SKAITMENINIS SIGNALŲ APDOROJIMAS IR MAŠININIS MOKYMASIS 2022
Laboratorinis darbas nr. 3

ADAPTYVIŲJŲ FILTRŲ TYRIMAS

Ž. Marma, E MEI-2 gr. Dėstytojas D. Sokas
KTU, Elektros ir elektronikos fakultetas

Įvadas

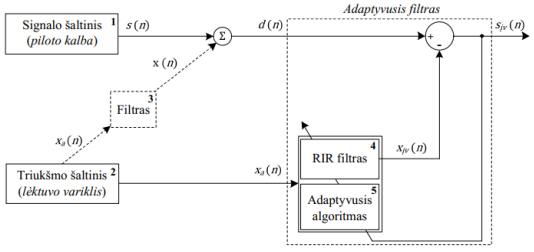
Adaptyvieji filtrai naudojami siekiant prisitaikyti prie kintančių triukšmo charakteristikų adaptuodami filtro koeficientus taip, kad jie duotuoju laiko momentu būtų optimalūs. Šios savybės neturintys filtrai su pastoviais koeficientais tampa neefektyvūs esant nestacionariam pašaliniui triukšmui. Todėl šiame darbe yra tiriami mažiausių vidutinių kvadratų (MVK), normalizuotų mažiausių vidutinių kvadratų (NMVK) ir dekoreliuotas rekursinio mažiausių vidutinių kvadratų algoritmai. Laboratorinio darbo tikslas – išmokti įgyvendinti ir tirti adaptyviuosius filtrus. Laboratorinio darbo užduotis yra adaptyviųjų filtrų pagalba prislopinti lėktuvo variklio triukšmo dedamąją piloto kalbos signale.

Metodai

Adaptyvusis filtras

Laboratorinio darbo schemoje (1 pav.) matyti, kad adaptyvusis filtras turi du įėjimus ir vieną išėjimą. Į vieną įėjimą patenka piloto kalbos $s(n)$ ir variklio triukšmo $x(n)$ sudėtinis signalas $d(n)$.Signalą $d(n)$ galima vadinti lėktuvo kabinos garsų signalu. Į kitą, atraminį įėjimą, patenka variklio triukšmo signalas $x_a\,(n)$. Variklio triukšmo signalas turi būti registruojamas toje vietoje, kurioje nebūtų pašalinių garsų, o ypatingai – piloto kalbos. Reikia atkreipti dėmesį, kad signalai $x(n)$ ir $x_a(n)$ yra tarpusavyje koreliuoti, tačiau nevienodi, t.y. į piloto mikrofoną patenkantis variklio garso signalas būna pakitęs dėl perėjimo per kabinos sienas, kurias galima įsivaizduoti kaip tam tikrą filtrą.

Adaptyvusis algoritmas (blokas 5) kas kartą turi taip parinkti skaitmeninio filtro koeficientus (blokas 4), kad triukšmo įverčio signalas $x_{įv}(n)$ taptų kuo panašesnis į triukšmo dedamąją, registruojamą kartu su piloto kalbos signalu. Tuomet iš kabinos mikrofono užregistruoto signalo $d(n)$ atėmus triukšmo įverčio signalą $x_{įv}(n)$, gaunamas piloto kalbos signalo įvertis $s_{įv}(n)$. Idealiu atveju, adaptyviuoju filtru išskirto piloto kalbos signalo įvertis turėtų būti identiškas triukšmu nepaveiktam piloto kalbos signalui $s(n)$.



1 pav. Mažiausių vidutinių kvadratų adaptyviojo filtro schema piloto kalbos signalui išskirti.

Adaptyvusis filtras adaptuoja koeficientus tokiu būdu, kad klaida tarp tikrojo piloto kalbos signalo $s(n)$ ir jo įverčio $s_{įv}(n)$ būtų minimali. Adaptyviojo algoritmo konvergavimo sparta ir stabilumas apsprendžiamas adaptacijos žingsniu μ. Adaptacijos žingsnis įprastai parenkamas iš intervalo $0 < \mu \ll 1$. Reikia pažymėti, kad didesnė adaptacijos žingsnio μ vertė leidžia greičiau adaptuoti skaitmeninio filtro koeficientus prie pasikeitusių signalo charakteristikų, tačiau parinktus per didelę μ vertę, algoritmas gali tapti nestabilus

Mažiausių vidutinių kvadratų adaptyviojo filtro įgyvendinimas

Filtro įgyvendinimas sudarytas iš šių etapų:

a) Pasirenkamos pradinės adaptyviojo filtro parametrų vertes – skaitmeninio filtro eilė M ir adaptacijos žingsnis μ.

b) Inicializuojami pradiniai filtro koeficientai. Filtro koeficientų vektoriaus narių skaičius atitinka filtro eilę. Pradinius filtro koeficientus patogu prilyginti nuliams (1)
$$w(0) = [0\ 0\ \dots\ 0]^T. \quad (1)$$

c) Inicializuojamas atraminio įėjimo signalo vektorius. Atraminio įėjimo signalo vektoriaus narių skaičius atitinka filtro eilę. Pradinį atraminio įėjimo signalo vektorių taip pat sudaro nuliai (2)
$$x_a(0) = [0\ 0\ \dots\ 0]^T. \quad (2)$$

d) Sudaromas atraminio įėjimo signalo vektorius $\boldsymbol{x_a}(n)$. Vektorius yra sudarytas iš dabartinės signalo atskaitos n ir prieš tai buvusių signalo verčių. Atraminio įėjimo signalo vektoriaus narių skaičius atitinka filtro eilę M (3)
$$x_a(0) = [x_a(n),\ x_a(n-1)\ \dots x_a(n-M+1)]^T. \quad (3)$$

e) Variklio triukšmo signalo įvertis n-ajai atskaitai randamas atraminio įėjimo signalo vektorių nufiltravus koeficientais $\mathbf{w}(n)$ (4)
$$x_{įv}(n) = w^T(n)x_a(n). \quad (4)$$

f) Piloto kalbos signalo įvertis randamas iš kabinoje užregistruoto signalo atėmus variklio garso įverčio signalą (5)
$$s_{įv}(n) = d(n) - x_{įv}(n). \quad (5)$$

g) Skaitmeninio filtro koeficientai $\mathbf{w}(n+1)$, kurie yra naudojami diskretinio laiko momentu n + 1 yra atnaujinami pagal išraišką (6)
$$w(n+1) = w(n) + 2\mu \cdot s_{įv}(n) \cdot x_a(n). \quad (6)$$

Klaidos kriterijumi dažnai naudojamas vidutinės kvadratinės klaidos *MSE* (angl. *mean squared error*) įvertis, kuriuo įvertinama klaida tarp norimo signalo $s(n)$ ir adaptyviuoju filtru išskirto signalo įverčio $s_{įv}$(n) (7)

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(s(n) - s_{iv}(n))^2. \quad (7)$$

Normalizuotas mažiausių vidutinių kvadratų adaptyvusis algoritmas

NMVK algoritmas yra įgyvendinamas analogiškai MVK algoritmui, tačiau skiriasi koeficientų adaptacijos principas. Adaptacijos koeficientas μ yra normuojamas pagal atraminio įėjimo signalo vektoriaus energiją (8)

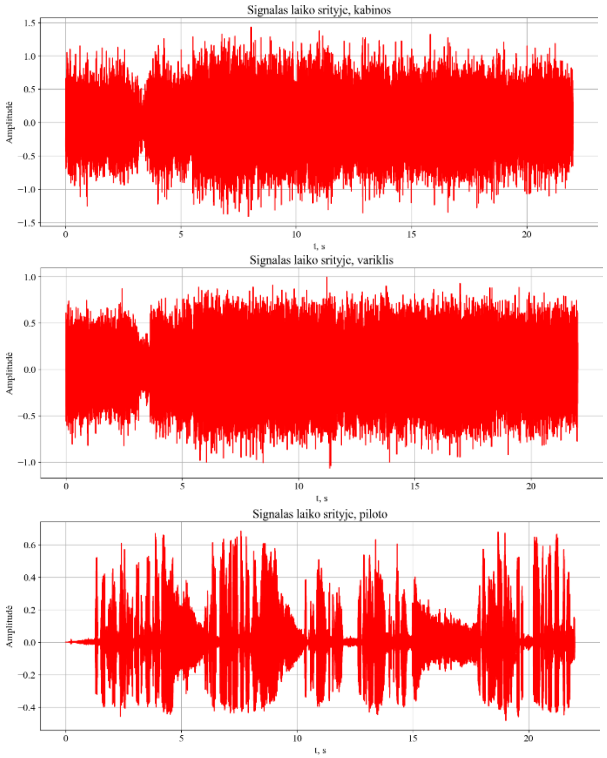
$$w(n+1) = w(n) + \frac{\mu}{x_a^T(n)\cdot x_a(n)} \cdot s_{iv}(n) \cdot x_a(n). \quad (8)$$

Dekoreliuotas rekursinis mažiausių kvadratų algoritmas

MVK ir NMVK adaptyvūs algoritmai yra paprasčiau įgyvendinami, naudoja mažesnį matematinių operacijų skaičių, tačiau lėtai adaptuojasi, todėl gerai tinka tik stacionariems arba beveik stacionariems signalams filtruoti. Kur kas spartesniu konvergavimu pasižymi rekursinis mažiausių kvadratų (RMK) adaptyvusis algoritmas. Greitas RMK algoritmo konvergavimas pasiekiamas dėl koeficientų rekursijos, t.y. koeficientai, apskaičiuoti laiko momentu $n-1$, naudojami rasti koeficientus laiko momentu n.

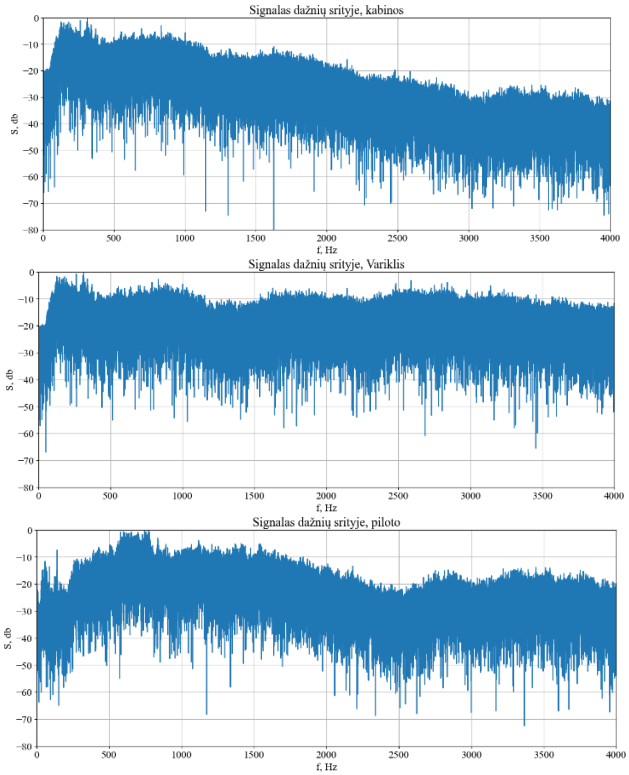
Rezultatai

Šiame darbe buvo analizuojami trys (piloto, kabinos ir variklio triukšmo) 22 sek. trukmės garso signalai, diskretizuoti 8000 Hz diskretizavimo dažniu. Analizuojant signalus laiko srityje (2 pav.) galima pastebėti, kad kabinos signalas yra užterštas variklio triukšmu. Net ir klausantis garso įrašo sunku suprasti ką tiksliai sako pilotas.



2 pav. Darbe analizuoti signalai laiko srityje

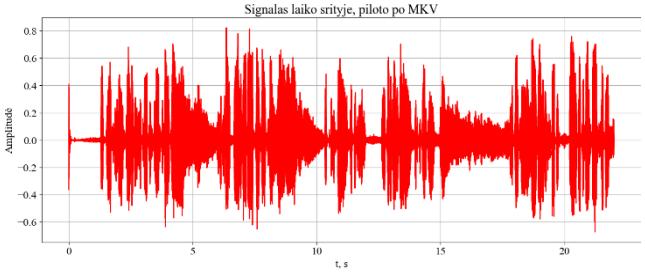
Analizuojant signalus dažnių srityje (3 pav.) galima pastebėti, jog variklio ir piloto signalai persidengia ties 400 – 1700 Hz dažniais.



3 pav. Darbe analizuoti signalai dažnių srityje

Mažiausių vidutinių kvadratų algoritmo įgyvendinimas

Įgyvendinant pradinį adaptyvųjį filtrą adaptacijos žingsnis buvo pasirinktas μ = 0,1 filtro eilė M = 20. Analizuojat piloto signalą laiko srityje (4 pav.) galima matyti, jog kalba buvo išskirta gerai. Klausanti garso įrašo tariami žodžiai yra suprantami, vienintelis trūkumas yra pašalinis garsas pradiniu laiko momentu (0-0,2 sekundė).

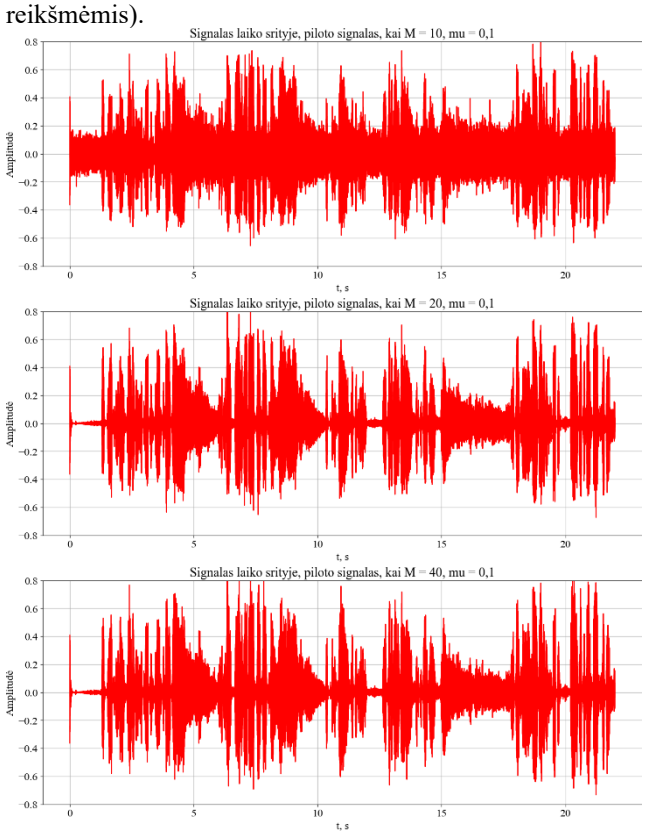


4 pav. Naudojant MVK algoritmą gautas piloto signalas laiko srityje

Mažiausių vidutinių kvadratų algoritmo parametrų parinkimas

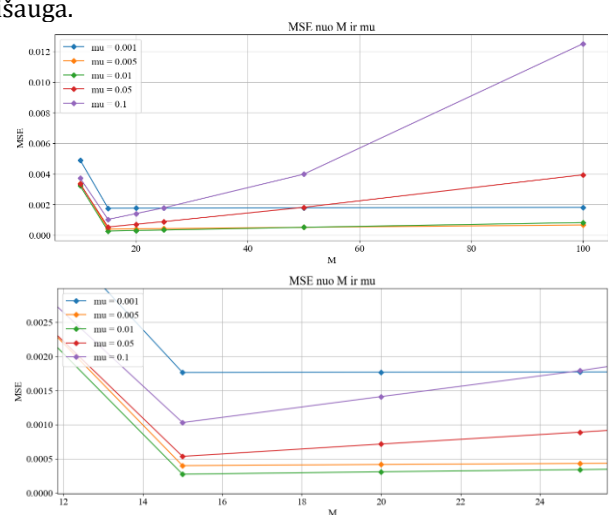
Darbe buvo ištirta kokią įtaką filtro veikimui turi jo eilė – M. Eksperimentas buvo atliktas su pradiniu M = 20, dvigubai didesniu (M=40) ir dvigubai mažesniu (M=10). Adaptacijos žingsnis nekito ir buvo lygus μ = 0,1. Pastebima (5 pav.), kad esant mažesniam M signale lieka daugiau triukšmo dedamosios. Tačiau palygus signalus, kai M=20 ir M=40, galima matyti jog esant dvigubai didesnei filtro eilei

išgautas piloto signalas neša daugiau informacijos. Remiantis tuo galime teigti, kad toliau didinant filtro eilė galime gauti nestabilų signalą (su labai didelėmis reikšmėmis).



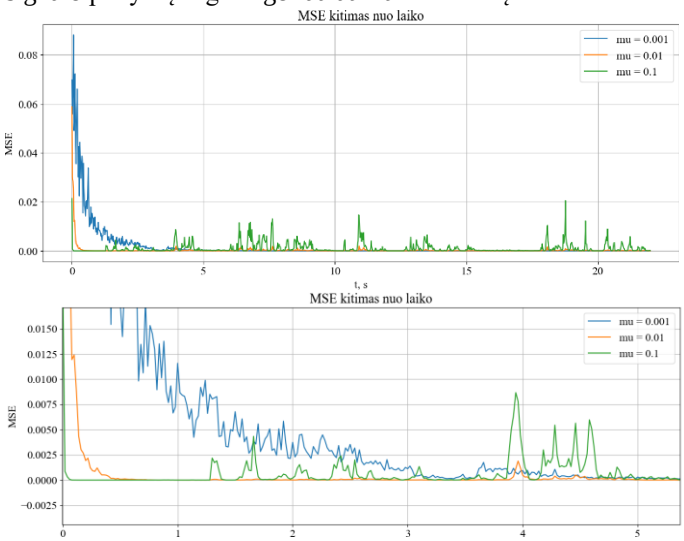
5 pav. MVK algoritmo palyinimas naudojant skirtingas M vertes

Įgyvendinto MKV algoritmo pagalba buvo rasti geriausi filtro parametrai. Filtro kokybė buvo įvertinta pagal MSE vertę (7). Ieškant optimalios filtro eilės ir adaptacijos žingsnio buvo analizuojamos šios vertės: M = [10, 15, 20, 25, 50, 100], μ = [0.001, 0.005, 0.01, 0.05, 0.1]. Gauti rezultatai yra pateikti grafiniu formatu (6 pav.). Pagal MSE rodiklį galima teigti, kad geriausia filtro parametrų pora yra M = 15 ir $\mu = 0{,}01$. Galima teigti, jog didinant filtro eilę nebūtinai gaunami geresni rezultatai, kaip pavyzdžiui M=100, tačiau resursai reikalingi skaičiavimams tikrai išauga.



6 pav. MVK algoritmo filtro MSE priklausomybė nuo M ir μ

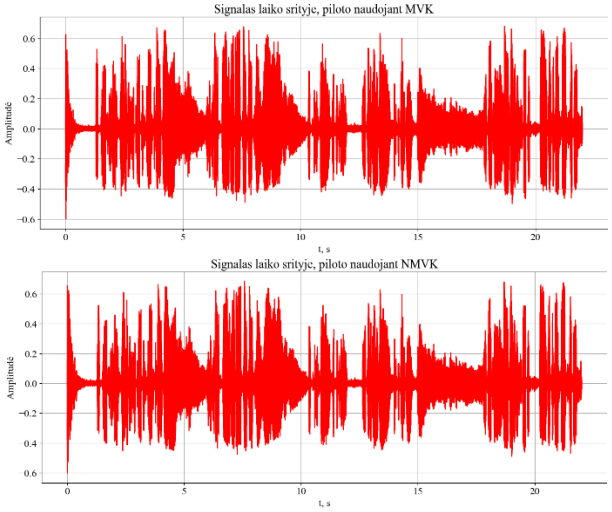
Darbe buvo atlikta filtro kokybės pagal MSE analizė kintant laiku su skirtingomis adaptacijos žingsnio vertėmis (μ = [0.001 0.01 0.1]). Analizės intervalas buvo pasirinktas 20ms, o filtro eilė M=15. Analizuojant rezultatus (7 pav.) galima pastebėti, kad mažiausią paklaidą pagal MSE rodiklį pasiekia filtras kurio μ = 0,01. Nors filtras su šiomis charakteristikomis ne taip greitai prisitaiko prie pradinio signalo kaip filtras naudojant μ = 0,1, tačiau prie tolimesnių signalo pokyčių sugeba geriau atlikti filtravimą.



7 pav. MVK algoritmo filtro MSE priklausomybė nuo laiko su skirtingomis μ vertėmis

Normalizuoto mažiausių vidutinių kvadratų adaptyviojo algoritmo įgyvendinimas

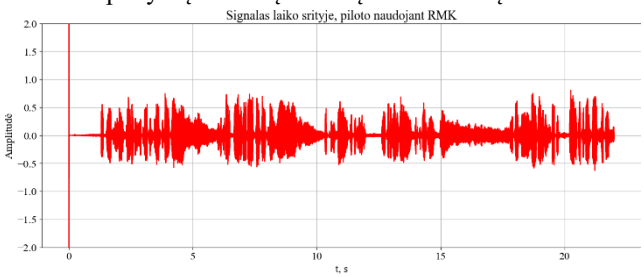
Programiškai įgyvendinus NMVK algoritmą buvo suskaičiuota MSE vertę ir palyginta su MVK algoritmu (M=15, μ = 0,01). MSE MKV = $2.75 \cdot 10^{-4}$, MSE NMVK = $3.22 \cdot 10^{-4}$. Tiek įvertinus MSE rodiklius, tiek pasiklausius gautų piloto signalų galima teigti, kad algoritmų rezultatai yra labai panašūs. Pagal MSE rodiklį šio atveju filtras naudojanti MKV algoritmą būtų geresnis. Grafiškai analizuojant gautus signalus (8 pav.) galima matyti, kad vienintelis pastebimas skirtumas yra neženkliai išaugusi amplitudės reikšmė NMVK atveju.



8 pav. MVK ir NMVK algoritmų palyginimas laiko srityje

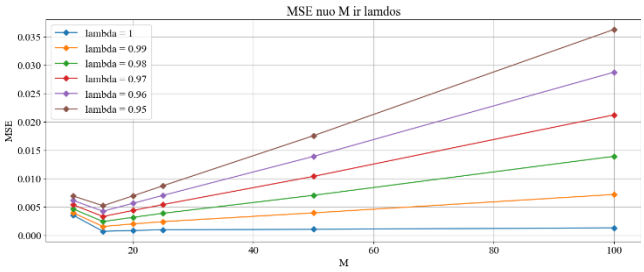
Dekoreliuotas rekursinis mažiausių kvadratų algoritmas

Realizuojant rekursinį algoritmą pradines vertės buvo M = 15, λ = 0,99. Naudojant šias reikšmes MSE vertė gauta lygi 1,53 ·10^{-3}. Gautas piloto garso signalas (9 pav.) laiko srityje yra suprantamai panašus į gautus signalus naudojant ankstesnius algoritmus šiame darbe. Tačiau priešingai nei MVK ar NMVK pradiniu laiko momentu (0-0,2s) pastebimas itin staigus signalo amplitudės šuolis. Būtent šis laiko tarpas, kol algoritmas nesugeba prisitaikyti prie triukšmo pokyčių ir lemią didesnį MSE rodiklį.



9 pav. Piloto signalas naudojant RMK algoritmo adaptyvųjį filtrą

Analogiškai MVK algoritmui RMK filtro algoritmui buvo rasti optimalūs parametrai M ir λ. Vertės buvo keičiamos taip: M = [10, 15, 20, 25, 50, 100], λ = [1, 0.99, 0.98, 0.97, 0.96, 0.95]. Grafiškai (10 pav.) galima matyti jog didėjant užmiršimo konstantai MSE vertė mažėja. Geriausi rezultatai pasiekti naudojant M = 15 ir λ = 1. Šiuo atveju MSE vertė buvo 6,6·10^{-4}.



10 pav. RMK algoritmo filtro MSE priklausomybė nuo M ir λ

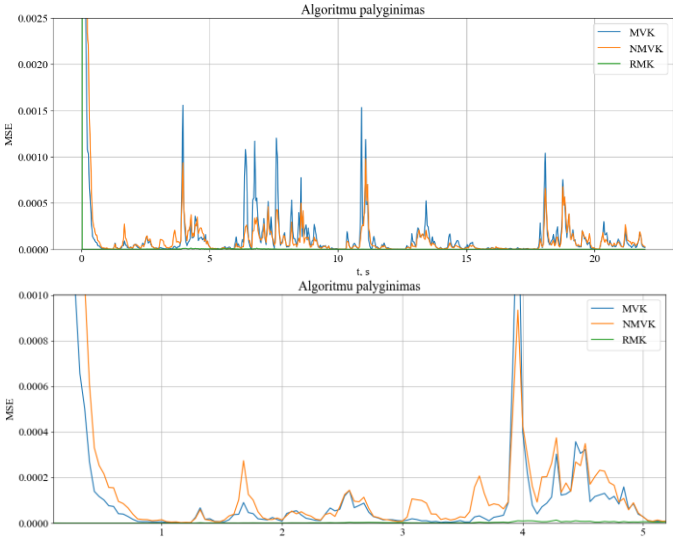
Mažiausių vidutinių kvadratų RMK adaptyviųjų filtrų palyginimas

Galiausiai MVK, NMVK ir RMK algoritmai buvo palyginti pagal MSE kitimą nuo laiko. Analizuojamų filtrų parametrai buvo pasirinkti atsižvelgiant į anksčiau gautas geriausias reikšmes: M = 15, λ = 1, μ = 0,01.

Remiantis MSE paklaidų kitimu nuo laiko (11 pav.) galima teigti, kad tinkamiausias algoritmas adaptyviajam filtrui yra RMK. Tačiau remiantis laboratoriniu darbo apraše esančiu MSE (1 lentelė) skaičiavimu geriausias algoritmas būtų MVK. Taip atsitinka, nes piloto signale gautu naudojant RMK pradiniu laiko momentu esanti paklaida iškreipia skaičiavimo tikslumą.

1 lentelė. Algoritmų palyginimas pagal MSE

Metodas	MSE vertė
MVK	2,75 ·10^{-4}
NMVK	3,22 ·10^{-4}
RMK	6,60 ·10^{-4}



11 pav. RMK, MVK ir NMVK algoritmų palyginimas pagal MSE priklausomybė nuo laiko

Diskusija

Darbe buvo analizuojamas ganėtinai stacionarus triukšmo – variklio signalas. Įdomu būtų panagrinėti, kaip tokie adaptyviųjų filtrų algoritmai veikia su kintančiu motoro garsu – pvz. mašinoje keičiantis pavaroms. Galimai RMK algoritmo rezultatai būtų blogesni.

Adaptyviųjų filtrų vienas iš trūkumas yra pašalinis garsas pradiniu laiko momentu (0-0,2 sekundė).

RMK algoritmo apskaičiavimas užtruko ganėtinai ilgą laiką. Nors darbas buvo darytas su Python programa kuri vienu metu gali išnaudoti tik vieną centrinio procesoriaus branduolį, optimalių parametrų paieška užtruko gan ilgai. Į tai reikėtų atsižlėgti norit įgyvendinti šį algoritmą įterptinėje sistemoje su ribotais skaičiavimo resursais.

Išvados

Atlikus laboratorinį darbą įsitikinome, kad kintančio triukšmo signalams yra reikalingas adaptyvusis filtras.

Įgyvendinus MKV, NMKV ir RMK algoritmus galima teigti, kad iš esmės visi darbe nagrinėti algoritmai yra tinkami adaptyviajam filtrui.

Darbe buvo rasti geriausi parametrai MKV ir NVMK algoritmams: adaptacijos žingsnis μ=0.01 ir filtro eilė M=15. Ištyrus RMK algoritmą geriausi parametrai buvo rasti: M=15 ir λ = 1. Palyginus MKV ir NVMK algoritmus geriau triukšmą pašalina MKV algoritmas

Darbe taip pat rastos MSE vertės algoritmams: MVK (2,75 ·10^{-4}), NMVK (3,22 ·10^{-4}) ir RMK (6,60·10^{-4}). Tačiau, vizualiai analizuojant paklaidos grafika buvo nustatyta, kad tinkamiausias yra RMK algoritmas.

Literatūra

Laboratorinio darbo aprašymas

Priedai

Laboratorinio darbo Python programos kodas:

```
import scipy.io
import matplotlib.pyplot as plt
import numpy as np
from scipy.io import wavfile
from sklearn import preprocessing
class Adapt:
    def __init__(self, debug = False):
        self.debug = debug
        self.time_s = 22
        self.Fs_Hz = 8000
        self.saveDir = "out/"
    def drawSignal(self, signal_y, title, show=False):
            tn = np.linspace(0, self.time_s,
num=len(signal_y))
            plt.figure
            plt.plot(tn, signal_y, 'r-')
            plt.title('Signalas laiko srityje, ' + title)
            plt.xlabel('t, s')
            plt.ylabel('Amplitudė')
            plt.grid(True)
          plt.savefig(self.saveDir + "amp_" +
title,  bbox_inches='tight',
                        pad_inches=0)
            if show:
                plt.show()
            plt.close()
    # 5 FFT and draw spectrum
    def drawSpectrum(self, signal_y, title, show= False):
        nfft = len(signal_y)
        yf = np.fft.fft(signal_y)
        spectrum = np.abs(yf) / nfft
        spectrum_db = 20 *
np.log10(spectrum/np.max(spectrum))
        k = list(range(0, nfft))
        f_Hz = [i * (self.Fs_Hz/nfft) for i in k]
        ax = plt.axes()
        ax.plot(f_Hz, spectrum_db)
        ax.set_xlim(0, self.Fs_Hz/2)
        ax.set_ylim(-80, 0)
        plt.title('Signalas dažnių srityje, ' + title)
        plt.xlabel('f, Hz')
        plt.ylabel('S, db')
        plt.grid(True)
        plt.savefig(self.saveDir + "spec_" +
title,  bbox_inches = 'tight',
                    pad_inches= 0)
        if show:
            plt.show()
        plt.close()
    def saveSignalAsWav(self, noteData, filename):
        wavfile.write(filename=filename, rate=self.Fs_Hz,
data=noteData.astype(np.float32))

# 3.4.1 MVK algoritmas
def calculate_MVK(inNoise, inCombine, filterOrder=20, step =
0.1):
    if len(inCombine) != len(inNoise):
        print("Signals are not the same length")
        exit -1
    w   = np.transpose(np.zeros((1, filterOrder)))
    x_a = np.transpose(np.zeros((1, filterOrder)))
    s_iv = np.zeros((len(inNoise),))
    for n in range(len(inNoise)):
        x_a = np.roll(x_a, 1)
        x_a[0] = inNoise[n]
        x_iv = np.matmul(np.transpose(w), x_a)
        s_iv[n] = inCombine[n] - x_iv[0]
        w = w + 2*step*s_iv[n]*x_a
    return s_iv
#3.6
def calculate_NMVK(inNoise, inCombine, filterOrder=20, step =
0.1):
    if len(inCombine) != len(inNoise):
        print("Signals are not the same length")
        exit -1
    w   = np.transpose(np.zeros((1, filterOrder)))
    x_a = np.transpose(np.zeros((1, filterOrder)))
    s_iv = np.zeros((len(inNoise),))
    for n in range(len(inNoise)):
        x_a = np.roll(x_a, 1)
        x_a[0] = inNoise[n]
        x_iv = np.matmul(np.transpose(w), x_a)
        s_iv[n] = inCombine[n] - x_iv[0]

        w = w + (step / (np.matmul(np.transpose(x_a), x_a)))
* s_iv[n] * x_a # only this differs from MVK
    return s_iv

def calculate_RMK(inNoise, inCombine, filterOrder=15,
Lam=0.99):
     if len(inCombine) != len(inNoise):
        print("Signals are not the same length")
        exit -1
    delta = 0.01
    I = np.eye(filterOrder)
    P = (delta**-1) * I
    w   = np.transpose(np.zeros((1, filterOrder)))
    x_a = np.transpose(np.zeros((1, filterOrder)))
    s_iv = np.zeros((len(inNoise),))
```

```
    for n in range(len(inNoise)):
        x_a = np.roll(x_a, 1)
        x_a[0] = inNoise[n]
        v = np.matmul(P, x_a)
        u = np.matmul(np.transpose(P), v)
        v_norm = np.linalg.norm(v)
        k = 1 / (Lam + v_norm**2 +
np.sqrt(Lam)*np.sqrt(Lam+v_norm**2))
        P = (P - k * np.matmul(v, np.transpose(u))) /
np.sqrt(Lam)
        x_iv = np.matmul(np.transpose(w), x_a)
        s_iv[n] = inCombine[n] - x_iv[0]
        if np.isnan(s_iv[n]):
            print(s_iv[n])
        w = w + ((s_iv[n] * u) / (Lam + v_norm**2))
    return s_iv

def calculate_MSE(real, prediction):
    mse = (np.square(real - prediction)).mean()
    return mse
matData = scipy.io.loadmat('signalai/lab3_signalai.mat')
print(matData)
print(type(matData))
variklioSig = matData.get('variklioSig')[0]
kabinosSig = matData.get('kabinosSig')[0]
pilotoSig = matData.get('pilotoSig')[0]

# # Normalize in 1/-1 range
# variklioSig = preprocessing.minmax_scale(variklioSig,
feature_range=(-1,1))
# kabinosSig = preprocessing.minmax_scale(kabinosSig,
feature_range=(-1,1))
# pilotoSig = preprocessing.minmax_scale(pilotoSig,
feature_range=(-1,1))
print("Loaded signals")
print(type(kabinosSig))

plt.rcParams.update({'font.family': "Times New Roman"})
plt.rcParams.update({'font.size': 16})
plt.rcParams.update({'figure.figsize': (16, 6)}) #
horizontally longer figure

filterObj = Adapt()

# Show signal over time and signal spectrum
if 1:
    # 3.3.1
    filterObj.drawSignal(variklioSig, "variklis")
    filterObj.drawSignal(kabinosSig, "kabinos")
    filterObj.drawSignal(pilotoSig, "piloto")
    filterObj.drawSpectrum(variklioSig, "variklis")
    filterObj.drawSpectrum(kabinosSig, "kabinos")
    filterObj.drawSpectrum(pilotoSig, "piloto")
    # 3.3.2
    filterObj.saveSignalAsWav(variklioSig,
f"out/variklis.wav")
    filterObj.saveSignalAsWav(kabinosSig, f"out/kabina.wav")
    filterObj.saveSignalAsWav(pilotoSig, f"out/pilotas.wav")
    print("Saved wav files")
# 3.4 MVK
if 1:
    pilot_afterMVK = calculate_MVK(variklioSig, kabinosSig)
    filterObj.drawSignal(pilot_afterMVK, "piloto po MKV")
    filterObj.drawSpectrum(pilot_afterMVK, "piloto po MKV")
    filterObj.saveSignalAsWav(pilot_afterMVK,
f"out/pilotasMVK.wav")
    print("Calculated and saved signal after MKV")

# 3.5.6 - Compare with different M
if 1:
    M_filterOrder = 20
    mu_step = 0.1
    M_array = [M_filterOrder, M_filterOrder*2,
M_filterOrder//2]
    mu_string = str(mu_step)
    mu_string = mu_string.replace('.', ',')  # Convert for
saving
    print("Analysis with different filter order")
    for M in M_array:
        sig_MVK = calculate_MVK(variklioSig, kabinosSig, M,
mu_step)
        filterObj.drawSignal(sig_MVK, f"piloto signalas, kai
M = {M}, mu = {mu_string}")
        filterObj.drawSpectrum(sig_MVK, f"piloto signalas,
kai M = {M}, mu = {mu_string}")
        filterObj.saveSignalAsWav(sig_MVK, f"out/piloto
signalas, kai M = {M}, mu = {mu_string}.wav")
# 3.5.7
# Filtro koeficientų skaičių M galite keisti logaritminiu
masteliu: 10,
# 20, 50, 100. Adaptacijos žingsnio μ vertę galite keisti
dėsniu 0.001, 0.005, 0.01, 0.05, 0.1.
if 1:
    sig_MVK = calculate_MVK(variklioSig, kabinosSig)
    print(calculate_MSE(pilotoSig, sig_MVK))
    M = [10, 15, 20, 25, 50, 100]
    u_mu = [0.001, 0.005, 0.01, 0.05, 0.1]
    print("Calculating MSE for different M and mu
combinations")
    MSE_array = np.zeros((len(M), len(u_mu)))
```

```
    for i in range(len(M)):
        for j in range(len(u_mu)):
            MSE_array[i, j] = calculate_MSE(pilotoSig,
calculate_MVK(variklioSig, kabinosSig, M[i], u_mu[j]))
    print("Calculated MSE")
    print(MSE_array)
    minIdx = np.unravel_index(np.argmin(MSE_array,
axis=None), MSE_array.shape)
    print(minIdx[0])
    print(minIdx[1])
    print(f"Min MSE is with: M={M[minIdx[0]]}, mu =
{u_mu[minIdx[1]]}")

    plt.figure
    for i_miu in range(len(u_mu)):
        temp_mu = u_mu[i_miu]
        plt.plot(M, MSE_array[:, i_miu], marker='D',
label=f"mu = {temp_mu}")
    plt.title('MSE nuo M ir mu')
    plt.xlabel('M')
    plt.ylabel('MSE')
    plt.grid(True)
    plt.legend(loc="upper left")
    plt.savefig(filterObj.saveDir +
"mse_compare",  bbox_inches='tight', pad_inches=0)
    plt.show(block=False)

# plot MSE_array over time
# Pavaizduokite adaptacijos greičio kreives (MSE
priklausomybes nuo laiko) esant adaptacijos
# žingsnio vertėms μ = 0.001, μ = 0.01, μ = 0.1. MSE galite
skaičiuoti 10 ms ar ilgesniuose
# intervaluose.
if 1:
    intervalMSE_s = 20 * 10**-3
    timeMSE_ms = np.arange(0, filterObj.time_s*1000,
intervalMSE_s)
    bestM = 15
    test_mu = [0.001, 0.01, 0.1]
    pilotEstimate1 = calculate_MVK(variklioSig, kabinosSig,
bestM, test_mu[0])
    pilotEstimate2 = calculate_MVK(variklioSig, kabinosSig,
bestM, test_mu[1])
    pilotEstimate3 = calculate_MVK(variklioSig, kabinosSig,
bestM, test_mu[2])

    MSE_overTime_array = np.zeros((3, (len(timeMSE_ms))))

    start_i = 0
    end_i = int(intervalMSE_s*filterObj.Fs_Hz) # python needs
int

    print("Calculating MSE over time")
    increaseInterval = end_i
    for i in range(len(timeMSE_ms)):
        MSE_overTime_array[0, i] =
calculate_MSE(pilotoSig[start_i:end_i],
pilotEstimate1[start_i:end_i])
        MSE_overTime_array[1, i] =
calculate_MSE(pilotoSig[start_i:end_i],
pilotEstimate2[start_i:end_i])
        MSE_overTime_array[2, i] =
calculate_MSE(pilotoSig[start_i:end_i],
pilotEstimate3[start_i:end_i])
        start_i = start_i + increaseInterval
        end_i = end_i + increaseInterval
    plt.figure
    for i in range(len(test_mu)):
        plt.plot(timeMSE_ms, MSE_overTime_array[i],
label=f"mu = {test_mu[i]}")
    plt.title('MSE kitimas nuo laiko')
    plt.xlabel('t, s')
    plt.ylabel('MSE')
    plt.grid(True)
    plt.legend(loc="upper right")
    plt.savefig(filterObj.saveDir +
"mse_overtime_mu",  bbox_inches='tight', pad_inches=0)
    plt.show(block=False)

# 3.6 - NMVK
# Normalizuoto mažiausių vidutinių kvadratų adaptyviojo
algoritmo įgyvendinimas
if 1:
    best_mu = 0.01
    bestM = 15
    print("Comparing MSE for MVK and NMVK ")
    pilot_mvk = calculate_MVK(variklioSig, kabinosSig, bestM,
best_mu)
    pilot_nmvk = calculate_NMVK(variklioSig, kabinosSig,
bestM, best_mu)
    filterObj.drawSignal(pilot_mvk, "piloto naudojant MVK")
    filterObj.drawSignal(pilot_nmvk, "piloto naudojant NMVK")
    mse_mvk = calculate_MSE(pilotoSig, pilot_mvk)
    mse_mvk_mean = calculate_MSE(pilotoSig, pilot_nmvk)
    print(f"MSE of MVK: {mse_mvk}")
    print(f"MSE of MVK mean: {mse_mvk_mean}")
# 4 Papildoma
if 1:
    sig_RMK = calculate_RMK(variklioSig, kabinosSig)
```

```
    filterObj.drawSignal(sig_RMK, "piloto naudojant RMK")
    print(f"RMK MSE value: {calculate_MSE(pilotoSig,
sig_RMK)}")

    M = [10, 15, 20, 25, 50, 100]
    lambda_val = [1, 0.99, 0.98, 0.97, 0.96, 0.95]
    MSE_array_RMK = np.zeros((len(M), len(lambda_val)))

    print("Started calculting MSE for RMK filter")
    for i in range(len(M)):
        for j in range(len(lambda_val)):
            MSE_array_RMK[i, j] = calculate_MSE(pilotoSig,
calculate_RMK(variklioSig, kabinosSig, M[i], lambda_val[j]))
    print("Calculated MSE for RMK filter")

    minIdx = np.unravel_index(np.argmin(MSE_array_RMK,
axis=None), MSE_array_RMK.shape)
    print(f"Min MSE for RMK is with: M = {M[minIdx[0]]},
lambda = {lambda_val[minIdx[1]]}")
    plt.figure
    for i_lam in range(len(lambda_val)):
        temp_lam = lambda_val[i_lam]
        plt.plot(M, MSE_array_RMK[:, i_lam], marker='D',
label=f"lambda = {temp_lam}")
    plt.title('MSE nuo M ir lamdos')
    plt.xlabel('M')
    plt.ylabel('MSE')
    plt.grid(True)
    plt.legend(loc="upper left")
    plt.savefig(filterObj.saveDir + "rmk_diff_lam",
bbox_inches='tight', pad_inches=0)
    plt.show(block=False)
# compare all there metdods overtime
if 1:
    print("Comparing all 3 metdods")
    intervalMSE_s = 40 * 10**-3
    timeMSE_ms = np.arange(0, filterObj.time_s*1000,
intervalMSE_s)
    bestM = 15
    bestLambda = 1
    best_mu = 0.01
    pilotEstimate1 = calculate_MVK(variklioSig, kabinosSig,
bestM, best_mu)
    pilotEstimate2 = calculate_NMVK(variklioSig, kabinosSig,
bestM, best_mu)
    pilotEstimate3 = calculate_RMK(variklioSig, kabinosSig,
bestM, bestLambda)
    MSE_overTime_array = np.zeros((3, (len(timeMSE_ms))))
    start_i = 0
    end_i = int(intervalMSE_s*filterObj.Fs_Hz) # python needs
int
    increaseInterval = end_i
    for i in range(len(timeMSE_ms)):
        MSE_overTime_array[0, i] =
calculate_MSE(pilotoSig[start_i:end_i],
pilotEstimate1[start_i:end_i])
        MSE_overTime_array[1, i] =
calculate_MSE(pilotoSig[start_i:end_i],
pilotEstimate2[start_i:end_i])
        MSE_overTime_array[2, i] =
calculate_MSE(pilotoSig[start_i:end_i],
pilotEstimate3[start_i:end_i])
        start_i = start_i + increaseInterval
        end_i = end_i + increaseInterval
    plt.figure
    plt.plot(timeMSE_ms, MSE_overTime_array[0], label="MVK")
    plt.plot(timeMSE_ms, MSE_overTime_array[1], label="NMVK")
    plt.plot(timeMSE_ms, MSE_overTime_array[2], label="RMK")
    plt.legend(loc="upper right")
    plt.title('Algoritmu palyginimas')
    plt.xlabel('t, s')
    plt.ylabel('MSE')
    plt.grid(True)
    plt.ylim(0, 0.0025)
    plt.savefig(filterObj.saveDir + "Algoritmu
palyginimas",  bbox_inches='tight', pad_inches=0)
    plt.show(block=False)
```