



TREND TRACKING SOFTWARE DOCUMENTATION

16.10.2019

İçindekiler

GİRİŞ	3
PROGRAM ALGORİTMASI.....	4
PROGRAMA GENEL BAKIŞ	5
DOSYANIN BULUNMASI VE ANALİZ	9
File Search Modülü	9
“.csv” Uzantılı Dosyalar Nasıl Tespit Edilir ?	10
“.csv” Dosyası İçindeki Veriler Nasıl Ayrıştırılır ?	10
Veriayır() Fonksiyonu	11
VERİ ANALİZ MODÜLÜ	12
Analiz Class’ı.....	12
Analiz Classındaki İşlemler	13
Denetle Fonksiyonu	13
Açı Bul fonksiyonu	13
Peak Detect Fonksiyonu	16
Hata Arama Fonksiyonları	18
Basınç Hatası Bulmak.....	18
Enjeksiyon Süre Hatası Bulmak	18
Peak Sayısı Hatası Bulmak	18
Hata Kayıt	18
Verilere Ait Grafik Oluşturma	19
SETTINGS MODÜLÜ.....	19
KENDİ ANALİZ SCRIPT’İNİZİ OLUŞTURUN	21

GİRİŞ

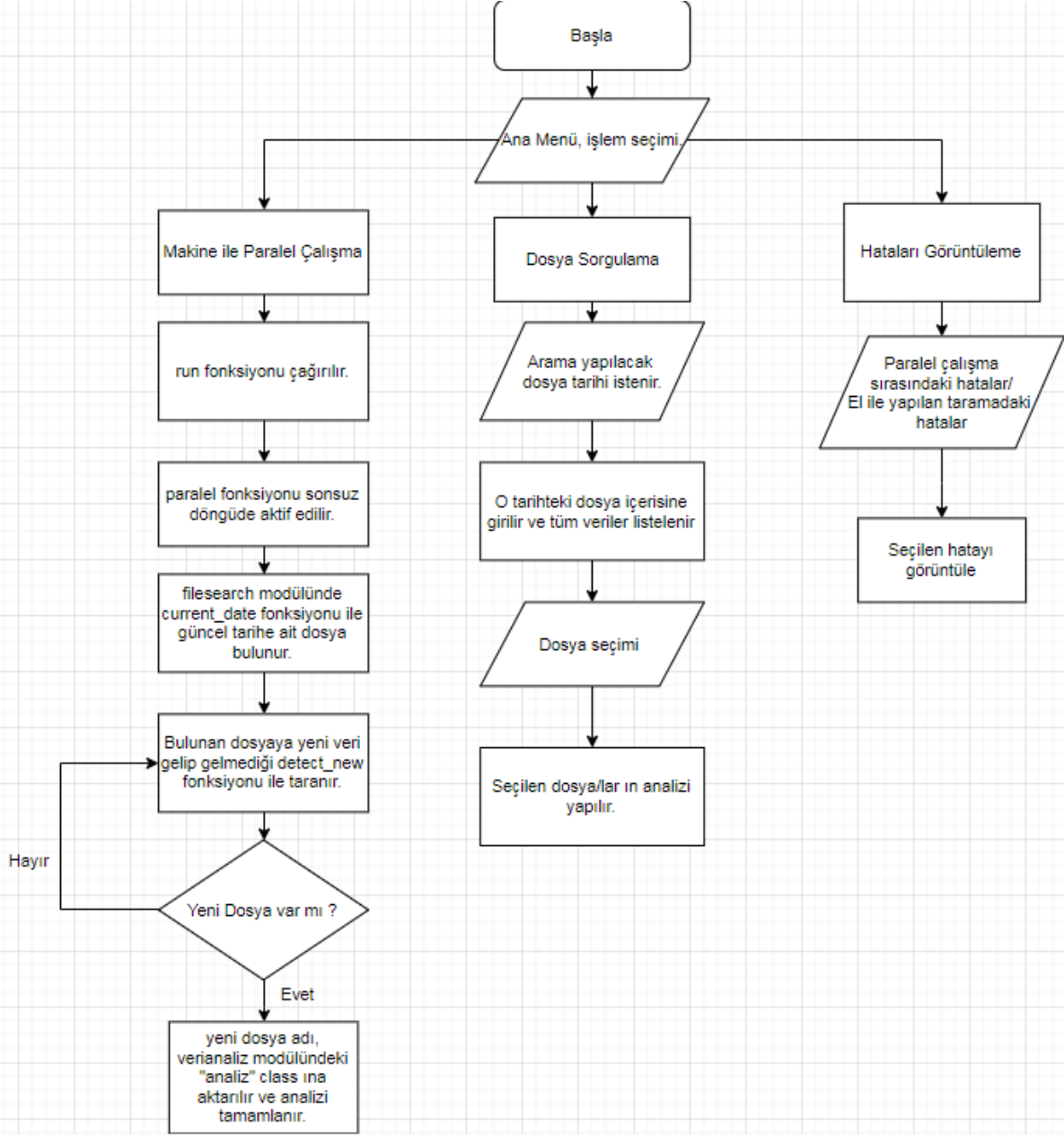
Trend Tracker, zaman domaininde oluşturulan verilerin analizini sağlayan yazılımdır. Veriler için doğru kabul edilen koşullar programa tanıtılır ve programın diğer tüm veriler için bu koşullar ile sorgu yapılması sağlanır. Bu sayede birçok veri kısa bir sürede analiz edilmiş ve hatalı veriler ayrılmış olur.

Trend Tracker'in bu versiyonunda plastik enjeksiyon yapan bir makineden çıkan sensör verilerinin analizi yapılmıştır. Makineden alınan veriler bir csv dosyasına sensör 1, sensör ve zaman verileri virgül ile ayrılarak kaydedilir. TT (trend tracker) bu veriyi parçalar ve belirli registerlara kaydeder. Kaydedilen değerler ile veri grafiği oluşturulur ve verilerin hareket açıları hesaplanır(yükselme, alçalma miktarı). Bu sonuçlara göre grafikten zirve değerleri, tepe sayısı gibi bilgiler elde edilir.

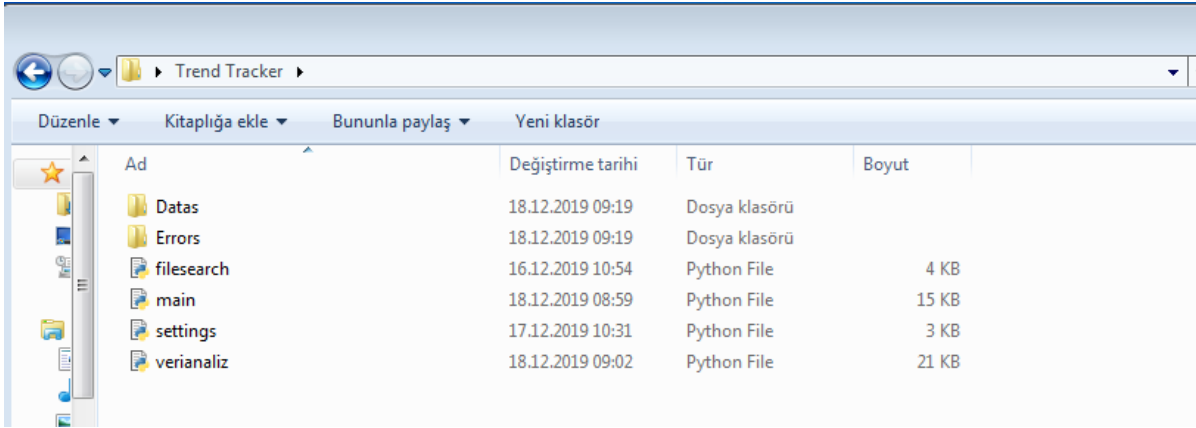
Makine hakkında çalışma koşulları programa yazıldıktan sonra TT, verilerin raporlanmasını, makineyi canlı takip ederek veya el ile yapılan dosya sorgulamaları aracılığıyla gerçekleştirir.

Program, Python dili üzerinde yazılmıştır. GUI için tkinter, grafik oluşturmak için matplotlib, csv dosyalarını açabilmek için csv, modüllerinden yararlanılmıştır

PROGRAM ALGORİTMASI



PROGRAMA GENEL BAKIŞ



Ad	Değiştirme tarihi	Tür	Boyut
Datan	18.12.2019 09:19	Dosya klasörü	
Errors	18.12.2019 09:19	Dosya klasörü	
filesearch	16.12.2019 10:54	Python File	4 KB
main	18.12.2019 08:59	Python File	15 KB
settings	17.12.2019 10:31	Python File	3 KB
verianaliz	18.12.2019 09:02	Python File	21 KB

Program 4 modül ve 2 klasörden oluşmaktadır :

- Main.py modülü ile program başlar. Ana menü GUI'si buradadır.
- Verianaliz modülü, data dosyasını açar ve verilerin analizini gerçekleştirir.
- Filesearch modülü, yeni dosyanın gelip gelmediğini kontrol eder ve yeni dosya geldiğinde ana modüle dosya adı ve yolunun bilgisini gönderir.
- Settings modülü, analiz ile ilgili parametreleri ayarlamak için kullanılır.
- Datan klasörü, analiz edilecek dosyaları içeren klasördür. Veri kaynağından çıkan verilerin direkt olarak buraya aktarılması gerekmektedir.
- Errors klasörü , analizden elde edilen hataları içeren klasördür.

Main Modülü

Programın başladığı modüldür. 3 ana işlemi gerçekleştirmek için gerekli fonksiyonları içerisinde barındırır. Bu modülde dışarıdan import edilen diğer modüller :

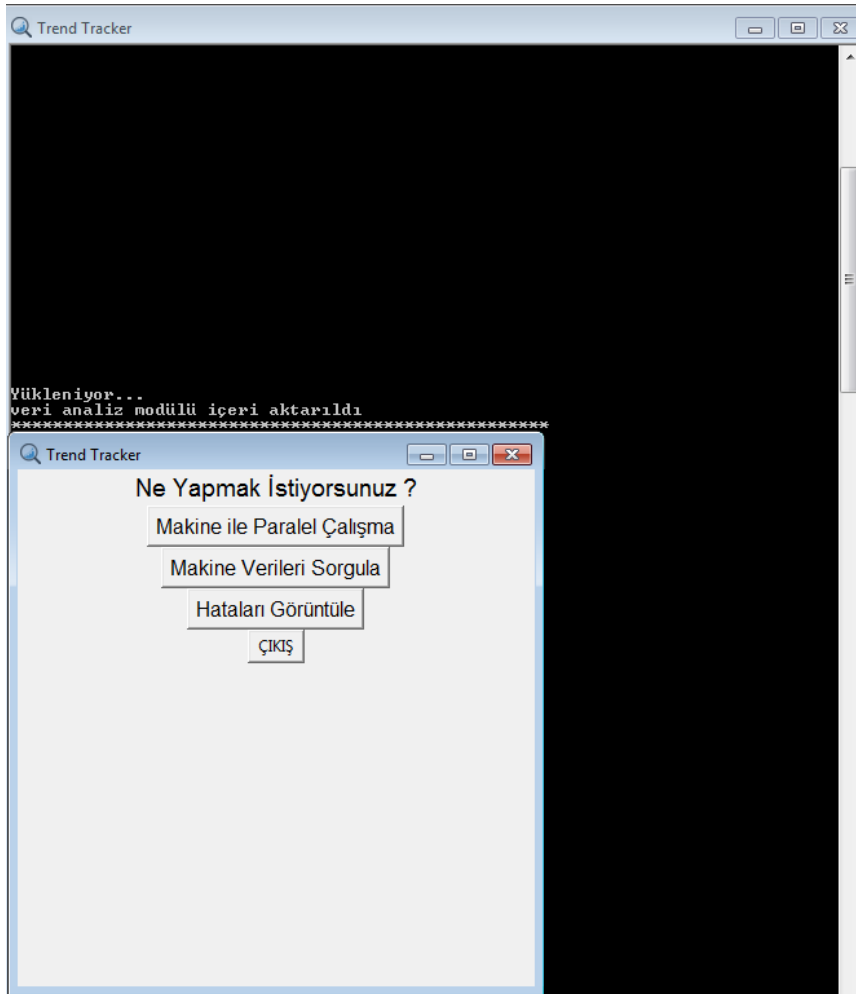
Os ve Sys : Sistem komutlarını çalıştırabilmek için kullanıldı.

Tkinter : GUI Tasarımı için kullanıldı.

Threading : Birden fazla proses oluşturup aynı anda yürütebilmek için kullanıldı.

Time : Zamansal fonksiyonları kullanabilmek için.

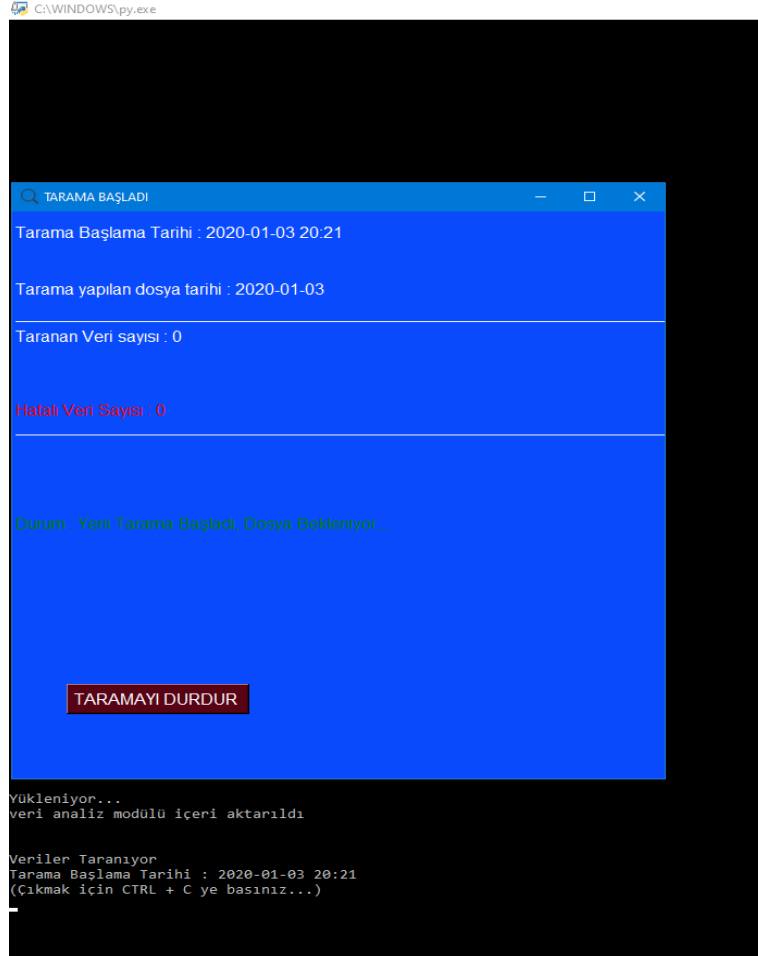
Main modülü çalıştırıldığında ilk çağırılan fonksiyon “Menu” fonksiyonudur. Program çalıştırıldığında durum aşağıdaki gibidir.



Menüdeki seçenekler programın 3 ana işlemini belirtir.

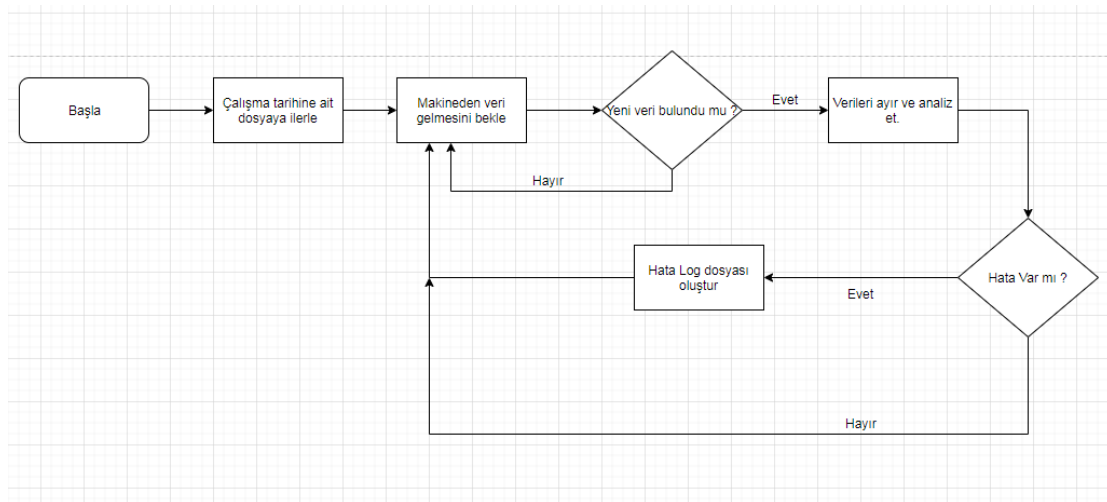
- 1) Makine ile Paralel Çalışma
- 2) Makine Verileri Sorgula
- 3) Hataları Görüntüle

Makine ile Paralel Çalışma



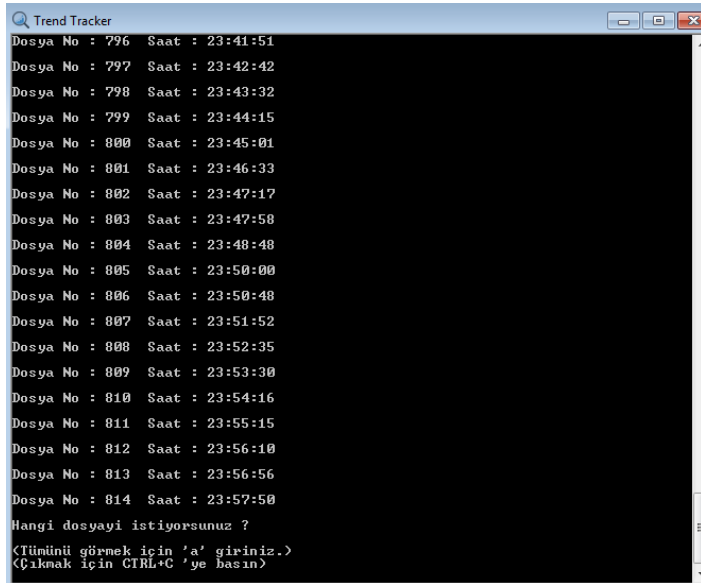
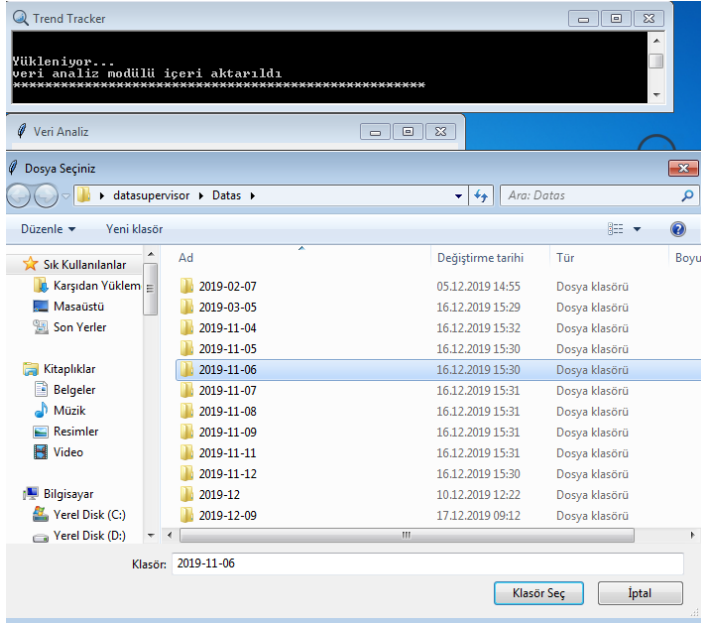
Program bu fonksiyona geçtiğinde makineden veri bekler. Veri geldiğinde “verianaliz” modülüne dosya bilgilerini gönderir ve analiz yapılır.

Makine ile Paralel Çalışma Akışı



Makine Verileri Sorgula

Veri sorgulama fonksiyonu ile seçilen bir tarihteki veriler sorgulanabilir. Sorgulanan verilerin grafikleri çıkartılabilir. İstenirse gün içinde bulunan tüm dosyalar tek bir taramada analiz edilebilir. 1000 adet verilik bir dosyayı 17 saniyede taramayı başarmıştır. Bu süre bilgisayarın hızına göre ve print edilen veri sayısına göre değişmektedir. Yapılan denemelerde terminale sürekli olarak veri print etmenin programın çalışma süresini iki katına kadar çıkardığı gözlemlenmiştir. Ayrıca programı Python Shell’ de çalıştırmak da çalışma süresini uzatan etkenler arasındadır.



Hataları Görüntüle

Programın otomatik kaydettiği veya elle yapılan sorgudaki hata kayıtları bu işlem ile görüntülenir. Hatalar görüntüledikten sonra yine o veriye ait olan dosyanın analizi buradan da yapılabilir.

DOSYANIN BULUNMASI VE ANALİZ

File Search Modülü

Programı makine ile paralel çalıştırdığınızda çağırılan modüldür. Güncel olan tarihe ait klasöre ilerler , o klasördeki veri sayısını hesaplar ve yeni verinin gelmesini bekler. Yeni veri tespit edilince veri adını, tarihini çıkış olarak verir.

Kullanımı :

1. Modül içerisinde bulunan dosyabul isimli class bir değişkene atanılarak çağırılır

```
import filesearch  
file = filesearch.dosyabul()
```

2. Daha sonra güncel tarihe ait dosyayı bulması için findate isimli fonksiyonu çağırılır.

```
import filesearch  
file = filesearch.dosyabul()  
  
file.findate()
```

3. Tarama yapılan tarihe ait dosya bulunamazsa program kendisi oluşturur. Taramaya başlaması için detect_new isimli fonksiyon çağırılır

```
import filesearch  
file = filesearch.dosyabul()  
  
file.findate()  
  
file.detect_new()  
|
```

4. Detect_new isimli fonksiyonda program sonsuz while döngüsüne girer. Bu döngüyü kırmak için 2 koşul eklenmiştir : Yeni dosyanın bulunmuş olması veya yeni bir güne geçilmiş olmasıdır. Yeni dosya bulunduğunda program bu fonksiyondan çıkacak ve dosya ismi tarihi gibi çıktıları oluşturmuş olacaktır. File.data_name dosyanın tam adını , file.filedate dosyanın tarihini , file.data_path dosyanın bulunduğu klasörün tam konumunu içerir. Elde edilen bu bilgileri daha sonra analiz edebilmek için veri analiz modülüne göndereceğiz.

“.csv” Uzantılı Dosyalar Nasıl Tespit Edilir ?

Bir klasörde bulunan tüm dosyaların isimlerini içeri aktarabilmek için os modülündeki “listdir” komutu kullanılır. Bu komut bir liste içeriği döndermektedir. Dolayısıyla listenin her elemanına erişebilmek için bir for döngüsü kullanılabilir. Python algoritmasında for ve in komutları kullanılarak kolaylıkla liste elemanlarına ulaşılır.

for dosya_adi in os.listdir():

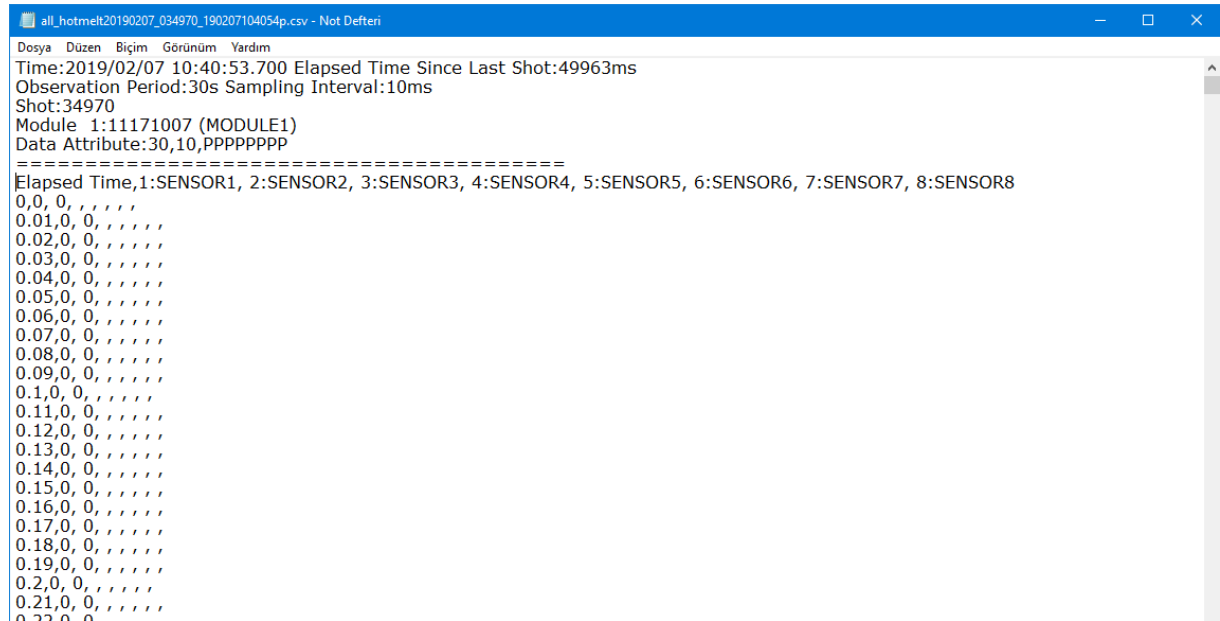
Komut satırı ile dosya_adi isimli değişken list.dir komutundan dönen tüm elemanların içeriğini sırasıyla edinir. Bu komuttan sonra içeri aktarılan dosya ismi ile ilgili komutlar gelmelidir. Edinilen dosyanın .csv uzantılı olup olmadığı aşağıdaki yöntemle kolaylıkla anlaşılır.

```
for dosya_adi in os.listdir():  
    if str(dosya_adi)[-4:] == “.csv”:  
        print(“CSV Dosyası bulundu !”)
```

Dosya_adi isimli değişken str() fonksiyonu ile string değişkenine dönüşmüştür. Bunun sebebi dosya isminin her karakterine erişebilmek ve başka bir string değişkeni ile işleme sokabilmektir. Dosya adının son 4 elemanını seçmek için [-4:] yazılır. Bu son 4. Karakter ve kalanına bak anlamına gelmektedir. Son 4 eleman .csv ise csv dosyası bulunmuş anlamına gelir.

“.csv” Dosyası İçindeki Veriler Nasıl Ayırılır ?

Csv uzantılı dosyaların içeriği verileri virgül ile ayırarak tablo görünümünü oluşturma biçimindedir. Virgülle ayrılma mantığı sayesinde tüm ofis programları tarafından ortak olarak açılabilir. Hotmelt’den gelen csv dosyasının içeriği aşağıdaki gibidir :



```
all_hotmelt20190207_034970_190207104054p.csv - Not Defteri  
Dosya Düzen Biçim Görünüm Yardım  
Time:2019/02/07 10:40:53.700 Elapsed Time Since Last Shot:49963ms  
Observation Period:30s Sampling Interval:10ms  
Shot:34970  
Module 1:11171007 (MODULE1)  
Data Attribute:30,10,PPPPPPPP  
=====Elapsed Time,1:SENSOR1, 2:SENSOR2, 3:SENSOR3, 4:SENSOR4, 5:SENSOR5, 6:SENSOR6, 7:SENSOR7, 8:SENSOR8  
0,0,0,0,0,0,0,0  
0.01,0,0,0,0,0,0,0  
0.02,0,0,0,0,0,0,0  
0.03,0,0,0,0,0,0,0  
0.04,0,0,0,0,0,0,0  
0.05,0,0,0,0,0,0,0  
0.06,0,0,0,0,0,0,0  
0.07,0,0,0,0,0,0,0  
0.08,0,0,0,0,0,0,0  
0.09,0,0,0,0,0,0,0  
0.1,0,0,0,0,0,0,0  
0.11,0,0,0,0,0,0,0  
0.12,0,0,0,0,0,0,0  
0.13,0,0,0,0,0,0,0  
0.14,0,0,0,0,0,0,0  
0.15,0,0,0,0,0,0,0  
0.16,0,0,0,0,0,0,0  
0.17,0,0,0,0,0,0,0  
0.18,0,0,0,0,0,0,0  
0.19,0,0,0,0,0,0,0  
0.2,0,0,0,0,0,0,0  
0.21,0,0,0,0,0,0,0  
0.22,0,0,0,0,0,0,0
```

Python'daki csv modülü bu tür dosyaları kolaylıkla okuyabilmemizi sağlar. Csv modülü ile bir dosya aktardığımızda her satır bir dizi içerisinde aktarılır ve virgüller sütunları ayırmaya yarar.

```
#-----DOSYA AÇILIR-----  
if self.aramaflag == True:  
    time.sleep(settings.wait_for_data)  
    with open(isim,newline='') as file:  
        spm = csv.reader(file,delimiter=',',quotechar = '|')  
        for self.r in spm: # DATA İÇERİSİNDEKİ HER BİR VERİYİ DENETLER  
            self.veriyayir()  
            self.vericount+=1
```

Bu kod satırında “aramaflag” isimli değişken, analiz modülünün “makine ile paralel çalışma” fonksiyonu ile mi yoksa “manuel sorgu fonksiyonu” ile mi çağırıldığını belirtir. Bu ikisi arasındaki fark şudur : Makine ile paralel çalışma durumunda Hotmelt'in yeni bir csv dosyası oluşturması beklenir. Program sürekli bir döngüde yeni dosya bekler ve yeni dosya oluşturulduğu anda onu çekmeye çalışır. Ancak Windows henüz bu dosyayı oluşturma aşamasında olduğundan , programın erişmesine izin vermeyecektir. Bunun için programı bir süre bekletmek tek çözüm yolu olur. Ayrıca Hotmelt'in 30 saniyelik çalışma süresi boyunca csv dosyasına anlık olarak veriler eklenir. Bu sebeple dosyaya erişmek için minimum 30 saniye beklenmelidir. Ancak zaten var olan bir dosyaya erişmek için beklemenize gerek yok yoktur. Bu sebeple böyle bir değişken oluşturulmuştur.

With open komutu ile dosya “file” isimli değişkene atanarak açılır. With komutu dosya ile ilgili komutlar bittiğinde dosyanın otomatik olarak kapanmasını sağlar. Dosya açıldıktan sonra csv.reader fonksiyonu ile içindeki veriler import edilir ve spm isimli liste değişkenine aktarılmış olur. Liste elementlerine for döngüsü kullanarak erişiyorduk. Burada da spm elemanlarını sırasıyla self.r isimli değişkene yazdırıyoruz. Her elemanı veriyayir isimli kendi fonksiyonumuza göndererek ayırıyoruz ve veri sayısını tutan “self.vericount” değişkenini arttırıyoruz.

Veriyayir() Fonksiyonu

```
def veriyayir(self,):  
    self.csv_time = self.r[0].split(',') # SURE VERİSİNİ AYIRIR  
    if self.vericount>6: #ilk 6 veri info satırlarıdır  
        self.csv_sensor2 = self.r[1].split(',') #SENSOR 2 'nin verilerini ayırır  
        self.surelist.append(str(self.csv_time[0]))  
        self.sensor1.append(str(self.csv_time[1]))  
        self.sensor2.append(str(self.csv_sensor2[0])) #VERİLERİ AYIRIP LİSTELERE KAYDEDER
```

Dosyamızdan aktarılan liste elementlerinin yazım mantığını anlamak için print ettik ve ayırma yöntemimizi oluşturduk. Her satırdaki virgüle kadarki ilk veri süre bilgisini içermekteydi. Bu sebeple self.r olan satır verisinin ilk elemanını virgül ile ayırarak self.csv_time isimli değişkene atadık. Virgülden sonraki elemanı sensör 1 verisini içermektedir. Daha sonra ilk 6 satır bilgi satırı olduğundan buraları atlادık ve verilerimizi 3 ayrı listeye ayırmak için append (bir listeye eleman ekleme) komutunu kullandık.

Bu işlemlerin sonucunda zaman ve sensör verileri ayrı ayrı ancak aynı indislere sahip olarak elimizde oldu. Bu listelerden istediğimiz elemanlara erişebilir ve artık onları matematiksel işlemlere sokabiliriz.

VERİ ANALİZ MODÜLÜ

Csv dosyalarını import ederek içerisindeki verileri ayıran ve analizini yapan modüldür.

Yaptığı işlemler :

- Csv dosyasını import edip liste değişkenlerine atamak.
- Sensör verilerini tek tek inceleyerek veri değerlerinde istenmeyen veriler olup olmadığını kontrol etmek.
- Sensör verilerindeki yükselmeyi tespit etmek için matematiksel işlem yaparak peak (zirve) durumlarını tespit etmek.
- Enjeksiyon makinesi için geliştirilern , enjeksiyon başlama ve bitme anlarını tespit ederek enjeksiyon süresini hesaplamak.
- Tespit ettiği hataları bir klasöre raporlamak.

Analiz Class'ı

```
class analiz():  
    def __init__(self, isim, yol=None, aramaflag =None, noprint = None, date = None):
```

Yukarıda görüldüğü gibi analiz classını çağırmak için bazı veriler girilmelidir. Not: Class çağırmak için bir değişken modül adına eşitlenir. Örneğin : tarama = verianaliz.analiz(). Bu sayede class'ta self. Yazan herşey tarama ismine dönüşür. Yani class'a ait değişkenler tarama değişkenine aktarılır.) None yazan değişkenler girilmek zorunda değildir. Bu class için öncelikle csv dosyasının adı ve bulunduğu klasör konumu bilinmelidir. İsim değişkenine dosyanın adı, yol değişkenine ise dosya konumu atanır. Eğer otomatik tarama olarak çalıştırılacak ise "aramaflag == True" atanmalıdır. Terminalde hiçbir veri görülmemesi isteniyorsa noprint değişkeni True atanmalıdır. Hata Kayıt için gerekli tarih bilgisi date değişkenine verilir.

Analiz Class'ı bir kez çalıştırıldığında tüm işlemler otomatik olarak gerçekleşir. Bir sonraki adım bu işlemlerin çıktılarını toplamaktır. Analiz Class'ının çıkışları :

- **self.info** : Analiz sonucundaki bilgileri anlaşılır şekilde saklayan değişkendir. Analizden sonra bu değişken çağırılarak sonuçlar görüntülenebilir.
- **self.peaklist1 veya 2** : Sensör verilerine ait tepe değerlerin saklandığı liste değişkeni. Algılanan peakları görüntülemek için kullanılabilir.
- **self.error_info** : Hatalara ait bilgi değişkenidir.
- **self.plot()** : Verileri ve peakleri grafiğe dökmek isteniyorsa class'ın atandığı değişken adı örneğin tarama ise, tarama.plot() komutu kullanılarak grafik görüntülenir.

Analiz Classındaki İşlemler

Veri Ayır isimli fonksiyon çağrıldıktan sonra denetle fonksiyonu çağrılır ve verilerin analizi başlar.

Denetle Fonksiyonu

```
#-----VERİLERİ İNCELE-----  
def denetle(self,):  
    count1 = 0 # VERİ İÇERİSİNDE ALINACAK ÖRNEKLER İÇİN SAYICI VE BOŞ DİZİ TANIMLANIR  
    count2 = 0  
    sample1 = []  
    sample2 = []  
    for i in range (0,len(self.surelist)): # Zamanı teker teker arttırır.  
        self.acibull(i,self.gen1)  
        self.acibul2(i,self.gen2) # açı hesaplama için zaman bilgisi girilir.  
        self.peakdetect(i) # Hesaplanan açılar üzerinden peak denetlenir.
```

Sensör verilerinin tutulduğu liste ve süre listesinin indislerinin aynı olduğunu söylemiştik. Bu da elemanları süreyi referans alarak senkronize edebiliriz demek oluyor. Yine for döngüsü ile veri süresi boyunca sensör verileri sırayla çağırılır. Burada eleman sayısı len komutu ile öğrenilir.

```
for i in range (0,len(self.surelist)):
```

Bu satırın anlamı; 0 dan başla ve sürelist listesinin eleman sayısına ulaşana kadar i değişkenini 1 arttır demektir.

Böylece sensör verilerini sırayla açıbul isimli fonksiyona aktarırız.

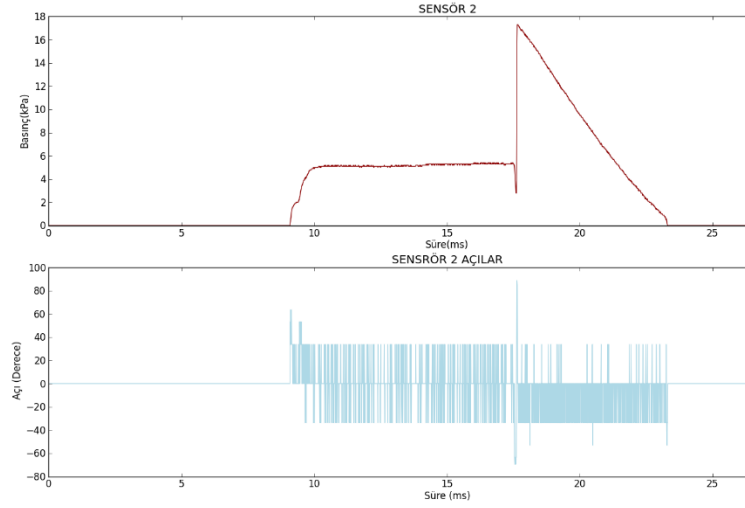
Açı Bul fonksiyonu

Acibull ve acibul2 olarak iki farklı sensör için ayrılmıştır. Bu fonksiyonun kullanım amacı ardışık veriler arasındaki yükselme değerini tanjantlarını alarak hesaplamaktır.

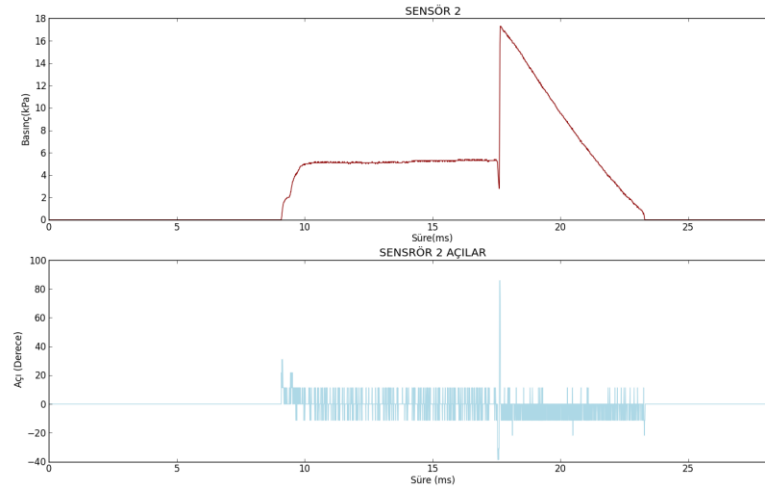
Bunun için aşağıdaki işlem yapılır :

```
#-----VERİLERİN AÇILARINI BULMAK-----  
def acibull(self,veri,gen):  
    if veri < len(self.surelist)-1:  
        self.y = float(self.sensor1[veri+1]) - float(self.sensor1[veri])  
        self.x = gen*float(self.surelist[veri+1]) - gen*float(self.surelist[veri])  
        self.tan = self.y / self.x  
        self.angle = math.degrees(math.atan(self.tan))  
        self.anglelist1.append(self.angle)
```

Acibul fonksiyonuna girilmesi gereken veri isimli değişken sensör verisinin indisidir. Self.y değişkeni iki boyutta y eksenini temsil eder ve bu sensör verisidir. Self.x ise yatay ekseninde süreyi temsil eder. Burada ilk satıra konulan if komutunun nedeni son veriye gelindiğinde açı bulma işleminin yapılmasını engellemektir. Liste indisi 0 dan başladığından son veri indisi toplam eleman sayısının 1 eksiklidir. Burada dikkat çeken nokta gen değişkeninin neden konulduğudur. Sensör değerleri 1 ile 20 arasında değişen değerlerdir. Ancak süre Hotmelt'teki ayara göre 0.2 veya 0.02 gibi küçük farklarla artarak gider. Yani Sensör verisindeki değişimler zaman verisinden çok daha büyüktür. Bu sebeple ark tanjant değeri en ufak değişiklikte yüksek değerlere ulaşacak ve hassasiyetin bozulmasına yol açacak. Bunun önüne geçmek için belirli bir genlik değeri ile süre verisi genişletilir.



Şekil 1 Genlik Değeri 15 iken ortaya çıkan grafikler.

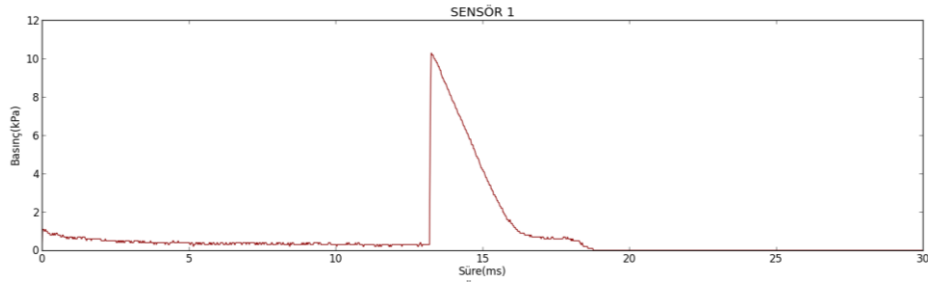


Şekil 2 Genlik Değeri 50 iken ortaya çıkan grafikler.

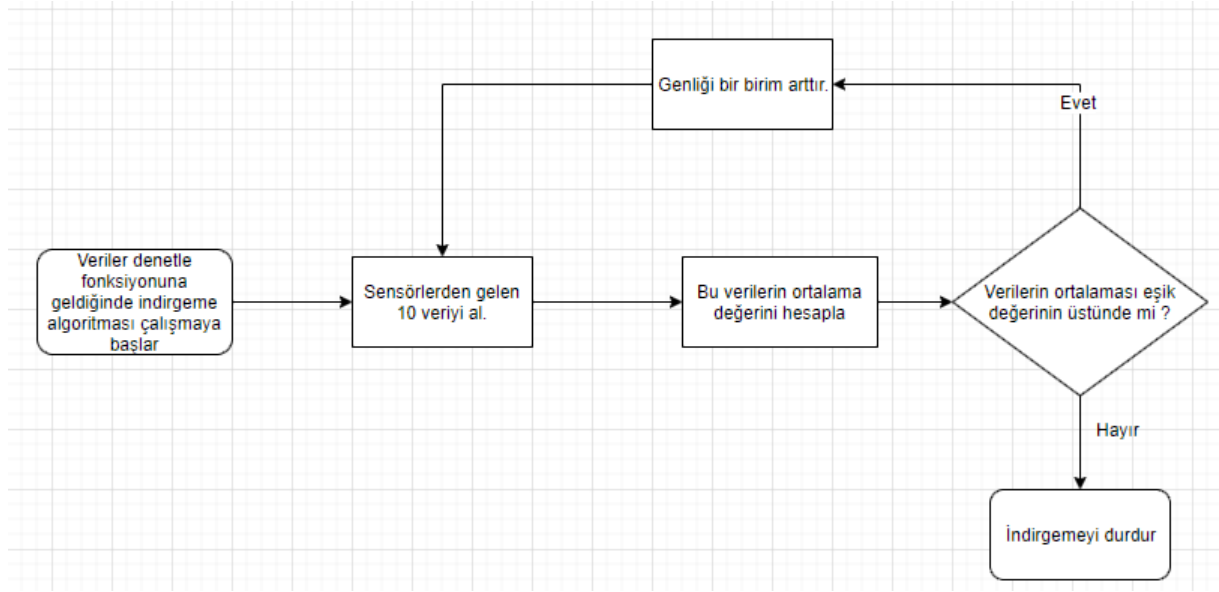
Genlik Değerinin Ayarlanması : Genlik değeri, sürenin artış hassasiyetine göre değiştirilmelidir.

- Çok yüksek girilen (50 den yüksek) genlik değeri 0.2 gibi artışlarda olan veriler için tanjantların düşük seviyede kalmasına ve ancak çok yüksek değişimlerin algılanabilmesine neden olur.
- Az girilen genlik değeri ise (30 dan daha az) en ufak bir değişimde dahi yüksek tanjant değerlerinin elde edilmesine sebep olur.

Genlik değeri görüldüğü gibi tanjant değerini doğrudan etkiler. Ancak bu değer ayarlanırken yalnızca sürenin artış miktarı göz önüne alınmaz. Sensörlerden gelen veriler bazen fazla titreşimli olabilir. Bu durumda şöyle bir grafik çıkar :



Veri değeri 0 iken en ufak değişiklikler dahi tanjant açısının çok yüksek değerlere çıkmasına neden olur. Bu yüzden titreşimli verilerde genlik değeri değiştirilerek açı değerlerinin istenilen eşikte tutulması sağlanabilir. Ancak bu da her verinin yine insan tarafından kontrol edilip genlik değerinin değiştirilmesi anlamına gelecektir. Bundan kaçınmak için bir veri indirgeme algoritması geliştirildi.



```

def denetle(self,):
    count1 = 0 #
    count2 = 0
    sample1 = []
    sample2 = []
  
```

Denetle fonksiyonunu tanıtırken bu 4 değişkeni anlatmamıştık. Sensörlerden gelen verileri örneklem olarak kabul edeceğimiz değişkenlere atayacağız. Bunun için kaç veri geldiğini tutan bir integer count değişkeni ve verileri depolayacağımız bir liste değişkeni tanımladık. İlk adım olarak 10 veriyi nasıl depolayacağımızı inceleyelim.

```

for i in range (0,len(self.surelist)): # Zamanı teker teker arttırır.
    self.acibull(i,self.gen1)
    self.acibul2(i,self.gen2) # açı hesaplama için zaman bilgisi girilir.
    self.peakdetect(i) # Hesaplanan açılar üzerinden peak denetlenir.

    if settings.sample_range_start < i < settings.sample_range_stop: # 0.06 ve 2. saniye arasındaki değerler örnek olarak alınır.
        if count1<10: # ilk 10 veri hesaplanır.
            if int(self.anglelist1[-1]) != 0: # 0 değerlerini katma
                sample1.append(abs(self.anglelist1[-1]))
                count1+=1
            if count2<10:
                if int(self.anglelist2[-1]) != 0:
                    sample2.append(abs(self.anglelist2[-1]))
                    count2+=1
  
```

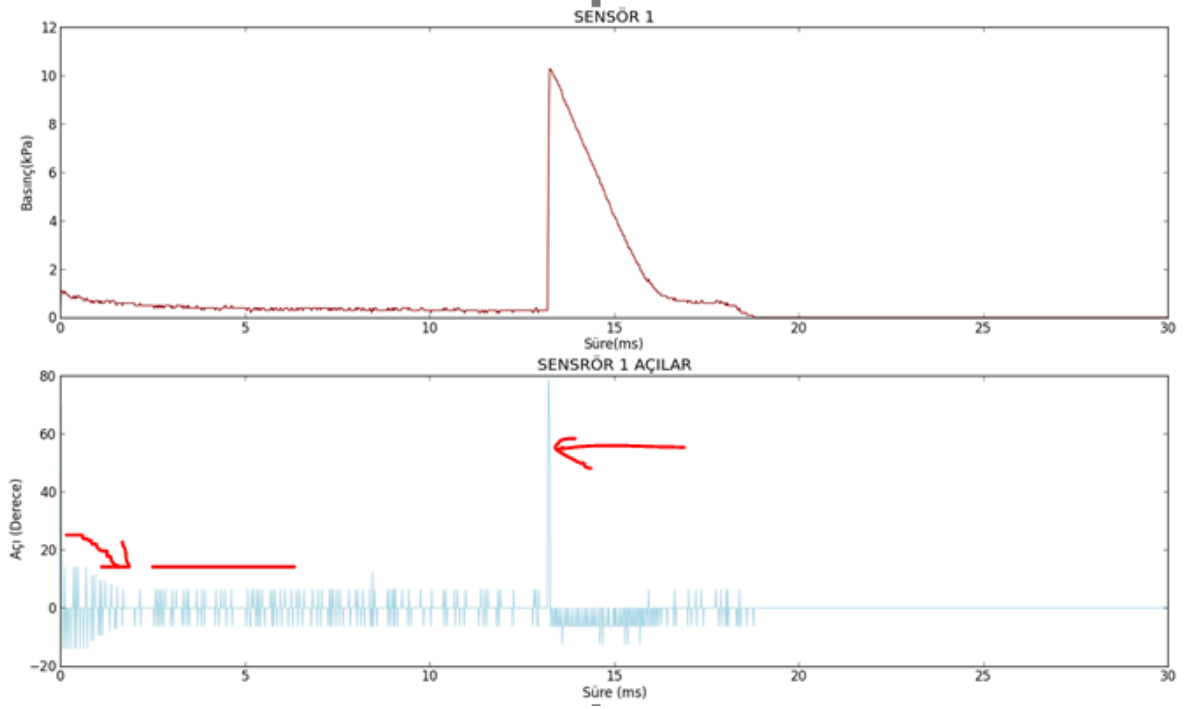
For döngüsü ile her veriye ait açı bilgilerini bulduğumuzu anlatmıştık. For döngüsündeki “i” değişkeni o andaki süreyi temsil eder . settings.sample_range start ve stop değişkenleri programa hangi zaman aralığında indirgeme yapacağını belirtir (settings modülünden bu ayarlar değiştirilebilir). Count1 < 10 koşulu ile 10 kez veri alma işlemini yapmasını söyledik. Ayrıca 0 değerlerini örnekleme katmayacağımızdan (Bizim için önemli olan değişim büyüklüğü. 0 derece değişim yok demektir) != 0 komutu ile 0 a eşit değilken örnek al komutunu verdik. Self.anglelist 1 veya 2 hesaplanan açı değerlerini tutan listedir. [-1] yani son elemanına göre işlem yapmaktayız.

```

try :
    if count1 == 10 and settings.min_angle < self.mean(sample1) < settings.max_angle:#Settings dosyasından alınan veriye göre açı değerini
        if int(self.anglelist1[-1]) >=settings.min_angle:
            self.gen1 += 5 #çekebilmek için genliği arttırır.
            count1 = 0
            #print("1.Genlik 5 arttı ! ")
    if count2 == 10 and settings.min_angle < self.mean(sample2) < settings.max_angle:
        if int(self.anglelist2[-1]) >=settings.min_angle:
            self.gen2 += 5
            count2 = 0
            #print("2.Genlik 5 arttı ! ")
  
```

Örnekler başarıyla alındıktan sonra sıra ortalamalarını almaya gelir. Buradaki if komutuna eklediğimiz koşulları açıklayalım :

Count == 10 yani 10 örnek alındığında, ayarlar dosyasından içeri aktarılan min_angle ve max_angle değerleri eşik olarak kabul edilerek self.mean() fonksiyonundan gelen ortalama değer uygunsa aşağıdaki komutları uygula. Burada yine bir koşul ekledik. Bunun sebebi doğrultma yapılırken alınan örneklemin sonlarına doğru doğrultma başarılı olmuş olabilir. Bundan dolayı son elemanın yine hatalı olup olmadığı kontrol edilir. İndirgeme işlemi sonucunda ortaya çıkan veri :



İndirgeme işleminden sonra açılar hesaplanır ve peak olup olmadığını kontrol eden peakdetect fonksiyonuna gönderilir.

Peak Detect Fonksiyonu

```
#-----PEAK BULMA-----  
def peakdetect(self,i):  
    if len(self.peaklist1) < 1 and float(self.anglelist1[-1]) > settings.first_peak_value and float(self.sensor1[i+10]) > 2:  
        self.peak1()  
    elif len(self.peaklist1) >= 1 and float(self.anglelist1[-1]) > settings.last_peak_value :  
        self.peak1()
```

İncelenen sisteme göre peak bulma koşulları değişkenlik gösterebilir. Peak bu programda tanjant açısının belirli bir değerin üstünde çıkması demektir. Sisteminizde peakleri hesaplamak için sabit bir değer olacaksa yalnızca settings.peak_treshold isimli değişkeni kullanıp açı listesinin son elemanının bu değerden büyük olup olmadığını if komutu ile karşılaştırabilirsiniz. Ancak bizim sistemimizde bazı özel durumlar olduğu için farklı koşullar ekledik.

İlk peki hesaplamak için öncelikle peaklistin hiçbir elemanı olmaması gerekir bunun için len(self.peaklist) < 1. Açı değerinin belirlenen değerin üstüne olması gerekir. Self.anglelist1[-1] yani son açı elemanı first_peak_value den büyükse. Sistemde düşük basınç değerindeki değişimlerin peak olarak algılanmasını istemediğimizden bir basınç eşiği koyabiliriz. Örneğin

Basınç değerinin 1 iken 1.5 olması %50 artışken, 14 ten 14.5 olması %3.5 artış anlamına gelir. Bu sebeple belirli bir değerin aşağısını yok sayabiliriz.

```
def peakl(self,):  
    self.peakno1 = len(self.anglelist1)-1  
    self.reftime1 = float(str(self.surelist[self.peakno1]))  
  
    if len(self.peaklist1) >= 1 and self.reftime1 - float(self.peaklist1[-1]) >= settings.wait_for_next_peak :  
        self.peaklist1.append(self.reftime1)  
        self.peakcount1 += 1  
    elif len(self.peaklist1) < 1 :  
        self.peaklist1.append(self.reftime1)  
        self.peakcount1 += 1
```

Peak algılandığında bunu kaydetmek için self.peak fonksiyonu çağrılır. Peakno isimli değişken hangi zamanda peak olduğunu belirtir. (Açılış listesinin genliği süre indisi ile aynıdır). Bunu reftime isimli değişkene atayarak peakin bulunduğu zamanı referans olarak alıp diğer peakler ile karşılaştırabiliriz. Burada wait_for_next_peak isimli ayar değişkeninin amacı şudur : Basınç artışı bir anda değil belirli bir süre boyunca olmuş olabilir. Tanjant değeri aynı artış için birden fazla kez büyük olacağından program daha fazla peak algılayacaktır. Bunun önüne geçmek için bir kez peak algılandığında bir süre beklemesini isteriz. Ancak ilk peak için böyle bir koşul olamayacağından boyutu 1 den küçükse onu koşulsuz olarak peak algıla komutu verdik.

Peakler algılandıktan sonra artık eldeki verileri analiz etme zamanı geldi. Öncelikle ilk ve son peak arasını hesaplayarak enjeksiyon süresini buluruz :

```
self.enjeksiyon_suresi1 = str(self.peaklist1[-1]-self.peaklist1[0])
```

Daha sonra basınç hatalarının olup olmadığına bakılır.

```
for i in range (0,len(self.surelist)):  
    if int(float(self.surelist[i])) < self.peaklist1[-1]*1000:  
        self.basinchatadetec(i)
```

Son peak sırasında enjeksiyon sonlanacağı ve bu sırada basıncın bir süre doğal olarak artacağından dolayı bu değerden önceki sürelerde basınç hataları aranır. Peaklistteki değer saniye , surelist ise ms olduğunda 1000 ile çarpılır.

Hata Arama Fonksiyonları

Basınç Hatası Bulmak

```
def basinchatatadetect(self,i):
    if float(self.sensor1[i]) > settings.max_pressure or float(self.sensor2[i]) > settings.max_pressure:
        self.error_info += "\n\nHata Türü : Sensör Basınç Hatası\n" + "Süre : " + str(self.surelist[i]) + \
            "\nSensör 1 Basınç Değeri (kPa): " + str(self.sensor1[i]) + \
            "\nSensör 2 Basınç Değeri (kPa): " + str(self.sensor2[i])
        self.errorflag = True
    return
```

Eğer sensörün o anki değeri ayarlarda belirtilen değerden büyükse hata error_info ya kaydedilir ve hata olduğunu bildiren errorflag e True atanır.

Enjeksiyon Süre Hatası Bulmak

```
def zaman_hatasi(self,):
    if float(self.peaklist1[-1]-self.peaklist1[0]) < settings.min_enj_time or float(self.peaklist2[-1]-self.peaklist2[0]) < settings.min_enj_time):
        self.error_info += "\n\nHata Türü : Enjeksiyon Süre Hatası"
        if self.noprint != True:
            print(self.error_info)
            print("*** * 50)
        self.errorflag = True
```

Peak Sayısı Hatası Bulmak

```
def peak_sayi_hatasi(self,):
    if self.peakcount1 < settings.default_peak_count or self.peakcount2 < settings.default_peak_count:
        if self.noprint != True:
            print("\nHATA ! Sensörlerin birinde Peak az, Hata bulunmuş olabilir.")
        if self.peakhata != True:
            self.error_info += "\n Hata Tipi : Sensör Peak Sayısı Hatası\n"
            self.error_info += "\nTespit edilen Sensör 1 Peakler : \"
                +str(self.peaklist1) + "\nTespit edilen Sensör 2 Peakler : \" +str(self.peaklist2)
            self.errorflag = True
```

Hata Kayıt

```
#-----ANALIZ SONRASI SON İŞLEMLER-----
if self.aramaflag == True and self.errorflag == True: # ARAMA YAPILIYORSA VE HATA OLUŞMUŞSA HATAYI KAYDET
    self.hatakayit(date)
```

Self.hatakayit fonksiyonu :

```
def hatakayit(self,date):
    os.chdir(program_files.auto_errors)
    try:
        os.mkdir(date)
    except:
        pass
    finally:
        os.chdir(date)
    self.error_filename = "20" + self.isim[-17:-15] + "-" + self.isim[-15:-13] + "-" + self.isim[-13:-11] + "_" + self.isim[-11:-9] + "-" + self.isim[-9:-7] + ".txt"
    self.errorlog = open(self.error_filename,'a+')
    self.errorlog.write("\nOriginal dosya adı :"+self.isim+"\n")
    self.errorlog.write(self.info)
    self.errorlog.write(self.error_info)
    self.errorlog.write('\n' + '*'*50 + '\n')
    self.errorlog.close()
```

Errors klasöründeki Auto klasörüne girilir ve hata bulunan tarih adına ait klasör oluşturulur eğer böyle bir klasör varsa hata alınacaktır. Bu sebeple try except kullanılmıştır ancak hata alsada almasada bugüne ait olan klasöre girmesi için finally : os.chdir kullanılır.

Klasöre girildiğinde dosya adı csv uzantılı dosya adından son 17 hanesi tarih olduğundan parçalara ayrılarak Yıl-Ay-Gün_Saat şeklinde hata dosyası .txt uzantısı ile oluşturulur ve hata bilgileri içine yazılıp kaydedilir.

Bu işlemden sonra Veri Analiz modülünün otomatik işlemleri tamamlanır.

Verilere Ait Grafik Oluşturma

Self.plot komutu ile grafik oluşturabildiğimizden yukarıda bahsetmiştik.

```
-----GRAFIK ÇİZDİRME-----
def plot (self,):
    def draw(name,ccolor,title,xlabel,ylabel,axisx,axisy):
        name.plot(axisx,axisy,color = ccolor)
        name.set_title(title)
        name.set_xlabel(xlabel)
        name.set_ylabel(ylabel)
    fig,(sub1,sub3) = plt.subplots(2)
    fig,(sub2,sub4) = plt.subplots(2)
    draw(sub1,"darkred","SENSÖR 1","Süre (s)","Basınç (MPa)",self.surelist,self.sensor1)
    draw(sub2,'darkred',"SENSÖR 2","Süre (s)","Basınç (MPa)",self.surelist,self.sensor2)
    draw(sub3,'lightblue',"SENSÖR 1 AÇILAR", "Süre (s)"," Açı (Derece)",self.surelist,self.anglelist1)
    draw(sub4,'lightblue',"SENSÖR 2 AÇILAR", "Süre (s)"," Açı (Derece)",self.surelist,self.anglelist2)
    plt.show()
```

Matplotlib kütüphanesinden pyplot isimli modül plt olarak içeri aktarıldı ve draw fonksiyonu oluşturularak grafik çizimi için taslak yapıldı.

SETTINGS MODÜLÜ

Programdaki kullanıcının değiştirebileceği , analizi etkileyen parametreleri içeren , ayrıca programdaki dosyaların konumunu içeri aktaran modüldür.

```
import os
class settings():
    def __init__(self,):
        self.first_peak_value = 10          # İLK PEAK İÇİN MİN DEĞER ( Default = 14 degrees)
        self.last_peak_value = 20           # SON PEAK İÇİN MİN DEĞER ( Default = 20 degrees)
        self.min_enj_time = 5               # ENJEKSİYON SÜRESİ ( Default = 5 s)
        self.max_pressure = 20              # MAX BASINÇ ( Default = 20 MPa)
        self.wait_for_data = 1              #Verinin oluşması için bekleme süresi (Default = 38 sec)
        self.min_angle = 10                 # ( Default = 10 degrees)
        self.max_angle = 20                 # Doğrultulacak olan veri için aralık belirtilir. ( Default = 20)
        self.amp = 20                       # 3000 ve aşağıı sayıdaki veriler için genlik değeri ( Default = 20)

#-----EK AYARLAR-----
        self.wait_for_next_peak = 0.5       # Peak bulduktan sonra bekleme süresi (default = 1)
        self.amp1 = 100                     # 3000 den fazla veri sayısı için genlik değeri ( Default = 100)
        self.jump = 20                      # düzensiz veriler için atlanacak süre miktarı
                                           # ( Default = 20 *t. t, birim zamandır. veri sayısına göre değişir.)
        self.sample_range_start = 3         #Doğrultma için alınan örneğin başlama zamanı (30sn/veri sayısı)* range_start (Default : 1500 veri için 30. -0.06 . saniye-)
        self.sample_range_stop = 101        # Doğrultma bitiş zamanı ( Default : 1500 veri için 101 -2. saniye-)
        self.default_path = os.getcwd()     #Programın varsayılan çalışma dosyası.
        self.default_peak_count = 2         # Minimum peak sayısı
        self.process_time = 30              # Proses Süresi
```

Yazılım sürekli olarak dosyalar arasında dolaştığından klasör içerisinde kaybolup hataya düşmemesi için program klasöründe gezilen tüm konumlar program_files isimli fonksiyon ile kaydedilir.

```
class program_files:
    def __init__(self,):
        self.main = os.getcwd()
        os.chdir("Datas")
        self.datas = os.getcwd()
        os.chdir("../")
        os.chdir("Errors")
        self.errors = os.getcwd()
        os.chdir("Auto")
        self.auto_errors = os.getcwd()
        os.chdir("../")
        os.chdir("Manual")
        self.manu_errors = os.getcwd()
        os.chdir(self.main)
        os.chdir("bin/Errors")
        self.perrors = os.getcwd()
        os.chdir(self.main)
```

Ayarların kurulumu için aşağıdaki komutlar uygulanmalıdır :

```
from settings import settings, program_files
settings = settings()
program_files()
```

KENDİ ANALİZ SCRIPT'İNİZİ OLUŞTURUN

Kendi analiz scriptinizi oluşturmak için import edilmesi gereken modüller ve ilk komutlar şu şekilde olmalıdır :

```
import time
import filesearch
import os
import sys
import verianaliz
from settings import settings, program_files
from datetime import datetime # tarih ile ilgili fonksiyonlar için.
#İHTİYACA GÖRE KULLANILABİLECEK MODÜLLER :
from matplotlib import pyplot # Grafik çizebilmek için
import threading # Birden fazla döngü oluşturabilmek için
import tkinter # programı bir gui üzerinde kullanabilmek için.
# Ayarların kurulumu
settings = settings()
program_files()
#Yeni dosya otomatik bulunacaksa :
file_auto = filesearch.dosyabul()

file_auto.finddate()

file_auto.detect_new()

#El ile tarama yapılacaksa :
file_manual = verianaliz.sorgu() # bu komut sonrası dosya adı ve tarihi
                                # verino isimli değişkene aktarılmış olur.

#Bulunan dosyayı veri analiz modülüne gönderin : dosya_adi ve konumu file_auto veya
                                # file_manual e ait çıktıdır.

analiz = verianaliz.analiz( dosya_adi, dosya_konum )
#Analiz sonuçlarını ekrana yazdırın:
print ( analiz.info)
print ( analiz.error_info )
```