript kod, koji je prikazan Sl. 6.5. Tipka connect sa JavaScript kodom povezana je korištenjem jQuery JavaScript knjižnice, na način da atribut `id` html znaka referencira sa znakom `#` iza kojeg ide vrijednost id polja, u ovom slučaju connect. Unutar JavaScript funkcije tipke connect definirane su dvije varijable `ipws` i `ipimg` koje služe za preglednije spremanje URL addresa, a sastoje se od protokola, IP adrese i broja kanala. Varijabla `ipws` korištena je prilikom otvaranja WebSocket komunikacije što se unutar JavaScripta ostvaruje s kodom koji se nalazi na

liniji 18. Drugi kraj WebSocket kanala se nalazi na serveru, u zato napravljenoj serverskoj aplikaciji, i naknadno će biti opisan. Linija 19 predstavlja dio koda koji poveziva html `<img>` znak sa atributom `id` nazvanim img sa variablom `ipimg` koja predstavlja addresu na kojoj se sa servera, pomoću Motion serverske aplikacije, šalje vremenski promjenjljiva slika robota. Sučelje nakon spajanja je prikazano na Sl. 6.7.

```
11  <script type="text/javascript" >
12
13      $(document).ready(function(){
14
15          $("#connect").click(function(){
16              ipws="ws://127.0.0.1:5555"
17              ipimg="http://127.0.0.1:8081"
18              connecttows=new WebSocket(ipws);
19              $("#img").attr("src", ipimg);
20              connecttows.onmessage = function(event){
21                  $("#outCL").text(event.data);
22              };
23          });
```

Sl. 6.5: JavaScript kod tipke connect

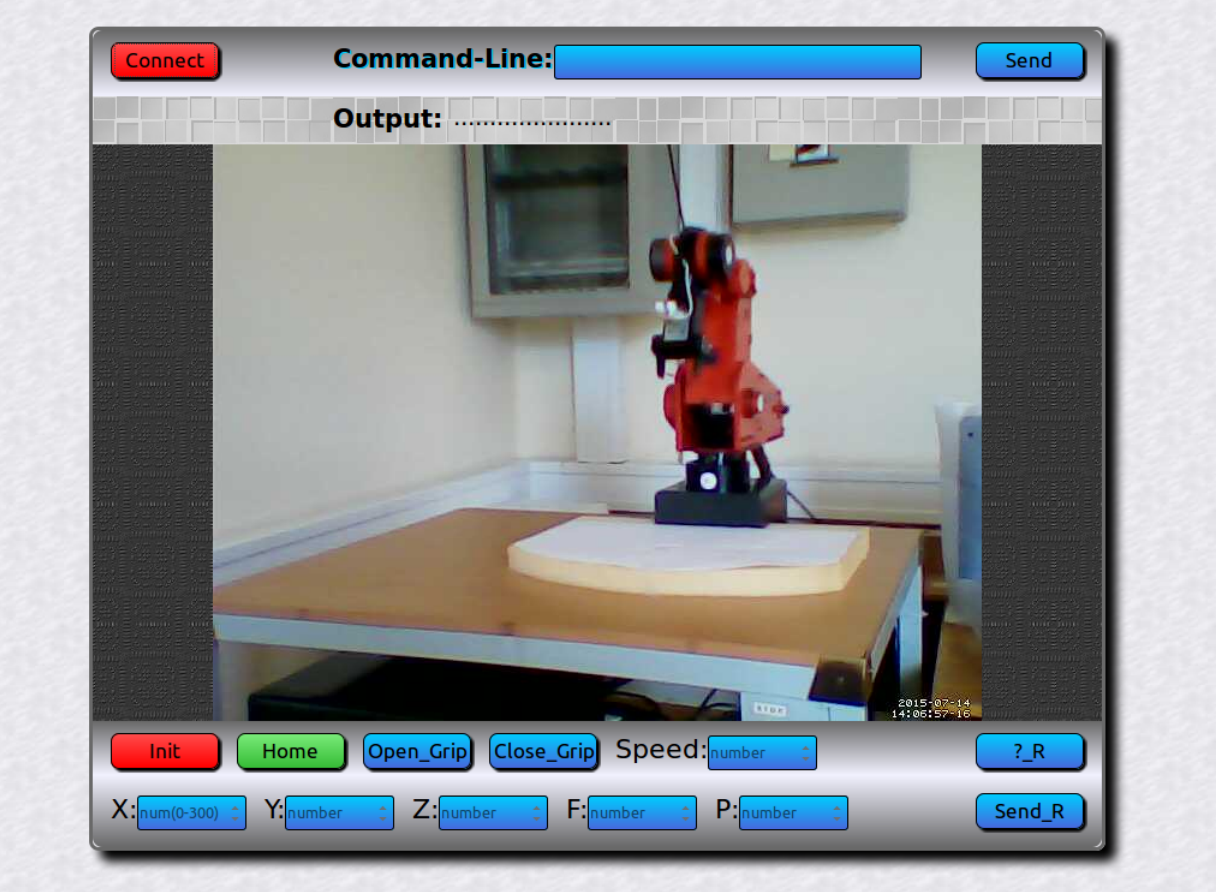
Slanje i primanje podataka preko WebSocket kanala ostvaruje se pomoću ugrađenih metoda na način da se iza variable `connecttows`, koja sada predstavlja otvoreni WebSocket kanal, upiše kod `.send` za slanje ili kod `.onmessage` za primanje podataka. Kod za primanje podataka se nalazi na na linijijama 20 i 21, a funkcionira na način da na dobivenu poruku, iz WebSocket kanala, se pokreće funkcija koja ispisuje podatke u zato predviđenom HTML `<p>` znaku, čiji atribut `id` je referenciran sa `#outCL`. Za razliku od primanja i prikazivanja podataka unutar preglednika, koje se odvija samo na jednom mjestu, slanje podataka prema serverskoj aplikaciji se odvija na više mjesta. Radi sličnosti procesa slanja, navesti će se samo jedan pokazni primjer i to na procesu slanja naredbe za otvaranje prstiju hvataljke.

```
$("#openg").click(function(){
    connecttows.send("OPEN");
});
```

Sl. 6.6: JavaScript kod tipke open

Pritiskom tipke open korisnik će aktivirati JavaScript kod koji šalje naredbu za otvaranje prstiju hvataljke i prikazan je na Sl. 6.6. Tipka open je unutar HTML-a

napravljena sa znakom `<button>`, čiji atribut `id` vrijednosti openg je unutar JavaScript koda referenciran sa `#openg`. Definirana akcija tipke je slanje naredbe `OPEN` pomoću već spomenute `connecttows.send` naredbe.



Sl. 6.7: Sučelje nakon spajanja s robotom

6.3 Python Tornado web server aplikacija

Ova aplikacija vrši primarnu funkciju dostavljanja dviju HTML skripta putem Hypertext Transfer Protocol-a (HTTP) korisničkom pregledniku, i to tako da na početku dostavlja skriptu login.html koji služi autentifikaciji korisnika, a samo nakon uspiješne autentifikacije dostavlja i site.html skriptu koja predstavlja grafičko korisničko sučelje za upravljane robotom. Sama autentifikacija dio je aplikacijske logike i bit će objašnjena u nastavku ovog teksta. Općenita struktura Tornado web aplikacije sastoji se od jedne ili više rukovoditeljske podklase, aplikacijskog objekta koji usmjerava dolazeće zahtjeve od strane korisničkog agenta pripadajućim rukovoditeljskim klasama, te glavne funkcije koja pokreće server. U nastavku je prikazan kod Tornado web server aplikacije

iz kojeg je vidljivo da su nakon uvoza naredbom `import` potrebnih programskih knjižnica definirane dvije klase `mainh` i `check` kao podklase ugrađene `tornado.web.RequestHandler` klase, iza koje sljedi kod glavne funkcije za pokretanje servera, unutar koje je definiran aplikacijski objekt `app` koji nastaje iz zato namjenjene klase `tornado.web.Application`.

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


class mainh(tornado.web.RequestHandler):
    def get(self):
        self.render("login.html")
        print "rendered login.html..."

class check(tornado.web.RequestHandler):
    def get(self):
        self.password = self.get_argument("pass", None)
        if self.password=="pass2":
            self.render("site.html")
            print "renderd site.html..."
        else:
            self.render("login.html")
            print "rendered login.html wrong password... {password}"


if __name__ == '__main__':
    app=tornado.web.Application(handlers = [
        ("/", mainh),
        ("/checkme.html", check),
        (r'/Site/(.*)', tornado.web.StaticFileHandler,
        {'path': "/home/Desktop/Site"}),])
    app.listen(80)
    tornado.ioloop.IOLoop.instance().start()
```

Unutar klase `mainh`, koja se aktivira dolaskom na početnu web adresu, definirana je metoda `def get(self):` nazvana po HTTP metodi kojom rukovodi. Njezin je zadatak da pomocu `self.render` funkcije dobavi resurs login.html korisničkom pregledniku.

Preglednik će iscrtati dobiveni resurs u kodom definirani oblik web stranice, koji u ovom slučaju predstavlja početnu stranicu za autentifikaciju. Zadnja `print` naredba samo služi za lakši pregled toka programa, a ispisuje zadani tekst u terminalu unutar kojeg je pokrenuta Python skripta.

```
class mainh(tornado.web.RequestHandler):
    def get(self):
        self.render("login.html")
        print "rendered login.html..."
```

Unutar druge rukovodeće `check` klase, koja se aktivira slanjem autentifikacijske lozinke preko zato predviđene adrese, nalazi se njena `def get(self):` metoda, čiji je zadatak uvjetno dobavljanje site.html resursa pregledniku. Ona pomoću naredbe `self.password = self.get_argument(‘‘pass’’, None)` dobavlja argument sa imenom ‘‘pass’’, koji pripada HTML `<input>` znaku unutar login.html datoteke. Dobavljeni argument predstavlja lozinku te se pomoću naredba `if` i `else` definiraju dva moguća programska toka provjere lozinke. Program pomoću naredbe `if self.password ==’’xxxxx’’:` provjerava da li dobavljena lozinka, koja se sada nalazi u varijabli `self.password`, odgovara zadanoj lozinki. Ako je jednakost postignuta pomoću naredbe `self.render(‘‘site.html’’)` iscrtava web stranicu, koja predstavlja graficko sucelje za upravljanje robota. Ukoliko jednakost nije ostvarena programski tok dolazi do narebe `else:`, koja pomocu svog `self.render(‘‘login.html’’)` ponovno korisniku iscrtava početnu stranicu za autentifikaciju.

```
class check(tornado.web.RequestHandler):
    def get(self):
        self.password = self.get_argument("pass", None)
        if self.password=="pass2":
            self.render("site.html")
            print "renderd site.html..."
        else:
            self.render("login.html")
            print "rendered login.html wrong password... {password}"
```

Kod na kraju skripte predstavlja glavnu funkciju `main`, koja služi za pokretanje aplikacije. Unutar glavne funkcije definiran je aplikacijski objekt `app`, koji nastaje iz zato predviđene gotove tornadove klase `tornado.web.Application`, a zadužen je za povezivanje adrese koju korisnik izravno ili posredovanjem aplikacije potražuje i zato

odgovarajuće rukovoditeljske klase. Tako je početna adresa stranice označena znakom / povezana sa njenom rukovoditeljskom klasom `mainh`, a pod adresa stranice, `/checkme.html` na koju se korisnik preusmjerava slanjem lozinke, povezana je sa klasom `check`. Treća adresa služi za lociranje svih dodatnih statičnih resursa korištenih u html datotekama (npr. slike korištene u izradi grafičkog sučelja). Zadnje dvije linije koda odnose se na podešavanje aplikacije da sluša zahtjeve na kanalu 80, koji je standardni HTTP kanal, te pokretanje aplikacije u petlju pomocu tornadovog `IOloop` objekta.

```
if __name__ == '__main__':
    app=tornado.web.Application(handlers = [
        ("/", mainh),
        ("/checkme.html", check),
        (r'/Site/(.*)', tornado.web.StaticFileHandler,
        {'path': "/home/Desktop/Site"}),])
    app.listen(80)
```

6.4 Python Tornado WebSocket aplikacija

Ova aplikacija predstavlja implementaciju WebSocket protokola na serverskoj strani koji služi kao kanal za razmijenu informacija između korisnika tj. web preglednika i servera, te implementaciju RFCOMM protokola koji služi koo kanal za razmijenu informacija između serverskog računala i kinematičkog kontrolera robota. Slijedi strukturu općenite Tornado web aplikacije, te je iz njezinog koda vidljivo da nakon uvoza potrebnih programskih knjižnica ima samo jednu `wshandler` klasu koja je napravljena kao podklasa `tornado.websocket.WebSocketHandler` klase i definira ponašanje WebSocket komunikacije na serverskoj strani. Na kraju skripte su vidljivi već u prethodnom podpoglavlju spomenuti aplikacijski objekt `wsapp` koji je ovdje podešen da sluša dolazeće zahtjeve na kanalu 5555, te glavna funkcija koja pokreće websocket server. Rukovodeća `wshandler` klasa upravlja komunikacijom pomoću četri definirane metode. Prva metoda `def check_origin(self, origin):` vraća vrijednost `True` i tako omogućuje Cross-origin resource sharing (CORS) mehanizam koji iz sigurnosnih razloga regulira upotrebu resursa sa vanjskih domena. Ostale tri metode `def open(self):`, `def on_message(self , webmessage):`, `def on_close(self):` definiraju ponašanje kod otvarnja kanala, primanja poruke od preglednika čijim slanjem rukovodi JavaScript WebSocket implementacija, te kod zatvaranja kanala.

```
import tornado.ioloop
import tornado.websocket
from serial import Serial
from time import sleep

class wshandler(tornado.websocket.WebSocketHandler):

    def check_origin(self, origin):
        return True

    def open(self):
        self.connecttoserial=Serial("/dev/rfcomm1", 115200, timeout=1)
        sleep(5)
        print "ws opened..."

    def on_message(self, webmessage):
        print "web message:  ", webmessage
        m = webmessage+"\r"
        self.connecttoserial.writelines(m)
        sleep(2)
        messagerecser = self.connecttoserial.read(2000)
        self.write_message(messagerecser)
        print "received:  ", messagerecser

    def on_close(self):
        self.connecttoserial.flushInput()
        self.connecttoserial.flushOutput()
        self.connecttoserial.close()
        print "connection closed...  "

if __name__ == "__main__":
    appws = tornado.web.Application(handlers=[(r'/', wshandler), ])
    appws.listen(5555)
    tornado.ioloop.IOLoop.instance().start()
```

Nakon što se iz preglednika otvori WebSocket kanal i aktivira se `def open(self):` metoda, pomoću naredbe `self.connecttoserial=Serial("/dev/rfcomm1", 115200 , timeout =1)` se otvara još jedan kanal, koji čini vezu između servera i kinematičkog kontrolera robota. Njihova veza je fizički uspostavljena bluetooth vezom uz upotrebu

RFCOMM protokola, koji simulira serijsku vezu. Uspavljivanje programa na 5 sekundi dodano je da bi se osiguralo uspostavljanje veze prije izmjene poruka, te tako program osiguralo od rušenja zbog slanja podataka na još neotvoreni kanal.

```
def open(self):
    self.connecttoserial=Serial("/dev/rfcomm1", 115200, timeout=1)
    sleep(5)
    print "ws opened..."
```

U metodi `def on_message(self , webmessage):`, koja se aktivira u trenutku kada server primi poruku poslanu iz preglednika, nizom naredbi definirano je da se dobivena poruka uredi za slanje prema kinematičkom kontroleru, pošalje prema robotu, te se nakon izvršenja očita odgovor robota koji se potom šalje natrag prema pregledniku.

```
def on_message(self, webmessage):
    print "web message:  ", webmessage
    m = webmessage+"\r"
    self.connecttoserial.writelines(m)
    sleep(2)
    messagerecser = self.connecttoserial.read(2000)
    self.write_message(messagerecser)
    print "received:  ", messagerecser
```

U posljednoj `def on_close(self):` metodi koja se aktivira zatvaranjem WebSocket kanala definirano je čišćenje uloznih i izlaznih međuspremnika, te zatavranje bluetooth kanala.

```
def on_close(self):
    self.connecttoserial.flushInput()
    self.connecttoserial.flushOutput()
    self.connecttoserial.close()
    print "connection closed...  "
```

6.5 Motion webcam serverka aplikacija

Motion posjeduje jednostavni webcam server koji daje živu sliku u mjpeg formatu. U tu svrhu prije pokretanja Motiona potrebno je izvršiti podešavanje njegove konfiguracijske tekstualne datoteke za sve relevantne parametre. Za ovu aplikaciju podešen je da dostavlja živu sliku na kanal 8081 sa kojeg se ona preuzima, a što je pokazano u poglavlju 6.2. U

Tab. 6.1 dan je prikaz relevantnih parametara sa njihovim objašnjenjem, te početnih i odbranih vrijednosti.

Parametar	**Moguće vrijednosti** **Početna vrijednost** **Postavljena vrijednost**	**Opis**
videodevice	Max 4095 znakova /dev/video0 /dev/video1	Odabir kamere koja će biti korištena.
framerate	2 - 100 100 (neograničeno) 15	Max. broj kamerom snimljenih slika u jednoj sekundi.
width	Ovisi o kameri 352 640	Širina slike u pikselima. Dozvoljen raspon ovisi o kameri.
height	Ovisi o kameri 288 480	Visina slike u pikselima. Dozvoljen raspon ovisi o kameri.
webcam_port	0 - 65535 0 (onemogućeno) 8081	TCP kanal na kojem će Motion osluškivat zahtjeve za spajanje na njegov webcam server
webcam_quality	1 - 100 50 50	Kvaliteta isporučenih mjpeg slika zadana postotkom.
webcam_maxrate	1 - 100 1 15	Ograničavanje slanja kamerom snimljenih slika u jednoj sekundi. Vrijednost 100 predstavlja neograničeno.
webcam_limit	0 - 2147483647 0 (neograničeno) 0	Količina dozvoljenih slika koja se može pogledati prije nego Motion odspoji korisnika.
webcam_localhost	on, off on off	Ograničavanje pristupa web kameri na lokalnog korisnika.

Tab. 6.1: Parametri Motion aplikacije

7 Zaključak

U ovom radu opisan je jedan od mnogobrojnih načina izrade web aplikacije koja bi služila upravljanju robotske ruke uz video nadzor. Aplikacija je testirana lokalno u pregledniku serverskog računala koje je bilo spojeno sa kinematičkim kontrolerom robota preko bluetooth veze. Za njenu realizaciju kao globalno dostupne aplikacije bilo bi potrebno omogućiti slobodan ulaz i izlaz na mrežu preko kanala (engl. port) usmjernika (engl. router). Njegov je zadatak da, ako mu je to dopušteno, prosljedi sav promet koji ide preko nekog kanala prema serverskom računalu. Otvoreni kanal usmjernika potencijalno predstavlja sigurnosni rizik za cijelu mrežu. Zbog sigurnosti i vremena potrebnog za dobivanje dozvole za otvaranje kanala usmjernika na jednoj kompleksnijoj mreži institucije kao što je fakultet, ovaj rad nije u potpunosti testiran. Na privatnoj mreži tj. na kućnom usmjerniku testirana je dostupnost aplikacije globalno, ali tada je ona služila samo za prijenos žive slike jer serversko računalo nije bilo u mogućnosti biti spojeno sa udaljenim robotom.

Zbog današnje sveprisutnosti interneta i praktičnosti upotrebe web preglednika kao korisničkog sučelja web aplikacije predstavljaju rastući tržišni trend programske podrške. Njihovom upotrebom se omogućava nadzor i upravljanje različitim procesima na daljinu. Web aplikacije su praktične zbog lakoće nadopunjavanja i održavanja, bez potrebe distribuiranja i instaliranja programske podrške na računala pojedinačnih, potecijalno mnogobrojnih korisnika. Za njihovo upravljanje može poslužiti svako računalo sa ažuriranim web preglednikom, neovisno o operativnom sustavu. Osim mnogobrojnih mogućnosti koje pružaju bitno je spomenuti da njihova jednostavna dostupnost na globalnoj internetskoj mreži ujedno i predstavlja njihovu negativnu stranu sa aspekta sigurnosti. Osim pitanja sigurnosti za koje je potrebno uložiti dodatne napore kod izrade aplikacije, postoje još i potencijalni problemi kašnjenja i stabilnosti kojima se je potrebno posvetiti koliko priroda procesa kojim se upravlja to zahtjeva.

Literatura

[1] https://bib.irb.hr/datoteka/578851.Final_Crnekovic_Zorc-Kinematic_controller.pdf

[2] Tugomir Šurina, Mladen Crneković: "Industrijski roboti". Sveučilište u Zagrebu, Školska knjiga, 1990.

[3] https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#Radio_frequency_communication_.28RFCOMM.29

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

[5] http://tools.ietf.org/html/rfc6455

[6] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[7] http://tools.ietf.org/html/rfc2616

[8] http://www.webopedia.com/DidYouKnow/Internet/HowWebServersWork.asp

[9] https://www.python.org/doc/

[10] http://www.tornadoweb.org/en/stable/

[11] http://www.lavrsen.dk/foswiki/bin/view/Motion/WebcamServer

[12] http://www.webopedia.com/TERM/H/HTML.html

[13] http://www.webopedia.com/TERM/C/CSS.html

[14] http://www.webopedia.com/TERM/J/jQuery.html

[15] http://www.webopedia.com/TERM/J/JavaScript.html

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

[17] http://tornado.readthedocs.org/en/latest/websocket.html

Prilozi

CD-ROM.

Skripta login.html

```
<html>
<head>
    <style>

        body {
            font-family: "Trebuchet MS", Verdana, sans-serif;
            background-image: url("/Site/pap3.jpg");
            color: #000000;
        }

        .row {
            position: relative;
            margin-left: auto;
            margin-right: auto;
            width: 600px;
            height: 100px;
            top:  250px;

            background-image: url("/Site/pap4.jpg");
            border-radius: 8px 8px 8px 8px;
            border-style: solid;
            border-width: 2px;
            border-color: #000000;
            box-shadow: 8px 8px 7px #000000;
        }

        .inputlogin {

            padding-left: 10px;
            padding-right: 10px;

            color: #000000;
            font-size: 17px;

            background: #00CCFF;
            background-image: linear-gradient(to bottom, #00CCFF, #4769DE);
            border-radius: 8px 8px 8px 8px;
            border-style: solid;
            border-width: 1px;
            border-color: #000000;
```

```
        }

        #inpass {
            position: absolute;
            top: 55px;
            left: 70px;
            width: 55%;
            height: 30px;
        }

        #inbtn {
            position: absolute;
            top: 55px;
            right: 15px;
            width: 100px;
            height: 30px;
            border-width: 2px;
            border-radius: 7px 7px 7px 7px;
        }
        #inbtn:hover {
            opacity: 0.8;
        }



        #text2 {
            position: absolute;
            left: 150px;
            bottom: 20px;
            text-align: center;
            font-size: 35px;
            color: #000000;
            opacity: 0.9;
        }

        .imglogo {
            position: absolute;
            left: -55px;
            bottom: 65px;
            vertical-align: baseline;
            height: 70%;
            box-shadow: 7px 7px 6px #000000;
        }
        .imglogo:hover {
            opacity: 0.2;
        }

    </style>
</head>
```

```
<body>

    <div class="row" id="row1">

        <p id="text2"><b>Log In:</b></p>

        <img src="/Site/fsblogo2.png" class="imglogo">

        <form action="checkme.html" method="get">
            <input type="text" name="pass"
class="inputlogin" id="inpass" placeholder="    password">
            <input type="submit" value="Send" class="inputlogin" id="inbtn">
        </form>

    </div>

</body>
</html>
```

Skripta site.html

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>RM501_web_app</title>

    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>

    <script type="text/javascript" >

        $(document).ready(function(){

            $("#connect").click(function(){
                //ipws="ws://192.168.0.10:5555"
                //ipimg="http://192.168.0.10:8081"
                ipws="ws://127.0.0.1:5555"
                ipimg="http://127.0.0.1:8081"
                connecttows=new WebSocket(ipws);
                $("#img").attr("src", ipimg);
                connecttows.onmessage = function(event){
                        $("#outCL").text(event.data);
                };
            });

            $("#inputCL").keypress(function (e) {
                if (e.which == 13) {
                    connecttows.send($("#inputCL").val());
                }
```

```
});
$("#inputX").keypress(function (e) {
    if (e.which == 13) {
        connecttows.send("x"+$("#inputX").val());
    }
});
$("#inputY").keypress(function (e) {
    if (e.which == 13) {
        connecttows.send("y"+$("#inputY").val());
    }
});
$("#inputZ").keypress(function (e) {
    if (e.which == 13) {
        connecttows.send("z"+$("#inputZ").val());
    }
});
$("#inputF").keypress(function (e) {
    if (e.which == 13) {
        connecttows.send("f"+$("#inputF").val());
    }
});
$("#inputP").keypress(function (e) {
    if (e.which == 13) {
        connecttows.send("p"+$("#inputP").val());
    }
});
$("#inputS").keypress(function (e) {
    if (e.which == 13) {
        connecttows.send("SPEED="+$("#inputS").val());
    }
});

$("#wssend").click(function(){
    connecttows.send($("#inputCL").val());
});
$("#openg").click(function(){
    connecttows.send("OPEN");
});
$("#closeg").click(function(){
    connecttows.send("CLOSE");
});
$("#home").click(function(){
    connecttows.send($("HOME");
});
$("#init").click(function(){
    connecttows.send($("INIT");
});
$("#whereR").click(function("?R"){
    connecttows.send();
```

```
        });
        $("#wssend2").click(function(){
            var x=$("#inputX").val();
            var y=$("#inputY").val();
            var z=$("#inputZ").val();
            var f=$("#inputF").val();
            var p=$("#inputP").val();
            var sum="x"+x+"y"+y+"z"+z+"f"+f+"p"+p;
            connecttows.send(sum);
        });
    });
</script>

<style>

    body {
        font-family: "Trebuchet MS", Verdana, sans-serif;
        background: #FFFFFF;
        background-image: url("/Site/pap3.jpg");
        font-size: 22px;
    }

    .fix {
        position: relative;
        margin-left: auto;
        margin-right: auto;
        width: 840px;
        height: 680px;
        top:  20px;

        border-top-left-radius: 8px;
        border-top-right-radius: 8px;
        border-bottom-left-radius: 8px;
        border-bottom-right-radius: 8px;
        border-style: solid;
        border-top-width: 3px;
        border-bottom-width: 3px;
        border-left-width: 3px;
        border-right-width: 3px;
        border-color: #696969;
        box-shadow: 9px 9px 8px #000000;
    }

    .row {
        position: relative;
        margin-left: auto;
        margin-right: auto;
        width: 840px;
    }
```

```
#row1 {
    height: 55px;
    border-top-left-radius: 8px;
    border-top-right-radius: 8px;

    background: #F2F2FF;
    background-image: linear-gradient(to top, #F2F2FF, #696969);
}
#row2 {
    height: 40px;
    background-image: url("/Site/pap4.jpg");
}
#row3 {
    height: 480px;
    background-image: url("/Site/gr1.jpg");
}
#row35 {
    height: 45px;
    background: #F2F2FF;
    background-image: linear-gradient(to top, #F2F2FF, #696969);
}
#row4 {
    height: 60px;
    border-bottom-left-radius: 8px;
    border-bottom-right-radius: 8px;

    background: #F2F2FF;
    background-image: linear-gradient(to bottom, #F2F2FF, #696969);
}
```

```
.inline {
    width: 840px;
    height: 55px;
    padding-top: 10px;
}
.inline2{
    padding-top: 10px;
}
.inline35{
    padding-top: 10px;
}
.inline4 {
    width: 840px;
    height: 60px;
```

```
    padding-top: 15px;
}

.boxl {
    float: left;
    padding-left: 15px;
}
.boxr {
    float: right;
    padding-right: 15px;
}
.boxa {
    position: absolute;
    left: 200px;
    top: -10px;
}
.boxa2 {
    position: absolute;
    left: 200px;
    top: -15px;
}
.boxa3 {
    position: absolute;
    left: 300px;
    top: -15px;
}


.input {
    width: 85px;
    height: 25px;
    color: #000000;
    background: #00CCFF;
    background-image: linear-gradient(to bottom, #00CCFF, #4769DE);
    border-radius: 3px 3px 3px 3px;
    border-style: solid;
    border-width: 1px;
    border-color: #000000;
    padding-left: 2px;
    padding-right: 2px;
}

.btn {
    width: 90px;
    height: 30px;
    color: #000000;
    font-size: 17px;
    background: #00CCFF;
    background-image: linear-gradient(to bottom, #00CCFF, #4769DE);
```

```
        border-radius: 7px 7px 7px 7px;
        border-style: solid;
        border-width: 1px;
        border-color: #000000;
        padding-left: 2px;
        padding-right: 2px;
        box-shadow: 2px 2px 1px #000000;
    }
    .btn:hover {
        opacity: 0.8;
    }
    #connect {
        background-image: linear-gradient(to bottom, #FF4C4C, #FF0000);
    }
    #init {
        background-image: linear-gradient(to bottom, #FF4C4C, #FF0000);
    }
    #home {
        background-image: linear-gradient(to bottom, #7AEB7A, #36BD36);
    }

    #textCL {
        text-shadow: 1px 1px #00CCFF;
        font-size: 21px;
        color: #000000;
    }
    #inputCL {
        width: 300px;
    }
    #inputCLdiv{
        padding-right: 45px;
    }

    #text2{
        font-size: 21px;
        color: #000000;
    }
    #outCL{


        font-size: 20px;
        color: #000000;
    }




    .containerimg {
```

```
            margin-left: auto;
            margin-right: auto;
            width: 640px;
        }

    </style>
</head>

<body>
<div class="fix" >

    <div id="row1" class="row">
        <div class="inline">
            <div id="div11" class="boxl">
                <button id="connect" class="btn" type="submit">Connect</button>
            </div>
            <div id="div12" class="boxr" >
                <button id="wssend" class="btn" type="submit">Send</button>
            </div>
            <div class="boxa">
                <p id="textCL"><b>Command-Line:</b></p>
            </div>
            <div id="inputCLdiv" class="boxr">
                <input id="inputCL" class="input" type="text">
            </div>
        </div>
    </div>

    <div id="row2" class="row">
        <div class="inline2">
            <div id="div21" class="containerout ">
                <div class="boxa2">
                <p id="text2"><b>Output:</b></p>
                </div>
                <div class="boxa3">
                <p id="outCL">.....</p>
                </div>
            </div>
        </div>
    </div>

    <div id="row3" class="row">
        <div class="containerimg">
            <div id="div31" class="box">
                <img  id="img"/>
            </div>
        </div>
    </div>
```

```
    <div id="row35" class="row">
        <div class="inline35">
            <div id="div351" class="boxl">
                <button id="init" class="btn" type="submit">Init</button>
            </div>
            <div id="div352" class="boxl">
                <button id="home" class="btn" type="submit">Home</button>
            </div>
            <div id="div354" class="boxl" >
                <button id="openg" class="btn" type="submit">Open_Grip</button>
            </div>
            <div id="div355" class="boxl" >
                <button id="closeg" class="btn" type="submit">Close_Grip</button>
            </div>
            <div id="div353" class="boxl">
                Speed:<input id="inputS" class="input" type="number">
            </div>
            <div id="div356" class="boxr" >
                <button id="whereR" class="btn" type="submit">?_R</button>
            </div>
        </div>
    </div>

    <div id="row4" class="row">
        <div class="inline4">
            <div id="div41" class="boxl">
                X:<input id="inputX" class="input" type="number">
            </div>
            <div id="div42" class="boxl">
                Y:<input id="inputY" class="input" type="number">
            </div>
            <div id="div43" class="boxl">
                Z:<input id="inputZ" class="input" type="number">
            </div>
            <div id="div44" class="boxl">
                F:<input id="inputF" class="input" type="number">
            </div>
            <div id="div45" class="boxl">
                P:<input id="inputP" class="input" type="number">
            </div>
            <div id="div46" class="boxr">
                <button id="wssend2" class="btn" type="submit">Send_R</button>
            </div>
        </div>
    </div>

</div>
</body>
</html>
```

Skripta MainA.py

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


class mainh(tornado.web.RequestHandler):
    def get(self):
        self.render("login.html")
        print "rendered login.html..."

class check(tornado.web.RequestHandler):
    def get(self):
        self.password = self.get_argument("pass", None)
        if self.password=="pass2":
            self.render("site.html")
            print "renderd site.html..."
        else:
            self.render("login.html")
            print "rendered login.html wrong password... {password}"


if __name__ == '__main__':
    app=tornado.web.Application(handlers = [
        ("/", mainh),
        ("/checkme.html", check),
        (r'/Site/(.*)', tornado.web.StaticFileHandler,
        {'path': "/home/Desktop/Site"}),])
    app.listen(80)
    tornado.ioloop.IOLoop.instance().start()
```

Skripta wshandler.py

```
import tornado.ioloop
import tornado.websocket
from serial import Serial
from time import sleep
```

```
class wshandler(tornado.websocket.WebSocketHandler):

    def check_origin(self, origin):
        return True

    def open(self):
        self.connecttoserial=Serial("/dev/rfcomm1", 115200, timeout=1)
        sleep(5)
        print "ws opened..."

    def on_message(self, webmessage):
        print "web message:  ", webmessage
        m = webmessage+"\r"
        self.connecttoserial.writelines(m)
        sleep(2)
        messagerecser = self.connecttoserial.read(2000)
        self.write_message(messagerecser)
        print "received:  ", messagerecser

    def on_close(self):
        self.connecttoserial.flushInput()
        self.connecttoserial.flushOutput()
        self.connecttoserial.close()
        print "connection closed...  "

if __name__ == "__main__":
    appws = tornado.web.Application(handlers=[(r'/', wshandler), ])
    appws.listen(5555)
    tornado.ioloop.IOLoop.instance().start()
```