Računalo u eksperimentu

Barbančić, Marko

Master's thesis / Diplomski rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:076710

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-08-07**

Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

Marko Barbančić

RAČUNALO U EKSPERIMENTU

Diplomski rad

Zagreb, 2017.

SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO-MATEMATIČKI FAKULTET
FIZIČKI ODSJEK

SMJER: FIZIKA I INFORMATIKA - SMJER NASTAVNIČKI

Marko Barbančić

Diplomski rad

Računalo u eksperimentu

Voditelj diplomskog rada: izv. prof. dr. sc. Mario Basletić

Ocjena diplomskog rada: _____

Povjerenstvo: 1. _____

2. _____

3. _____

Datum polaganja: _____

Zagreb, 2016.

Sažetak

U ovom radu ćemo koristiti Arduino, model Leonardo, kako bi nam olakšao prikupljanje podataka tijekom izvođenja eksperimenta. Za prikupljanje podataka bit će nam potreban i nekakav senzor. Osim senzora potrebno je i napisati program na Arduinu koji će mu dati do znanja koje podatke da prikupi, koliko često da ih prikuplja te gdje i kad da ih šalje kad ih je prikupio. Na drugom kraju, kad smo te podatke primili, trebamo nešto s njima i napraviti. Za to nam služi računalo na kojem ćemo napisati program da dobivene podatke prikupi, obradi i prikaže u grafičkom sučelju te kad smo gotovi s eksperimentom obrađene podatke spremi u trajnu memoriju. U našem slučaju odlučili smo se usredotočiti na ultrazvučni senzor pomoću kojeg ćemo mjeriti udaljenost. Mjerenje udaljenosti ćemo iskoristiti kako bi promatrali gibanje utega, prvo kao njihanje na opruzi, a zatim kao osciliranje na niti. Razlog zašto smo se odlučili za ovakav izbor senzora i eksperimenta jest jednostavnija kontrola varijabli koja će nam omogućiti bolji pregled mogućnosti samog Arduina i njegovih granica. Tijekom provođenja eksperimenta također ćemo pripaziti na njegovu izvedivost u učionici unutar okvira nastavnog sata i na razumljiv prikaz sadržaja učenicima. Iz tog je razloga također titranje optimalan izbor.

Computerized experiments

Abstract

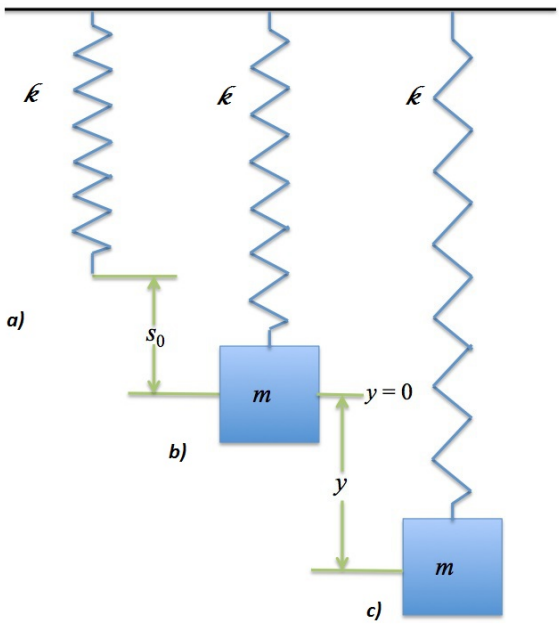
In this thesis we will be using the Arduino, model Leonardo, to facilitate the gathering of data during our experiment. To gather the data we will also be needing a sensor. It is also necessary to write a program on the Arduino to specify which data needs to be collected, how often it needs to be collected and where and when to send the data once it has been gathered. On the other end, once we've received our data, we need to do something with it. For that purpose we are using a computer on which we will write a program to gather, process and present the data in a graphical user interface. Once the experiment has been finished the program will store the processed data in the computers' permanent memory. In our case we decided to focus on the ultrasonic sensor with which we will be measuring distance. The distance measurement will be used to observe the movement of a weight, first as oscillations on a spring, and afterwards as swinging on a thread. The reason we opted for this particular choice of sensor and experiment is the ease of variable control which will give us a better overview of the Arduinos' possibilities and its limitations. During the experiment we will pay attention to the experiments' feasibility in the classroom during a lesson and to the understandable display of the content to the students. For that reason as well oscillations are the optimal choice.

Sadržaj

1 Uvod

Kao praktični primjer korištenja računala u eksperimentu koristit ćemo ultrazvučni senzor za mjerenje udaljenosti spojen na Arduino. Arduino je spojen na računalo preko serijskog porta. Takav postav nam služi kao mjerni instrument. Za eksperimentalni postav koristit ćemo stalak na kojem je obješena opruga na kojoj visi uteg.

1.1 Opruga



Slika 1.1: Stanja opruge: a) opruga u ravnotežnom stanju bez utega; b) opruga u ravnotežnom stanju s obješenim utegom; c) opruga pomaknuta iz ravnotežnog stanja

Uzmimo da je konstanta elastičnosti naše opruge k. Silu opruge opisujemo Hookeovim zakonom:

$$\vec{F} = -k\vec{x} \tag{1.1}$$

gdje je $\vec{F}$ ukupna sila na uteg, k konstanta opruge i $\vec{x}$ pomak iz ravnotežnog položaja. Hookeov zakon vrijedi za područje linearnosti opruge, tj. za male amplitude. Kad na oprugu objesimo uteg mase m na njega djeluju dvije sile, gravitacijska sila i sila opruge:

$$F = -mg - kx \tag{1.2}$$

Ako uzmemo slučaj ravnoteže (slika 1.1, slučaj b) u kojem je zbroj svih sila $F = 0$ i za koji smo uzeli da je $x = s_0$ ostaje nam relacija

$$ks_0 = -mg \tag{1.3}$$

$$s_0 = -\frac{k}{mg} \tag{1.4}$$

Sljedeći slučaj (slika 1.1, slučaj c) je kada uteg pomaknemo iz ravnotežnog položaja. S obzirom na to da jednadžba glasi

$$m\ddot{x} = -mg - kx, \tag{1.5}$$

nakon uvrštavanja $x = s_0 + y$, gdje je y odmak od ravnotežnog položaja s_0, dobiva se

$$m\ddot{y} = -mg - ks_0 - ky \tag{1.6}$$

Ovdje smo iskoristili činjenicu da je $a = \ddot{x} = \ddot{y}$ pošto je derivacija konstante jednaka nuli. Kao što slijedi iz jednadžbe 1.3 članovi mg i ks_0 nam se ponište te dobivamo:

$$m\ddot{y} = -ky \tag{1.7}$$

Ovu jednadžbu možemo drugačije zapisati kao

$$\ddot{y} + \omega_0^2 y = 0 \tag{1.8}$$

ako napravimo zamjenu

$$\omega_0^2 = \frac{k}{m}. \tag{1.9}$$

Rješenje ovakve linearne diferencijalne jednadžbe glasi:

$$y(t) = A\sin(\omega_0 t + \phi) \tag{1.10}$$

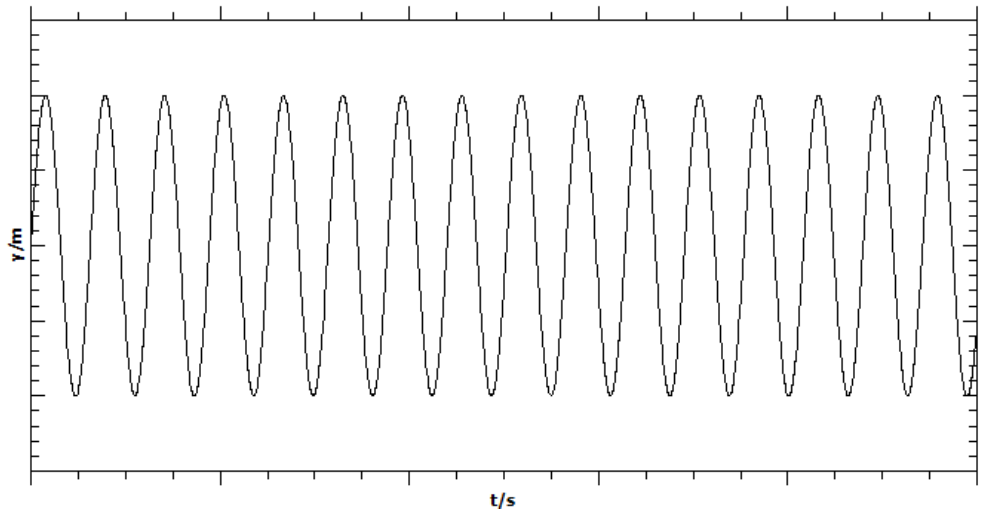
gdje je A amplituda, ω_0 kružna frekvencije titranja i ϕ pomak u fazi (vidi sliku 1.2).

Iz ω_0 možemo izračunati frekvenciju

$$f = \frac{\omega_0}{2\pi} \tag{1.11}$$

te period titranja

$$T = \frac{1}{f} = \frac{2\pi}{\omega_0} \tag{1.12}$$



Slika 1.2: Titranje opruge; grafički prikaz rješenja 1.10

U realnom slučaju postoji i gušenje te se, u slučaju kada je trenje proporcionalno s brzinom, oblik jednadžbe iz 1.8 mijenja u:

$$\ddot{y} + 2\gamma\dot{y} + \omega_0^2 y = 0, \tag{1.13}$$

gdje je intenzitet trenja opisan koeficijentom γ. Još općenitiji slučaj oscilatora je tjerani oscilator za koji vrijedi sljedeća diferencijalna jednadžba:

$$\ddot{y} + 2\gamma\dot{y} + \omega_0^2 y = A\sin\omega t. \tag{1.14}$$

Član na desnoj strani odgovara vanjskoj periodičnoj sili.

1.2 Arduino Leonardo

Arduino Leonardo je mikrokontrolerska ploča temeljena na ATmega32u4 mikrokontroleru. Ona ima 20 digitalnih ulazno/izlaznih pinova od kojih 7 se mogu koristiti kao PWM (modulacija širine impulsa) izlazi i 12 kao analogni ulazi, 16 MHz kristalni oscilator, micro USB priključak, ulaz za napajanje i tipku za resetiranje. Može se napajati preko USB kabela ili vanjskim napajanjem (baterija ili adapter).

Leonardo se razlikuje od prethodnih Arduina po tome što ATmega32u4 ima ugrađenu USB komunikaciju, eliminirajući potrebu za sekundarnim procesorom. To omogućuje Leonardu da se prijavi na spojeno računalo kao miš i tipkovnica, uz virtualni (CDC) serijski/COM port.

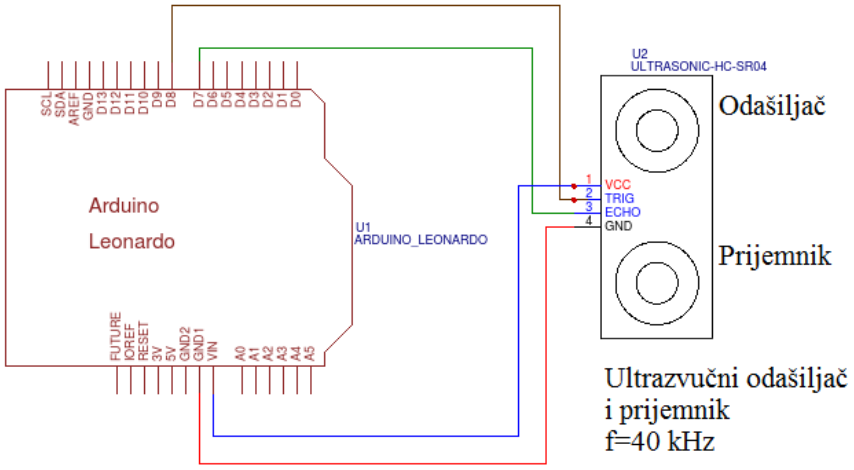
Za određivanje položaja utega koristimo dodatni elektronički sklop s odašiljačem i prijemnikom ultrazvuka("ultrazvučni senzor") spojenim na Arduino s četiri žice (slika 1.3). Princip rada zasniva se na tome da Arduino u određenom trenutku inicira emitiranje kratkog ultrazvučnog impulsa (odašiljač) te nakon toga mjeri vrijeme potrebno da taj puls, nakon odbijanja od prepreke, registrira prijemnik. Ultrazvučni senzor za napajanje koristi 5 V iz Arduina.

Jezik u kojem se Arduino programira je baziran na C/C++ te ima iste osnovne funkcije i sličnu sintaksu. Programi za Arduino se nazivaju *sketchevima* i sastoje se od nekoliko dijelova. Prvi dio je *setup* u kojem inicijaliziramo potrebne postavke:

```
1 void setup(){
2   //postavke programa
3   .
4   .
5 }
```

Drugi dio je *loop* u kojem Arduinu dajemo komande koje će izvršavati dok se Arduino ne ugasi.

```
1 void loop(){
2   //naredbe koje se izvršavaju u beskonačnoj petlji
3   .
```



Slika 1.3: Shema spajanja Arduina sa senzorom

```
    .
}
```

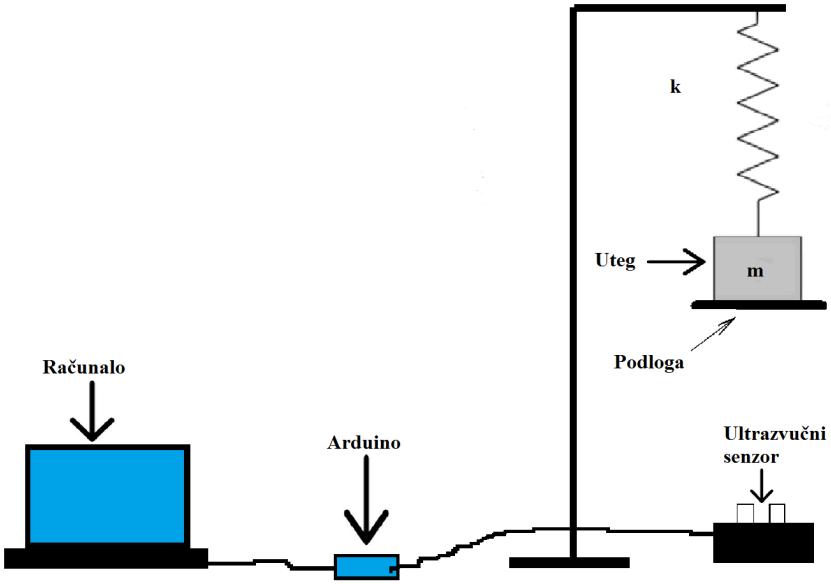
Prema potrebi prije prvog dijela (*setup*) može se dodati inicijalizacijski dio u kojem se najčešće definiraju neke globalne varijable:

```
const int var1 = 100;
float var2 = 7.4;
```

Programi za Arduino mogu sadržavati i funkcije, ali mi ih u našem radu nismo koristili.

2 Opis eksperimentalnog postava

2.1 Metoda mjerenja



Slika 2.1: Mjerni postav za mjerenje oscilacija na opruzi

Za mjerenja nam trebaju računalo, Arduino, ultrazvučni senzor, opruga, utezi, podloga i stalak s kojeg će uteg visiti. Podloga, koju stavljamo ispod utega srazmjerno malih dimenzija, nam služi kako bi povećali površinu s koje se ultrazvučni signali mogu odbijati da dobijemo što preciznije rezultate. Ukupno opterećenje opruge predstavljali su utezi od 10 g i njih smo dodavali i micali sa šipke obješene na oprugu. Prije samog mjerenja računalo spojimo s Arduinom koji je spojen na senzor. Iznad senzora postavimo uteg sa širom podlogom da visi obješen o oprugu (slika 2.1). Nakon toga provjerimo je li senzor dobro spojen tako da pokrenemo program na računalu dok sve miruje. Kad smo se uvjerili da sve radi pomaknemo uteg iz ravnotežnog položaja i pustimo da titra. Uteg smo uvijek vukli prema dolje i puštali ga 5 cm ili 10 cm iznad senzora. Nakon što smo se uvjerili da program prikuplja podatke pritisnemo tipku *Clear*, te ponovo tipku *Start* i počinjemo promatrati eksperiment.

2.2 Program za Arduino

Pogledajmo pobliže kôd za Arduino. (Potpuni program se nalazi u dodatku B)

```
1 const int trigPin = 8;
2 const int echoPin = 7;
```

U prve dvije linije definiramo dvije cjelobrojne konstante koje ćemo koristiti kao pin koji šalje impuls (trigPin) i pin koji prima impuls (echoPin).

```
1  void setup() {
2    Serial.begin(115200);

3  .
4  .
5  .

6    pinMode(trigPin, OUTPUT);
7    pinMode(echoPin, INPUT);
8  }
```

U *setup* djelu programa, koji služi za postavljanje osnovnih postavki programa, prvo pokrenemo serijsku komunikaciju, a zatim pin trigPin (8) postavljamo u izlazni (OUTPUT) mod i pin echoPin (7) u ulazni (INPUT) mod.

```
1  void loop() {
2    long pulse_duration;
3    //postavljamo trigPin na HIGH 5 mikrosekundi što će poslati puls od 5
         mikrosekundi
4    digitalWrite(trigPin, LOW);
5    delayMicroseconds(2);
6    digitalWrite(trigPin, HIGH);
7    delayMicroseconds(5);
8    digitalWrite(trigPin, LOW);
9    //postavljamo echoPin da očekuje signal u sljedeće 2 sekunde
10   pulse_duration = pulseIn(echoPin, HIGH, 2000000);
11   //ako Arduino dobije znak preko serijskog porta pošalje informacije o
         trajanju programa i pulsa na taj port
12   if (Serial.available() > 0) {
13     inByte = Serial.read();
14
15     Serial.print(millis());
16     Serial.print(",");
17     Serial.println(pulse_duration);
18   }
19   delay(10);
20 }
```

U *loop* djelu programa, koji se konstantno ponavlja dok je program na Arduinu pokrenut, imamo tri djela. U liniji 2 je definiranje varijable u koju ćemo spremati vrijeme putovanja signala. Linijama 4 do 10 se odašilje signal, mjeri vrijeme potrebno da sig-

nal dođe od odašiljača do objekta i natrag te sprema to vrijeme. Pomoću linija 12 do 18 šaljemo podatke na računalo preko serijske veze. Od podataka se šalje vrijeme trajanja programa na Arduinu i vrijeme putovanja pulsa. Vrijeme trajanja programa na Arduinu ćemo koristiti za određivanje vremena na vremenskoj osi. Vrijeme putovanja pulsa ćemo koristiti za određivanje elongacije opruge na osi za udaljenost. Ovaj program se pokrene kad se Arduino priključi na napajanje (ili kada Arduinu pošaljemo novi program) te vrijeme trajanja programa koje dobivamo je vrijeme od početka rada programa.

2.3 Program za računalo

Na računalu možemo koristiti bilo koji programski jezik koji može komunicirati sa serijskim portom. Zbog jednostavnosti koristili smo Python. Pogledajmo pobliže Python kod. (Potpuni program se nalazi u dodatku A)

```
1  import os
2  import pprint
3  import random
4  import sys
5  import wx
6
7  import matplotlib
8  matplotlib.use('WXAgg')
9  from matplotlib.figure import Figure
10 from matplotlib.backends.backend_wxagg import \
11     FigureCanvasWxAgg as FigCanvas, \
12     NavigationToolbar2WxAgg as NavigationToolbar
13 import numpy as np
14 import pylab
15
16 import time
17 import serial
```

Za izradu grafičkog sučelja (GUI) koristimo biblioteku *WxPython* zbog jednostavnosti crtanja i prikazivanja grafova koje nam nudi. Za crtanje grafa i računanje koristimo biblioteke *numpy*, *matplotlib* i *pylab* kako bi preračunali podatke dobivene od Arduina u nama potrebne podatke te također te podatke prikazali na grafu. Za komunikaciju s Arduinom nam je potrebna biblioteka *serial* kojom ostvarujemo komunikaciju preko serijskog porta.

```
1  class DataGen(object):
2
3      def __init__(self, init=50):
4          try:
```

```
5             self.ser = serial.Serial(2, 115200, timeout=10)
6         except serial.serialutil.SerialException:
7             self.ser = None
8
9
10    def next(self):
11        if not self.ser:
12            return 0, 0
13
14        self.ser.write(b'A')
15        arduino_time, arduino_pulse_duration =
              self.ser.readline().strip().split(',')
16        return float(arduino_pulse_duration)/29/2, float(arduino_time)/1000
17
18    def __del__(self):
19        if self.ser:
20            self.ser.close()
```

Klasa *DataGen* nam služi za komunikaciju i pri njenoj inicijalizaciji (u linijama 3 do 7) Python provjerava postoji li serijski port. U funkciji *next*(linije 10 do 16) Python šalje znak (u našem slučaju slovo *A*) i očekuje natrag dva broja odvojena zarezom. U našem slučaju očekujemo pronaći na portu Arduino i preko funkcije *next* dobiti podatke o trajanju programa na Arduinu i vremenu putovanja pulsa. Za dobiti udaljenost utega od senzora vrijeme putovanja pulsa, koji je u mikrosekundama, prvo preračunamo u sekunde te zatim to vrijeme pomnožimo sa brzinom zvuka. Put koji tako dobijemo jest put koji je signal prošao iz odašiljača do utega i nazad do prijemnika. Taj put treba podijeliti s dva kako bi dobili udaljenost utega od senzora.
Klasa *BoundControlBox* služi za lakše postavljanje kontrola u grafičkom sučelju.

```
1  class GraphFrame(wx.Frame):
2      title = 'Distance in time'
3
4      def __init__(self):

5          .
6          .
7          .

8          self.redraw_timer = wx.Timer(self)
9          self.Bind(wx.EVT_TIMER, self.on_redraw_timer, self.redraw_timer)
10         self.redraw_timer.Start(500)
11
12         self.data_timer = wx.Timer(self)
13         self.Bind(wx.EVT_TIMER, self.on_data_timer, self.data_timer)
```

```
14          self.data_timer.Start(10)
```

U klasi *GraphFrame* nalazi se većina koda i funkcionalnosti ovog programa. Pri inicijalizaciji klase postavljamo dva brojača. Prvi brojač je *redraw_timer* i služi da bi program znao kad da ponovo crta graf. Postavljen je na 500 ms. Drugi brojač je *data_timer* te služi da bi program znao kad da zatraži nove podatke od Arduina. Postavljen je na 10 ms. Razlog zašto nam trebaju dva brojača je što je crtanje grafa zahtjevno za računalo i želimo što rjeđe to raditi dok s druge strane želimo što češće dobivati nove podatke za graf da bi nam graf bio što precizniji.

Unutar funkcije *create_main_panel* slažemo izgled našeg grafičkog sučelja.

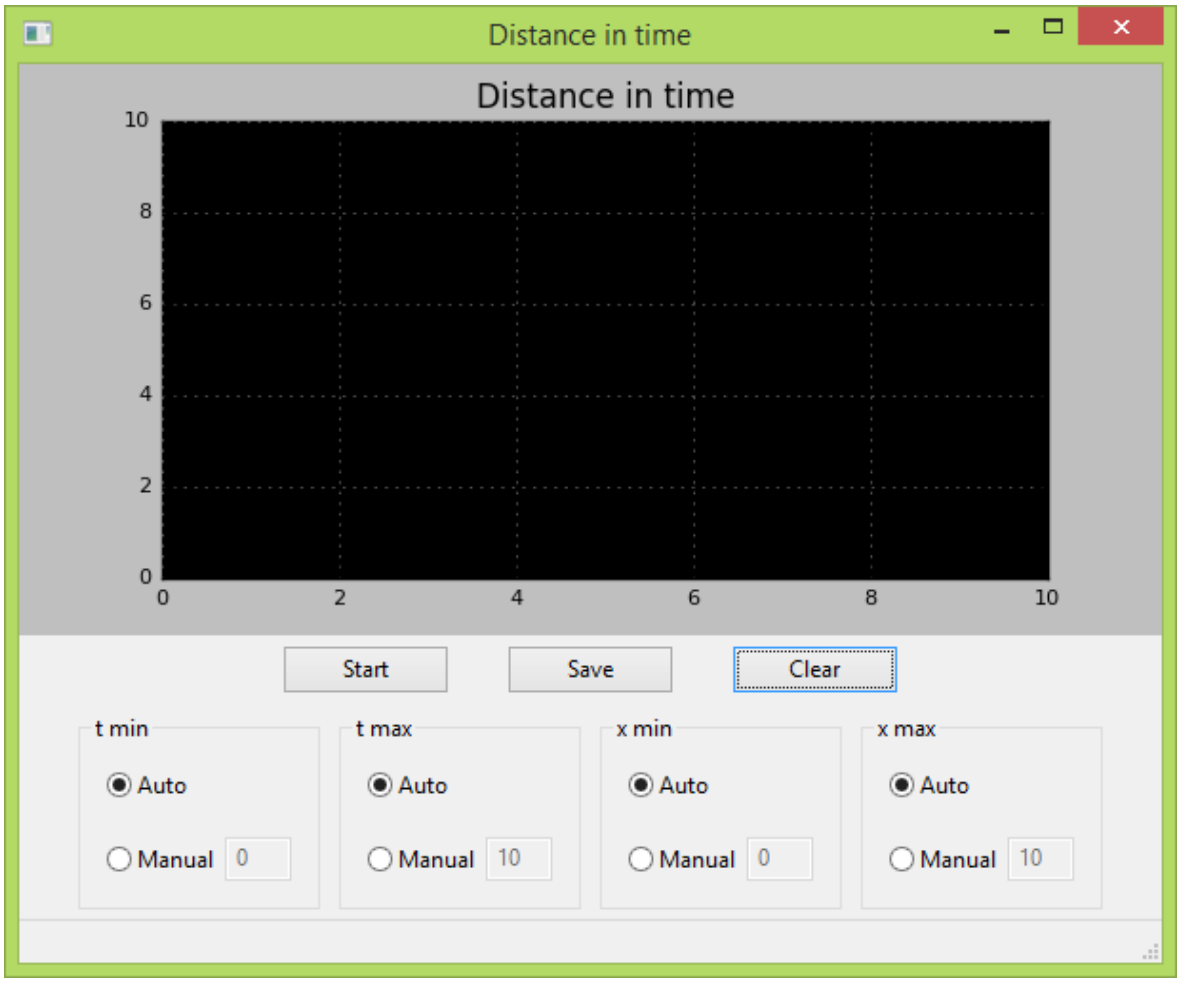
Funkcija *init_plot* nam služi za postavljanje izgleda grafa, dok nam funkcija *draw_plot* služi za postavljanje minimalne i maksimalne vrijednosti na osima i postavljanje točaka na grafu.

```
1       def on_pause_button(self, event):
2           self.paused = not self.paused
3           self.started = True
4
5           event.Skip()
6
7
8       def on_update_pause_button(self, event):
9           if not self.started:
10              label = "Start"
11          else:
12              label = "Resume" if self.paused else "Pause"
13
14          self.pause_button.SetLabel(label)
15
16      def on_start_button(self, event):
17          if self.pause_button.GetLabel() == "Start":
18              self.temp = self.datagen.next()
19              self.start_time = self.temp[1]
20              self.time.append(self.temp[1]-self.start_time) #time axis (x)
21              self.elong.append(self.temp[0]) #elongation axis (y)
22
23          event.Skip()
24
25
26
27      def on_clear_button(self, event):
28          self.time = list()
29          self.elong = list()
30          self.paused = True
```

```
31          self.started = False
32
33      def on_save_button(self, event):
34          file_choices = "TEXT (*.txt)|*.txt"
35
36          dlg = wx.FileDialog(
37              self,
38              message="Save data as...",
39              defaultDir=os.getcwd(),
40              defaultFile="data.txt",
41              wildcard=file_choices,
42              style=wx.SAVE | wx.FD_OVERWRITE_PROMPT)
43
44          if dlg.ShowModal() == wx.ID_OK:
45              path = dlg.GetPath()
46              fp = file(path, 'w')
47              fp.write("time[s]\telongation[cm]\n")
48              for i in range(len(self.time)):
49                  fp.write( str(self.time[i])+"\t"+str(self.elong[i])+"\n")
50              fp.close()
51
52          dlg.Destroy()
53          event.Skip()
54
55      def on_redraw_timer(self, event):
56          self.draw_plot()
57
58      def on_data_timer(self, event):
59          if not self.paused:
60              self.temp = self.datagen.next()
61              self.time.append(self.temp[1]-self.start_time) #time axis (x)
62              self.elong.append(self.temp[0]) #elongation axis (y)
63
64
65      def on_exit(self, event):
66          sys.exit(0)
67          self.Destroy()
68          self.datagen.__del__()
```

Ostale funkcije unutar klase *GraphFrame* služe za davanje funkcionalnosti gumbima za pokretanje i pauziranje grafa (*Start*/*Pause*, brisanje podataka na grafu (*Clear*), spremanje prikupljenih podataka (*Save*) i zatvaranje prozora (*X*).

Kad pokrenemo program na računalu, vidimo da je prozor podijeljen u dva dijela (slika 2.2). Prvi, gornji, dio služi za prikazivanje grafa, te x os predstavlja vrijeme



Slika 2.2: Izgled pokrenutog programa na računalu.

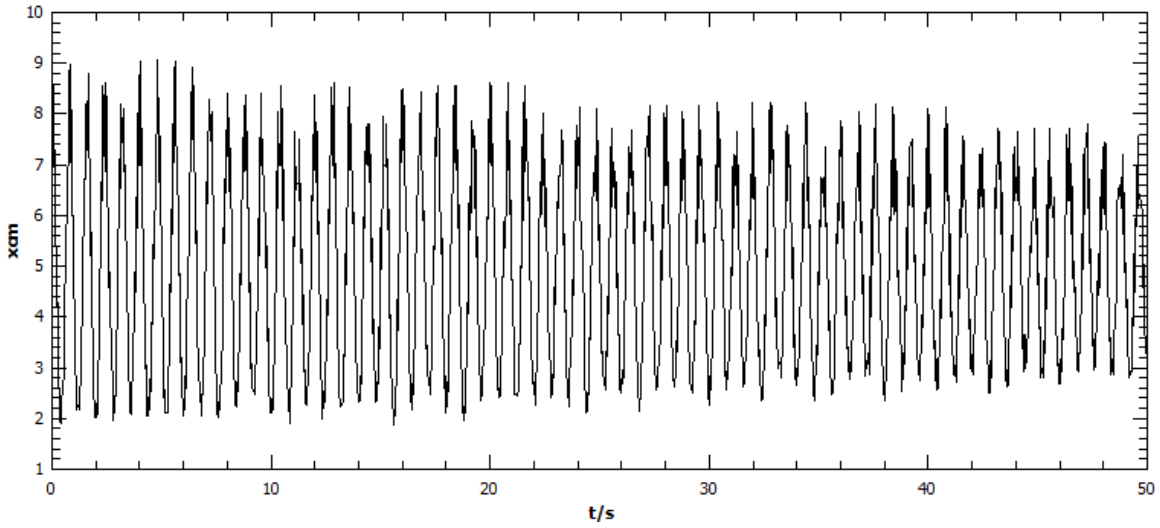
od početka mjerenja, dok y os predstavlja udaljenost podloge od senzora. U drugom, donjem, dijelu se nalaze komande. Na vrhu drugog dijela se nalaze tri tipke. Tipka *Start* služi za pokretanje i zaustavljanje crtanja grafa. Tekst na toj tipki može prikazivati *Start*, *Pause* ili *Resume* i mijenja se ovisno o stanju programa. Tipka *Save* služi za spremanje prikupljenih podataka u tekstualni dokument u dva stupca. Prvi stupac prikazuje vrijeme od kad je pritisnuta tipka *Start*, dok drugi stupac prikazuje udaljenost podloge od senzora. Tipka *Clear* služi za brisanje grafa i svih prikupljenih podataka koji se nisu spremili. Ispod tipki nalazi se četiri kvadrata. Opcije u tim kvadratima služe za postavljanje vrijednosti koje graf prikazuje. Prva dva kvadrata ($t\ min$ i $t\ max$) služe za postaviti minimalnu i maksimalnu vrijednost na vremenskoj osi. Ako se ostavi opcija *Auto* maksimalna vrijednost će biti posljednja prikupljena vrijednost, dok će minimalna vrijednost biti 10 sekundi manja od maksimalne. Druga dva kvadrata ($x\ min$ i $x\ max$) također služe za postaviti minimalnu i maksimalnu vrijednost, no ovaj put na osi koja prikazuje udaljenost. Ako se ostavi opcija *Auto* u ovom slučaju maksimalna vrijednost će biti najveća zapisana udaljenost, dok će minimalna vrijednost biti najmanja zapisana udaljenost. Sva četiri kvadrata imaju opciju *Manual* koja služi kako bi mogli promatrati određene dijelove grafa, te ta opcija prihvaća cijele

i decimalne brojeve.

3 Rezultati mjerenja

Eksperimenti su obavljeni na oprugama s 3 različite podloge (drvena, plastična i obje istovremeno) te na njihalu. Opruga je puštana s 5 cm od senzora dok je njihalo bilo duljine 65 cm. Mase korištenih utega su bile između 10 g i 80 g kako bi mogli dobiti dobra mjerenja i ostati u području linearnosti opruge. Promjer podloga iznosi 5.9 cm.

3.1 Opruga



Slika 3.1: Primjer izgleda rezultata za oprugu s drvenom podlogom i utegom od 50 g.

Na slici 3.1 prikazana je ovisnost položaja o vremenu za slučaj kada imamo drvenu podlogu i uteg od 50 g. Primjenjujući Fourierovu analizu na te podatke možemo dobiti frekvenciju titranja našeg oscilatora.

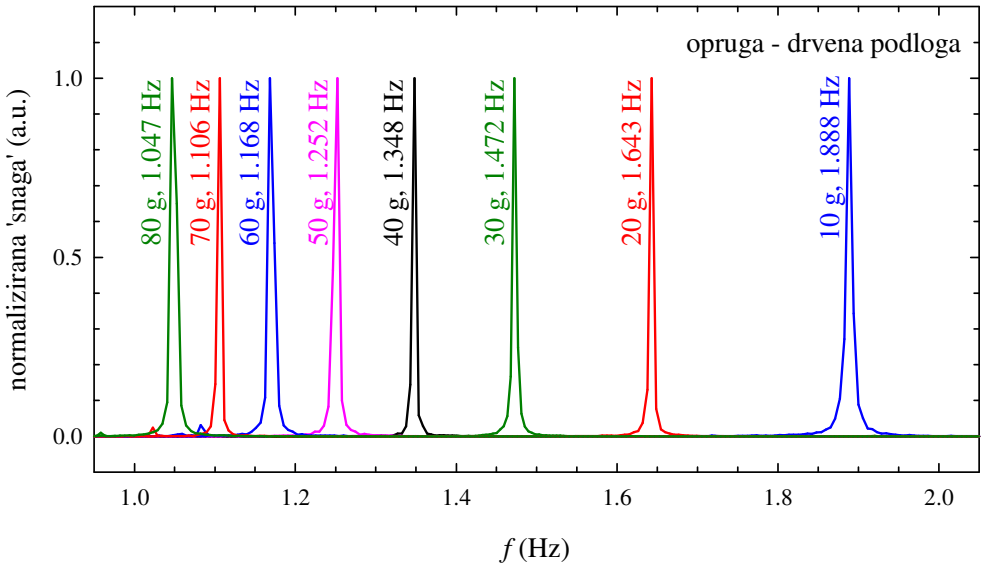
Na slici 3.2 prikazani su frekvencijski spektri za oscilator na koji smo stavili samo drveno podlogu. Možemo vidjeti da nam se frekvencija povećava smanjenjem mase i imamo samo jedan izražen vrh za svaku masu, što nam ukazuje na nikakve do male smetnje pri mjerenju.

Na slici 3.3 prikazani su frekvencijski spektri za oscilator na koji smo stavili samo plastičnu podlogu. Možemo ponovo vidjeti da frekvencija raste sa smanjenjem mase te izostanak smetnji.

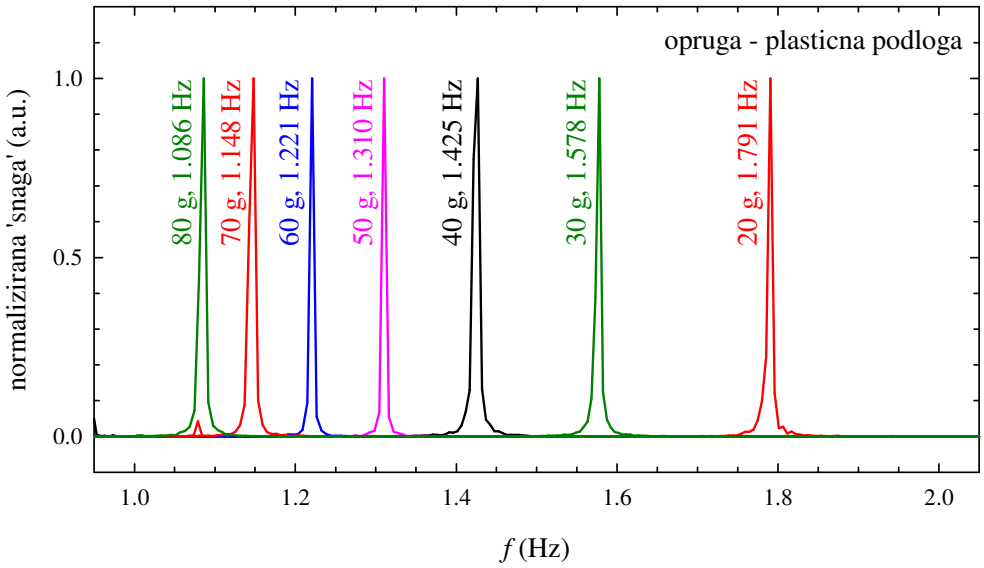
Na slici 3.4 prikazani su frekvencijski spektri za oscilator na koji smo stavili obje podloge. Ponovo se vidi pokazuje analogni odnos rezonantne frekvencije i mase te izostanak smetnji.

Kako bi vidjeli da li se dobiveni rezultati slažu s teorijskim predviđanjima počnimo od jednadžbe

$$T = 2\pi\sqrt{\frac{m}{k}}$$



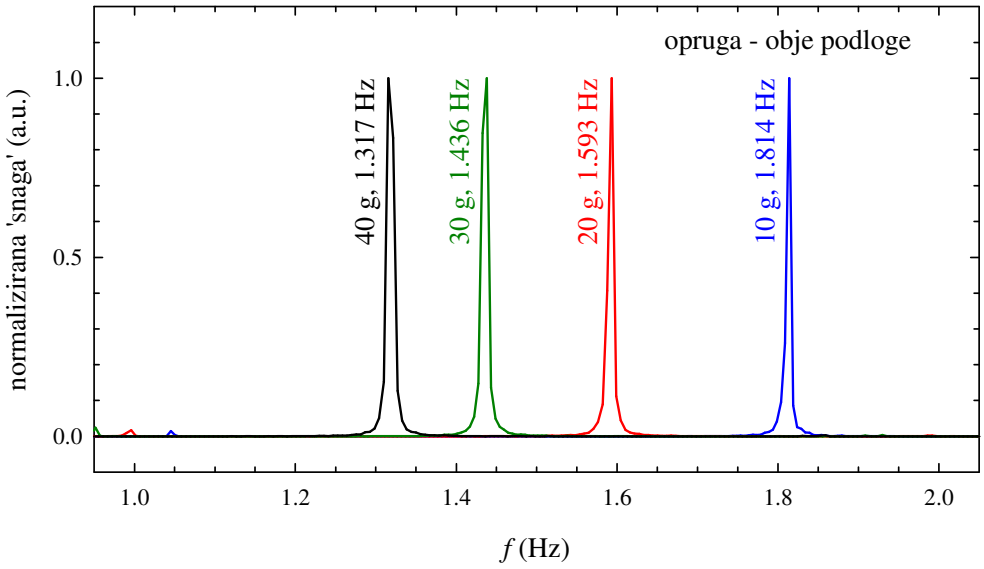
Slika 3.2: Frekvencijski spektri za oscilator s drvenom podlogom uz frekvenciju sistema i masu utega istaknutima na najizraženijim vrhovima.



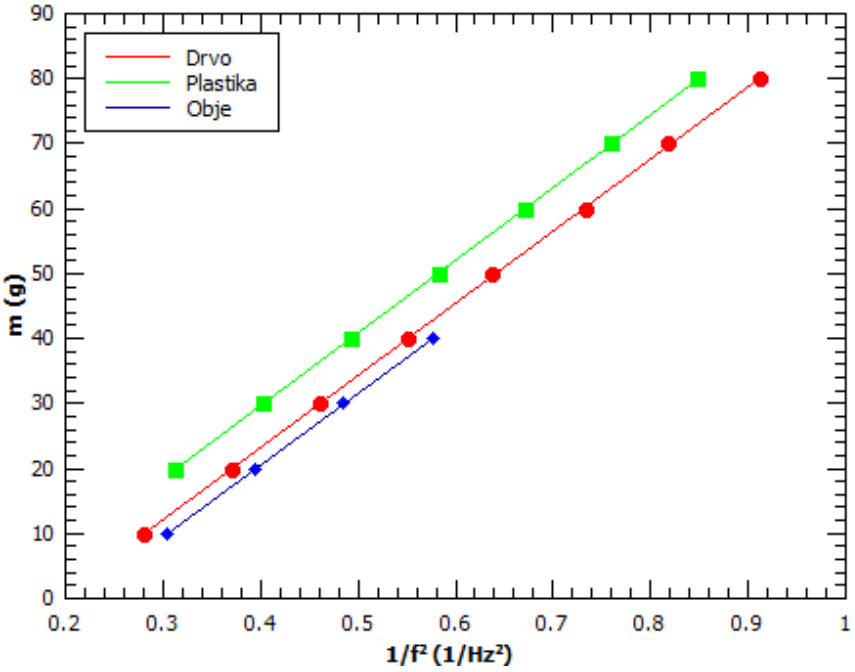
Slika 3.3: Frekvencijski spektri za oscilator s plastičnom podlogom uz frekvenciju sistema i masu utega istaknutima na najizraženijim vrhovima.

u kojoj je T period titranja, m ukupna masa na opruzi i k konstanta opruge. Kvadriranjem dobivamo:

$$T^2 = 4\pi^2 \frac{m}{k}.$$



Slika 3.4: Frekvencijski spektri za oscilator s obje podloge uz frekvenciju sistema i masu utega istaknutima na najizraženijim vrhovima.



Slika 3.5: Ovisnost mase utega o inverzu kvadrata frekvencije

Nakon toga uvrstimo zamjenu $m \to m+m_0$ gdje je novi m masa utega, a m_0 nepoznata masa podloge i šipke na kojoj se nalaze utezi. Slijedi:

$$T^2 = 4\pi^2 \frac{m + m_0}{k}$$

Uvrstivši $T^2 = 1/f^2$ dobijemo jednadžbu:

$$\frac{1}{f^2} = \frac{4\pi^2}{k}(m + m_0).$$

Iz prijašnje jednadžbe nam slijedi:

$$m + m_0 = \frac{k}{4\pi^2}\frac{1}{f^2}$$

te na kraju imamo

$$m = \frac{k}{4\pi^2}\frac{1}{f^2} - m_0. \tag{3.1}$$

Stoga očekujemo linearnu ovisnost između mase m i $1/f^2$ te da će nam odsječak na osi y odgovarati nepoznatoj masi m_0.

Na slici 3.5 zaista vidimo linearnu ovisnost mase o inverzu kvadrata frekvencije. Možemo primijetiti kako nam se odsječak na y-osi mijenja ovisno o podlozi što nam govori o odnosima nepoznatih masa i masa podloga. Nagibi pravaca na slici su jako bliski iz čega proizlazi da će nam i konstante opruga biti slične. Pošto je korištena uvijek ista opruga takav rezultat je poželjan.

Rezultati za pojedine podloge dani su u tablici 3.1. Vrijednosti za konstantu opruge (zajedno s greškom) su:

$$k_D = 4\pi^2 \times (0.1111 \pm 0.0004)\ \text{N/m}$$

$$k_P = 4\pi^2 \times (0.1119 \pm 0.0003)\ \text{N/m}$$

$$k_B = 4\pi^2 \times (0.1100 \pm 0.0003)\ \text{N/m}$$

To nam na kraju daje:

$$k = 4\pi^2 \times (0.1110 \pm 0.0003)\ \text{N/m}$$

$$k = (4.38 \pm 0.01)\ \text{N/m}$$

Iz jednadžbe 3.1 vidimo da nam je $-m_0$ odsječak na osi y te ga možemo jednostavno iščitati. Za izračunavanje masa podloga imamo 3 jednadžbe za svaki od promatranih slučaja:

$$m_{0D} = m_D + m_{\check{S}}$$

$$m_{0P} = m_P + m_{\check{S}}$$

$$m_{0B} = m_D + m_P + m_{\check{S}}$$

Ovdje je m_{0D} odsječak na osi y u slučaju s drvenom podlogom, m_{0P} odsječak na osi y u slučaju s plastičnom podlogom, m_{0B} odsječak na osi y u slučaju s obje podloge, m_D masa drvene podloge, m_P masa plastične podloge te $m_{\check{S}}$ masa šipke za utege. Nakon rješavanja tog sustava dobijemo da su mase podloga:

$$m_D = (8.6 \pm 0.2)\ \text{g}$$

$$m_P = (2.3 \pm 0.2)\ \text{g}$$

te masa šipke

$$m_{\check{S}} = (12.6 \pm 0.2)\ \text{g}.$$

Usporedba masa podloga dobivenih iz analize podataka i direktnim vaganjem nalazi se u tablici 3.2.

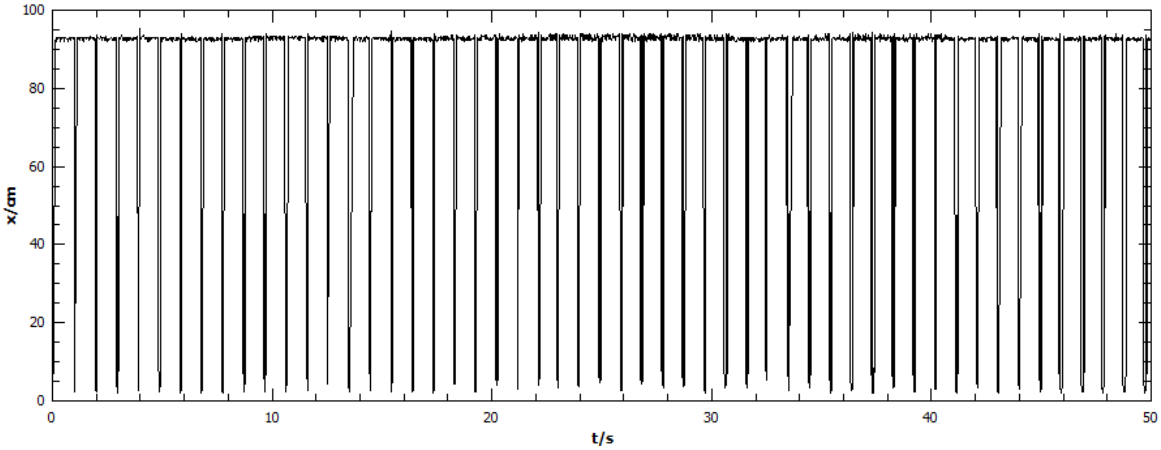
podloga	drvo	plastika	drvo+plastika
$\frac{1}{4}\text{k}\pi^{-2}/\text{Nm}^{-1}$	0.1111	0.1119	0.1100
k/Nm^{-1}	4.386	4.418	4.343
m_0/g	21.2	15.0	23.4

Tablica 3.1: Konstanta opruge i masa podloga m_0

	mjerenje	vaga
$m(\text{drvo})/\text{g}$	8.6	9.1
$m(\text{plastika})/\text{g}$	2.3	2.7

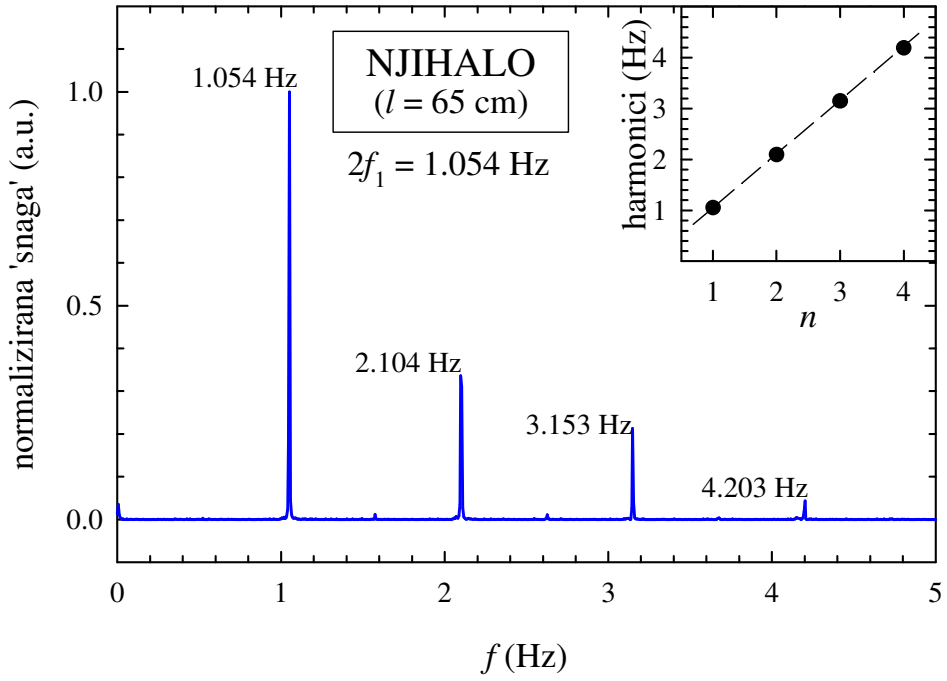
Tablica 3.2: Mase podloga dobivene iz mjerenja i direktnim vaganjem podloga.

3.2 Njihalo



Slika 3.6: Primjer izgleda rezultata za njihalo

Postav eksperimenta je sličan onome na slici 2.1, samo umjesto opruge imamo nit, koja je u našem slučaju duljine 65 cm, na koju objesimo uteg. Kao i prije potrebna nam je podloga ispod utega, no ovdje nam je trebala i gornja podloga na vrhu stalka kako bi nam se poslani signal vratio. Pri odmicanju njihala iz ravnotežnog položaja pripazili smo da kut ne bude prevelik te smo nastojali početni kut otklona postaviti što bliže 15°.



Slika 3.7: Frekvencijski spektar za njihalo. Umetak: sve opažene frekvencije spektra odgovaraju harmonicima fundamentalnog moda.

Ovisnost oscilacija s Arduina u vremenu je prikazana na slici 3.6. Pošto je uteg samo na kratko tijekom svog njihanja prolazio iznad senzora podaci koje smo dobili tvore pravokutni signal. Za analizu smo i ovdje koristili Fourierovu transformaciju i dobili rezultate prikazane na slici 3.7. Vidimo da se kod njihala za razliku od opruge pojavljuju viši harmonici fundamentalnog moda ($n = 1$), no unatoč tome prvi harmonik je najizraženiji i on odgovara dvostrukoj frekvenciji titranja njihala.

Period za njihala dan je sa

$$T = 2\pi\sqrt{\frac{l}{g}}.$$

Ako u to ponovo uvrstimo izraz $T = 1/f$ dobivamo:

$$f = \frac{1}{2\pi}\sqrt{\frac{g}{l}}$$

što nam za nit duljine 65 cm daje

$$2f = \frac{1}{T/2} = (1.24 \pm 0.01)\ \text{Hz}.$$

Uspoređujući taj rezultat s onim dobivenim iz analize na slici 3.7, $2f_1 = 1.054$ Hz, vidimo da se ne podudaraju. Pogledajući sliku 3.6 možemo vidjeti i zašto. Iako je naša nit bila duljine 65 cm, sama šipka na kojoj su se nalazili utezi je imala svoju duljinu koja je utjecala na duljinu cijelog njihala. Osim toga opažena frekvencija je manja od teorijski predviđene i zbog utjecaja trenja niti i/ili utega sa zrakom.

4 Zaključak

U ovom radu smo pokazali neke od mogućnosti i primjena Arduina u eksperimentalnim postavima. Za potrebe ovog rada koncentrirali smo se na ultrazvučni senzor kako bi lakše primijetili poteškoće u takvoj primjeni Arduina.

Za početak smo napisali program za sam Arduino koji nam dobivene podatke o vremenu trajanja programa i putovanja pulsa šalje na serijski port. Nakon toga smo izradili program u Pythonu koji nam te podatke obrađuje i prikazuje u grafičkom sučelju. Kad smo završili s pripremama došlo je vrijeme da te programe i Arduino testiramo u pravom eksperimentu. Eksperiment koji smo odabrali jest titranje tijela, prvo na opruzi, a zatim i na niti (njihalo).

Grafove koje nam naš program crta iz prikupljenih podataka su zadovoljavajući na prvi pogled. Daljnjom analizom prikupljenih podataka ispostavlja se da oni slijede ponašanje predviđeno teorijom kao što možemo vidjeti na slici 3.5. Nažalost iz tablice 3.2 možemo primijetiti da su vrijednosti masa dobivenih iz analize podataka manji od izvagane vrijednosti. Te vrijednosti su za drvo 8.6 g iz analize naspram 9.1 g direktnim vaganjem i za plastiku 2.3 g iz analize naspram 2.7 g direktnim vaganjem. Takvi rezultati nas upućuju na to da je još nešto utjecalo na naše podatke, no ne na izgled ovisnosti. Takav utjecaj , kao i kod njihala, pripisujemo trenju koji se može opaziti i na slici 3.1 u obliku gušenja. Također dio greške može proizaći iz činjenice da smo za brzinu zvuka uzeli tabličnu vrijednost.

Vidimo da se ovakav eksperiment može bez ikakvih poteškoća provesti i u školama tijekom nastave kako bi učenici mogli lakše razumjeti titranja. Možemo na kraju zaključiti da je Arduino korisno i prilagodljivo računalo zahvaljujući svojim mnogobrojnim senzorima koje se može jednostavno upotrijebiti u eksperimentima.

Dodaci

Dodatak A Python kod

```
import os
import pprint
import random
import sys
import wx

import matplotlib
matplotlib.use('WXAgg')
from matplotlib.figure import Figure
from matplotlib.backends.backend_wxagg import \
    FigureCanvasWxAgg as FigCanvas, \
    NavigationToolbar2WxAgg as NavigationToolbar
import numpy as np
import pylab

import time
import serial


class DataGen(object):

    def __init__(self, init=50):
        try:
            self.ser = serial.Serial(2, 115200, timeout=10)
        except serial.serialutil.SerialException:
            self.ser = None


    def next(self):
        if not self.ser:
            return 0, 0

        self.ser.write(b'A')
        arduino_time, arduino_pulse_duration =
            self.ser.readline().strip().split(',')
        return float(arduino_pulse_duration)/29/2, float(arduino_time)/1000

    def __del__(self):
        if self.ser:
            self.ser.close()

class BoundControlBox(wx.Panel):

    def __init__(self, parent, ID, label, initval):
        wx.Panel.__init__(self, parent, ID)

        self.value = initval

```

```
48          box = wx.StaticBox(self, -1, label)
49          sizer = wx.StaticBoxSizer(box, wx.VERTICAL)
50
51          self.radio_auto = wx.RadioButton(self, -1,
52              label="Auto", style=wx.RB_GROUP)
53          self.radio_manual = wx.RadioButton(self, -1,
54              label="Manual")
55          self.manual_text = wx.TextCtrl(self, -1,
56              size=(35,-1),
57              value=str(initval),
58              style=wx.TE_PROCESS_ENTER)
59
60          self.Bind(wx.EVT_UPDATE_UI, self.on_update_manual_text, self.manual_text)
61          self.Bind(wx.EVT_TEXT_ENTER, self.on_text_enter, self.manual_text)
62
63          manual_box = wx.BoxSizer(wx.HORIZONTAL)
64          manual_box.Add(self.radio_manual, flag=wx.ALIGN_CENTER_VERTICAL)
65          manual_box.Add(self.manual_text, flag=wx.ALIGN_CENTER_VERTICAL)
66
67          sizer.Add(self.radio_auto, 0, wx.ALL, 10)
68          sizer.Add(manual_box, 0, wx.ALL, 10)
69
70          self.SetSizer(sizer)
71          sizer.Fit(self)
72
73      def on_update_manual_text(self, event):
74          self.manual_text.Enable(self.radio_manual.GetValue())
75
76      def on_text_enter(self, event):
77          self.value = self.manual_text.GetValue()
78
79      def is_auto(self):
80          return self.radio_auto.GetValue()
81
82      def manual_value(self):
83          return self.value
84
85  class GraphFrame(wx.Frame):
86      title = 'Distance in time'
87
88      def __init__(self):
89          wx.Frame.__init__(self, None, -1, self.title)
90
91          self.datagen = DataGen()
92          self.temp = self.datagen.next()
93          self.elong = []
94          self.time = []
95          self.start_time = self.temp[1]
96
97          self.paused = True
98          self.started = False
99
100         self.create_status_bar()
101         self.create_main_panel()
```

```
102
103         self.redraw_timer = wx.Timer(self)
104         self.Bind(wx.EVT_TIMER, self.on_redraw_timer, self.redraw_timer)
105         self.redraw_timer.Start(500)
106
107         self.data_timer = wx.Timer(self)
108         self.Bind(wx.EVT_TIMER, self.on_data_timer, self.data_timer)
109         self.data_timer.Start(10)
110
111     def create_main_panel(self):
112         self.panel = wx.Panel(self)
113
114         self.init_plot()
115         self.canvas = FigCanvas(self.panel, -1, self.fig)
116
117         self.xmin_control = BoundControlBox(self.panel, -1, "t min", 0)
118         self.xmax_control = BoundControlBox(self.panel, -1, "t max", 10)
119
120         self.ymin_control = BoundControlBox(self.panel, -1, "x min", 0)
121         self.ymax_control = BoundControlBox(self.panel, -1, "x max", 10)
122
123         self.pause_button = wx.Button(self.panel, -1, "Start")
124         self.Bind(wx.EVT_BUTTON, self.on_start_button, self.pause_button)
125         self.Bind(wx.EVT_BUTTON, self.on_pause_button, self.pause_button)
126         self.Bind(wx.EVT_UPDATE_UI, self.on_update_pause_button, self.pause_button)
127
128         self.save_button = wx.Button(self.panel, -1, "Save")
129         self.Bind(wx.EVT_BUTTON, self.on_pause_button, self.save_button)
130         self.Bind(wx.EVT_BUTTON, self.on_save_button, self.save_button)
131
132         self.clear_button = wx.Button(self.panel, -1, "Clear")
133         self.Bind(wx.EVT_BUTTON, self.on_clear_button, self.clear_button)
134         self.Bind(wx.EVT_UPDATE_UI, self.on_update_pause_button, self.clear_button)
135
136         self.hbox1 = wx.BoxSizer(wx.HORIZONTAL)
137         self.hbox1.Add(self.pause_button, border=5, flag=wx.ALL |
                wx.ALIGN_CENTER_VERTICAL)
138         self.hbox1.AddSpacer(20)
139         self.hbox1.Add(self.save_button, border=5, flag=wx.ALL |
                wx.ALIGN_CENTER_VERTICAL)
140         self.hbox1.AddSpacer(20)
141         self.hbox1.Add(self.clear_button, border=5, flag=wx.ALL |
                wx.ALIGN_CENTER_VERTICAL)
142
143         self.hbox2 = wx.BoxSizer(wx.HORIZONTAL)
144         self.hbox2.Add(self.xmin_control, border=5, flag=wx.ALL)
145         self.hbox2.Add(self.xmax_control, border=5, flag=wx.ALL)
146         self.hbox2.Add(self.ymin_control, border=5, flag=wx.ALL)
147         self.hbox2.Add(self.ymax_control, border=5, flag=wx.ALL)
148
149         self.vbox = wx.BoxSizer(wx.VERTICAL)
150         self.vbox.Add(self.canvas, 1, flag=wx.LEFT | wx.TOP | wx.GROW)
151         self.vbox.Add(self.hbox1, 0, flag=wx.ALIGN_CENTER | wx.TOP)
152         self.vbox.Add(self.hbox2, 0, flag=wx.ALIGN_CENTER | wx.TOP)
```

```
153
154         self.panel.SetSizer(self.vbox)
155         self.vbox.Fit(self)
156
157     def create_status_bar(self):
158         self.statusbar = self.CreateStatusBar()
159
160     def init_plot(self):
161         self.dpi = 100
162         self.fig = Figure((6.0, 3.0), dpi=self.dpi)
163
164         self.axes = self.fig.add_subplot(111)
165         self.axes.set_axis_bgcolor('black')
166         self.axes.set_title('Distance in time', size=12)
167
168         pylab.setp(self.axes.get_xticklabels(), fontsize=8)
169         pylab.setp(self.axes.get_yticklabels(), fontsize=8)
170
171         self.plot_data = self.axes.plot(
172             self.elong,
173             linewidth=1,
174             color=(1, 1, 0),
175             )[0]
176
177     def draw_plot(self):
178         if self.xmax_control.is_auto() and len(self.time) >= 1:
179             xmax = self.time[-1] if self.time[-1] > 10 else 10
180         else:
181             try:
182                 xmax = float(self.xmax_control.manual_value())
183             except ValueError:
184                 print "invalid value"
185                 xmax = 10
186
187         if self.xmin_control.is_auto():
188             if xmax >= 10:
189                 xmin = xmax - 10
190             else:
191                 xmin = 0
192         else:
193             try:
194                 xmin = float(self.xmin_control.manual_value())
195             except ValueError:
196                 print "invalid value"
197                 xmin = 0
198
199         if self.ymax_control.is_auto() and len(self.elong) >= 1:
200             ymax = round(max(self.elong), 0) + 1
201         else:
202             try:
203                 ymax = float(self.ymax_control.manual_value())
204             except ValueError:
205                 print "invalid value"
206                 ymax = 10
```

```
207
208         if self.ymin_control.is_auto() and len(self.elong) >=1:
209             ymin = round(min(self.elong), 0) - 1
210         else:
211             try:
212                 ymin = float(self.ymin_control.manual_value())
213             except ValueError:
214                 print "invalid value"
215                 ymin = 0
216
217         self.axes.set_xbound(lower=xmin, upper=xmax)
218         self.axes.set_ybound(lower=ymin, upper=ymax)
219
220         self.axes.grid(True, color='gray')
221
222         self.plot_data.set_xdata(np.array(self.time))
223         self.plot_data.set_ydata(np.array(self.elong))
224
225         self.canvas.draw()
226
227     def on_pause_button(self, event):
228         self.paused = not self.paused
229         self.started = True
230
231         event.Skip()
232
233
234     def on_update_pause_button(self, event):
235         if not self.started:
236             label = "Start"
237         else:
238             label = "Resume" if self.paused else "Pause"
239
240         self.pause_button.SetLabel(label)
241
242     def on_start_button(self, event):
243         if self.pause_button.GetLabel() == "Start":
244             self.temp = self.datagen.next()
245             self.start_time = self.temp[1]
246             self.time.append(self.temp[1]-self.start_time) #time axis (x)
247             self.elong.append(self.temp[0]) #elongation axis (y)
248
249         event.Skip()
250
251
252
253     def on_clear_button(self, event):
254         self.time = list()
255         self.elong = list()
256         self.paused = True
257         self.started = False
258
259     def on_save_button(self, event):
260         file_choices = "TEXT (*.txt)|*.txt"
```

```
261
262         dlg = wx.FileDialog(
263             self,
264             message="Save data as...",
265             defaultDir=os.getcwd(),
266             defaultFile="data.txt",
267             wildcard=file_choices,
268             style=wx.SAVE | wx.FD_OVERWRITE_PROMPT)
269
270         if dlg.ShowModal() == wx.ID_OK:
271             path = dlg.GetPath()
272             fp = file(path, 'w')
273             fp.write("time[s]\telongation[cm]\n")
274             for i in range(len(self.time)):
275                 fp.write( str(self.time[i])+"\t"+str(self.elong[i])+"\n")
276             fp.close()
277
278         dlg.Destroy()
279         event.Skip()
280
281     def on_redraw_timer(self, event):
282         self.draw_plot()
283
284     def on_data_timer(self, event):
285         if not self.paused:
286             self.temp = self.datagen.next()
287             self.time.append(self.temp[1]-self.start_time) #time axis (x)
288             self.elong.append(self.temp[0]) #elongation axis (y)
289
290
291     def on_exit(self, event):
292         sys.exit(0)
293         self.Destroy()
294         self.datagen.__del__()
295
296
297 if __name__ == '__main__':
298     app = wx.App(False)
299     app.frame = GraphFrame()
300     app.frame.Show()
301     app.MainLoop()
```

Dodatak B Arduino kod

```
1  const int trigPin = 8;
2  const int echoPin = 7;
3
4  int inByte;
5
6  void setup() {
7    Serial.begin(115200);
```

```
8
9   while (!Serial) {
10    ;
11  }
12
13  pinMode(trigPin, OUTPUT);
14  pinMode(echoPin, INPUT);
15 }
16
17 void loop() {
18  long pulse_duration;
19  //postavljamo trigPin na HIGH 5 mikrosekundi što će poslati puls od 5 mikrosekundi
20  digitalWrite(trigPin, LOW);
21  delayMicroseconds(2);
22  digitalWrite(trigPin, HIGH);
23  delayMicroseconds(5);
24  digitalWrite(trigPin, LOW);
25  //postavljamo echoPin da očekuje signal u sljedeće 2 sekunde
26  pulse_duration = pulseIn(echoPin, HIGH, 2000000);
27  //ako Arduino dobije znak preko serijskog porta pošalje informacije o trajanju
        programa i pulsa na taj port
28  if (Serial.available() > 0) {
29    inByte = Serial.read();
30
31    Serial.print(millis());
32    Serial.print(",");
33    Serial.println(pulse_duration);
34  }
35  delay(10);
36 }
```

Literatura

[1] Young, H. D.; Freedman, R. A.; Ford, A. L. Sears and Zemansky's University physics with modern physics, 12th ed. London: Pearson Addison-Wesley, 2008.

[2] Kittel, C.; Knight, W. D.; Ruderman, M. A. Udžbenik fizike sveučilišta u Berkleyu, prvi svezak: Mehanika, 2nd ed. Zagreb: Tehnička knjiga, 2003.

[3] Službene stranice za Arduino Leonardo, `https://www.arduino.cc/en/Main/ArduinoBoardLeonardo`

[4] Službene stranice za ultrazvučni senzor, `http://playground.arduino.cc/Main/UltrasonicSensor`

[5] Službene stranice Arduina, `https://www.arduino.cc/en/Tutorial/Blink`, 28.7.2015.

[6] Službene stranice za programski jezik Python, `https://www.python.org/`