Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Oražem

Sistem za merjenje hitrosti z Dopplerjevim radarjem

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Načrtujte sistem za merjenje hitrosti na osnovi Dopplerjevega efekta. Za implementacijo sistema uporabite razvojni sistem STM32F4 z mikrokrmilnikom ARM Cortex-M4 ter Dopplerjev radarski modul K-MC1. Sistem naj bo povezan z osebnim računalnikom na katerem naj teče aplikacija za računanje in prikazovanje hitrosti.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Miha Oražem sem avtor diplomskega dela z naslovom:

Sistem za merjenje hitrosti z Dopplerjevim radarjem

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Patricia Bulića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 5. feburar 2017 Podpis avtorja:

Zahvaljujem se mentorju izr. prof. dr. Patriciu Buliću za strokovno pomoč pri izdelavi naloge.

Urški se zahvaljujem za pomoč in potrpežljivost tekom izdelave.

Posebno zahvalo pa bi naklonil staršem, bratom ter sestram za podporo med študijem.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GNU	GNU's Not Unix	GNU ni unix
FFT	Fast Fourier transform	hitra fouriereva transformacija
USB	Universal serial bus	univerzalno serijsko vodilo
MOC	Meta object compiler	prevajalnik, ki C++ razredom doda meta podatke
WAV	Waveform audio format	zapis zvočne datoteke
PCM	Pulse code modulation	modulacija zvočnega zapisa
RMS	Root mean square	kvadratna sredina
JTAG	Joint test action group	vmesnik za testiranje naprav in razhroščevanje
DAC	Digital to analog converter	digitalno analogni pretvornik
USB OTG	On The Go	način komunkacije z računalnikom
AHB	Advanced High-Preformance Bus	vodilo, ki povezuje periferne naprave in ostala vodila z procesorjem
APB	Advanced Peripheral Bus	vodilo na katerega so priklopljene periferne naprave
ADC	Analog to digital converter	analogno digitalni pretrovrnik
GPIO	General-purpose input/output	naprava, s katero lahko nadzorjuemo ali beremo logična stanja priklopov

Povzetek

Naslov: Sistem za merjenje hitrosti z Dopplerjevim radarjem

Namen diplomske naloge je razvoj sistema, ki s pomočjo Dopplerjevega radarja meri hitrost avtomobilov. Sistem je ustvarjen z razvojno ploščo STM32F4 Discovery in Dopplerjevim radarskim modulom RFBeam K-MC1. Pri tem je razvita še testna aplikacija, ki nam omogoča testiranje algoritma in parametrov, ki se uporabljajo za izračun hitrosti. Program za mikrokrmilnik je spisan v programskem jeziku C. Testna aplikacija je spisana v programskem jeziku C++ in za uporabniški vmesnik uporablja knjižnico QT. Diplomska naloga predstavi Dopplerjev pojav, na katerem temelji Dopplerjev radar. Predstavi tudi Fourierjevo transformacijo za obdelavo signala, dobljenega iz radarskega modula. Nadalje je opisan še razvoj testne aplikacije ter programa za mikrokrmilnik.

Ključne besede: Doppler radar, Dopplerjev pojav, Fouriereva transformacija, C, C++, QT, ARM.

Abstract

Title: System for measuring speed using a Doppler radar

The purpose of this thesis is to develop a system that measures the speed of passing cars using the Doppler radar. The system was created with an STM32F4 Discovery development board and the RFBeam K-MC1 Doppler radar module. An application, that enables testing of the algorithm and parameters, used for speed computation, has also been developed. The program for the microcontroller is written in the C programming language. The testing application is written in C++ programming language and uses the QT framework for the graphical user interface. The thesis explains the Doppler Effect, which is the foundation for the Doppler radar. It also explains Fourier transform used for analyzing the signal, acquired from the radar module. The last part of the thesis describes the process of developing the test application and the program for the microcontroller.
Keywords: Doppler radar, Doppler effect, Fourier transform, C, C++, QT, ARM.

Poglavje 1

Uvod

Dopplerjev radar se uporablja za merjenje hitrosti objektov. V naprednejši obliki ga lahko uporabljamo tudi za merjenje razdalje od izhodiščne točke do objektov. V osnovi temelji na Dopplerjevem pojavu, ki ga je opisal avstrijski fizik Chistian Doppler, leta 1842. O Dopplerjevih radarjih je bilo napisano že veliko in je celotna tehnologija že dobro poznana.

Dopplerjev radar se uporablja v veliko različnih aplikacijah. V vojski se uporablja za zaznavanje sovražnih objektov. Prvič so ga uporabili v drugi svetovni vojni v letalstvu. Radar jim je omogočil, da so zaznavali letala ali ladje tudi ponoči. Dandanes se uporablja tudi kot pomoč pri pristajanju letal na letalonosilke. S konstantnim merjenjem oddaljenosti, hitrosti objekta in upoštevanjem svoje hitrosti lahko sistemi na letalu natančno izračunajo, s kakšno hitrostjo mora letalo leteti, da bo lahko pristalo. Uporablja se tudi v meteorologiji za napoved vremena. Po svetu najdemo ogromne radarske sisteme, s katerimi lahko natančno napovejo, kakšno vreme lahko pričakujemo v prihodnjih dneh. Policisti ga uporabljajo za merjenje hitrosti mimoidočih avtomobilov.

V začetku razvoja tehnologije je bilo zaznavanje predmetov realizirano z analognimi vezji, ki so skrbela za pridobivanje podatkov. Ta sistem je bil velik in zelo težak, zato je bila njegova uporaba omejena. Kasneje so se razvili mikroprocesorji, ki so lahko računali Fourierevo transformacijo. Zaradi njih

Slika 1.1: Opozorilna tabla [25]

se je tehnologija močno razvila in se na enak način uporablja še danes [23].

Cilj diplomske naloge je predstavitev razvoja sistema, ki meri hitrost objektov z uporabo Dopplerjevega radarja. V diplomski nalogi sem opisal vse potrebno za razumevanje delovanja. Razvil sem tudi program, s katerim lahko testiramo in prilagajamo izračun hitrosti. Na koncu sem sistem realiziral še na mikrokrmilniku, da je sistem sploh uporaben.

Sistem je bil razvit kot merilni modul za table, ki prikazujejo našo hitrost in nas opozarjajo, če se vozimo prehitro. Table prikazujejo tudi opozorilno sporočilo, ki nam pove, če je v bližini šola ali klinični center, zaradi česar moramo še posebej paziti na ljudi in promet [25]. Primer table lahko vidimo na sliki 1.1.

V začetku diplomske naloge sem opisal teorijo Dopplerjevega pojava in radarja. Nato sem opisal Fourierevo transformacijo v poglavju 3. S tema dvema poglavjema se opis teorije zaključi in nadaljujemo z opisom uporabljenih orodji v poglavju 4. Nato predstavim in opišem testni program, ki sem ga uporabljal za testiranje parametrov in celotne implementacije v poglavju 5. Na koncu pa je predstavitev, kako deluje program za mikrokrmilnik v poglavju 6 in sklepne ugotovitve.

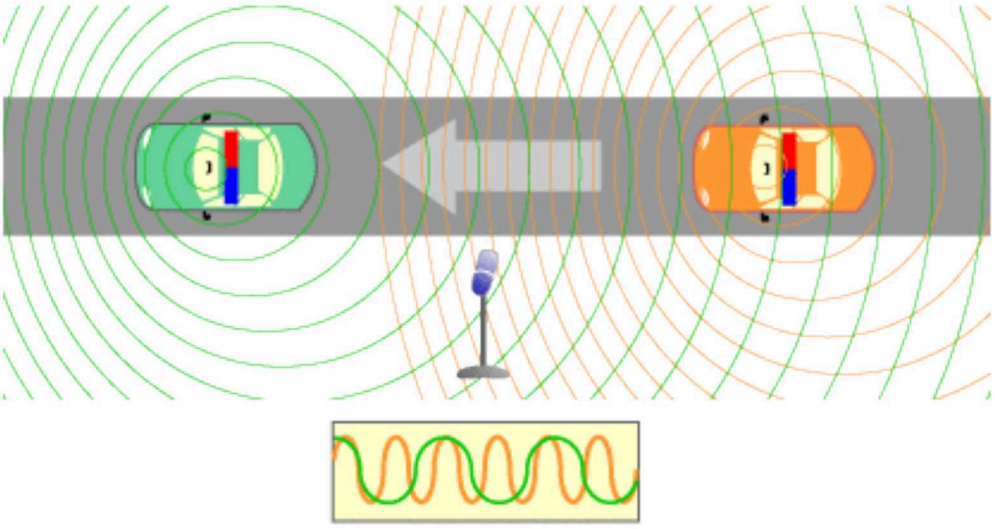
Poglavje 2

Merjenje hitrosti

Za izdelavo diplomske naloge sem uporabil Dopplerjev radar, ki temelji na Dopplerjevem pojavu. Za razumevanje delovanja Dopplerjevega radarja je naprej treba razumeti Dopplerjev pojav.

2.1 Dopplerjev pojav

Dopplerjev pojav je poimenovan po avstrijskem matematiku in fiziku Christianu Andreasu Dopplerju. Pojav lahko opišemo z naslednjo razlago. Policijski avtomobil z vklopljeno policijsko sireno se vozi po ulici. Mi, kot sprejemnik zvoka sirene, se nahajamo na sredini ulice. Ko se nam policijski avtomobil približuje, slišimo policijsko sireno z višjo frekvenco zvoka kot je v resnici. Takrat, ko je policijski avtomobil točno pred nami, slišimo zvok z isto frekvenco, kot jo oddaja policijska sirena. Ko pa se policijski avtomobil od nas oddaljuje, pa slišimo policijsko sireno z nižjo frekvenco zvoka. Frekvenca zvoka sirene se med premikanjem avtomobila nikoli ne spremeni, vendar sprejemnik zaradi Dopplerjevega pojava zazna različne frekvence zvoka. Do enakega pojava pride tudi v primeru, da je policijski avtomobil z vklopljeno sireno stacionaren in se sprejemnik pelje mimo policijskega avtomobila. Isti pojav lahko vidimo, če opazujemo raco, ki se s konstanto hitrostjo premika po gladini jezera, razlika med dolžino valov, ki se ustvarjajo, je za raco večja.



Slika 2.1: Primer Dopplejevega pojava. Na sliki lahko vidimo, da sprejemnik (mikrofon) sprejme zvok z višjo frekvenco, če se avtomobil pelje proti njemu. To je predstavljeno z oražnimi valovi. Medtem ko je zvok, ki ga oddaja zeleni avtomobil z zelenimi valovi sprejet z nižjo frekvenco [11].

Medtem ko je razlika med dolžino valov pred raco manjša. To velja za vse vrste valovanj.

Pojav, v primeru stacionarnega oddajnika valovanja, lahko opišemo z naslednjo izpeljano enačbo [9]:

$$f = f_0(1 \pm \frac{v}{c}) \tag{2.1}$$

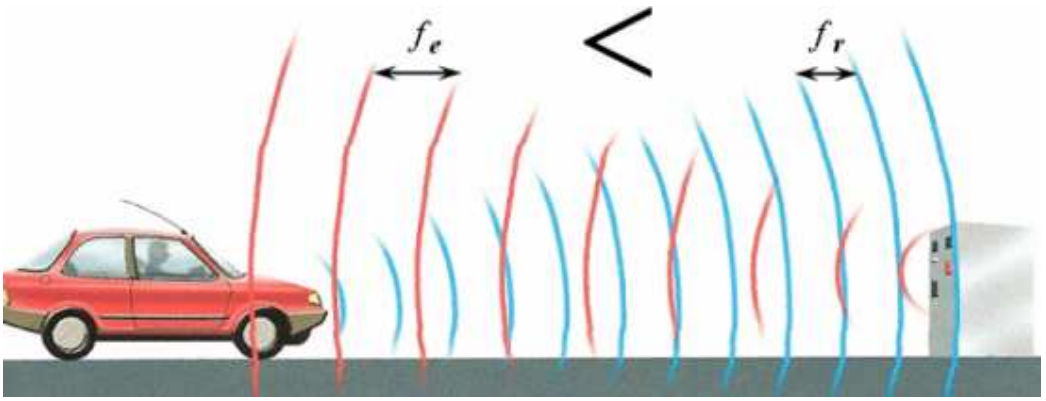
V nasprotnem primeru pojava, ko se oddajnik premika in je sprejemnik stacionaren, je izpeljana enačba naslednja [9]:

$$f = \frac{f_0}{(1 \pm \frac{v}{c})} \tag{2.2}$$

Z enačbo (2.1) in (2.2) lahko izračunamo sprejeto frekvenco. Sprejeto frekvenco izračunamo iz frekvence zvoka f_0, ki jo oddajnik oddaja, hitrosti premikajočega sprejemnika v in hitrostjo zvoka c [5].

2.2 Dopplerjev radar

Dopplerjev radar uporablja Dopplerjev pojav za zajemanje hitrosti o oddaljenih objektih. Objekt, za katerega nas zanima hitrost, se mora premikati, saj nam Dopplerjev pojav narekuje, da je frekvenca valovanja mirujočega objekta nespremenjena za mirujočega sprejemnika. Zajemanje hitrosti poteka tako, da Dopplerjev radar odda mikroval, ki se odbije od opazovanega objekta. Odbiti mikroval ima drugačno frekvenco kot oddani mikroval. Sprememba frekvence je odvisna od tega ali se merjeni objekt približuje ali oddaljuje. Ko se odbiti mikroval vrne do radarja, lahko z vgrajenim sprejemnikom zaznamo signal. Nato analiziramo sprejet signal in izračunamo hitrost. Sprejeta frekvenca je poleg hitrosti opazovanega objekta odvisna tudi od kota med objektom in radarjem. Za natančno meritev in analizo je potrebno opraviti več opisanih meritev. V mojem primeru sem opravljal analizo na mikrokrmilniku in računalniku na približno 20 milisekund.



Slika 2.2: Primer Dopplejevega radarja. Na sliki vidimo, da je frekvenca f_e, ki jo radar odda, manjša kot frekvenca f_r, ki se odbije od avtomobila [12].

Na sliki 2.2 vidimo delovanje Dopplerjevega radarja. Zaradi premikanja avtomobila je sprejeta frekvenca f_r večja od oddane f_e. Razlika med oddano in sprejeto frekvenco je odvisna od hitrosti avtomobila.

2.2.1 Izračun hitrosti

Hitrost premikajočega predmeta lahko izračunamo iz naslednje enačbe:

$$v = \frac{\Delta f c}{2f} \text{ [10]} \tag{2.3}$$

Hitrost se izračuna po enačbi (2.3) z uporabo oddane frekvence f, hitrosti svetlobe c in razlike med sprejeto in oddano frekvenco Δf [6].

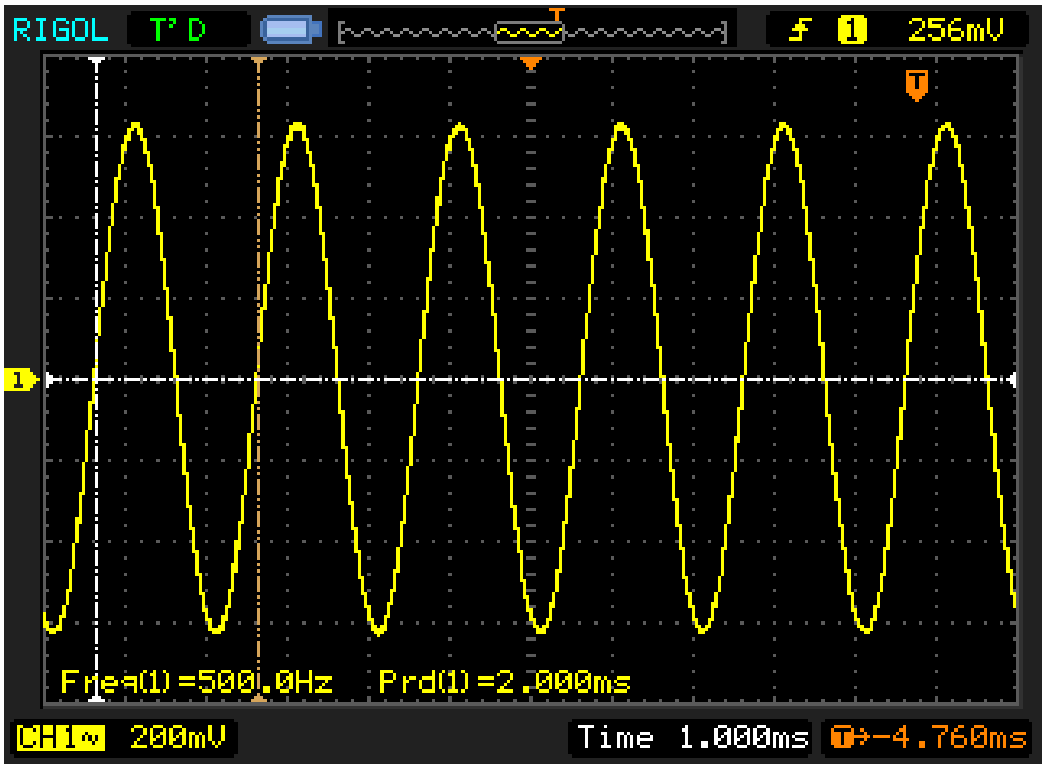
Poglavje 3

Fouriereva transformacija

Za izračun hitrosti sem moral opravljati Fourierevo transformacijo na pridobljenem signalu iz katere sem kasneje izračunal hitrost. Fouriereva transformacija je proces spremembe signala iz časovne domene v frekvenčno domeno. Vzame podan signal v časovni domeni in izračuna amplitudo frekvenčnih komponent [8].

Na sliki 3.1 je podan primer signala v časovni domeni, ki sem ga zajel s pomočjo osciloskopa. Slika predstavlja dvodimenzionalni graf. Horizontalna os nam predstavlja čas, vodoravna pa amplitudo ali moč signala. Z uporabo računalnika sem prek zvočne kartice ustvarjal sinusni signal s frekvenco 500 Hz. V tem primeru je Fouriereva transformacija preprosta, saj imamo samo eno frekvenco, ki jo mora najti.

Na sliki 3.2 sem na osciloskopu prikazal Fourierevo transformacijo. Slika je prav tako dvodimenzionalni graf. Horizontalna os nam predstavlja najdene frekvence, vodoravna pa amplitudo najdene frekvence. Večkrat ko se frekvenca pojavi v zajetem signalu, večja bo njena amplituda. V našem primeru je v zajetem signalu samo ena frekvenca, zato se tudi po transformaciji pojavlja samo ena.



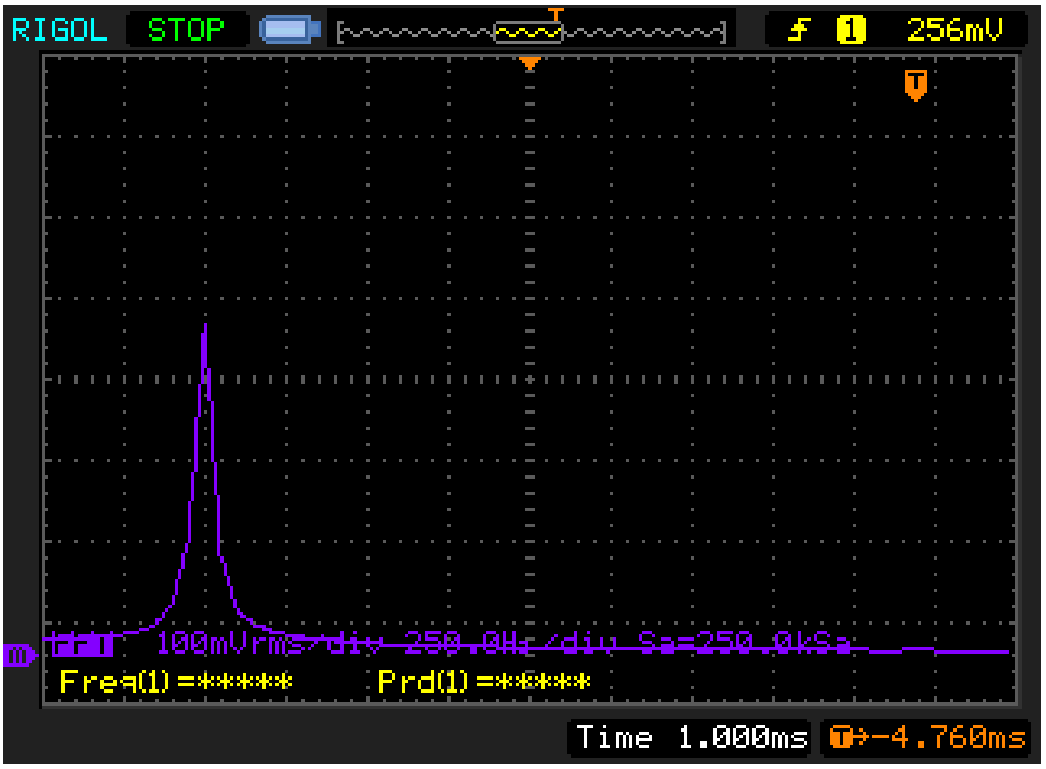
Slika 3.1: Primer signala v časovni domeni

3.1 Diskretna Fouriereva transformacija

Diskretna Fouriereva transformacija pretvori zajet signal v vektor amplitud frekvenčnih komponent. Rezultat je vektor, ki je urejen po frekvencah. Diskretna oblika Fouriereve transformacije je preprosta za izračun, vendar je časovno potratna. Zato se v praski uporablja algoritem FFT (Fast Fourier Transform), s katerim hitreje izračunamo rezultat. Algoritem FFT je učinkovit način izračuna Diskretne Fouriereve transformacije.

3.1.1 Hitra Fouriereva transformacija

Hitra Fouriereva transformacija (FFT) je učinkovit algoritem za računanje diskretne Fouriereve transformacije. Glavna prednost je hitrejše računanje. To je predvsem pomembno, če želimo računati FFT na mikrokrmilniku, saj



Slika 3.2: Primer signala 3.1 v frekvenčni domeni

je sistem v celoti velikokrat šibkejši od računalnikov, ki jih uporabljamo. Diskretna Fourierjeva transformacija za transformacijo N vzorcev izvede $O(N^2)$ operacij, medtem ko FFT potrebuje $O(NlogN)$ operacij [14].

3.1.2 Podatki

Podatki za računanje diskretne Fouriereve transformacije morajo biti podani kot vektor vzorcev, ki opisujejo signal. Ko pripravljamo podatke, moramo paziti, da so podatki vzorčeni s konstanto frekvenco. Če želimo opisati signal s 1024 vzorci in ga vzorčimo s frekvenco 44100 Hz, mora biti vsak vzorec pridobljen na 22 mikrosekund. Velikost vzorca in frekvenca vzorčenja se spreminja glede na to, kakšen signal poskušamo vzorčiti in kaj poskušamo s transformacijo doseči.

3.1.3 Opis računanja

Pri diskretni Fourierevi transformaciji računamo vsako frekvenco, ki se lahko pojavlja v signalu posebej. Rezultat izračuna ene frekvence je amplituda frekvence v vhodnem signalu, ki je predstavljeno kot kompleksno število, ki predstavlja magnitudo in fazni zamik. Če imamo vhodni signal dolg N vzorcev, bomo izračunali $N - 1$ frekvenc. Torej, če imamo vektor velik 100 vzorcev, bomo računali frekvence od 0 do 99 enot. Enote so odvisne od hitrosti vzorčenja vhodnega signala in za izračun niso pomembne, upoštevati pa jih moramo pri interpretaciji rezultata transformacije. Vsak izračun je sestavljen iz vsote korelacije trenutne računane frekvence z vsakim vzorcem iz vhodnega signala. Korelacija je število, ki opisuje povezanost med dvema spremenljivkama. Torej, če neke frekvence v signalu ni, bo tudi rezultat transformacije za to frekvenco nič. Če pa se neka frekvenca pojavlja večkrat v signalu, bo pa rezultat transformacije skladen s pojavom frekvence.

Diskretno Fourierevo transformacijo je najlažje razložiti s formulo, ki se uporablja za izračun.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \text{ [7]} \tag{3.1}$$

Enačba 3.1 predstavlja izračun magnitude in faznega zamika za frekvenco k. Za izračun moramo sešteti vse korelacije na intervalu $[0, N - 1]$, kjer x_n predstavlja n-ti vzorec signala. Rezultat celotne transformacije je vektor kompleksnih števil, ki opisujejo vhodni signal v frekvenčnem prostoru. Indeks v vektorju predstavlja, za katero frekvenco gre [13].

Poglavje 4

Uporabljena orodja

Namizno aplikacijo za testiranje algoritma (poglavje 5) in program za mikrokrmilnik (poglavje 6) sem razvijal na operacijskem sistemu Linux. Vsa orodja, ki sem jih uporabljal, so odprtokodna in delujejo na vseh glavnih operacijskih sistemih.

4.1 Programski jezik C++

Programski jezik C++ je razvil Bjarne Stroustrup. Razvoj se je začel leta 1979 in od takrat se je jezik razvil v enega najbolj priljubljenih jezikov. Jezik je zelo priljubljen za razvoj iger ter sistemov, ki potrebujejo nizkonivojski dostop do spomina, saj nam jezik omogoča ročno uporabljanje s pomnilnikom. To je zelo uporabna lastnost, saj nam omogoča, da naredimo sisteme, pri katerih je zelo pomembno, da se izvajajo v realnem času. Programski jezik C++ lahko uporabimo tudi za razvijanje programov za mikrokrmilnik [1]. Glavne lastnosti jezika so:

- Objekti
- Polimorfna preobložitev metod
- Generične metode in objekti
- Preobložitev operatorjev

- Ročno upravljanje s pomnilnikom

4.2 CMake

Za izdelavo namizne aplikacije sem potreboval sistem za generacijo datotek za prevajanje projekta. Odločil sem se, da bom uporabil odprtokodni sistem CMake. Glavna prednost sistema je generacija prevajalnih datotek za veliko razvojnih okolij in prevajalnih sistemov. Uporaba sistema je preprosta. Najprej napišemo pravila, ki jih sistem uporabi za generacijo datotek. Nato si izberemo želen sistem, ki ga bomo uporabili za prevajanje projekta. V mojem primeru je sistem ustvaril datoteke, ki jih uporablja sistem Make [2].

4.3 Knjižnica QT

Knjižnico QT sem uporabil za prikaz grafičnega vmesnika pri namizni aplikaciji. Knjižnica je napisana v programskem jeziku C++ in deluje na vseh glavnih operacijskih sistemih. Grafični vmesnik ima na vsaki podprti platformi nativen izgled. Poleg grafičnega vmesnika lahko QT uporabimo še za druge stvari, saj je zelo obsežna in vsebuje veliko stvari, ki transparentno delujejo na vseh podprtih platformah [3].

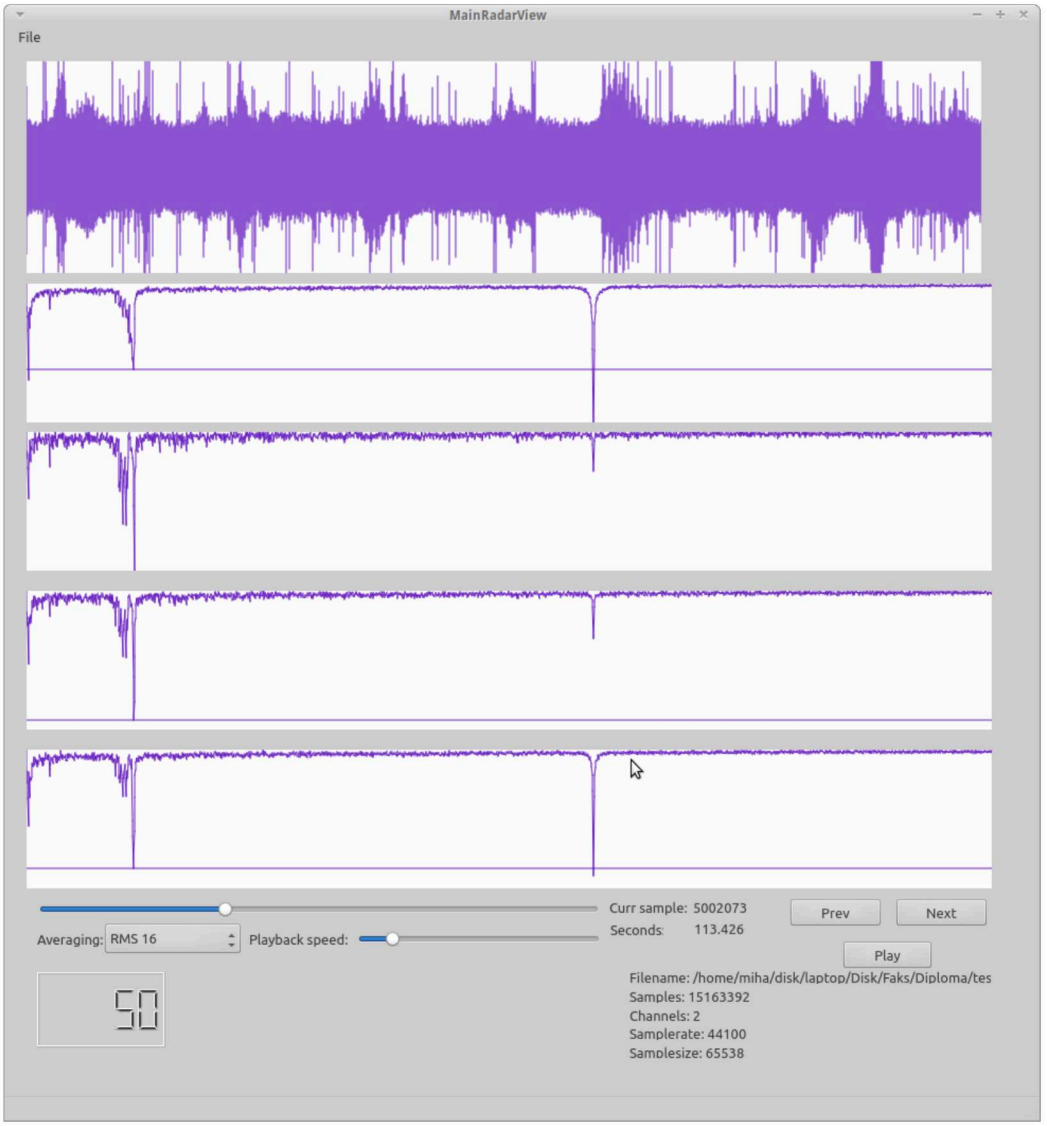
4.4 Prevajalnik C in C++ kode

Pri izdelavi namizne aplikacije in programa za mikrokrmilnik sem uporabil odprtokodni prevajalnik GCC. Prevajalnik je glavni del GNU (GNU's Not Unix) orodij za razvijanje programov in operacijskih sistemov. Uporablja se kot glavni prevajalnik za veliko odprtokodnih projektov. Med njimi najdemo tudi Linux jedro [4].

Poglavje 5

Testni program

Najprej sem za diplomsko nalogo naredil testni program, s katerim sem testiral algoritem in različne implementacije Fouriereve transformacije. S tem sem dosegel hitrejši razvoj celotnega sistema za merjenje hitrosti. Dopplerjev radar, s katerim sem meril hitrost, je prišel z referenčnim vezjem, ki nam omogoča priklop na računalnik preko USB (Universal serial bus) vodila. Ko napravo priklopimo v računalnik, se pokaže kot naprava za zajemanje zvoka. To sem izkoristil za snemanje zvočnih datotek, na katerih sem kasneje izvajal FFT. To mi je zelo olajšalo razvoj diplomske, ker sem posnel nekaj primerov merjenja hitrosti in to kasneje uporabljal za testiranje. Uporaba programa je preprosta. Najprej izberemo, katero zvočno datoteko bomo uporabili za računanje Fouriereve transformacije. Nato lahko to zvočno datoteko predvajamo in opazujemo več grafov, ki prikazujejo trenutne vrednosti FFT. Poleg tega pa prikaže, katera frekvenca je najbolj izpostavljena in iz te izračuna hitrost, ki se potem prikaže na uporabniškem vmesniku.

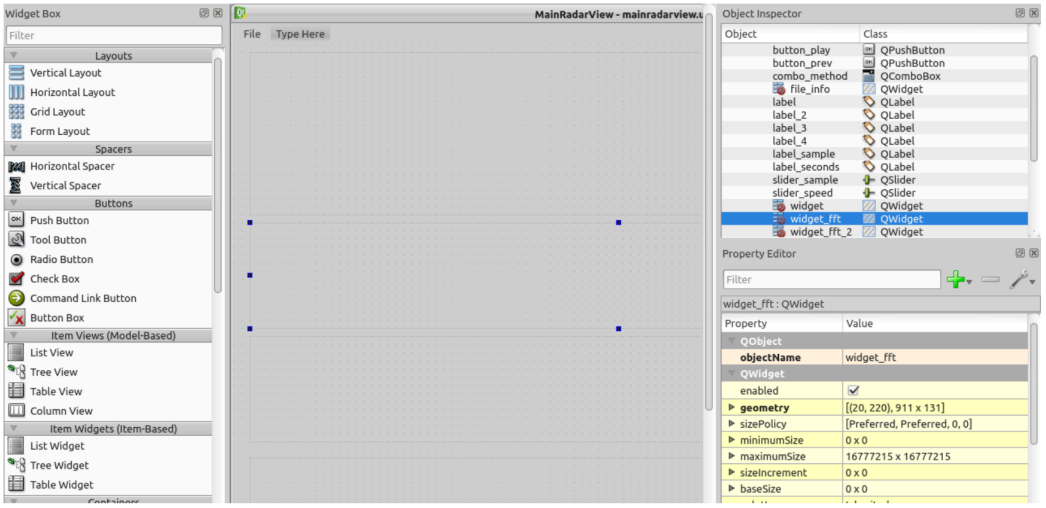


Slika 5.1: Testni program

5.1 Uporabniški vmesnik

Uporabniški vmesnik (slika 5.1) sem izdelal z uporabo programa QT Designer. Delo s programom je preprosto. Izberemo si, kateri gradnik želimo v našem uporabniškem vmesniku in ga postavimo na želeno mesto. Temu

gradniku nastavimo ime, ki ga nato uporabljamo v kodi za upravljanje. Uporabljamo lahko tudi bolj napredne konstrukte, ki sami prilagajajo pozicijo in velikost gradnikov glede na velikost celotnega okna. To je uporabno v primeru, ko imamo veliko gradnikov na enem oknu in bi bilo prilagajanje pozicije in velikosti preveč zamudno.



Slika 5.2: QT Designer

Na sliki 5.2 lahko vidimo primer grajenja uporabniškega vmesnika. Na levi strani imamo na izbiro, kateri gradnik lahko uporabimo v našem uporabniškem vmesniku. Na sredini lahko pregledujemo, katere elemente že imamo na oknu. Lahko jim nastavljamo pozicijo in velikost. Na desni strani pa lahko vsem gradnikom nastavljamo bolj podrobne nastavitve. Na primer pri gradniku, ki izpisuje tekst, lahko nastavimo, katero pisavo želimo in njeno velikost.

Ko naredimo uporabniški vmesnik, ga lahko uporabimo v našem programu. Ker naš program ne more samodejno vedeti, kako zgleda naš uporabniški vmesnik, mu je to treba najprej povedati. Proces za to je pogojen s programskim jezikom, ki ga uporabljamo. V mojem primeru je to C++. Naprej je treba datoteko, v kateri je shranjen opis uporabniškega vmesnika, pretvoriti v ".h"in ".cpp"datoteki, ki vsebujeta definicije za upravljanje z

gradniki. Za to uporabimo predprocesor MOC (Meta object compiler), ki to naredi za nas. Predprocesor je vključen poleg knjižnice. Nato dodamo prevedene datoteke v naš projekt.

Izvorna koda 5.1: Primer razreda

```
class MainRadarView : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainRadarView(QWidget *parent = 0);
    ~MainRadarView();
public slots:
    void openFileDialog();
    void drawPrev();
    void drawNext();
    void play();
    void sliderMoved();
    void speedSliderMoved();
    void methodChanged(int index);
private:
    std::unique_ptr<Ui::MainRadarView> ui;
    std::unique_ptr<WavefileInfo> file_info;
    std::unique_ptr<FFTDrawWidget> fft_widget;
    std::unique_ptr<FFTDrawWidget> fft_widget_2;
    std::unique_ptr<FFTDrawWidget> fft_widget_3;
    std::unique_ptr<FFTDrawWidget> fft_widget_4;
    std::unique_ptr<QTimer> timer;
    int current_sample;
    float speed_factor;
    SoundFile file;
    bool play_state;
```

```
    void updateGUIState();
    void calculateAndUpdate(int sample);
protected:
    virtual void keyPressEvent(QKeyEvent *keyEvent) override;

    std::shared_ptr<WaveformWidget> waveform;
};
```

V projektu moramo ustvariti razred, ki predstavlja okno, za katerega smo ustvarili uporabniški vmesnik. Nato v temu razredu vključimo prevedene datoteke, ki predstavljajo uporabniški vmesnik. Sedaj lahko uporabljamo vse gradnike, ki smo jih definirali.

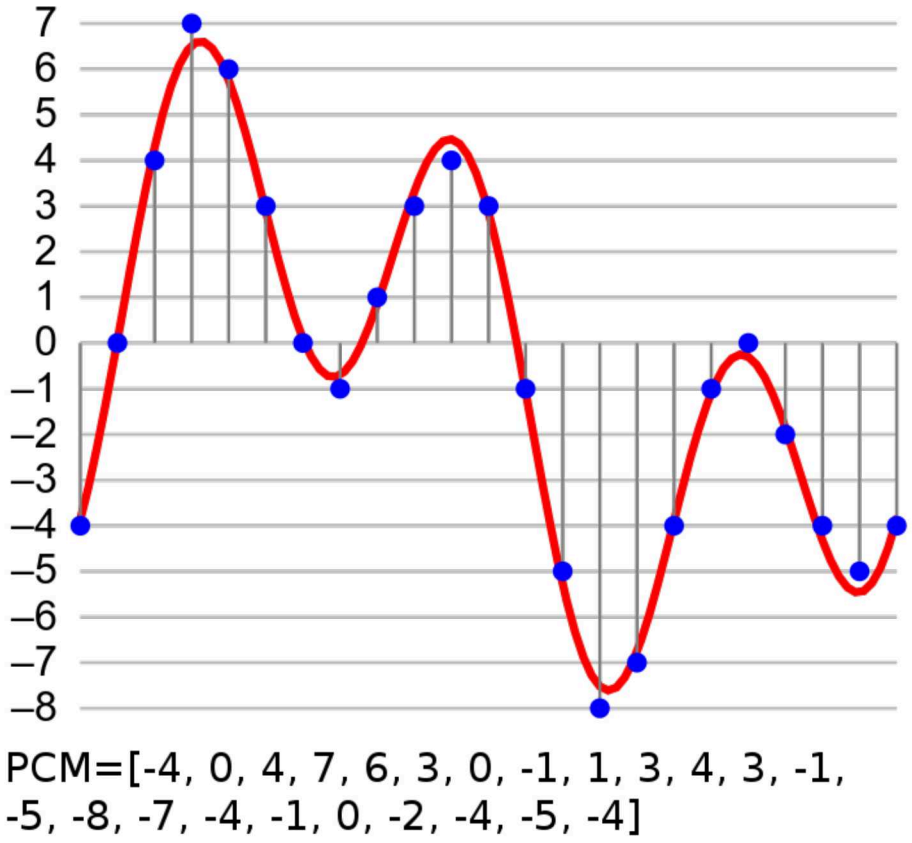
5.1.1 Signali in reže

Knjižnica QT podpira definiranje signalov in rež, ki se uporabljajo za komunikacijo med različnimi razredi. To nam zelo olajša razvoj uporabniških vmesnikov, saj ti po naravi temeljijo na dogodkih. Dogodek je akcija, ki jo sproži uporabnik, na primer klik na gumb, premik miške, premikanje okna itd. Signali in reže pomagajo pri pisanju lepše in čistejše kode, vendar je zaradi njih potrebna uporaba predprocesorja MOC (ki je tako ali tako nujen za uporabo QT knjižnice). V razredu definiramo režo, na katero lahko kasneje vežemo signale. Ko se signal sproži, se bo v razredu poklicala reža (metoda), ki je na ta signal vezana. Prav tako lahko v razredu definiramo signale, ki jih potem prožimo.

5.2 Zvočne datoteke

V testnem programu lahko prebiramo zvočne datoteke tipa WAV (Waveform audio format). Za ta format sem se odločil, ker je preprost in ker obstaja veliko knjižnic za branje podatkov. Datoteka v WAV formatu vsebuje zvočni signal, zapisan v PCM (Pulse code modulation) shemi. Ta shema je najbolj

preprost način za zapis analognega signala v digitalni obliki. Zvočni signal je zapisan kot vektor realnih števil, ki predstavljajo analogno vrednost. Shema predstavlja samo podatke, medtem ko parametri v formatu narekujejo, kako interpretirati podatke. Posamezni vzorci so lahko 8 ali 16 bitni, odvisno od tega, kako natančno želimo reproducirati zvok. Pomemben podatek je tudi frekvenca vzorčenja zvoka. Večja kot je frekvenca, boljša je kvaliteta zvoka. Datoteka ima lahko več kanalov, najpogosteje ima dva kanala (stereo) ali en kanal(mono). Z večanjem velikosti vzorcev in frekvenco vzorčenja narašča tudi velikost datoteke. Podatki zapisani v PCM shemi niso zgoščeni, zato je reprodukcija zvoka bolj natančna kot pri ostalih formatih, ki so zgoščeni z izgubo podatkov (npr MP3).



Slika 5.3: Primer PCM zvočnega zapisa [21]

5.3 Prebiranje podatkov

Za prebiranje podatkov iz WAV datoteke sem uporabil knjižnico "libsndfile" [15]. Knjižnica je napisana v C programskem jeziku in deluje na vseh operacijskih sistemih. Ker je zvok lahko zapisan v WAV formatu v različnih konfiguracijah, knjižnica poskrbi za pretvarjanje podatkov in nam jih predstavi v enotni obliki ne glede na konfiguracijo prebrane datoteke. Za prebiranje podatkov moramo najprej določiti, za katero datoteko gre.

Izvorna koda 5.2: Prebiranje zvočne datoteke

```
bool SoundFile::open(std::string filename) {
    handle = sf_open(filename.c_str(), SFM_READ, &file_info);
    return (handle != nullptr);
}
```

To storimo s funkcijo *sf_open*, ki sprejme pot do datoteke, način odprtja datoteke in kazalec do strukture *SF_INFO*, ki vsebuje vse informacije o datoteki. Če se funkcija izvede neuspešno, nam vrne NULL, v nasprotnem primeru pa nam vrne kazalec do datoteke, ki ga kasneje uporabljamo za prebiranje podatkov.

Ko želimo prebrati podatke, to lahko storimo na dva načina, uporabimo lahko funkcjo *sf_read_x* ali *sf_readf_x*, kjer je x tip podatka, ki ga želimo prebrati (*short*, *int*, *float*). V prvem primeru preberemo točno toliko podatkov kot želimo, vendar je treba upoštevati, da ti podatki lahko vsebujejo vzorce iz več kanalov. V drugem primeru pa določimo, koliko podatkov želimo. Rezultat je vektor vzorcev, ki vsebuje vzorce za vse kanale.

5.4 Računanje hitrosti

Ko program "predvaja"datoteko, iz nje bere po 1024 vzorcev. Te vzorce nato pretvorimo iz časovne domene v frekvenčno z uporabo FFT algoritma. Za pretvorbo uporabljam knjižnico KissFFT [16]. Za to knjižnico sem se odločil, ker je napisana samo v jeziku C. To je bila v mojem primeru prednost, saj sem lahko isto knjižnico uporabil tudi na mikrokrmilniku, kar ne bi mogel, če bi bil kak del knjižnice spisan v strojnem jeziku. S tem sem dosegel konsistentne rezultate. Ko je transformacije opravljena, odstranim imagnirani del rezultata in realni del shranim v nov vektor. Realni vektor pretvorim v absolutnega, da lahko poiščem, katera vrednost je največja.

$$f = \frac{if_s}{N} \tag{5.1}$$

Za pridobljeno vrednost poiščem indeks v vektorju. Iz indeksa nato izračunamo frekvenco z uporabo enačbe (5.1), kjer je i indeks v vektorju, f_s predstavlja frekvenco vzorčenja in N dolžino vzorca. Nato z uporabo enačbe (2.3) izračunamo hitrost. Izračunana hitrost je v metrih na sekundo, kar je treba še pretvoriti v kilometre na uro. Izračunana in pretvorjena hitrost se nato prikaže na uporabniškem vmesniku.

5.5 Obdelava frekvenčne domene

Rezultat frekvenčne domene lahko poljubno obdelamo. Eksperimentalno sem dodal "glajenje"rezultata v frekvenčni domeni z uporabo RMS (Root mean square) [17]. To povzroči, da se spremembe rezultata dogajajo bolj počasi in celotni rezultat postane bolj stabilen.

$$X_{rms} = \sqrt{\frac{1}{n}(x_1^2 + x_2^2 + \cdots + x_n^2)}[17] \tag{5.2}$$

Obdelava se izračuna z uporabo enačbe (5.2). Za izračun uporabimo prejšnje shranjene vzorce FFT analize. Koliko jih uporabimo, je odvisno od stopnje obdelave n. V programu lahko izbiramo med stopnjami 2, 4, 8 in 16.

Slika 5.4: Primerjava med RMS 16 in RMS 2 obdelavo

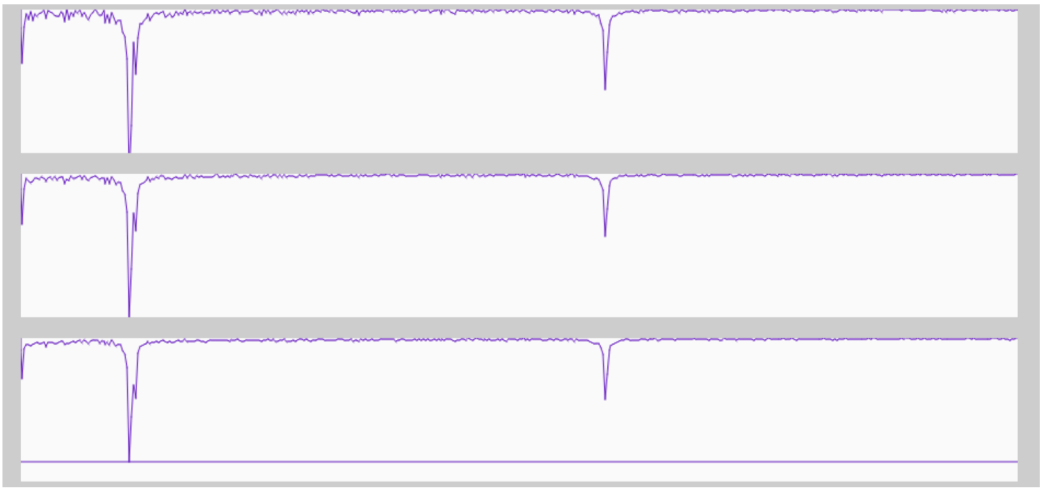
Na sliki 5.4 se vidi razlika med RMS 16 (zgornja) in RMS 2 (spodnja) obdelave. V primeru RMS 16 je graf veliko bolj čist. Ker gre v osnovi za računanje povprečja, se prisotnost frekvence zmanjša, ampak ker obdelujemo celoten graf, se zaznava hitrosti ne spremeni.

5.6 Uporaba

Naprej je treba določiti, iz katere datoteke bomo prebirali zvočni zapis. To storimo tako, da gremo na meni "File"in "Open". Nato se nam na zgornjem grafu prikaže celotna zvočna datoteka v časovni domeni.

Slika 5.5: Zvočna datoteka v časovni domeni

Na naslednjih štirih grafih lahko opazujemo FFT graf trenutnega vzorca. Razlika med posameznimi grafi je, da imajo nekateri nastavljeno glajenje signala.



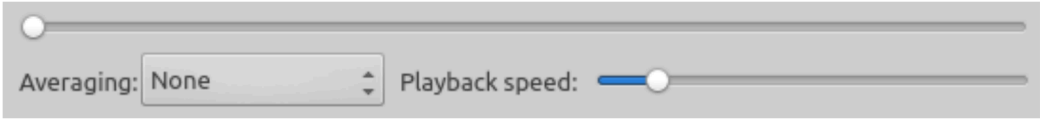
Slika 5.6: Grafi, ki prikazujejo FFT

Na prvem FFT grafu glajenja ni. Na naslednjih pa so RMS 2, RMS 4, RMS 8 v tem vrstnem redu. Glajenje signala pomaga pri iskanju hitrosti,

saj se graf spreminja počasneje. Na prvem grafu lahko nastavljamo, katero glajenje želimo uporabiti.

Slika 5.7: Gumbi za upravljanje predvajanja

Pod grafi imamo še gumbe, s katerimi lahko začnemo/ustavimo predvajanje in se premikamo naprej/nazaj po zvočni datoteki.



Slika 5.8: Drsniki

Na voljo imamo še dva drsnika. Prvi prikazuje čas predvajanja. S premikanjem drsnika nastavljamo čas v datoteki. Drugi drsnik pa upravlja s hitrostjo predvajanja datoteke.

Slika 5.9: Prikazovalnik hitrosti

V spodnjem levem kotu imamo prikazovalnik hitrosti. Hitrost je prikazana v kilometrih na uro. Podatek pa se spremeni vsakič, ko se spremeni obdelovani vzorec.

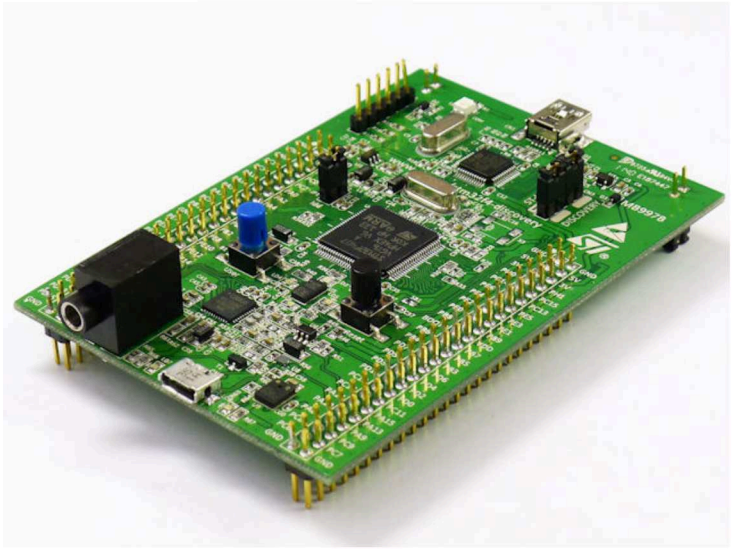
Poglavje 6

Mikrokrmilnik

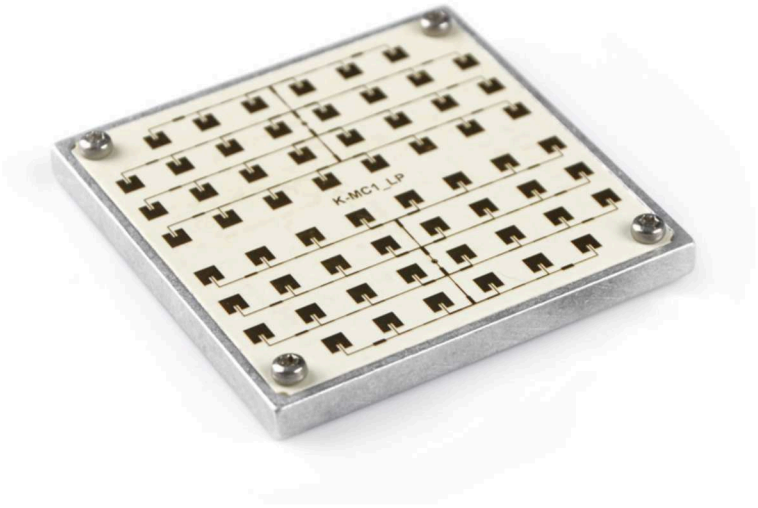
Za izdelavo diplomske naloge sem uporabil STM32F4 DISCOVERY razvojno ploščo. Razvojna plošča je namenjena za predstavitev zmogljivosti STM32F407 mikroprocesorja, ki deluje s frekvenco 168 MHz, vsebuje enoto za računanje števil s plavajočo vejico in 192 KB rama. Mikroprocesor temelji na ARM Cortex M4 jedru in je še dodatno razširjen s strojnimi ukazi za digitalno procesiranje signalov [18]. Na plošči najdemo še:

- merilnik pospeškov
- vgrajen programator in JTAG (Joint test action group) vmesnik
- DAC (Digital to analog converter) z vgrajenim ojačevalnikom
- Mikrofon
- 8 svetlobnih diod
- 2 gumba
- USB OTG (On The Go)

Za merjene hitrosti sem uporabil RFBeamov K-MC1 Dopplerjev radarski modul. Modul ima vgrajene ojačevalnike na izhodih, kar ga naredi primernega za uporabo z mikrokrmilniki brez uporabe dodatnih vezij [19].



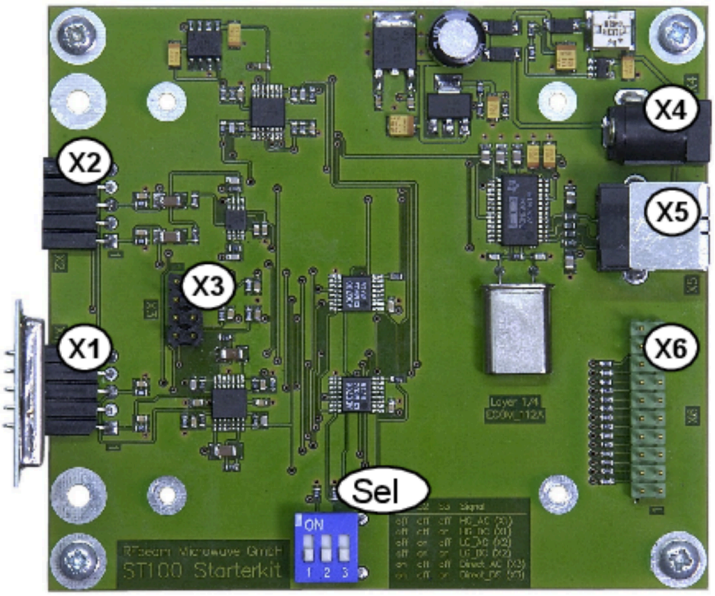
Slika 6.1: STM32F4 DISCOVERY razvojna plošča [20]



Slika 6.2: Radarski modul [22]

Za upravljanje z napravami na mikrokrmilniku pa sem uporabljal knjižnico od STMicroelectronics za ta določen mikroprocesor.

6.1 Povezava radarskega modula z mikrokrmilnikom



Slika 6.3: Referenčno vezje RFBeam ST100 za K-MC1 radarski modul [24]

Naprave sem povezal po naslednji shemi:

Priključek na mikrokrmilniku	**Prikluček na referenčnem vezju X6 (slika 6.3)**
GND	GND
PC2	X2_HI_I

6.2 Inicializacija periferije

Pri mikroprocesorjih, ki temeljilo na ARM jedrih, moramo periferne naprave naprej vklopiti. Ob vklopu ali ponovnem zagonu procesorja so vse periferne naprave izklopljene zaradi porabe energije. Mikroprocesor z perifernimi napravami komunicira preko AHB (Advanced High-Preformance Bus). To vodilo se razdeli še v dva APB (Advanced Peripheral Bus) vodila, na

katere so priklopljene periferne naprave. V mojem primeru sem uporabil ADC (Analog to digital converter), GPIO (General-purpose input/output), časovnik in USART (Universal asynchronous receiver/transmitter) naprave. Vsako napravo je treba pred uporabo vklopiti. Naprave vžigamo z uporabo funkcij *RCC_APBxPeriphClockCmd* in *RCC_AHBxPeriphClockCmd*, kjer je x vodilo, na katerega je naprava priklopljena. Funkcije sprejmejo vrednost, ki predstavlja napravo in stanje, v katero želimo napravo spraviti (vklop/izklop).

Izvorna koda 6.1: Vklop ADC in GPIO naprav

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1ENR_GPIOCEN, ENABLE);
```

Izvorna koda 6.2: Vklop USART in GPIO naprav

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
```

Izvorna koda 6.3: Vklop časovnika

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
```

6.3 Zajemanje analognih vrednosti

Za zajemanje signala iz radarja sem uporabil vgrajen ADC ali analogno digitalni pretvornik. ADC prebere trenutno električno napetost na priključku in nam poda vrednost kot število. Pretvorniki imajo različne resolucije. Pri tem mikroprocesorju je resolucija 12 bitna, kar pomeni, da nam vrača vrednosti v intervalu $[0, 4095]$. Parametri za uporabljen ADC, ki so pomembni v mojem primeru, so naslednji:

- Resolucija
- Poravnava podatkov

- Število konverzij
- Čas vzorčenja enega vzorca

Resolucijo lahko nastavimo na 6, 8, 10 ali 12 bitov. Katero vrednost uporabimo, je odvisno od želene natančnosti. Natančnost zajema signala je odvisna od uporabe. V mojem primeru sem resolucijo nastavil na 12 bitov, saj sem želel dobiti najbolj natančno obliko signala.

Poravnava podatkov je odvisna od interpretacije. Tukaj lahko izberemo levo ali desno poravnavo.

Število konverzij je parameter, ki narekuje, kolikokrat se bo izvedla pretvorba podatkov.

Čas vzorčenja nam pove, koliko časa bo trajala meritev enega vzorca. V tem času se opravi več konverzij, iz katerih se na koncu izračuna povprečje.

Izvorna koda 6.4: Konfiguracija ADC naprave

```
ADC_init_structure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_init_structure.ADC_Resolution = ADC_Resolution_12b;
ADC_init_structure.ADC_ContinuousConvMode = ENABLE;
ADC_init_structure.ADC_ExternalTrigConv =
                    ADC_ExternalTrigConv_T1_CC1;
ADC_init_structure.ADC_ExternalTrigConvEdge =
                    ADC_ExternalTrigConvEdge_None;
ADC_init_structure.ADC_NbrOfConversion = 16;
ADC_init_structure.ADC_ScanConvMode = DISABLE;
ADC_Init(ADC1, &ADC_init_structure);
ADC_Cmd(ADC1, ENABLE);

ADC_RegularChannelConfig(ADC1, ADC_Channel_10,
                         1, ADC_SampleTime_480Cycles);
```

6.4 GPIO

Izvorna koda 6.5: Konfiguracija GPIO naprave

```
GPIO_InitTypeDef GPIO_initStructre;
GPIO_StructInit(&GPIO_initStructre);
GPIO_initStructre.GPIO_Pin = GPIO_Pin_0;
GPIO_initStructre.GPIO_Mode = GPIO_Mode_AN;
GPIO_initStructre.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOC, &GPIO_initStructre);
```

Za uporabo ADC pretvarjanja moramo imeti nek priklop, iz katere pretvornik bere napetost. Za to uporabimo GPIO napravo, ki omogoča priklop zunanjih naprav, na primer radar, na ploščo. GPIO naprave se lahko uporablja za več stvari. Možni načini uporabe so:

- Vhod
- Izhod
- Analogno
- Alternativna funkcija

Uporaba naprave v vhodnem načinu pomeni, da lahko beremo, kakšno je logično stanje na priključku. Logično stanje je lahko visoko (3.3V) ali nizko (0V).

Če uporabljamo napravo v izhodnem načinu, lahko priključku nastavljamo, kakšno logično stanje naj bo na izhodu.

V primeru, da napravo uporabljamo v analognem načinu, lahko iz priključka beremo analogne vrednosti električne napetosti. To storimo z uporabo ADC periferne naprave.

Uporaba GPIO naprave v načinu Alternativna funkcija, pomeni da z napravo upravlja ena izmed ostalih perifernih naprav. V tem primeru izgubimo nadzor nad priključkom in ga prepustimo periferni napravi. To se uporablja v primeru uporabe USART komunikacije. USART napravi nastavimo, kakšno sporočilo naj pošlje in potem naprava poskrbi za komunikacijo ter spreminjane in branje logičnih stanj na priključkih.

6.5 Vzorčenje signala

Vzorčenje signala sem opravljal z uporabo časovnika. Časovnik je posebna naprava, ki se uporablja za generacijo signalov, proženje prekinitev in merjenje časa. Časovnik, ki sem ga uporabil, teče s frekvenco $SysClock/2$. $SysClock$ je frekvenca, pri kateri deluje celotni mikroprocesor in je v mojem primeru 168 MHz. Za dodatno nastavljanje frekvence lahko uporabimo delilnik frekvence (Prescaler). Z delilnikom lahko nastavimo vrednost delilnika frekvence. Interval delilnika je $[0, 65535]$. Poleg delilnika lahko nastavimo tudi vrednost, do katere šteje in se nato ponastavi. Kadar se časovnik ponastavi, lahko sproži prekinitev.

Izvorna koda 6.6: Konfiguracija časovnika

```
TIM_BaseStruct.TIM_Prescaler = 0;
TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
TIM_BaseStruct.TIM_Period = 1903; /* 44kHz */
TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
TIM_BaseStruct.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(TIM4, &TIM_BaseStruct);
TIM_Cmd(TIM4, ENABLE);
TIM_ITConfig(TIM4, TIM_IT_Update, ENABLE);

NVIC_InitTypeDef nvic_init;
nvic_init.NVIC_IRQChannelCmd = ENABLE;
nvic_init.NVIC_IRQChannel = TIM4_IRQn;
nvic_init.NVIC_IRQChannelSubPriority = 1;
nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_Init(&nvic_init);
```

Časovnik sem nastavil, da šteje do števila 1903, kar z nastavljeno frekvenco pomeni, da se bo ponastavil vsakih 22 mikrosekund. Nastavil sem, da se ob ponastavljanlju sproži prekinitev, v kateri vzorčim signal z ADC periferno napravo. Program je spisan tako, da vsakič, ko se zgodi prekinitev, shrani

vrednost ADC-ja v pomnilnik in poveča naslov, kamor se podatki shranjujejo. Ko pride do želene velikosti vzorca v *main* funkciji, to zaznam in predam vzorec za obdelavo s FFT algoritmom. Ko se obdelava konča, se proces začne ponovno.

6.6 Računanje hitrosti

Ko iz časovnika dobimo vzorčen signal, ga naprej s pomočjo FFT algoritma spremenimo v frekvenčni prostor. Nato iz rezultata FFT algoritma najdemo največjo vrednost in iz nje izračunamo hitrost. Postopek je isti, kot je opisano v poglavju 5.4.

Poglavje 7

Sklepne ugotovitve

Za izdelavo diplomske naloge sem razvil sistem za merjenje hitrosti objektov. Sistem se bo uporabljal v tablah za merjenje hitrosti, ki sem jih opisal v uvodnem poglavju. Za pomoč pri razvoju sem spisal testni program, kjer lahko nastavljamo različne parametre za računanje hitrosti. Pri uporabi sistema na različnih lokacijah lahko pride do motenj, ki vplivajo na natančnost meritve. Ena izmed nadgradenj sistema bo razvoj algoritma, ki bo poskušal odpraviti te motnje, da bo meritev čimbolj natančna. Zaenkrat se lahko teh motenj rešimo z nastavljanjem parametrov pri računanju. Testni program se za ta primer izkaže kot zelo uporaben, saj lahko na lokaciji, na kateri bo sistem meril hitrost, zajamemo meritev in kasneje prilagodimo parametre. Ker je implementacija algoritma in Fouriereve transformacije v obeh sistemih enaka, lahko te parametre uporabimo tudi na mikrokrmilniku, kar bo izboljšalo natančnost meritev. Trenutna verzija sistema pokriva vse cilje, ki sem si jih zadal za izdelavo diplomske naloge. V prihodnosti bom sistem še nadgradil. Končni izdelek je sistem, ki uporablja mikrokrmilnik, na katerega je priključen Dopplerjev radar. Ta sistem meri hitrost mimoidočih avtomobilov in z uporabo USART komunikacije sporoči hitrost.

V prihodnosti imam namen sistem razširiti z naslednjimi nadgradnjami:

- nastavljanje parametrov z uporabo konfiguracijske datoteke
- pridobivanje podatkov v frekvenčni domeni prek USB vodila za ta-

kojšno analizo

- razvoj algoritma, ki bo odpravil motnje
- pošiljanje hitrosti z uporabo različnih komunikacijskih vodil
- uporaba RMS obdelave na mikrokrmilniku

Literatura

[1] C++ programski jezik [Online]. Dosegljivo: http://www.cplusplus.com/info/description/. [Dostopano februar 2016].

[2] CMake [Online]. Dosegljivo: https://cmake.org/overview/. [Dostopano 06. 03. 2016].

[3] QT Knjižnica [Online]. Dosegljivo: http://wiki.qt.io/About_Qt. [Dostopano februar 2016].

[4] GCC [Online]. Dosegljivo: https://gcc.gnu.org/ [Dostopano februar 2016].

[5] Dopplerjev pojav [Online]. Dosegljivo: https://sl.wikipedia.org/wiki/Dopplerjev_pojav [Dostopano februar 2016].

[6] Dopplerjev radar [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Doppler_radar [Dostopano februar 2016].

[7] Jurij F. Tasič, "Postopki digitalne obdelave signalov", Univerza v Ljubljani, Fakulteta za elektrotehniko, 2002.

[8] Fouriereva transformacija [Online]. Dosegljivo: http://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/ [Dostopano februar 2016].

[9] Dopplerjev pojav [Online]. Dosegljivo: https://si.openprof.com/wb/dopplerjev_pojav?ch=340 [Dostopano februar 2016].

[10] Dopplerjev radar [Online]. Dosegljivo: http://www.radartutorial.eu/11.coherent/co06.en.html [Dostopano februar 2016].

[11] Dopplerjev radar slika [Online]. Dosegljivo: https://upload.wikimedia.org/wikipedia/commons/thumb/7/7f/Doppler-effect-two-police-cars-diagram.png/350px-Doppler-effect-two-police-cars-diagram.png [Dostopano februar 2016].

[12] Dopplerjev radar slika [Online]. Dosegljivo: http://s1.e-monsite.com/2009/02/27/11/3477400effet-doppler-fizeau-png.png [Dostopano februar 2016].

[13] Diskretna Fouriereva transformacija [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Discrete_Fourier_transform [Dostopano februar 2016].

[14] Hitra Fouriereva transformacija [Online]. Dosegljivo: http://mathworld.wolfram.com/FastFourierTransform.html [Dostopano februar 2016].

[15] Knjižnica za prebiranje WAV datotek libsndfile [Online]. Dosegljivo: http://www.mega-nerd.com/libsndfile/ [Dostopano februar 2016].

[16] Knjižnica za računanje FFT [Online]. Dosegljivo: http://kissfft.sourceforge.net/ [Dostopano februar 2016].

[17] Root mean square [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Root_mean _square [Dostopano februar 2016].

[18] STM32F4 disocovery [Online]. Dosegljivo: http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419?sc=internet/e [Dostopano februar 2016].

[19] RFBeam K-MC1 [Online]. Dosegljivo: http://www.rfbeam.ch/products/k-mc1-transceiver/ [Dostopano februar 2016].

[20] Slika STM32F4 Discovery [Online]. Dosegljivo: http://nicot31.fr/wp-content/uploads/2014/12/M-05313.jpg [Dostopano februar 2016].

[21] Slika PCM [Online]. Dosegljivo: https://camo.githubusercontent.com/f7df526cf9c643f61ad6701240c19652c8cc16bc/68747 [Dostopano februar 2016].

[22] Slika RFBeam K-MC1 [Online]. Dosegljivo: http://www.beyd.com.cn/uploads/image/20140713/1405248438.jpg

[23] Zgodovina Dopplerjevga radarja [Online]. Dosegljivo: https://en.wikipedia.org/wiki/Doppler_radar#History [Dostopano marec 2016].

[24] Slika RFBeam ST100 [Online]. Dosegljivo: http://www.rfbeam.ch/fileadmin/downloads/datasheets/UserManual_ST100.pdf [Dostopano marec 2016].

[25] Podjetje KBros, ki izdeluje opozorilne table [Online]. Dosegljivo: http://www.kbros.si/produkti.html [Dostopano marec 2016].