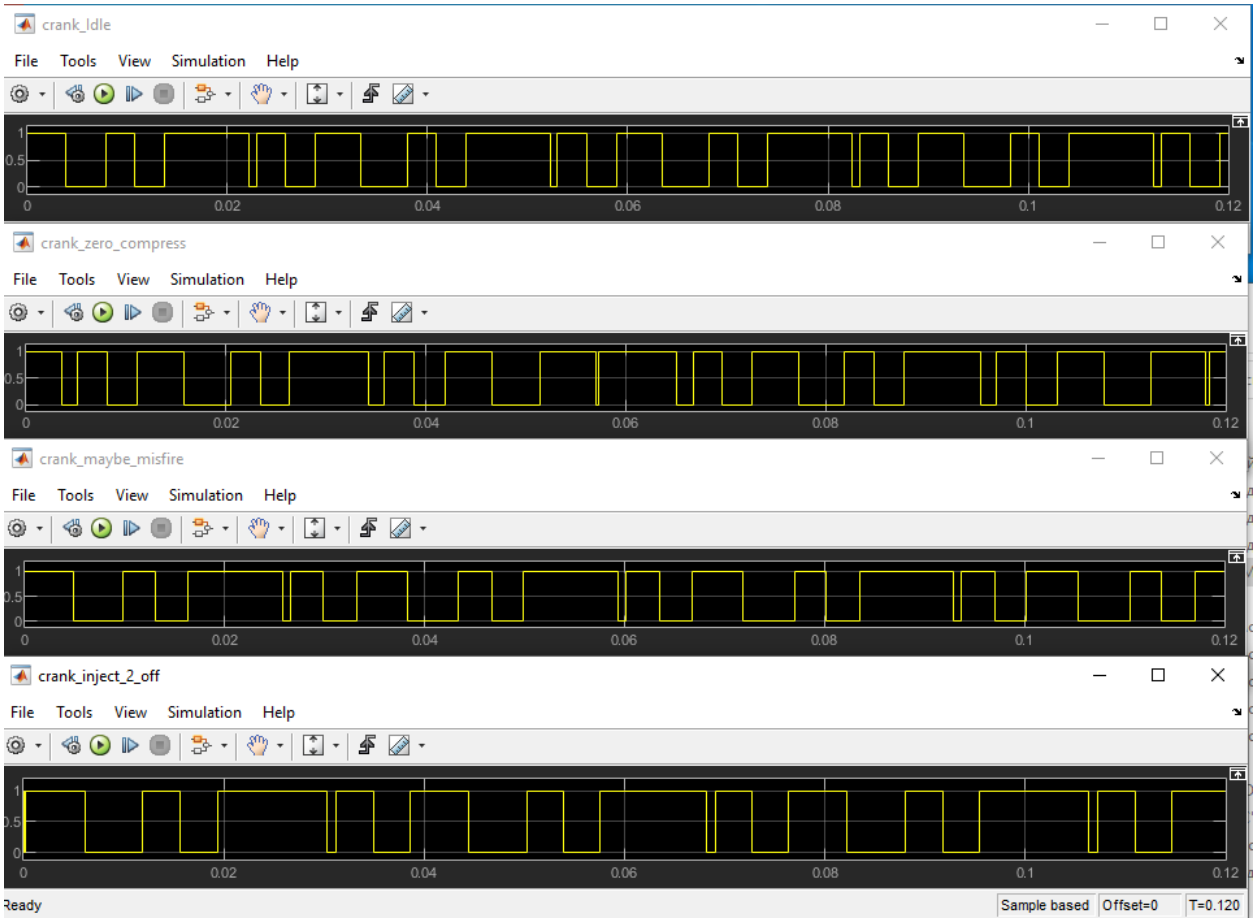
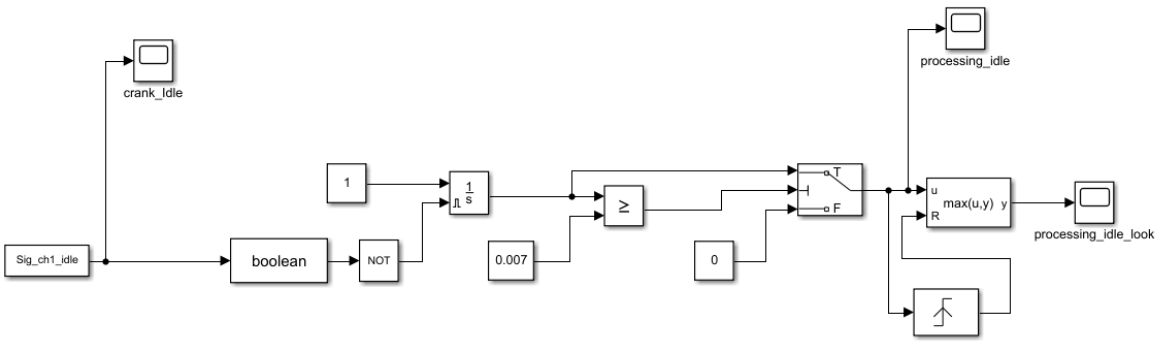
Content

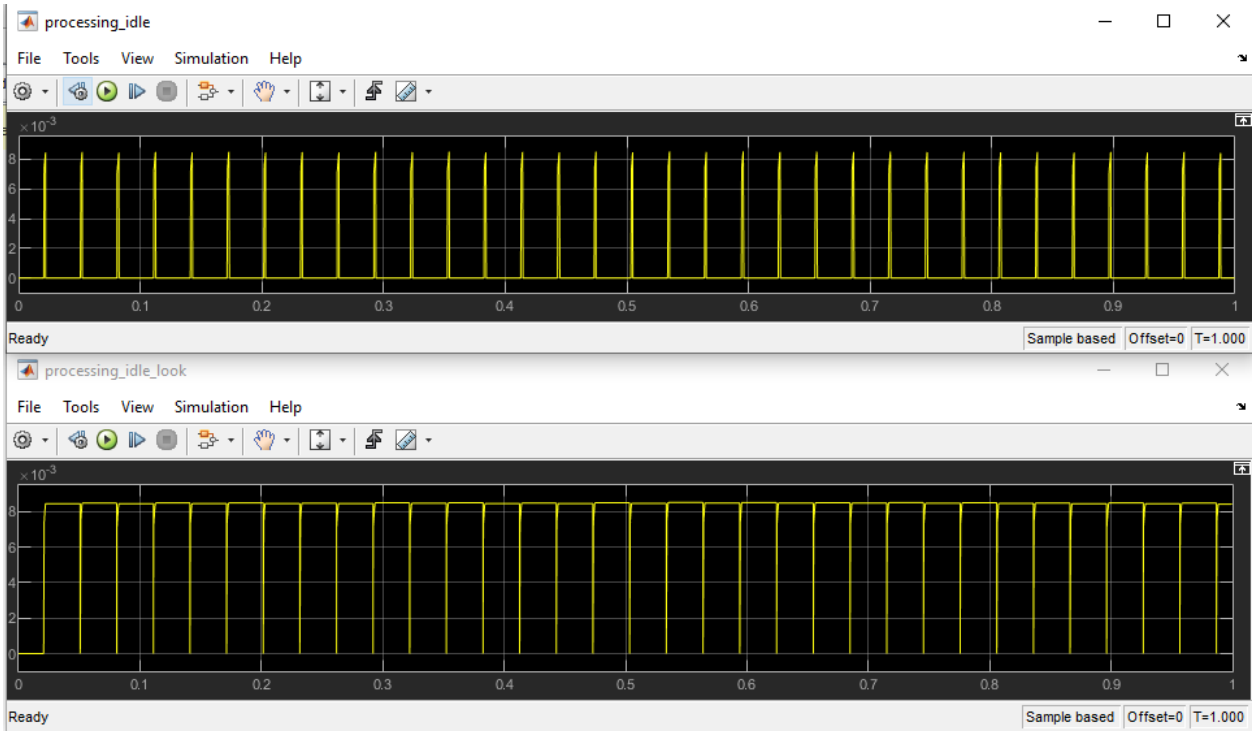
Part 1 Matlab Simulink processing



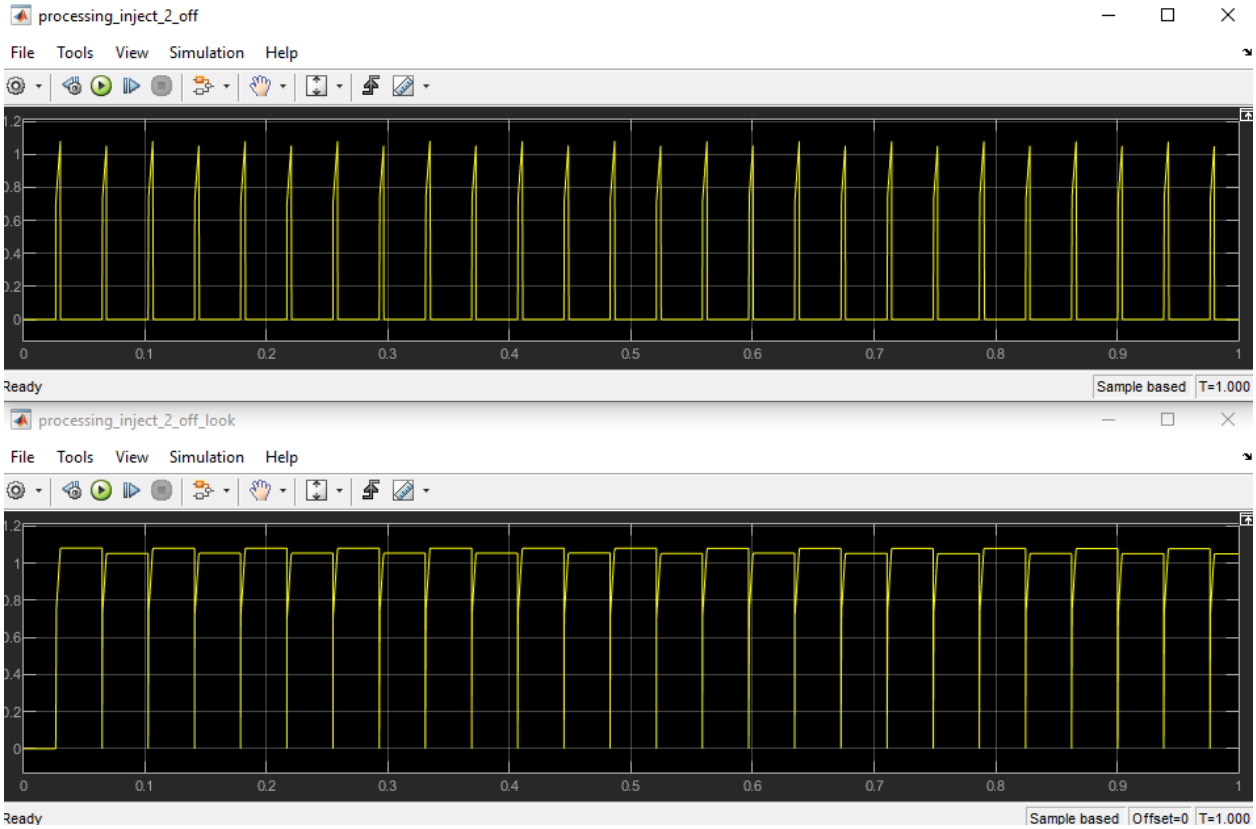
Raw_signal_of_crank (matlab simulink), duration 0.12 s



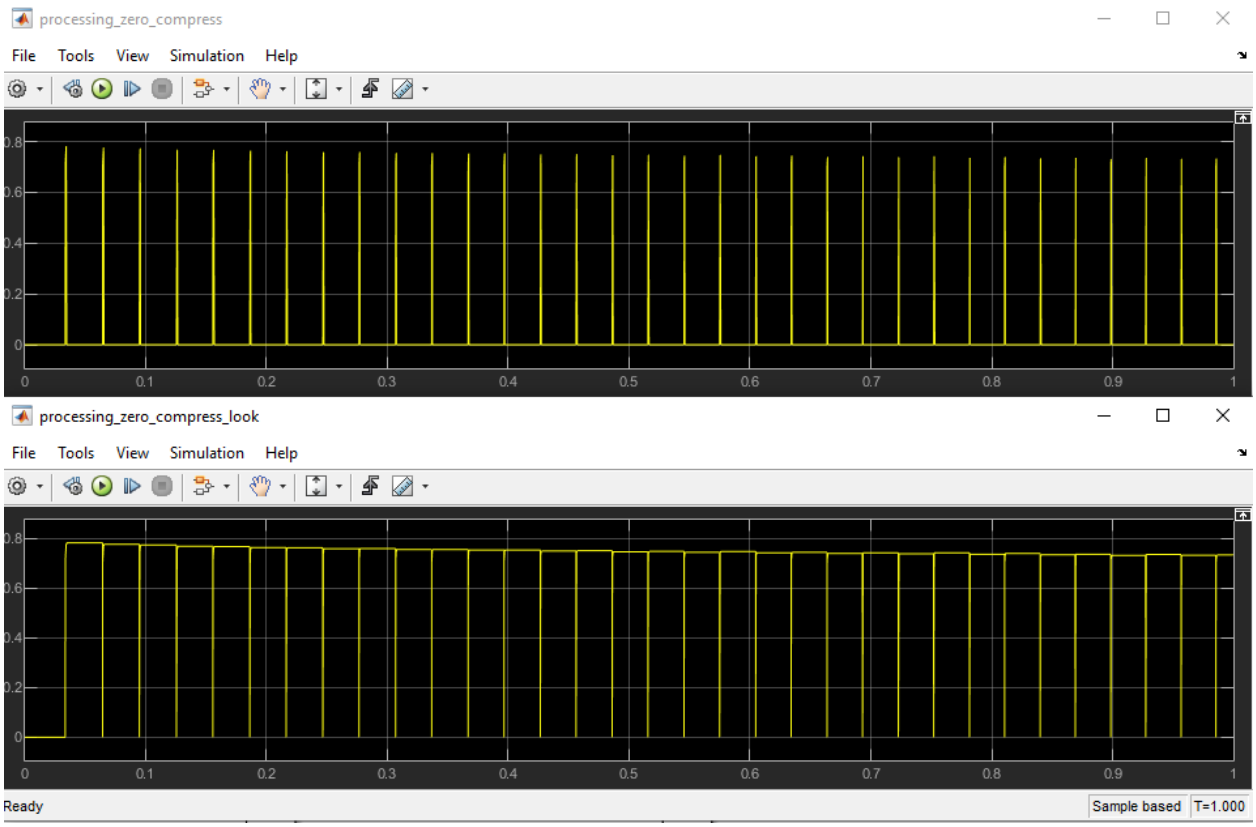
Simulink model



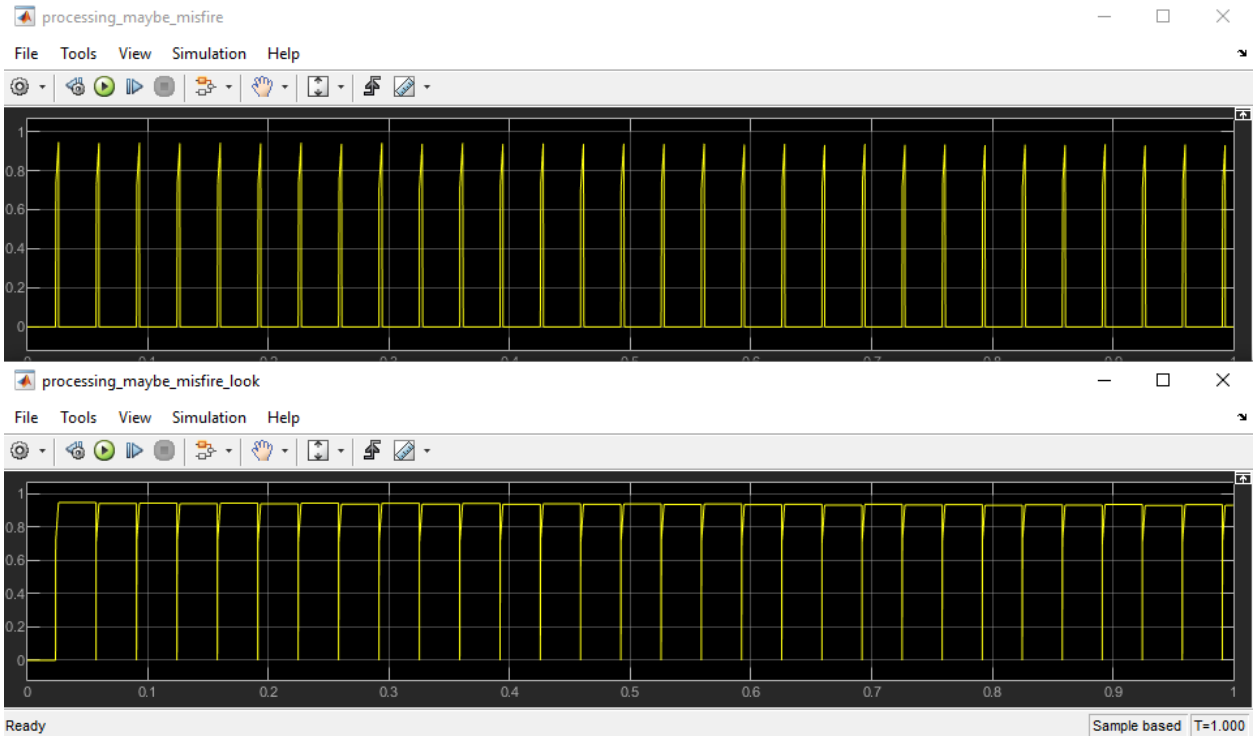
Result idle signal processing, duration 1 s



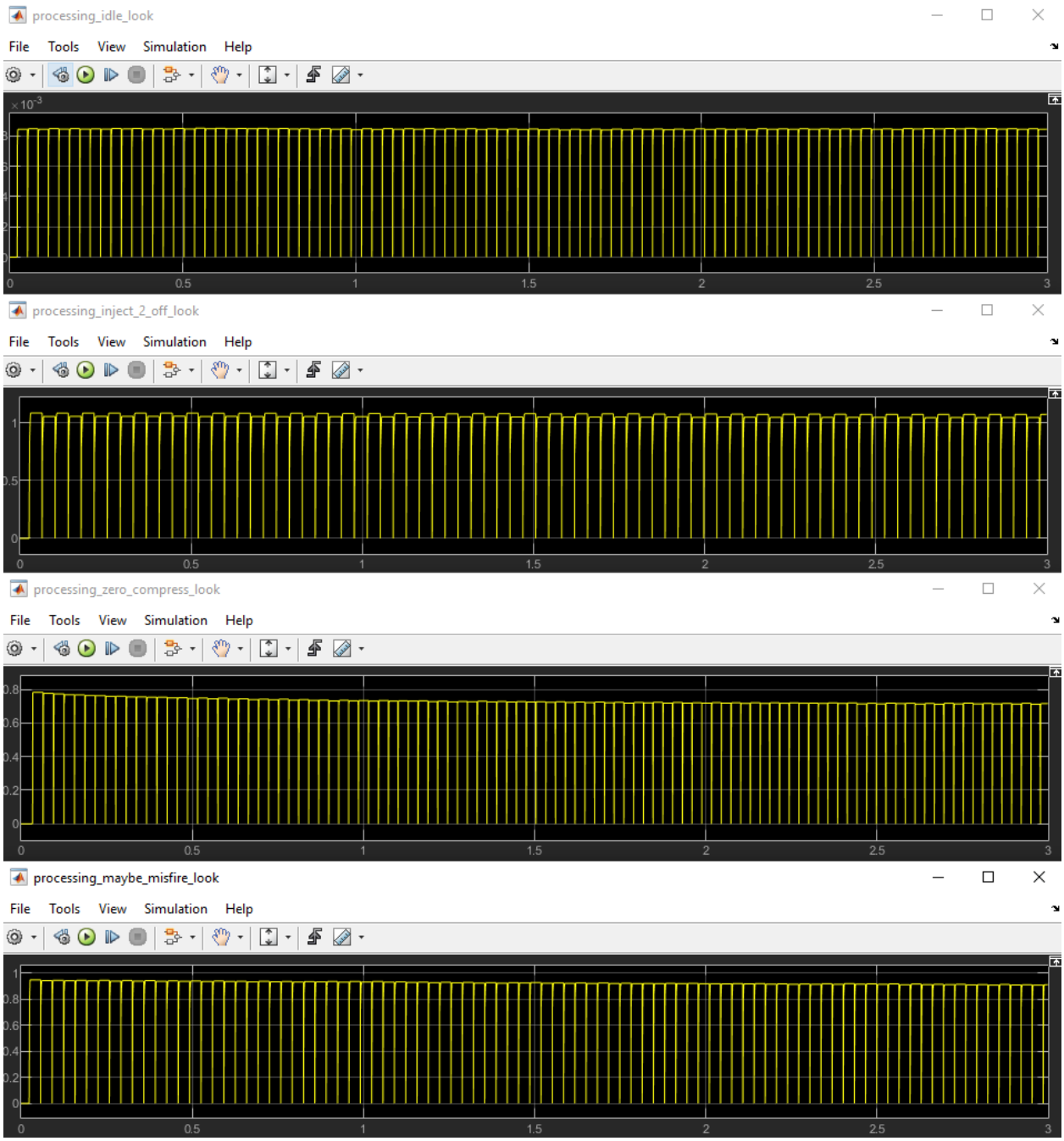
Result inject_2 off signal processing, duration 1 s



Result zero compressing signal processing, duration 1 s



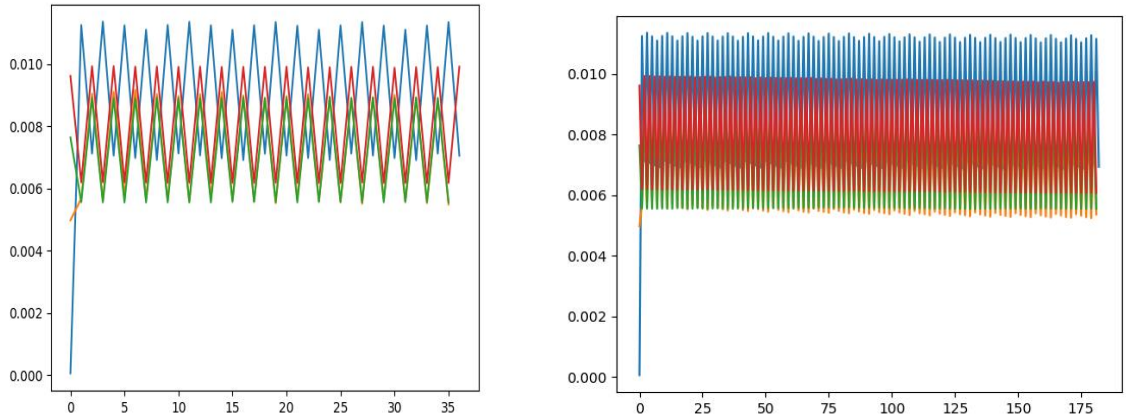
Result maybe misfire signal processing, duration 1 s



Signals processing, duration 3 s

Part 2 Python processing

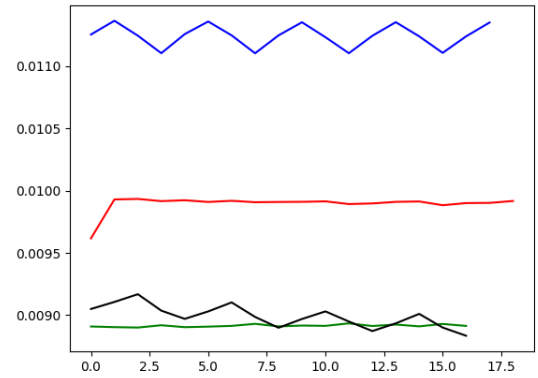
Period calc by crank signal and time series data:

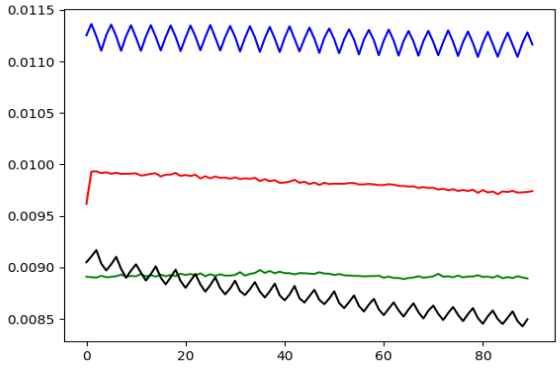


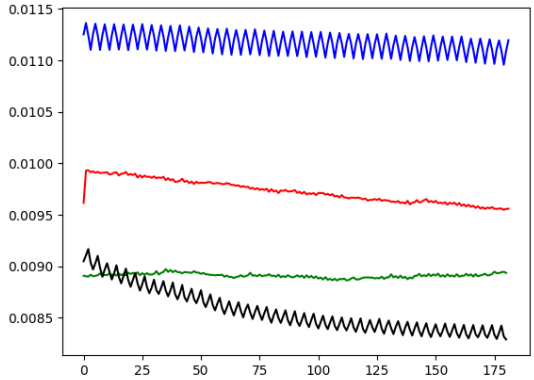
green – idle, blue – inject 2 off, yolo (bed look on picture..) – zero compression, red – suspected bad sparkplug

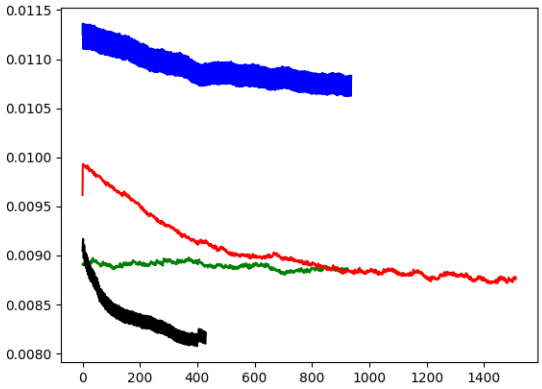
About Y axis – calc period, About X axis – number of data

Filtered by level (0.007 s) period data:





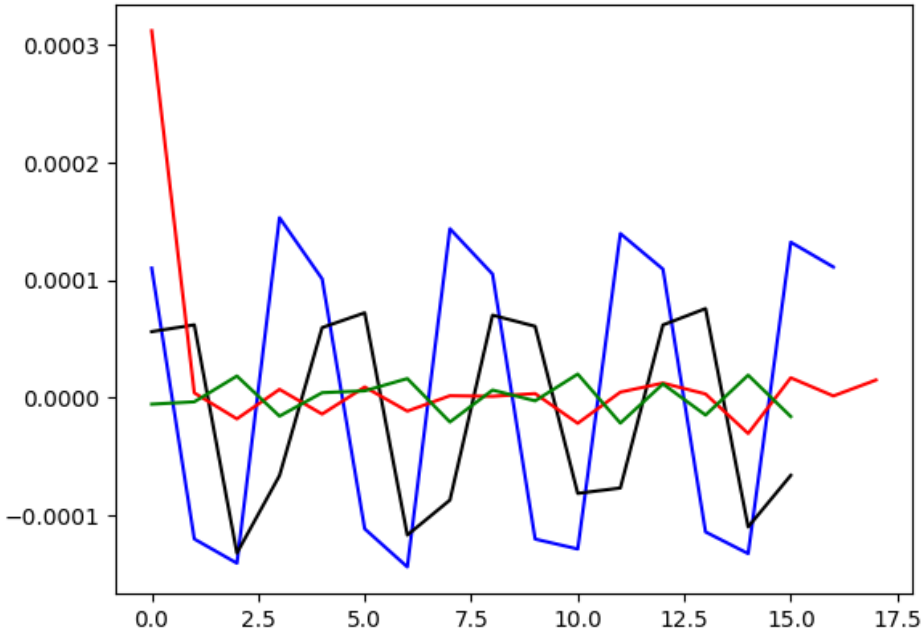


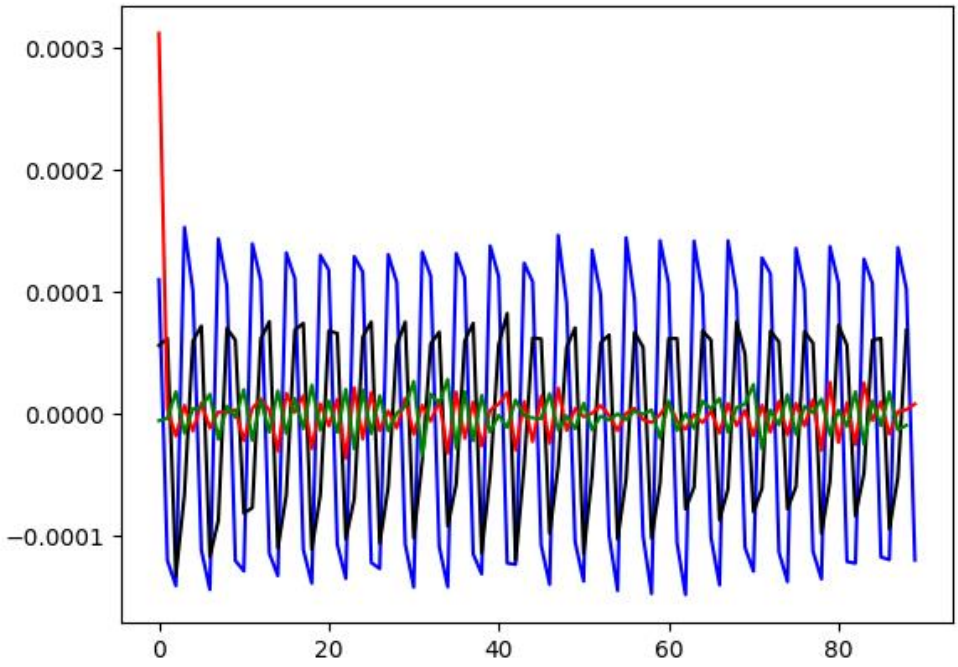


green – idle, blue – inject 2 off, black – zero compression, red – suspected bad sparkplug

About Y axis – calc period, About X axis – number of data

Filtered and diff period data:

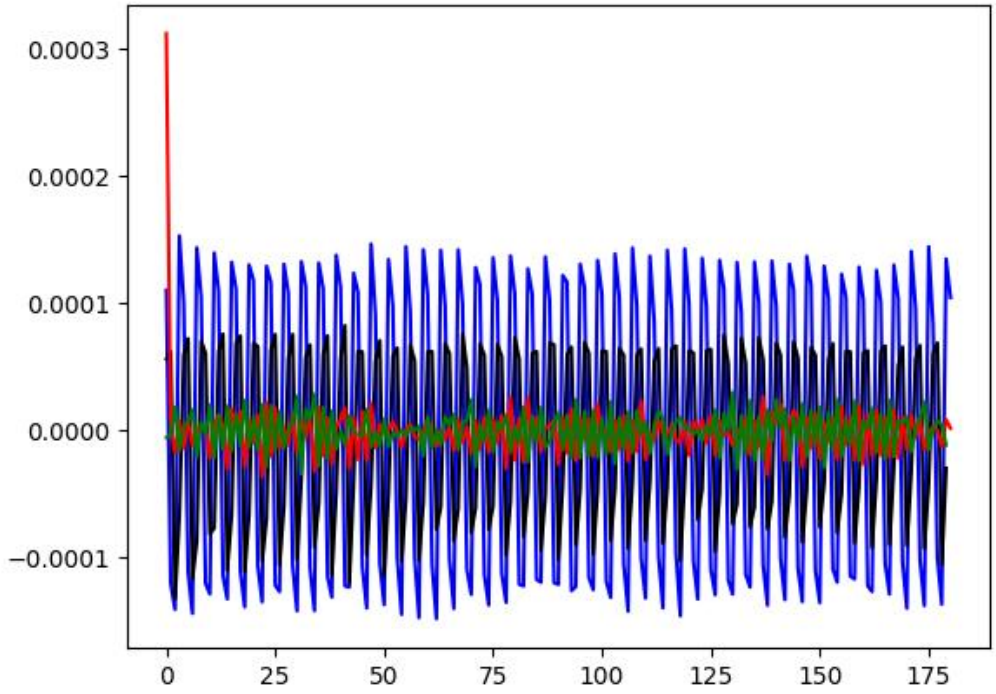


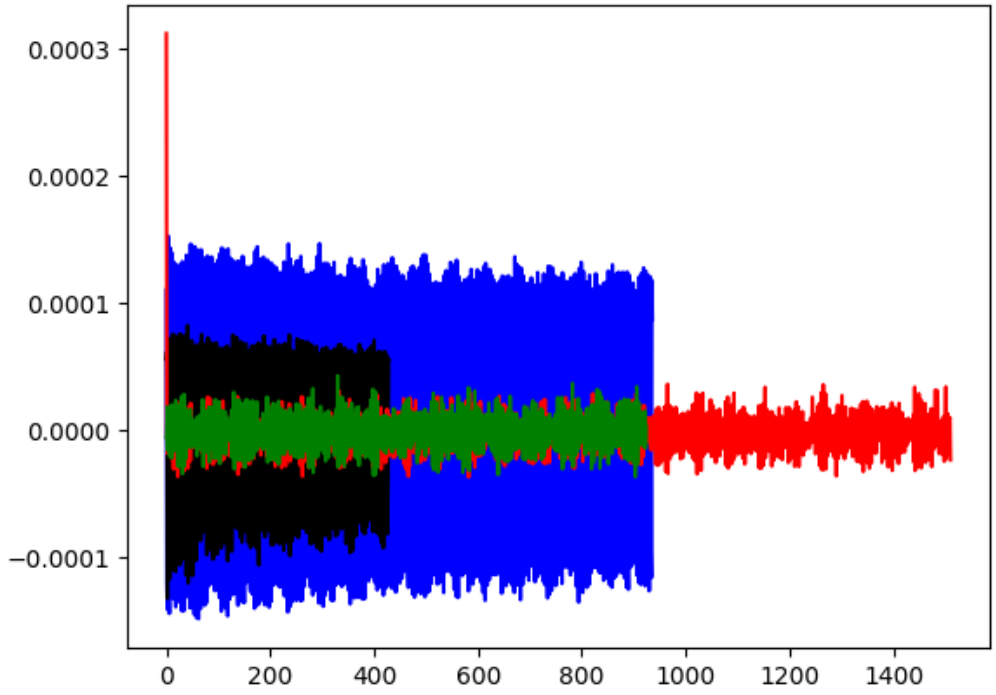


green – idle, blue – inject 2 off, black – zero compression,

red – suspected bad sparkplug

(see next page)





green – idle, blue – inject 2 off, black – zero compression,

red – suspected bad sparkplug

About Y axis – calc period, About X axis – number of data

Part 3 Python code

```
import csv
import numpy as np
from matplotlib import pyplot as plt



path_file_idle = r'C:\Users\home\Documents\nb2-happy-warmup-idle.csv'
path_file_miss = r'C:\Users\home\Documents\nb2-idle-without-injector-2.csv'
path_file_zero = r'C:\Users\home\Documents\nb2-sparkplug-1-completely-removed-zero-compression-in-1.csv'
path_file_maby = r'C:\Users\home\Documents\nb2-suspected-bad-sparkplug-1.csv'



def parse_data(path_file_idle):
    data = np.array([[]])
    time_s = []
    cam = []
    crank = []
    with open(path_file_idle) as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            cam.append(row['cam'])
```

```
            time_s.append(row['Time [s]'])
            crank.append(row['crank'])

    cam = np.array(cam)
    time_s = np.array(time_s)
    crank = np.array(crank)

    cam = cam.astype(int)
    time_s = time_s.astype(float)
    crank = crank.astype(int)

    #plt.plot(crank[:100])

    data = np.vstack((crank, time_s)).T
    print(data.shape)
    return data

data_idle = parse_data(path_file_idle)
data_miss = parse_data(path_file_miss)
data_zero = parse_data(path_file_zero)
data_maby = parse_data(path_file_maby)



def invert_value(in_val):
    out_val = []
```

```
    for i in range(in_val.shape[0]):
        if in_val[i] == 0:
            out_val.append(1)
        else:
            out_val.append(0)
    return out_val


def calc_period(data):
    sig_period = []
    t_0 = data[0, 1]
    t_1 = float()
    print('perid in data len --> ', data.shape[0])
    for i in range(data.shape[0]):# 100
        if i == 0:
            crank_current = data[0, 0]
            print('ferst_data')
            pass
        crank_mem = crank_current
        crank_current = data[i, 0]

        if crank_current == crank_mem:
            pass

        if crank_current > crank_mem:
            t_0 = data[i, 1]
```

```
        if crank_current < crank_mem:
            t_1 = data[i, 1]
            period = t_1 - t_0
            sig_period.append(period)
        '''
        if i == 2000:
            break
        '''



    sig_period = np.array(sig_period)

    sig_period = sig_period.astype(float)
    print(sig_period.shape)
    return sig_period

sig_period_idle = calc_period(data_idle)

sig_period_miss = calc_period(data_miss)

sig_period_zero = calc_period(data_zero)

sig_period_maby = calc_period(data_maby)

#print(sig_1)
```

```
plt.figure('period')
plt.plot(sig_period_miss)
plt.plot(sig_period_zero)
plt.plot(sig_period_idle)
plt.plot(sig_period_maby)

level = 0.008
def filt_level(level, data_in):
    data_out = []
    for i in range(data_in.shape[0]):
        if data_in[i] >= level:
            data_out.append(data_in[i])
        else:
            pass
    return data_out

filt_level_data_idle = filt_level(level, sig_period_idle)

filt_level_data_miss = filt_level(level, sig_period_miss)

filt_level_data_zero = filt_level(level, sig_period_zero)

filt_level_data_maby = filt_level(level, sig_period_maby)
```

```
plt.figure('filt')
plt.plot(filt_level_data_idle, color='green')
plt.plot(filt_level_data_miss, color='blue')
plt.plot(filt_level_data_zero, color='black')
plt.plot(filt_level_data_maby, color='red')


filt_and_diff_level_data_idle = np.diff(filt_level_data_idle)
filt_and_diff_level_data_miss = np.diff(filt_level_data_miss)
filt_and_diff_level_data_zero = np.diff(filt_level_data_zero)
filt_and_diff_level_data_maby = np.diff(filt_level_data_maby)

plt.figure('filt_and_diff')
plt.plot(filt_and_diff_level_data_miss, color='blue')
plt.plot(filt_and_diff_level_data_zero, color='black')
plt.plot(filt_and_diff_level_data_maby, color='red')
plt.plot(filt_and_diff_level_data_idle, color='green')


sensor_level = 0.00007
def sensor(sensor_level, data_in):
    miss = []
    miss_value = []
    print('count_cycle -->', data_in.shape[0])
    for i in range(data_in.shape[0]):
```

```
        if i == 0:
            val_current = data_in[0]
            pass
        val_mem = val_current
        val_current = data_in[i]

        if abs(val_current - val_mem) >= sensor_level:
            miss.append(i)
            miss_value.append(abs(val_current - val_mem))
    print('count miss --> ', len(miss))
    if len(miss) <= 30:
        print('numba_of_cycle_miss -->', miss)
        print('numba_of_cycle_miss_value -->', miss_value)
    return miss

miss_idle = sensor(sensor_level, filt_and_diff_level_data_idle)
miss_miss = sensor(sensor_level, filt_and_diff_level_data_miss)
miss_miss = sensor(sensor_level, filt_and_diff_level_data_zero)
miss_maby = sensor(sensor_level, filt_and_diff_level_data_maby)

def SKO():
    pass

plt.show()
```

```
#print(cam_idle)
#print(time_idle)
#print(data_idle)
```