



BCO 607 Hareket Analizi Sistemleri

OpenCV ile Görüntü İşleme 3



SERDAR ARITAN

serdar.aritan@hacettepe.edu.tr

Biyomekanik Araştırma Grubu
www.biomech.hacettepe.edu.tr
Spor Bilimleri Fakültesi
www.sbt.hacettepe.edu.tr
Hacettepe Üniversitesi, Ankara, Türkiye
www.hacettepe.edu.tr



Yansıtıcı işaret etiketleme

6 işaret



Yansıtıcı işaret etiketleme

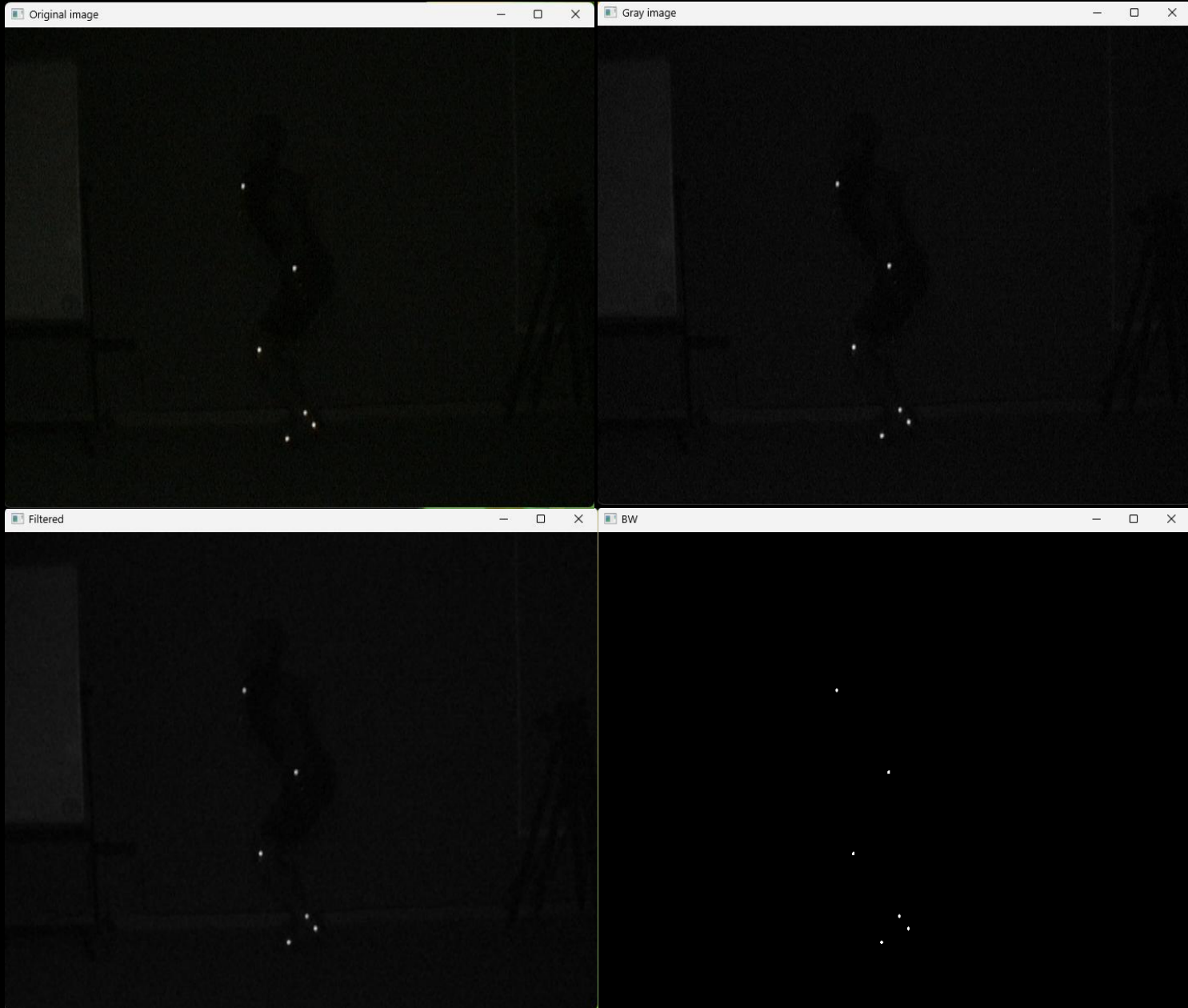
```
import cv2 as cv

# Read the image
BGR = cv.imread('jump_106.jpg')
# cv.imshow('Original image',BGR)
# cv.waitKey(0)
# Convert to the gray
imgray = cv.cvtColor(BGR, cv.COLOR_BGR2GRAY)
# cv.imshow('Gray image', imgray)
# cv.waitKey(0)
# median filter salt&pepper
filtered = cv.medianBlur(imgray, 3)
# cv.imshow('Filtered', filtered)
# cv.waitKey(0)
# Convert to the Black & White binary image
_,bw = cv.threshold(filtered, 127, 255, cv.THRESH_BINARY)
# cv.imshow('BW', bw)
# cv.waitKey(0)
cv.destroyAllWindows()
```



Yansıtıcı işaret etiketleme

Name ▲	Type	Size	Value
BGR	Array of uint8	(576, 720, 3)	[[[6 8 8] [4 6 6] [0 0 0] ... 0 0 0]
bw	Array of uint8	(576, 720)	[[8 6 6 ... 8 8 5] [8 6 6 ... 7 6 5] [11 8 6 ... 7 8 5]
filtered	Array of uint8	(576, 720)	[[8 6 6 ... 8 13 5] [11 8 6 ... 7 8 5]
imgray	Array of uint8	(576, 720)	



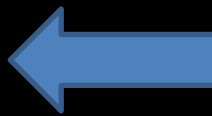
Yansıtıcı işaret etiketleme

```
totalLabels, label_ids, values, centroid = \
    cv.connectedComponentsWithStats(bw)

# put text and highlight the center
for i in range(len(centroid)):

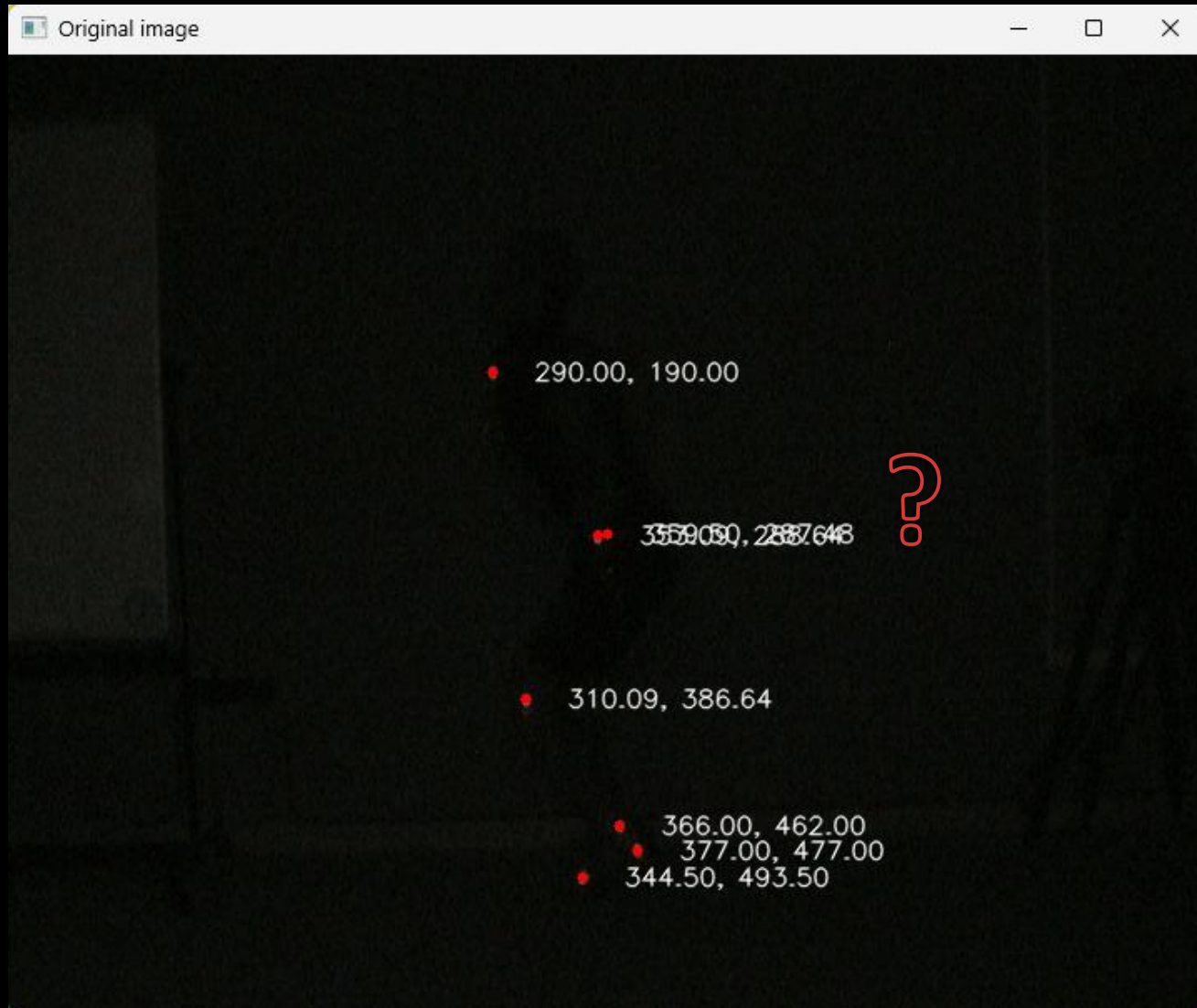
    cv.circle(BGR, (int(centroid[i][0]), \
                      int(centroid[i][1])), 3, (0,0,255), -1)
    cv.putText(BGR, "{:.2f}, {:.2f}".format(centroid[i][0], \
                      centroid[i][1]), \
               (int(centroid[i][0]) + 25, int(centroid[i][1]) + 5), \
               cv.FONT_HERSHEY_SIMPLEX, 0.5, \
               (255, 255, 255), 1, cv.LINE_AA)

cv.imshow('Original image', BGR)
cv.waitKey(0)
cv.destroyAllWindows()
```





Yansıtıcı işaret etiketleme



7 işaret



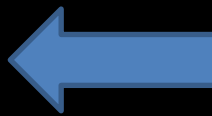
Yansıtıcı işaret etiketleme

```
totalLabels, label_ids, values, centroid = \
    cv.connectedComponentsWithStats(bw)

# put text and highlight the center
for i in range(1, len(centroid)):

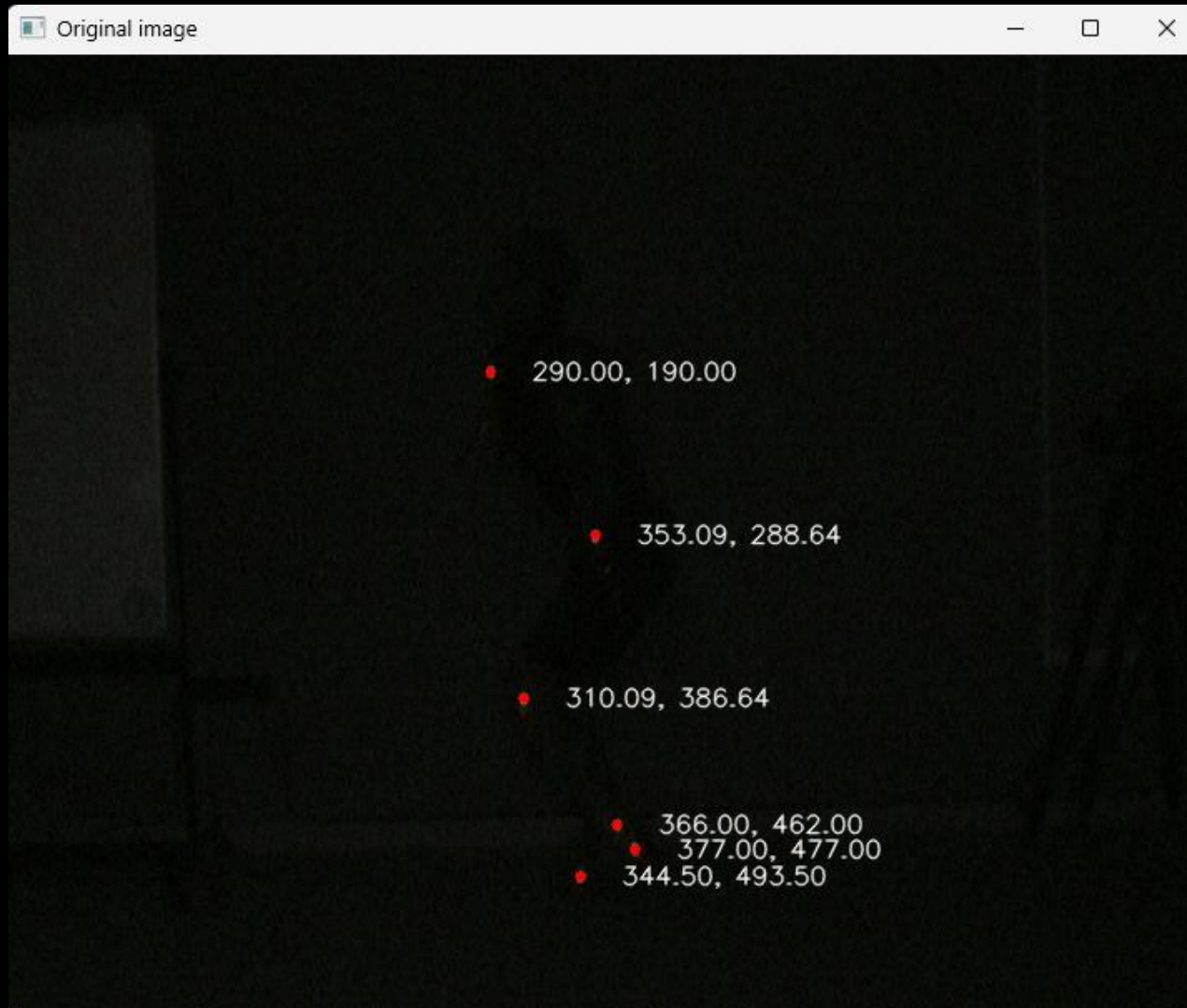
    cv.circle(BGR, (int(centroid[i][0]), \
                      int(centroid[i][1])), 3, (0,0,255), -1)
    cv.putText(BGR, "{:.2f}, {:.2f}".format(centroid[i][0], \
                      centroid[i][1]), \
               (int(centroid[i][0]) + 25, int(centroid[i][1]) + 5), \
               cv.FONT_HERSHEY_SIMPLEX, 0.5, \
               (255, 255, 255), 1, cv.LINE_AA)

cv.imshow('Original image', BGR)
cv.waitKey(0)
cv.destroyAllWindows()
```





Yansıtıcı işaret etiketleme



Yansıtıcı işaret etiketleme

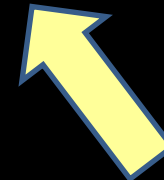
Video Yaratma

```
import cv2 as cv
import os

path = os.getcwd()
files = []
myColor = [(0, 0, 255), (0, 255, 0), (255, 0, 0), \
            (0, 255, 255), (255, 255, 0), (255, 0, 255) ]

# r=root, d=directories, f = files
files = [os.path.join(r, file) \
         for r, d, f in os.walk(path) for file in f if 'jump_1' in file]

img = cv.imread(files[0], cv.IMREAD_UNCHANGED)
vidsize = (img.shape[1],img.shape[0])
out = cv.VideoWriter('jumpLabel.avi', \
                    cv.VideoWriter_fourcc(*'MJPG'), 15, vidsize)
```



Yansıtıcı işaret etiketleme

fourcc.org

FourCC is a 4-byte code used to specify the video codec. The list of available codes can be found in fourcc.org. It is platform dependent.

- In Fedora: DIVX, XVID, MJPG, X264, WMV1, WMV2. (XVID is more preferable. MJPG results in high size video. X264 gives very small size video)
- In Windows: **DIVX** (More to be tested and added)
- In OSX: **MJPG** (.mp4), **DIVX** (.avi), **X264** (.mkv).

FourCC code is passed as

```
cv.VideoWriter_fourcc('M','J','P','G') or  
cv.VideoWriter_fourcc(*'MJPG')`for MJPG.
```

```
for i, f in enumerate(files):
    print(f"Image {i+1} : {os.path.basename(f)} is being processed.")
    # Read the image
    BGR = cv.imread(f)
    if BGR is None:
        print(f'Could not open the image {f} or find it:', BGR)
        exit(0)

    # Convert to the gray
    imgray = cv.cvtColor(BGR, cv.COLOR_BGR2GRAY)
    # median filter salt&pepper
    filtered = cv.medianBlur(imgray, 3)
    # Convert to the Black & White binary image
    _,bw = cv.threshold(filtered, 127, 255, cv.THRESH_BINARY)

    totallabels, label_ids, values, centroid = cv.connectedComponentsWithStats(bw)
    # put text and highlight the center
    for i in range(1, len(centroid)):

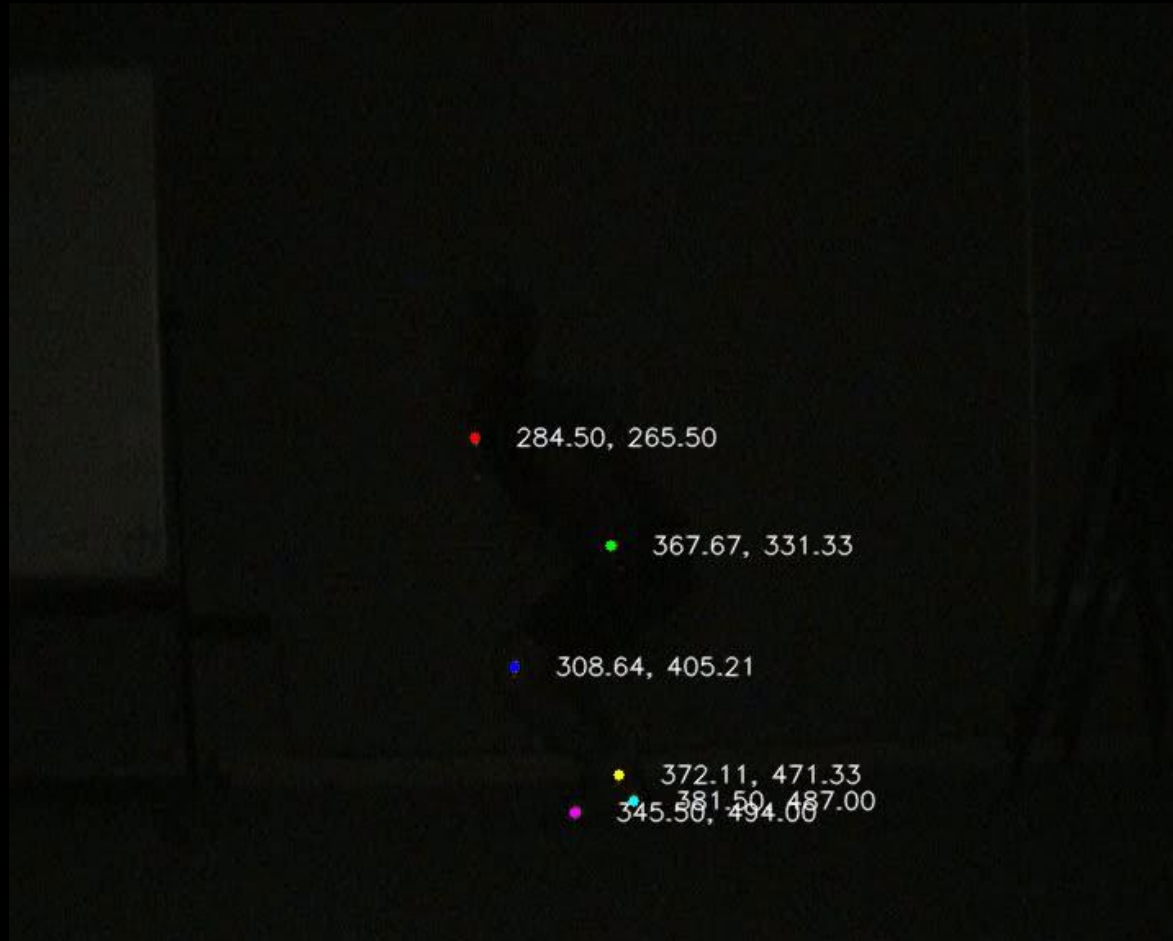
        cv.circle(BGR, (int(centroid[i][0]), int(centroid[i][1])), 3, myColor[i-1], -1)
        cv.putText(BGR, "{:.2f}, {:.2f}".format(centroid[i][0], centroid[i][1]), \
                    (int(centroid[i][0]) + 25, int(centroid[i][1]) + 5), \
                    cv.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1, cv.LINE_AA)

    cv.imshow('Original image', BGR)
    out.write(BGR)
    cv.waitKey(100)
out.release()
cv.waitKey(5)
cv.destroyAllWindows()
```



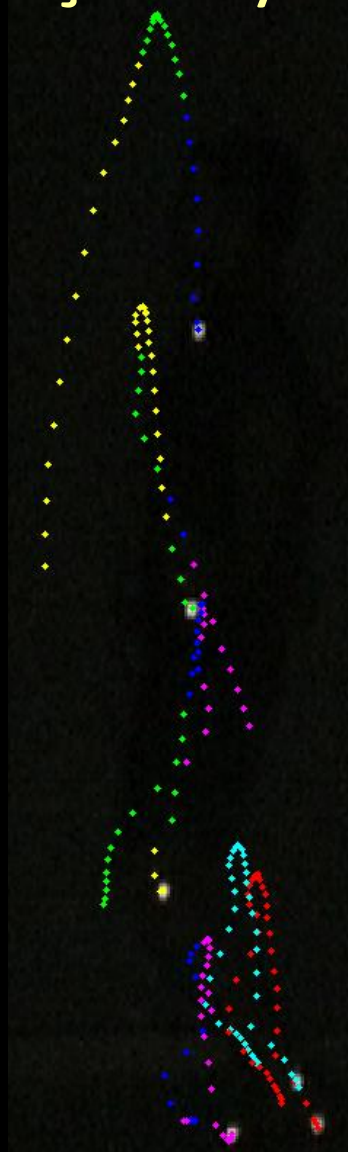

Yansıtıcı işaret etiketleme

Video Yaratma





Yansıtıcı işaret yakalama





Yansıtıcı işaret etiketleme

Sınıf Çalışması : Bisiklet Pedal Cevirme



Yansıtıcı işaret etiketleme

Sonuç : Bisiklet Pedal Cevirme





Yansıtıcı işaret etiketleme

Ev Ödevi 1: Geriye Salto hareketi yapan sporcunun eklem koordinatlarını bulan bir program yazınız?

1.Omuz

2.Dirsek

3.El Bileği

4.Kalça

5.Diz

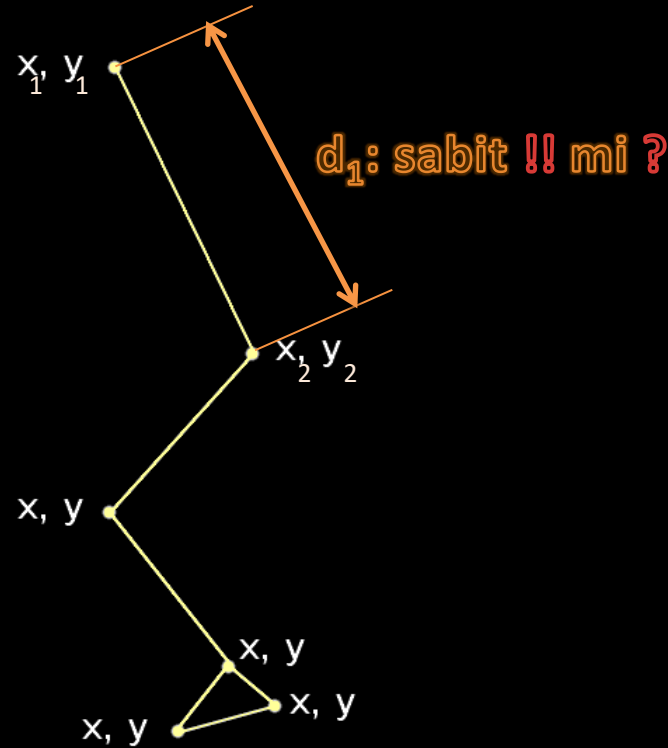
6.Ayak Bileği

7.Ayak Ucu



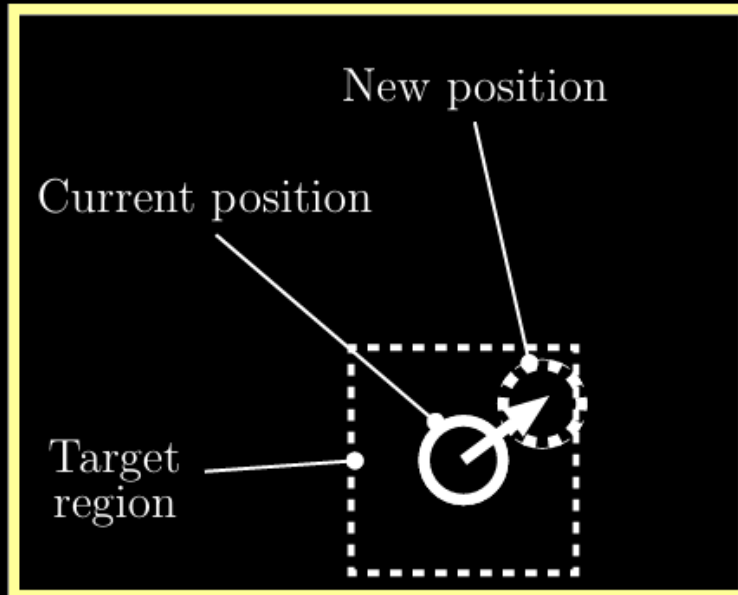
Yansıtıcı işaret etiketleme

Ev Ödevi 1: Geriye Salto hareketi yapan sporcunun eklem koordinatlarını bulan bir program yazınız?

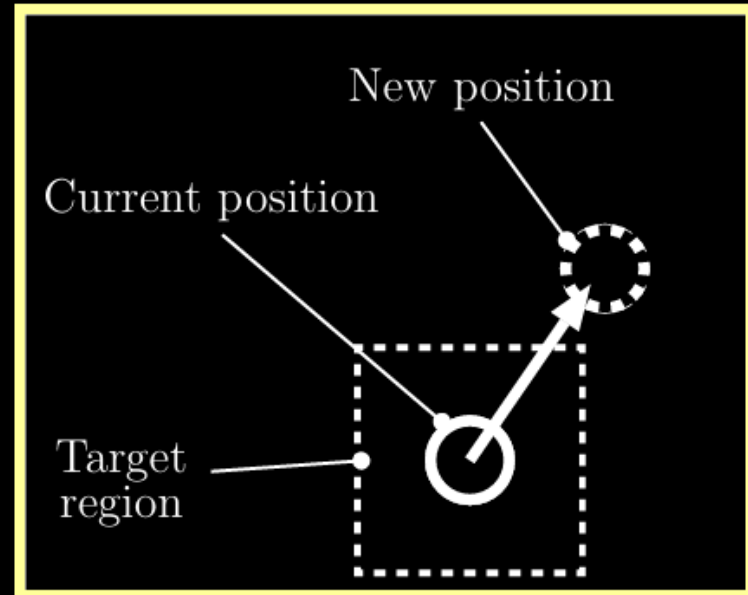


Yansıtıcı işaret takibi

Case 1



Case 2



Case 1: Tracking the object without position prediction might be successful,
Case 2: Tracking without position prediction will fail

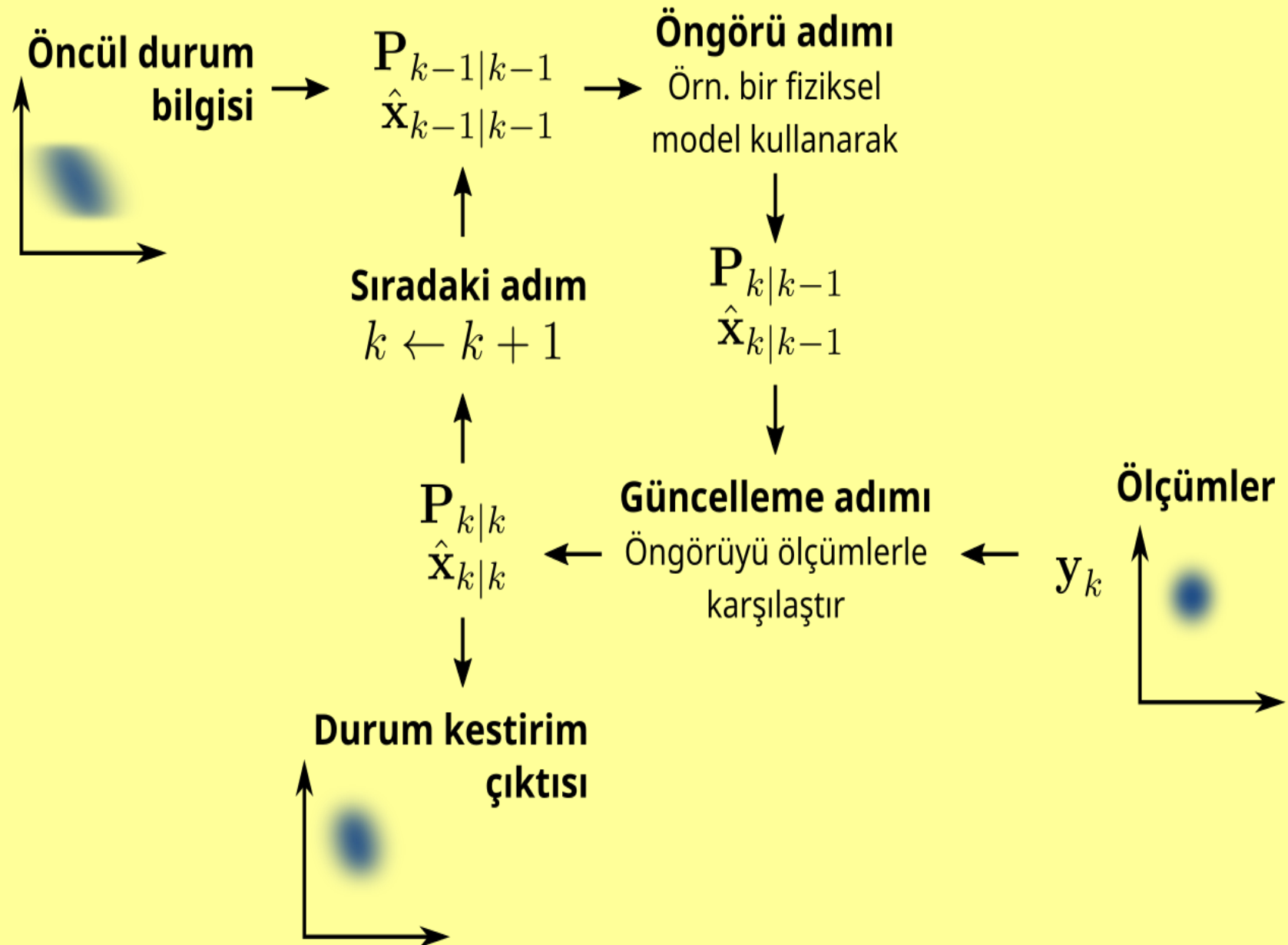
- The object is moving too fast.
- The frame rate is too low.
- The searched region is too small.

Yansıtıcı işaret etiketleme

Kalman Tahmin Edicisi işe yarar mı?

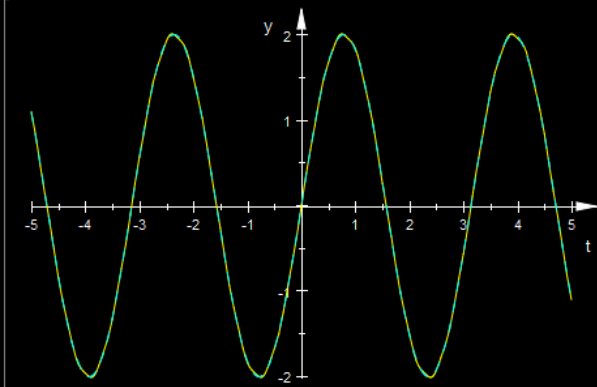
Kalman Filtresi, durum uzayı modeli ile gösterilen bir dinamik sistemde, modelin önceki bilgileriyle birlikte giriş ve çıkış bilgilerinden sistemin durumlarını tahmin edilebilen filtredir. Macar asıllı Amerikan matematiksel sistem teoristi Rudolf Kalman tarafından bulunmuştur.

<https://machinelearningspace.com/2d-object-tracking-using-kalman-filter/>
<https://github.com/RahmadSadli/2-D-Kalman-Filter>

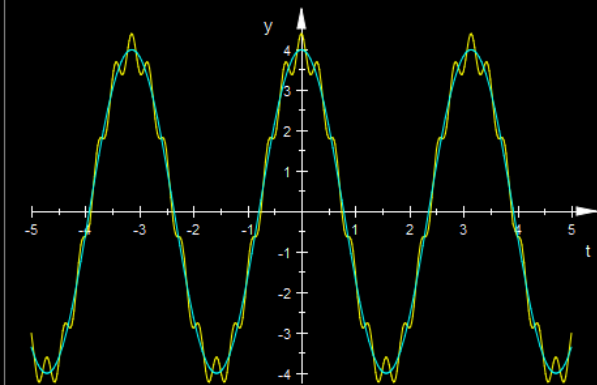


İvme Hesaplamaları Gürültüye Yatkındır

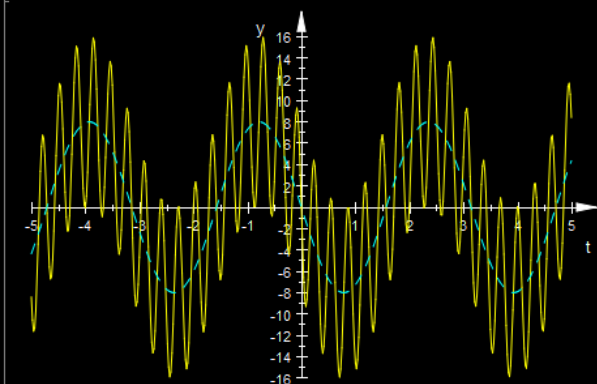
```
plot(2*sin(2*t)+0.02*sin(20*t), 2*sin(2*t))
```



```
plot(diff(2*sin(2*t)+0.02*sin(20*t), t), diff(2*sin(2*t), t))
```



```
plot(diff(diff(2*sin(2*t)+0.02*sin(20*t), t), t), diff(diff(2*sin(2*t), t), t))
```



$$r = \underbrace{2\sin(2t)}_{\text{Sinyal}} + \underbrace{0.02\sin(20t)}_{\text{Gürültü}} \Rightarrow \%1 \text{ Gürültü}$$

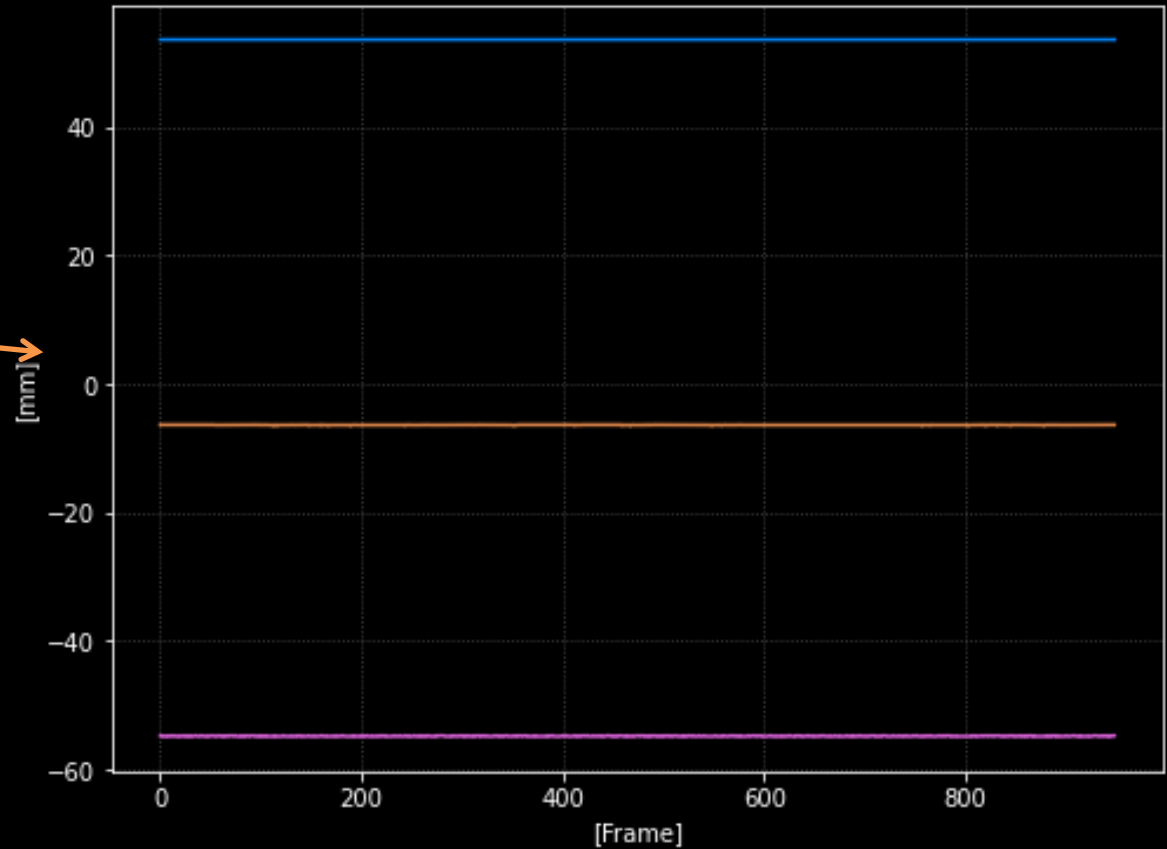
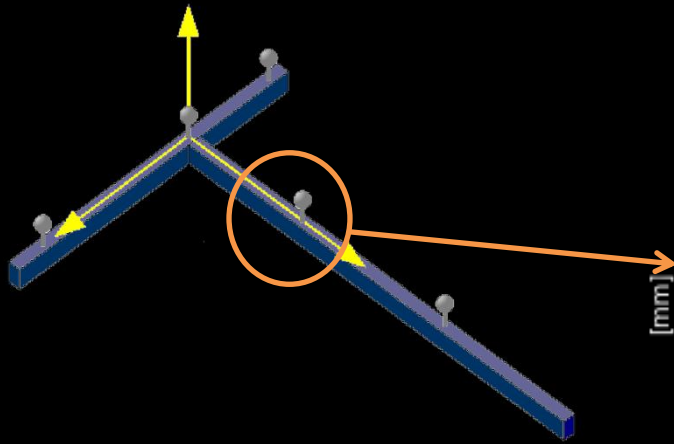


$$\frac{dr}{dt} = \underbrace{4\cos(2t)}_{\text{Sinyal}} + \underbrace{0.4\cos(20t)}_{\text{Gürültü}} \Rightarrow \%10 \text{ Gürültü}$$



$$\frac{d^2r}{dt^2} = \underbrace{-8\sin(2t)}_{\text{Sinyal}} + \underbrace{-8\sin(20t)}_{\text{Gürültü}} \Rightarrow \%100 \text{ Gürültü}$$

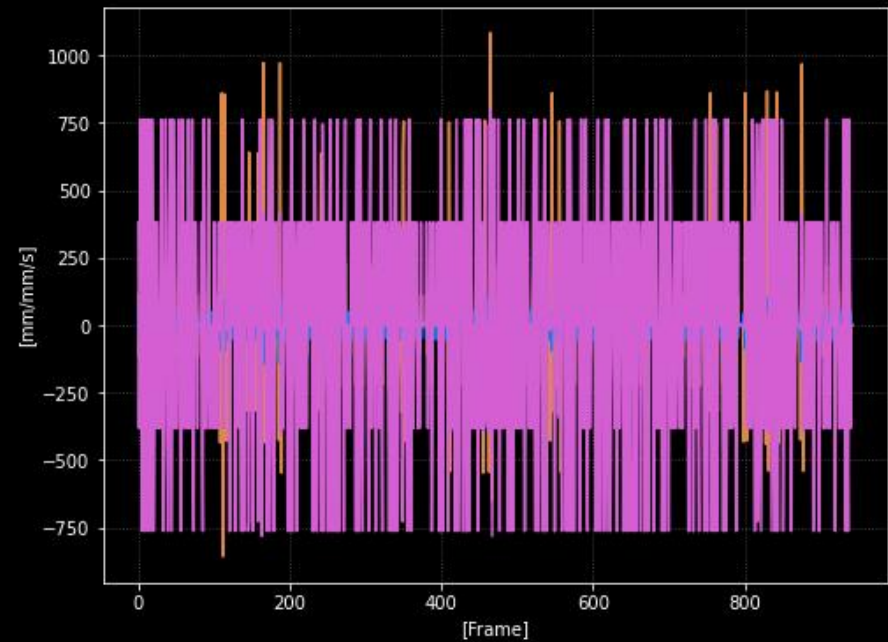
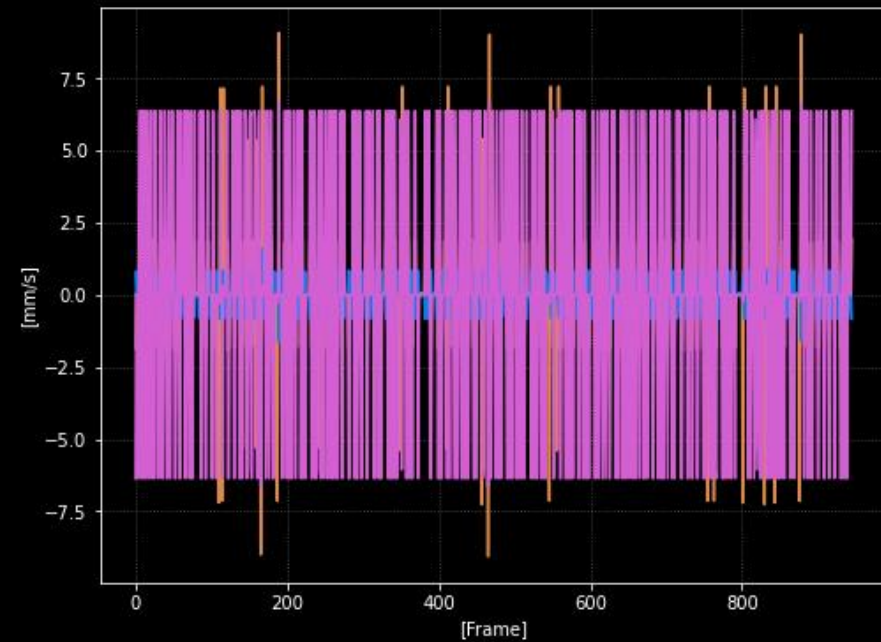
Vicon Kalibrasyon Çubuğu: hareketsiz



Vicon Kalibrasyon Çubuğu: hareketsiz

hız

ivme

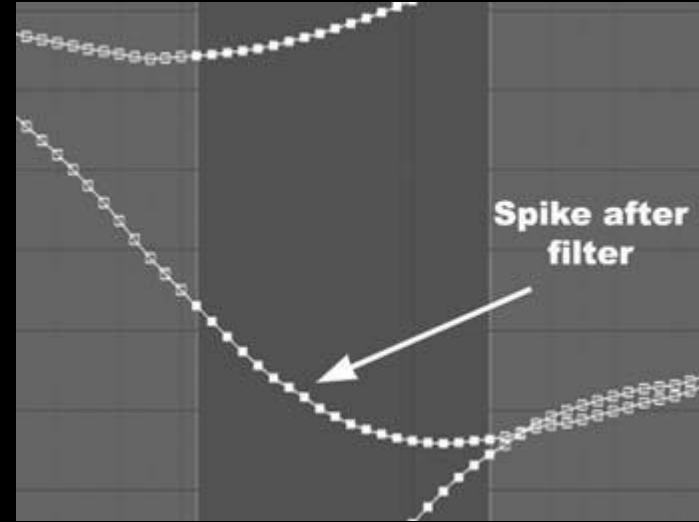
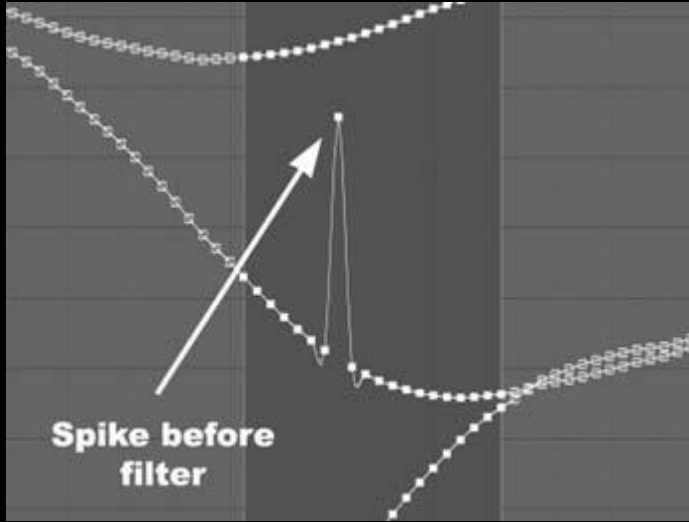


Süzgeçler (Filters)

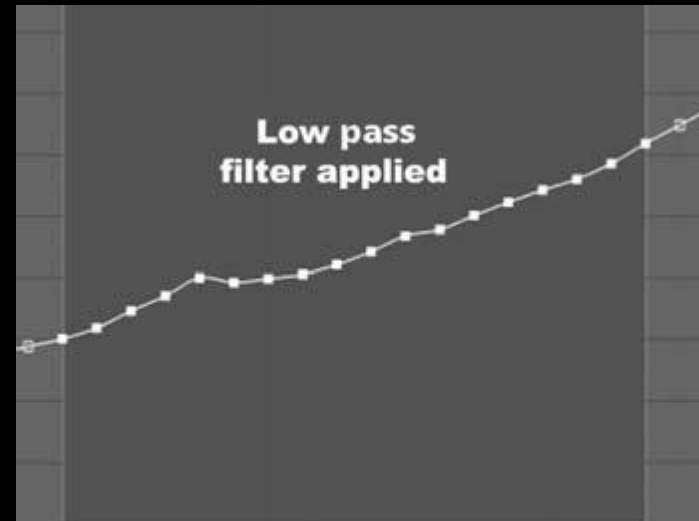
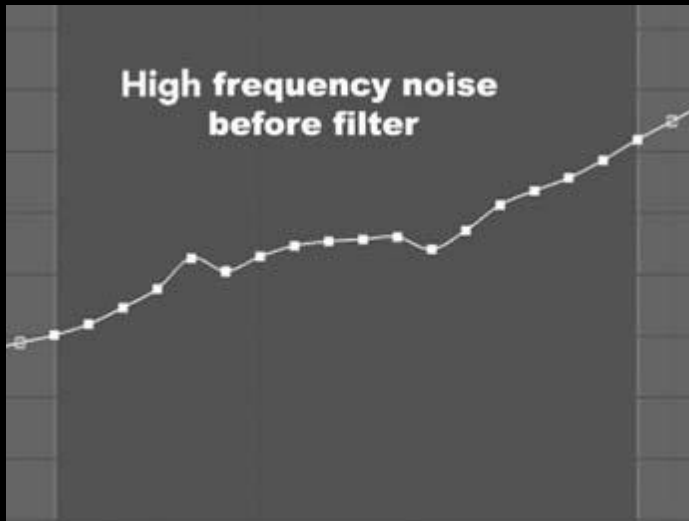
Hareket yakalama tamamlandıktan sonra yapılacak ilk iş verinin temizlenmesi olacaktır. Süzgeç (Filter), verideki yüksek veya alçak frekansları azaltan matematiksel bir uygulamadır. Süzgeç verinin bütünlüğünü bozmadan ani değişimleri veya sürüklenmeleri veriden ayıklar. Günümüzde hareket analizi verilerinde birçok farklı süzgeç algoritması kullanılmaktadır. Hareket yakalama sistemi herhangi bir sebeple yansıtıcıları göremediğinde yanlış bir tahmin yaparak veride ani bir değişime (spike) sebep olur. Alçak geçiren süzgeçler (Low-pass filters) bu ani yükselişleri hızlı bir şekilde ortadan kaldırır. Ayrıca, kötü kalibrasyon yapıldığı durumlarda yüksek frekanslı gürültüler hareket analizi verilerine de eklenir. **Butterworth filter** alçak geçiren süzgeç için çok güzel bir örnek olacaktır



Süzgeçler (Filter)



Her filtrenin kullanım amacına göre değiştirebileceğiniz kendisine özgü işlevleri ve ayarları vardır.



Süzgeçler (Butterworth Filter)

SciPy kütüphanesi içerisinde sinyal işleme kısmında bulunan (`scipy.signal.butter`) butterworth analog ve sayısal süzgeç tasarımı için `butter` komutu kullanılır.

```
scipy.signal.butter(N,Wn,btype='low',analog=False,output='ba',fs=None)
```

Butter komutu alçak geçiren, band geçiren, yüksek geçiren ve band durduran **n**. dereceden butterworth süzgeç tasarlamak için kullanılır. Buradaki **n** filtrenin derecesidir. **Butter** komutu **n+1** uzunluğunda olan ve filtre katsayılarını içeren **b** (numerator), **a** (denominator) vektörlerini geri döndürür. **Wn** kesim frekansıdır ve **Wn** değeri normalize edilmelidir. Örneğin kinematik verideki gibi 120Hz lik bir sinyalde 6 Hz i filtre ederek almak için 120 Hz in yarısı alınarak 6 ya bölünür. $Wn = 6 / 60 = 0.24$ olmalıdır.

```
# Creation of the Butterworth filter
```

```
N = 2          # Filter order
```

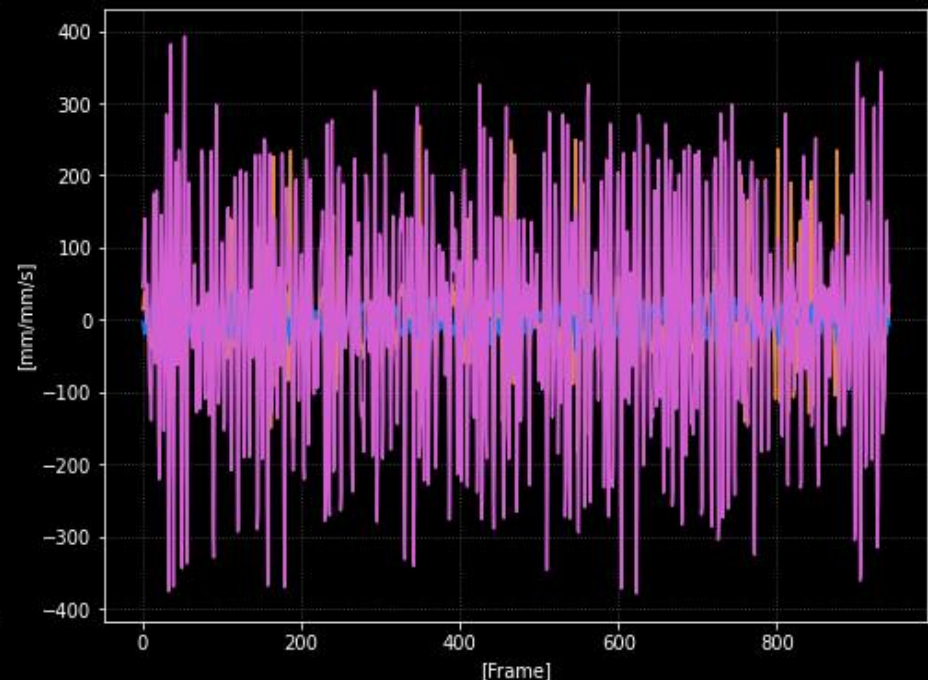
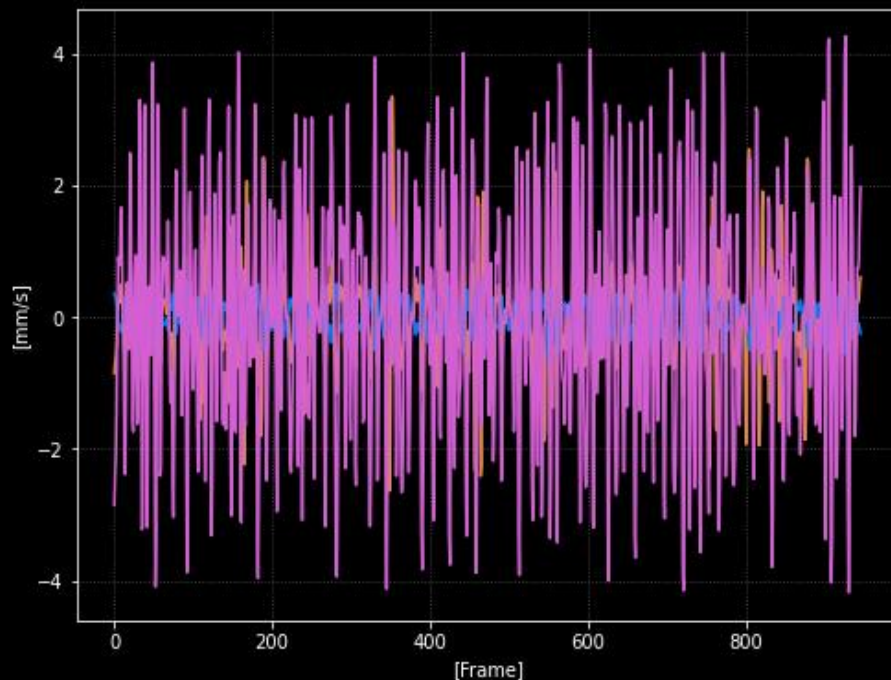
```
fc = 6/60      # Cutoff frequency, normalized
```

```
b, a = signal.butter(N, fc, 'low', analog=False)
```

```
filtered = signal.filtfilt(b=a, a=b, x=rawData.T).T
```

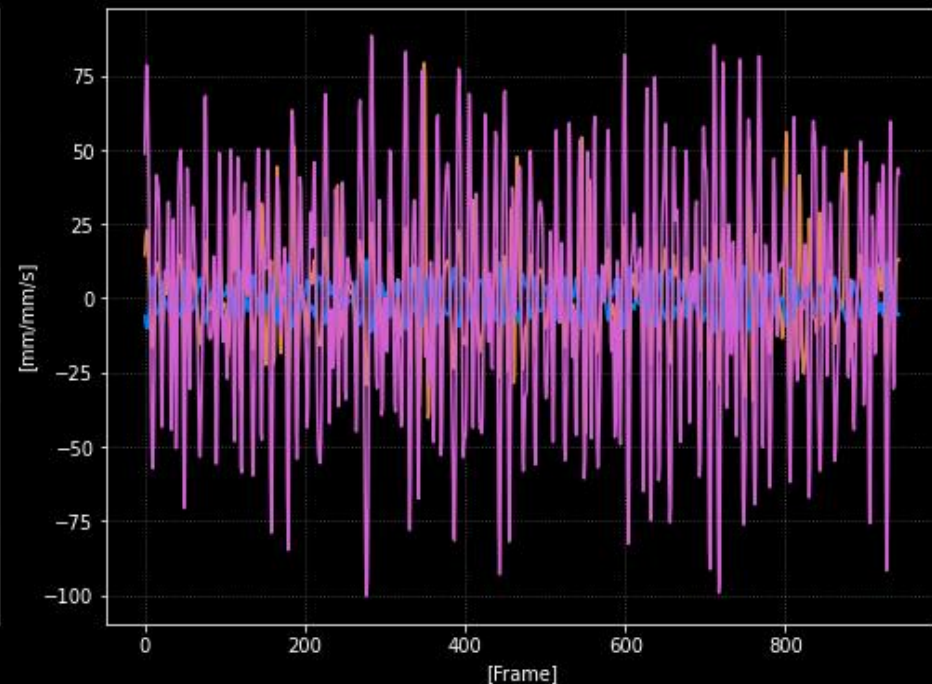
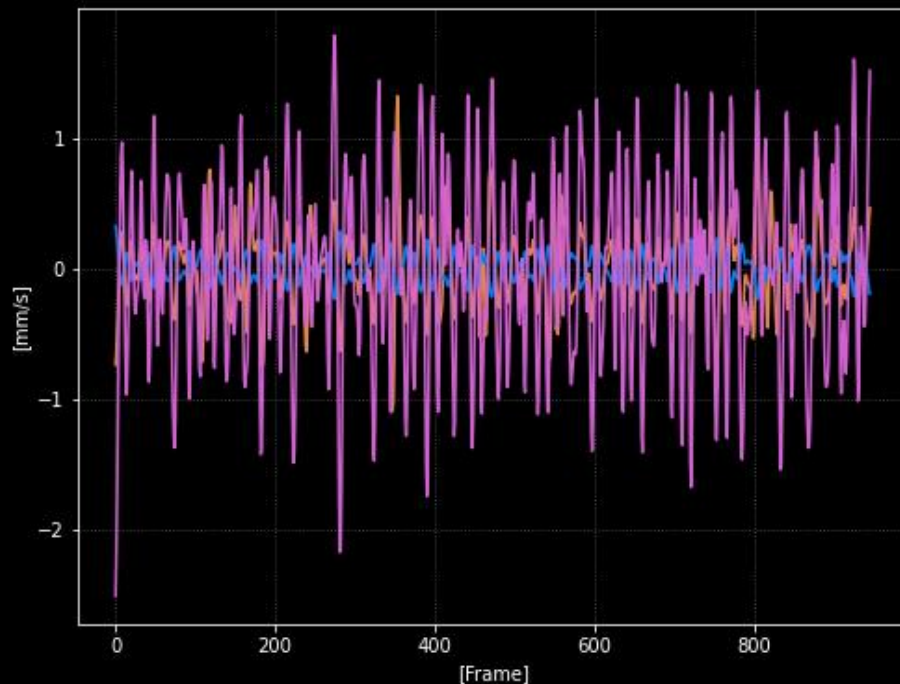

Süzgeçler (Butterworth Filter)

```
# Creation of the Butterworth filter  
N = 2          # Filter order  
fc = 20/60     # Cutoff frequency, normalized  
b, a = signal.butter(N, fc, 'low', analog=False)  
filtered = signal.filtfilt(b=a, a=b, x=rawData.T).T
```



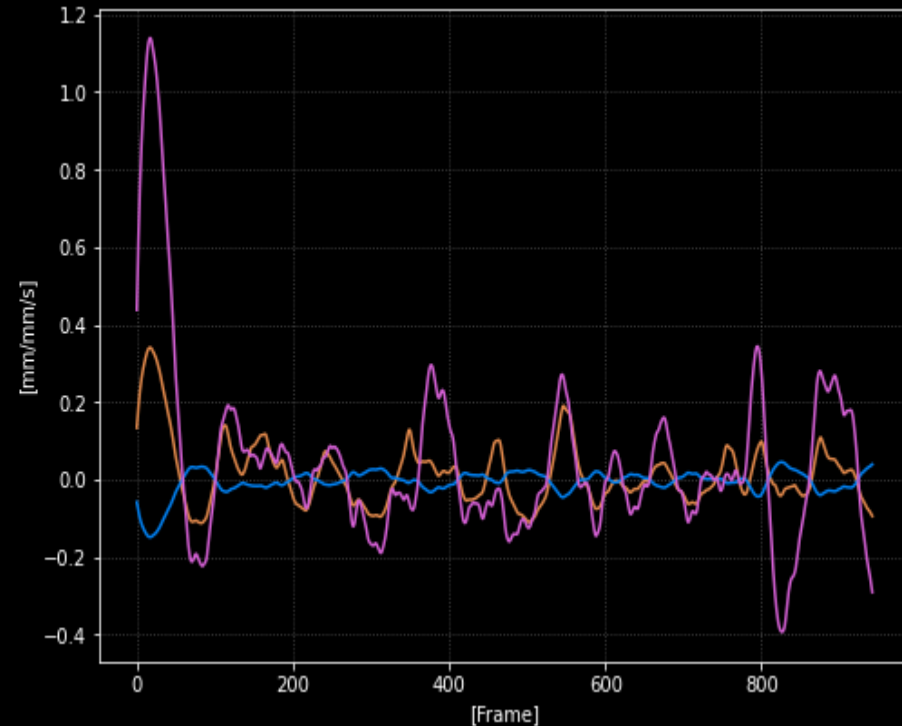
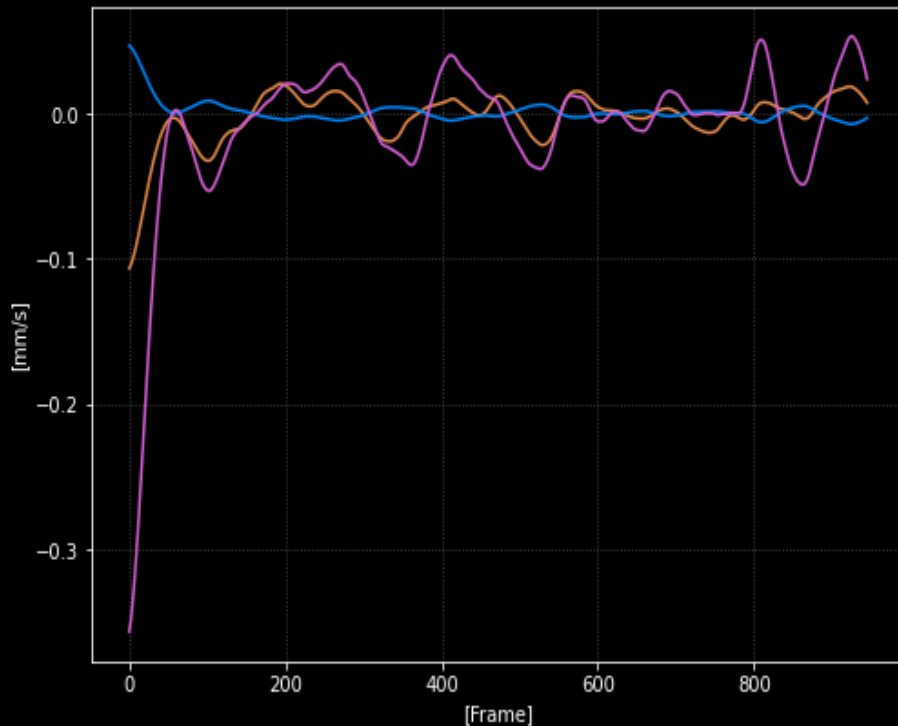
Süzgeçler (Butterworth Filter)

```
# Creation of the Butterworth filter  
N = 2          # Filter order  
fc = 10/60     # Cutoff frequency, normalized  
b, a = signal.butter(N, fc, 'low', analog=False)  
filtered = signal.filtfilt(b=b, a=a, x=rawData.T).T
```



Süzgeçler (Butterworth Filter)

```
# Creation of the Butterworth filter  
N = 2          # Filter order  
fc = 1/60      # Cutoff frequency, normalized  
b, a = signal.butter(N, fc, 'low', analog=False)  
filtered = signal.filtfilt(b=b, a=a, x=rawData.T).T
```



Süzgeçler (Butterworth Filter)

Örnek : El Hareketi için Farklı Süzgeç Kesme Frekansları



5 Hz Ağır Süzgeç (Heavy Filtering):

Hızlı ve ani hareketleri oldukça fazla yumuşatacaktır. Ancak normal kontrollü el hareketinin ana özelliklerini izlenebilecektir.

10 Hz Orta Süzgeçleme (Medium Filtering):

Elin normal ve orta hızlı hareketlerinin ana özelliklerini kaybolmaz iken bazı yüksek frekanslarda aralıklı bozulmalar olabilir.

15 Hz Hafif Süzgeçleme (Light Filtering):

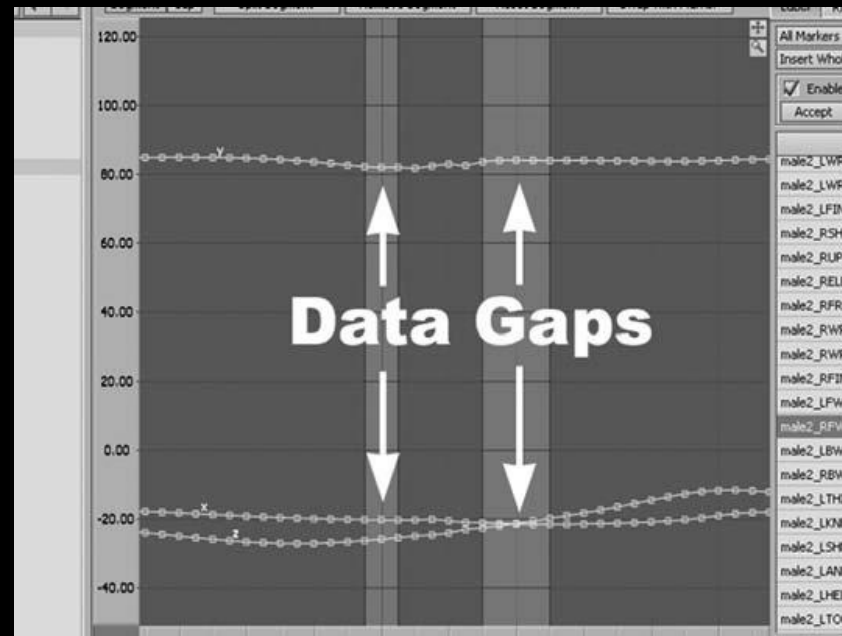
Elin normal ve hızlı hareketlerinin ana özelliklerini korumaktadır. Sadece çok yüksek hızlardaki ani hareketlerde bölgesel bozulmalar olabilir.

Süzgeçler (Butterworth Filter)

- Genellikle iki-yönlü filtre kullanılmalıdır, (**filtfilt**), aksi durumda süzgeçlenmiş veride faz farkı olacaktır.
- İki-yönlü filtre kullanıldığında süzgeçin derecesinin (order) 2 katı arttığı unutulmamalıdır.
- Süzgeçlenmiş ve süzgeçlenmemiş veriler mutlaka çizilerek uygunluğu kontrol edilmelidir.
- Çalışma rapor edildiğinde kullanılan süzgeçin tipi (alçak-yüksek), kesme frekansı ve katsayısı belirtilmelidir. *Örneğin; kesme frekansı 20Hz olan 2. dereceden iki-yönlü alçak geçiren Butterworth süzgeç konum verisine uygulanmıştır.*
- Hız ve ivme hesaplamalarında süzgeçler konum verisine uygulanmalıdır.

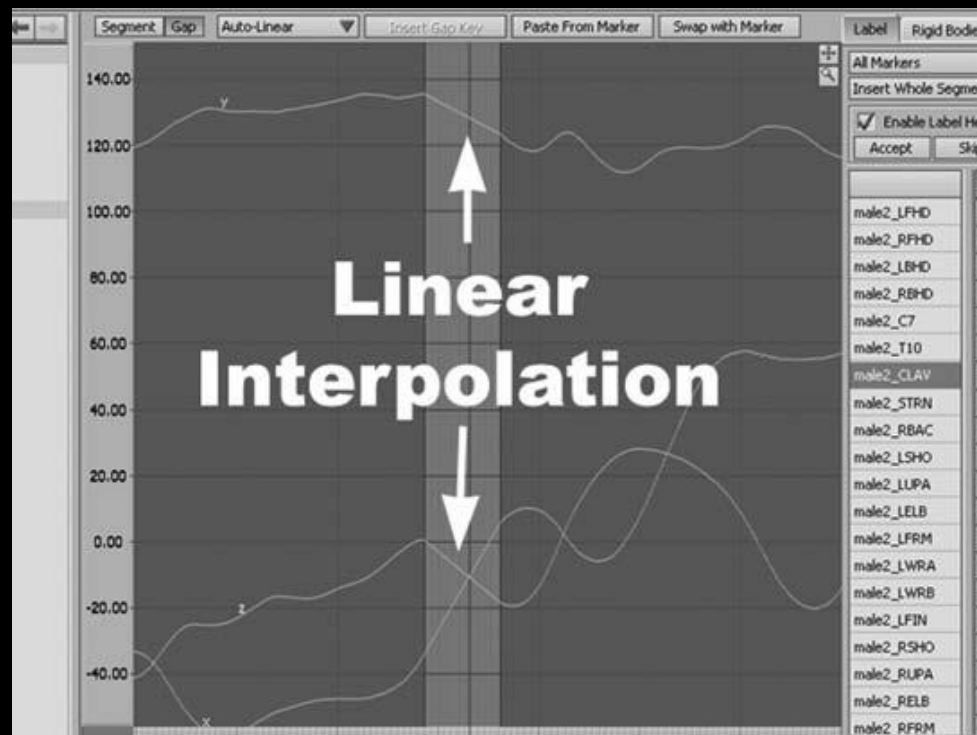
Kayıp Veri (Eliminating gaps)

The most common problem with optical mocap data is the gap caused by an absence of data or by removing an irregular peak that is a result of incorrect solution for computing marker locations. Gaps caused by the absence of data can occur for many different reasons, but occlusion is probably the most familiar one. One of the places where markers suffer from occlusion most frequently is the hands. A gap in the data looks like in the Figure below.



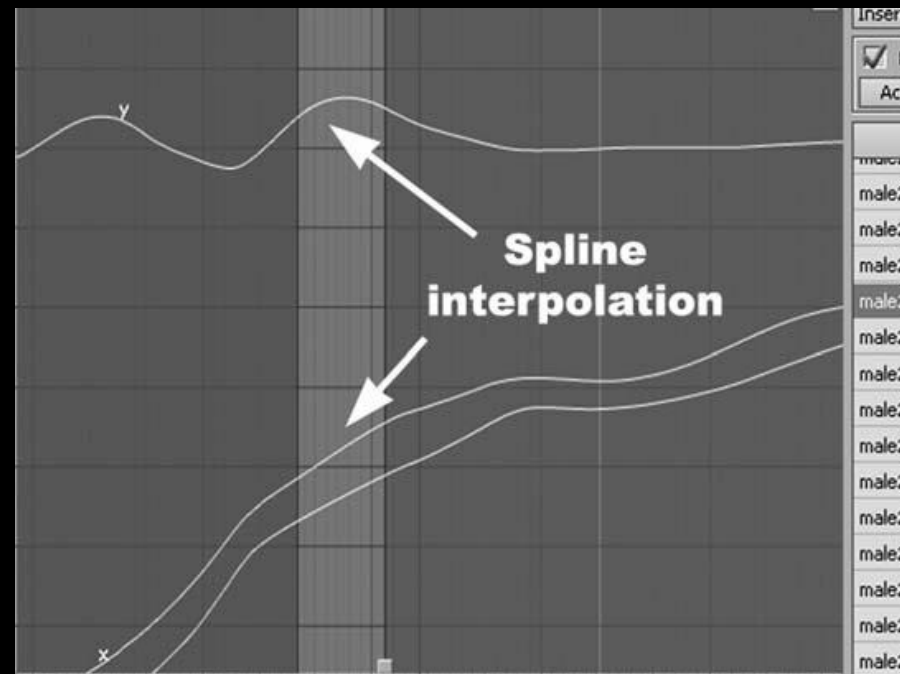
Kayıp Veri (Eliminating gaps)

The quickest and most direct way to fill the gap is using linear interpolation (Figure). Linear interpolation is connecting the last good data point before the gap and the first good data point after the gap with a straight line between them. There is no ease in or ease out but a straight line filling the gap. So, the resulting motion may look mechanical for that section.

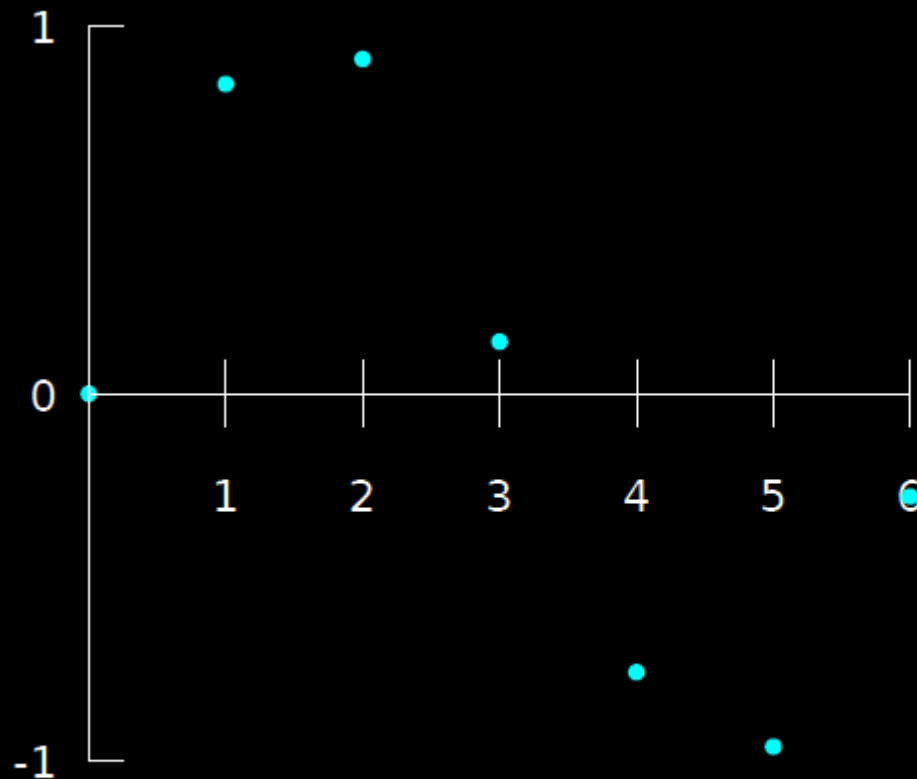


Kayıp Veri (Eliminating gaps)

Another way to fill the gap is using spline interpolation (Figure 4.3). The type of spline most commonly implemented for a graphical data editing tool is a cubic spline defined by the positions and tangents of two end points. Spline tools normally match the tangent of the curve's beginning point to the tangent of the last data point before the gap and match the tangent of the curve's end point to the tangent of the first data point after the gap.



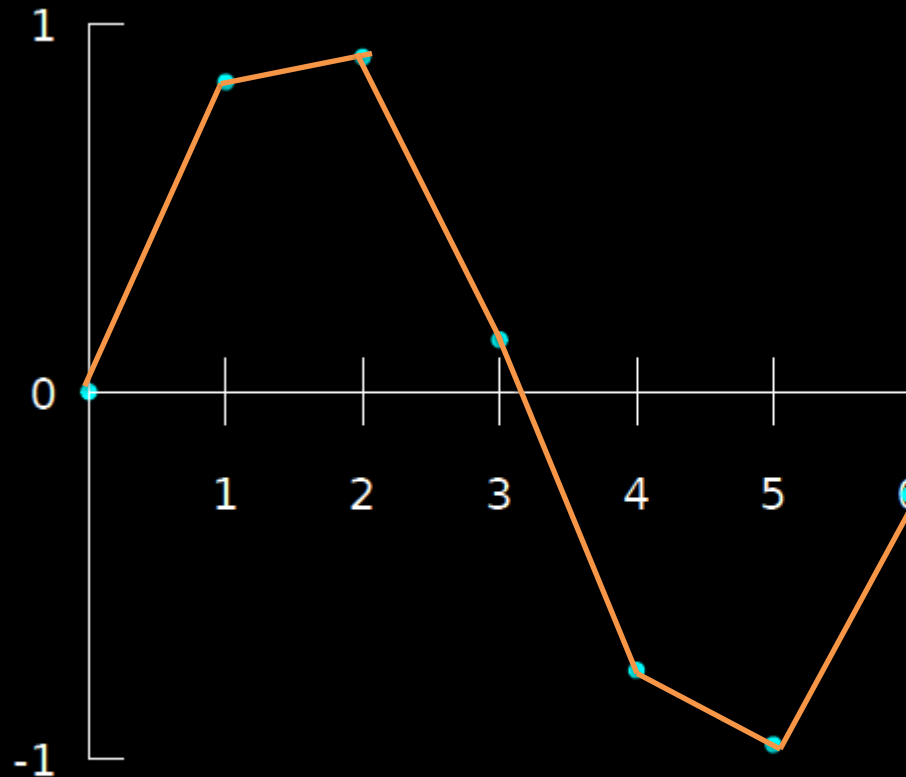
Kayıp Veri (Eliminating gaps)



- Interpolation is a way of filling in the gaps

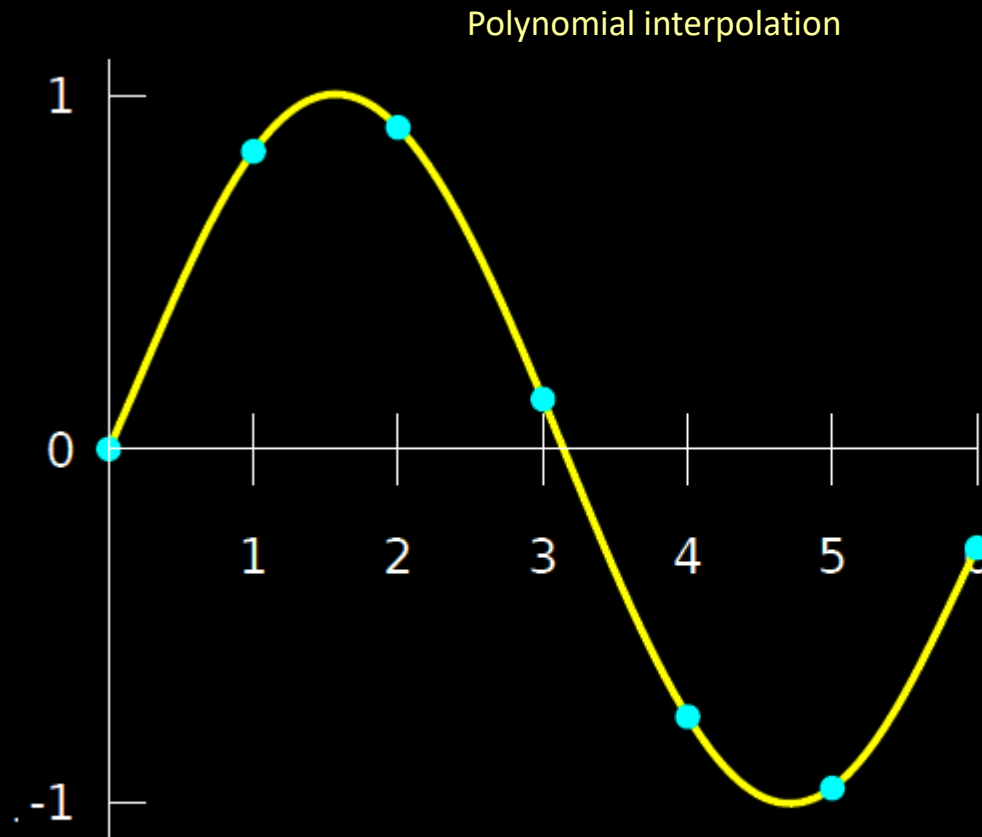
Kayıp Veri (Eliminating gaps)

Linear interpolation: Basically draw a line between the two nearest points



- Interpolation is a way of filling in the gaps

Kayıp Veri (Eliminating gaps)



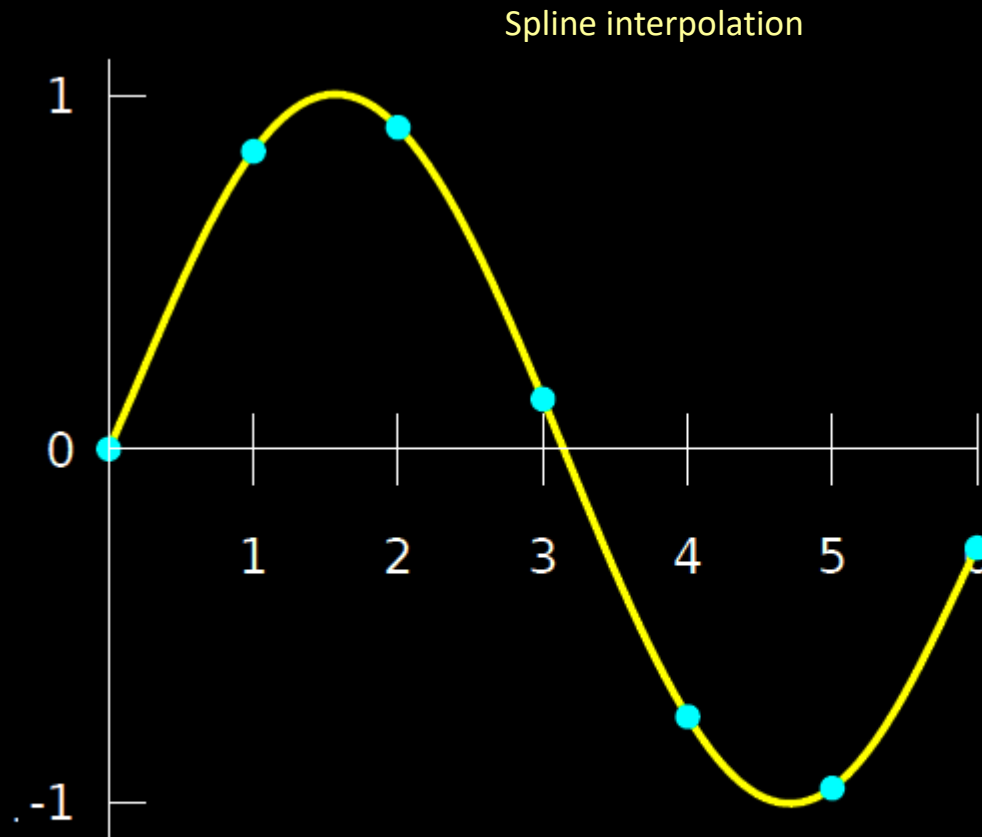
- Interpolation is a way of filling in the gaps

Kayıp Veri (Eliminating gaps)

Polynomial interpolation

- Draw a polynomial through all the data points
- Need a polynomial of degree one less than the number of points
- In general is problematic - noisy data can cause very strange results, so usually not used for movement data

Kayıp Veri (Eliminating gaps)



- Interpolation is a way of filling in the gaps

Kayıp Veri (Eliminating gaps)

Spline interpolation

- Use low order polynomials (typically 3rd order - cubic) between each pair of points, and match the gradient at the data points (so it will be smooth)
- Missing samples can be estimated using spline interpolation
- **Splines** can be used also to **smooth the data**
- If less splines are used than there are data points, the splines will not go through every point.

Kayıp Veri (Eliminating gaps)

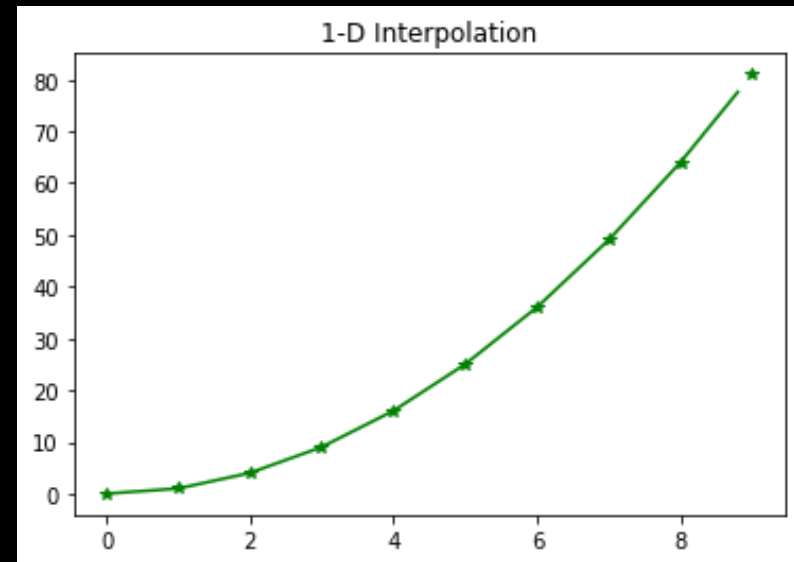
1-D Interpolation

```
import matplotlib.pyplot as plt  
from scipy import interpolate  
import numpy as np
```

```
### 1-D Interpolation  
# Initialize input values x and y  
x = np.arange(0, 10)  
y = x**2
```

```
# Interpolation  
temp = interpolate.interp1d(x, y)  
xnew = np.arange(0, 9, 0.2)  
ynew = temp(xnew)
```

```
plt.title("1-D Interpolation")  
plt.plot(x, y, '*', xnew, ynew, '-', color="green")  
plt.show()
```



Spline Interpolation

```
%% Spline Interpolation

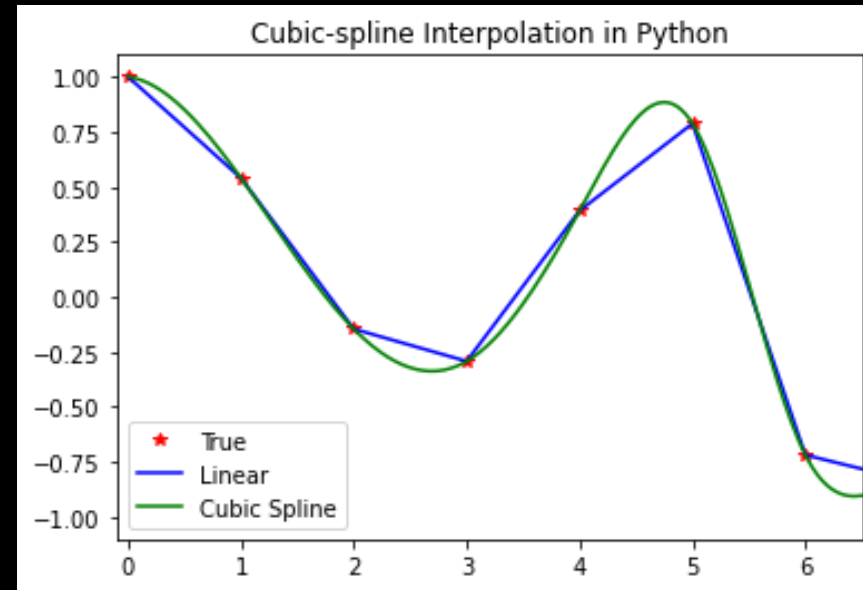
# Initialize the input values
x = np.arange(0, 10)
y = np.cos(x**3)

# Interpolation
# To find the spline representation of a
# curve in a 2-D plane using the function
# splrep
temp = interpolate.splrep(x, y, s=0)
xnew = np.arange(0, np.pi**2, np.pi/100)
ynew = interpolate.splev(xnew, temp, der=0)

plt.figure()

plt.plot(x, y, 'r*', \
         x, y, 'b', \
         xnew, ynew, 'g')

plt.legend(['True', 'Linear', 'Cubic Spline'])
plt.axis([-0.1, 6.5, -1.1, 1.1])
plt.title('Cubic-spline Interpolation in Python')
plt.show()
```



Univariate Spline Interpolation

```
%% Univariate Spline
# The scipy.interpolate.UnivariateSpline is used to fit a spline
# y = spl(x) of degree k to the provided x, y data. s specifies
# the number of knots by specifying a smoothing condition.

rng = np.random.default_rng()
x = np.linspace(-3, 3, 50)
y = np.exp(-x**2) + 0.1 * rng.standard_normal(50)
plt.plot(x, y, 'ro', ms=5)

# Use the default value for the smoothing parameter
spl = interpolate.UnivariateSpline(x, y)

xs = np.linspace(-3, 3, 1000)
plt.plot(xs, spl(xs), 'g', lw=3)

# Manually change the amount of smoothing
spl.set_smoothing_factor(0.5)
plt.plot(xs, spl(xs), 'b', lw=3)
plt.show()
```

