SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4506

Izvedba algoritma za mjerenje brzine i smjera vjetra

Bernard Radman

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 18. ožujka 2016.

ZAVRŠNI ZADATAK br. 4506

Pristupnik: **Bernard Radman (0036473424)**
Studij: Računarstvo
Modul: Računalno inženjerstvo

Zadatak: **Izvedba algoritma za mjerenje brzine i smjera vjetra**

Opis zadatka:

U okviru završnog rada potrebno je realizirati algoritam za mjerenje brzine i smjera vjetra koji je temeljen na mjerenju vremena propagacije ultrazvučnog višeharmonijskog signala. Sustav je temeljen na četiri ultrazvučna pretvarača koji su raspoređeni na uglovima kvadrata pogodne dimenzije čije su dijagonale (okomite osi) orijentirane prema glavnim geografskim osima (sjever, jug istok i zapad). Mjerenja se provodi u dvije okomite osi (S-N, E-W) i četiri dijagonalna puta (N-W, W-S, S-E i E-N) i to u oba smjera širenja mjernog signala. Ulaz u algoritam su izmjerene vrijednosti kašnjenja na četiri odabrane staze (po dvije okomite osi i po dva dijagonalna puta) a konkretne staze su ovisne o odabiru odašiljačkih i prijemničkih pretvarača za osam mogućih kombinacija konfiguracije mjerenja. Iznosi kašnjenja se učitavaju u sustav serijskom USB sučељem iz mjernog uređaja, a potrebno je razviti programsku potporu za prihvat ovih ulaznih podataka u stvarnom vremenu. Izračun brzine i smjera vjetra treba implementirati u skladu s referentnim programskim modelom u Matlabu korištenjem 2, 4 ili 8 mjerenja na različitim mjernim stazama uz robusnu estimaciju izmjerene vrijednosti koja mora biti neosjetljiva na izlučnike. Analizirati mogućnost prilagodbe algoritma za cjelobrojnu izvedbu te diskutirati složenost i točnost izvedbe u odnosu na referentni model. Program razviti u jeziku C na ugradbenoj platformi RaspberryPI pod Linux operacijskim sustavom. Za dodatne informacije obratiti se mentoru.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.

Mentor:
Prof. dr. sc. Davor Petrinović

Djelovođa:
Prof. dr. sc. Danko Basch

Predsjednik odbora za
završni rad modula:
Prof. dr. sc. Mario Zagar

SADRŽAJ

1 Uvod

U današnje vrijeme nemoguće je zamisliti svijet u kojem ne možemo odrediti brzinu i smjer vjetra. Poznavanje karakteristika te jednostavne prirodne pojave od velikog je značenja za mnoga područja, od izgradnje visokih građevina, letu aviona pa sve do poznavanja cestovnim prilika. Zanimanje za izračun brzine vjetra pojavilo se i prije 15. stoljeću kada su zabilježene najranije izgradnje anemometra, ali sve do tada čovječanstvo nije imalo načina za precizniji izračun. Klasični mehanički anemometar se od svog postanka do danas jako malo promijenio. Veliki iskorak u suvremenoj meteorologiji dogodio se 1994. godine kada je Andrews Pflitsch izumio zvučni anemometar. Zvučni anemometar radi na principu izračuna vremena propagacije ultrazvučnog signala koji se šalje između dva pretvarača. [5] Slično tomu, u okviru ovog završnog rada potrebno je realizirati algoritam za mjerenje brzine i smjera vjetra temeljen na mjerenju vremena propagacije ultrazvučnog više-harmonijskog signala. Ulaz u algoritam su izmjerene vrijednosti kašnjenja koje je potrebno učitati s mjernog uređaja. Kao platforma za razvoj algoritma odabrano je Raspberry Pi računalo, a ulazne vrijednosti primaju se preko serijskog USB sučelja. Raspberry Pi je odabran jer je i osmišljen upravo u svrhe edukacije i približavanja svijeta tehnologije mlađim uzrastima. Sa svojim skromnim dimenzijama i cijenom, Raspberry Pi je pravo čudo tehnike koje oduševljava korisnike širom svijeta.

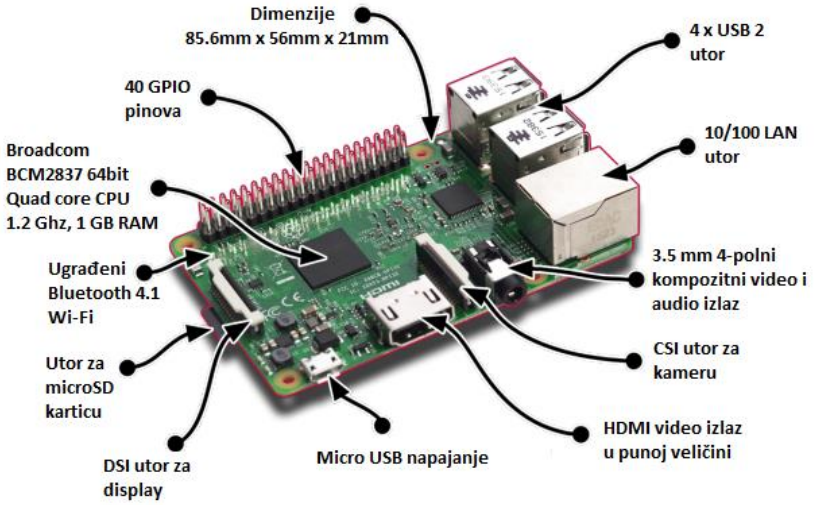
1.1 Povijest Raspberry Pi-a

Raspberry Pi nastao je 2006. godine kad se javila ideja o malom i pristupačnom računalu dostupnom svima. Uvidjevši koliko malo praktičnog znanja imaju studenti koji se prijavljuju na računarske predmete, Eben Upton, uz pomoć svojih kolega Robba Mullinsa, Jacka Langa i Alana Mycrofta, sa Sveučilišta u Cambridgeu, odlučio je napraviti računalo koje bi bilo pristupačno cijenom te ujedno pružalo odličnu okolinu za pisanje programa. Inspiriran Acornovim BBC Micro iz 1981., nastao je tako Raspberry Pi, računalo s naglaskom na edukaciju. Narednih nekoliko godina uvelike se radilo na razvoju, ali se velika prekretnica dogodila pojavom procesora za mobilne uređaje koji su imali dovoljno snage za grafičku obradu a ujedno su bili pristupačni cijenom. Razvijena su dva modela, Raspberry Pi model A i model B. Model B koštao je 35 USD (američkih dolara) i na tržištu se pojavio u travnju 2012. godine. Jeftiniji model A, od svega 25 USD, pojavio se gotovo godinu dana kasnije, u veljači 2013. godine. Već samom pojavom na tržištu, Raspberry Pi privukao je veliku pažnju. Malih dimenzija, ali velikog potencijala i mogućnosti, privukao je poglede ne samo programera, nego i brojnih entuzijasta koji su željeli iskušati mogućnosti tog novog uređaja. Ubrzo je uslijedila velika društvena podrška, te je trenutno na internetu moguće pronaći mnogo uputa i praktičnih izvedbi za razne primjene, od izrada robota, retro i arkadnih igraćih platformi kao na slici 1.1, pa sve do mjernih uređaja sposobnih za rad čak na međunarodnoj svemirskoj postaji. Raspberry Pi unio je revoluciju u svijet računala. [2]

Slika 1.1 Raspberry Pi kao arkadna igra [12]

1.2 Hardware

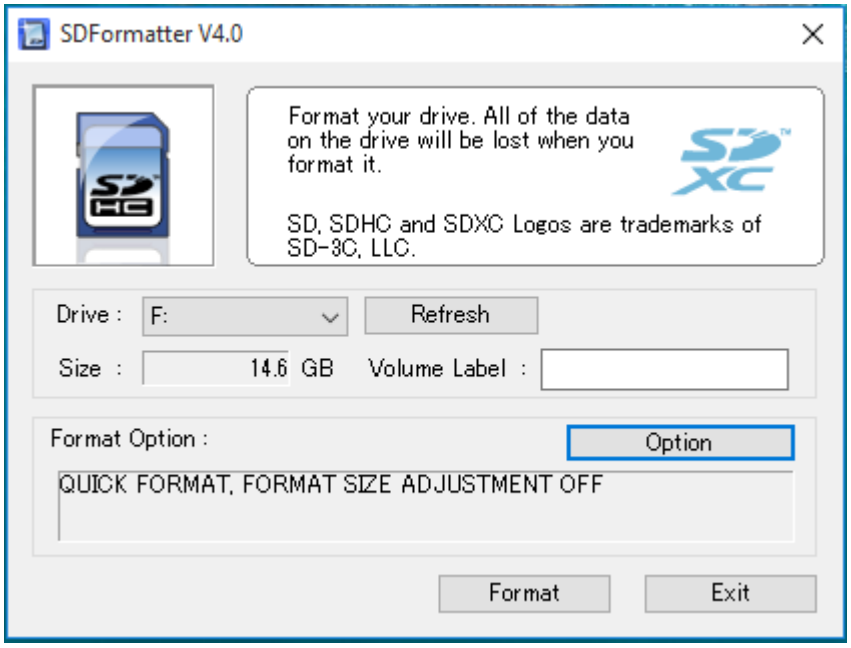
Raspberry Pi 3 model B, najnoviji je model te je na tržište izašao u veljači 2016. godine, s početnom cijenom od 35 USD. Taj model bit će korišten i u ovom radu. Kao i njegovi predhodnici, sadrži Broadcom integrirani sustav na čipu, verziju BCM2837 s arhitekturom ARMv8, 64 ili 32-bitnom. Procesor je ARM Cortex-A53 koji sa svoje četiri jezgre radi na taktu od 1,2 GHz-a, a uz 1 GB dijeljene radne memorije s grafičkim procesorom Broadcom VideoCore IV, s frekvencijom od 250 MHz, nudi široki spektar mogućnosti za nisku cijenu. Kao sve dosadašnje verzije, RPi (Raspberry Pi) 3 također ne sadrži internu memoriju za pohranu, već koristi MicroSDHC karticu za spremanje podataka i operacijskog sustava. Takav način podizanja sustav možda se na prvu ruku čini nezgodan, ali korisnicima omogućava da uz laku zamjenu kartica isti RPi koriste za više funkcija bez dodatnih napora poput formatiranja i ponovnog instaliranja sustava. Na službenim stranicama Raspberry Pi Foundation-a mogu se pronaći gotove distribucije operacijskih sustava koji se brzo i jednostavno instaliraju na memorijsku kraticu te omogućuju korištenje RPi-a u svega nekoliko minuta. Jedan od popularnijih operacijskih sustava, koji je korišten i u ovom radu, je Raspbian, Linux distribucija temeljena na popularnom Debian Jessie-u. RPi 3 ima 4 USB utora na koje se mogu spojiti generičke tipkovnice i miševi, te HDMI izlaz za spajanje monitora ili televizora. Za mrežnu komunikaciju, tu su ugrađeni 10/100 Mbit/s Ethernet konektor, te 802.11n bežična kartica i Bluetooth 4.1. Također, na pločici postoji i 40 GPIO pinova koje korisnik može definirati po želji i potrebi. Cijeli RPi napaja se s 5 V preko MicroUSB konektora. [2]



Slika 1.2 Glavni dijelovi Raspberry Pi 3 modela B (prilagođeno prema [13])

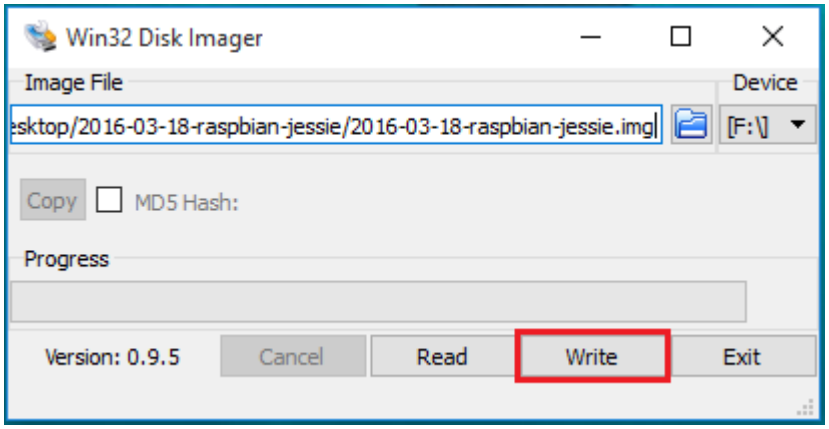
2 Instalacija operacijskog sustava na Raspberry Pi i povezivanje s laptopom

Raspberry Pi ne dolazi sa internom memorijom, stoga sam nakon preuzimanja RPi-a kod mentora, pribavio memorijsku MicroSDHC karticu i na nju postavio željeni operacijski sustav. Odlučio sam se za karticu od 16 GB, što je dovoljno memorijskog prostora, čak više nego što je potrebno za izvedbu ovog rad. Prije rada s Raspberry-em, potrebno je instalirati odabrani operacijski sustav, za to je potrebno drugo računalo s utorom ili adapterom za memorijsku karticu. Sa službene stranice RPi-a [6] sam skinuo *image* operacijskog sustava Raspbian. Memorijsku karticu potrebno je fomatirati prije postavljanja *image*-a. Koristio sam aplikaciju *SDFormater* koju je moguće pronaći na stranici [7], ali mogu se koristiti i drugi sličani proizvodi.



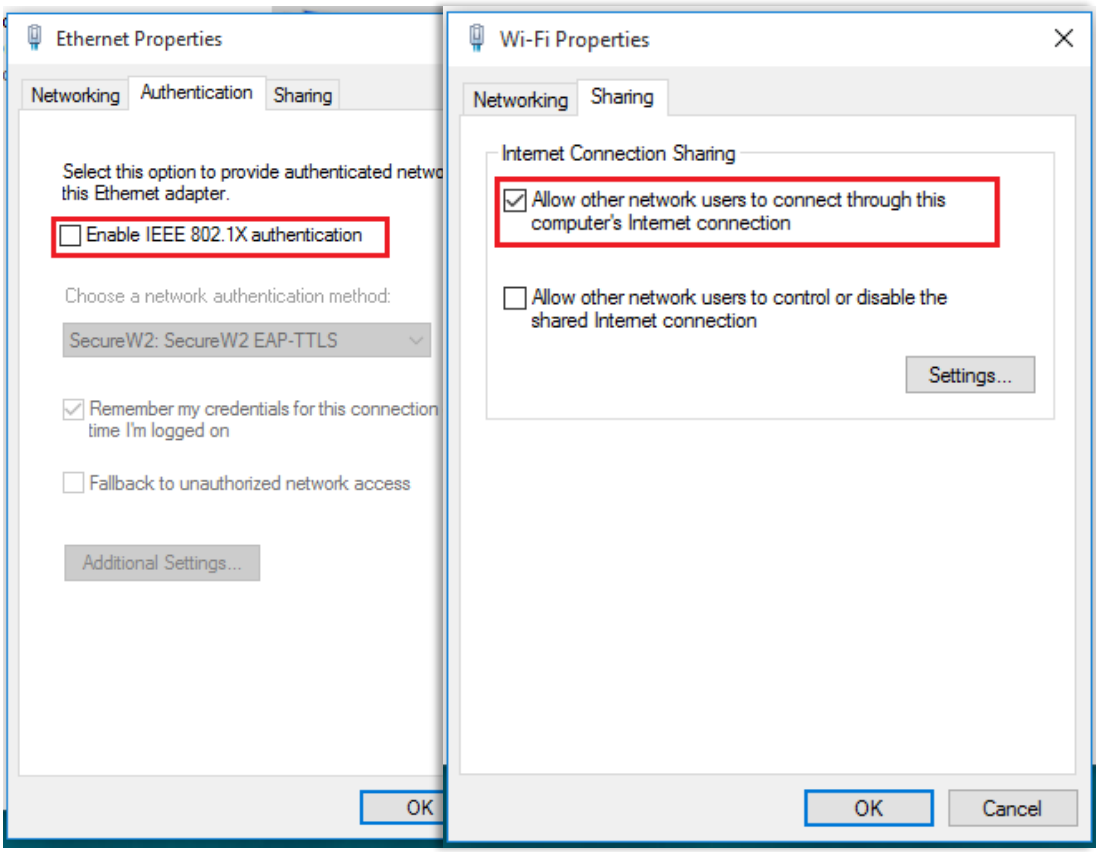
Slika 2.1 Formatiranje MicroSDHC kartice

Nakon uspješnog formatiranja memorijske karticu, uslijedilo je da na nju snimim sliku operacijskog sustava. Za to sam koristio aplikaciju Win32DiskImager koju sam preuzeo sa stranice [8]. Korištenje navedenog alata potpuno je intuitivno i lagano, potrebno je samo odabrati put do odgovorajuće slike i pritisnuti „*Write*".



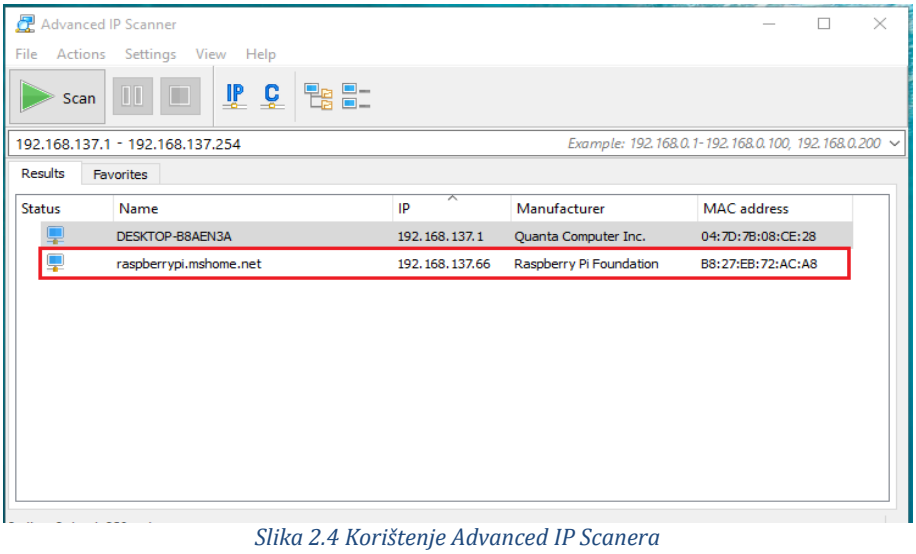
Slika 2.2 Pisanje operacijskog sustava na MicroSDHC karticu

Sada je memorijska kartica spremna za rad. Dovoljno bi bilo istu staviti u odgovarajući utora na Raspberry Pi-ju, prikopčati USB tipkovnicu i miša, spojiti HDMI monitor i priključiti napajanje. U mom slučaju to nije bilo tako jednostavno. Naime, stanujem u studentskom domu gdje posjedujem laptop, te osim USB miša, nisam imao drugu potrebnu periferiju za rad s RPi-em. Stoga sam morao pronaći alternativu. Pretraživanjem na internetu pronašao sam nekoliko uputa za spajanje Raspberry Pi-ja s laptopom uz pomoć Ethernet kabla i SSH protokola. Naime, moguće je Raspberry-u pristupati pomoću „udaljenog pristupa“, iako je od laptopa udaljen svega desetak centimetara. Za spajanje je potreban Ethernet kabel, PuTTY alat za uspostavu prve konekcije [9], te aplikacija za uspostavu udaljenog pristupa, ovdje korištena je apikacija Real VNC [10]. Osim toga, koristio sam aplikaciju Advanced IP Scaner [11]. Otežavajuća okolnost bila je što sam u studentskom domu,te svaki put kada želim pristupiti Raspberry-ju, morao sam mijenjati mrežne postavke. Potrebno je isključiti IEEE 802.1X autentifikaciju u Ethernet postavkama, te omogućiti dijeljenje mreže u Wi-Fi postavkama kako bi se dinamički dodijelila IP adresa Raspberry Pi-ju.



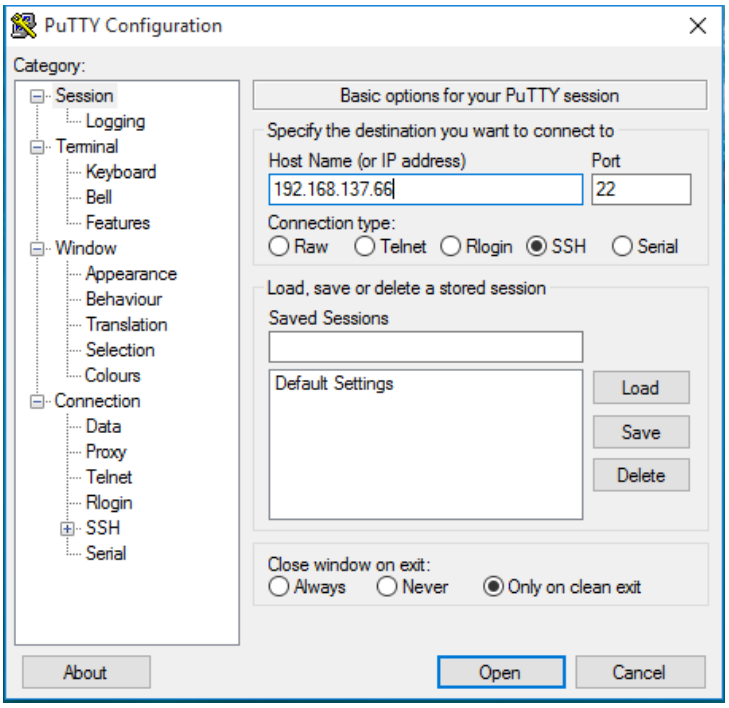
Slika 2.3 Namještanje mrežnih postavki

Nakon što sam umetnuo memorijsku karticu u RPi, spojio ga s laptopom pomoću Ethernet kabla i priključio na napajanje, potrebo je pronaći koju je IP adresu moj laptop dodjelio Raspberry-u. Koristio sam aplikaciju Advanced IP Scaner u kojoj se klikom na gumb *Scan* izlistaju sve dodijeljene IP adrese lokalne mreže s pripadajućim imenom. Potrebno je pronaći onu koja u imenu sadrži *raspberrypi.mshome.net*.

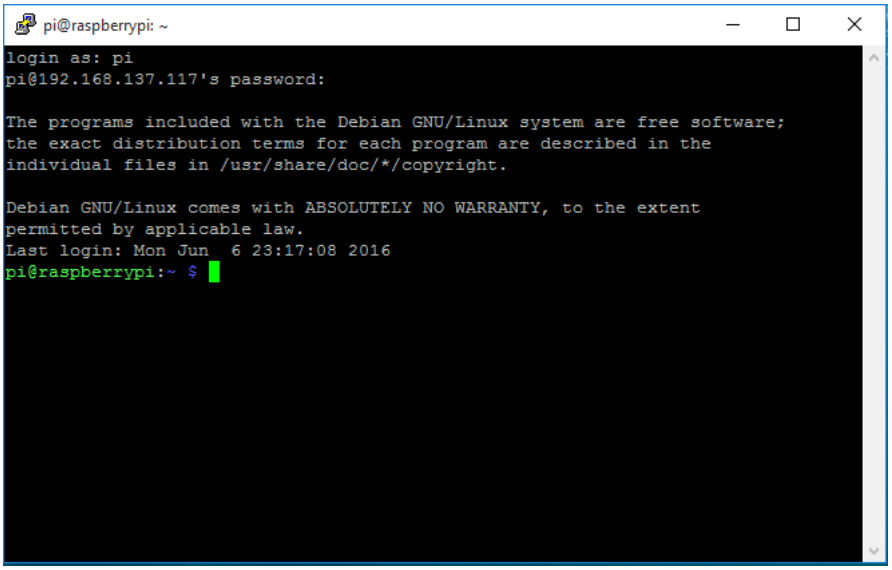


Slika 2.4 Korištenje Advanced IP Scanera

Nakon pronalaska IP adrese, mogao sam pristupiti komandnoj liniji Raspberry Pi-ja pomoću alata PuTTY. Potrebno je samo upisati IP adresu i odabrati SSH za vrstu konekcije te kliknuti na gumb za otvaranje veze. Ako su sve postavke uredno napravljene, otvara se terminal, a početno korisničko ime i lozinka za pristup su zadane, *pi* i *raspberry* slijedno.



Slika 2.5 Korištenje PuTTY-a



Slika 2.6 Terminal iz PuTTY-a

Koristio sam PuTTY kako bih kroz nekoliko naredbi u komandnoj liniji na Raspberry Pi instalirao VNC server na kojeg sam se kasnije spajao s laptopa. Potrebno je upisati sljedeće naredbe:

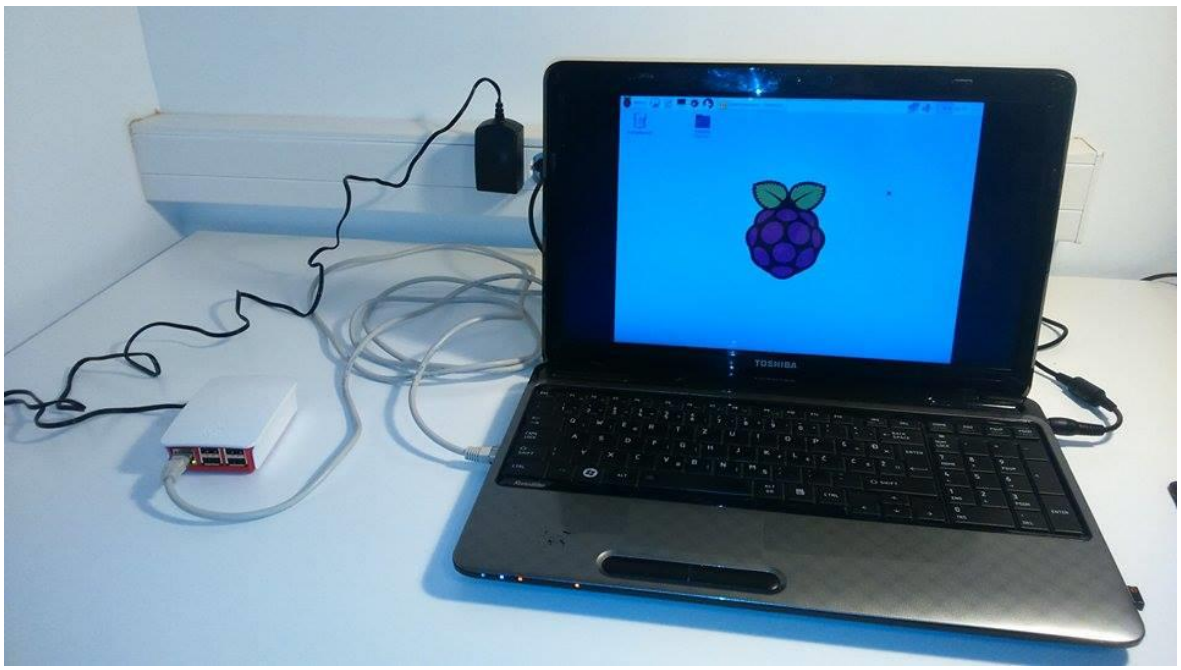
sudo apt-get update

sudo apt-get install tightvncserver

Za izvršavanje prethodnih naredbi, potrebno je da Rasberry Pi bude spojen na internet, a to je moguće ako je laptop spojen na postojeću Wi-Fi mrežu jer je dijeljenje mreže omogućeno u postavkama. Ako ne postoji Wi-Fi mreža, te se laptop spaja na internet pomoću Ethernet kabla, moguće je upaliti žarišnu točku na mobitelu te se zatim s laptopa spojiti na tu mrežu. Za pokretanje VNC-a potrebno je u terminal upisati naredbu:

vncserver :1

nakon čega je potrebno upisati željenu 8-znakovnu lozinku. To je omogućilo korištenje tipkovnice, miša i ekrana laptopa pomoću VNC klijenta te ostvarivanje potpune kontrole nad Raspberry Pi-jem. Potreno je upaliti aplikaciju VNC Viewer, upisati IP adresu RPi-ja s portom 1, potom upisati lozinku koju smo odabrali i time je uspostavljen udaljeni pristup na Raspberry Pi.



Slika 2.7 Udaljeni pristup na Raspberry Pi s laptopa

Ako se dogodi da Rasberry krene u ponovno podizanje sustava (*reboot*) morao bih iznova svaki put pristupati terminalu pomoću PuTTY-a i paliti VNC server. Kako bi to izbjegao, podesio sam VNC server da se pali automatski svaki put kad se RPi *reboot*-a. Potrebno je pomoću terminala, locirati se u skrivenu konfiguracijsku mapu. Sada je to moguće i grafičkim putem pomoću VNC-a, nije potreban PuTTY, na način:

cd /home/pi
cd .config

Unutar te mape napravio sam mapu *autostart*, te zatim unutar nje pomoću tekstualnog editora gnome datoteku *tightvnc.desktop*.

mkdir autostart
cd autostart
gnome tightvnc.desktop

Unutar te datoteke treba zapisati sljedeće:

[Desktop Entry]
Type=Application
Name=TightVNC
Exec=vncserver :1
StartupNotify=false

Time sam omogućio da se VNC server pali automatski svaki put kada se Raspberry *reboota*.

3 Algoritam za mjerenje brzine i smjera vjetra

Glavni zadatak ovog završnog rada je realizacija algoritma za mjerenje brzine i smjera vjetra koji se temelji na mjerenju vremena propagacije ultrazvučnog više-harmonijskog signala. Sustav je temeljen na četiri ultrazvučna pretvarača koji su raspoređeni na uglovima kvadrata pogodne dimenzije, čije su dijagonale okomite osi orijentirane prema glavnim geografskim osima (sjever (N) – jug (S), istok (E) – zapad (W)). Mjerenje se provodi u dvije okomite osi (S-N, E-W) i četiri dijagonalna puta (N-W, W-S, S-E i E-N) i to u oba smjera širenja mjernog signala. Ulaz u algoritam su izmjerene vrijednosti kašnjenja na četiri odabrane staze (po dvije okomite osi i po dva dijagonalna puta), a konkretne staze su ovisne o odabiru odašiljačkih i prijemničkih pretvarača za osam mogućih kombinacija konfiguracije mjerenja. Iznosi kašnjenja se učitavaju u sustav serijskim USB sučeljem iz mjernog uređaja. Program je razvijen u jeziku C na ugradbenoj platformi Raspberry Pi.

Izračun brzine i smjera vjetra implementiran je u skladu s referentnim programskim modelom u Matlabu korištenjem 2, 4 ili 8 mjerenja na različitim mjernim stazama. Veliki dio ovog poglavlja je prolaz kroz razvijeni program i opisivanje pojedinih dijelova i njihovih funkcija u programu kao cjelini. Za početak, potrebno je uključiti sva potrebna zaglavlja datoteka kao i konfiguracijsku datoteku.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <complex.h>
#include <signal.h>
#include <limits.h>
#include <sys/file.h>
#include <fcntl.h>

#include "confFile.h"
```

Isječak koda 3.1 Uključivanje zaglavlja

Pri samom ulasku u glavnu funkciju programa, potrebno je pronaći identifikacijski broj procesa programa za prikaz i kronološku registraciju mjernih podataka kako bi se uspješno odvijala međuprocesna komunikacija. Prilikom svakog obrađeng mjerenja, program za obradu šalje signal programu za prikaz kako bi on mogao osvježiti prikaz najnovijim podatcima. Komunikacija je ostvarena uz pomoć slanja signala SIGUSR1.

```
int main(int argc, char* argv[]){

	int pid;
	char line[1024];
	FILE *p;
	p = popen("ps ax | grep main.py | grep -v grep", "r");
	if(!p)
	{
		printf("Greška prilikom traženja PID-a.\n");
		exit(1);
	}
	fgets(line, sizeof(line)-1, p);
	pid=atoi(strtok(line, " "));
	pclose(p);
```

Isječak koda 3.2 Traženje identifikacijskog broja procesa za prikaz

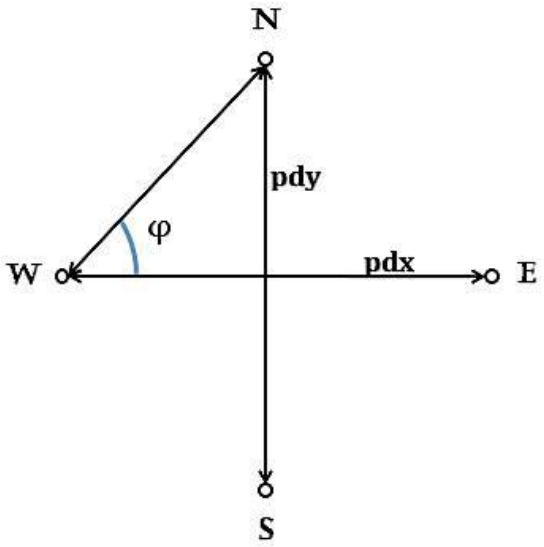
Traženje *process ID*-a (PID-a) ostvareno je funkcijom *popen* koja vraća pokazivač na tok podataka ostvaren naredbom „*ps ax | grep main.py | grep -v grep*“. Ukoliko je tok podataka ostvaren bez greški, pročitamo prvu liniju i prvi element te linije što je ujedno traženi podatak, no ukoliko je došlo do greške, ispisujemo poruku o grešci i izlazimo iz programa.

Zatim slijedi deklaracija svih potrebnih varijabli za rad programa. Tu se nalaze varijable za spremanje procjene brzine iz učitanih podataka, procjenu propagacijske brzine zvuka na svim mjernim stazama te pomoćne varijable. Varijable za pojedinu metodu izračuna bit će objašnjene kasnije, uz objašnjavanja pripadajuće metode.

```
double vx1, vx2, vx3, vx4;          // Varijable za spremanje brzine vjetra na
double vx[4];                       // horizontalnoj...

double vy1, vy2, vy3, vy4;          // ...vertikalnoj,...
double vy[4];

double vwn1, vwn2, vse1, vse2;      // ...dijagonalnoj (+45°),...
double vwnse[4];

double vws1, vws2, vne1, vne2;      // ...i drugoj dijagonalnoj  traci (-45°)
double vwsne[4];

double cj[16];                      // Rekonstrukturirana propagacijska brzina na
svih 16 traka

double phi1=atan(pdy/pdx);          // Rotacijski kut dobiven iz geometrije
postavljenog mjerenja

int i, procitao=0;                  // Pomoćne varijable
int greska = 0;

FILE *spremi;                       // Tok podataka za spremanje obrađenih
podataka
```

Isječak koda 3.3 Deklaracija dijela potrebnih varijabli

Na slici 3.1 nalazi se objašnjenje kuta φ između glavnih mjernih osi. Budući da pdx i pdy tvore dijagonale nepravilnog kvadrata njihove vrijednosti nisu potpuno jednake, ali opet su dovoljno slične da φ ispada blizu 45°.



Slika 3.1 Objašnjenje kuta φ

Glavna petlja za obradu podataka vrti se beskonačno, sve dok ne dođe do greške pri čitanju podataka s mjernog uređaja, ili dok korisnik ne ugasi program. Odnosno, sve što ćemo iduće obraditi, osim deklaracije varijabli, nalazi se u *while(true)* petlji. Za početak, tu je poziv funkcije za dohvat podataka (samu funkciju obradit ćemo kasnije). Ukoliko je došlo do greške, javlja se poruka o grešci i izlazi iz programa.

```
while(true){

        // Dohvati podatke sa mjernog uređaja
        greska = dohvatiPodatke();
        if (greska == -1)
        {
                printf("Pogreška u čitanju podataka!\n");
                exit(-1);
        }
```

Isječak koda 3.4 Ulazak u petlju i poziv funkcije za čitanje podataka

Dobiveni podatci spremaju se u varijablu *kas_tot*, točno određenim redoslijedom. Totalno kašnjenje razdijeljeno je po pojedinim stazama i mjerenju. Točan redoslijed primljenih podataka i njihova odgovarajuća staza i mjerenje su sljedeći:

Mjerenje 0

tau1_mea_NS1=kas_tot[0]

tau2_mea_WS1=kas_tot[1]

tau2_mea_WE1=kas_tot[2]

tau1_mea_NE1=kas_tot[3]

Mjerenje 1

tau1_mea_WE1=kas_tot[4]

tau2_mea_SE1=kas_tot[5]

tau2_mea_SN1=kas_tot[6]

tau1_mea_WN1=kas_tot[7]

Mjerenje 2

tau1_mea_SN1=kas_tot[8]

tau2_mea_EN1=kas_tot[9]

tau2_mea_EW1=kas_tot[10]

tau1_mea_SW1=kas_tot[11]

Mjerenje 3

tau1_mea_EW1=kas_tot[12]

tau2_mea_NW1=kas_tot[13]

tau2_mea_NS1=kas_tot[14]

tau1_mea_ES1=kas_tot[15]

Mjerenje 4

tau1_mea_WE2=kas_tot[16]

tau2_mea_NE2=kas_tot[17]

tau2_mea_NS2=kas_tot[18]

tau1_mea_WS2=kas_tot[19]

Mjerenje 5

tau1_mea_SN2=kas_tot[20]

tau2_mea_WN2=kas_tot[21]

tau2_mea_WE2=kas_tot[22]

tau1_mea_SE2=kas_tot[23]

Mjerenje 6

tau1_mea_EW2=kas_tot[24]

tau2_mea_SW2=kas_tot[25]

tau2_mea_SN2=kas_tot[26]

tau1_mea_EN2=kas_tot[27]

Mjerenje 7

tau1_mea_NS2=kas_tot[28]

tau2_mea_ES2=kas_tot[29]

tau2_mea_EW2=kas_tot[30]

tau1_mea_NW2=kas_tot[31]

, a imenovanje varijabli je:

tau(X)_mea_(track)(Y)

gdje je X=1 ili 2 (ovisno o signalu s1 ili s2 za mjerenje) i gdje je Y=1 ili 2 (1 za prvih 4 mjerenja i 2 za drugi set od 4 mjerenja s obrnutim s1/s2), *track* je smjer (sve kombinacije 4 kardinalna smjera).

Kada smo dobili podatke za obradu, potrebo je izračunati procjene brzine vjetra za četiri smjera: horizontalni, vertikalni i dva dijagonalna (+45° i -45°). Procjene se računaju pomoću jednostavne fizikalne formule za izračun brzine uz poznat put i vrijeme, odnosno **v=s/t**. U ovom slučaju to su udaljenosti između dva prijemnika kao put, te razlika u kašnjenima kao vrijeme.

```
// Pronadjemo 4 procjene za vx iz mjerenih kasnjenja na direktinm horizontalnim
trasama
vx1=pdx/2*(kas_tot[12]-kas_tot[4])/(kas_tot[12]*kas_tot[4]);       // s1 M1 & M3
vx2=pdx/2*(kas_tot[10]-kas_tot[2])/(kas_tot[10]*kas_tot[2]);       // s2 M0 & M2
vx3=pdx/2*(kas_tot[24]-kas_tot[16])/(kas_tot[24]*kas_tot[16]);     // s1 M4 & M6
vx4=pdx/2*(kas_tot[30]-kas_tot[22])/(kas_tot[30]*kas_tot[22]);     // s2 M5 & M7
vx[]={vx1, vx2, vx3, vx4};

// Pronadjemo 4 procjene za vx iz mjerenih kasnjenja na direktinm vertikalnim
trasama
vy1=pdy/2*(kas_tot[0]-kas_tot[8])/(kas_tot[0]*kas_tot[8]);         // s1 M0 & M2
vy2=pdy/2*(kas_tot[14]-kas_tot[6])/(kas_tot[14]*kas_tot[6]);       // s2 M1 & M3
vy3=pdy/2*(kas_tot[28]-kas_tot[20])/(kas_tot[28]*kas_tot[20]);     // s1 M5 & M7
vy4=pdy/2*(kas_tot[18]-kas_tot[26])/(kas_tot[18]*kas_tot[26]);     // s2 M4 & M6
vy[]={vy1, vy2, vy3, vy4};

// Takodjer pronadjemo brzinu vjetra za dijagonalne projekcije WN i SE (+45
stupnjeva)
vwn1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[31]-kas_tot[7])/(kas_tot[31]*kas_tot[7]);
        // s1 M1 & M7
vwn2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[13]-kas_tot[21])/(kas_tot[13]*kas_tot[21]);
        // s2 M3 & M5
vse1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[15]-kas_tot[23])/(kas_tot[15]*kas_tot[23]);
        // s1 M3 & M5
vse2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[29]-kas_tot[5])/(kas_tot[29]*kas_tot[5]);
        // s2 M1 & M7
vwnse[]={vwn1, vwn2, vse1, vse2};

// Takodjer pronadjemo brzinu vjetra za dijagonalne projekcije WS i NE (-45
stupnjeva)
vws1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[11]-kas_tot[19])/(kas_tot[11]*kas_tot[19]);
        // s1 M2 & M4
vws2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[25]-kas_tot[1])/(kas_tot[25]*kas_tot[1]);
        // s2 M0 & M6
vne1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[27]-kas_tot[3])/(kas_tot[27]*kas_tot[3]);
        // s1 M0 & M6
vne2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[9]-kas_tot[17])/(kas_tot[9]*kas_tot[17]);
        // s2 M2 & M4
vwsne[]={vws1, vws2, vne1, vne2};
```

Isječak koda 3.5 Izračun procjena brzine vjetra

Osim toga, potrebno je izračunati i efektivnu propagacijsku brzinu duž svih 16 mjernih staza.

```
// Rekonstruiramo efektivnu propagacijsku brzinu duz svih 16 staza
cj[0]=pdx/2*(kas_tot[12]+kas_tot[4])/(kas_tot[12]*kas_tot[4]);
cj[1]=pdx/2*(kas_tot[10]+kas_tot[2])/(kas_tot[10]*kas_tot[2]);
cj[2]=pdx/2*(kas_tot[24]+kas_tot[16])/(kas_tot[24]*kas_tot[16]);
cj[3]=pdx/2*(kas_tot[30]+kas_tot[22])/(kas_tot[30]*kas_tot[22]);
cj[4]=pdy/2*(kas_tot[0]+kas_tot[8])/(kas_tot[0]*kas_tot[8]);
cj[5]=pdy/2*(kas_tot[14]+kas_tot[6])/(kas_tot[14]*kas_tot[6]);
cj[6]=pdy/2*(kas_tot[28]+kas_tot[20])/(kas_tot[28]*kas_tot[20]);
cj[7]=pdy/2*(kas_tot[18]+kas_tot[26])/(kas_tot[18]*kas_tot[26]);

cj[8]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[31]+kas_tot[7])/(kas_tot[31]*kas_tot[7]);
cj[9]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[13]+kas_tot[21])/(kas_tot[13]*kas_tot[21]);
cj[10]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[15]+kas_tot[23])/(kas_tot[15]*kas_tot[23]);
cj[11]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[29]+kas_tot[5])/(kas_tot[29]*kas_tot[5]);
cj[12]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[11]+kas_tot[19])/(kas_tot[11]*kas_tot[19]);
cj[13]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[25]+kas_tot[1])/(kas_tot[25]*kas_tot[1]);
cj[14]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[27]+kas_tot[3])/(kas_tot[27]*kas_tot[3]);
cj[15]=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[9]+kas_tot[17])/(kas_tot[9]*kas_tot[17]);
```

Isječak koda 3.6 Izračun efektivnih propagacijskih brzina

Jednom kada smo izračunali procjene brzine vjetra za četiri glavne osi i efektivnu propagacijsku brzinu po mjernim stazama, kreće se s preciznim izračunom brzine i smjera vjetra. Uz brzinu i smjer vjetra, računa se i virtualna akustična temperatura za svaku stazu mjerenja čime se ispituje valjanost pojedinog mjerenja. Konačno rješenje moguće je odrediti iz 8, 4 ili 2 para mjerenja. Koja opcija će biti korištena prilikom izvedbe programa moguće je odrediti pomoću varijable *opcija*. U nastavku su pobliže objašnjene sve tri opcije izračuna, a potom i pomoćne funkcije.

3.1 Izračun korištenjem osam mjerenja

Korištenjem 8 mjerenja, 4 mjerna para, iz svih 16 staza moguće je izračunati jedno zajedničko rješenje. Takav način izračuna ranjiv je na pogreške jer ukoliko se pokaže da bar za jednu mjernu stazu virtualna akustična temperatura puno odudara od prosjeka, cijelo mjerenje se odbacuje. No, takav račun daje i najtočnije rješenje.

Za početak, objasnimo varijable koje će se koristiti za izračun. Varijable *vxj8* i *vyj8* služe za spremanje izračunate brzine vjetra u x (istok – zapad), odnosno y (sjever – jug) smjeru, dok *vj8* služi za spremanje konačne, apsolutne brzine vjetra. *thetj8* je varijabla u koju se sprema kut izvora vjetra gdje 0° predstavlja sjever, 90° istok, 180° jug a 270° zapad. Komponente bočnog vjetra spremaju se u *v_meas_WN_SE_side8* i *v_meas_WS_NE_side8*. Tu su još varijable za spremanje brzine zvuka i virtualne akustične temperature po stazi, te varijable za srednju vrijednost temperature i maksimalno odstupanje iste. Osim toga, koristimo i varijable za bočni vjetar. U svim nazivima varijabli, *j8* predstavlja kraticu za *joint 8*, označavajući da se radi o korištenju 8 mjerenja.

```
double vxj8, vyj8, vj8;
double thetj8;
double cj8[16] = {0};
double tempj8[16];
double tempj8_med;
double tempj8_mvar;
double v_meas_WN_SE_side8, v_meas_WS_NE_side8;
```

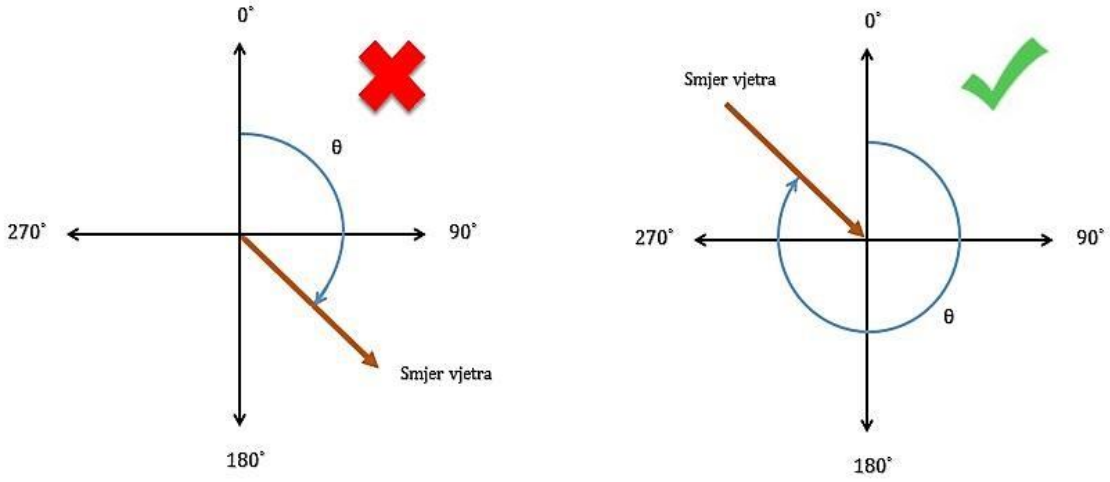
Isječak koda 3.7 Deklaracija potrebnih varijabli

Brzine vjetra izračunate su pomoću *macro* naredbe *SUM* koja vraća zbroj sva četiri elementa polja, tj. *SUM(X)* u izvođenju programa mijenja se s *(X[0]+X[1]+X[2]+X[3])*. Iz geometrije mjerenja, uz pravilnu kompenzaciju bočnog vjetra, izračunamo brzine u dva glavna smjera, a potom i apsolutnu brzinu kao duljinu hipotenuze pravokutnog trokuta s katetama *vxj8* i *vyj8*. Konačno rješenje pomnožimo s 3,6 kako bi preračunali brzinu iz m/s u km/h.

```
// Rjesenje se moze pronaci iz svih mjerenja (svih 16 mjerenih staza)
// Optimalno  rjesenje za vx i vy se moze naci iz:
vxj8=(SUM(vx)+cos(phi1)*(SUM(vwnse)+SUM(vwsne)))/4/(1+2*pow(cos(phi1),2));
vyj8=(SUM(vy)+sin(phi1)*(SUM(vwnse)-SUM(vwsne)))/4/(1+2*pow(sin(phi1),2));

// Magnituda vjetra (u km/h)
vj8=sqrt(pow(vxj8,2)+pow(vyj8,2))*3.6;

// Azimut izvora vjetra (u stupnjevima)
thetj8=atan2(-vxj8,-vyj8)/PI*180;
if (thetj8 < 0)
{
        thetj8 += 360;
}
```

Isječak koda 3.8 Izračun brzine i smjera vjetra

Smjer vjetra računamo pomoću ugrađene funkcije *atan2(y, x)* koja prima dva argumenta a vraća kut u radijanima između pozitivne x-osi i pravca koji prolazi točkom određenom kordinatama (x, y) i ishodištem sustava. Potrebno je obratiti pažnju da u pozivu funkcije elemente predajemo obrnutim redoslijedom od onoga kako je predviđeno u definiciji funkcije, te im k tomu negiramo vrijednost. Razlog je da na taj način dobivamo koordinatni sustav koji odgovara našim potrebama, tj. 0° predstavlja sjever, 90° istok itd., ali isto tako, smjer vjetra (θ) zapravo predstavlja smjer iz kojeg vjetar puše, ne smjer u kojem puše. Dobiveni rezultat dijelimo s pi (π) i množimo sa 180 kako bi dobili rješenje u stupnjevima a ne u radijanima, te mu pribrajamo 360 ukoliko je negativan.



Slika 3.2 Neispravno (lijevo) i ispravno (desno) označavanje smjera vjetra

```
// Komponenta bocnog vjetra za dijagonalne smjerove: WN_SE i WS_NE u m/s
v_meas_WN_SE_side8=-vj8*cos(thetj8/180*PI+phi1)/3.6;
v_meas_WS_NE_side8=-vj8*cos(thetj8/180*PI-phi1)/3.6;

// Kompenziranjem bocnog vjetra mozemo odrediti stvarni c na svih 16 mjernih staza
for (i=0; i<16; i++){ cj8[i]=cj[i]; }              // cj8 = cj
for (i=0; i<4; i++){ cj8[i] = sqrt(pow(cj8[i],2)+pow(vyj8,2)); }
for (i=4; i<8; i++){ cj8[i] = sqrt(pow(cj8[i],2)+pow(vxj8,2)); }
for (i=8; i<12; i++){ cj8[i] = sqrt(pow(cj8[i],2)+pow(v_meas_WN_SE_side8,2)); }
for (i=12; i<16; i++){ cj8[i] = sqrt(pow(cj8[i],2)+pow(v_meas_WS_NE_side8,2)); }

// Mozemo odrediti virtualnu akusticnu temperaturu za svaku stazu mjerenja
for (i=0; i<16; i++){ tempj8[i] = pow(cj8[i],2)/402.3147-273.15; }
tempj8_med=median(16, &tempj8[0]);              // medijan temperature
// Maksimalno apsolutno odstupanje od medijana
tempj8_mvar= fabs(tempj8[0]-tempj8_med);
for (i=1; i<16; i++){
                if ((fabs(tempj8[i]-tempj8_med)) > tempj8_mvar)
                        tempj8_mvar = fabs(tempj8[i]-tempj8_med);
}
```

Isječak koda 3.9 Izračun brzine zvuka i virtualne akustične temperature

Zatim izračunamo komponente bočnog vjetra za dijagonalne smjerove što nam je potrebno kako bi odredili stvarnu brzinu zvuka za svih 16 mjernih staza. Pomoću izračunate brzine, možemo odrediti virtualnu akustičnu temperaturu. To računamo uz pomoć formule:

$$\boldsymbol{T = \frac{c^2}{R \times \gamma}}$$

, gdje je *c* izračunata brzina zvuka, a R×γ umnožak plinske konstante i omjera specifičnih toplinskih kapaciteta što otprilike iznosi 402,3147 $JK^{-1}kg^{-1}$. Dobiveni rezultat pretvaramo iz Kelvina u stupnjeve Celzijusove. Pomoću funkcije *median* pronađemo vrijednost medijana temperature. Ukoliko virtualna akustična temperatura pojedine staze puno odudara od medijana svih izračunatih, cijelo mjerenje se odbacuje kao neispravno.

Na kraju nam još, ukoliko je mjerenje ispravno, preostaje spremiti dobiveni rezultat u datoteku. Uz izračunate podatke, u datoteku se upisuje i vremenska oznaka dobivena s mjernog uređaja. Spremanje je osigurano pomoću semafora kako bi se datoteka zaključala za korištenje samo našem programu dok je upis u tijeku i time se izbjegnu greške u radu. Kada je upis gotov, datoteka se otključa za sve ostale procese. Podizanjem korisničkog signala SIGUSR1, javljamo programu za prikaz da je novo mjerenje spremno za prikaz.

```
if (tempj8_mvar<=bmpr)
{
        spremi = fopen("Spremljeni_podatci.txt", "ab");
        flock(spremi, LOCK_EX | LOCK_NB);              // Zakljucavanje datoteke
        fprintf(spremi, "%d;%f;%f\r\n", vremenska_oznaka, vj8, thetj8);
        flock(spremi, LOCK_UN);                        // Otkljucavanje datoteke
        fclose(spremi);
        kill(pid, SIGUSR1);
}
```

Isječak koda 3.10 Spremanje podataka

3.2 Izračun korištenjem četiri mjerenja

Umjesto jednog zajedničkog rješenja za svih 8 mjerenja, možemo izračunati 2 konačna rješenja iz 2 seta od 4 mjerenja. Iz 4 parna mjerenja (M0, M2, M4 i M6) dobijemo 2 horizontalne, 2 vertikalne i 4 dijagonalne (-45°) projekcije. Analogno, iz 4 neparna (M1, M3, M5 i M7) mjerenja također dobijemo 2 horizontalne, 2 vertikalne i 4 dijagonalne (+45°) projekcije.

Za izračun konačnih rješenja, potrebno nam je nekoliko varijabli sličnih onima u izračunu korištenjem 8 mjerenja, ali ovaj puta neke od njih su 2 puta većih dimenzija jer očekujemo i dva puta više konačnih rješenja. Slično kao i prije, *j4* označava da se radi o korištenju 4 mjerenja, tj. to predstavlja *joint 4*. Nove varijable koje se do sada nisu pojavile su *M_even* i *M_odd* koje predstavljaju „matrice" važnosti pojedinog parnog odnosno neparnog mjerenja, te *v_even* i *v_odd* koje služe za izračunavanje po dva rješenja za parna i neparna mjerenja.

```
double M_even[4],  M_odd[4];
double v_even[2], v_odd[2];
double vxj4[4], vyj4[4], vj4[2];
double thetj4;
double cj4[16] = {0};
double tempj4[16], tempj4_med[2], tempj4_mvar[2];
double v_meas_WN_SE_side4, v_meas_WS_NE_side4;
```

Isječak koda 3.11 Deklaracija potrebnih varijabli

„Matrice“ važnosti računaju se uz pomoć varijable *phi1*, a nakon toga izračunamo i dva rješenja brzine vjetra za parna, i dva rješenja za neparna mjerenja. Kombinacijom dva rješenja za parna mjerenja izračunamo apsolutnu brzinu vjetra za jedan set mjerenja (parna), a kombinacijom rješenja za neparna dobivamo apsolutno rješenje za drugi set (neparna). Dobivene vrijednosti su u km/h.

```
M_even[0]=(2-cos(2*phi1))/6;
M_even[1]=(sin(2*phi1))/6;
M_even[2]=(sin(2*phi1))/6;
M_even[3]=(cos(2*phi1)+2)/6;
M_odd[0]=(2-cos(2*phi1))/6;
M_odd[1]=(-sin(2*phi1))/6;
M_odd[2]=(-sin(2*phi1))/6;
M_odd[3]=(cos(2*phi1)+2)/6;

// Izracun dva rjesenja za parna i neparna mjerenja
v_even[0]=(M_even[0]*(vx2+vx3+cos(phi1)*SUM(vwsne)))+
                (M_even[1]*(vy1+vy4-sin(phi1)*SUM(vwsne)));
v_even[1]=(M_even[2]*(vx2+vx3+cos(phi1)*SUM(vwsne)))+
                (M_even[3]*(vy1+vy4-sin(phi1)*SUM(vwsne)));
v_odd[0]=(M_odd[0]*(vx1+vx4+cos(phi1)*SUM(vwnse)))+
                (M_odd[1]*(vy2+vy3+sin(phi1)*SUM(vwnse)));
v_odd[1]=(M_odd[2]*(vx1+vx4+cos(phi1)*SUM(vwnse)))+
                (M_odd[3]*(vy2+vy3+sin(phi1)*SUM(vwnse)));

// X & Y projekcije (prvo parni, zatim neparni)
vxj4[0]=v_even[0];
vxj4[1]=v_odd[0];
vyj4[0]=v_even[1];
vyj4[1]=v_odd[1];
// Dva konacna rjesenja za  procjene vrijednosti brzine vjetra (u km/h)
vj4[0]=sqrt(pow(vxj4[0],2)+pow(vyj4[0],2))*3.6;
vj4[1]=sqrt(pow(vxj4[1],2)+pow(vyj4[1],2))*3.6;
```

Isječak koda 3.12 Izračun brzine vjetra

Izračunom komponenti bočnog vjetra i njihovom kompenzacijom na odgovarajućim trakama, možemo odrediti efektivnu propagacijsku brzinu zvuka za svaku mjernu traku.

```
v_meas_WN_SE_side4=-vj4[1]*cos(thetj4[1]/180*PI+phi1)/3.6;  // za neparna
v_meas_WS_NE_side4=-vj4[0]*cos(thetj4[0]/180*PI-phi1)/3.6;  // za parna mjerenja

cj4[]={cj[1], cj[2], cj[4], cj[7], cj[12], cj[13], cj[14], cj[15],  // parna mjerenja
                  cj[0], cj[3], cj[5], cj[6], cj[8],  cj[9], cj[10], cj[11]};  // neparna mjernja
cj4[0]=sqrt(pow(cj4[0],2)+pow(vyj4[0],2));
cj4[1]=sqrt(pow(cj4[1],2)+pow(vyj4[0],2));
cj4[8]=sqrt(pow(cj4[8],2)+pow(vyj4[1],2));
cj4[9]=sqrt(pow(cj4[9],2)+pow(vyj4[1],2));
cj4[2]=sqrt(pow(cj4[2],2)+pow(vxj4[0],2));
cj4[3]=sqrt(pow(cj4[3],2)+pow(vxj4[0],2));
cj4[10]=sqrt(pow(cj4[10],2)+pow(vxj4[1],2));
cj4[11]=sqrt(pow(cj4[11],2)+pow(vxj4[1],2));

cj4[4]=sqrt(pow(cj4[4],2)+pow(v_meas_WS_NE_side4,2));
cj4[5]=sqrt(pow(cj4[5],2)+pow(v_meas_WS_NE_side4,2));
cj4[6]=sqrt(pow(cj4[6],2)+pow(v_meas_WS_NE_side4,2));
cj4[7]=sqrt(pow(cj4[7],2)+pow(v_meas_WS_NE_side4,2));
cj4[12]=sqrt(pow(cj4[12],2)+pow(v_meas_WN_SE_side4,2));
cj4[13]=sqrt(pow(cj4[13],2)+pow(v_meas_WN_SE_side4,2));
cj4[14]=sqrt(pow(cj4[14],2)+pow(v_meas_WN_SE_side4,2));
cj4[15]=sqrt(pow(cj4[15],2)+pow(v_meas_WN_SE_side4,2));
```

Isječak koda 3.13 Izračun efektivne propagacijske brzine zvuka

Kada smo izračunali brzinu zvuka po stazamo, možemo odrediti virtualnu akustičnu temperaturu za svih 8 mjerenja. Ukoliko neka vrijednost puno odudara od medijana za taj skup, odbacujemo samo taj pripadajući skup, ne i cijelo mjerenje. Na taj način povećali smo robusnost programa i njegovu otpornost na greške.

```
// Mozemo izracunati virutalnu akusticnu temperaturu za oba rjesenje za svih 8 traka
for (i=0; i<16; i++){ tempj4[i] = pow(cj4[i],2)/402.3147-273.15; }
tempj4_med[0]=median(8, &tempj4[0]);                    // medijan temperature
tempj4_med[1]=median(8, &tempj4[8]);
// Maksimalno apsolutno odstupanje od medijana
tempj4_mvar[0]= fabs(tempj4[0]-tempj4_med[0]);
for (i=1; i<8; i++){
        if ((fabs(tempj4[i]-tempj4_med[0])) > tempj4_mvar[0])
                tempj4_mvar[0] = fabs(tempj4[i]-tempj4_med[0]);
}
tempj4_mvar[1]= fabs(tempj4[8]-tempj4_med[1]);
for (i=8; i<16; i++){
        if ((fabs(tempj4[i]-tempj4_med[1])) > tempj4_mvar[1])
                tempj4_mvar[1] = fabs(tempj4[i]-tempj4_med[1]);
}
```

Isječak koda 3.14 Izračun virtualne akustične temperature

Slično kao i za metodu s 8 mjerenja, izračunamo azimut izvora vjetra pomoću funkcije *atan2*. Razlika je u tome što ovaj put u funkciju predajemo srednje, ispravne vrijednosti za brzinu vjetra u x i y smjeru. Ne možemo izračunati vrijednosti smjera vjetra za svaki mjerni set, pa iz njih računati srednju vrijednost jer tako postoji kritična točka u 0°, pa stoga računamo jednu vrijednost kuta iz srednjih vrijednosti brzina.

```
// Konacno rjesenje za rekonstruirani kut smjera vjetra (u stupnjevima)
thetj4=atan2(-mean(2, &vxj4[0], &tempj4_mvar[0]),
                -mean(2, &vyj4[0], &tempj4_mvar[0]))/PI*180;
if(thetj4<0){
        thetj4 += 360;
}
```

Isječak koda 3.15 Izračun smjera vjetra

Za kraj, potrebno je spremiti dobivene rezultate u datoteku. Funkcija *provjeri* koja se nalazi u dijelu za uvjet *if* naredbe vratit će pozitivnu vrijednost ako je barem jedna od dvije izračunate brzine ispravna. Potpuni opis funkcije *mean* se nalazi kasnije. Uz izračunate rezultate, u datoteku se također zapisuje i vremenska oznaka, a upisivanje je osigurano zaključavanjem datoteke dok se upis ne završi.

```
if (provjeri(2,  &tempj4_mvar[0])) {
        spremi = fopen("Spremljeni_podatci.txt", "ab");
        flock(spremi, LOCK_EX | LOCK_NB);              // Zakljucavanje datoteke
        fprintf(spremi, "%d;%f;%f\r\n", vremenska_oznaka,
                mean(2, &vj4[0], &tempj4_mvar[0]), thetj4);
        flock(spremi, LOCK_UN);                        // Otkljucavanje datoteke
        fclose(spremi);
        kill(pid, SIGUSR1);
}
```

Isječak koda 3.16 Spremanje podataka

3.3 Izračun korištenjem dva mjerenja

Za najbolju robusnost programa, računanje ćemo obaviti grupiranjem 8 mjerenja u 4 para (M0 i M2, M4 i M6, M1 i M3, M5 i M7). Iz svakog para dobijemo jedno horizontalno rješenje, jedno vertikalno, te dva dijagonalna rješenja s mješanim signalima (s1 u jednom smjeru i s2 u drugom).

Varijable koje se koriste za ovaj izračun su ponovno slične varijablama za predhodne izračuna. Pri ovom izračunu također se koriste „matrice“ važnosti kao u izračunu uz pomoć 4 mjerenja. Varijable *v_02*, *v_46*, *v_13*, *v_57* služe sa spremanje izračunate brzine vjetra u x i y smjeru za svaki par mjerenja. Varijable za spremanje konačnih rješenja dvostruko su veće od prethodnog računanja, odnosno četiri puta veće od prvog načina računanja.

```
double vmwn1, vmwn2, vmse1, vmse2;
double vmws1, vmws2, vmne1, vmne2;
double cmws1, cmne1, cmws2, cmne2;
double cmwn1, cmse1, cmwn2, cmse2;
double v_02[2], v_46[2], v_13[2], v_57[2];
double vxj2[4], vyj2[4], vj2[4];
double cj2[16];
double thetj2;
double tempj2[16], tempj2_med[4], tempj2_mvar[4];
double v_meas_WN_SE_side_13, v_meas_WN_SE_side_57,
       v_meas_WS_NE_side_02, v_meas_WS_NE_side_46, v_meas_diag_side_2[4];
```

Isječak koda 3.17 Deklaracija potrebnih varijabli

Pri izračunu brzine i smjera vjetra ovom metodom, najprije pronađemo brzinu vjetra za dijagonalne projekcije a potom i rekonstruiramo efektivnu propagacijsku brzinu duž četiri dijagonalne mješovite trake. Ti rezultati su nam potrebni za daljnji rad algoritma.

```
// Pronadjemo brzinu vjetra za dijagonalne projekcije WN i SE (+45°)
vmwn1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[13]-
                kas_tot[7])/(kas_tot[13]*kas_tot[7]);          // s1&2 M1 & M3
vmwn2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[31]-
                kas_tot[21])/(kas_tot[31]*kas_tot[21]);        // s1&2 M5 & M7
vmse1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[15]-
                kas_tot[5])/(kas_tot[15]*kas_tot[5]);          // s1&2 M1 & M3
vmse2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[29]
                -kas_tot[23])/(kas_tot[29]*kas_tot[23]);       // s1&2 M5 & M7

// Takodjer, pronadjemo brzinu vjetra za dijagonalne projekcije WS i NE (-45°)
vmws1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[11]-
                kas_tot[1])/(kas_tot[11]*kas_tot[1]);          // s1&2 M0 & M2
vmws2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[25]-
                kas_tot[19])/(kas_tot[25]*kas_tot[19]);        // s1&2 M4 & M6
vmne1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[9]-
                kas_tot[3])/(kas_tot[9]*kas_tot[3]);           // s1&2 M0 & M2
vmne2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[27]-
                kas_tot[17])/(kas_tot[27]*kas_tot[17]);        // s1&2 M4 & M6

// Mozemo rekonstruirati efektivnu propagacjsku brzinu duz 4 dijagonalne
// mjesovite trake. Imamo dvije procjene za svaki mjerni par.
cmws1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[11]+kas_tot[1])/(kas_tot[11]*kas_tot[1]);
cmne1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[9]+kas_tot[3])/(kas_tot[9]*kas_tot[3]);
cmws2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[25]+kas_tot[19])/(kas_tot[25]*kas_tot[19]);
cmne2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[27]+kas_tot[17])/(kas_tot[27]*kas_tot[17]);
cmwn1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[13]+kas_tot[7])/(kas_tot[13]*kas_tot[7]);
cmse1=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[15]+kas_tot[5])/(kas_tot[15]*kas_tot[5]);
cmwn2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[31]+kas_tot[21])/(kas_tot[31]*kas_tot[21]);
cmse2=sqrt(pdx*pdx+pdy*pdy)/4*(kas_tot[29]+kas_tot[23])/(kas_tot[29]*kas_tot[23]);
```

Isječak koda 3.18 Izračun pomoćnih varijabli

Sljedeće što je potrebno napraviti je uz pomoć matrica važnosti izračunati brzine vjetra za sva četiri seta mjerenja, u oba glavna smjera.

```
// Rjesenje nadjemo kao produkt vaznosti mjerenja sa matricom (M_even za parni set
mjerenja, i M_odd za neparni set):
M_even[0]=(2-cos(2*phi1))/3;
M_even[1]=sin(2*phi1)/3;
M_even[2]=sin(2*phi1)/3;
M_even[3]=(cos(2*phi1)+2)/3;

M_odd[0]=(2-cos(2*phi1))/3;
M_odd[1]=-sin(2*phi1)/3;
M_odd[2]=-sin(2*phi1)/3;
M_odd[3]=(cos(2*phi1)+2)/3;

// Izracun cetiri rjesenja iz 2 parna i 2 neparna mjerenja
v_02[0]=(M_even[0]*(vx2+cos(phi1)*(vmws1+vmne1)))+
                (M_even[1]*(vy1-sin(phi1)*(vmws1+vmne1)));
v_02[1]=(M_even[2]*(vx2+cos(phi1)*(vmws1+vmne1)))+
                (M_even[3]*(vy1-sin(phi1)*(vmws1+vmne1)));
v_46[0]=(M_even[0]*(vx3+cos(phi1)*(vmws2+vmne2)))+
                (M_even[1]*(vy4-sin(phi1)*(vmws2+vmne2)));
v_46[1]=(M_even[2]*(vx3+cos(phi1)*(vmws2+vmne2)))+
                (M_even[3]*(vy4-sin(phi1)*(vmws2+vmne2)));

v_13[0]=(M_odd[0]*(vx1+cos(phi1)*(vmwn1+vmse1)))+
                (M_odd[1]*(vy2+sin(phi1)*(vmwn1+vmse1)));
v_13[1]=(M_odd[2]*(vx1+cos(phi1)*(vmwn1+vmse1)))+
                (M_odd[3]*(vy2+sin(phi1)*(vmwn1+vmse1)));
v_57[0]=(M_odd[0]*(vx4+cos(phi1)*(vmwn2+vmse2)))+
                (M_odd[1]*(vy3+sin(phi1)*(vmwn2+vmse2)));
v_57[1]=(M_odd[2]*(vx4+cos(phi1)*(vmwn2+vmse2)))+
                (M_odd[3]*(vy3+sin(phi1)*(vmwn2+vmse2)));
```

Isječak koda 3.19 Izračun brzina vjetra za sve skupove mjerenja

Iz izračunatih podataka dobivamo 4 rezultata za apsolutnu brzinu i smjer vjetra. Iz svakog para mjerenja dobili smo jedno konačno rješenje. Upravo zbog toga kažemo da ova metoda ima najveću robusnost, jer ukoliko se pokaže da bila greška u jednom setu mjerenja, ostaju nam još uvijek 3. Ponovno, iznos brzine vjetra računamo kao hipotenuzu pravokutnog trokuta s katetama brzine vjetra u x i y smjeru. Izračun kuta smjera vjetra ostvaren je pomoću funkcija *atan2* i *mean*, a nužno je da se izvede nakon određivanja virtualnih akustičnih temperatura (zbog prepravki koda, u isječcima koda ostalo je na istom mjestu).

```
// X & Y projekcije
vxj2[0]=v_02[0];
vxj2[1]=v_46[0];
vxj2[2]=v_13[0];
vxj2[3]=v_57[0];

vyj2[0]=v_02[1];
vyj2[1]=v_46[1];
vyj2[2]=v_13[1];
vyj2[3]=v_57[1];

// Cetiri procjene jakosti vjetra (u km/h)
for (i=0; i<4; i++){
        vj2[i]=sqrt(pow(vxj2[i],2)+pow(vyj2[i],2))*3.6;
}
// Konacno rjesenje za rekonstruirani kut smjera vjetra (u stupnjevima)
thetj2=atan2(-mean(4, &vxj2[0], &tempj2_mvar[0]),
                -mean(4, &vyj2[0], &tempj2_mvar[0]))/PI*180;
 (thetj2<0){
        thetj2 += 360; }
```

Isječak koda 3.20 Izračun brzine i smjera vjetra

Nakon što smo izračunali brzinu i smjer vjetra za svaki par, potrebno je provjeriti ispravnost dobivenih podataka. To ćemo ponovno ostvariti ispitvanjem virtualne akustične temperature, no da bismo došli do tih podataka, najprije trebamo izračunati komponente bočnog vjetra za dijagonalne smjerove. Dobivene rezultate spremamo u zasebno polje kako bi poslije s njima lakše rukovali.

```
// Komponenta bocnog vjetra za dijagonali vektor smjera WN_SE i WS_NE u m/s
v_meas_WN_SE_side_13=-vj2[2]*cos(thetj2[2]/180*PI+phi1)/3.6;  // za neparna
v_meas_WN_SE_side_57=-vj2[3]*cos(thetj2[3]/180*PI+phi1)/3.6;  // za neparna
v_meas_WS_NE_side_02=-vj2[0]*cos(thetj2[0]/180*PI-phi1)/3.6;  // za parna
v_meas_WS_NE_side_46=-vj2[1]*cos(thetj2[1]/180*PI-phi1)/3.6;  // za parna

// Vektor bocnog vjetra za dijagonalne trake
v_meas_diag_side_2[0]=v_meas_WS_NE_side_02;
v_meas_diag_side_2[1]=v_meas_WS_NE_side_46;
v_meas_diag_side_2[2]=v_meas_WN_SE_side_13;
v_meas_diag_side_2[3]=v_meas_WN_SE_side_57;
```

Isječak koda 3.21 Izračun komponenti bočnog vjetra

Potom, potrebno je napraviti propagacijsku kompenzaciju u procjenama efektivne propagacijske brzine zvuka u x i y smjeru, te u oba dijagonalna, za svaku traku u sva četiri mjerna para.

```
// Procjena efektivne propagacijske brzine zvuka za sva cetiri
// mjerna para (cetiri trake: 1xX, 1xY i 2xdiag za svaki red)
cj2[]={cj[1], cj[4], cmws1, cmne1,     // M0_2
       cj[2], cj[7], cmws2, cmne2,     // M4_6
       cj[0], cj[5], cmwn1, cmse1,     // M1_3
       cj[3], cj[6], cmwn2, cmse2};    // M5_7
// Propagacijska kompenzacija u X smjeru zbog bocnog vjetra vy
// (za horizontalnu traku)
for (i=0; i<4; i++){
       cj2[4*i]=sqrt(pow(cj2[4*i],2)+pow(vyj2[i],2));
}
// Propagacijska kompenzacija u Y smjeru zbog bocnog vjetra vx
// (za vertikalnu traku)
for (i=0; i<4; i++){
       cj2[4*i+1]=sqrt(pow(cj2[4*i+1],2)+pow(vxj2[i],2));
}
// Propagacijska kompenzacija u dijagonalnom smjeru zbog bocnog vjetra
for (i=0; i<4; i++){
       cj2[4*i+2]=sqrt(pow(cj2[4*i+2],2)+pow(v_meas_diag_side_2[i],2));
              // za prvu dijagonalnu traku
       cj2[4*i+3]=sqrt(pow(cj2[4*i+3],2)+pow(v_meas_diag_side_2[i],2));
              // za drugu dijagonalnu traku
}
```

Isječak koda 3.22 Izračun efektivnih propagacijskih brzina zvuka

Konačno, uz pomoć izračunatih propagacijskih brzina zvuka, možemo izračunati virtualne akustične temperature za svaku stazu mjerenja. Izračuni medijana i maksimalnog odstupanja, grupirani su po odgovarajućim setovima. Ispitivanje ispravnosti jednog seta, ne utječe na ostale, a moguće je odbaciti čak 3 seta a da mjerenje još uvijek smatramo uspješnim.

```
// Konacno, odredimo virtualnu akusticnu temperaturu za sva cetiri mjerna rjesenja
for (i=0; i<16; i++){ tempj2[i] = pow(cj2[i],2)/402.3147-273.15; }
tempj2_med[0]=median(4, &tempj2[0]);                // medijani temperatura
tempj2_med[1]=median(4, &tempj2[4]);
tempj2_med[2]=median(4, &tempj2[8]);
tempj2_med[3]=median(4, &tempj2[12]);
// Maksimalno apsolutno odstupanje od medijana za svaki set
tempj2_mvar[0]= fabs(tempj2[0]-tempj2_med[0]);
for (i=1; i<4; i++){
        if ((fabs(tempj2[i]-tempj2_med[0])) > tempj2_mvar[0])
                tempj2_mvar[0] = fabs(tempj2[i]-tempj2_med[0]);
}
tempj2_mvar[1]= fabs(tempj2[4]-tempj2_med[1]);
for (i=5; i<8; i++){
        if ((fabs(tempj2[i]-tempj2_med[1])) > tempj2_mvar[1])
                tempj2_mvar[1] = fabs(tempj2[i]-tempj2_med[1]);
}
tempj2_mvar[2]= fabs(tempj2[8]-tempj2_med[2]);
for (i=9; i<12; i++){
        if ((fabs(tempj2[i]-tempj2_med[2])) > tempj2_mvar[2])
                tempj2_mvar[2] = fabs(tempj2[i]-tempj2_med[2]);
}
tempj2_mvar[3]= fabs(tempj2[12]-tempj2_med[3]);
for (i=13; i<16; i++){
        if ((fabs(tempj2[i]-tempj2_med[3])) > tempj2_mvar[3])
                tempj2_mvar[3] = fabs(tempj2[i]-tempj2_med[3]);
}
```

Isječak koda 3.23 Izračun virtualnih akustičnih temperatura

Preostaje još spremiti izračunate podatke u datoteku. Kraj ove metode ujedno je i kraj glavnog programa. Još jednom napomena da cijeli algoritam, od čitanja podataka sa serijskog USB ulaza do upisa podataka pomoću jedne od metoda, se vrti u beskonačnoj *while* petlji.

```
if (provjeri(4, &tempj2_mvar[0])) {
        srednji = mean2(4, &thetj2[0], &tempj2_mvar[0]);
        if(srednji<0) srednji += 360;
        spremi = fopen("Spremljeni_podatci.txt", "ab");
        flock(spremi, LOCK_EX | LOCK_NB);               // Zakljucavanje datoteke
        fprintf(spremi, "%d;%f;%f\r\n", vremenska_oznaka,
                mean(4, &vj2[0],&tempj2_mvar[0]), srednji);
        flock(spremi, LOCK_UN);                         // Otkljucavanje datoteke
        fclose(spremi);
        kill(pid, SIGUSR1);}
```

Isječak koda 3.24 Spremanje podataka

3.4 Pomoćna funkcija *mean*

Pomoćna funkcija *mean* služi za vraćanje srednje vrijednosti izračunatih podataka, ali u račun uzima samo one podatke koji su ispravni. Funkcija prima tri argumenta za ulaz. Prvi argument je broj elemenata u polju za računanje i provjeru. Pri pozivima funkcije u metodi izračuna korištenjem 4 mjerenja, taj argument iznosi 2, odnosno u metodi korištenjem 2 mjernja, on iznosi 4. Drugi argument je polje vrijednosti izračunatih rješenja. Treći elemet je polje za provjeru, odnosno u pozivima funkcije to je polje maksimalnih odstupanja virtualne akustične temperature od medijana za svaki set mjerenih parova. Funkcija vraća vrijednost prvog elementa ukoliko je ispravan, tj. ako je vrijednost maksimalnog odstupanja temperature od medijana manja od zadane granice, u protivnom funkcija vraća 0. Ako je pak zadano više od jednog elementa, u petlji će se proći kroz sve vrijednosti polja za izračun i provjeru i zbrojiti samo ispravne vrijednosti. Ukoliko je pronađena bar jedna ispravna vrijednost, vratit će se srednji iznos svih na taj način da će se zbroj ispravnih vrijednosti podijeliti s njihovim brojem. Pozivi funkcije u argumentu *if* naredbe u izračunima, omogućit će ulazak u naredbu i spremanje podataka samo ako je barem jedno rješenje za bilo koju mjernu stazu ispravno. Ako nijedno mjerenje nije ispravno, funkcija *mean* će povratkom nule onemogućiti ispunjenje naredbe *if*.

```
double mean(int n, double polje[], double provjera[]) {
	int i, j=0;
	double sum=0;
	if (n==1 && provjera[0]<=bmpr) return polje[0];
	if (n==1 && provjera[0]>bmpr)  return 0;
	for(i=0;i<n;i++){
		if (provjera[i]<=bmpr){
			sum+=polje[i];
			j++;
		}
	}
	if (sum == 0) return 0;
	return sum/j;
}
```

Isječak koda 3.25 Pomoćna funkcija mean

3.5 Pomoćna funkcija *median*

Funkcija *median*, kako joj i samo ime kaže, traži vrijednost medijan ulaznog polja. Ulazi u funkciju su dva argumenta: broj elemenata u polju i samo polje u kojem će se tražiti medijan. Funkcija najprije napravi pomoćno polje *x* i njega popuni istim podatcima kao i ulazno polje. Nakon toga, na tom novom, pomoćnom polju, uz dvije *for* petlje prođemo i sortiramo ga uzlazno *bubble sortom*. Ako je broj elemenata paran, a u našem slučaju to je uvijek istina, jer se funkcija poziva pri traženju medijana za virtualnu akustičnu temperaturu i broj mjerenja je uvijek paran u svakom setu, vratit će se srednja vrijednost dva elementa u sredini polja. Ako se pak kojim slučajem dogodi da je broj elemenata u bolju neparan, vratit ćemo vrijednost elementa u sredini.

```
double median(int n, double polje[]) {
   double temp;
   int i, j;
         double *x=malloc(n*sizeof(double));
         for (i=0;i<n;i++){
                  x[i]=polje[i];
         }
   /* Sljedece dvije petlje sortiraju polje x u uzlaznom redu */
   for(i=0; i<n-1; i++) {
      for(j=i+1; j<n; j++) {
         if(x[j] < x[i]) {
            /* zamijeni elemente */
            temp = x[i];
            x[i] = x[j];
            x[j] = temp;
         }
      }
   }

   if(n%2==0) {
      /* Ako je broj elemenata paran, vrati srednju vrijednost srednja dva elementa*/
      return((x[n/2] + x[n/2 - 1]) / 2.0);
   } else {
      /* U protivnom vrati vrijednost elementa u sredini */
      return x[n/2];
   }
}
```

Isječak koda 3.26 Pomoćna funkcija median

3.6 Pomoćna funkcija *dohvatiPodatke*

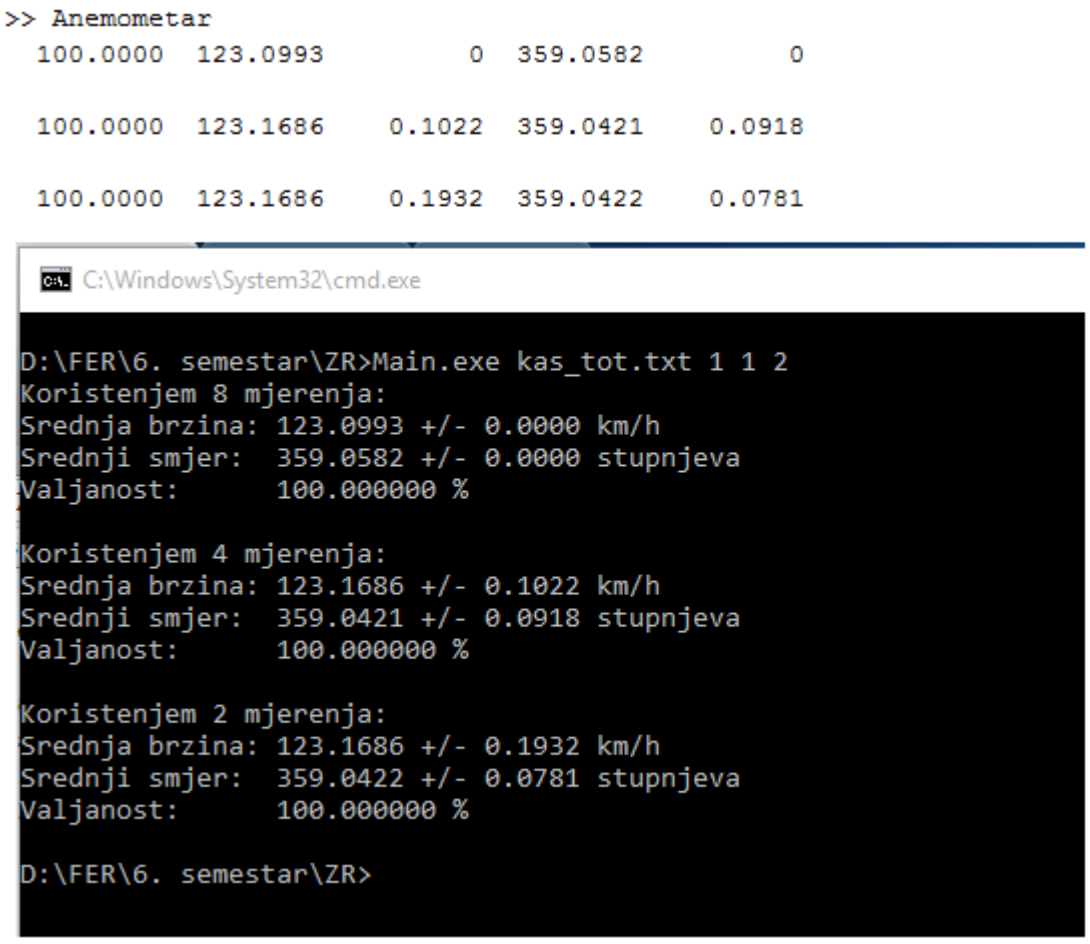
Budući da mjerni uređaj nije u potpunosti implentiran i nema podataka za primanje sa serijskog USB sučelja, funkcija za dohvat podataka simulira čitanje ulaznih podataka na način da ih zapravo čita iz datoteke. Ulazni podatci su generirani uz pomoć Matlaba i predstavljaju podatke kakvi bi se uistinu mogli pojaviti na serijskom sučelju: cijelobrojne vrijednosti u intervalu $[-2^{15} , 2^{15}]$ koje zapravo predstavljaju vrijednosti između -1 i 1. Kako bismo ponovno dobili točne vrijednosti za računanje, potrebno je pročitane podatke dijeliti s 2^{15}, a dobivene vrijednosti spremaju se u globalnu varijablu *kas_tot* koja se koristi u glavnom programu. Implementacija ispravne funkcije za čitanje sa serijskog USB sučelja ostavljena je za kasniji rad.

```
int dohvatiPodatke(){
	int podatak, i;
	FILE *file = fopen("kas_totCB.txt", "rb");;
	if (file == NULL){
		printf("Nije moguce otvoriti file\n");
		return -1;
	}

	for (i = 0 ; i < 32; i++){
		fscanf(file, "%d", &podatak);
		kas_tot[i] = podatak/(2^15);
	}
	return 0;

}
```

Isječak koda 3.27 Pomoćna funkcija dohvatiPodatke

3.7 Usporedba implementacije s referentnim Matlab kodom

Razvijeni program, za iste ulazne podatke, daje potpuno iste, do bita identične rezultate kao i referentni Matlab kod. Rezultat jedne takve izvedbe moguće je vidjeti na slici 3.3. Rezultati Matlaba su redom po stupcima: broj točnih mjerenja (u ovom testnom slučaju to je jedno, ispravno mjerenje), srednja brzina vjetra, standardna devijacija brzine vjetra, srednji smjer vjetra te standardna devijacija smjera. Prvi red su rezultati za izračun korištenjem 8 mjerenja, drugi red izračun korištenjem 4 mjerenja, a treći korištenjem 2. Rezultati dobiveni implementiranim algoritmom u jeziku C, vidljivi na donjem dijelu slike, u potpunosti su identični rezultatima iz Matlaba. Razlog tomu je taj da programski jezik C i Matlab koriste jednak broj bitova za spremanje i operacije s brojevima s *double* preciznošću, ali i sa svim ostalim tipovima podataka.



Slika 3.3 Usporedba rezultata iz Matlaba (gore) s rezultatima in C-a (dolje)

Izvedba algoritma je doduše bila složenija jer je bilo potrebno sve matrične operacije iz Matlaba razbiti na jednostavne skalarne operacije kakve je moguće izvesti u C-u. Jedna od takvih operacija je i računanje „matrica“ važnosti i njihovo množenja za izračun brzine vjetra, vidljiva na slici 3.4. Dok su one u Matlabu zbilja matrice, u C-u ih je bilo lakše izvesti u obliku polja. Samo množenje najprije sam morao izvesti na papiru i vidjeti koji element se množi s kojim kako bih to mogao prevesti u C. Drugi primjer je operacija propagacijske kompenzacije u x smjeru zbog bočnog vjetra koja je u Matlabu izvedena kao jedna linija koda koja se obavlja nad svim elementima te matrice, a u C-u sam tu istu operaciju morao raspisati u *for* petlji.

```
M_even=[2-cos(2*phi1) sin(2*phi1); sin(2*phi1) cos(2*phi1)+2]/3;
M_odd=[2-cos(2*phi1) -sin(2*phi1); -sin(2*phi1) cos(2*phi1)+2]/3;

% Compute four joint solutions from 2 even and 2 odd measurements
v_02=M_even*[vx2+cos(phi1)*sum([vmws1 vmne1]); vy1-sin(phi1)*sum([vmws1 vmne1])];
v_46=M_even*[vx3+cos(phi1)*sum([vmws2 vmne2]); vy4-sin(phi1)*sum([vmws2 vmne2])];
v_13= M_odd*[vx1+cos(phi1)*sum([vmwn1 vmse1]); vy2+sin(phi1)*sum([vmwn1 vmse1])];
v_57= M_odd*[vx4+cos(phi1)*sum([vmwn2 vmse2]); vy3+sin(phi1)*sum([vmwn2 vmse2])];
```

```
// Rjesenje nadjemo kao produkt tezinskog mjerenja sa matricom (M_even za parni set mjerenja, i M_odd za neparni set):
M_even[0]=(2-cos(2*phi1))/3;
M_even[1]=sin(2*phi1)/3;
M_even[2]=sin(2*phi1)/3;
M_even[3]=(cos(2*phi1)+2)/3;

M_odd[0]=(2-cos(2*phi1))/3;
M_odd[1]=-sin(2*phi1)/3;
M_odd[2]=-sin(2*phi1)/3;
M_odd[3]=(cos(2*phi1)+2)/3;

// Izracun cetiri rjesenja iz 2 parna i 2 neparna mjerenja
v_02[0]=(M_even[0]*(vx2+cos(phi1)*(vmws1+vmne1)))+(M_even[1]*(vy1-sin(phi1)*(vmws1+vmne1)));
v_02[1]=(M_even[2]*(vx2+cos(phi1)*(vmws1+vmne1)))+(M_even[3]*(vy1-sin(phi1)*(vmws1+vmne1)));

v_46[0]=(M_even[0]*(vx3+cos(phi1)*(vmws2+vmne2)))+(M_even[1]*(vy4-sin(phi1)*(vmws2+vmne2)));
v_46[1]=(M_even[2]*(vx3+cos(phi1)*(vmws2+vmne2)))+(M_even[3]*(vy4-sin(phi1)*(vmws2+vmne2)));

v_13[0]=(M_odd[0]*(vx1+cos(phi1)*(vmwn1+vmse1)))+(M_odd[1]*(vy2+sin(phi1)*(vmwn1+vmse1)));
v_13[1]=(M_odd[2]*(vx1+cos(phi1)*(vmwn1+vmse1)))+(M_odd[3]*(vy2+sin(phi1)*(vmwn1+vmse1)));

v_57[0]=(M_odd[0]*(vx4+cos(phi1)*(vmwn2+vmse2)))+(M_odd[1]*(vy3+sin(phi1)*(vmwn2+vmse2)));
v_57[1]=(M_odd[2]*(vx4+cos(phi1)*(vmwn2+vmse2)))+(M_odd[3]*(vy3+sin(phi1)*(vmwn2+vmse2)));
```

Slika 3.4 Usporedba izvedbe iste naredbe u Matlabu (gore) i u C-u (dolje)

3.8 Analiza složenosti i vremena izvedbe

Složenost razvijenog programa *a priori* analizom je O(n) jer se unutar cijelog algoritma sve petlje obavljaju najviše n puta. Točno vrijeme izvedbe za jedno mjerenje, s 32 ulazna podatka, nije moguće precizno izmjeriti jer se algoritam toliko brzo izvede da nije moguće izmjeriti proteklo vrijeme. Stoga ćemo pri analizi vremena izvođenja cijeli algoritam zavrtjeti unutar *for* petlje za 1000 mjerenja te iz dobivenih rezultata aproksimirati vrijeme potrebno da se obavi obrada jednog mjerenja. Računanje vremena prikazano je na isječku koda 3.28.

```
clock_t start, end;
double cpu_time_used;
start = clock();
/*
        Izvođenje algoritma...
*/
end = clock();
cpu_time_used = ((double)(end-start))/CLOCKS_PER_SEC;
printf("Proteko vrijeme za 1000 mjerenja za opciju %d:  %f\n", opcija, cpu_time_used);
```

Isječak koda 3.28 Izračun vremena za obavljanje 1000 mjerenja

U tablici 1 dani su rezultati u sekundama dobiveni izvedbom algoritma za sve tri opcije izračuna. Koristimo pet različitih skupova ulaznih podataka kako bi dobili što točniju srednju vrijednost.

Tablica 1 Rezultati vremena izvođenja za pet skopova od 1000 mjerenja

vrijednost u sekundama	**1. skup podataka**	**2. skup podataka**	**3. skup podataka**	**4. skup podataka**	**5. skup podataka**	**srednja vrijednost**
korištenjem 8 mjerenja	0,166483	0,168088	0,165842	0,165748	0,168541	0,166940
korištenjem 4 mjerenja	0,178918	0,178452	0,180509	0,178239	0,179518	0,179307
korištenjem 2 mjerenja	0,182518	0,184831	0,184433	0,184992	0,185096	0,184374

Vidimo da je za izvođenje 1000 obrada, za 32 ulazna kašnjenja po svakoj obradi, metodom izračuna korištenjem združenih 8 mjerenja potrebno u prosjeku 0,166940 sekundi. Kada taj broj podijelimo s 1000 kako bi dobili vrijeme potrebno za obradu jednog skupa, dobijemo da je za to potrebno 0,166940 milisekundi, odnosno 166,49 μs. Analogno, za metodu izračuna korištenjem 4 mjerenja potrebno je 179,307 μs za obradu jednog ulaznog skupa, a za metodu korištenjem 2 mjerenja 184,374 μs.

4 Zaključak

Raspberry Pi pokazao se kao malo računalo ogromnog potencijala. Njegovo korištenje bilo je sasvim intuitivno i lako. Oduševljenje je tim veće kada shvatim koliko je svih mogućih primjena već prepoznato i iskušano od strane korisnika širom svijeta: izgradnja „pametne kuće“ ili putnog računala za automobile; to su samo neki od mnoštva primjera. Za svrhu ovog završnog rada izgrađen je vrlo precizni anemometar koji brzinu i smjer vjetra određuje iz kašnjenja ultrazvučnog višeharmonijskog signala, no iako se cijeli sustav i pristup čini kompliciranim, Raspberry Pi sve odrađuje bez problema, ostavljajući dosta prostora za nadogradnju cijelog sustava, možda čak i izgradnju cijele meteorološke postaje s raznim senzorima. Razvijeni algoritam daje potpuno iste rezultate kao i referentni Matlab kod stoga smatram glavni zadatak ovog završnog rada uspješno ispunjenim. Nadam se da ću se ubuduće ponovno sresti s Raspberry Pi-em jer mi je ovaj rad bilo jedno ugodno i poučno iskustvo te mi je potaknulo želju da ostvarim neke vlastite ideje.

5 LITERATURA

[1] *Anemometar.m*, referentni Matlab kod. Davor Petrinović

[2] *Raspberry Pi*. Wikipedia, The Free Encyclopedia.

URL: https://en.wikipedia.org/wiki/Raspberry_Pi

[3] Sohil Patel. *THE BEST WAY TO CONNECT RASPBERRY PI TO LAPTOP DISPLAY*.

URL: http://diyhacking.com/connect-raspberry-pi-to-laptop-display/

[4] *How to Setup a Raspberry Pi Without a Monitor or Keyboard*.

URL: http://www.circuitbasics.com/raspberry-pi-basics-setup-without-monitor-keyboard-headless-mode/

[5] *History of the Anemometer*.

URL: http://www.logicenergy.com/articles/history-anemometer/

[6] URL: https://www.raspberrypi.org/downloads

[7] *SD Formatter for Windows Download.*

URL: https://www.sdcard.org/downloads/formatter_4/eula_windows/

[8] URL: https://sourceforge.net/projects/win32diskimager/

[9] *PuTTY download page*

URL: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html

[10] URL: http://filehippo.com/download_advanced_ip_scanner/

[11] *Download VNC® Viewer*

URL: https://www.realvnc.com/download/viewer/

[12] *25 zabavnih stvari koje možete činiti s Raspberry Pi*

URL: http://gadgetzona.com/25-zabavnih-stvari-koje-mozete-ciniti-s-raspberry-pi/

[13] *Raspberry Pi 3 Model B Technical Specifications*

URL: https://www.element14.com/community/docs/DOC-80899/l/raspberry-pi-3-model-b-technical-specifications

Izvedba algoritma za mjerenje brzine i smjera vjetra

Sažetak

Raspberry Pi malo je ali snažno ugradbeno računalo bazirano na Linux operacijskom sustavu. Idealno je za sve uzraste a mlađim generacijama omogućava da se kroz igru što prije susretnu s programskim jezicima. U okviru završnog rada bilo je potrebno implementirati algoritam za mjerenje brzine i smjera vjetra na ugradbenoj platformi Raspberry Pi a ulazne podatke čitati s mjernog uređajem preko serijskog USB sučelja. Razvijeni program potpuno je u skladu s referentnim programskim modelom u Matlabu. Obrađeni podatci spremaju se u datoteku i ostavljaju za analizu i prikaz za to predviđenom programu.

Ključne riječi: ultrazvučni anemometar, Raspberry Pi, ultrazvučni više-harmonijski signal, mjerenje brzine vjetra, mjerenje smjera vjetra

Implementation of an algorithm for wind speed and direction measurement

Abstract

Raspberry Pi is a small but powerfull embedded computer based on a Linux operating system. It is ideal for all ages and it enables younger generation to encounter programming languages as soon as possible through a game. As a part of the assignment it was required to implement an algorithm for wind speed and direction measurement on embedded platform Raspberry Pi and read input data from measuring device via serial USB interface. The developed program is fully consistent with the reference software model in Matlab. The processed data is stored in the file and leaft for analysis and display to a designated program.

Keywords: ultrasonic anemometer, Raspberry Pi, ultrasonic multi-harmonic signal, wind speed measurement, wind direction measurement