



T.C.
KARADENİZ TEKNİK ÜNİVERSİTESİ
FAKÜLTE ÖĞRENCİ STAJ DOSYASI

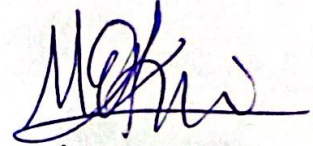
ÖĞRENCİNİN

ADI SOYADI : Osman Can AKSOY
BÖLÜMÜ : Mühendislik Fakültesi
PROGRAMI : Bilgisayar Mühendisliği
SINIFI : 3. Sınıf
NUMARASI : 394797

STAJ DÖNEMİ: 17/07/2023 - 25/08/2023

STAJ KOMİSYONU BAŞKANLIĞINA


Bilgisayar Mühendisliği Bölümü öğrenciniz 394797 numaralı Osman Can AKSOY stajını kurumumuz bünyesinde çalışan Bilgisayar Mühendisi Murat EKİNCİ'nin gözetiminde tamamlamıştır. Bu belge öğrencinizin isteği üzerine düzenlenmiştir. Gereğini bilgilerinize arz/rica ederiz.



İmza ve Mühür
Tarih

11.10.2023

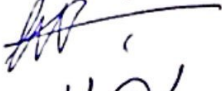
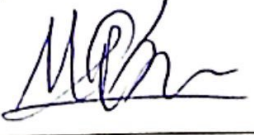
**17/07/2023 TARİHİNDEN 21/07/2023 TARİHİNE KADAR BİR HAFTALIK
ÇALIŞMA**

GÜNLER	YAPILAN İŞ	İZAHATIN BULUNDUĞU YAPRAK NO'SU
PAZARTESİ	Yapılacak Olan Çalışmanın Staj Sorumlusu ile Belirlenmesi ve Haftalık Çalışma Planının Oluşturulması	5
SALI	Object Tracking Algoritmaları Üzerine Araştırma	6
ÇARŞAMBA	Histogram Tabanlı Object Tracking Üzerine Araştırma	7
PERŞEMBE	OpenCV ile Web-Cam Üzerinden Belirli Sayıda Frameler Alınip Veri Seti Oluşturma	8, 9
CUMA	Resimde Belirli Bir Alanı Seçme	10
ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : 		

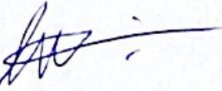

**24/07/2023 TARİHİNDEN 28/07/2023 TARİHİNE KADAR BİR HAFTALIK
ÇALIŞMA**

GÜNLER	YAPILAN İŞ	İZAHATIN BULUNDUĞU YAPRAK NO'SU
PAZARTESİ	Gri Seviye Resmin Histogramını Çıkarma	11
SALI	Genel Değerlendirme	12, 13
ÇARŞAMBA	Histogram Karşılaştırılma Üzerine Araştırma	14
PERŞEMBE	Kosinüs Benzerliği ve Kesişim Yöntemlerinin Kodlarını Yazma	15
CUMA	Öklit Uzaklığı ve Korelasyon Yöntemlerinin Kodlarını Yazma	16
ÖĞRENCİNİN İMZASI :  KONTROL EDENİN İMZASI : 		



**31/07/2023 TARİHİNDEN 04/08/2023 TARİHİNE KADAR BİR HAFTALIK
ÇALIŞMA**

GÜNLER	YAPILAN İŞ	İZAHATIN BULUNDUĞU YAPRAK NO'SU
PAZARTESİ	Histogram Karşılaştırma Yöntemlerini Test Etme	17, 18
SALI	Seçili Alanı Manuel Olarak Kaydırma ve Histogram Karşılaştırmasını Uygulama	19
ÇARŞAMBA	Program Testi ve Genel Değerlendirme	20
PERŞEMBE	Seçili Alanın Etrafında Search Alanı Oluşturma	21
CUMA	Search Alanı ve Tarama Yönteminin Kodunu Yazma	22
ÖĞRENCİNİN İMZASI : 		
KONTROL EDENİN İMZASI : 		

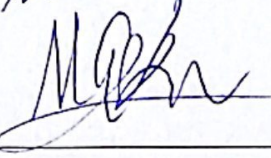
**07/08/2023 TARİHİNDEN 11/08/2023 TARİHİNE KADAR BİR HAFTALIK
ÇALIŞMA**

GÜNLER	YAPILAN İŞ	İZAHATIN BULUNDUĞU YAPRAK NO'SU
PAZARTESİ	Search Alanı ve Tarama Yöntemini Test Etme	23
SALI	Farklı Testler Gerçekleştirme ve Gözlemleme	24, 25
ÇARŞAMBA	Farklı Data Setler Oluşturma	26, 27
PERŞEMBE	Farklı Data Set Üzerinde Test Gerçekleştirme	28
CUMA	Staj Sorumlusu İle Genel Değerlendirme	29
ÖĞRENCİNİN İMZASI : 		
KONTROL EDENİN İMZASI : 		

**14/08/2023 TARİHİNDEN 18/08/2023 TARİHİNE KADAR BİR HAFTALIK
ÇALIŞMA**

GÜNLER	YAPILAN İŞ	İZAHATIN BULUNDUĞU YAPRAK NO'SU
PAZARTESİ	Kullanıcı Arayüzü Üzerine Araştırma	30
SALI	Kullanıcı Arayüzü Tasarlama	31, 32
ÇARŞAMBA	Tasarlanan Arayüzün Kodlarını Yazma	33
PERŞEMBE	Arayüz Üzerinde Histogram Değerlerini Grafikleştirme	34
CUMA	Arayüz Üzerinde Benzerlik Değerlerini Grafikleştirme	35
ÖĞRENCİNİN İMZASI : 		
KONTROL EDENİN İMZASI : 		

**21/08/2023 TARİHİNDEN 25/08/2023 TARİHİNE KADAR BİR HAFTALIK
ÇALIŞMA**

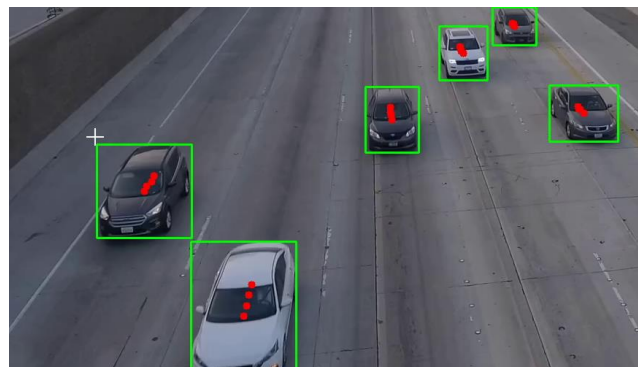
GÜNLER	YAPILAN İŞ	İZAHATIN BULUNDUĞU YAPRAK NO'SU
PAZARTESİ	Arayüz Testi ve Takip Algoritmasını Hızlandırma	36, 37
SALI	Spiral Tarama Algoritmasını Oluşturma	38
ÇARŞAMBA	Spiral Tarama Algoritmasının Kodunu Yazma	39, 40
PERŞEMBE	Threadler ve Mikroservis Düzeyinde Programlama Üzerine Araştırma	41, 42
CUMA	Object Tracking'de Thread Kullanımı Üzerine Araştırma	43
ÖĞRENCİNİN İMZASI : 		
KONTROL EDENİN İMZASI : 		

KISIM	Staj Döneminde Yapılacak Çalışmanın Belirlenmesi	Yaprak No:8
YAPILAN İŞ	Yapılacak Olan Çalışmanın Staj Sorumlusu ile Belirlenmesi ve Haftalık Çalışma Planının Oluşturulması	Tarih: 17/07/2023

Stajımın ilk iş günü olan 17 Temmuz tarihinde öğlen saat 12’de **KTU Bilgisayar Mühendisliği Bölümüne** gittim. Burada staj sorumlum olan **Prof. Dr. Murat EKİNCİ** ile staj sürecim hakkında konuşmalar yaptık. Staj sürecim boyunca yapacak olduğum proje üzerine tartıştık. C++ programla dili ile **OpenCV** kütüphanesi kullanarak **Object Tracking** alanında bir proje üzerinde hem fikir olduk. **Prof. Dr. Murat EKİNCİ** **Object Tracking** ile ilgili teknik bilgiler verdi. Daha sonra 1 haftalık çalışma planımda yapmam gerekenleri anlattı. Haftalık çalışmamda öncelikle **Object Tracking** alanında araştırmalar yapma, **Object Tracking** ile ilgili algoritmaları incelemeydi.



Geliştireceğim proje hakkında kısaca bahsedecek olursam başlangıçta bilgisayar Web-Cam’i üzerinden görüntüyü aktarma, aktarılan görüntüde manuel olarak bir alanı seçme, seçilen alanın histogramını hesaplama yapma.. Daha sonra Web-Cam üzerinden istenilen sayıda **Frame** diske kaydedip veri seti oluşturma. Oluşturulan veri setindeki **Frame’ler** üzerinde gezerek search alanı oluşturmak ve seçili alanı search alanında tarayarak yeni konumunu histogram karşılaştırması yaparak belirleme. Eğer seçili alanda değişim varsa seçili alanı güncelleyerek basit bir **Object Tracking** yapma.



(Temsili Object Tracking Algoritması)

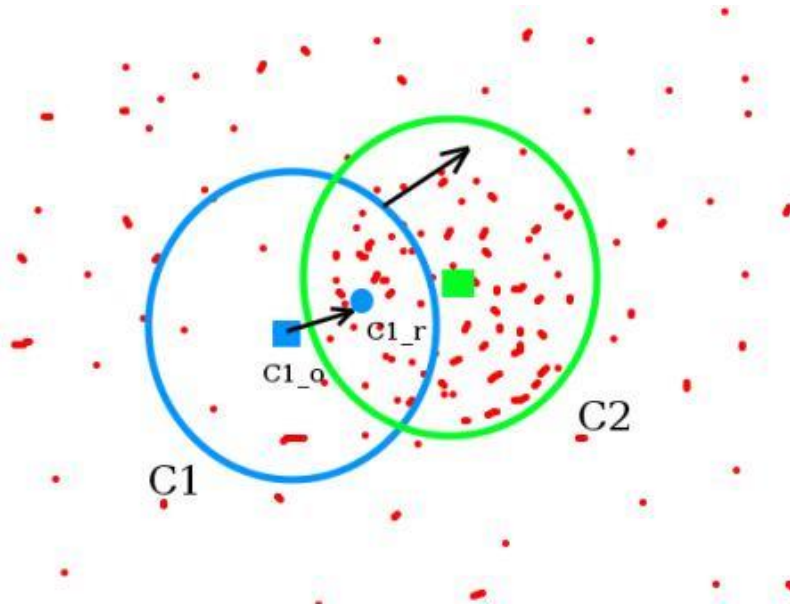
KONTROL SONUCU 

KISIM	Object Tracking Algoritmaları Araştırmaları	Yaprak No:9
YAPILAN İŞ	Object Tracking Algoritmaları Üzerine Araştırma	Tarih: 18/07/2023

Stajımın 2. Gününde **Object Tracking** algoritmalarını araştırmaya başladım. Araştırmalarım sonucunda **OpenCV** kütüphanesinde kullanılan iki tane **Object Tracking** algoritması olduğunu öğrendim. Bunlardan bir tanesi **MeanShift** diğeri de **CamShift**'dir.

MeanShift: Bir nesneyi bir video akışında ya da bir görüntüde otomatik olarak izlemek için kullanılan bir takip algoritmasıdır. Nesnenin renk dağılımını temel alarak çalışır. Çalışma prensibi, bir başlangıç penceresi (bölge) belirlemek ve bu pencere içindeki piksel değerlerinin renk dağılımını hesaplamaktır. Ardından, bu pencerenin ağırlık merkezini, yani piksellerin ağırlıklı ortalamasını bulur. Daha sonra, ağırlık merkeziyle başlangıç penceresi arasındaki farkı hesaplayarak yeni bir pencere konumunu belirler ve bu işlem tekrarlanarak nesnenin takip edildiği konumu günceller.

CamShift: **MeanShift** algoritmasının geliştirilmiş bir versiyonudur ve özellikle nesnenin boyutu değiştiğinde daha iyi performans gösterir. **CamShift**'in çalışma prensibi, **MeanShift**'in ağırlık merkezi hesaplama yöntemine dayanır, ancak ayrıca nesnenin boyutunu da adaptif olarak günceller. İlk adımda, **MeanShift** ile benzer şekilde, nesnenin takip edileceği başlangıç bölgesi (pencere) belirlenir ve bu bölgedeki piksel değerlerinin renk dağılımı hesaplanır. Daha sonra, her iterasyonda, bölgenin ağırlık merkezi hesaplanır ve bu merkeze göre yeni bir adaptif pencere boyutu belirlenir. Bu sayede, nesnenin boyutu değiştiğinde bile takip algoritması başarılı bir şekilde uyum sağlayabilir.



MeanShift Algoritması'nın Gösterimi

KONTROL SONUCU

KISIM	Object Tracking Algoritmaları Araştırmaları	Yaprak No:7
YAPILAN İŞ	Histogram Tabanlı Object Tracking Üzerine Araştırma	Tarih: 19/08/2023

Columbia Üniversitesi'nde bulunan Shree K. Nayar'ın yayınlamış olduğu First Principles of Computer Vision eğitim serisinde bulunan **Object Tracking using Template Matching** bölümünü izledim. Oluşturacağım algoritma histogram karşılaştırması sonucuna göre **Template Matching** yapmak. **Template Matching**'den bahsedecek olursam;

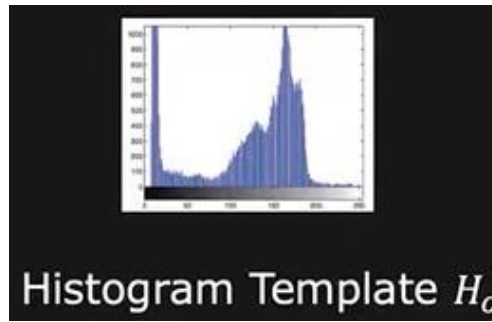
Histogram Tabanlı Template Matching, template matching yöntemlerinden biridir ve şablon ve görüntü arasındaki benzerliği histogramlar kullanarak belirlemeyi amaçlar. Bu yöntemde, görüntü ve şablonun renk dağılımı histogramları alınır ve bu histogramlar arasındaki benzerlik ölçütleri kullanılarak eşleşen bölgeler belirlenir.

Histogram Tabanlı Template Matching'in temel adımları şunlardır:

1. **Şablonun Belirlenmesi: Template Matching** işleminde olduğu gibi, nesneyi temsil eden bir şablon belirlenir. Şablon, küçük boyutta ve genellikle tek renkte olmalıdır.
2. **Histogramların Oluşturulması:** Hem şablonun hem de görüntünün renk dağılımını temsil eden histogramlar oluşturulur. Bu, renk kanallarına göre (örneğin RGB veya HSV) histogramların hesaplanması anlamına gelir.
3. **Histogramlar Arasında Benzerlik Ölçümü:** Şablon histogramı ve görüntü histogramı arasındaki benzerlik ölçütleri kullanılarak eşleşen bölgeler belirlenir. Benzerlik ölçütü olarak genellikle korelasyon katsayısı veya kesişim benzerliği gibi metrikler kullanılır.
4. **Eşleşen Bölgelerin Belirlenmesi:** Histogramlar arasındaki benzerlik ölçümü sonucunda, en yüksek benzerlik değerine sahip konumlar şablonun eşleşen bölgeleri olarak kabul edilir.



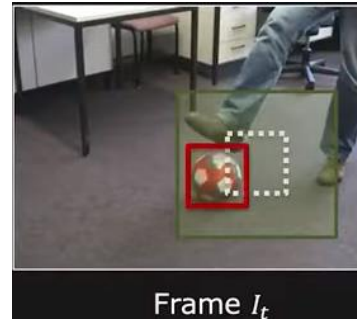
1- Şablonun Belirlenmesi



2 – Histogramların Oluşturulması




3 – Histogramlar Arasında Benzerlik Ölçümü



4 – Eşleşen Bölgelerin Belirlenmesi

KONTROL SONUCU 

KISIM	OpenCV ile Uygulamalar	Yaprak No:8
YAPILAN İŞ	OpenCV ile Web-Cam Üzerinden Belirli Sayıda Frameler Alınip Veri Seti Oluşturma	Tarih: 20/07/2023
<p>Bilgisayarıma OpenCV kütüphanesini yükleyerek sistem path'ine gerekli ayarlamaları Daha sonra OpenCV kütüphanesiyle Web-Cam'den görüntü alma işlemini gerçekleştirmek için aşağıdaki adımları takip ettim:</p> <p>1– Visual Studio Projesi Oluşturma Gerekli Kütüphanelerin Eklenmesi: C++ projesi oluşturdum ve OpenCV kütüphanesini projeme ekledim. C++ projesi için genellikle Visual Studio, Code::Blocks veya Cmake gibi geliştirme ortamları kullanılabilir. Ben kullanım rahatlığı açısından Visual Studio'yu tercih ettim.</p> <p>2 – Web-Cam Üzerinden Görüntü Alma Kodunu Yazma:</p> <pre> #include <opencv2/opencv.hpp> int main() { // Kamera nesnesini oluşturma cv::VideoCapture cap(0); // 0, birincil kamerayı temsil eder. Eğer birden fazla kamera varsa, 1, 2 gibi değerlerle diğer kameraları da seçebilirsiniz. // Kamera açılıp açılmadığını kontrol etme if (!cap.isOpened()) { std::cerr << "Kamera açılmadı!" << std::endl; return -1; } // Görüntü almak için sonsuz döngü while (true) { cv::Mat frame; cap >> frame; // Kameradan görüntüyü al // Görüntü alınmadıysa döngüyü sonlandır if (frame.empty()) { std::cerr << "Görüntü alınamadı!" << std::endl; break; } // Görüntüyü ekrana göster cv::imshow("Kamera Görüntüsü", frame); // Klavyeden 'q' tuşuna basıldığında döngüyü sonlandır if (cv::waitKey(1) == 'q') { break; } } // Kamera nesnesini serbest bırak cap.release(); // Pencereleeri kapat cv::destroyAllWindows(); return 0; } </pre>		
<p>KONTROL SONUCU </p>		

KISIM	OpenCV ile Uygulamalar	Yaprak No:9
YAPILAN İŞ	OpenCV ile Web-Cam Üzerinden Belirli Sayıda Frameler Alınip Veri Seti Oluşturma	Tarih:20/07/2023

Web-Cam üzerinden başarılı bir şekilde görüntü aldıktan sonra **Web-Cam** üzerinden belirli bir sayıda frame kaydederek veri setimi oluşturmak üzere **save_frames** adlı bir fonksiyon yazdım.

```
#include <opencv2/opencv.hpp>

void save_frames(cv::VideoCapture& cap, int frameSayisi, const std::string& klasorAdi) {
    // Kayıt için klasörü oluştur
    cv::utils::fs::createDirectory(klasorAdi);

    // Frame kaydetme döngüsü
    for (int i = 0; i < frameSayisi; i++) {
        cv::Mat frame;
        cap >> frame;

        if (frame.empty()) {
            std::cerr << "Goruntu alinamadi!" << std::endl;
            break;
        }

        // Frame'i diske kaydet
        std::stringstream dosyaAdi;
        dosyaAdi << klasorAdi << "/frame_" << i << ".jpg";
        cv::imwrite(dosyaAdi.str(), frame);
    }

    std::cout << frameSayisi << " frame kaydedildi." << std::endl;
}

int main() {
    // Kamera nesnesini oluşturma
    cv::VideoCapture cap(0); // 0, birincil kamerayı temsil eder. Birden fazla kamera varsa, 1, 2 gibi değerlerle diğer kameraları da seçebilirsiniz.

    // Kamera açılıp açılmadığını kontrol etme
    if (!cap.isOpened()) {
        std::cerr << "Kamera acilamadi!" << std::endl;
        return -1;
    }

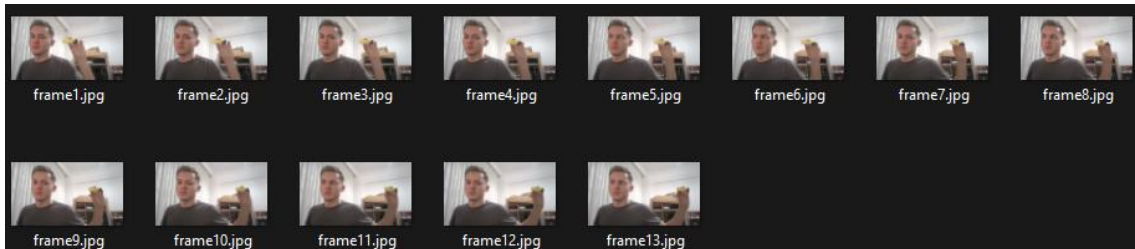
    int frameSayisi;
    std::cout << "Kac frame kaydedilecek? ";
    std::cin >> frameSayisi;

    std::string klasorAdi = "kaydedilen_frame'ler";
    save_frames(cap, frameSayisi, klasorAdi);

    // Kamera nesnesini serbest bırak
    cap.release();

    // Pencereleri kapat
    cv::destroyAllWindows();

    return 0;
}
```



Başlangıçta Oluşturmuş Olduğum Veri Seti

KONTROL SONUCU 

KISIM	OpenCV İle Uygulamalar	Yaprak No:10
YAPILAN İŞ	Resimde Belirli Bir Alanı Seçme	Tarih:21/07/2023

Öncelikle **OpenCV** kütüphanesiyle bir resimde belirli bir alanı nasıl seçebiliriz üzerine araştırma yaptım. Araştırma yaparken **OpenCV**'nin dökümantasyonundan yararlandım. Araştırmalarım sonucunda **OpenCV** kütüphanesinde bulunan **ROI(Region of Interest)**'i kullanmaya karar verdim.

ROI(Region of Interest):

Region of Interest (ROI) veya İlgiilenilen Bölge, bir görüntü içerisinde belirli bir bölgenin seçilmesini ve bu bölge üzerinde işlemlerin yapılmasını sağlayan bir tekniktir.

ROI(Region of Interest) Kullanım Alanları:

1. **Nesne Tespiti ve Takip:** ROI, nesne tespiti ve takip algoritmalarında kullanılarak ilgilenilen nesneyi belirlemek ve takip etmek için kullanılabilir.
2. **Nesne Tanıma ve Sınıflandırma:** ROI, görüntü içindeki önemli bölgeyi belirleyip nesne sınıflandırma algoritmaları için daha küçük ve odaklanmış bir alan oluşturmak için kullanılabilir.
3. **Görüntü Düzenleme:** ROI, görüntü üzerinde belirli bir bölgeyi değiştirmek veya düzenlemek için kullanılabilir. Örneğin, parlaklık, kontrast veya renk düzenlemesi yapılabilir.

ROI Oluşturma:

ROI oluşturmak için OpenCV kütüphanesinde **cv::Rect** (Dikdörtgen) türünden bir nesne kullanılır. Bu tür, başlangıç noktası (x, y) ve boyut (genişlik, yükseklik) bilgilerini içerir. ROI oluşturmak için izlenen adımlar şunlardır:

1. Başlangıç noktası (x, y) ve boyut (genişlik, yükseklik) belirlenir.
2. **cv::Rect** türünden bir nesne oluşturulur ve bu değerlerle başlatılır.
3. Görüntü matrisi üzerinde, oluşturulan dikdörtgeni kullanarak dilimleme işlemi yapılır.

```
#include <opencv2/opencv.hpp>

using namespace cv;

int main()
{
    //Görüntüyü yükle
    Mat frame = imread("test_images/frame0.png");

    //ROI ile görüntü üzerinden belirli bir alanı seçme
    Rect roi = selectROI("Select Area", frame);

    //Seçili alanı görüntü üzerinde işaretleme
    rectangle(frame, roi, Scalar(0, 255, 0), 2);

    //Görüntüyü göster
    imshow("Görüntü", frame);

    // Pencereleeri kapat
    destroyAllWindows();

    return 0;
}
```

KONTROL SONUCU 

KISIM	OpenCV İle Uygulamalar	Yaprak No:11
YAPILAN İŞ	Gri Seviye Resmin Histogramını Çıkarma	Tarih:24/07/2023

OpenCV ile okuduğum bir resmin histogramını hesaplamak için bir takım araştırmalar yaptım. Araştırmalarımda **OpenCV**'nin dökümantasyonu ve **stackoverflow**'dan yararlandım. Araştırmalarım sonucunda ilgili resmi önce gri seviyeye çevirip daha sonra her pikseli gezerek parlaklık değerlerini okuyup tek boyutlu oluşturduğum histogram dizisinde ilgili indisin değerini artırarak gri seviye resmin histogramını çıkardım. Aşağıda yapmış olduğum işlemlerin kodları mevcuttur.

```
#include <opencv2/opencv.hpp>
using namespace cv;

int main() {
    // Gri seviye bir resmi yükleyin
    Mat image = imread("deneme.jpg", IMREAD_GRAYSCALE);

    // Histogramı hesaplamak için bir dizi oluşturun
    int histogram[256] = { 0 };

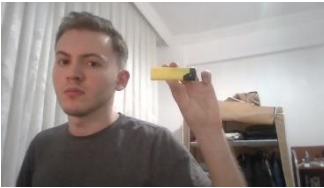
    // Resmin her pikselini dolaşarak histogramı doldurun
    for (int i = 0; i < image.rows; i++) {
        for (int j = 0; j < image.cols; j++) {
            int pixel_value = static_cast<int>(image.at<uchar>(i, j));
            histogram[pixel_value]++;
        }
    }

    // Histogramı ekrana yazdırın veya işlem yapın
    for (int i = 0; i < 256; i++) {
        cout << "Parlaklik Degeri " << i << ": " << histogram[i] << " piksel" << endl;
    }

    // OpenCV penceresini kapatın ve programı sonlandırın
    destroyAllWindows();

    return 0;
}
```

Örnek bir resim ve histogram değerlerinin hesaplanması, konsol üzerinde gösterilmesi:



deneme.jpg

```
Microsoft Visual Studio Debug Console
Parlaklik Degeri 0: 0 piksel
Parlaklik Degeri 1: 0 piksel
Parlaklik Degeri 2: 0 piksel
Parlaklik Degeri 3: 0 piksel
Parlaklik Degeri 4: 0 piksel
Parlaklik Degeri 5: 0 piksel
Parlaklik Degeri 6: 0 piksel
Parlaklik Degeri 7: 0 piksel
Parlaklik Degeri 8: 0 piksel
Parlaklik Degeri 9: 0 piksel
Parlaklik Degeri 10: 0 piksel
Parlaklik Degeri 11: 0 piksel
Parlaklik Degeri 12: 4 piksel
Parlaklik Degeri 13: 52 piksel
Parlaklik Degeri 14: 100 piksel
Parlaklik Degeri 15: 201 piksel
Parlaklik Degeri 16: 324 piksel
Parlaklik Degeri 17: 438 piksel
Parlaklik Degeri 18: 643 piksel
Parlaklik Degeri 19: 635 piksel
Parlaklik Degeri 20: 814 piksel
Parlaklik Degeri 21: 1024 piksel
Parlaklik Degeri 22: 1153 piksel
Parlaklik Degeri 23: 1308 piksel
Parlaklik Degeri 24: 1571 piksel
Parlaklik Degeri 25: 1757 piksel
Parlaklik Degeri 26: 1671 piksel
Parlaklik Degeri 27: 1814 piksel
Parlaklik Degeri 28: 1897 piksel
Parlaklik Degeri 29: 1889 piksel
```

Konsol Ekranı

KONTROL SONUCU

KISIM	Uygulamaların Sentezi	Yaprak No:12
YAPILAN İŞ	Genel Değerlendirme	Tarih:25/07/2023
<p>OpenCV ile gerçekleştirdiğim uygulamaların tamamını bir araya getirerek parçadan bütüne doğru ilerlemeye başladım. Parçadan bütüne doğru ilerleyerek hem problemleri daha iyi algılayıp hem de problemlerin çözümüne ulaşma yolunda kolaylık sağladığımı fark ettim.</p> <p>Parçadan bütüne doğru ilerleyerek OpenCV ile yapmış olduğum uygulamaları sentezleyerek proje iskeletini oluşturmaya başladım. Aşağıda iskeletini oluşturmaya başladığım projenin kaynak kodları mevcuttur.</p> <pre> #include <opencv2/opencv.hpp> #include <vector> #include <algorithm> using namespace std; using namespace cv; void save_frames(cv::VideoCapture& cap, int frameSayisi, const std::string& klasorAdi) { // Kayıt için klasörü oluştur cv::utils::fs::createDirectory(klasorAdi); // Frame kaydetme döngüsü for (int i = 0; i < frameSayisi; i++) { cv::Mat frame; cap >> frame; if (frame.empty()) { std::cerr << "Goruntu alinamadi!" << std::endl; break; } // Frame'i diske kaydet std::stringstream dosyaAdi; dosyaAdi << klasorAdi << "/frame_" << i << ".jpg"; cv::imwrite(dosyaAdi.str(), frame); } std::cout << frameSayisi << " frame kaydedildi." << std::endl; } void calculate_intensity_histogram(Mat mat,vector<int>& source) { int histSize = 256; source.resize(histSize, 0); for (int y = 0; y < mat.rows; ++y) { for (int x = 0; x < mat.cols; ++x) { int pixelValue = static_cast<int>(mat.at<uchar>(y, x)); source[pixelValue]++; } } } int main() { // Kamera nesnesini oluşturma cv::VideoCapture cap(0); // Kamera açılıp açılmadığını kontrol etme if (!cap.isOpened()) { std::cerr << "Kamera acilamadi!" << std::endl; return -1; } int frameSayisi; std::cout << "Kac frame kaydedilecek? "; std::cin >> frameSayisi; std::string klasorAdi = "test_images"; save_frames(cap, frameSayisi, klasorAdi); </pre>		
<p>KONTROL SONUCU </p>		

KISIM	Uygulamaların Sentezi	Yaprak No:13
YAPILAN İŞ	Genel Değerlendirme	Tarih:25/07/2023
<pre>// Kamera nesnesini serbest bırak cap.release(); // Pencereleeri kapat cv::destroyAllWindows(); Mat frame = imread("test_images/erase01.jpg"); Rect2d roi = selectROI("Select Area", frame); destroyAllWindows(); if (roi.width == 0 roi.height == 0) return -1; Mat roiFrame = frame(roi); cvtColor(roiFrame, roiFrame, COLOR_BGR2GRAY); vector<int> histogram; calculate_intensity_histogram(roi_frame, histogram); return 0; }</pre> <p>Yukarda eklemiş olduğum kaynak kod da öncelikle Web-Cam üzerinden görüntü alıp veri setimi oluşturuyorum. Daha sonra oluşturmuş olduğum veri setinin ilk frame'i okuyup takip etmek istediğim alanı resim üzerinden seçip histogramını hesaplıyorum.</p>		
KONTROL SONUCU 		

KISIM	Histogram Karşılaştırılması Yöntemleri	Yaprak No:14
YAPILAN İŞ	Histogram Karşılaştırılması Üzerine Araştırma	Tarih:26/07/2023

Histogram karşılaştırma yöntemleri, iki veya daha fazla veri kümesinin benzerliğini veya farklılığını değerlendirmek için kullanılan önemli bir istatistiksel araçtır. Bu yöntemlerden bazıları Kosinüs Benzerliği(Cosine Similarity), Kesişim(Intersection), Öklid Uzaklığı(Euclidean Distance) ve Korelasyon(Correlation)'dur.

1. Kosinüs Benzerliği (Cosine Similarity)

Kosinüs Benzerliği, iki vektör arasındaki açıyı kullanarak benzerliği ölçen bir metriktir. Histogramlar bir vektör olarak düşünülür ve bu metrik, bu iki vektörün kosinüs benzerliği hesaplayarak çalışır. Kosinüs benzerliği 0 ile 1 arasında bir değer alır. 0, iki histogram arasında hiçbir benzerlik olmadığını, 1 ise tam bir benzerliği temsil eder.

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}}$$

2. Kesişim (Intersection)

Kesişim yöntemi, iki histogram arasındaki ortak alanı ölçer. İki histogramın piksellerinin ortak olduğu alanın büyüklüğüne dayanarak hesaplanır. Bu yöntem, iki histogramın benzerliğini yüzde cinsinden ifade eder. Daha yüksek bir kesişim değeri, iki histogramın daha fazla benzer olduğunu gösterir.

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

3. Öklid Uzaklığı (Euclidean Distance)

Öklidyen uzaklık, iki histogram arasındaki farklılık miktarını ölçer. Histogramları vektör olarak ele alır ve bu vektörler arasındaki öklidyen uzaklığı hesaplar. Daha küçük bir öklidyen uzaklık, iki histogramın daha benzer olduğunu gösterir.

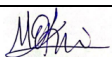
$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

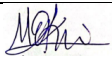
4. Korelasyon (Correlation)

Korelasyon yöntemi, iki histogram arasındaki ilişkiyi ölçer. Korelasyon katsayısı, iki histogram arasındaki ilişkinin yönünü ve gücünü gösterir. Pozitif bir korelasyon, iki histogramın benzer olduğunu, negatif bir korelasyon ise farklı olduğunu gösterebilir.

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

KONTROL SONUCU 

KISIM	Histogram Karşılaştırılması Yöntemleri	Yaprak No:15
YAPILAN İŞ	Kosinüs Benzerliği ve Kesişim Yöntemlerinin Kodlarını Yazma	Tarih:27/07/2023
<p>Kosinüs Benzerliği:</p> <pre> double dot_product(const std::vector<double>& v1, const std::vector<double>& v2) { double result = 0.0; for (size_t i = 0; i < v1.size(); ++i) { result += v1[i] * v2[i]; } return result; } double norm(const std::vector<double>& v) { double sumOfSquares = 0.0; for (double val : v) { sumOfSquares += val * val; } return std::sqrt(sumOfSquares); } double cosine_similarity(const std::vector<double>& v1, const std::vector<double>& v2) { double dot = dot_product(v1, v2); double normV1 = norm(v1); double normV2 = norm(v2); if (normV1 == 0 normV2 == 0) { return 0.0; } return dot / (normV1 * normV2); } </pre> <p>Kesişim:</p> <pre> double intersection(const std::vector<double>& hist1, const std::vector<double>& hist2) { if (hist1.size() != hist2.size() hist1.size() == 0) { return 0.0; } double intersection = 0.0; for (size_t i = 0; i < hist1.size(); i++) { intersection += std::min(hist1[i], hist2[i]); } return intersection; } </pre>		
KONTROL SONUCU 		

KISIM	Histogram Karşılaştırılması Yöntemleri	Yaprak No:16
YAPILAN İŞ	Öklit Uzaklığı ve Korelasyon Yöntemlerinin Kodlarını Yazma	Tarih:28/07/2023
<p>Öklit Uzaklığı:</p> <pre> double euclidean_distance(const std::vector<double>& vec1, const std::vector<double>& vec2) { if (vec1.size() != vec2.size()) { throw std::runtime_error("Vektör boyutları eşleşmiyor."); } double sum = 0.0; for (size_t i = 0; i < vec1.size(); i++) { double diff = vec1[i] - vec2[i]; sum += diff * diff; } return std::sqrt(sum); } double mean(const std::vector<double>& vec) { double sum = std::accumulate(vec.begin(), vec.end(), 0.0); return sum / vec.size(); } </pre> <p>Korelasyon:</p> <pre> double correlation(const std::vector<double>& vec1, const std::vector<double>& vec2) { if (vec1.size() != vec2.size() vec1.size() == 0) { throw std::invalid_argument("Vektör boyutları eşleşmiyor."); } double mean1 = mean(vec1); double mean2 = mean(vec2); double numerator = 0.0; double denominator1 = 0.0; double denominator2 = 0.0; for (size_t i = 0; i < vec1.size(); i++) { double diff1 = vec1[i] - mean1; double diff2 = vec2[i] - mean2; numerator += diff1 * diff2; denominator1 += diff1 * diff1; denominator2 += diff2 * diff2; } double correlation = numerator / (std::sqrt(denominator1) * std::sqrt(denominator2)); return correlation; } </pre>		
KONTROL SONUCU 		

KISIM	Histogram Karşılaştırılması Yöntemleri	Yaprak No:17
YAPILAN İŞ	Histogram Karşılaştırma Yöntemlerini Test Etme	Tarih:31/07/2023

Kodlarını yazmış olduğum Histogram Karşılaştırma Yöntemlerini test etmek için OpenCV'nin dökümantasyonlarında bulunan Histogram Comparison(Histogram Karşılaştırma) kısımda kullanılan test resimlerini test amacıyla kendi yazdığım kodları denemek amacıyla kullanmaya karar verdim.

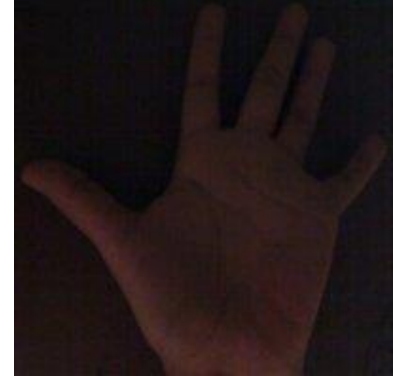
Test resimlerinde 3 adet avuç içi görüntüleri mevcuttur. Bu 3 avuç resimlerden bir tanesi ana resim, diğer 2 tanesi test resim yani ana resimle histogram kıyaslanması yapılacak resimlerdir.



Ana Resim



Test_1



Test_2

Bu 3 resmi yazmış olduğum histogram karşılaştırma yöntemleriyle test etmek amacıyla aşağıdaki kod satırlarını yazdım.

```
int main() {

    Mat base_image = imread("base_image.jpg", IMREAD_GRAYSCALE);
    vector<double> base_histogram;

    calculate_intensity_histogram(base_image, base_histogram);

    Mat test_image_1 = imread("test_image_1.jpg", IMREAD_GRAYSCALE);
    vector<double> test_1_histogram;

    calculate_intensity_histogram(test_image_1, test_1_histogram);

    Mat test_image_2 = imread("test_image_2.jpg", IMREAD_GRAYSCALE);
    vector<double> test_2_histogram;

    calculate_intensity_histogram(test_image_2, test_2_histogram);

    double cosine_1, intersection_1, eucliden_1, correlation_1;

    cosine_1 = cosine_similarity(base_histogram, base_histogram);
    intersection_1 = intersection(base_histogram, base_histogram);
    eucliden_1 = euclidean_distance(base_histogram, base_histogram);
    correlation_1 = correlation(base_histogram, base_histogram);
}
```

KONTROL SONUCU 

KISIM	Histogram Karşılaştırılması Yöntemleri	Yaprak No:18
YAPILAN İŞ	Histogram Karşılaştırma Yöntemlerini Test Etme	Tarih:31/07/2023

```

double cosine_2, intersection_2, eucliden_2, correlation_2;

cosine_2 = cosine_similarity(base_histogram, test_1_histogram);
intersection_2 = intersection(base_histogram, test_1_histogram);
eucliden_2 = euclidean_distance(base_histogram, test_1_histogram);
correlation_2 = correlation(base_histogram, test_1_histogram);

double cosine_3, intersection_3, eucliden_3, correlation_3;

cosine_3 = cosine_similarity(base_histogram, test_2_histogram);
intersection_3 = intersection(base_histogram, test_2_histogram);
eucliden_3 = euclidean_distance(base_histogram, test_2_histogram);
correlation_3 = correlation(base_histogram, test_2_histogram);

cout << "Method          " << "          Base-Base          " << "          Base-Test(1)          " <<
"          Base - Test(2)          " << endl;
cout << "Cosine:          " << "          /          " << cosine_1 << "          /          " << "          /          " <<
cosine_2 << "          /          " << "          /          " << cosine_3 << endl;
cout << "Intersection:          " << "          /          " << intersection_1 << "          /          " << "          /          " <<
<< intersection_2 << "          /          " << "          /          " << intersection_3 << endl;
cout << "Euclidean:          " << "          /          " << eucliden_1 << "          /          " << "          /          " <<
eucliden_2 << "          /          " << "          /          " << eucliden_3 << endl;
cout << "Correlation:          " << "          /          " << correlation_1 << "          /          " << "          /          " <<
correlation_2 << "          /          " << "          /          " << correlation_3 << endl;

return 0;
}

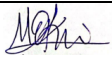
```

Test Sonuçları:

Method	Base-Base	Base-Test(1)	Base - Test(2)
Cosine:	/ 1 /	/ 0.70987 /	/ 0.00931206 /
Intersection:	/ 40000 /	/ 23014 /	/ 388 /
Euclidean:	/ 0 /	/ 2927.36 /	/ 7949.32 /
Correlation:	/ 1 /	/ 0.518733 /	/ -0.259422 /

Test sonuçlarını incelediğimde başlangıç olarak ana resmi kendisiyle yöntemlerle kıyasladım. Daha sonra test_1 ve test_2 resimleri ile kıyasladım. Sonuçlara göre ana resmin kendisiyle kıyas sonuçlarında Kosinüs benzerliği ve Korelasyon da sonucun 1 olduğunu yani benzerlik oranının 1 olduğunu görüyoruz. Öklit uzaklığında ise sonucun 0 yani ana resmin kendisiyle histogramlarının uzaklıklarının 0 olduğunu fark ediyoruz. Kesişim(Intersection) da ise sonuc değeri ne kadar yüksek ise benzerliğin o kadar fazla olduğunu söyleyebiliriz. Diğer resimlerle yani test_1 ve test_2 sonuçlarına baktığımızda test_1 resminin sonuçları ana resme daha yakın değerler olduğunu gözlemleyebiliyoruz. Test_2 ise daha karanlık bir resim olduğu için test_1'e göre daha az benzerlik değerleri elde edildiğini gözlemliyoruz.

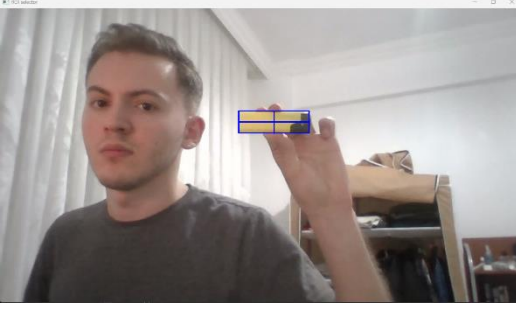
KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:19
YAPILAN İŞ	Seçili Alanı Manuel Olarak Kaydırma ve Histogram Karşılaştırmasını Uygulama	Tarih:01/08/2023
<p>Şimdiye kadar yapmış olduğum araştırmalar, OpenCV ile gerçekleştirdiğim uygulamalar ile birlikte yavaş yavaş projemin ana hatlarını oluşturmaya başladım. En son yaptığım Histogram Karşılaştırma yöntemlerini Object Tracking’de kullanmak amacıyla daha önce Web-Cam üzerinden oluşturmuş olduğum data setimin üzerinde denemeye çalıştım. Başlangıç olarak data setimin ilk frame’ini OpenCV ile açarak frame üzerinde takip etmek istediğim nesneyi veya alanı ROI ile seçip farklı bir Mat objesine aktarıyorum. Seçtiğim alanı öncelikle gri-seviyeye(grayscale) formatına dönüştürüp daha sonra gri-seviyeye dönüştürdüğüm seçili alanın histogramını hesaplıyorum.</p> <p>Data setimde bulunan frame sayısı kadar for döngüsü oluşturun. For döngüsünde her iterasyonda diskten sırayla data setteki frame’leri okuyorum. İlk frame üzerinden oluşturmuş olduğum ROI alanını seçip gri-seviyeye dönüştürüp, histogramını hesaplıyorum. Histogram hesaplamasını yaptıktan sonra ilk seçtiğim alanla diskten okuduğum frame üzerindeki alanların histogram kıyaslamasını korelasyon yöntemi ile hesaplayıp double türü değişken olan similarity adlı değerde tutuyorum. Eğer similarity değeri kendim elle verdiğim eşik değeri yani 0.5 den büyük ise bunun anlamı iki alanın benzerliği yakınsa seçili alanı yani ROI’yi x ekseninde 20 birim ilerletiyorum ve iterasyonlar frame’ler bitene kadar ilerlemeye devam ediyor. İlgili senaryonun kodları aşağıdaki gibidir.</p> <pre> int main() { Mat image = imread("test_images/frame1.jpg"); Rect roi = selectROI(image); Mat selected_area = image(roi); cvtColor(selected_area, selected_area, COLOR_BGR2GRAY); vector<double> selected_area_histogram; calculate_intensity_histogram(selected_area, selected_area_histogram); for (int i = 1; i <= 10; i++) { Mat temp = imread("test_images/frame" + to_string(i) + ".jpg"); Mat temp_selected_area = temp(roi); cvtColor(temp_selected_area, temp_selected_area, COLOR_BGR2GRAY); vector<double> temp_selected_area_histogram; calculate_intensity_histogram(temp_selected_area, temp_selected_area_histogram); double similarity = correlation(selected_area_histogram, temp_selected_area_histogram); cout << to_string(i) + ". Frame: " << similarity << endl; if (similarity > 0.5) { roi.x += 20; } rectangle(temp, roi, cv::Scalar(0, 255, 0), 2); imshow("Frame" + to_string(i), temp); waitKey(0); } return 0; } </pre>		
KONTROL SONUCU 		

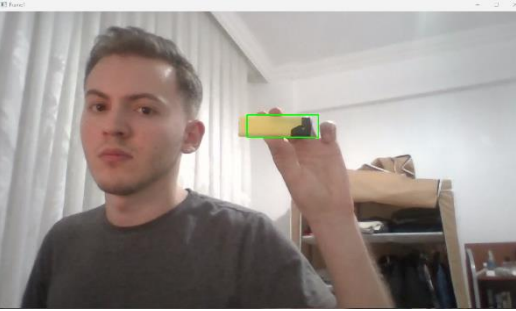
KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:20
YAPILAN İŞ	Program Testi ve Genel Değerlendirme	Tarih:02/08/2023

Seçili alanı manuel olarak kaydırarak histogram karşılaştırma yöntemlerinden korelasyon işlemini uygulayarak ufak bir test gerçekleştirdim. Test sonuçlarıma göre yazdığım korelasyon yöntemi kodunun doğru çalıştığını gözlemledim.

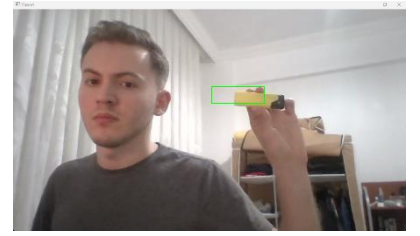
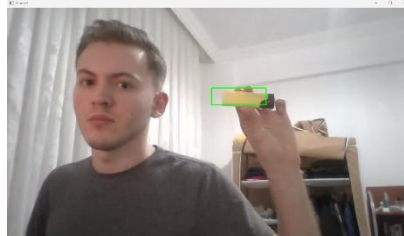
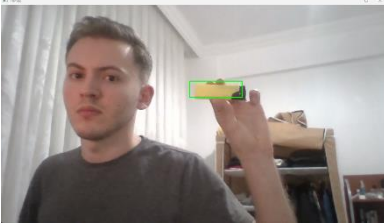
Test sonuçlarına dair bazı değerler ve görüntüler aşağıda mevcuttur.



Başlangıç olarak diskten okuduğum resimde takip etmek istediğim alanı ROI ile seçiyorum. Seçtiğim alanı gri seviyeye çevirip histogramını hesaplıyorum.



İlk frame de histogram karşılaştırması yaparak ilk seçili alana yakın bir değerde ROI'yi işaretliyorum. Burda karşılaştırma sonucunda çıkan değeri elle belirlediğim 0.5 eşik değeri ile kontrol ediyorum. Yani değer 0.5 den yüksek ise ROI'yi güncelleyip işaretliyorum.



2., 3. ve 4. Frameler de benzerlik değeri düşmeye başlıyor ve artık sağlıklı sonuçlar elde edememeye başlıyorum.

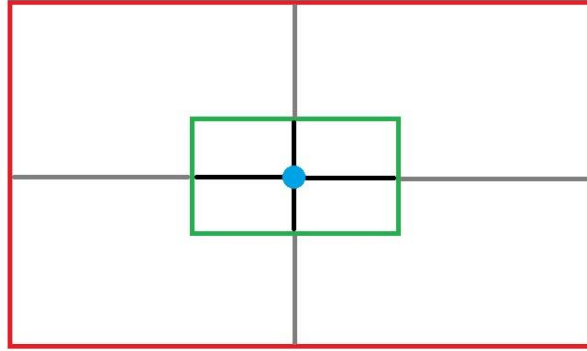
```
1. Frame: 1
2. Frame: 0.779772
3. Frame: 0.625724
4. Frame: 0.392704
```

Konsol üzerinde yazdırdığım benzerlik değerlerini incelediğimde değerlerin yavaş yavaş düştüğünü yani benzerliği yakalayamadığımı fark ediyorum. Manuel olarak ROI'yi hareket ettirmektense seçili alanın etrafında tarama yaparak daha optimize ve doğruluğu yüksek bir sonuçlar elde edeceğime kanaat getirdim.

KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:21
YAPILAN İŞ	Seçili Alanın Etrafında Search Alanı Oluşturma	Tarih:03/08/2023

Seçili alanı elle girdiğim değerle x ekseni veya y ekseninde manuel olarak kaydırarak takip edilecek nesne veya alanı yakalamada varsayımsal olarak kısmen de olsa başarılı bir sonuç elde ettim. Şimdi ise doğruluğu daha yüksek bir yöntemi oluşturmaya çalıştım. Yöntemden bahsedecek olursam seçili alan genellikle dikdörtgen veya kare şeklinde oluyor. Dikdörtgen veya kare şeklinde olan seçili alanın merkez noktalarını baz alarak seçili alanın genişliği ve yüksekliği nin 3 katı olacak şekilde bir search alanı oluşturmaya karar verdim. Bahsettiğim bu yöntemi aşağıdaki görselle açıklayacağım.



Yukardaki görselde bulunan yeşil renkli dikdörtgenimiz bizim ilk frame de takip etmek istediğimiz nesne veya alanı seçip aldığımız ROI oluyor. Mavi renkli dairemiz ise yeşil dikdörtgenin merkez noktalarını ifade ediyor. Kırmızı renkli dikdörtgenimiz ise search alanımız yani bizim takip etmek istediğimiz nesne veya alanı tarayacağımız alanı ifade ediyor. Kırmızı renkli dikdörtgenimiz yeşil renkli dikdörtgenin 3 katı büyüklüğünde bir alanı oluşturuyor.

Search alanımı oluşturduktan sonra artık tarama işlemine başlayacağım. Tarama işleminde search alanının sol üstünden başlayacak şekilde 1'er birim atlayarak yeşil dikdörtgen'in alanı kadar bölgeyi alıp histogram karşılaştırması yaparak en yüksek benzerlik değerini yakalamaya çalışacağım. En yüksek benzerlik değerini yakaladığımız yerde seçili alanı güncelleyerek takip etmek istediğimiz nesne veya alanı takip etmeye çalışacağım.

Tarama işlemini gerçekleştirirken iç içe iki for döngüsü kullanarak search alanının yüksekliği ve genişliği kadar itearasyonlarla ilerleyeceğim. Bu tarama işlemi görüntü işlemede kullanılan Convolution(Evrişim) yöntemine benzemektedir. Convolution yani Evrişim yöntemi genellikle çekirdek matrisin ana matris üzerinde gezdirilerek belirli işlemlerin yapılmasıdır. Bu işlemlere örnek verecek olursam, resimlere uygulanan filtreler diyebilirim.(Mean Filter, Gaussian Filter, Sobel Filter vs.)

KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:22
YAPILAN İŞ	Search Alanı ve Tarama Yönteminin Kodunu Yazma	Tarih:04/08/2023

Aşağıda yazmış olduğum kod bloğunda ilk frame de takip etmek istediğim nesne veya alanı ROI ile seçtikten sonra seçili nesne veya alanın etrafında tarama yapmak için ROI'nin x, y kordinatları ve yükseklik, genişlik bilgileri ile merkez noktasının x ve y noktalarını buluyorum. Daha sonra bulduğum merkez noktalarını kullanarak search alanının sınır bölgelerini belirliyorum. Tarama işleminde ise search alanının yüksekliği ve genişliği kadar dönecek 2 tane for döngüsü oluştuyorum. For döngüsünde her x ve y birimde ROI kadar bir alanı alıp histogramını hesaplayıp takip etmek istediğim nesne veya alanın histogramını kıyaslayarak en yüksek benzerlik noktasını bulmaya çalışıyorum. En sonunda en yüksek benzerlik yakaladığım koordinatta ROI'yi güncelleyip takip etmek istediğim nesne veya alanın yeni konumunu resim üzerinde işaretliyorum ve diske kaydediyorum.

```
for (int i = 1; i <= 4; i++) {

    Mat frame = imread("test_images/frame" + to_string(i) + ".jpg");

    int centerX = (roi.x + (roi.width / 2));
    int centerY = (roi.y + (roi.height / 2));

    int x_start = std::max(0, static_cast<int>(centerX) - (int)roi.width);
    int y_start = std::max(0, static_cast<int>(centerY) - (int)roi.height);

    int x_end = std::min(frame.cols, static_cast<int>(centerX) + (int)roi.width);
    int y_end = std::min(frame.rows, static_cast<int>(centerY) + (int)roi.height);

    cv::Point pt1(x_start, y_start);
    cv::Point pt2(x_end, y_end);
    rectangle(frame, pt1, pt2, cv::Scalar(0, 0, 255), 2);
    rectangle(frame, roi, cv::Scalar(255, 0, 0), 2);

    cv::Rect best_search_window = roi;
    double max_value = -1;

    for (int y = y_start; y < y_end && y < frame.rows; y++)
    {
        for (int x = x_start; x < x_end && x < frame.cols; x++)
        {
            cv::Rect search_window(x, y, roi.width, roi.height);
            if (x + search_window.width < frame.cols && y + search_window.height < frame.rows && x +
search_window.width < x_end && y + search_window.height < y_end) {

                Mat temp = frame(search_window);
                vector<double> temp_histogram;
                calculate_intensity_histogram(temp, temp_histogram);

                double similarity = correlation(selected_area_histogram, temp_histogram);

                if (similarity > max_value) {
                    max_value = similarity;
                    best_search_window = search_window;
                }
            }
        }
    }

    roi = best_search_window;
    rectangle(frame, roi, cv::Scalar(0, 255, 0), 2);

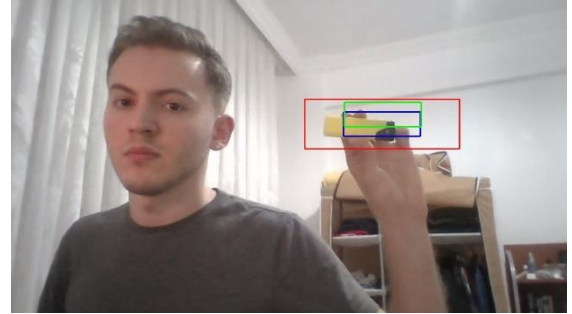
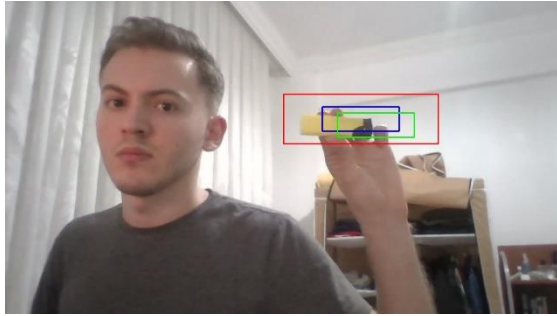
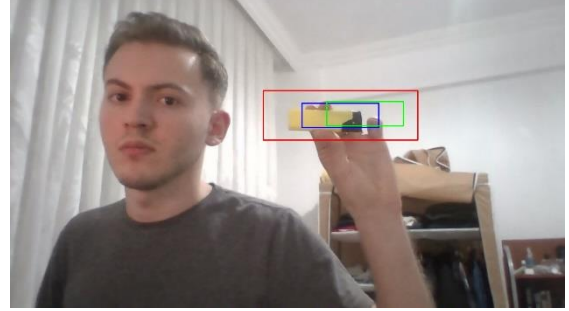
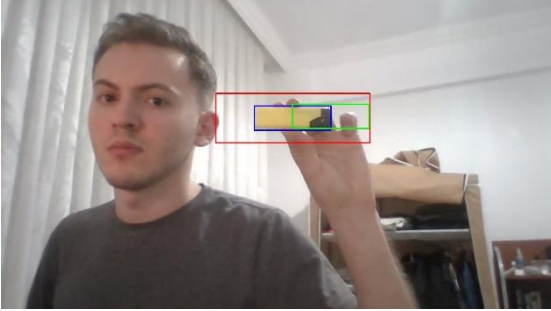
    cout << i << ". iterasyon tamamlandi." << endl;
    imwrite("tracking_object" + to_string(i) + ".jpg", frame);

}
```

KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:23
YAPILAN İŞ	Search Alanı ve Tarama Yöntemini Test Etme	Tarih:07/08/2023

Yazmış olduğum kodu Web-Cam üzerinden oluşturduğum data set ile test etmeye başladım. Data setimdeki framelere elimde sarı renkli bir silgi bulunuyor. Amacım her frame de hareket ettirdiğim elimde bulunan sarı silgiyi takip etmektir. Data setimden 4 adet frame işleyip, histogram karşılaştırma yöntemlerinde korelasyon yöntemini kullanarak ilk testimi gerçekleştirdim. Aşağıda test sonuçlarıma ait işlenmiş framelere ve elde ettiğim benzerlik oranları mevcuttur.



İşlenmiş Frameler

```
1. iterasyon tamamlandı. Benzerlik Oranı:0.573719
2. iterasyon tamamlandı. Benzerlik Oranı:0.589849
3. iterasyon tamamlandı. Benzerlik Oranı:0.527128
4. iterasyon tamamlandı. Benzerlik Oranı:0.543398
```

İşlenmiş Framelere Ait Benzerlik Oranları

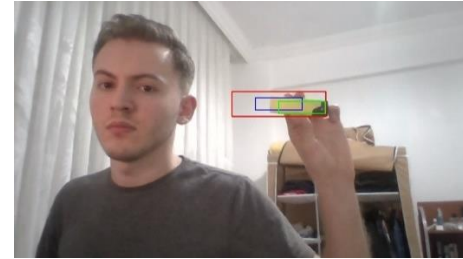
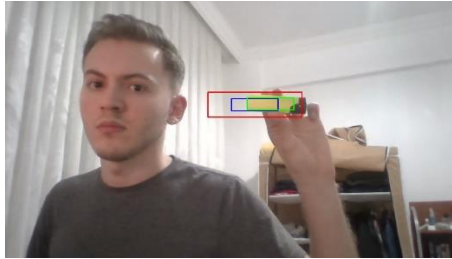
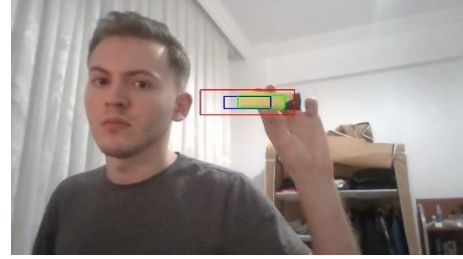
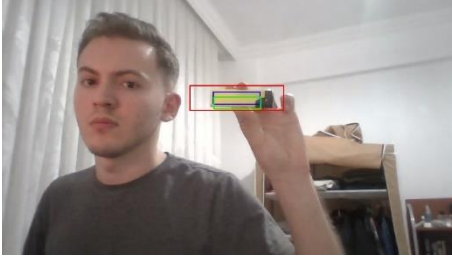
Yaptığım test sonucunu değerlendirecek olursam yaklaşık 0.5 ve 0.6 aralarında benzerlik değerleri yakalayarak nesneyi kısmen de olsa takip etmeye başladım. İşlenmiş görüntülerde bulunan kırmızı renkli dikdörtgen alan search alanı, mavi renkli dikdörtgen bir önceki ROI konumu, yeşil renkli dikdörtgen ise yakalamış olduğumuz en yüksek benzerlik oranına sahip konumun bulunduğu alandır. Testlerime diğer histogram karşılaştırma yöntemleriyle devam edip sonuçlarımı gözlemleyeceğim.

KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluřturma	Yaprak No:24
YAPILAN İř	Farklı Testler Gerçekleřtirme ve Güzlemleme	Tarih:08/08/2023

Bir önceki gün geliřtirmiř olduėum histogram tabanlı object tracking algortimasın da histogram karřılařtırma yöntemlerinden korelasyon yöntemini kullanarak testimi gerçekleřtirdim. Bugün ise diėer histogram karřılařtırma yöntemleriyle testlerimi gerçekleřtireceėim ve test sonuçlarımı gözlemleyeceėim.

Testime ilk olarak Kosinüs Benzerliėi yöntemiyle bařlıyorum.



İřlenmiř Frameler

```

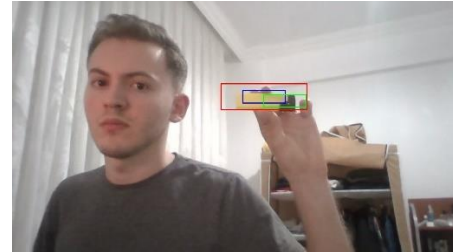
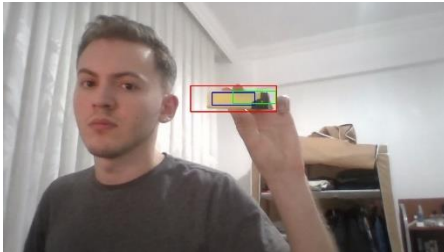
1. iterasyon tamamlandi. Benzerlik Orani:0.448937
2. iterasyon tamamlandi. Benzerlik Orani:0.466022
3. iterasyon tamamlandi. Benzerlik Orani:0.476556
4. iterasyon tamamlandi. Benzerlik Orani:0.424196

```

İřlenmiř Framelere Ait Benzerlik Oranları

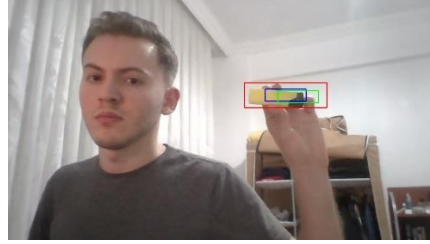
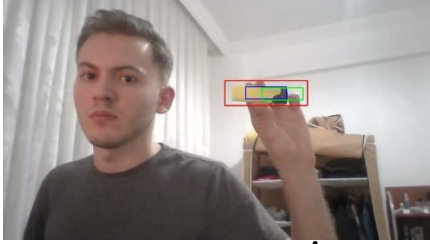
Kosinüs Benzerliėi yöntemiyle yapmıř olduėum test sonuçları yukardaki gibidir. İřlenmiř framelere baktıėımda Korelasyon yöntemine göre daha iyi sonuçlar aldıėımı söyleyebilirim.

İkinci testimi de Intersection(Keřiřim) yöntemi ile devam ediyorum.



KONTROL SONUCU *Mehmet*

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluřturma	Yaprak No:25
YAPILAN İř	Farklı Testler Gerekleřtirme ve Gzlemleme	Tarih:08/08/2023



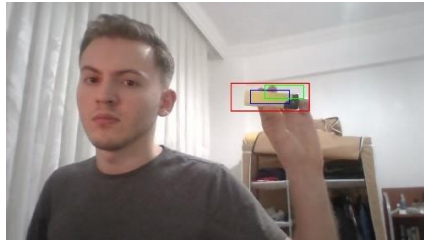
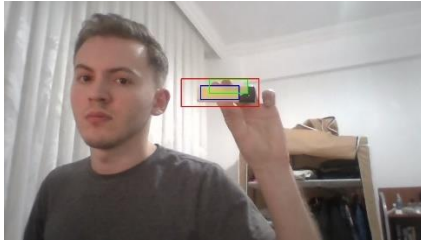
İřlenmiř Framelere Ait Grntler

```
1. iterasyon tamamlandi. Benzerlik Orani:1724
2. iterasyon tamamlandi. Benzerlik Orani:1994
3. iterasyon tamamlandi. Benzerlik Orani:2269
4. iterasyon tamamlandi. Benzerlik Orani:2154
```

İřlenmiř Framlere Ait Benzerlik Oranları

Intersection(Kesiřim) yntemiyle yapmıř olduėum test sonuları yukardaki gibidir. İřlenmiř framleri incelediėimde Kosins Benzerliėi yntemine gre daha kt sonular elde ettim. Almıř olduėum sonular Korelasyon yntemindeki sonulara benzer nitelikte.

nc ve son testimi de klit Uzaklıėı yntemi ile devam ediyorum.



İřlenmiř Framlere Ait Grntler

```
1. iterasyon tamamlandi. Benzerlik Orani:804.047
2. iterasyon tamamlandi. Benzerlik Orani:869.361
3. iterasyon tamamlandi. Benzerlik Orani:821.18
4. iterasyon tamamlandi. Benzerlik Orani:851.295
```

İřlenmiř Framlere Ait Benzerlik Oranları

klit Uzaklıėı yntemiyle yapmıř olduėum test sonuları yukardaki gibidir. Intersection yntemi gibi kt sonular elde ettim. řu ana kadar en iyi sonuları Kosins Benzerliėi yntemiyle elde edebildim.

KONTROL SONUCU *Maki*

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:26
YAPILAN İŞ	Farklı Data Setler Oluşturma	Tarih:09/08/2023

Geliştirmiş olduğum Histogram Tabanlı Object Tracking algoritmamı Web-Cam üzerinden oluşturmuş olduğum Data-Set ile farklı Histogram Karşılaştırma yöntemleriyle testler gerçekleştirdim. Bu testlerimin sonucunda sadece Kosinüs Benzerliği yönteminde güzel sonuçlar elde edebildim.

Algoritmamı farklı Data-Set'ler ile test etmemin daha iyi olacağını düşünüyorum. Farklı Data-Set'ler için internet ortamında Data-Setler üzerine araştırmalar gerçekleştirdim. Araştırma alanımı Object Tracking üzerinde kullanılan Data-Setler üzerine yoğunlaştırdım. Araştırmalarım sonucunda Kaggle internet sitesinde bulduğum **K. Scott Mader** adlı kullanıcının **Video Object Tracking** adlı gönderisinde bulunan dosya boyutu yaklaşık 6 GB olan Data-Set dosyasını indirdim.

Dosyayı incelediğimde yaklaşık 50 Bin 300 adet fotoğrafın bulunduğu bir Data-Set bulunuyor. İçerlerinde farklı senaryoların bulunduğu örneğin koşu yapan atlet, insansız hava aracından görüntülenen bir araç, basketbol müsabakası, buz pateni yapan sporcular gibi farklı ortamlardan, farklı parlaklık ve ışık koşullarından oluşan bir Data-Set havuzuna erişim sağladım.

Bu Data-Setler den kullanmak üzere iki adet Data-Set seçtim. Bunlardan bir tanesi hareket halinde bulunan kırmızı bir top ve basketbol müsabakasında oyuncuların sürekli hareket halinde olduğu bir Data-Set'dir. Bu iki Data-Set'den 50 adet frame'i Python dili kullanarak ufak bir script yazarak dosyaların ismini 0 dan başlayacak şekilde etiketlendirdim. Yazmış olduğum Python kodu aşağıdaki gibidir.

```
import os

def rename_images(folder_path):
    if not os.path.exists(folder_path):
        print("Belirtilen klasör bulunamadı.")
        return

    image_extensions = ['.jpg', '.jpeg', '.png', '.gif'] # İşlem yapılacak resim dosyası uzantıları
    image_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f)) and any(f.lower().endswith(ext) for ext in image_extensions)]

    if not image_files:
        print("Klasörde işlenecek resim dosyası bulunamadı.")
        return

    image_files.sort() # Dosya isimlerine göre sıralama

    counter = 1
    for old_filename in image_files:
        ext = os.path.splitext(old_filename)[1] # Dosya uzantısı
        new_filename = f"{counter:04d}{ext}" # Yeni dosya adı (örneğin: 0001.jpg)
        old_path = os.path.join(folder_path, old_filename)
        new_path = os.path.join(folder_path, new_filename)

        os.rename(old_path, new_path) # Dosya adını değiştir

        print(f"{old_filename} -> {new_filename}")

        counter += 1

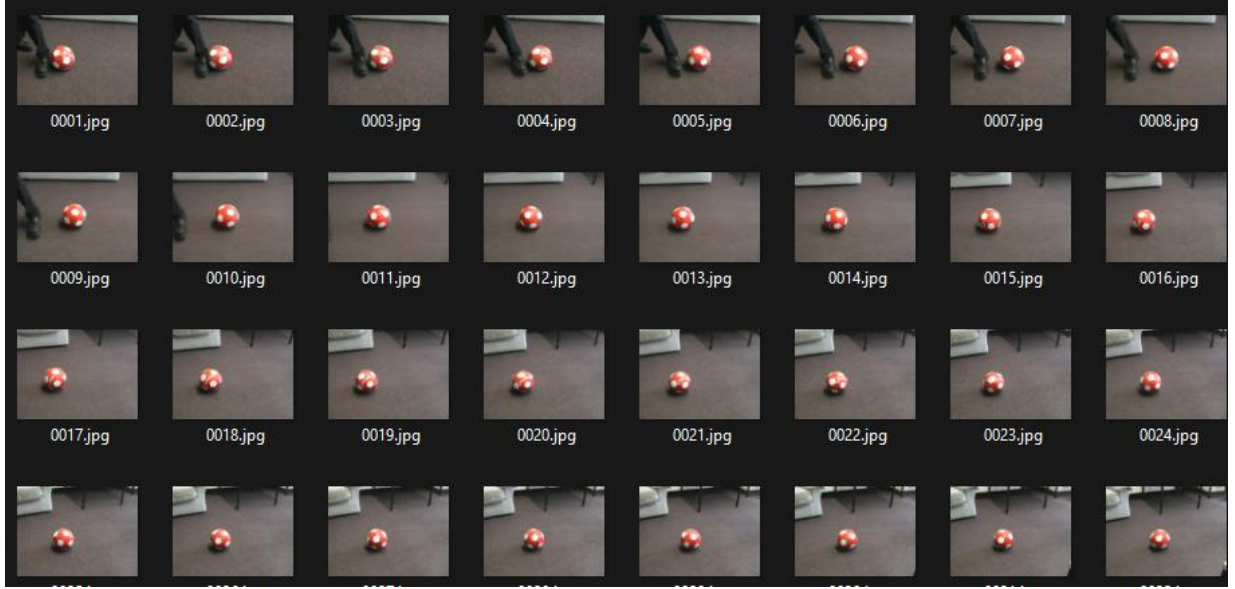
    print("İşlem tamamlandı.")

if __name__ == "__main__":
    target_folder = "C:\\Users\\Osman\\Documents\\Qt\\data_sets\\player" # İşlem yapılacak klasörün yolu
    rename_images(target_folder)
```

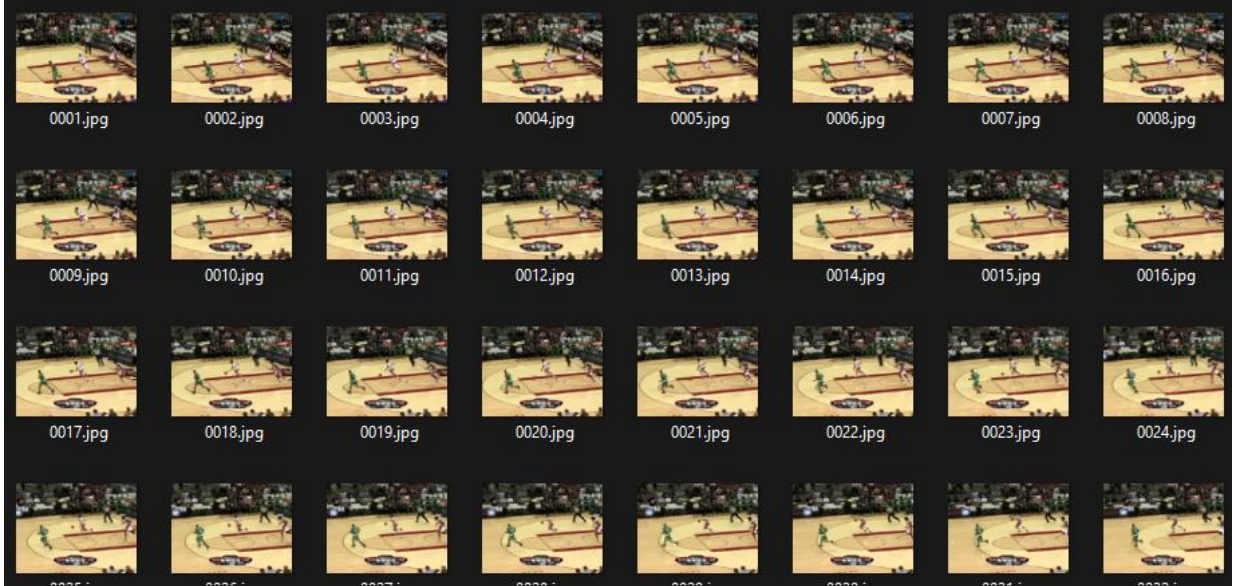
Oluşturmuş olduğum **Ball** ve **Player** adlı data setlerime ait etiketlendirme ve resimlere ait görüntüler aşağıdaki gibidir.

KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluřturma	Yaprak No:27
YAPILAN İř	Farklı Data Setler Oluřturma	Tarih:09/08/2023



Ball Data-Set



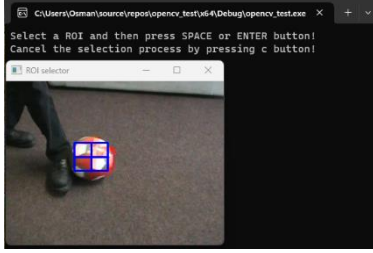
Player Data-Set

KONTROL SONUCU 

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:28
YAPILAN İŞ	Farklı Data Set Üzerinde Test Gerçekleştirme	Tarih:10/08/2023

Oluşturmuş olduğum yeni data-set'ler ile yazmış olduğum histogram tabanlı object tracking algoritmamı bu yeni data-setler ile testlerini gerçekleştireceğim.

İlk olarak Ball Data-Set isimli data-set'imle testlerime başlayacağım. Histogram karşılaştırma yöntemlerinden Kosinüs Benzerliği yöntemini kullanacağım.

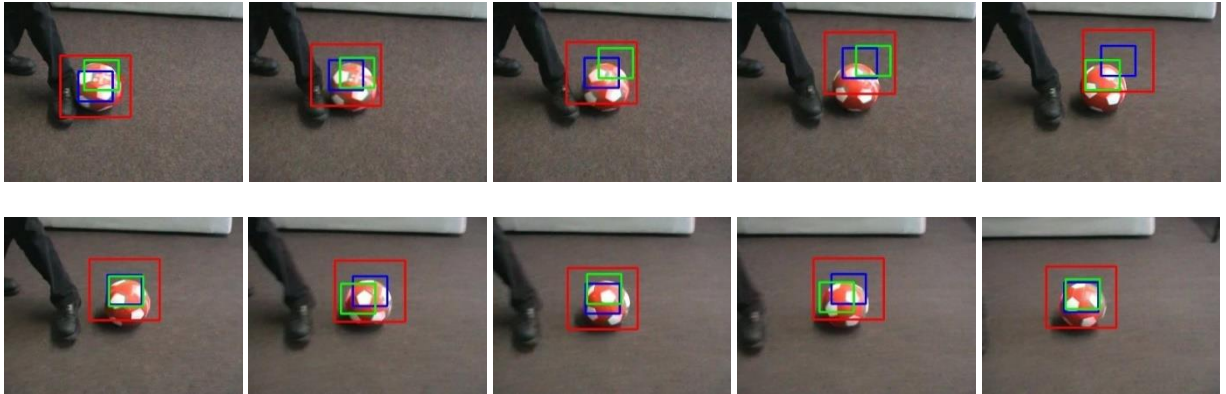


Başlangıç olarak takip etmek istediğim alanı ROI(Region of Interest) ile seçiyorum. Ardından diğer frame'leri işleme ve algoritmanın koşması için klavyeden SPACE tuşuyla ROI işleminden çıkış yapıyorum.

```
1. iterasyon tamamlandı. Benzerlik Orani:0.456786
2. iterasyon tamamlandı. Benzerlik Orani:0.438864
3. iterasyon tamamlandı. Benzerlik Orani:0.427956
4. iterasyon tamamlandı. Benzerlik Orani:0.411221
5. iterasyon tamamlandı. Benzerlik Orani:0.640941
6. iterasyon tamamlandı. Benzerlik Orani:0.456309
7. iterasyon tamamlandı. Benzerlik Orani:0.418183
8. iterasyon tamamlandı. Benzerlik Orani:0.440915
9. iterasyon tamamlandı. Benzerlik Orani:0.461109
10. iterasyon tamamlandı. Benzerlik Orani:0.561856
11. iterasyon tamamlandı. Benzerlik Orani:0.452494
12. iterasyon tamamlandı. Benzerlik Orani:0.434346
13. iterasyon tamamlandı. Benzerlik Orani:0.514017
14. iterasyon tamamlandı. Benzerlik Orani:0.445655
15. iterasyon tamamlandı. Benzerlik Orani:0.413846
16. iterasyon tamamlandı. Benzerlik Orani:0.559079
17. iterasyon tamamlandı. Benzerlik Orani:0.52821
18. iterasyon tamamlandı. Benzerlik Orani:0.432644
19. iterasyon tamamlandı. Benzerlik Orani:0.401616
20. iterasyon tamamlandı. Benzerlik Orani:0.547484
21. iterasyon tamamlandı. Benzerlik Orani:0.504946
22. iterasyon tamamlandı. Benzerlik Orani:0.411975
23. iterasyon tamamlandı. Benzerlik Orani:0.424851
24. iterasyon tamamlandı. Benzerlik Orani:0.51025
25. iterasyon tamamlandı. Benzerlik Orani:0.439071
26. iterasyon tamamlandı. Benzerlik Orani:0.447325
27. iterasyon tamamlandı. Benzerlik Orani:0.462947
28. iterasyon tamamlandı. Benzerlik Orani:0.444194
```

```
29. iterasyon tamamlandı. Benzerlik Orani:0.444605
30. iterasyon tamamlandı. Benzerlik Orani:0.472013
31. iterasyon tamamlandı. Benzerlik Orani:0.419883
32. iterasyon tamamlandı. Benzerlik Orani:0.482885
33. iterasyon tamamlandı. Benzerlik Orani:0.479437
34. iterasyon tamamlandı. Benzerlik Orani:0.46265
35. iterasyon tamamlandı. Benzerlik Orani:0.394631
36. iterasyon tamamlandı. Benzerlik Orani:0.442973
37. iterasyon tamamlandı. Benzerlik Orani:0.449228
38. iterasyon tamamlandı. Benzerlik Orani:0.469353
39. iterasyon tamamlandı. Benzerlik Orani:0.474613
40. iterasyon tamamlandı. Benzerlik Orani:0.540344
41. iterasyon tamamlandı. Benzerlik Orani:0.458646
42. iterasyon tamamlandı. Benzerlik Orani:0.44584
43. iterasyon tamamlandı. Benzerlik Orani:0.418929
44. iterasyon tamamlandı. Benzerlik Orani:0.423499
45. iterasyon tamamlandı. Benzerlik Orani:0.458769
46. iterasyon tamamlandı. Benzerlik Orani:0.451319
47. iterasyon tamamlandı. Benzerlik Orani:0.454618
48. iterasyon tamamlandı. Benzerlik Orani:0.454636
49. iterasyon tamamlandı. Benzerlik Orani:0.454563
50. iterasyon tamamlandı. Benzerlik Orani:0.419328
```

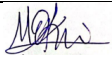
Data-Set den yaklaşık 50 adet frame işleyip algortimam sonlanıyor. İşlenen her frame sonucunca işlenmiş olan görüntüyü diske kaydediyorum. İşlenmiş olan frame'lere ait görüntüler aşağıdaki gibidir.



İşlenmiş Frame'ler

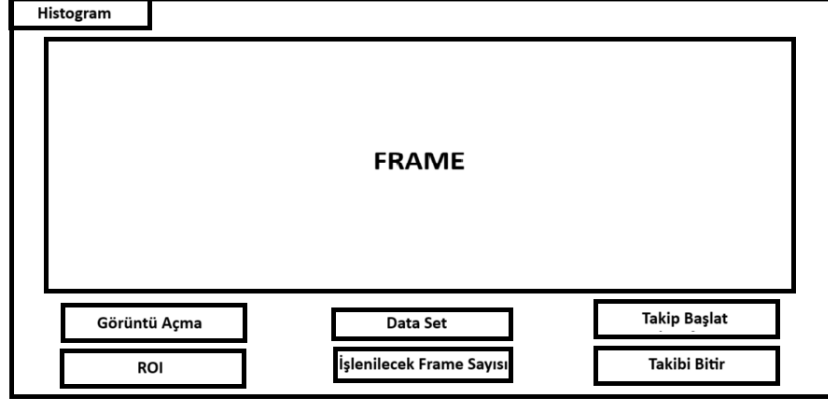
KONTROL SONUCU

KISIM	Histogram Tabanlı Object Tracking Algoritmasını Oluşturma	Yaprak No:29
YAPILAN İŞ	Staj Sorumlusu İle Genel Değerlendirme	Tarih:11/08/2023
<p>Staj sürecimin 4. Haftasının son iş gününde staj sorumlum Prof. Dr. Öğretim Görevlisi Murat EKİNCİ ile şu ana kadar yapmış olduğum çalışmaları, oluşturmuş olduğum algoritma ve yazmış olduğum kodları incelemek ve değerlendirme yapmak amacıyla toplantı gerçekleştirdik.</p> <p>Toplantı sonucunda staj sorumlum Prof. Dr. Murat EKİNCİ'nin geri dönüşleri, vermiş olduğu tavsiyeler ve projenin bir sonraki adımları hakkında bilgiler verdi.</p> <p>Projenin bundan sonraki adımlarında öncelikle kullanım kolaylığı açısından kullanıcı arayüzü tasarlamaya karar verdim. Histogram hesaplama sonucunda ilgili resmin histogram değerlerinin grafik üzerinde gösterilmesi, işlenmiş görüntülerin arayüz üzerinde dinamik olarak gösterilmesi ve işlenen her frame de elde edilen en yüksek benzerlik oranlarının bütün frame'ler işlendikten sonra grafiksel olarak arayüz üzerinde gösterilmesi gibi hem kullanım kolaylığı ve yapılan işlemlerin görsel olarak takibinin kolay olması için arayüz tasarlamaya karar verdim.</p> <p>Arayüz tasarımı sonrasında yazmış olduğum Histogram Tabanlı Object Tracking algoritmasının hızının yetersiz olduğundan dolayı algoritmamı hızlandırmak üzerine çalışmalar yapacağım. Şu an bir frame'in işlenmesi için geçen süre ortalama 3 – 4 saniye sürüyor. Bu süreyi daha aşağıya milisaniye değerlerine indirmeye çalışacağım. Ayrıca search alanının da yapmış olduğum tarama işleminde sırayla her pikseli gezmek yerine seçili alanın veya nesnenin etrafında dairesel spiral tarama yapmayı hedefliyorum. Bu sayede tarama işleminde harcamış olduğum sürenin ciddi şekilde düşeceğini düşünüyorum.</p> <p>Bu çalışmaları bitirince en son olarak eğer staj sürecim bitmemişse Search Alanını Threadlere bölerek tarama işlemini birden çok Thread ile gerçekleştirerek Microservice düzeyinde işlem yaparak bu alanda hem bilgi hem deneyim kazanmayı hedefliyorum.</p>		
KONTROL SONUCU 		

KISIM	Kullanıcı Arayüzü	Yaprak No:30
YAPILAN İŞ	Kullanıcı Arayüzü Üzerine Araştırma	Tarih:14/08/2023
<p>İlk olarak geliştirmekte olduğum projeyi C++ dili ile yazdığım için C++ ile yazılmış arayüz kütüphanelerine ve frameworklerini araştırmaya başladım. Staj sorumlum Prof. Dr. Murat EKİNCİ'nin tavsiye ettiği Visual Studio'da bulunan Form Application'ın üzerine araştırma yaptım. Visual Studio Form Application dan bahsedecek olursam;</p> <p>Visual Studio Form Application, Microsoft Visual Studio geliştirme ortamını kullanarak Windows tabanlı kullanıcı arayüzü (GUI) uygulamaları oluşturmanızı sağlayan bir proje şablonudur. Bu programlama, özellikle C++ veya C# gibi programlama dillerini kullanarak Windows uygulamalarını geliştirmeyi hedefleyen geliştiriciler için tasarlanmıştır.</p> <p>Daha önce Visual Studio Form Application ile geliştirme yapmadığım için ve ayrıca bilgim, deneyimim olmadığından Visual Studio Form Application dan vazgeçtim. Onun yerine daha önce bir çok arayüz geliştirdiğim ve deneyimli olduğum C++ diline ait Qt Framework'u kullanmaya karar verdim. Qt nedir ve neden Qt kullanacağımın nedenlerini açıklayacak olursam;</p> <p>Qt, kullanıcı arayüzlerinden ağ programlamaya ve veritabanı işlemlerine kadar birçok farklı uygulama alanında kullanılabilen bir yazılım geliştirme kütüphanesidir. İşte Qt hakkında bazı temel bilgiler:</p> <ol style="list-style-type: none">1. Çapraz Platform Desteği: Qt, Windows, macOS, Linux ve diğer birçok işletim sistemi üzerinde çalışabilen uygulamalar oluşturmanıza olanak tanır. Bu, uygulamanızı birden fazla platformda kullanılabilir hale getirmenizi kolaylaştırır.2. Zengin Kütüphane: Qt, bir dizi kullanışlı kütüphane içerir. Bu kütüphaneler, grafiklerden multimedya işlevlerine, veritabanı bağlantılarından ağ programlamasına kadar birçok farklı alanda yardımcı olur. <div data-bbox="635 1691 949 1921"></div>		
KONTROL SONUCU 		

KISIM	Kullanıcı Arayüzü	Yaprak No:31
YAPILAN İŞ	Kullanıcı Arayüzü Tasarlama	Tarih:15/08/2023

Tasarlamak istediğim Kullanıcı Arayüzü için öncelikle Paint ortamında grafiksel olarak basit bir arayüz tasarladım. Tasarlamış olduğum taslak aşağıdaki gibidir.



Arayüz Taslağı

Taslakta sol üstte bulunan histogram butonunda data setin ilk frame'nin gri seviye histogram dağılımını grafik olarak gösterilecek şekilde planladım. Orta kısımda FRAME adlı bölgede işlenen frame'lerin dinamik olarak her frame'i işlendikten sonra arayüz üzerinde gösterilecek olan kısım olarak planladım.

Sol altta bulunan Görüntü açma seçilen data setin ilk frame'ini gösterecek şekilde planladım. Altında bulunan ROI butonunda ise resimde takip edilecek alan veya nesnenin ROI(Region of Interest) ile seçileceği kısımdır.

Orta kısımda bulunan Data Set adlı kısımda hangi data setinin kullanılacağını kullanıcı tarafından seçilebilecek bir liste şeklinde olmasını planlıyorum. Altında bulunan İşlenecek Frame Sayısı adlı alanda ise ilgili data set den kaç adet frame'in işleneceği kullanıcı tarafından belirlenebilecektir.

Sağ tarafta ise kullanıcı tarafından seçilen data set, ROI'nin ayarlanması ve işlenecek frame sayısının belirlenmesinden sonra Takibi Başlat butonu ile frame'lerin işlenmesi ve takip algoritmasının başlamasını sağlayacak olan butondur. Altında bulunan Takibi Bitir butonu ise takip algoritmasının sonlandırılmasını sağlar.

Taslağı bitirdikten sonra Qt ortamına geçiş yapıp Qt projemi oluşturdum. Qt design ortamında .ui uzantılı Qt Form dosyası üzerinde arayüzümü oluşturdum. Oluşturmuş olduğum arayüz aşağıdaki gibidir.

KONTROL SONUCU 

KISIM	Kullanıcı Arayüzü	Yaprak No:32
YAPILAN İŞ	Kullanıcı Arayüzü Tasarlama	Tarih:15/08/2023

Histogram Type Here

Select Data Set:

Ball

Number Of Frames:

Open Image

Select ROI

Start Tracking

Stop Tracking

Qt Ortamında Tasarlanan Arayüz

KONTROL SONUCU

KISIM	Kullanıcı Arayüzü	Yaprak No:33
YAPILAN İŞ	Tasarlanan Arayüzün Kodlarını Yazma	Tarih:16/08/2023

Qt ortamında tasarlamış olduğum kullanıcı arayüzü'nün Backend(Arka Uç) tarafını yazmaya başladım. Öncelikle eklemiş olduğum butonların aktif bir şekilde çalışabilmesi için butonlara basıldığında hangi işlemlerin yapılacağını kodlarını yazdım. Open Image, Select ROI, Histogram ve Start Tracking butonlarının Backend kısımlarını oluşturdum. Yazmış olduğum kodlar aşağıdaki gibidir.

```
void MainWindow::on_roi_button_clicked()
{
    roi = selectROI("Select Area", image.mat);
    cv::destroyAllWindows();

    roi_image.mat = image.mat(roi);
    cvtColor(roi_image.mat, roi_image.mat, cv::COLOR_BGR2GRAY);
    calculate_intensity_histogram(roi_image);
    imshow("Selected Area", roi_image.mat);
}

void MainWindow::on_actionIntensity_Histogram_triggered()
{
    show_histogram(roi_image.histogram);
}

void MainWindow::on_open_image_button_clicked() {
    QString data_set_name = ui->data_set_combo_box->currentText().toLower();

    this->image.mat = cv::imread("../data_sets/" + data_set_name.toStdString() + "/0001.jpg");
    QImage img = QImage((uchar*)image.mat.data, image.mat.cols, image.mat.rows, QImage::Format_BGR888);
    QPixmap pixel = QPixmap::fromImage(img);
    ui->image_label->setPixmap(pixel);
    ui->image_label->setScaledContents(true);
}

void MainWindow::on_tracking_button_clicked()
{
    QString data_set_name = ui->data_set_combo_box->currentText().toLower();
    int number_of_frames = ui->frames_line_edit->text().toInt();
    if (number_of_frames == 0) {
        number_of_frames = images.size();
    }

    QVector<double> similarity_values;
    for(int i = 0; i < number_of_frames; i++) {

        QElapsedTimer timer;
        timer.start();

        cv::Mat frame = gray_images[i].mat.clone();
        cv::Mat color_frame = images[i].mat.clone();

        int centerX = (roi.x + (roi.width / 2));
        int centerY = (roi.y + (roi.height / 2));

        int x_start = std::max(0, static_cast<int>(centerX) - (int)roi.width);
        int y_start = std::max(0, static_cast<int>(centerY) - (int)roi.height);
    }
}
```

Backend Kodları

KONTROL SONUCU



KISIM	Kullanıcı Arayüzü	Yaprak No:34
YAPILAN İŞ	Arayüz Üzerinde Histogram Değerlerini Grafikleştirme	Tarih:17/08/2023

Nesne veya bir alanı takip etmek amacıyla seçtiğim data setlerin ilk frame'ini başarıyla arayüzde gösterdikten sonra seçmiş olduğum nesne veya alanı gri-seviye histogram değerlerini hesapladıktan sonra histogram dağılımını grafiksel olarak gözlemleme ve yorumlamanın kolay bir şekilde yapılabilmesi için arayüz üzerinde bar şeklinde grafik olarak göstermeyi planladım. Qt Framework'ünde grafiksel işlemlerin nasıl yapıldığı verilerin grafikte nasıl gösterildiğini öğrenmek için öncelikle araştırmaya koyuldum. Araştırmalarımnda Qt'nin resmi dökümantasyonundan yararlandım. Araştırmalarım sonucunda gri-seviye histogram dağılımını grafiğe dökmek amacıyla kodlamaya başladım. Yazmış olduğum kod aşağıdaki gibidir.

```
void show_histogram(std::vector<double> histogram)
{
    QBarSet *barSet = new QBarSet("Histogram Values");
    barSet->setColor(QColor(0,0,0));
    for (int i = 0; i < 256; i++) {
        *barSet << histogram[i];
    }

    QBarSeries *series = new QBarSeries();
    series->append(barSet);

    QChart *chart = new QChart();
    chart->addSeries(series);
    chart->setTitle("Histogram");
    chart->setAnimationOptions(QChart::SeriesAnimations);

    QValueAxis *axisX = new QValueAxis();
    axisX->setRange(0, 255);
    chart->addAxis(axisX, Qt::AlignBottom);
    series->attachAxis(axisX);

    QValueAxis *axisY = new QValueAxis();
    chart->addAxis(axisY, Qt::AlignLeft);
    series->attachAxis(axisY);

    QChartView *chartView = new QChartView(chart);
    chartView->setRenderHint(QPainter::Antialiasing);

    QWidget *window = new QWidget;
    window->setMinimumSize(1024,768);
    window->setWindowTitle("Intensity Histogram");
    QHBoxLayout *lay = new QHBoxLayout;
    lay->addWidget(chartView);
    window->setLayout(lay);
    window->show();
}
```

1. Histogram verileri, **std::vector<double>** türünde bir girdi olarak kabul edilir.
2. **QBarSet** ve **QBarSeries** oluşturulur. Bu, histogram verilerini çubuk grafik (bar chart) olarak temsil eder. Her bir çubuk, histogramın bir binini temsil eder.
3. **QChart** oluşturulur ve çubuk serisi bu grafik üzerine eklenir. Grafik başlığı "Histogram" olarak ayarlanır ve animasyon seçenekleri etkinleştirilir.
4. X eksen (bottom axis) ve Y eksen (left axis) oluşturulur. X eksen, histogramın değerlerini temsil eden bir aralığı (0 ila 255) kapsar.
5. X ve Y eksenleri grafikte ilişkilendirilir. X eksen alt tarafta, Y eksen ise sol tarafta görüntülenir.
6. **QChartView** oluşturulur ve grafik görünümünü pürüzsüz çizim (antialiasing) ile ayarlar.
7. Bir pencere (QWidget) oluşturulur ve grafik görünümü bu pencereye eklenir. Pencerenin boyutu 1024x768 olarak ayarlanır ve başlığı "Intensity Histogram" olarak ayarlanır.
8. Son olarak, pencere görüntülenir, kullanıcıya histogramı görsel olarak sunar.

Bu kod, histogram verilerini çubuk grafik ile görselleştirmek için bir Qt tabanlı arayüz oluşturur ve bu arayüzü bir pencerede görüntüler.

KONTROL SONUCU 

KISIM	Kullanıcı Arayüzü	Yaprak No:35
YAPILAN İŞ	Arayüz Üzerinde Benzerlik Değerlerini Grafikleştirme	Tarih:18/08/2023

Gri-seviye histogram dağılımını grafikleştirdikten sonra Tracking işlemi bittiğinde Tracking işlemi boyunca her frame de elde edilen en yüksek benzerlik oranı değerlerini grafiksel olarak gözlemlemek ve yorumlamak amacıyla kodunu yazmaya başladım. Yazmış olduğum kod aşağıdaki gibidir.

```

QLineSeries *series = new QLineSeries();
for (int j = 0; j < similarity_values.size(); ++j) {
    series->append(j + 1, similarity_values[j]);
}
QChart *chart = new QChart();
chart->addSeries(series);
chart->setTitle("Similarity Values Over Frames");
chart->createDefaultAxes();

QValueAxis *xAxis = qobject_cast<QValueAxis *>(chart->axes(Qt::Horizontal).back());
QValueAxis *yAxis = qobject_cast<QValueAxis *>(chart->axes(Qt::Vertical).back());
xAxis->setTitleText("Frame");
yAxis->setTitleText("Similarity");

QChartView *chartView = new QChartView(chart);
chartView->setRenderHint(QPainter::Antialiasing);

QWidget *window = new QWidget;
window->setMinimumSize(1024,768);
window->setWindowTitle("Similarity Values");
QHBoxLayout *lay = new QHBoxLayout;
lay->addWidget(chartView);
window->setLayout(lay);
window->show();

```

Bu C++ kodu, Qt kütüphanesini kullanarak benzerlik değerlerini (similarity_values) çizgi grafik (line chart) ile görüntülemek için kullanılır. İşte kodun ana adımları:

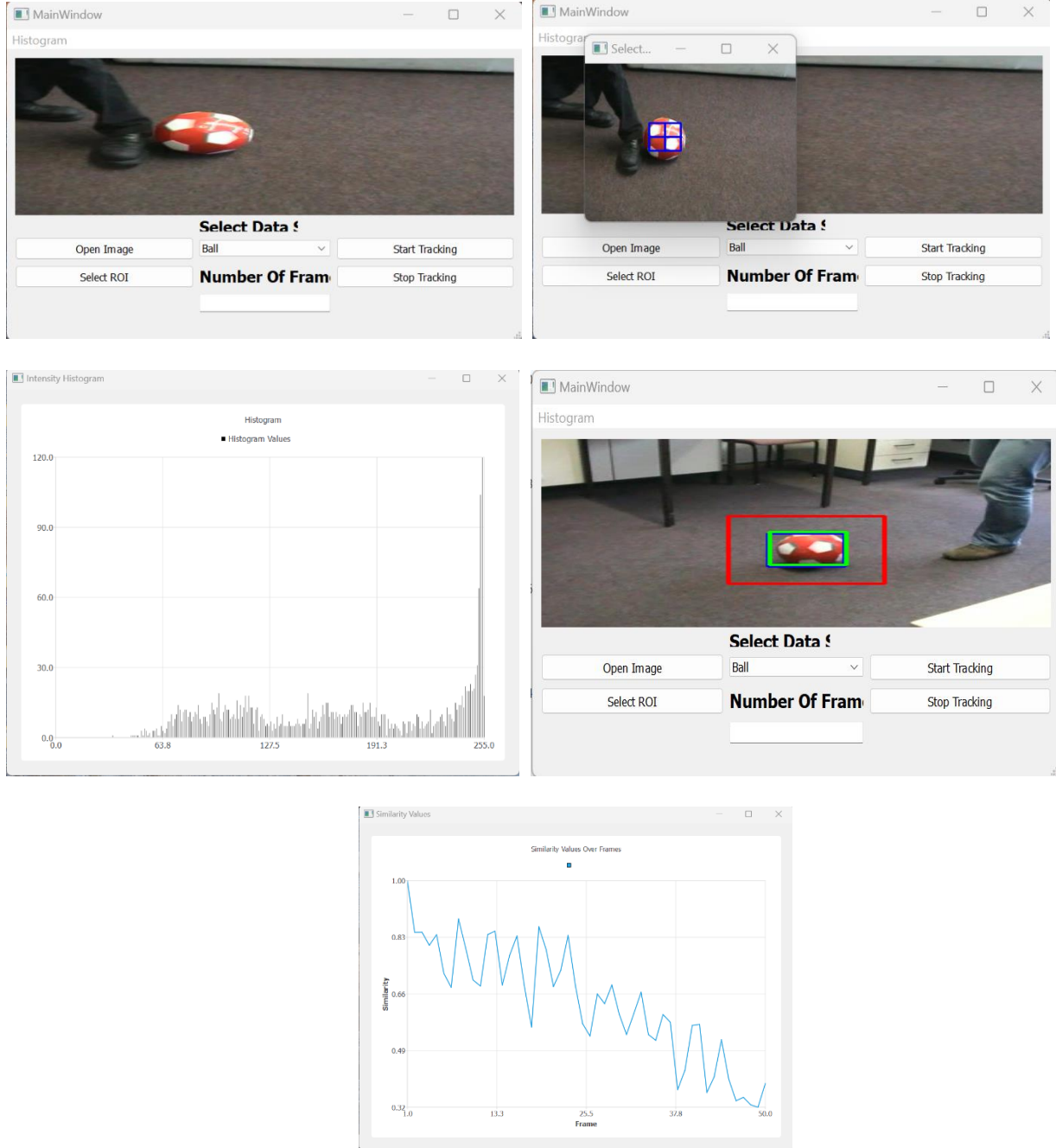
1. **QLineSeries** oluşturma: Benzerlik değerlerini çizgi serisine ekler. Her çizgi noktası, bir çerçeveyi ve bu çerçevenin benzerlik değerini temsil eder.
2. **QChart** oluşturma: Çizgi serisini bir grafikte görüntüler. Grafik başlığı "Similarity Values Over Frames" olarak ayarlanır.
3. Default eksenleri oluşturma: **createDefaultAxes** yöntemi kullanılarak varsayılan x ve y eksenleri eklenir.
4. X ve Y eksenlerinin başlıklarını ayarlama: X eksenini "Frame" (Çerçeve) olarak, Y eksenini "Similarity" (Benzerlik) olarak adlandırılır.
5. **QChartView** oluşturma: Grafik görünümünü oluşturur ve bu görünümü pürüzsüz çizim (antialiasing) ile ayarlar.
6. Bir pencere (QWidget) oluşturma: Benzerlik değerlerini içeren bir pencere oluşturur. Pencerenin boyutu 1024x768 olarak ayarlanır ve başlığı "Similarity Values" olarak ayarlanır.
7. Layout ve Widget ayarları: Bir **QHBoxLayout** oluşturur ve grafik görünümünü bu düzen içine ekler. Ardından bu düzeni pencereye yerleştirir ve pencereyi görüntüler.

Sonuç olarak, bu kod, benzerlik değerlerini çizgi grafik olarak görüntülemek için Qt tabanlı bir arayüz oluşturur ve bu arayüzü bir pencerede gösterir. Her çerçeve için benzerlik değerlerini görsel olarak temsil eder.

KONTROL SONUCU 

KISIM	Test ve Algoritma Optimizasyonu	Yaprak No:36
YAPILAN İŞ	Arayüz Testi ve Takip Algoritmasını Hızlandırma	Tarih: 21/08/2023

Tasarlamış olduğum kullanıcı arayüzü'nün kodlarını yazdıktan sonra arayüzü test etmeye başladım. Aşağıda arayüze ait görseller mevcuttur. Test görüntülerini daha iyi raporlanması amacıyla video görüntüsü olarak kaydedeceğim. Kaydetmiş olduğum video görüntüsünü YouTube platformu üzerine yükleyip video linkini aşağıya ekleyeceğim.



Öncelikle Ball Data-Set'imizi seçerek data setimize ait ilk frame'i Open Image butonu ile arayüzümüze getiriyoruz.

KONTROL SONUCU *Mehmet*

KISIM	Test ve Algoritma Optimizasyonu	Yaprak No:37
YAPILAN İŞ	Arayüz Testi ve Takip Algoritmasını Hızlandırma	Tarih: 21/08/2023

Daha sonra Select ROI butonu ile takip edeceğimiz nesneyi veya alanı belirliyoruz. Takip etmek istediğimiz nesne veya alanı belirleme işleminden sonra sol üstte bulunan Histogram butonu ile seçmiş olduğumuz nesne veya alanın gri-seviye histgoram dağılımına ait grafiği görsel olarak elde ediyoruz. Grafiği gözlemleme işleminden sonra Start Tracking butonu ile eğer Number Of Frames alanına bir değer girmemişsek data setteki bütün frameleri işleyerek Takip Algoritmasını çalıştırıyoruz eğer Number of Frames alanına bir değer girmişsek girdiğimiz değer kadar frame işlenip Takip Algoritmamız sona eriyor. Takip algoritmasına ait video görüntüsünün YouTube linki aşağıya ekliyorum.

Test 1: https://youtu.be/tl_pmtfuvhE

Test 2: <https://youtu.be/B2nbsR7Pr8k>

Arayüzümü test ettikten sonra sırada Takip Algoritmamı optimize etmeye geldi. Algoritmamın temel halinde bir frame işlerken geçen sürenin yaklaşık olarak 5 saniye olduğunu gözlemledim. Tahminsel olarak değilde gerçek değerlere ulaşmak amacıyla her frame işlenmeye başlamadan önce bir adet Timer(Zamanlayıcı) tuttum. Bu zamanlayıcı kullanmak için Qt Framework'üne ait QTimer kütüphanesinden yararlandım. Frame işlenmeye başladığı zaman timer tetikleniyor frame işlendiği zaman sonlanıyor. Timer değerlerini incelediğimde tahminime yakın değerler elde ettim frame başına 4 ila 5 saniye aralarında değişen zaman değerleri bulunuyor. Bu süreyi daha düşük değerlere düşürmek amacıyla kodumda ne gibi optimizasyon işlemleri yapabilirim, algoritmamda zaman harcayan işlemlerin hangileri olabileceğini düşünmeye başladım. Düşüncelerim ve incelemelerim sonucunda şu sonuçları elde ettim. Öncelikle en çok vakit yiyen işlemin her frame'i diskten teker teker okumak olduğunu yakaladım. Bunu hızlandırmak amacıyla başta hangi data setin seçildiği ve Open Image butonuna basıldığı zaman ilgili data setteki bütün framleri bir buffer'da tutmaya karar verdim. Yani kısacası ilgili data sete ait bütün resimleri belleğe okudum. Böylelikle disten tek tek okumak ile harcadığım zamanın aksine bellekten okuyarak harcanan sürenin aşağıya ineceğini tahmin ediyorum. Bunun yanı sıra bir diğer yakalamış olduğum sorunda okumuş olduğum her frame'i renkli okuduğum için resmi gri-seviye'ye çeviriyordum. Her frame'i okuduktan sonra gri-seviyeye çevirmenin de çok süre alacağını tahmin ettim. Bu sorunu da aşmak amacıyla başta data setteki bütün frameleri diskten okuyup buffer'da tutarken hem renkli halini hem de gri-seviyeye çevrilmiş hallerini buffer da tutmaya karar verdim. Bu şekilde frame işlenirken bufferda bulunan gri-seviye hallerinde işlemler yapılacak işlemler bittiğinde renkli hali arayüzde gösterilecek şekilde algoritmamı güncelledim. Bu optimizasyon işlemleri sonucunda algoritmamın baya bir hızlandığını QTimer'dan gelen verilerle doğruladım. Şu an 4-5 saniyelik değerlerden 0.013 saniye gibi değerlere kadar inmiş oldum. Optimizasyon sonucunda elde ettiğim zaman değerleri aşağıdaki gibidir.

```

Iterasyon 0 Sure: 0.014 sn Similarity 0.998366
Iterasyon 1 Sure: 0.013 sn Similarity 0.852707
Iterasyon 2 Sure: 0.014 sn Similarity 0.853017
Iterasyon 3 Sure: 0.013 sn Similarity 0.836058
Iterasyon 4 Sure: 0.013 sn Similarity 0.842125
Iterasyon 5 Sure: 0.013 sn Similarity 0.700129
Iterasyon 6 Sure: 0.013 sn Similarity 0.678311
Iterasyon 7 Sure: 0.015 sn Similarity 0.889219
Iterasyon 8 Sure: 0.014 sn Similarity 0.818518
Iterasyon 9 Sure: 0.014 sn Similarity 0.706349

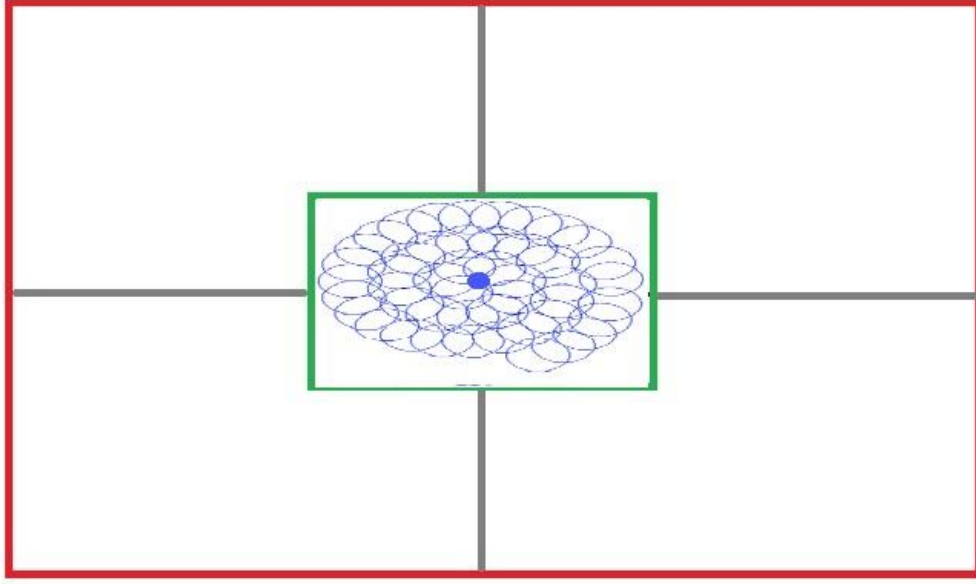
```

Optimizasyon Sonuçları

KONTROL SONUCU 

KISIM	Tarama Algoritması	Yaprak No:38
YAPILAN İŞ	Spiral Tarama Algoritmasını Oluşturma	Tarih: 22/08/2023

Arayüz testi ve takip algoritmasını optimize ettikten sonra şimdiki hedefim Search Alanın da yapmış olduğum tarama işlemini sol üstten başlayarak tek tek piksellerde gezmek yerine Search Alanı'nın merkez noktalarından başlayarak çapı birer birim artacak şekilde her çap artışında spiral şekilde tarama yaparak tarama işlemini ve tracking(takip) algoritmasını hızlandırmak. Bahsetmiş olduğum spiral taramayı açıklayacak olursam;



Search Alanı

Yukardaki görselde bulunan Search Alanı'nın merkez noktasından başlayacak şekilde çapı birer birim arttırarak her çap artışında 360 derece dönerek spiral bir tarama yaparak takip edilmesi istenen nesnenin veya alanın yakalanmasını hedefliyorum. Bu yöntemle şu anki takip algoritmasında yapmış olduğum tarama işleminin farkını ve spiral taramanın avantajını açıklayacak olursam;

Normal taramada Search Alanı'nın sol üst köşesinden başlayarak birer piksel ilerleyerek tarama işlemi yapıyordum bu tarama işleminin dezavantajı eğer takip edilecek nesne veya alanın bir önceki konuma yakın bir yerdeyse tarama işlemi sol üst köşeden başladığı için nesnenin veya alanın yakalanması için geçen süre bir hayli fazla olacaktır. Spiral Tarama ile nesnenin veya alanın yakalanması daha kolay olacaktır çünkü Search Alanının merkez noktasından dışarıya doğru tarama yapıldığı için normal taramaya göre daha hızlı bir tarama işlemi olacaktır. Yarınki staj günümde Spiral Tarama yönteminin kodunu yazmayı hedefliyorum.

KONTROL SONUCU 

KISIM	Tarama Algoritması	Yaprak No:39
YAPILAN İŞ	Spiral Tarama Algoritmasının Kodunu Yazma	Tarih: 23/08/2023

```

int centerX = (roi.x + (roi.width / 2));
int centerY = (roi.y + (roi.height / 2));

int x_start = std::max(0, static_cast<int>(centerX) - (int)roi.width);
int y_start = std::max(0, static_cast<int>(centerY) - (int)roi.height);

int x_end = std::min(frame.cols, static_cast<int>(centerX) + (int)roi.width);
int y_end = std::min(frame.rows, static_cast<int>(centerY) + (int)roi.height);

cv::Point pt1(x_start, y_start);
cv::Point pt2(x_end, y_end);
rectangle(color_frame, pt1, pt2, cv::Scalar(0, 0, 255), 2);

rectangle(color_frame, roi, cv::Scalar(255, 0, 0), 2);

cv::Rect best_search_window = roi;
double max_value = -1;

int spiralRadius = 1;
int maxSpiralRadius = std::min(x_end - x_start, y_end - y_start) / 2;

while (spiralRadius <= maxSpiralRadius) {
    for (double angle = 0; angle < 2 * M_PI; angle += 0.1) {
        int x = centerX + static_cast<int>(spiralRadius * cos(angle));
        int y = centerY + static_cast<int>(spiralRadius * sin(angle));

        if (x >= 0 && x + roi.width < frame.cols && y >= 0 && y + roi.height < frame.rows) {
            cv::Rect search_window(x, y, roi.width, roi.height);

            Image image;
            image.mat = frame(search_window);
            calculate_intensity_histogram(image);

            double similarity = correlation(roi_image.histogram, image.histogram);

            if (similarity > max_value) {
                max_value = similarity;
                best_search_window = search_window;
            }
        }
        spiralRadius++;
    }
}

```

- **centerX** ve **centerY**, ROI (Region of Interest - İlgilenilen Bölge) adı verilen bir bölgenin merkezini hesaplar. ROI, genellikle bir nesnenin belirli bir görüntüdeki konumunu temsil eder.
- **x_start** ve **y_start**, ROI'nin etrafındaki arama penceresinin başlangıç noktasını belirler. Bu pencere, ROI'nin çevresinde bir bölgeyi tarayarak nesneyi bulmaya çalışacak.
- **x_end** ve **y_end**, arama penceresinin sonunu belirler. Bu, pencerenin ROI'yi sınırlayan görüntü çerçevesinin sınırları içinde kalmasını sağlar.
- Ardından, **cv::Point** ve **rectangle** işlevleri kullanılarak, renkli bir görüntü çerçevesi üzerinde ROI ve arama penceresi etrafına dikdörtgenler çizilir. Bu, görüntü üzerinde takip edilen nesnenin konumunu görsel olarak işaretler.
- Bir sonraki adımda, **best_search_window** adlı bir **cv::Rect** (dikdörtgen) oluşturulur ve bu, başlangıçta ROI olarak tanımlanan bölgeyi temsil eder. Ayrıca, **max_value** adlı bir değişken, en yüksek benzerlik değerini ve bu benzerliğe sahip olan pencerenin konumunu izlemek için kullanılır.
- Kod, bir döngü içinde dairesel bir arama stratejisi kullanır. Başlangıçta, bir spiral yarıçapı **1** ile başlar ve maksimum spiral yarıçapına (**maxSpiralRadius**) kadar artırılır.
- İç içe iki döngü kullanılır. İçteki döngü, belirli bir spiral yarıçapında belirli bir açıda bir noktanın koordinatlarını hesaplar.

KONTROL SONUCU 

KISIM	Tarama Algortiması	Yaprak No:40
YAPILAN İŞ	Spiral Tarama Algortimasının Kodunu Yazma	Tarih: 23/08/2023

- Ardından, bu nokta çevresinde bir arama penceresi oluşturulur ve bu pencerenin içeriği işlenir. İşlenen içerik, bir histogram oluşturulması ve bu histogramın bir referans histogramla karşılaştırılması yoluyla bir benzerlik ölçüsü hesaplanmasıyla yapılır.
- Eğer hesaplanan benzerlik değeri (**similarity**) mevcut maksimum benzerlik değerini (**max_value**) aşarsa, yeni maksimum benzerlik değeri güncellenir ve **best_search_window** yeni bir pencereye ayarlanır.
- Spiral yarıçapı artırılarak, farklı noktalar etrafında arama devam eder. Bu, nesnenin takibini iyileştirmek için farklı bölgeleri kontrol etmek için kullanılır.
- Sonuç olarak, bu kod, bir nesnenin takibini yapmak için görüntü üzerinde bir arama penceresi stratejisi kullanır ve nesnenin en benzer olduğu konumu belirler.

Spiral Tarama Algoritmasının kodunu yazdıktan sonra kodumu test edip tarama işleminde geçen süreyi QTimer ile gözlemleyerek QDebug ile Qt konsolunda zaman değerlerini gözlemledim. Data-Set olarak aynı şekilde kırmızı top data-setini kullandım. Elde ettiğim zaman değerleri aşağıdaki gibidir.

```

Iterasyon 0 Sure: 0.008 sn Similarity 1
Iterasyon 1 Sure: 0.007 sn Similarity 0.858968
Iterasyon 2 Sure: 0.007 sn Similarity 0.840585
Iterasyon 3 Sure: 0.007 sn Similarity 0.815104
Iterasyon 4 Sure: 0.009 sn Similarity 0.787311
Iterasyon 5 Sure: 0.007 sn Similarity 0.691031
Iterasyon 6 Sure: 0.007 sn Similarity 0.642502
Iterasyon 7 Sure: 0.008 sn Similarity 0.890179
Iterasyon 8 Sure: 0.007 sn Similarity 0.854415
Iterasyon 9 Sure: 0.007 sn Similarity 0.747603
Iterasyon 10 Sure: 0.007 sn Similarity 0.545225
Iterasyon 11 Sure: 0.007 sn Similarity 0.489329
Iterasyon 12 Sure: 0.009 sn Similarity 0.77344
Iterasyon 13 Sure: 0.008 sn Similarity 0.721008
Iterasyon 14 Sure: 0.008 sn Similarity 0.552885
Iterasyon 15 Sure: 0.007 sn Similarity 0.781171
Iterasyon 16 Sure: 0.007 sn Similarity 0.760417
Iterasyon 17 Sure: 0.009 sn Similarity 0.650549
Iterasyon 18 Sure: 0.008 sn Similarity 0.869532

```

Elde ettiğim zaman değerlerini yorumlayacak olursam normal tarama yönteminde bir frame'i yaklaşık 0.014 ile 0.015 saniye arasında işleyebiliyordum. Spiral tarama yöntemi ile neredeyse frame başına 0.008 ile 0.007 saniye değerleri arasında zaman değerleri elde ettim.

KONTROL SONUCU 

KISIM	Thread Kullanarak Object Tracking	Yaprak No:41
YAPILAN İŞ	Threadler ve Mikroservis Düzeyinde Programlama Üzerine Araştırma	Tarih: 24/08/2023

Staj sürecimin son 2 gününe girerken Threadler, Mikroservis Düzeyinde Programlama ve Thread Kullanarak Object Tracking üzerine araştırmalar yapmaya başladım.

Thread Nedir?

Thread, bir işletim sistemi sürecinin içinde çalışan bağımsız iş parçacığıdır. Tek bir programın içinde birden fazla thread bulunabilir ve bu thread'ler aynı bellek alanını paylaşır. Her thread, kendi işlem sıralamasına sahip olabilir ve aynı programın farklı görevlerini eşzamanlı olarak gerçekleştirebilir.

Thread'in Özellikleri:

- Eşzamanlılık: Thread'ler, aynı anda birden fazla görevi yürütmek için kullanılır. Bu, işlemlerin daha hızlı ve verimli bir şekilde gerçekleştirilmesini sağlar.
- Paylaşılan Bellek: Thread'ler, aynı programın bellek alanını paylaşır, bu nedenle veri ve kaynaklar arasında kolayca iletişim kurabilirler.
- Hafiflik: Thread'ler işlem sürecine göre daha hafif ve daha az kaynak tüketirler. Bu nedenle çoklu işlemler için daha uygundur.
- İş Parçacığı Yönetimi: Thread'lerin yaratılması, sonlandırılması, duraklatılması ve devam ettirilmesi gibi işlemler özel iş parçacığı yönetim komutları kullanılarak yapılır.

Thread Kullanım Alanları:

- Çoklu İşlem: Birden fazla işlemi aynı anda yürütmek istediğinizde thread'ler kullanışlıdır.
- Paralel Programlama: İşlemleri paralel olarak yürütmek için thread'ler kullanılabilir, bu da performansı artırabilir.
- GUI Uygulamaları: Grafiksel kullanıcı arayüzü (GUI) uygulamaları, kullanıcı etkileşimini eşzamanlı olarak işlemek için thread'ler kullanabilir.
- İşlem Havuzları: Thread'ler, işlem havuzlarında görevleri paralel olarak işlemek için sıkça kullanılır.

Thread Güvenliği: Thread'ler arasında veri paylaşımı ve senkronizasyon sorunları olabilir. Bu nedenle, thread güvenliği için özel teknikler ve senkronizasyon mekanizmaları kullanmak önemlidir.

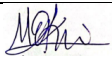
Thread Sorunları ve Çözümleri: Thread'lerin eşzamanlı çalışması bazı sorunları beraberinde getirebilir. Bu sorunların üstesinden gelmek için semaforlar, kilitleme mekanizmaları ve senkronizasyon teknikleri gibi araçlar kullanılır.

"Mikroservis düzeyinde programlama," mikroservis mimarisini uygularken her bir mikroservisin geliştirilmesi, dağıtılması ve yönetilmesini ifade eder. Bu yaklaşım, her bir mikroservisin bağımsız bir yazılım bileşeni olarak ele alındığı ve bu bileşenlerin birlikte çalıştığı bir uygulama geliştirme yöntemini içerir.

Mikroservis düzeyinde programlama yaparken aşağıdaki temel adımları takip edebilirsiniz:

- **Mikroservislerin Tanımlanması:** İlk adım, uygulamanızın işlevselliğini ve gereksinimlerini analiz ederek, bu gereksinimleri karşılamak üzere ayrılmış mikroservislerin belirlenmesidir. Her bir mikroservis, bir belirli işlevi veya hizmeti temsil etmelidir.
- **Bağımsız Geliştirme:** Her bir mikroservisi bağımsız olarak geliştirme süreci başlar. Her mikroservis için ayrı bir geliştirme ekibi veya kişiler sorumludur. Bu ekibin, mikroservisin kod tabanını, veritabanını ve geliştirme süreçlerini yönetmesi gerekir.
- **Bağımsız Dağıtım:** Her mikroservis, kendi dağıtım süreçlerine sahiptir. Bu, her bir servisin bağımsız olarak sunuculara dağıtılmasını, sürdürülmesini ve güncellenmesini sağlar. Bu, servislerin hızlı ve sorunsuz bir şekilde güncellenmesine olanak tanır.

KONTROL SONUCU 

KISIM	Thread Kullanarak Object Tracking	Yaprak No:42
YAPILAN İŞ	Threadler ve Mikroservis Düzeyinde Programlama Üzerine Araştırma	Tarih: 24/08/2023
<ul style="list-style-type: none"> • İletişim Protokolleri: Mikroservislerin birbiriyle iletişim kurabilmesi için hafif ve uygun iletişim protokolleri kullanılmalıdır. Bu protokoller, RESTful API'ler, RPC (Remote Procedure Call) veya diğer uygun iletişim yöntemlerini içerebilir. • Hata Yönetimi: Her mikroservisin kendi hata yönetimini ve günlüğünü tutması gereklidir. Ayrıca, servisler arası iletişimdeki hata durumlarına nasıl yanıt verileceği de belirlenmelidir. • Orkestrasyon ve Yönetim: Mikroservislerin birlikte çalışabilmesi için bir orkestrasyon mekanizması gereklidir. Bu, servislerin nasıl başlatılacağını, durdurulacağını ve ölçeklendirileceğini yönetir. Ayrıca, servislerin performansını ve sağlığını izlemek için yönetim araçlarına ihtiyaç vardır. • Güvenlik: Mikroservis düzeyinde programlama yaparken güvenlik büyük bir öneme sahiptir. Her bir servis, kendi güvenlik önlemlerini almalı ve kimlik doğrulama, yetkilendirme ve veri güvenliği gibi güvenlik gereksinimlerini karşılamalıdır. • Mikroservis düzeyinde programlama, karmaşık uygulamaları daha yönetilebilir ve ölçeklenebilir bir yapıya dönüştürmek için kullanılan bir yaklaşımdır. Her bir mikroservis, tek başına geliştirilebilir ve dağıtılabilir, bu da hızlı geliştirme ve hızlı dağıtımı mümkün kılar. Ancak, bu yaklaşımın iyi bir şekilde yönetilmesi ve koordine edilmesi gereklidir. 		
KONTROL SONUCU 		

KISIM	Thread Kullanarak Object Tracking	Yaprak No:43
YAPILAN İŞ	Object Tracking'de Thread Kullanımı Üzerine Araştırma	Tarih: 25/08/2023
<p>Thread kullanarak object tracking (nesne izleme) uygulama geliştirmek oldukça yaygın bir yöntemdir. Object tracking, video veya görüntü verileri üzerinde bir nesnenin hareketini veya pozisyonunu izlemeyi amaçlayan bir uygulamadır ve genellikle gerçek zamanlı uygulamalar içinde kullanılır. Thread'ler bu tür uygulamalarda birden fazla görevi eşzamanlı olarak yönetmek için oldukça yararlıdır. İşte object tracking üzerinde thread kullanımına ilişkin bazı önemli noktalar:</p> <ul style="list-style-type: none"> • Veri Akışı Yakalama ve İşleme: Nesne izleme için kamera veya video kaynağından sürekli veri akışı almanız gerekir. Bu verileri yakalamak ve işlemek için bir thread kullanabilirsiniz. Ana programınızın bir thread'i veri akışını yakalarken diğer thread'ler nesnenin izlenmesi ve pozisyonunun hesaplanması gibi işlemleri gerçekleştirebilir. • Görüntü İşleme ve Nesne Takibi: Görüntü işleme algoritmaları nesnenin görüntü üzerinde nasıl izleneceğini belirler. Bu işlem genellikle işlemci yoğun olduğundan, bu görevi bir thread üzerinde çalıştırmak, diğer işlemleri kesintiye uğratmadan gerçekleştirmenize olanak tanır. • Senkronizasyon ve Veri Paylaşımı: Thread'ler arasında güvenli veri paylaşımı ve senkronizasyon önemlidir. Özellikle nesnenin izleme sonuçlarını ana programa iletmek ve veri paylaşımını denetlemek için uygun senkronizasyon mekanizmalarını kullanmalısınız. • Çoklu Nesne Takibi: Eğer birden fazla nesneyi aynı anda izlemek istiyorsanız, her bir nesne izleme görevini ayrı bir thread üzerinde çalıştırabilirsiniz. Bu, daha fazla paralel işlem yapmanıza olanak tanır. • Performans ve Gerçek Zamanlılık: Thread kullanarak işlemleri paralel hale getirerek, nesne izleme uygulamanızın performansını artırabilirsiniz. Bu, gerçek zamanlı nesne izleme uygulamaları için kritik bir özelliktir. <p>Araştırmalarım sonucunda geliştirmiş olduğum Histogram Tabanlı Object Tracking algoritmamda bir sonraki hedefim takip edilmesi istenen nesne veya alanı bir sonraki framlerde tararken tarama işleminde Search Alanını threadlere bölerek tek threadle tarama yapmaktansa bir çok threadle tarama yaparak algortimamın daha hızlı çalışmasını sağlamak ve mikroservis düzeyinde programlama tecrübesi edinecektim. Staj sürecimin son 2 gününde bu konuda araştırma yaparak staj sürecimi sonlandırıyorum. Staj sürecimden sonra algortimamı thread düzeyinde programlamayı hedefliyorum.</p> <p>Lisans döneminde almış olduğum Görüntü İşleme dersinde hem teorik hem pratik olarak kazanmış olduğum deneyimlerimi bu staj sürecinde daha ileri bir seviyeye taşımış oldum.</p>		
KONTROL SONUCU 