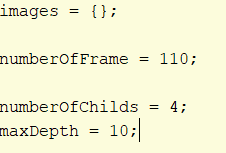
Ödev 5’te bizden istenen şey şuydu: Videodan 10 adet frame alıyoruz, daha sonra bu 10 frame’in SURF özniteliklerini çıkarıyoruz. Böylece 64 boyutlu bir uzayda bulunan öznitelik noktaları (SURFPoints) elde ediyoruz. Daha sonra bu noktaları kullanarak bir adet KD Tree oluşturmamız gerekiyor. Sonrasında ise 1 adet test frame’i alacağız. Bu test frame’inin de özniteliklerini çıkarıp, her bir özniteliği ağaç üzerinde gezdireceğiz ve ulaştığımız leaf’e oy verdireceğiz. Bu işlemi test frame’inin her bir özniteliği için yaptıktan sonra en çok oy alan leaf’lerin bulunduğu frame’lere bakacağız ve en yüksek oyu almış 3 frame’i bulacağız.

Hw5.m dosyasının içini inceleyelim:

Öncelikle VideoReader fonksiyonu ile videoyu okuyoruz.



Bu işlemin sonrasında bir takım değerlerimiz var.



images bir adet cell array ve frame’leri tutuyor. Başlangıç olarak içi boş, frame’leri okudukça images’in içine atacağız.

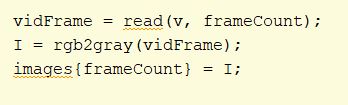
numberOfFrame değişkeni ise videodan kaç adet frame çekileceğini gösteriyor.

numberOfChilds ve maxDepth değişkenleri ise oluşturacağımız ağacın parametreleri. numberOfChilds değişkeni her bir node’un kaç çocuğu olacağını tutuyor, maxDepth ağacın derinliğini, yani aşağı inilecek seviyeyi belirtiyor.

Şimdi ise frame’leri işlemeye başlıyoruz.



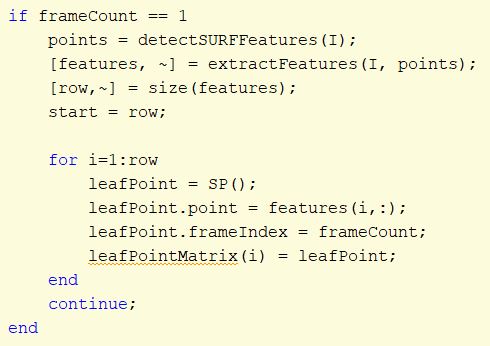
Bir adet for döngüsü yazdık. numberOfFrame kadar dönecek ve her bir frame için bir takım işlemler yapılacak.



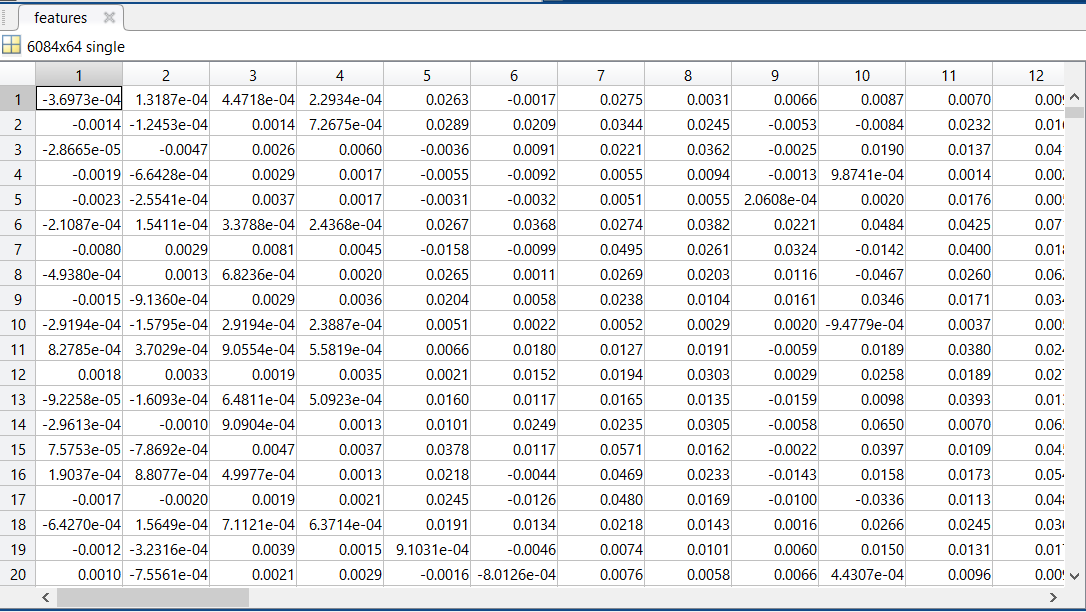
read() fonksiyonu ile numarası belirtilmiş frame’i okuyoruz. Kmeans algoritması gray scale resimlerin üzerinde çalıştığı için resmimizi rgb’den gray scale’a çeviriyoruz. Sonrasında ise bu resmi images dizisinin içine koyuyoruz. Daha sonra tekrar kullanacağız.

Okunacak ilk frame için ayrı bir kod bloğu, diğer frame’ler için ise ayrı bir kod bloğu, yani iki adet kod bloğu yazdık. Bu frame’leri bu kod bloklarının içinde işleyeceğiz.

İlk frame için yaptığımız işleme bakalım:

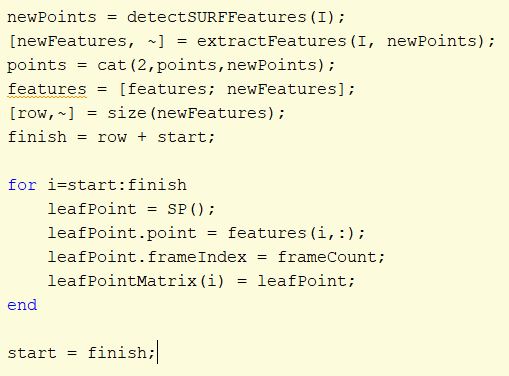


I frame’ini daha önce okumuştuk. Bunu detectSURFFeatures() fonksiyonuna vererek SURFPointleri elde ettik. Bunu da points adlı bir değişkende tutuyoruz. Daha sonra extractFeatures() fonsksiyonuna bu points dizisini ve frame I’yı verdik. Buradan da daha önce bulmuş olduğumuz noktaların feature’larını elde ettik. Bu feature’lar 64 boyutlu bir uzayda bulunuyor.



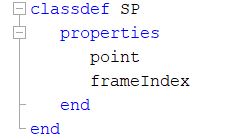
Şekilde görüldüğü gibi, extractFeatures() fonksiyonu ile nx64’lük bir matris elde ediyoruz.

İkinci kod bloğumuzda ise, yukarıda yaptığımız işlemleri tekrar ediyoruz. Yeni bulduğumuz noktaları daha önce bulunan features dizisine ekleyerek uzayı besliyoruz. Burada iki adet değişken var. Start ve finish. Bunları da sürekli güncelleyerek yeni frame’i geziyoruz.

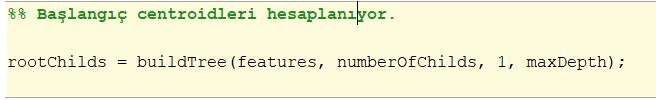


Burada dikkat edilmesi gereken bir nokta var, uzayı yeni bulunan feature’lar ile beslerken, hangi feature hangi frame’den elde edildi bilgisini kaybediyoruz. Bu bilgiyi kaybetmemek için bir adet sınıf yazdık. SP adlı sınıf içinde bir adet point ve bir adet de frameIndex tutuyor. Bu şekilde bulduğumuz objeleri SURFPoints olarak değil, SP objesi olarak tutuyor ve frame bilgisini kaybetmiyoruz.

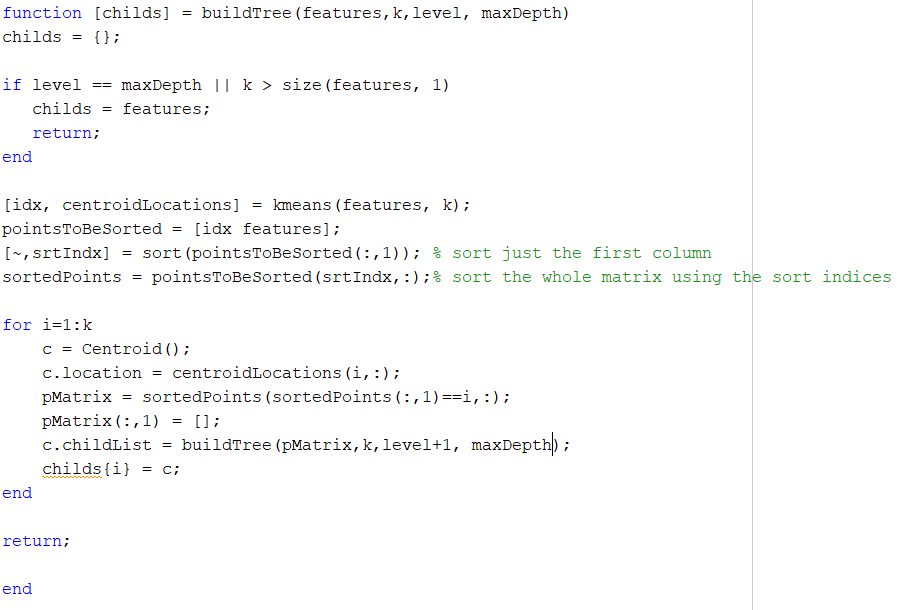
Yukarıdaki resimde görüldüğü gibi bir adet for döngüsü ile bu işlemi yapıyor ve elde ettiğimiz noktaları leafPointMatrix adlı bir dizide tutuyoruz.



Bu aşama ile birlikte frame’ler ve onlara ait öznitelikler elde edildi. Şimdi ise sırada ağacı oluşturma kısmı var. Seviye 1’de bulunacak node’lar burada hesaplıyor. Elimizde sadece ilk seviyede oluşan node’ları tutuyoruz. Oy verme işleminde iterasyon başlangıcı için bu node’ları kullanıyoruz.



Şimdi ise buildTree() fonksiyonunun içine bakalım:

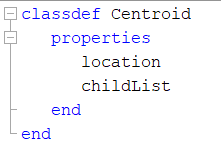


Bu fonksiyon parametre olarak feature dizisini, her bir node’un çocuk sayısı olan k değerini, şu an işlem yapılan level değerini ve en son gidilecek seviye olan maxDepth’i alıyor ve her seferinde bize bir child listesi dönüyor. Yukarıda bir adet if control statement’i var. Bununla recursion’un sonlanmasını kontrol ediyoruz. Eğer belirtilen son seviyeye gelmişsek ya da bölerek ilerlediğimiz sub-feature’ların sayısı çok az ise node oluşturma işlemini sonlandırıyor ve bu leaf node’a ait olan cluster’ı direk noda’a bağlıyoruz. Sonrasında return işlemini yapıyoruz.

Eğer bu şartlar sağlanmıyorsa yeni node’lar oluşturacağız demektir.

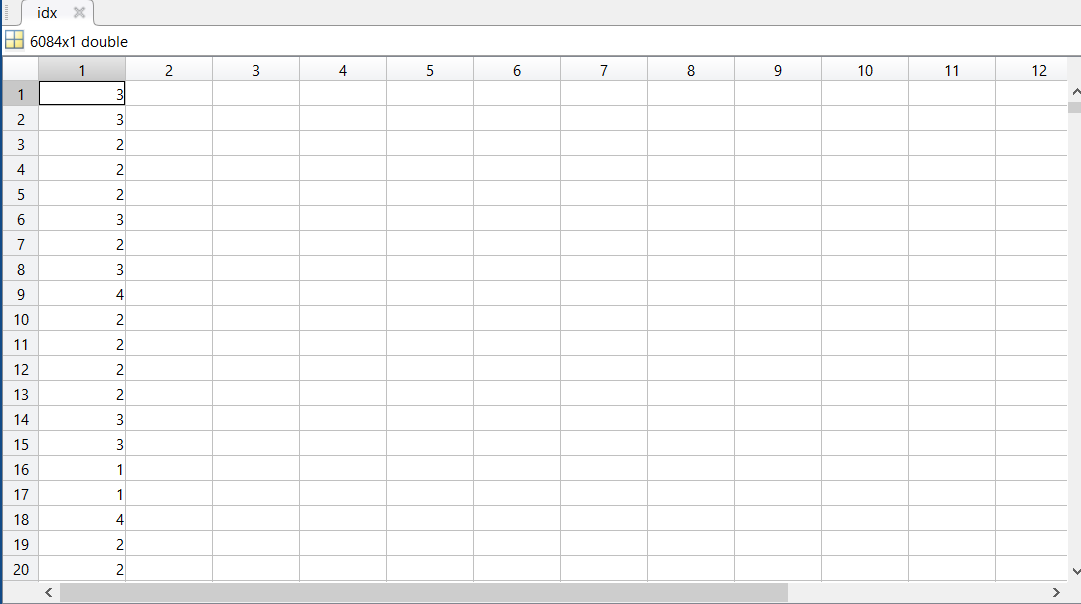
Burada şu işlemleri yapıyoruz: Önce kmeans() fonksiyonunu kullanarak feature dizisini cluster’lara bölüyoruz (k sayısı kadar cluster’a bölüyoruz). idx dizisi bize nx1’lik bir dizi dönüyor. Döndüğü bu değer ise o index’e sahip olan nokta hangi cluster’dadır bilgisidir, yani cluster numarasıdır.

Ağacın node’ları ise Centroid adlı bir sınıf. Her bir Centroid objesinin bir adet location’u yani koordinatı ve bir adet childList’i yani bir alt seviyede bulunan ve kendisine bağlanmış sub-Centroidler var.

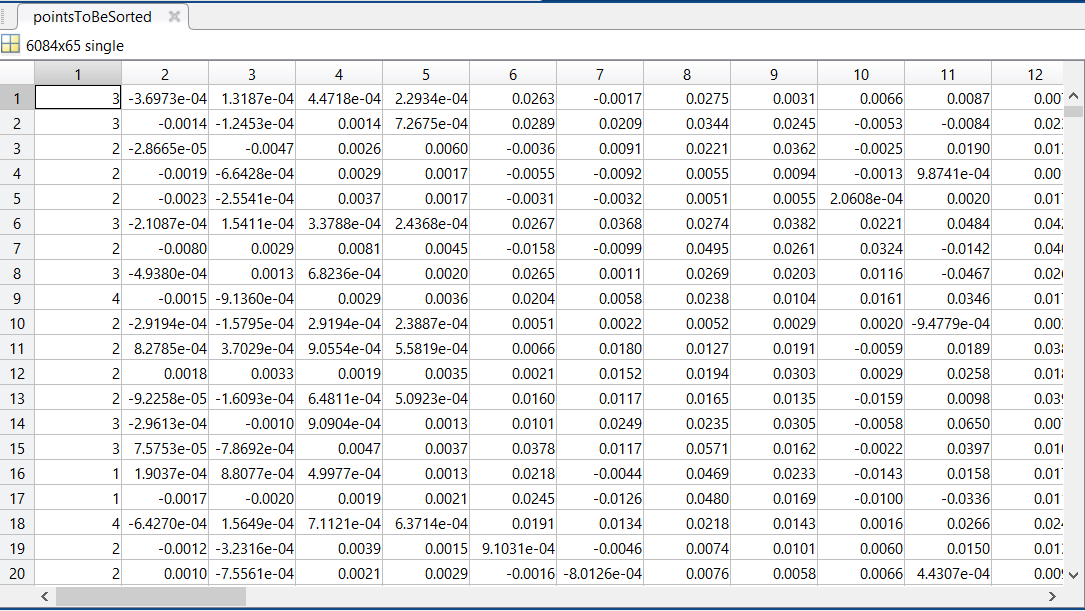


Burada şu işlemleri yapıyoruz:

Önce kmeans() fonksiyonunu kullanarak feature dizisini cluster’lara bölüyoruz (k sayısı kadar cluster’a bölüyoruz). idx dizisi bize nx1’lik bir dizi dönüyor. Döndüğü bu değer ise o index’e sahip olan nokta hangi cluster’dadır bilgisidir, yani cluster numarasıdır.

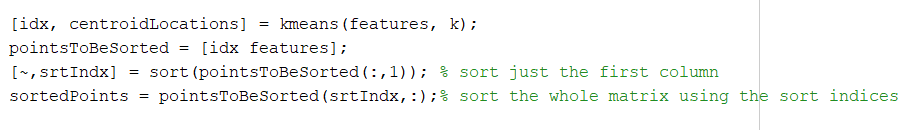


Sonrasında ise pointsToBeSorted adlı bir dizi oluşturuyoruz. Bu diziyi de idx ve features birleşiminden meydana getiriyoruz. Bu şekilde elde ettiğimiz matrisin ilk kolonunda o feature’ın cluster numarası, kalan kolonlarında ise 64 adet feature var.



Bir sonraki aşama olarak, bu matrisi cluster numarasının olduğu kolona, yani ilk kolona, göre sıralıyoruz. Bu sıralama işleminden elde ettiğimiz vektör ile diğer kolonları da sıralıyoruz ki bilgikaybı yaşamayalım. Bu şekilde tüm feature’lar cluster numarasına göre sıralanmış oluyor.

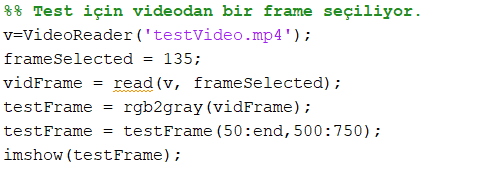
centroidLocations adlı dizi ise bu cluster’ların merkezi olan noktaları bize dönüyor. Bu noktaları da ağacın node’ları olarak kullanacağız.



Sonrasında ise: Sırasıyla cluster’ları alıyoruz. O cluster’ın centroid’ini alıp bundan bir adet node (bizim ağaçtaki temsilimiz ise Centroid adlı sınıf) oluşturuyoruz. Daha sonra buildTree() fonksiyonunu tekrar çağırarak, features olarak da bu cluster’ı vererek yeni sub-cluster’lar ve yeni sub-Centroid’ler oluşturuyoruz.

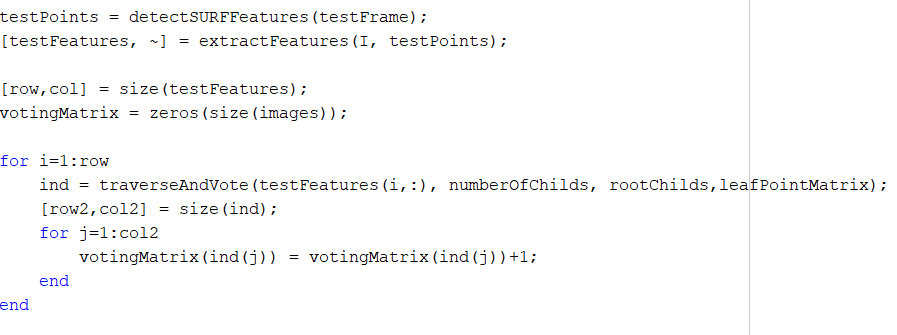
Bu şekilde ağacımız oluşmuş oluyor.

Bu aşamaya geldiğimizde train işlemini tamamlamış ve ağacımızı feature’lar ile oluşturmuş olduk. Şimdi ise test aşamasına geçelim.



Burada test için bir adet frame seçiyoruz ve o frame’in feature’larını çıkarıyoruz (Videonun boyutuna göre frame’den bi alt parça da alabiliyoruz).

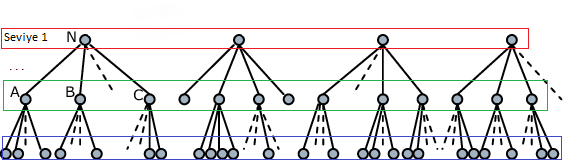
Şimdi ise elde ettiğimiz test feature’ları ağaç üzerinde gezdireceğiz ve ağacın leaf’lerinde bulunan feature’lara oy verdireceğiz.



Frame sayısı ile aynı boyuta sahip bir matris oluşturduk. İsmi ise votingMatrix. Bu matris hangi frame’e kaç adet oy verildi bilgisini tutuyor.

Bir adet for döngüsü yazıyoruz ve içinde ağacı gezeceğimiz traverseAndVote() fonksiyonunu çağırıyoruz. Bu for döngüsü kaç adet test feature varsa o kadar dönecek ve her çalıştığında bir frame numarası hesaplanacak. Bu numara o test feature’ın oy verdiği featurelar’ın bulunduğu frame numarası.

Şimdi ise traverseAndVote() fonksiyonunu inceleyelim. Öncesinde ise, üzerinde gezdiğimiz ağacın yapısını tam olarak anlamak için bir adet görsel ile ağacımızı görselleştirelim.



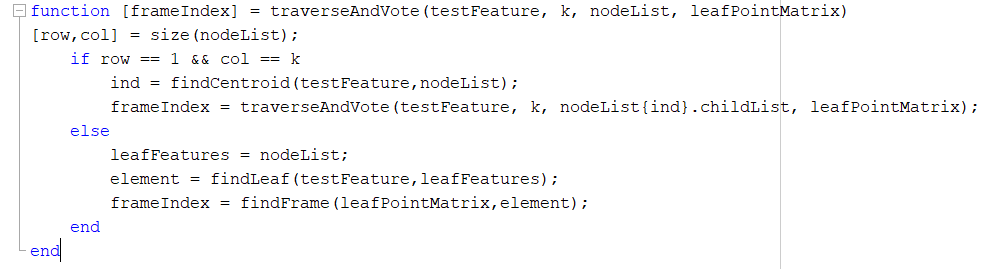
Kırmızı dikdörtgen içindeki node’lar bizim başlangıç seviyemiz. İterasyona bu node’lardan başlıyoruz. Arada diğer node’lar mevcut.

Yeşil dikdörtgen içindeki node’lar ise maxDepth seviyesinde bulunan node’lar. Bunlar bizim en altta bulunan Centroid objelerimiz.

En altta mavi dikdörtgenin içinde bulunan objeler ise artık Centroid objesi değil, bir Centroid’e bağlanmış olan cluster. Mavi dikdörtgen içinde bulunan her bir node aslında feature matrisindeki bir satır yani 1x64’lük bir feature.

Biz iterasyonu yaparken kırmızı dikdörtgen içinden başlayıp yeşil dikdörtgene ulaşacağız. Daha sonra ise test feature’ı mavi dikdörtgen içindeki hangi feature’a(yani feature matrisindeki hangi satıra) en çok benzerlik gösteriyorsa ona oy verdireceğiz. Önce oy vereceğimiz feature’ın SP objesi halini bulacağız, daha sonra ise bu SP objesinden frame numarasını elde edip bu frame numarasını sonuç olarak döneceğiz.

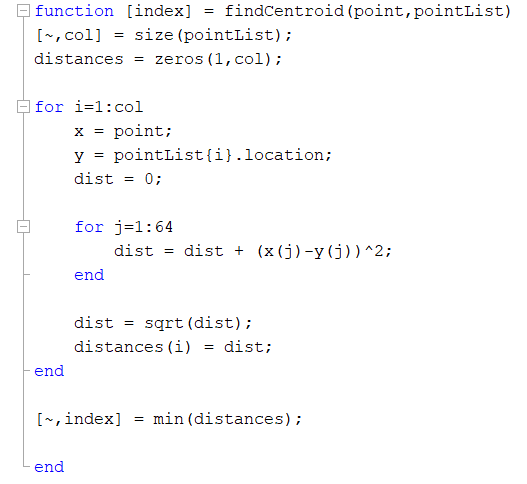
Fonksiyonun içerisi ise şu şekilde:



Bu fonksiyon parametre olarak test feature’ları alacak, her bir node’un çocuk sayısı olan k değerini alacak, iterasyona başlayacağı node listesini alacak ve son olarak da SP objelerinden oluşan leaf point matrisini alacak.

İterasyon ise şu şekilde: Eğer üzerinde bulunduğumuz şey bir Centroid ise henüz feature kısmına gelemedik demektir. O an ağacın üzerinde gezdirdiğimiz test feature node listesinden en çok hangisine benzerlik gösteriyorsa oraya ilerliyoruz. Benzerliği bulma işlemini ise findCentroid() fonksiyonu ile yapıyoruz. Bu şekilde Seviye 1’den Seviye maxDepth’e ulaşıyoruz.

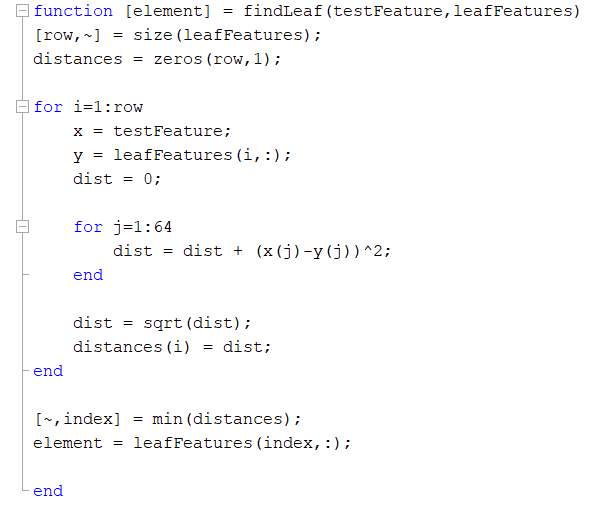
findCentroid() fonksiyonunu inceleyelim:



findCentroid() fonksiyonu parametre alarak bir adet test feature ve o an üzerinde bulunduğumuz node’un çocuklarının listesini alıyor ve sonuç olarak da bu listedeki node(Centroid)’ların hangisine daha çok benziyorsa onun index’ini dönüyor.

Seviye maxDepth+1’e geldiğimizde ise artık elimizde Centroid objeleri değil, feature’ların olduğu bir matris var. Bu kısımda ise artık findCentroid() fonksiyonunu değil, findLeaf() fonksiyonunu çağırıyoruz.

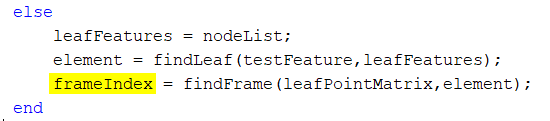
findLeaf() fonskiyonunun içini inceleyelim:



findLeaf() fonksiyonu parametre olarak test feature’ı ve karşılaştırma yapacağı feature matrisini alıyor. Matrisin elemanları ile teker teker karşılaştırma yapılıyor. Sonuç olarak ise test feature ile en çok benzerlik gösteren eleman dönülüyor.

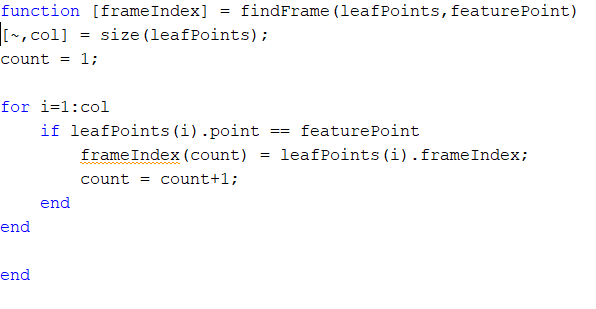
Bu aşamada traverseAndVote() fonksiyonundan çağrılan findLeaf() fonksiyonu görevini tamamladı ve bir adet 1x64 boyutunda feature elde edildi.

Şimdi ise sarı renk ile işaretlenmiş kısımdayız.



Burada elde ettiğimiz 1x64’lük feature hangi frame’den çekildi bilgisini bulmaya çalışacağız. Bunun için findFrame() adlı bir fonksiyon yazdık. Bu fonksiyon daha önce oluşturmuş olduğumuz ve içinde SP objeleri (SURFPoint – frameNumber ikililerini bulunduran objeler) olan bir matrisi yani leafPointMatrix’i ve iterasyon sonucu elde edilen elemanı alıyor ve bize bu elemanın ait olduğu frame numarasını dönüyor.

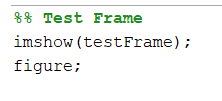
findFrame() fonksiyonunu inceleyelim:



Bu fonksiyon da bulduğumuz eleman ile tüm elemanları karşılaştırıyor ve elemanın matristeki karşılığını buluyor. Sonrasında ise matriste bulunan elemanların frame numaraları dönülüyor.

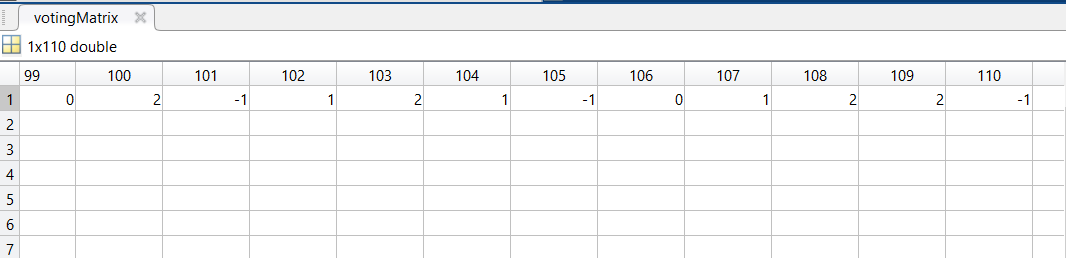
Buraya kadar yaptıklarımızı özetleyecek olursak: Test feature’ları teker teker ağaç üzerinde gezdirdik ve en yakın olan feature’ları bulduk. Bu feature’ların alındığı frame’leri elde ettik ve o frame’e +1 oy verdik.

Artık son kısımdayız. Önce test olarak seçilen frame’i ekrana bastırıyoruz.

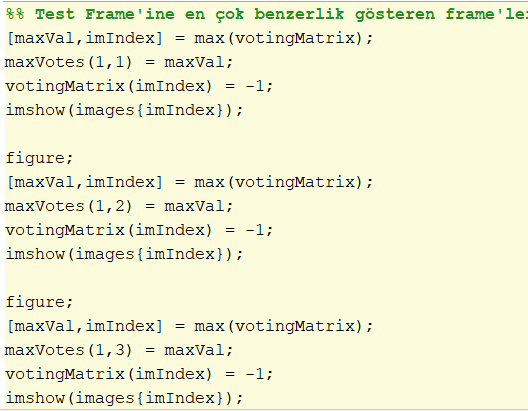


Test frame’imiz Figure 1 olarak ekrana basılacak.

Şimdi ise oylar sonucu elde ettiğimiz votingMatrix’e bakalım.

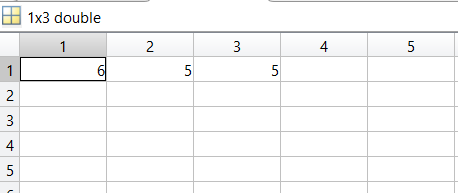


VotingMatrix’in içinde 110 adet frame’e ait oylar var. -1 oya sahip frame’ler maximum oya sahipler. Bunu da şu şekilde yapıyoruz:



Maksimum oya sahip 3 frame’i bulacağımız için, önce maximum oya sahip frame’i buluyor, onun oy sayısını maxVotes adlı bir yere kaydediyor, daha sonra oy sayısını -1’e set ediyoruz ki tekrar max fonksiyonunu kullandığımızda 2. en büyüğü bulabilelim. Bu işlemleri max değere sahip diğer 2 frame için de yapıyoruz.

Sonuç olarak: Seçmiş olduğumuz frame’e en çok benzerlik gösteren frame’ler 101. Frame, 105. Frame ve 110. Frame. Bu 3 frame’in oylarına bakalım:



Bu oylar ise sırayla 101., 105. ve 110. Frame’e ait.

Son olarak da votingMatrix’te oy sayısı -1 olan frame’leri ekrana bastırıyoruz.

testVideo1.mp4 videosundan alınan test Frame’i aşağıdaki gibidir:



Video1.mp4 adlı videoda bu frame’i arattığımızda aldığımız sonuçlar da aşağıdaki gibidir:

Sonuç 1:



Sonuç 2:



Sonuç 3:



Burada dikkatimize çarpan sonuç, test için seçilen frame identical değilse yani hesaplamalar sonucunda SURF Feature sayısı az çıkıyorsa, tree içerisinde yapacağı oylamadan sonra dönen sonuçlar kendisine çok benzemeyen sonuçlar olabilir. Bunun sebebi input olarak verilen test frame’deki feature sayısının az olmasından kaynaklanmaktadır.

Test frame’inde yeterli sayıda feature’ın bulunduğu koşullarda, k sayısı ve depth sayısı arttırıldıkça algoritma sonuç olarak daha belirgin ve doğru sonuçlar vermektedir.