Maketa fizikalnog njihala s digitalnim prikazom parametara titranja

Stanić, Siniša

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Preddiplomski sveučilišni studij računarstva

MAKETA FIZIKALNOG NJIHALA S DIGITALNIM PRIKAZOM PARAMETARA TITRANJA

Završni rad

Siniša Stanić

Osijek, 2017.

Sadržaj

1. UVOD

Tema ovog završnog rada je izrada makete s mikroupravljačkim sustavom za mjerenje parametara fizikalnog njihala te desktop aplikacije koja će očitane parametre obraditi i grafički prikazati. Maketa i aplikacija bi se koristile za potrebe laboratorijskih vježbi kolegija Fizika 2. Svrha rada je olakšavanje i povećavanje preciznosti mjerenja parametara fizikalnog njihala (perioda i elongacije) koja bi studentima trebala omogućiti lakše razumijevanje i iščitavanje parametara za daljnja računanja karakterističnih fizikalnih veličina.

Maketa se sastoji od digitalnog (rotacijskog) enkodera, na čiju se glavu mogu staviti različita tijela (prsten, štap, jednakostraničan trokut), koji je spojen na Arduino Mega pločicu. Putem serijskog priključka, podatci koje očitava enkoder šalju se desktop aplikaciji koja će biti realizirana u Java programskom jeziku, unutar razvojnog okruženja Eclipse.

1.1. Zadatak završnog rada

Zadatak završnog rada je izrada desktop aplikacije koja će na ekranu prikazati kutnu elongaciju fizikalnog njihala u ovisnosti o vremenu. Kutna elongacija će biti mjerena putem digitalnog (rotacijskog) enkodera.

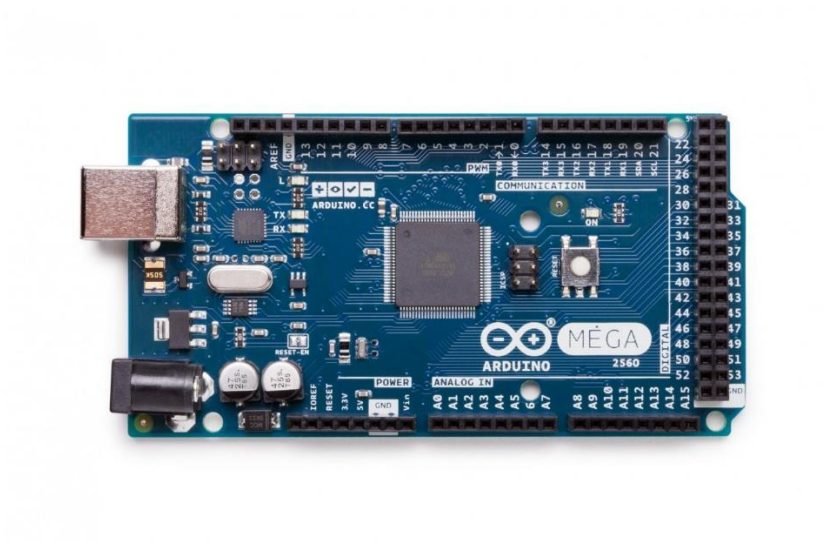
2. PRIMJENJENE TEHNOLOGIJE

2.1. Arduino

Arduino je elektronička platforma otvorenog koda koja se sastoji od sklopovlja i programske podrške. Sklopovlje čini fizički elektronički programabilni strujni krug poznatiji kao mikrokontroler, dok programsku podršku čini Arduino razvojno okruženje (engl. IDE (*Integrated Development Environment*)) u kojem se mikrokontroler programira. [1]

2.2. Arduino Mega

Arduino Mega (Slika 2.1.) je razvojna platforma bazirana na Atmelovom Atmega2560 mikrokontroleru, u kojemu se ujedno i nalazi programski kod kojeg neprestano izvršava.



Slika 2.1. Arduino Mega pločica.

Mikrokontroler	ATmega2560
Radni napon	5V
Ulazni napon (preporučeno)	7-12V
Ulazni napon (granica)	6-20V
Digitalni I/U pinovi	54 (od kojih su 15 PWM izlazi)
Analogni ulazni pinovi	16
DC struja po I/U pinu	20 mA
DC struja za 3.3V pin	50 mA

Flash memorija	256 KB od kojih 8 KB koristi bootloader
SRAM	8 KB
EEPROM	4 KB
Brzina rada	16 MHz

Tablica 1. Specifikacije Arduino Mega razvojne platforme.

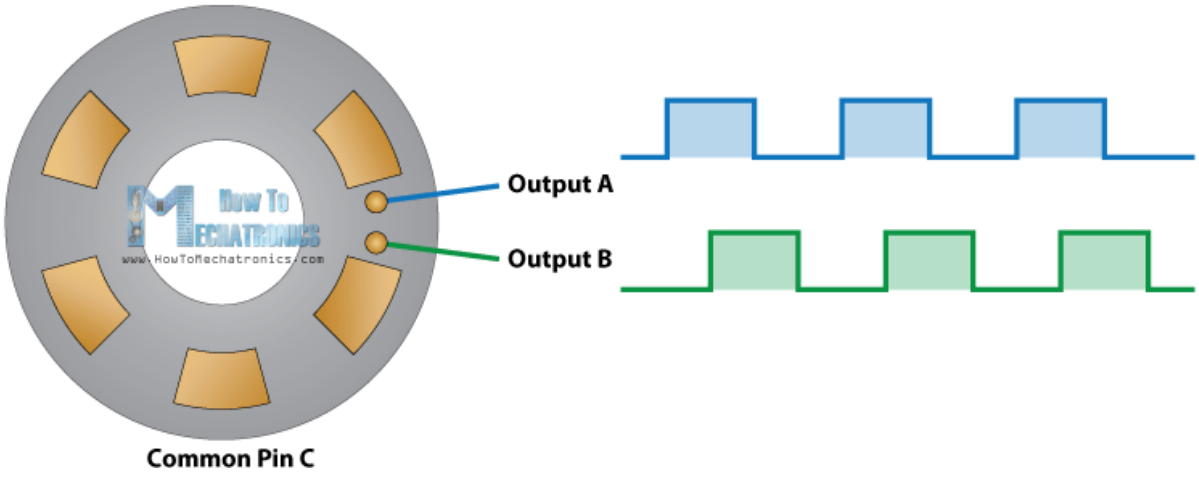
Kako bi se izbjegla oštećenja, prilikom spajanja vanjskih elemenata, naponska razina ne smije prelaziti 5V radnu naponsku razinu mikrokontrolera. [2]

2.3.Eclipse

Eclipse je integrirano razvojno okruženje za razvoj aplikacija, a primarno ga koriste Java programeri, iako okruženje nije ograničeno samo na Java programski jezik. Razvojnom okruženju mogu se dodati razni dodatci (engl. *libraries*) koji dodatno povećavaju funkcionalnost okruženja. Trenutno najnovija verzija platforme je Oxygen [3] i korištena je za izradu desktop aplikacije ovog završnog rada. Također korištena su dva dodatka : JFreeChart (kreiranje grafova) [6] i JSerialComm[7] (korištena za primanje podataka koje Arduino ploča šalje putem USB priključka).

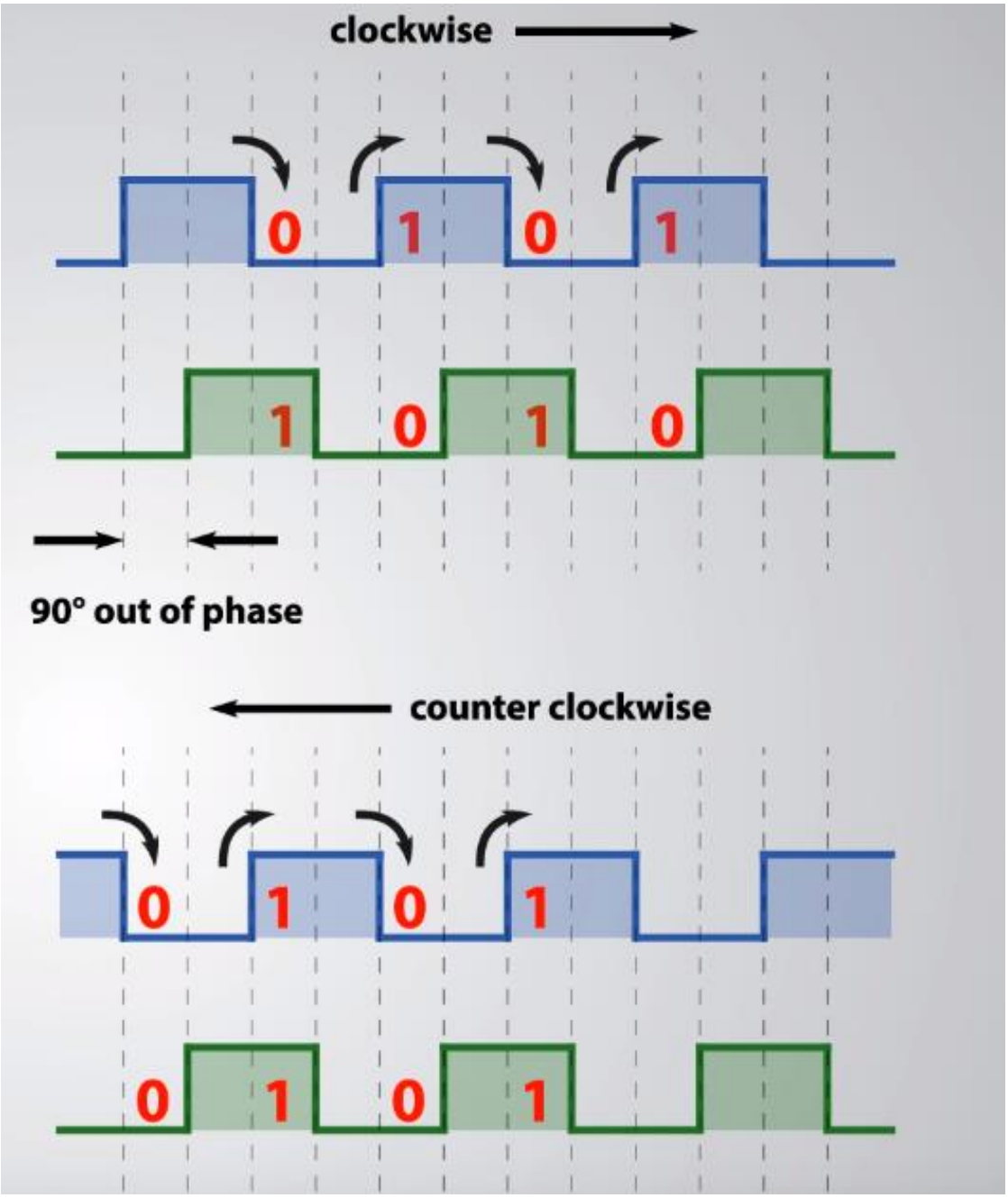
2.4.Digitalni enkoder

Digitalni rotacijski enkoder je tip pozicijskog senzora koji se koristi za određivanje kutnog položaja okretne osovine. Generira električni signal ovisno o rotacijskim kretnjama. Enkoder ima disk s jednako razmaknutim kontakt zonama (slika 2.2.) koje su spojene na zajednički *pin* C, i dva druga, odvojena *pina* A i B. Kada se disk krene okretati, *pinovi* A i B će uspostavljati kontakt sa zajedničkim *pinom* C i nastat će 2 pravokutna signala.



Slika 2.2. Shema enkodera.

Bilo koji od signala se može koristiti za određivanje položaja rotacije, samo je potrebno prebrojati pulseve signala. Međutim, za određivanje smjera rotacije, u obzir se moraju uzeti oba signala (istovremeno). Signali su razmaknuti u fazi za 90°, tako da, ako se osovina okreće u smjeru kazaljke na satu, signal A će biti ispred signala B, i obrnuto. Prilikom programiranja, potrebno je pratiti rastuće i padajuće bridove, i ako u određenom trenutku signali A i B imaju suprotne vrijednosti, osovina se okreće u smjeru kazaljke na satu. U suprotnom, ako su vrijednosti signala A i B jednake, osovina se okreće u smjeru suprotnom kazaljci na satu (slika 2.3). [5]



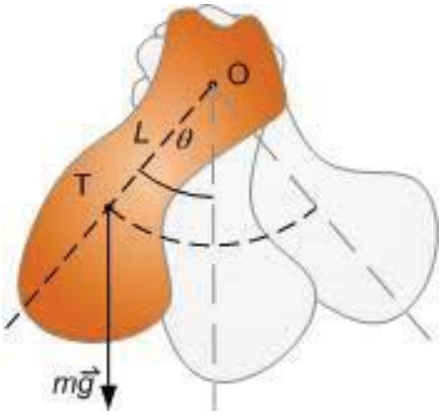
Slika 2.3. Princip rada enkodera.

Enkoder korišten u ovom radu je inkrementalni, njegov signal je pravokutan, ima dvije faze (A i B) te 600 mogućih položaja (Slika 2.4).

Slika 2.4. Digitalni enkoder.

2.5.Fizikalno njihalo

Fizikalno njihalo je svako kruto tijelo mase m koje se njiše oko vodoravne osi koja ne prolazi njegovim težištem (slika 2.5), za razliku od matematičkog njihala, koji se smatra približno idealnim harmonijskim oscilatorom, čija se masa prikazuje točkastom masom.



Slika 2.5. Fizikalno njihalo.

Period titranja fizikalnog njihala iznosi:

$$T = 2\pi\sqrt{\frac{I}{mgL}}$$

(2 - 1)

gdje je I moment tromosti tijela, m masa tijela, a L udaljenost između težišta tijela i njegova ovjesišta.

Bez matematičkog izvoda, iz priručnika [5] će se preuzeti izrazi za periode fizikalnih njihala u obliku štapa, prstena i jednakostraničnoga trokuta koji kasnije biti korišteni za računanje parametara simulacije.

Period titranja njihala u obliku prstena:

$$T = 2\pi\sqrt{\frac{2R}{g}}$$

(2 - 2)

gdje je R polumjer prstena.

Period titranja fizikalnog njihala u obliku jednakostraničnog trokuta:

$$T = 2\pi\sqrt{\frac{3L}{2g\sqrt{3}}}$$

(2 - 3)

gdje je L duljina osnovice.

Period titranja fizikalnog njihala u obliku štapa :

$$T = 2\pi\sqrt{\frac{L^2 + 12r^2}{12\ g\ r}}$$

(2 - 4)

gdje je L duljina štapa, a r udaljenost između težišta štapa i oslonca (osi rotacije).

Zbog lakše izvedbe makete i sučelja programa, obrađivat će se samo slučajevi gdje je štap ovješen o svoj vrh, odnosno $r = \frac{L}{2}$, pa iz jednadžbe (2-4) slijedi :

$$T = 2\pi \sqrt{\frac{L^2 + 12\left(\frac{L}{2}\right)^2}{12\, g \frac{L}{2}}} = 2\pi \sqrt{\frac{L^2 + 3L^2}{6\, g\, L}} = 2\pi \sqrt{\frac{L^2 + 3L^2}{6\, g\, L}}$$

$$T = 2\pi \sqrt{\frac{2\, L}{3\, g}}$$

(2 - 5)

2.5.1. Prigušeno titranje

U stvarnim titrajnim sustavima tijelo će se naći pod utjecajem vanjskih sila i njegova amplituda će se s vremenom smanjivati, a titranje u kojem se amplituda s vremenom postupno smanjuje nazivamo prigušenim titranjem. Dio mehaničke energije se najčešće troši na savladavanje trenja i/ili otpora zraka.

Jednadžba gibanja za prigušeno titranje je :

$$\frac{d^2x}{dt^2} + 2\delta \frac{dx}{dt} + \omega_0 x = 0$$

(2 - 6)

Gdje je ω_0 kružna frekvencija neprigušenog harmoničkog oscilatora, a δ faktor prigušenja. [5]

Rješenje jednadžbe (2-6) (za mala prigušenja $\omega_0 > \delta$) ima oblik:

$$x(t) = Ae^{-\delta t} \sin(\omega t + \varphi_0)$$

(2 - 7)

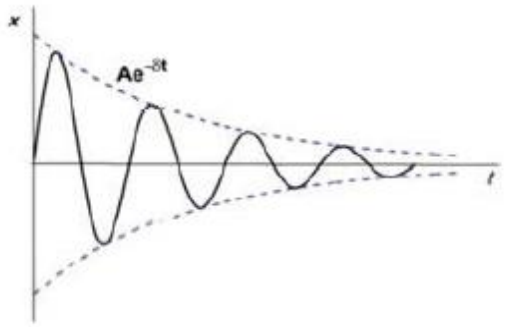
Gdje je $\omega = \sqrt{\omega_0^2 - \delta^2}$ kružna frekvencija prigušenog titranja. [5]

Amplituda s vremenom opada eksponencijalno, a što je faktor prigušenja veći, to se i amplituda brže smanjuje.

Omjer dviju susjednih amplituda kod prigušenog titranja je stalan i iznosi:

$$\frac{A(t)}{A(t+T)} = e^{\delta T}$$

(2 - 8)



Slika 2.6. Grafički prikaz prigušenog titranja.

Logaritmiranjem (s bazom *e*) izraza (2-7) i okretanjem jednadžbe dobivamo izraz za koeficijent prigušenja δ:

$$\ln\left(\frac{A(t)}{A(t+T)}\right) = \delta T$$

$$\delta = \frac{\ln\left(\frac{A(t)}{A(t+T)}\right)}{T}$$

(2 - 9)

Pa je logaritamski dekrement titranja:

$$\lambda = \ln\left(\frac{A(t)}{A(t+T)}\right) = \delta T$$

(2 - 10)

Prigušeni titrajni sustav može se, također, opisati tzv. Q-faktorom, odnosno faktorom dobrote ili kvalitete titrajnog sustava koji je određen relacijom:

$$Q = \frac{\pi}{\delta T}$$

(2 - 11)

Što je faktor dobrote veći, prigušenje je manje i manji je gubitak energije iz titrajnog sustava. U dobrim mehaničkim titrajnim sustavima Q-faktor je reda veličine 10^3 do 10^5.

3. REALIZACIJA MAKETE FIZIKALNOG NJIHALA

3.1. Sklopovski dio makete

Maketa se sastoji od stalka na koji je pričvršćen enkoder. Enkoder je zalemljen na Arduino, dok se Arduino spaja na računalo putem USB priključka. Na osovinu enkodera je stavljen nastavak koji također čini određenu osovinu na koju se mogu prikačiti tijela koja će titrati (slika 3.1.).

Slika 3.1. Stalak na koji je pričvršćen enkoder.

3.2. Programsko sučelje mikroupravljača

Pinovi A i B, objašnjeni u poglavlju 2.4 spojeni su na digitalne *pinove* 18 i 19 Arduino Mega pločice. U setup dijelu funkcije, dodjeljuje se interrupt *pinovima* A i B, koji će na rastući brid pozvati funkcije cntA i cntB, koje "broje" pulseve signala. Funkcije će, kada se pozovu, provjeriti stanje drugog signala, i kao što je objašnjeno u poglavlju 2.4, ovisno o tome je li drugi signal u stanju 1 ili 0, uvećati ili umanjiti brojač za 1 (slika 3.2.).

```
void cntA(){
  if(digitalRead(pin_B)==HIGH) {
    C++;
  }else    C--;
  }
void cntB(){
  if(digitalRead(pin_A)==HIGH) {
    C--;
  }else    C++;
}
```

Slika 3.2. Kod brojača impulsa.

Također je definiran Timer kojemu je dodijeljen interrupt kako bi svakih 5 milisekundi pozivao funkciju getSample. Funkcija getSample postavlja zastavicu u true, te varijabli sample dodjeljuje trenutnu vrijednost brojača, kako bi brojač nesmetano mogao nastaviti brojati. U loop() funkciji, provjerava se je li zastavica postavljena u true, i ako je, putem serijskog priključka poslat će se stanje brojača, a zastavicu vratiti u false (slika 3.3.).

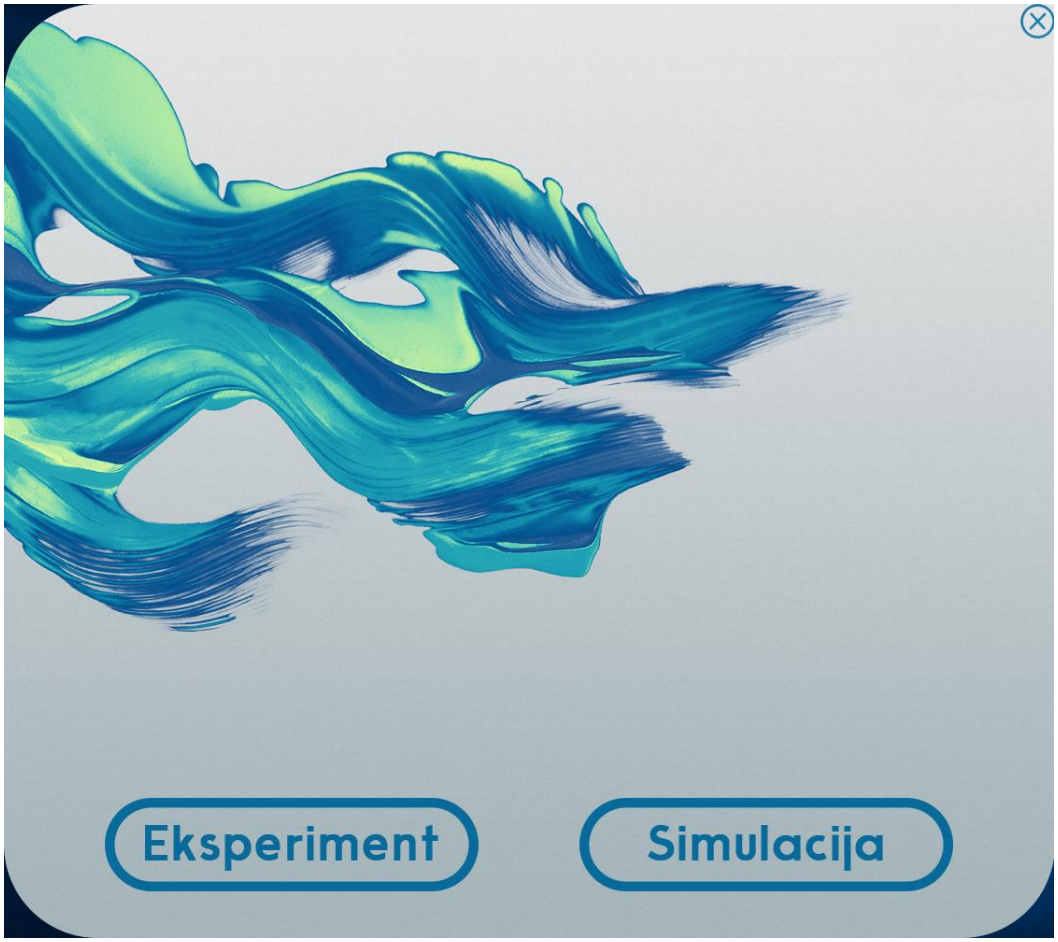
```
void getSample(){
  sample = C;
  bSample = true;
}
void loop() {
  if(bSample) {
    Serial.println(sample);
    bSample = false;
  }
}
```

Slika 3.3. Kod slanja trenutnog stanja brojača putem serial porta.

Nakon jednog punog kruga, stanje brojača će biti 1200, što znači da imamo preciznost od $\frac{360}{1200}$ odnosno 0.3°. Drugim riječima, prilikom primanja podataka na *SerialPortu*, ulazni podatci će se množiti sa 0.3 kako bi se dobilo stvarno stanje njihala u stupnjevima.

3.3. Programsko sučelje stolnog računala

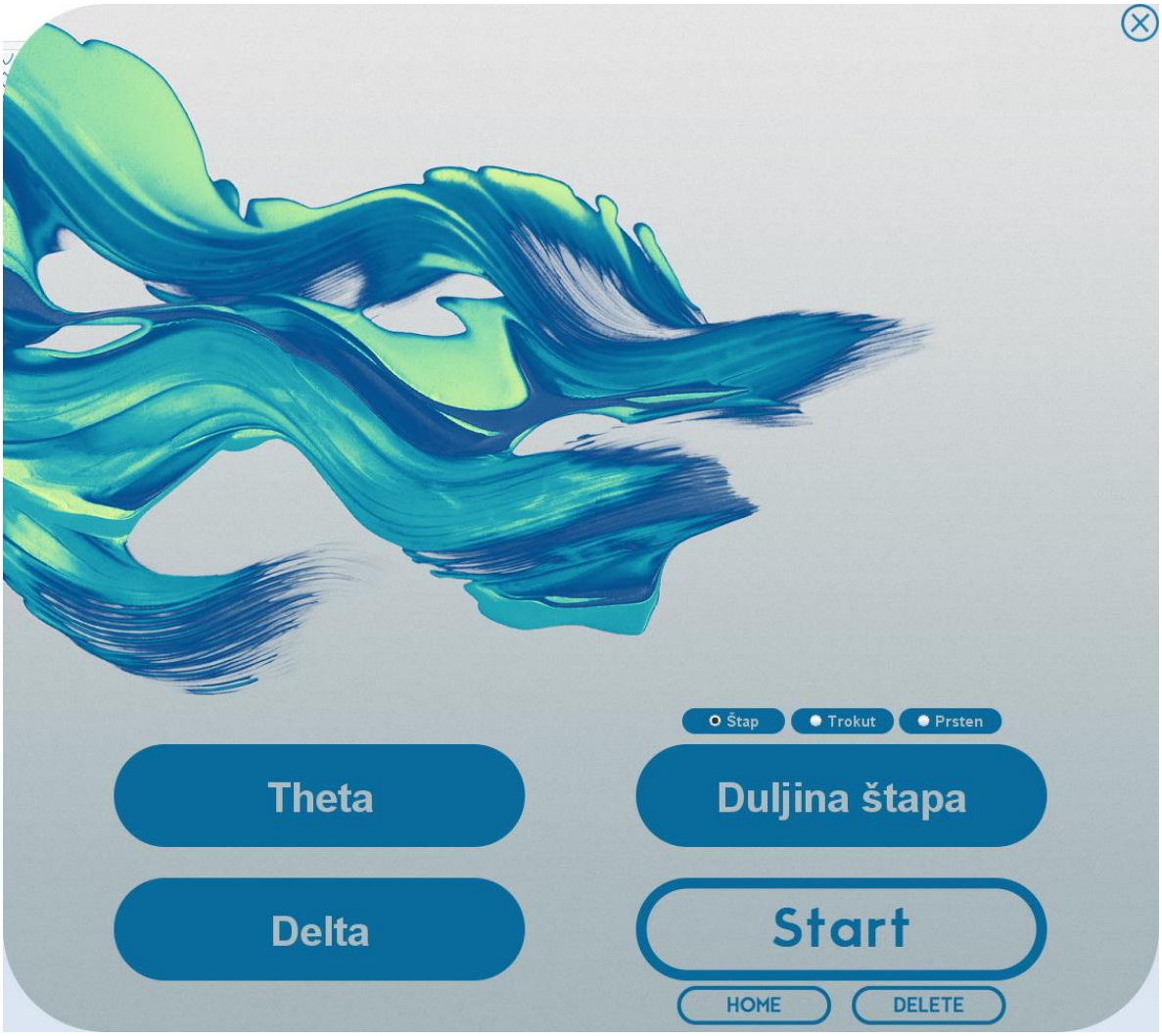
Aplikacija se sastoji od 4 klase: Window, Welcome, Simulacija i Eksperiment. U klasi Window se nalazi main funkcija u kojoj se nalaze svi *MouseListeneri*, te se kreira prozor kao i objekti Welcome, Simulacija i Eksperiment koji nasljeđuju klasu JPanel odnosno svaki objekt je svoj panel na koji se slažu tipke. Prilikom pokretanja aplikacije, vidljiv je samo Window objekt, na kojem se nalaze 2 tipke : eksperiment i simulacija. Klikom na odgovarajuću tipku, prikazuje se istoimeni panel.



Slika 3.4. Početni zaslon aplikacije.

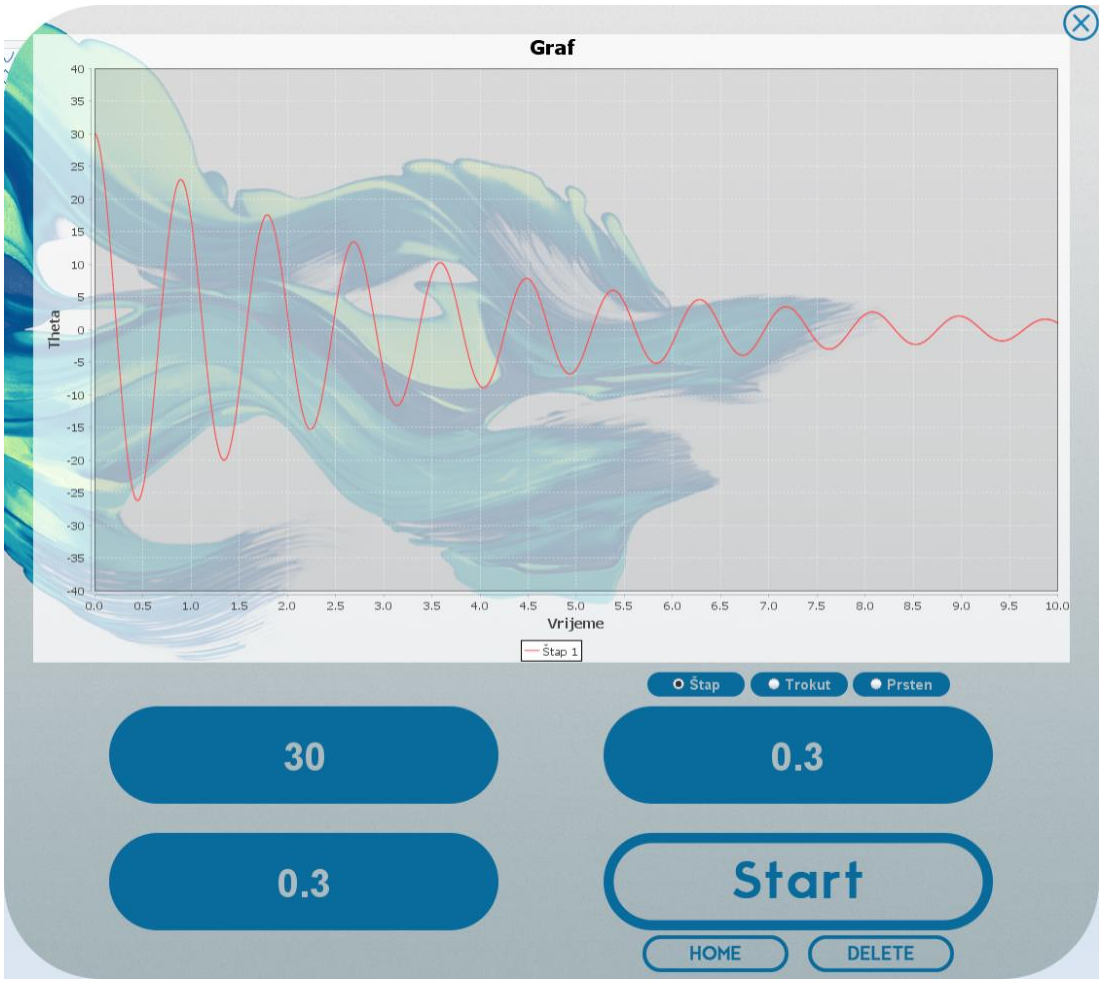
3.3.1. Simulacija

Klikom na “Simulacija”, otvara se novi zaslon (Slika 3.5) koji od korisnika zahtjeva unos sljedećih parametara titranja: Theta (θ, početni kut otklona), Delta (δ, faktor prigušenja) te Dimenzija. Polje Dimenzija će, ovisno o odabranom tijelu putem *RadioButtona* (štap, jednakostraničan trokut, prsten) zahtijevati odgovarajuću mjeru tijela (duljina štapa, duljinu osnovice, promjer prstena).



Slika 3.5. Sučelje simulacijskog dijela programa.

Nakon unošenja parametara, klikom na tipku start, simulacija će započeti. Nastanak grafa je animiran, a simulacija će prikazati vrijeme potrebno da se amplituda smanji na približno 10% početne vrijednosti. Za proračun se koristi jednadžba (2-6), pomoću koje se u koracima od 0.005 računa vrijednost i sprema u objekt *XYSeries*, koji predstavlja niz točaka, koje se odmah nakon izračuna prikazuju na grafu (slika 3.6).



Slika 3.6. Prikaz grafa.

Kako bi dobili privid nastajanja grafa, između svakog izračuna pozove se funkcija *sleep(2)* koja pauzira thread na 2 milisekunde. Varijabla „i“ ide od 0 do 60, u koracima od 0.005 što znači da će jedan graf sadržavati 12 000 točaka, koje neće nužno biti nacrtane, ovisno kada će postići 10% početne vrijednosti. (slika 3.7).

```
Thread simthread = new Thread(){
    @Override public void run(){
        double x;
        startbtn.setVisible(false);
        System.out.println(help);
        for(double i=0; i<60;i=(double) (i+0.005)){
            x=(double) (Theta*Math.exp(-Delta*i)*Math.sin(Math.sqrt(2*Math.PI*2*Math.PI/Period/Period-Delta*Delta)*i+Math.PI/2));
            series.add(i,x);
            if(help>i) {
            try {
                sleep(2);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        }
        startbtn.setVisible(true);
    }
};
simthread.start();
```

Slika 3.7. Thread za iscrtavanje grafa.

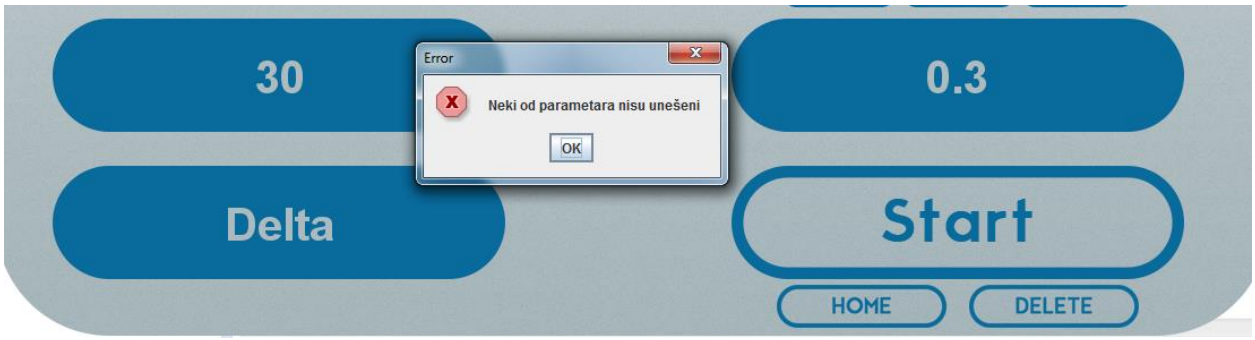
Ovisno o odabranom *RadioButtonu*, za izračun perioda će se koristiti odgovarajuća jednadžba (2-2, 2-3 ili 2-5) koji će se tada predati funkciji koja pokreće već spomenuti thread (slika 3.7.).

U polja se mogu unijeti samo brojevi i decimalna točka. Ako korisnik pokuša unijeti slovo ili neki drugi znak, on neće biti zabilježen već će biti „progutan“. (slika 3.8.)

```
DeltaTextField.addKeyListener(new KeyAdapter(){
    public void keyTyped (KeyEvent e){
        char input = e.getKeyChar();
        if((input<'0'||input>'9')&&input != '\b'&&input !='.'){
            e.consume();
        }
    }
});
```

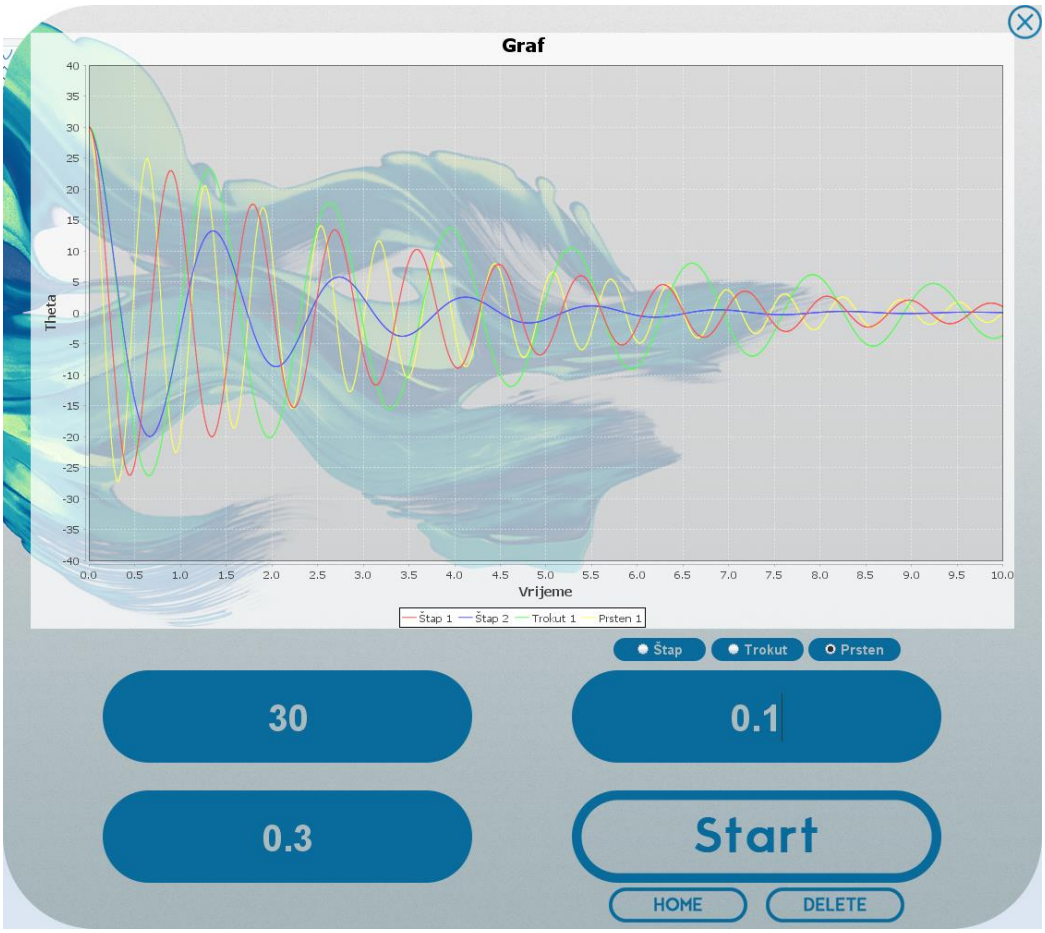
Slika 3.8. Kod za ignoriranje znakova.

Ako se jedan ili više parametara ne unese, a pritisne se tipka Start, program će izbaciti *pop-up* poruku uz naputak da neki od parametara nisu unešeni (slika 3.9).



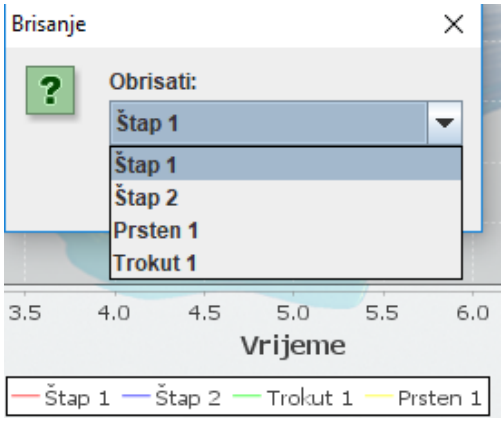
Slika 3.9. Neispravan unos parametara.

Svaki od grafova se sprema u objekt *ArrayListe*, pa istovremeno možemo prikazati onoliko grafova koliko imamo memorije. Nakon što se određeni graf nacrta, parametri se mogu promjeniti i ponovnim pritiskom na tipku Start, kreirati će se novi graf (slika 3.10.).



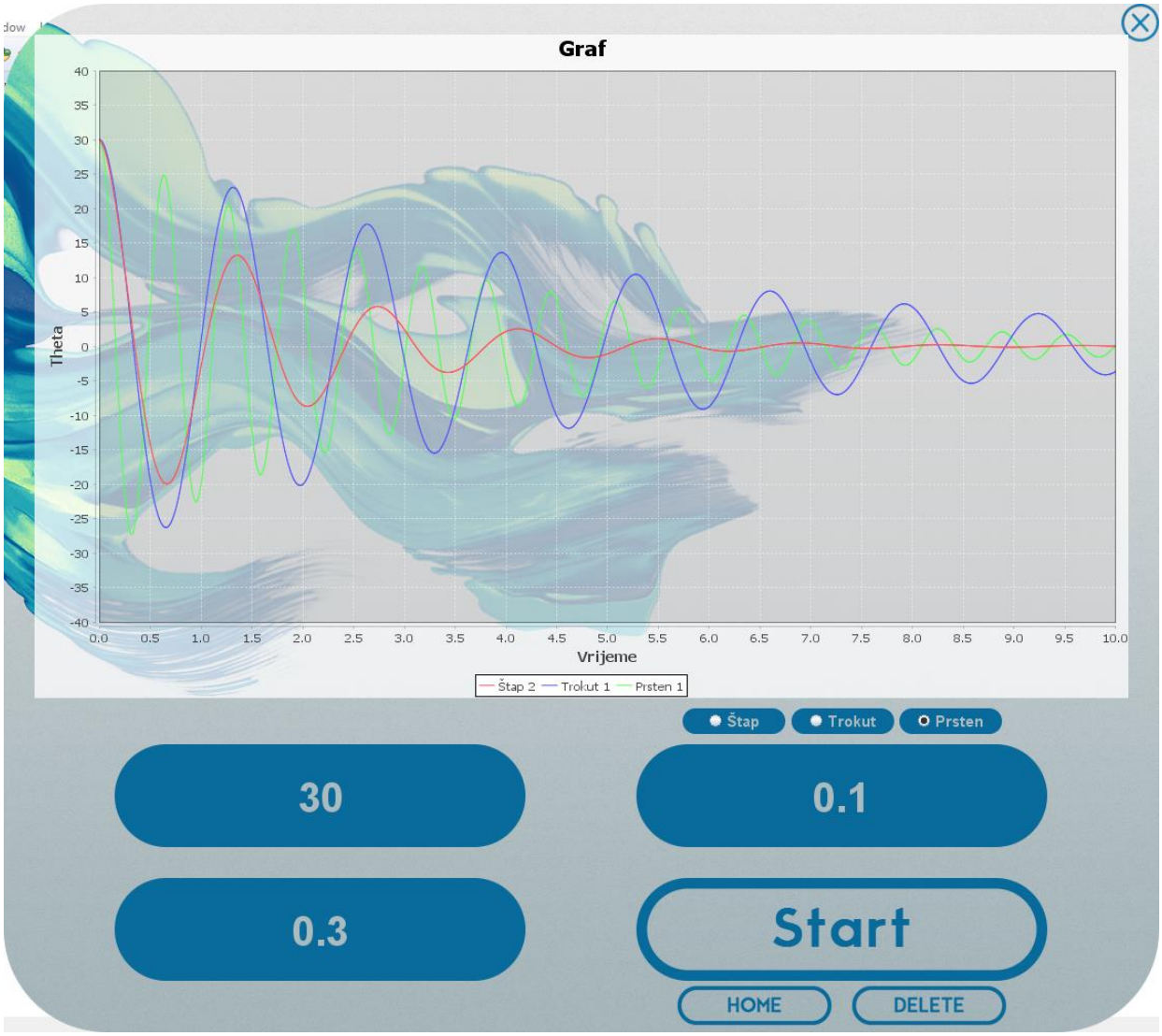
Slika 3.10. Više različitih grafova.

Ispod tipke Start, nalazi se tipka Home koja korisnika vraća na početni zaslon, te tipka Delete, koja omogućava brisanje do sada nacrtanih grafova. Klikom na tipku Delete otvorit će se *pop-up* prozor koji nudi brisanje jednog od postojećih grafova (slika 3.11.). Ukoliko se predomislimo i ne želimo obrisati graf, možemo odabrati tipku Cancel, no ako i dalje želimo obrisati graf, u ovom primjeru Štap 1, odabiremo ga iz padajućeg izbornika te pristinemo tipku OK.



Slika 3.11. Padajući izbornik za brisanje.

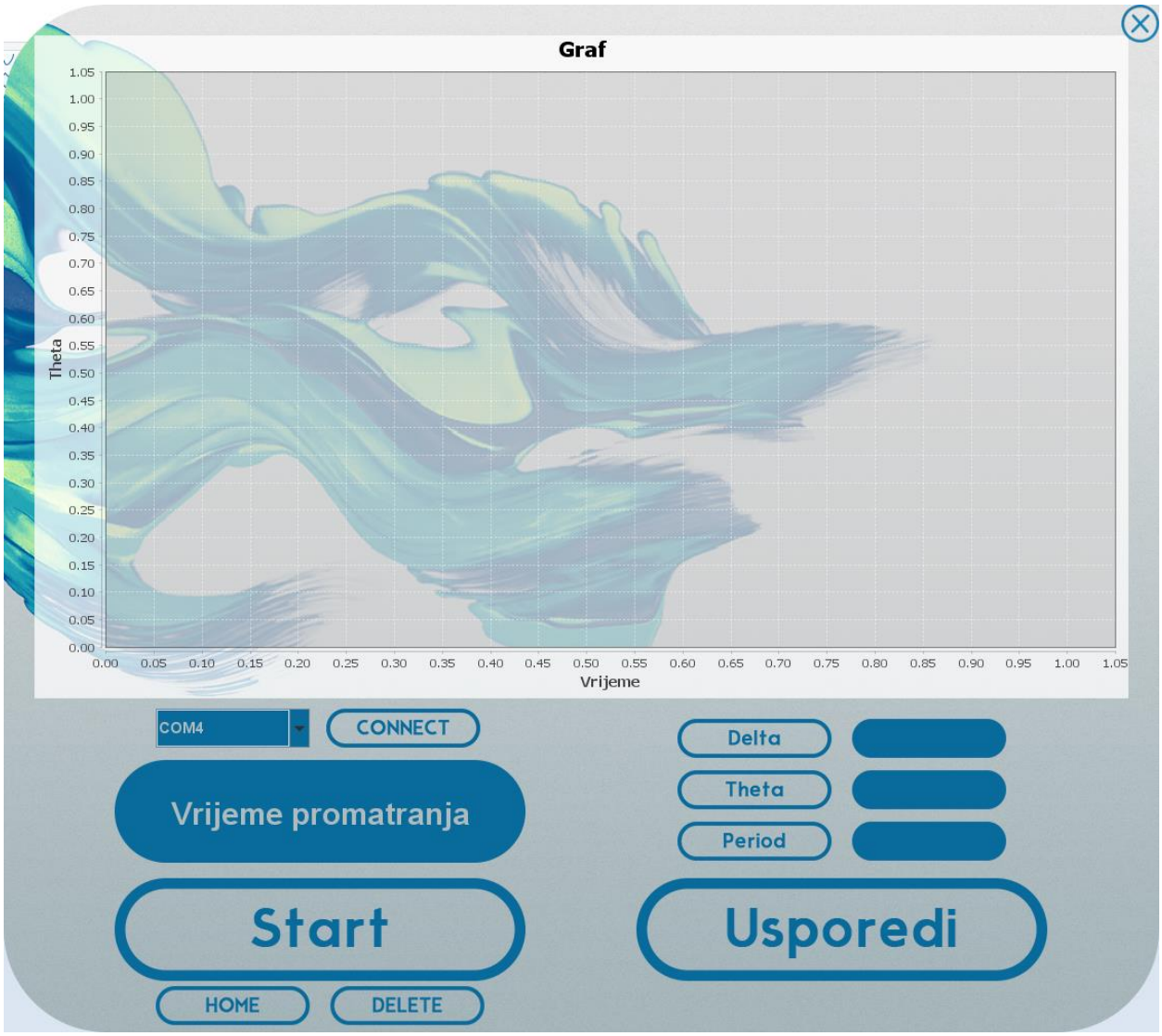
Odabrani objekt se tada uspješno uklanja s grafa, te se nesmetano može nastaviti dodavati nove ili brisati stare grafove (slika 3.11.).



Slika 3.12. Uspješno obirsan graf.

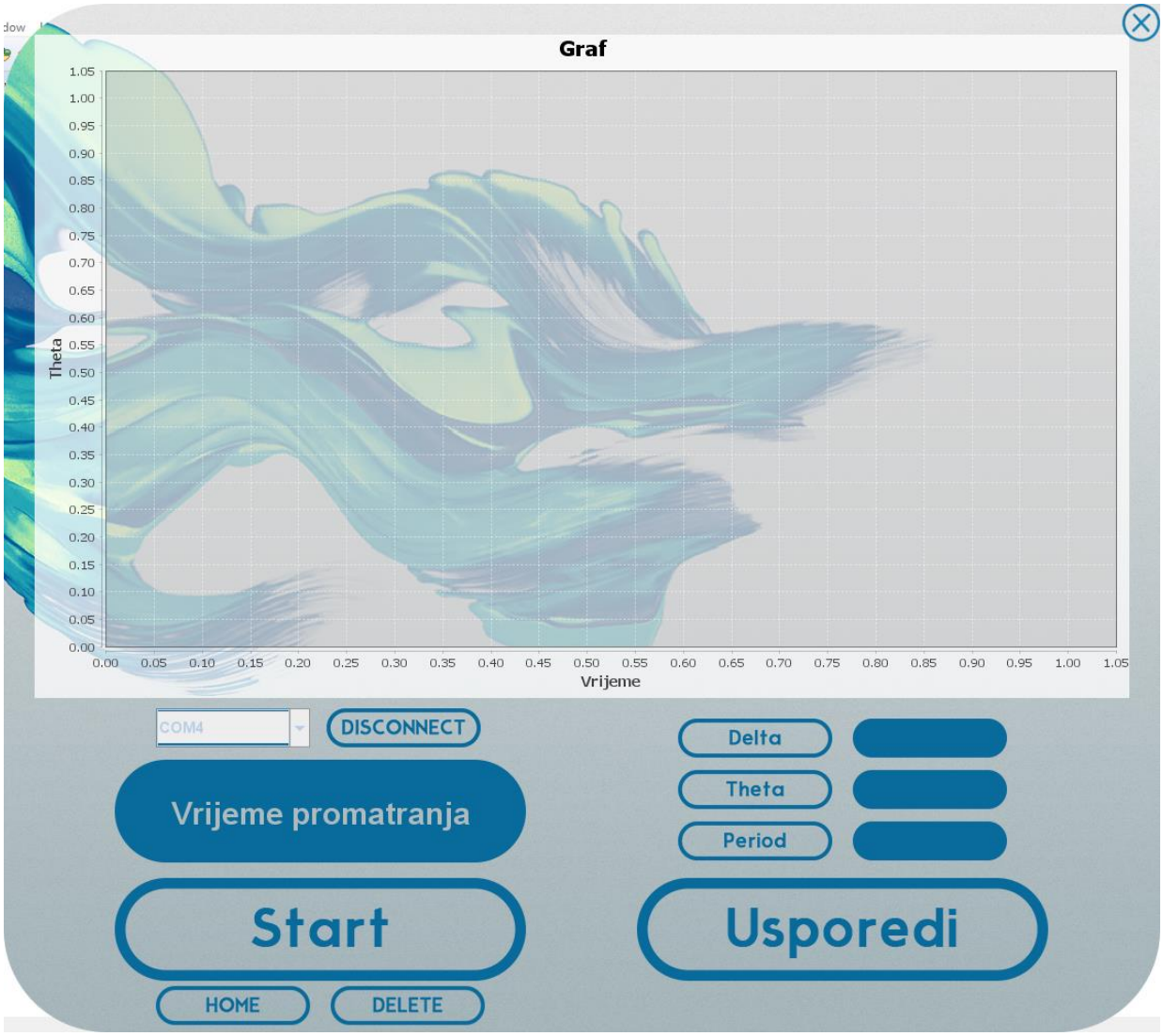
3.3.2. Eksperiment

Odabiranjem tipke „Eksperiment“ sa početnog zaslona, pokazuje se zaslon sa slike 3.13.



Slika 3.13. Sučelje eksperimentanog dijela programa.

Na zaslonu se od korisnika zahtjeva da u padajućem izborniku odabere odgovarajući *port* na kojem se nalazi enkoder spojen putem USB kabla te pritisne tipku Connect. Ako je spajanje uspješno, tipka Connect će biti zamijenjena tipkom Disconnect (slika 3.14), a padajući izbornik će se zaključati. Nakon toga korisnik može unijeti vrijeme promatranja u istoimeno polje, ako želi da se na grafu x-os postavi na točno određeni maksimalni iznos do kojega će promatrati titranje. Ako korisnik ne unese ništa, ili unese 0, x-os će se automatski prilagođavati, odnosno graf će nastajati u stvarnom vremenu skroz dok korisnik ne pritisne tipku Stop (koja će zamijeniti tipku Start nakon što tipka Start bude pritisnuta) ili Disconnect.



Slika 3.14. Uspješno spajanje s Arduinom.

Pritiskom na tipku start, pokreće se thread koji „sluša“ odabrani *port* te crta graf, no dobiveni podatci neće odmah biti prikazivani na grafu. Pretpostavlja se, da će se u trenutku pritiska na tipku Start tijelo koje treba titrati nalaziti u ravnotežnom položaju, stoga nema smisla na grafu prikazivati ravnu crtu već da graf krene nastajati od onog trenutka kada tijelo počne titrati. Iz tog razloga je implementirano „softversko rješenje" koje u 4 *while* petlje rješava taj problem. Prva *while* petlja se ponavlja skroz dok je stanje na ulazu 0. Kako bi se izbjegli neželjeni vanjski utjecaji koji mogu tijelo pomaknuti iz ravnotežnog položaja za tih 0.3°, u sljedećoj *while* petlji se čeka da se tijelo izvuče za više od 10° (ili -10°), jer se pretpostavlja da će u određenom trenutku izvođač eksperimenta odlučiti izvući tijelo iz ravnotežnog položaja. Nakon što je tijelo izvučeno iz ravnotežnog položaja, odnosno postignut je kut veći od 10°, *while* petlja čeka da se primi vrijednost 0, odnosno da je tijelo pušteno da titra i da je prošlo kroz ravnotežni položaj. Nakon primanja nule, prelazi se u zadnju *while* petlju. Naime, sada je tijelo prošlo kroz ravnotežni položaj

(0) i vrijednost kuta se povećava, skroz dok se ne postigne maksimum, i od tog trenutka je poželjno da se na zaslonu krene prikazivati graf. Tako zadnja *while* petlja uspoređuje zadnju i predzadnju vrijednost koja je primljena na *portu*, i prvi puta kada zadnja vrijednost bude manja od predzadnje, izlazi se iz zadnje *while* petlje i kreće se crtati graf (slika 3.15.).

```
while(RP) {
    Scanner scannerhelp = new Scanner (chosenPort.getInputStream());
    String help = scannerhelp.nextLine();
    try {number = Double.parseDouble(help);}
    catch (NumberFormatException e) {e.printStackTrace();}
    if (number!=0) {
        RP=false;
        scannerhelp.close();
    } //izašao iz ravnoležnog položaja
}
while(GreaterThan10==false) {
    Scanner scannerhelp1 = new Scanner (chosenPort.getInputStream());
    String help = scannerhelp1.nextLine();
    try {number = Double.parseDouble(help);}
        catch (NumberFormatException e) {e.printStackTrace();}
    if (number*0.3>10 || number*0.3<-10) {
        GreaterThan10=true;
        scannerhelp1.close();
        scannerhelp1.close();
    } //postigao kut veci od 10°
}
while(IsZero==false)  {
    Scanner scannerhelp2 = new Scanner (chosenPort.getInputStream());
    String help = scannerhelp2.nextLine();
    try {number = Double.parseDouble(help);}
        catch (NumberFormatException e) {e.printStackTrace();}
    if (number<6 && number>-6) {
        IsZero=true;
        scannerhelp2.close();
    } // ponovni prolazak kroz ravnotezni polozaj
}
while(IsMax==false) {
    Scanner scannerhelp3 = new Scanner (chosenPort.getInputStream());
    String help = scannerhelp3.nextLine();
    try {number = Double.parseDouble(help);}
        catch (NumberFormatException e) {e.printStackTrace();}
    if(number<0) number=number*(-1);
    max=number;
    if(max>maxhelp) {
        maxhelp=max;
    }
    if(maxhelp>number) {
        IsMax=true;
        scannerhelp3.close();
    } //postignut maksimum i krece crtanje grafa
```

Slika 3.15. Četiri while petlje.

Nakon pritiska na tipku Stop, program će temeljem nacrtanog grafa izračunati koliko iznose period, koeficijent prigušenja i početni kut, te ih ispisati na zaslon. Program računa period i koeficijent prigušenja na sljedeći način : u *for* petlji, koja ide od nultog do posljednjeg elementa *XYSeriesa*, imamo 2 zastavice koje okidaju hoće li se izvesti *if* (uvjet za traženje pozitivnog maksimuma odnosno amplitude) ili *else* (uvjet za traženje negativnog maksimuma odnosno amplitude). Također postoje 2 niza : nizAmplituda i nizVremena, koji će sadržavati jednak broj elemenata, a u njih će se spremati vrijednost amplitude te odgovarajući trenutak kada se ta amplituda postigla. Prije samog ulaska u petlju, provjerava se je li nulti element veći ili manji od nule (pozitivan ili negativan) jer znamo da je nulti element prva amplituda. Ovisno o prvom elementu, zastavica će se postaviti u true ili false te će se nulti element dodati u niz (slika 3.16.).

```
ExpTheta = (double) nizExp.get(nizExp.size()-1).getY(0);
if(ExpTheta>0) {
    nizAmplituda.add(ExpTheta);
    nizVremena.add((double) 0);
}
else {
    TraziMax=true;
    nizAmplituda.add(ExpTheta);
    nizVremena.add((double) 0);
```

Slika 3.16. Provjeravanje prvog elementa niza.

Nakon toga se u *for* petlji traže amplitude, gdje se naizmjenično traži najveća pa najmanja vrijednost (ili obrnuto, ako je prvi element bio negativan) te kada se amplituda nađe, stanje zastavice se promjeni u suprotno, jer se zna da kod titranja pozitivne i negativne vrijednosti dolaze naizmjenično. Amplituda se prepoznaje tako što se gleda je li i-ti element veći (za pozitivnu) ili manji (za negativnu) od (i+1)-tog elementa i ako je, znači da se vrijednosti počinju smanjivati(odnosno povećavati) te da se radi o maksimalnoj (minimalnoj) vrijednosti i tu vrijednost sprema u nizAmplituda, dok se pripadajuća X komponenta sprema u nizVremena (slika 3.17.).

```
for(int i = 0; i < nizExp.get(nizExp.size()-1).getItemCount()-1;i++) {
    if(TraziMax) {
        if((double)nizExp.get(nizExp.size()-1).getY(i)>(double)nizExp.get(nizExp.size()-1).getY(i+1)) {
            TraziMax=false;
            nizAmplituda.add((double) nizExp.get(nizExp.size()-1).getY(i));
            nizVremena.add((double) nizExp.get(nizExp.size()-1).getX(i));
        }
    }
    else {
        if((double)nizExp.get(nizExp.size()-1).getY(i)<(double)nizExp.get(nizExp.size()-1).getY(i+1)) {
            TraziMax=true;
            nizAmplituda.add((double) nizExp.get(nizExp.size()-1).getY(i));
            nizVremena.add((double) nizExp.get(nizExp.size()-1).getX(i));
        }
    }
}
```

Slika 3.17. Traženje amplituda.

Period se računa kao srednja vrijednost svih vremena između pojedinih amplituda, odnosno zadnju vrijednost u nizVremena podijelimo s brojem elemenata polja manje 1 zbog prvog elementa koji predstavlja vrijednost amplitude u početnom trenutku i onda podijeli još s dva, jer se jedan period nalazi između 2 pozitivne, ili negativne amplitude. Koeficijent prigušenja računa se prema jednadžbi (2-8) tako što se iz nizAmplituda svaka i-ta vrijednost podijeli sa (i+2)-tom vrijednošću te doda u sumu, a brojač poveća za 1, dok se ne dođe do kraja. Nakon toga se suma podijeli s brojačem koja rezultira srednjom vrijednošću prigušenja (slika 3.18).

```
ExpPeriod=nizVremena.get(nizVremena.size()-1)/brojAmplituda; //broj amplituda je ranije podijeljen s 2

PeriodTFE.setText(Float.toString((float)ExpPeriod));

brojAmplituda=0;
for(int i=0;i<nizAmplituda.size()-2;i=i+2) {
    ExpDeltaSum =ExpDeltaSum+ Math.log(nizAmplituda.get(i)/nizAmplituda.get(i+2))/ExpPeriod;
    brojAmplituda++;
}
ExpDelta=ExpDeltaSum/brojAmplituda;
DeltaTFE.setText(Float.toString((float)ExpDelta));
```

Slika 3.18. Izračunavanje koeficijenta prigušenja i perioda.

Ako želimo provjeriti točnost našeg mjerenja, klikom na tipku Usporedi (slika 3.14), automatski će se funkciji za simulaciju predati izračunati podatci prikazani na ekranu i simulirani graf će se prikazati uz eksperimentalni. Primjer će biti pokazan u sljedećem poglavlju.

Isto kao i kod „Simulacije“, ispod tipke Start imamo tipke Home i Delete, koje imaju istu funkcionalnost kao one opisane u dijelu „Simulacija“.

4. PROVEDBA EKSPERIMENTA I USPOREDBA S TEORIJSKIM VRIJEDNOSTIMA

Prilikom izvođenja eksperimenta potrebno je obratiti pozornost i postaviti tijelo u ravnotežni položaj, jer enkoder šalje podatke relativno u odnosu na položaj u kojem se nalazio u trenutku dobivanja struje (u tom trenutku brojač se postavlja na 0).

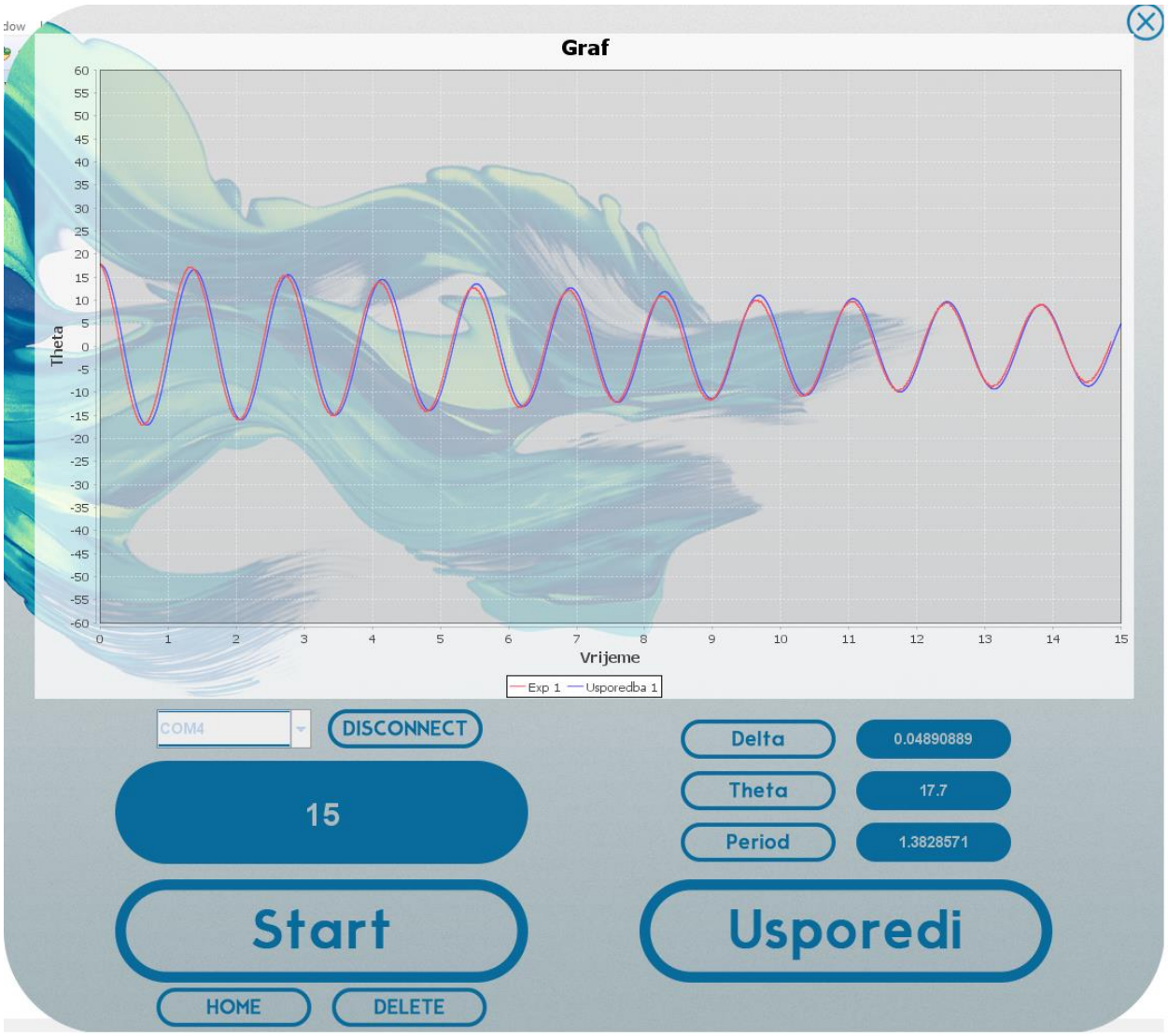
Za provjeru točnosti aplikacije, okačit će se štap duljine 71.5 cm (slika 4.1).

Slika 4.1. Eksperimentalna postava.

Period štapa prema jednadžbi (2-5) iznosi :

$$T = 2\pi\sqrt{\frac{2}{3}\times\frac{0.715}{9.81}} = 1.385\ s$$

Nakon uspješnog spajanja s Arduinom te odabira vremena simulacije 15 sekundi, pristiska na tipku Start te puštanja štapa da titra, dobiti će se određeni graf koji prikazuje eksperimentalno titranje postavljenog štapa. Pristiskom na tipku usporedi, izračunati parametri će se predati funkciju za simulaciju, te će se na zaslonu „simulirati“ graf s predanim podatcima (slika 4.2).



Slika 4.2. Rezultati eksperimenta.

Sa slike 4.2 možemo primijetiti gotovo savršeno preklapanje crvenog (eksperimentalnog) i plavog (simuliranog/teorijskog) graf. Također iznos teorijskog perioda je jako blizu eksperimentalnom iznosu. Dobiveno prigušenje nije veliko no ipak je s grafa vidljivo opadanje vrijednosti amplituda.

Koeficijent dobrote (jednadžba 2-11) za ovaj eksperiment iznosi :

$$Q = \frac{\pi}{\delta T} = \frac{\pi}{0.0489 \times 1.382} = 46.487$$

5. ZAKLJUČAK

U prvom dijeli ovog rada opisan je princip rada enkodera te teorijska podloga fizikalnih zakona korištenih/promatranih u ovom radu. Korišteni enkoder ima osjetljivost od 0.3° što je omogućilo jako precizno određivanje položaja njihala.

Zahvaljujući dodacima znatno je olakšano manipuliranje podacima te njihovo grafičko prikazivanje (JFreeChart) te povezivanje Arduina s računalom (JSerialComm).

Aplikacija je uspješno realizirana usprkos problemima s kojima se suočilo i koji su iziskivali dobro promišljanje za njihovo otklanjanje. Neki od njih su : računanje perioda i koeficijenta prigušenja iz jako dugačkog niza X i Y koordinata, manipuliranje s više različitih grafova te crtanje grafa tek od onog trenutka kada njihalo počne titrati. U prilog tomu ide obavljeno eksperimentalno mjerenje u kojem je dobiveno skoro identično poklapanje eksperimentalnog i teorijskog grafa. Eksperimentalni period je također jako blizu teorijskog, pa iz navedenog se može zaključiti da je trenje osovine veoma zanemarivo te da je i sam enkoder dobro napravljen.

U budućnosti bi se moglo dodatno proučiti mogućnost automatskog prilagođavanju grafa koje nisu najbolje opisane u dokumentaciji JFreeCharta. Također preciznost od 0.3° se može smatrati dovoljnom, iako za korake od 5 milisekundi bi veća preciznost enkodera dobro došla, jer proučavanjem podataka koje enkoder šalje, često se zna naći nekoliko istih vrijednosti, no u ovome slučaju je li amplituda postignuta 5, 10 ili 15 milisekundi prije ili kasnije ima zanemariv utjecaj. Trenutno vrijeme simulacije je 60 sekundi odnosno u 1 *XYSeries* objekt se spremi 12 000 točaka, što je za neke veće koeficijente prigušenja previše, a za manje premalo. Stoga, postoji mogućnost implementacije nekakvog dinamičkog dodjeljivanja vremena simulacije u ovisnosti o koeficijentu prigušenja.

LITERATURA

[1] „Arduino“ https://www.arduino.cc/en/Guide/Introduction , pristup ostvaren 28.6.2017

[2] „Arduino Mega“ https://www.arduino.cc/en/Main/arduinoBoardMega2560/ , pristup ostvaren 28.6.2017

[3] „Eclipse“ http://www.eclipse.org/org/ pristup ostvaren 9.9.2017

[4] http://howtomechatronics.com/tutorials/arduino/rotary-encoder-works-use-arduino/ pristup ostvaren 29.6.2017.

[5] Željka Mioković, Fizika 1, Priručnik iz laboratorijskih vježbi

[6] http://www.jfree.org/jfreechart/api/javadoc/index.html , pristup ostvaren 15.6.2017

[7] http://fazecast.github.io/jSerialComm/ , pristup ostvaren 4.9.2017

SAŽETAK

Cilj ovog završnog rada bila je izrada desktop aplikacije koja će na ekranu prikazati kutnu elongaciju fizikalnog njihala u ovisnosti o vremenu i time studentima omogućiti lakše praćenje i stjecanje znanja na području fizikalnog njihala. Teorijski dio obuhvaća podatke o Arduinu i Eclipse razvojnom okruženju uz objašnjavanje principa rada korištenog enkodera i fizikalnih zakona koje će biti korišteni ili promatrani. Aplikacija se sastoji od dva dijela : „Eksperiment“ i „Simulacija“ gdje se ovisno o odabiru prikazuje eksperimentalni graf temeljem podataka koji se primaju s Arduina ili simulira graf temeljem predanih parametara titranja. Paralelno se može prikazati više i eksperimentalnih i simulacijskih grafova što omogućava lagano uspoređivanje sličnosti/različitosti. Desktop aplikacija je napravljena u Java programskom jeziku.

Ključne riječi : Java, Eclipse, Arduino, Fizikalno njihalo, Enkoder

ABSTRACT

The purpose of this final thesis was creating a desktop application which displays dependency of physical pendulum angular elongation in time to make it easier for students to learn about physical pendulums. Theoretical part includes information about Arduino and Eclipse development environment, explanation of rotary encoder working principle and theoretical background of physics used in this experiment. Application consists of 2 parts : „Experiment“ and „Simulation“ where graph is drawn, depending on which one is selected. It will either draw an experimental graph from the data that Arduino has sent or the theoretical graph from the imputed values of pendulum parameters. There can be multiple graphs at the same time which allows easy comparison of differences/similarities. The application was made in Java programming language.

Key words : Java, Eclipse, Arduino, Physical pendulum, Rotary encoder

ŽIVOTOPIS

Siniša Stanić rođen je u Našicama. U Tenju završava Osnovnu školu „Tenja“, te 2010. upisuje Opću gimnaziju u Osijeku. 2014. upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku, smjer računarstvo.

PRILOG A. Program za Arduino

```
#define  SAMPLE_TIME_US    5000
#define  BAUD_RATE       115200
#define pin_A  18
#define pin_B  19
#include <TimerOne.h>
volatile int C, sample;
volatile bool bSample = false;
void setup() {
  Serial.begin(BAUD_RATE);
  pinMode(pin_A, INPUT_PULLUP);
  pinMode(pin_B, INPUT_PULLUP);
  Timer1.initialize(SAMPLE_TIME_US); // us period (sampling time)
  Timer1.attachInterrupt(getSample);
  attachInterrupt(digitalPinToInterrupt(pin_A),cntA,RISING);
  attachInterrupt(digitalPinToInterrupt(pin_B),cntB,RISING);
}
void cntA(){
  if(digitalRead(pin_B)==HIGH) {
    C++;
  }else
    C--;
}
void cntB(){
  if(digitalRead(pin_A)==HIGH) {
    C--;
  }else
    C++;
}
void getSample(){
  sample = C;
  bSample = true;
}
void loop() {
  if(bSample) {
    Serial.println(sample);
    bSample = false;
  }
}
```