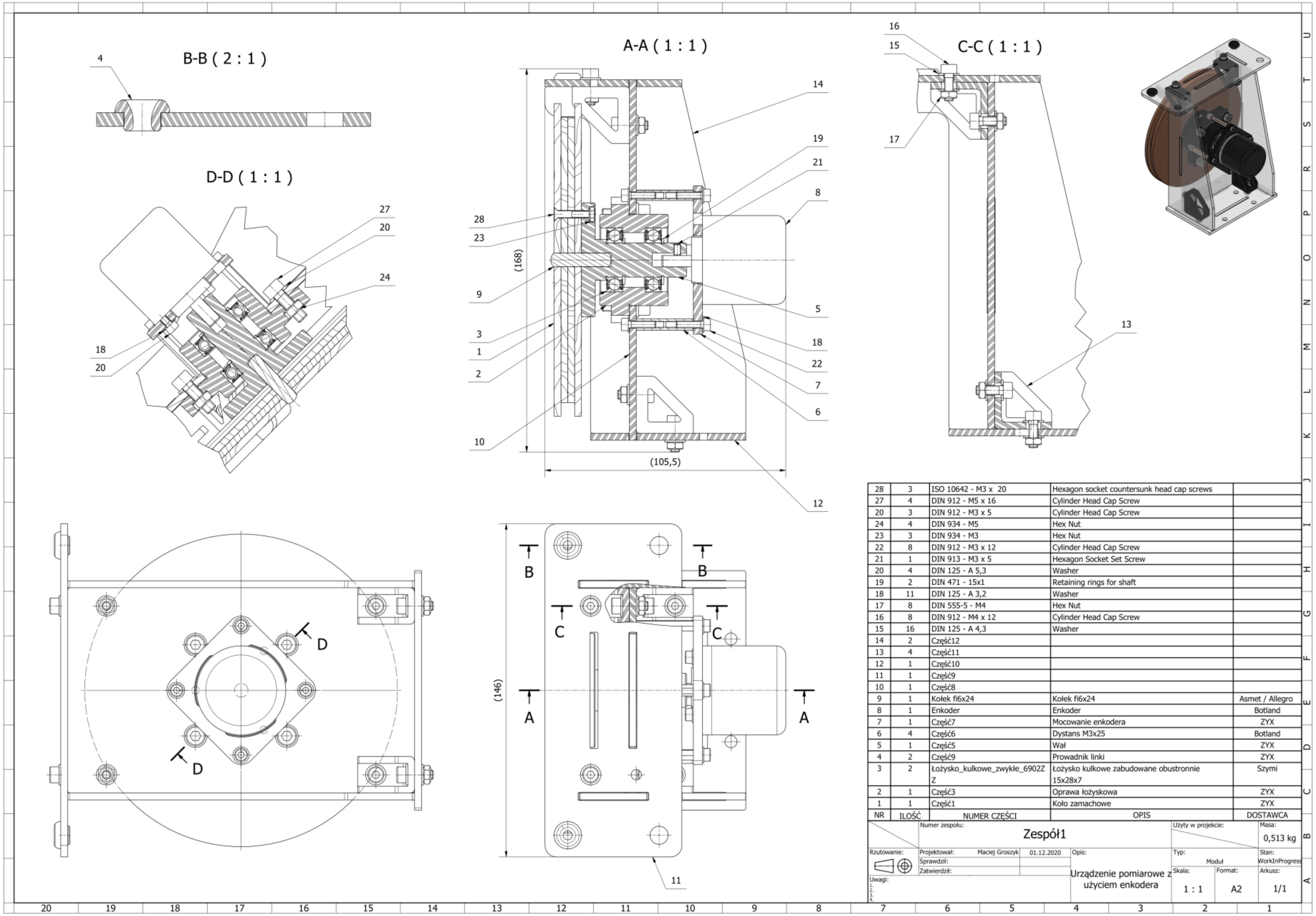
Załącznik nr 1 – „Rysunek techniczny projektu"



28	3	ISO 10642 - M3 x 20	Hexagon socket countersunk head cap screws	
27	4	DIN 912 - M5 x 16	Cylinder Head Cap Screw	
20	3	DIN 912 - M3 x 5	Cylinder Head Cap Screw	
24	4	DIN 934 - M5	Hex Nut	
23	3	DIN 934 - M3	Hex Nut	
22	8	DIN 912 - M3 x 12	Cylinder Head Cap Screw	
21	1	DIN 913 - M3 x 5	Hexagon Socket Set Screw	
20	4	DIN 125 - A 5,3	Washer	
19	2	DIN 471 - 15x1	Retaining rings for shaft	
18	11	DIN 125 - A 3,2	Washer	
17	8	DIN 555-5 - M4	Hex Nut	
16	8	DIN 912 - M4 x 12	Cylinder Head Cap Screw	
15	16	DIN 125 - A 4,3	Washer	
14	2	Część12		
13	4	Część11		
12	1	Część10		
11	1	Część9		
10	1	Część8		
9	1	Kołek fi6x24	Kołek fi6x24	Asmet / Allegro
8	1	Enkoder	Enkoder	Botland
7	1	Część7	Mocowanie enkodera	ZYX
6	4	Część6	Dystans M3x25	Botland
5	1	Część5	Wał	ZYX
4	2	Część9	Prowadnik linki	ZYX
3	2	Łożysko_kulkowe_zwykłe_6902Z Z	Łożysko kulkowe zabudowane obustronnie 15x28x7	Szymi
2	1	Część3	Oprawa łożyskowa	ZYX
1	1	Część1	Koło zamachowe	ZYX
NR	ILOŚĆ	NUMER CZĘŚCI	OPIS	DOSTAWCA

	Numer zespołu: Zespół1			Użyty w projekcie:	Masa: 0,513 kg
Rzutowanie:	Projektował: Maciej Groszyk	01.12.2020	Opis: Urządzenie pomiarowe z użyciem enkodera	Typ: Moduł	Stan: WorkInProgress
	Sprawdził:			Skala: 1 : 1	Format: A2
	Zatwierdził:				Arkusz: 1/1
Uwagi: 1. 2. 3. 4.					

Załącznik nr 2 – „Kod źródłowy programu Arduino”

```
#define  A_PHASE 2
#define  B_PHASE 3
unsigned int flag_A = 1000;  //Assign a value to the token bit
unsigned int flag_B = 0;
double t_probkowania = 10; //czas próbkowania w mili-sekundach !!!!

/** *  */
void setup() {
  pinMode(A_PHASE, INPUT_PULLUP);
  pinMode(B_PHASE, INPUT_PULLUP);
  Serial.begin(9600);   //Serial Port Baudrate: 9600
  attachInterrupt(digitalPinToInterrupt( A_PHASE), interrupt, RISING);
}
void loop() {

  Serial.print(",");
  Serial.println(flag_A-flag_B);
  Serial.print(millis());
  delay(t_probkowania);

}
void interrupt()
{
  char i;
  i = digitalRead(B_PHASE);
  if (i == 1)
    flag_A += 1;
  else
    flag_B += 1;
}
```

Załącznik nr 3 – „kod źródłowy aplikacji w pythonie"

Plik gui.py:

```
from tkinter import *
from tkinter import ttk
import numpy as np
import mplcursors as mplcursors
from matplotlib import style, animation
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk as NavigationToolbar2TkAgg
from PIL import Image, ImageTk

import metody

style.use('ggplot')
f = Figure(figsize=(12, 9), dpi=100)
a = f.add_subplot(411)
b = f.add_subplot(412)
c1 = f.add_subplot(413)
d = f.add_subplot(414)


def animate(i):
   pullData = open('przemieszczenie.txt', 'r').read()
   pullData1 = open('predkosc.txt', 'r').read()
   pullData2 = open('przyspieszenie.txt', 'r').read()
   pullData3 = open('moc.txt', 'r').read()
   dataArray = pullData.split('\n')
   dataArray1 = pullData1.split('\n')
   dataArray2 = pullData2.split('\n')
   dataArray3 = pullData3.split('\n')
   xar = []
   yar = []
   xbr = []
   ybr = []
   xcr = []
   ycr = []
   xdr = []
   ydr = []
   for eachLine in dataArray:
       if len(eachLine) > 1:
           x, y = eachLine.split(',')
           xar.append(float(x))
           yar.append(float(y))
   for eachLine in dataArray1:
       if len(eachLine) > 1:
           x1, y1 = eachLine.split(',')
           xbr.append(float(x1))
           ybr.append(float(y1))
   for eachLine in dataArray2:
       if len(eachLine) > 1:
           x2, y2 = eachLine.split(',')
           xcr.append(float(x2))
           ycr.append(float(y2))
   for eachLine in dataArray3:
       if len(eachLine) > 1:
           x3, y3 = eachLine.split(',')
           xdr.append(float(x3))
           ydr.append(float(y3))
   a.clear()
   a.plot(xar, yar)
   b.clear()
   b.plot(xbr, ybr)
   c1.clear()
   c1.plot(xcr, ycr)
   d.clear()
   d.plot(xdr, ydr)

   return a, b, c1, d


class Application(Frame):

   def __init__(self, master=None):
       Frame.__init__(self, master)
       self.grid()
       self.master.title("Aplikacja do pomiaru mocy")
       frame1 = LabelFrame(master, text='Dane ćwiczącego',
width=150, height=100)
       frame1.configure(background='gray85')
       frame1.grid(row=0, column=0, sticky=W + E + N, padx=10,
pady=5)
       label = ttk.Label(master=frame1, text="Masa człowieka [kg]:
")
       label.grid(column=0, row=0, sticky=W, padx=10, pady=5)
       label1 = ttk.Label(master=frame1, text="Masa obciążenia [kg]:
")
       label1.grid(column=0, row=1, sticky=W, padx=10, pady=5)
       label2 = ttk.Label(master=frame1, text="Wzrost ćwiczącego
[cm]: ")
       label2.grid(column=0, row=2, sticky=W, padx=10, pady=5)
       frame1.txtEntry = metody.Prox(frame1, width=15)
       frame1.txtEntry1 = metody.Prox(frame1, width=15)
       frame1.txtEntry2 = metody.Prox(frame1, width=15)
       frame1.txtEntry.grid(column=1, row=0, padx =10, pady=5)
       frame1.txtEntry1.grid(column=1, row=1, padx=10, pady=5)
       frame1.txtEntry2.grid(column=1, row=2, padx=10, pady=5)
       btn = ttk.Button(master=frame1, text="Zacznij pomiary",
width=15,
                        command=lambda:
metody.clicked(frame1.txtEntry, frame1.txtEntry1, frame1.txtEntry2))
       btn.grid(column=0, row=5, sticky=W, padx=10, pady=10)
       btn1 = ttk.Button(frame1, text="Koniec pomiarów", width=15,
command=lambda: metody.stop())
       btn1.grid(column=1, row=5, sticky=E, padx=10, pady=10)
       btn2 = ttk.Button(frame1, text="Wygładź pomiary", width=15,
command=lambda: [metody.smoothing("przyspieszenie.txt", 81),

      metody.smoothing("predkosc.txt", 11),

      metody.smoothing('moc.txt', 41)])
       btn2.grid(column=0, row=6, sticky=W, padx=10, pady=10)
       btn3 = ttk.Button(frame1, text="Wyczyść pomiary", width=15,
command=lambda: metody.clean())
       btn3.grid(column=1, row=6, sticky=E, padx=10, pady=10)
       btn4 = ttk.Button(frame1, text="Eksport wyników",
command=lambda: metody.export(frame1.txtEntry, frame1.txtEntry1,
frame1.txtEntry2))
       btn4.grid(column=0, row=7, sticky=W+E+S+N, padx=10, pady=10,
columnspan=2)

       frame2 = Frame(master)
       frame2.grid(row=0, column=1, sticky=W + E + N + S, padx=5,
pady=10)
       f.text(0.5, 0.04, 'Czas [s]', ha='center', va='center')
       f.text(0.07, 0.4, 'Przyspieszenie [m/s^2]', ha='center',
va='center', rotation='vertical')
       f.text(0.07, 0.6, 'Prędkość [m/s]', ha='center', va='center',
rotation='vertical')
       f.text(0.07, 0.8, 'Przemieszczenie [m]', ha='center',
va='center', rotation='vertical')
       f.text(0.07, 0.2, 'Moc [W]', ha='center', va='center',
rotation='vertical')

       frame3 = LabelFrame(master, width=300, height=30)
       frame3.grid(column=0, row=0, sticky=W+S, padx=10, pady=20)
       label = ttk.Label(master=frame3, text="Autor: Maciej
Groszyk")
       label.grid(sticky=N+W+S+E)


       canvas = FigureCanvasTkAgg(f, frame2)
       canvas.draw()
       canvas.get_tk_widget().pack(side=BOTTOM, fill=BOTH,
expand=True)
       toolbar = NavigationToolbar2TkAgg(canvas, frame2)
       toolbar.update()
       canvas._tkcanvas.pack(side=TOP, fill=BOTH, expand=True)


root = Tk()
mchtr = Image.open('mchtr.png')
photo = ImageTk.PhotoImage(mchtr)
lab = Label(image=photo).grid(column=0, sticky=E+W)

root.geometry("1580x950")
root.resizable(False, False)
app = Application(master=root)
ani1 = animation.FuncAnimation(f, animate, interval=1)
app.mainloop()
```

Plik metody.py:

```
import threading
import tkinter as ttk
import re
import pandas as pd
import numpy as np
from PIL import Image, ImageTk
from itertools import count
from scipy.signal import savgol_filter
from datetime import date

import serial


class Prox(ttk.Entry):
    def __init__(self, master=None, **kwargs):
        super().__init__(master, **kwargs)
        self.var = ttk.StringVar(master)
        self.var.trace('w', self.validate)
        ttk.Entry.__init__(self, master, textvariable=self.var,
**kwargs)
        self.get, self.set = self.var.get, self.var.set

    def validate(self):
        value = self.get()
        if not value.isdigit():
            self.set(''.join(x for x in value if x.isdigit()))


def clicked(a, b, c):
    if len(a.get()) == 0 or len(b.get()) == 0 or len(c.get()) == 0:
        popupmsg("Wprowadź poprawne dane!")
    else:
        global keepGoing
        keepGoing = True
        th = threading.Thread(target=readData, args=(float(a.get()),
float(b.get()), float(c.get())))
        th.daemon = True
        th.start()

def popupmsg(msg):
    popup = ttk.Tk()
    popup.wm_title("ERROR")
    label = ttk.Label(popup, text=msg)
    label.pack(side="top", fill="x", pady=10)
    B1 = ttk.Button(popup, text="Okay", command=popup.destroy)
    B1.pack()
    popup.mainloop()


def clean():
    with open("przyspieszenie.txt", "a") as file:
        file.truncate(0)
        file.close()
    with open("predkosc.txt", "a") as file1:
        file1.truncate(0)
        file1.close()
    with open("przemieszczenie.txt", "a") as file3:
        file3.truncate(0)
        file3.close()
    with open("moc.txt", "a") as file4:
        file4.truncate(0)
        file4.close()


def stop():
    global keepGoing
    keepGoing = False


def smoothing(file, size):
    i = 0
    setlist = []
    result = ""
    with open(file, "r") as file1:
        for line in file1:
            inner_list = [elt.strip() for elt in line.split(',')]
            setlist.append(float(inner_list[1]))
    yhat = savgol_filter(setlist, size, 3)
    with open(file, "r") as file2:
        for line in file2:
            list = line.split(",")
            list[1] = str(yhat[i])
            line = ",".join(list)
            result += line + '\n'
            i = i + 1
    f = open(file, "w")
    f.write(result)
    f.close()


def export(mcz, mo, wz):
    today = date.today()
    d1 = today.strftime("%b-%d-%Y")
    df = pd.merge(exportDislocation(), exportVelocity(),
left_on=['seria'], right_on=['seria']).drop(
        columns=['Czas_x', 'Czas_y'])
    df = df.merge(exportAcceleration(), left_on=['seria'],
right_on=['seria'])
    df = df.merge(exportPower(), left_on=['seria'],
right_on=['seria']).drop(columns=['Czas_x', 'Czas_y'])
    df.to_csv(r'{0}_{1}_{2}_{3}'.format(mcz.get(), mo.get(),
wz.get(), d1))

def exportVelocity():
    time = []
    velocity = []
    with open('predkosc1.txt') as f:
        for line in f:
            time.append(float(line.split(',')[0]))
            velocity.append(float(line.split(',')[-1]))

    d = {'Predkosc': velocity, 'Czas': time}
    df = pd.DataFrame(d)
    df1 = pd.DataFrame(d)
    df3 = pd.DataFrame(d)
    df3[df3.Predkosc < 0] = 0
    minSample = df['Predkosc'].min()
    maxSample = df['Predkosc'].max()
    gap = maxSample - minSample
    normalized_sample = ((df['Predkosc'] - minSample) / gap)
    df3['Predkosc'] = normalized_sample
    normalized_sample = (normalized_sample - 0.5) * 2
    df['Predkosc'] = normalized_sample
    booleans = df.Predkosc > 0.8
    booleans.tolist()
    a = booleans[0]
    start = []
    end = []
    for idx, item in enumerate(booleans):
        if item != a and item == True:
            start.append(idx)
        elif item != a and item == False:
            end.append(idx)
        a = item
    for idx, items in enumerate(start):
        df1.loc[items:end[idx], 'seria'] = idx + 1
    df2 = df1.groupby(['seria']).max()

    booleans1 = df3.Predkosc > 0.01
    booleans1.tolist()
    a1 = booleans1[0]
    start1 = []
    end1 = []
    for idx, item in enumerate(booleans1):
        if item != a1 and item == True:
            start1.append(idx)
        elif item != a1 and item == False:
            end1.append(idx)
        a1 = item
    for idx, items in enumerate(start):
        df3.loc[items:end[idx], 'seria'] = idx + 1

    df4 = df3.groupby(['seria']).mean()
    df4 = df4.rename(columns={'Predkosc': 'Srednia predkosc'})

    df5 = pd.merge(df2, df4, left_on=['seria'], right_on=['seria'])
    return df5


def exportAcceleration():
    time = []
    acceleration = []
    with open('przyspieszenie.txt') as f:
        for line in f:
            time.append(float(line.split(',')[0]))
            acceleration.append(float(line.split(',')[-1]))

    d = {'Przyspieszenie': acceleration, 'Czas': time}
    df = pd.DataFrame(d)
    df1 = pd.DataFrame(d)
    minSample = df['Przyspieszenie'].min()
    maxSample = df['Przyspieszenie'].max()
    gap = maxSample - minSample
    normalized_sample = ((df['Przyspieszenie'] - minSample) / gap)
    normalized_sample = (normalized_sample - 0.5) * 2
    df['Przyspieszenie'] = normalized_sample
    booleans = df.Przyspieszenie > 0.8
    booleans.tolist()
    a = booleans[0]
    start = []
    end = []
    for idx, item in enumerate(booleans):
        if item != a and item == True:
            start.append(idx)
        elif item != a and item == False:
            end.append(idx)
        a = item
    for idx, items in enumerate(start):
        df1.loc[items:end[idx], 'seria'] = idx + 1

    df2 = df1.groupby(['seria']).max()
    return df2



def exportPower():
    time = []
    power = []

    with open('moc.txt') as f:
        for line in f:
            time.append(float(line.split(',')[0]))
            power.append(float(line.split(',')[-1]))

    d = {'Moc': power, 'Czas': time}
    df = pd.DataFrame(d)
    df1 = pd.DataFrame(d)
    minSample = df['Moc'].min()
    maxSample = df['Moc'].max()
    gap = maxSample - minSample
    normalized_sample = ((df['Moc'] - minSample) / gap)
    normalized_sample = (normalized_sample - 0.5) * 2
    df['Moc'] = normalized_sample
    booleans = df.Moc > 0.8
    booleans.tolist()
    a = booleans[0]
    start = []
    end = []
    for idx, item in enumerate(booleans):
        if item != a and item == True:
            start.append(idx)
        elif item != a and item == False:
            end.append(idx)
        a = item
    for idx, items in enumerate(start):
        df1.loc[items:end[idx], 'seria'] = idx + 1

    df2 = df1.groupby(['seria']).max()
    return df2


def exportDislocation():
    time = []
    dislocation = []
    with open('przemieszczenie.txt') as f:
        for line in f:
            time.append(float(line.split(',')[0]))
            dislocation.append(float(line.split(',')[-1]))

    d = {'Przemieszczenie': dislocation, 'Czas': time}
    df = pd.DataFrame(d)
    df1 = pd.DataFrame(d)
    duplicate = pd.DataFrame(d)
    duplicate = duplicate.duplicated(["Przemieszczenie"])
    duplicate.to_csv(r'testy.csv')
    lista = df1.Przemieszczenie.tolist()

    minSample = df['Przemieszczenie'].min()
    maxSample = df['Przemieszczenie'].max()

    gap = maxSample - minSample
    normalized_sample = ((df['Przemieszczenie'] - minSample) / gap)
    normalized_sample = (normalized_sample - 0.5) * 2
    df['Przemieszczenie'] = normalized_sample
    booleans = df.Przemieszczenie < -0.8
    booleans.tolist()
    a = booleans[0]
    start = []
    end = []
    for idx, item in enumerate(booleans):
        if item != a and item == True:
            start.append(idx)
        elif item != a and item == False:
            end.append(idx)
        a = item
    for idx, items in enumerate(start):
        df1.loc[items:end[idx], 'seria'] = idx + 1

    df2 = df1.groupby(['seria']).min()
    ar = df2.Przemieszczenie.tolist()
    df2 = df2 * -1
    arboolean = []
    aridx = []
    arvalue = []
    arvalue2 = []
    a = 0
    for idx1, item1 in enumerate(lista):
        if idx1 == 0:
            arboolean.append(False)
        else:
            if a == 2:
                print(lista[idx1])
                print(arboolean[-1])
            if lista[idx1] != ar[a] and arboolean[-1] == False:
                arboolean.append(False)
            elif lista[idx1] > ar[a] and arboolean[-1] == True:
                if lista[idx1] >= lista[idx1 - 1]:
                    arboolean.append(True)
                else:
                    arboolean.append(False)
                    aridx.append(idx1 - 1)
                    v1 = (ar[a] - lista[idx1 - 1]) * -1

                    arvalue2.append(lista[idx1-1])
                    arvalue.append(v1)
                    if len(ar) - 1 > a:
                        a = a + 1
            else:
                arboolean.append(True)

    print(arvalue)

    print(arvalue2)
    print(ar)

    df2.Przemieszczenie = arvalue
    return df2.drop(columns='Czas')


def readData(m1, m2, h1):  # m1 - masa ciezaru, m2-masa miesni h1-
dlugosc ciala

    try:
        arduinoData = serial.Serial("/dev/ttyACM0", timeout=1)
    except:
        print('Please check the port')
    clean()

    timestamp = str(serial.time.time())
    count = 0
    rawdata = []
    P = []
    J = 0.000193152  # moment bezwładnosci krazka podany w kg*m^2
    r = 0.0625  # promien krazka w m +
    m_sum = 7.4484 + 0.76694 * m2 - 0.05192 * h1 + m1  # dla masy
ciala 80, ciezaru 50, wzrost 180 - 109,458kg
    g = 9.81  # przyspieszenie ziemnskie w m/s^2
    A = J + (m_sum * r ** 2)
    B = m_sum * g * r
    Fsmyczy = 150 / 1000 * g
    C = Fsmyczy * r
    fik = 0  # fi aktualne
    fik_1 = 0  # fi poprzednie
    fik_2 = 0  # fi poprzednie (fi akutalne -2)
    t_s = 0.01  # czas probkowania w sek
    P_lift = 0
    fikakt = 0
    fik_pop = 0
    fik_poppop = 0
    t = 0
    t1 = 0
    tc = 0
    v = 0
    i = 0
    a = 0
    while keepGoing:
        c = [float(s) for s in re.findall('\\d+',
str(arduinoData.readline()))]  # Wczytanie outputu z arduino
        if c:
            i = i + 1
            # OBLICZENIA
            fik_poppop = fik_pop
            fik_pop = fikakt
            fikakt = c[-1]

            fik_2 = fik_1
            fik_1 = fik
            fik = ((c[-1] - 1000) / 200) * 3.14  # odczyt kąta w
radianachw

            t1 = t
            t = c[0] / 1000
            t_s = t - t1
            if t_s > 1 or t_s == 0 or t_s < 0.0001:
                t_s = 0.01
            v1 = v
            v = (fik - fik_1) * r / t_s
            a1 = a
            a = (v - v1) / t_s
            tc = tc + t_s
            fi_prim = (fik - fik_1) / t_s  # pierwsza pochodna kąta
            fi_bis = (fik - 2 * fik_1 + fik_2) / (
                    t_s ** 2)  # druga pochodna kąta

            P_lift = A * fi_prim * fi_bis + B * fi_prim + C *
fi_prim  # moc całkowita wyrażona w W

            with open("przemieszczenie.txt", "a") as file:
                file.write("{},{}\n".format(tc, fik * r))
                file.close()
            with open("predkosc.txt", "a") as file1:
                file1.write("{},{}\n".format(tc, v))
                file1.close()
            with open("przyspieszenie.txt", "a") as file2:
                file2.write("{},{}\n".format(tc, a))
                file2.close()

            with open("moc.txt", "a") as file3:
```

```
file3.write("{},{}\n".format(tc, P_lift))
file3.close()
```