Daljinska istraživanja 2. DZ

Tea Teskera, 0036522056

22. siječnja 2025.

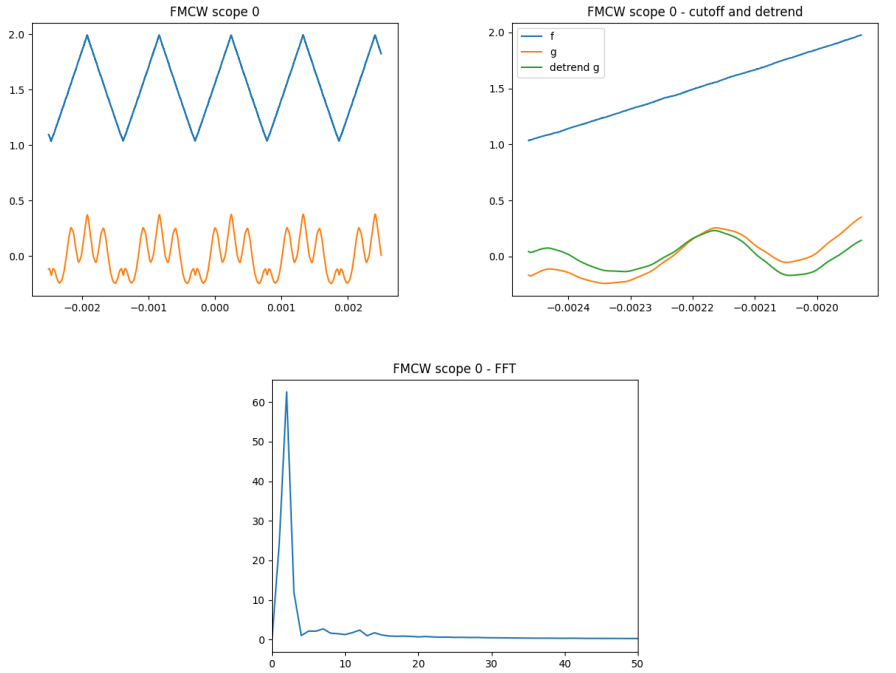
Koristila sam Python 3.10 i biblioteke numpy, scipy i matplotlib. Cijeli kod se nalazi na kraju dokumenta.

FMCW

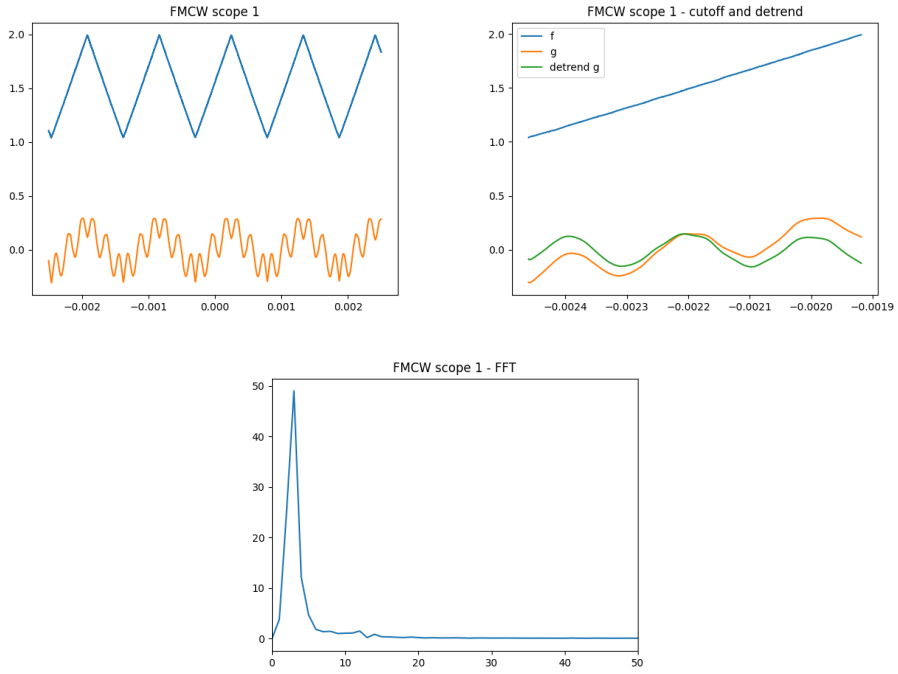
Tablica 1: Sažetak rezultata

scope_i	procjena stvarne udaljenosti [m]	dobivena frekvencija [Hz]	dobivena udaljenost [m]
0	0.30	3737	0.41616
1	0.50	5538	0.62425
2	0.55	5620	0.62424
3	0.25	3774	0.41615

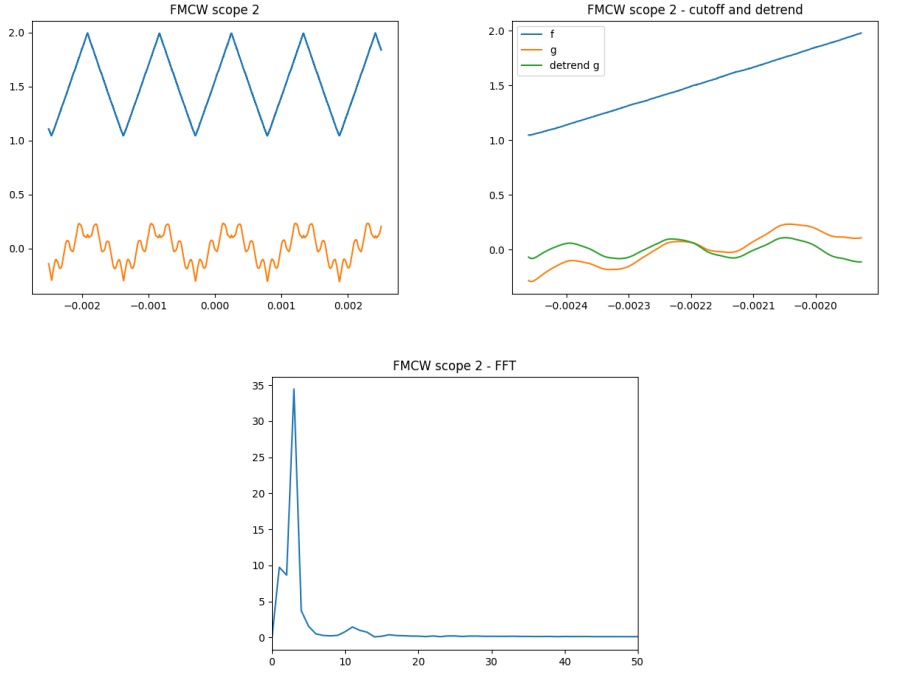
Vidimo kako postoji greška od desetak centimetara kod dobivene udaljenosti. Promjena stvarne udaljenosti za 5 cm nije značajno promijenila dobivenu udaljenost (promjena se vidi tek na 5. decimali). U nastavku donosimo generirane grafove.



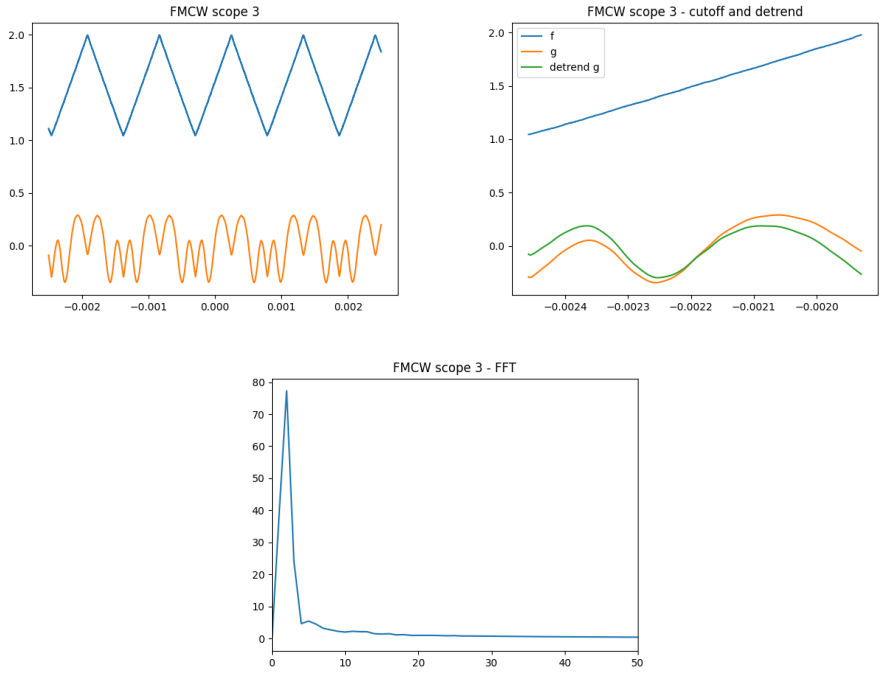
Slika 1: scope_0 - gore lijevo: generirani i primljeni signali (x os vrijeme u sekundama, y os napon u voltima); gore desno: izrezani signali zajedno s detrendanim primljenim signalom (x os vrijeme u sekundama, y os napon u voltima); dolje: spektar dobiven FFT-om (dio od interesa)



Slika 2: scope_1 - gore lijevo: generirani i primljeni signali; gore desno: izrezani signali zajedno s detrendanim primljenim signalom; dolje: spektar dobiven FFT-om (dio od interesa)



Slika 3: scope_2 - gore lijevo: generirani i primljeni signali; gore desno: izrezani signali zajedno s detrendanim primljenim signalom; dolje: spektar dobiven FFT-om (dio od interesa)



Slika 4: scope_3 - gore lijevo: generirani i primljeni signali; gore desno: izrezani signali zajedno s detrendanim primljenim signalom; dolje: spektar dobiven FFT-om (dio od interesa)

Postupak za dobivanje udaljenosti objekta FMCW radarom

Za svaki scope od 0 do 3 smo ponovili sljedeći postupak:

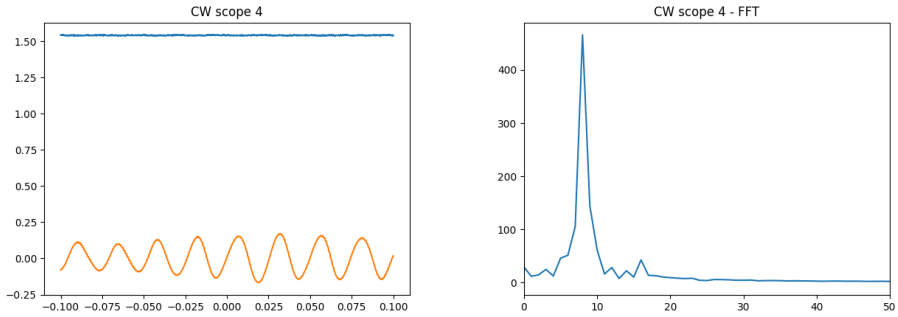
1. parsiranje CSV datoteka dobivenih iz osciloskopa (i vizualizacija dvaju signala)
2. izrezivanje signala tako da ostane samo dio gdje je generirani signal prvi put monotono rastuć
3. provođenje detrend postupka nad izrezanim dijelom primljenog signala (i vizualizacija 2. i 3. koraka)
4. izračun koji *bins* FFT-a odgovaraju kojim frekvencijama sa scipy.fft.fftfreq()
5. provođenje samog FFT postupka sa scipy.fft.fft() (i vizualizacija dobivenog)
6. korištenje rezultata prethodno spomenutih funkcija u dobivanju $\triangle f$
7. izračun udaljenosti po formuli: $\triangle R = \frac{c * \triangle f}{2 * \frac{B}{\tau}}$, gdje je B širina pojasa u iznosu od 720 MHz

CW (Doppler)

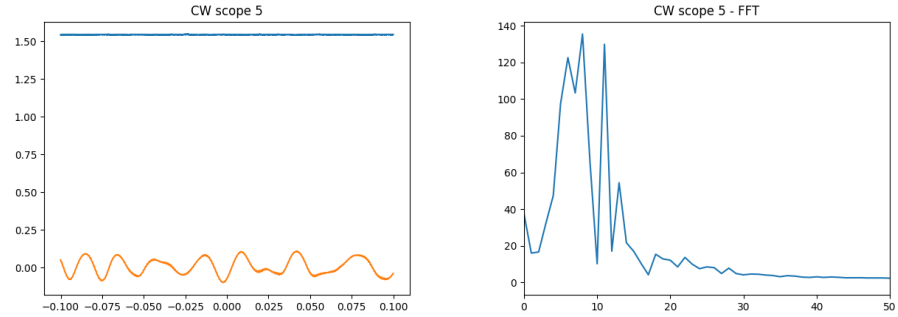
Tablica 2: Sažetak rezultata

scope_i	dobivena frekvencija [Hz]	dobivena brzina [m/s]
4	40.0	0.25
5	30.0	0.1875
6	20.0	0.125

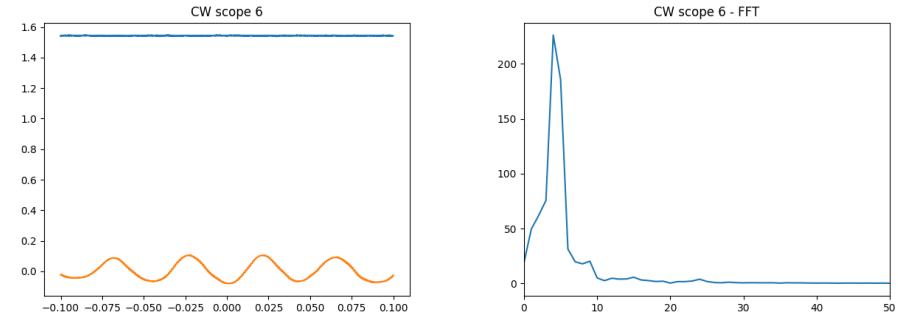
Dobivene frekvencije, a posljedično i brzine, ispadaju vrlo "zaokruženo". Ovo je vjerojatno zbog toga što diskretizirane *bins* kod FFT-a predstavljaju svakih 5 Hz, jer smo ovdje imali trajanje signala točno 0.2 sekunde (nismo izrezivali kao kod FMCW).



Slika 5: scope_4 - lijevo: generirani i primljeni signali; desno: spektar dobiven FFT-om (dio od interesa)



Slika 6: scope_5 - lijevo: generirani i primljeni signali; desno: spektar dobiven FFT-om (dio od interesa)



Slika 7: scope_6 - lijevo: generirani i primljeni signali; desno: spektar dobiven FFT-om (dio od interesa)

Postupak za dobivanje brzine objekta CW radarom

Za svaki scope od 4 do 6 smo ponovili sljedeći postupak:

1. parsiranje CSV datoteka dobivenih iz osciloskopa (i vizualizacija dvaju signala)
2. izračun koji *bins* FFT-a odgovaraju kojim frekvencijama sa scipy.fft.fftfreq()
3. provođenje samog FFT postupka sa scipy.fft.fft() (i vizualizacija dobivenog)
4. korištenje rezultata prethodno spomenutih funkcija u dobivanju $\triangle f$
5. izračun brzine po formuli: $v = \frac{c * \triangle f}{2 * f_c}$, gdje je f_c centralna frekvencija modula od 24 GHz

Programski kod

```python
import numpy as np
from scipy import signal
from scipy.fft import fft, fftfreq
import matplotlib.pyplot as plt
import argparse
import os

c = 3 * 10**8
B = 720 * 10**6
fc = 24 * 10**9

# Parsing command line arguments
parser = argparse.ArgumentParser()

parser.add_argument("-v", "--visualize", help="visualize with plots", action='store_true')
parser.add_argument("-nf", "--nofmcw", help="disable the FMCW part of the code", action='store_true')
parser.add_argument("-nd", "--nodoppler", help="disable the Doppler part of the code", action='store_true')

args = parser.parse_args()

print("Visualize with plots is set to", args.visualize)
print("Run 'python lab2.py --help' for more information on available flags")

if not args.nofmcw:
    n_fmcw_measurements = 4
    for i in range(n_fmcw_measurements):
        print(f"Current scope number: {i}")

        f_path = os.path.join('data', f'scope_{i}_1.csv')
        g_path = os.path.join('data', f'scope_{i}_2.csv')
        f = np.genfromtxt(f_path, delimiter=',')[2:]
        g = np.genfromtxt(g_path, delimiter=',')[2:]

        if args.visualize:
            plt.plot(f[:, 0], f[:, 1], label='generated signal')
            plt.plot(g[:, 0], g[:, 1], label='received signal')
            plt.title(f"FMCW scope {i}")
            plt.show()

        f_peaks, _ = signal.find_peaks(f[:,1], height=0.99*np.max(f[:,1]))
        max_f = f_peaks[0]
        min_f = np.argmin(f[:,1][0:max_f])
        print("Index of the first minimum of f:\n", min_f)
        print("Index of the first maximum of f:\n", max_f)
        print("^These two values will be where we do the signal cutoff by indices.\n")

        detrended_signal = signal.detrend(g[:, 1][min_f:max_f])

        if args.visualize:
            plt.plot(f[:, 0][min_f:max_f], f[:, 1][min_f:max_f], label='f')
            plt.plot(g[:, 0][min_f:max_f], g[:, 1][min_f:max_f], label='g')
            plt.plot(g[:, 0][min_f:max_f], detrended_signal, label='detrend g')
            plt.legend()
            plt.title(f"FMCW scope {i} - cutoff and detrend")
            plt.show()

        N = len(g[:,1][min_f:max_f]) # number of samples
        T = g[:,0][2]-g[:,0][1] # time between two samplings
        tau = g[:, 0][min_f:max_f][-1] - g[:, 0][min_f:max_f][0]
        frequencies = fftfreq(N, T)[:N // 2]
        res = np.abs(fft(detrended_signal)[:N // 2])
        peak_indices, _ = signal.find_peaks(res, height=0.5 * np.max(res))

        if args.visualize:
            plt.plot(res)
            plt.title(f"FMCW scope {i} - FFT")
```

```
            plt.show()

        freq = frequencies[peak_indices[0]]
        print("Frequency from FFT in Hz:", np.round(freq))
        R = c * freq * tau / (2 * B)
        print("R in meters:", np.round(R,5))
        print("--------------------------------")


#Doppler (CW Radar)
if not args.nodoppler:
    doppler_scopes = [4,5,6]
    for i in doppler_scopes:
        print(f"Current scope number: {i}")
        f_path = os.path.join('data', f'scope_{i}_1.csv')
        g_path = os.path.join('data', f'scope_{i}_2.csv')
        f = np.genfromtxt(f_path, delimiter=',')[2:]
        g = np.genfromtxt(g_path, delimiter=',')[2:]

        if args.visualize:
            plt.plot(f[:, 0], f[:, 1], label='generated signal')
            plt.plot(g[:, 0], g[:, 1], label='received signal')
            plt.title(f"CW scope {i}")
            plt.show()

        N = len(g[:, 1]) # number of samples
        T = g[:, 0][2] - g[:, 0][1] # time between two samplings
        frequencies_d = fftfreq(N, T)[:N // 2]
        res_d = np.abs(fft(g[:,1])[:N // 2])
        peak_indices_d, _ = signal.find_peaks(res_d, height=0.5 * np.max(res_d))

        if args.visualize:
            plt.plot(res_d)
            plt.title(f"CW scope {i} - FFT")
            plt.show()

        freq_d = frequencies_d[peak_indices_d[0]]
        print("Frequency from FFT in Hz:", np.round(freq_d, 3))
        v = freq_d * c / (2 * fc)
        print("v in m/s:", np.round(v,5))
        print("--------------------------------")
```