Mobil Nesne Takibinin Hızlandırılması
Accelerating Mobile Object Tracking

Mustafa Özuysal
Bilgisayar Mühendisliği Bölümü
İzmir Yüksek Teknoloji Enstitüsü
İzmir, Türkiye
{mustafaozuysal}@iyte.edu.tr

***Özetçe* —Bu bildiride modern işlemcilerin Tekil İşlem Çoklu Veri (TİÇV) komutlarıyla hızlandırılmış bir nesne takip modülü içeren mobil bir artırılmış gerçeklik uygulaması sunulmaktadır. Hem standart C++ hem de ARM işlemciler için geliştirilen TİÇV komut seti olan NEON ile kodlanmış verimli bir Sıfır Ortalamalı Farkların Kareleri Toplamı (SOFKT) yöntemi detaylandırılmıştır. Bu iki yöntemin mobil cihaz üzerinde çalışma hızları ölçülerek karşılaştırılmıştır.**

***Anahtar Kelimeler*—Nesne Takibi, Gerçek Zamanlı Algoritmalar, Artırılmış Gerçeklik, Tek Komut Çoklu İşlem**

***Abstract*—In this paper, we present a mobile augmented reality implementation with an accelerated tracking module using Single Instruction Multiple Data (SIMD) instructions available in modern CPUs. We detail the implementation of an efficient Zero–Mean Sum of Squared Differences (ZMSSD) algorithm both using standard C++ and the SIMD NEON instruction set of ARM processors. We compare the numerical measurements of tracking speed on a mobile device of both versions.**

***Keywords*—Object Tracking, Real-Time Algorithms, Augmented Reality, Single Instruction Multiple Data**

I. GİRİŞ

Nesne takibi, arttırılmış gerçeklikten güvenlik uygulamalarına kadar geniş bir yelpazede kullanılan temel bir bilgisayarlı görü adımıdır. Temel anlamıyla, bir görüntüde konumu bilinen bir nesnenin, bu görüntü ile ilişkili bir başka görüntü içerisindeki konumunun tespitidir. Özellikle gerçek zamanlı ardışık görüntülerin işlendiği uygulamalarda hızlı çalışması gereken bir adımdır. Çünkü gerçek zamanlı uygulamalar genelde saniyede yirmi beş kareden fazlasının işlenmesini gerektirir ki bu da kare başına toplan işlem süresini kırk milisaniye ile sınırlar. Görüntünün kameradan alınması ve çıktının ekranda görselleştirilmesi de toplam zaman içerisinde yer aldığından nesne takibi için çok kısa bir süre kalmaktadır.

Zaman sınırlamaları nedeniyle hem uygulama için doğru nesne takip yönteminin seçilmesi hem de donanım kaynaklarının verimli kullanılması önemlidir. Yaklaşık 2005 yılından itibaren işlemci hızındaki senelik artış sona ermiş ve bunun yerine işlemci içindeki paralel donanım artmaya başlamıştır. Bu paralel donanımın avantajları ancak kullanılan yöntem doğru bir şekilde kodlandığında görülmektedir. Derleyiciler pek çok iyileştirmeyi otomatik olarak uygulayabilseler de üst seviye donanım özelliklerine programcının müdahalesi ile erişilebilmektedir.

Bu bildiri, işlemcilerin aynı işlemin aynı anda birden fazla veri üzerinde uygulanabilmesini sağlayan Tekil İşlem Çoklu Veri (TİÇV) komut setlerinin nesne takip uygulamalarını hızlandırmak amacıyla kullanılması üzerinedir. Özellikle mobil cihazlarda sınırlı donanımdan en yüksek performansın alınması için nesne takibinin işlem yoğun kısımlarının hızlandırılması önemlidir. Bu amaçla ARM işlemciler için geliştirilen TİÇV komut seti olan NEON komutları kullanılmıştır. NEON komutlarını kullanan nesne takip algoritmasının bu komutları kullanmayan standard C++ ile geliştirilmiş yöntem ile hız karşılaştırılması da yapılmıştır. Karşılaştırma için Linux ve Android işletim sistemlerinde çalışan ve nesne takibini de içeren bir arttırılmış gerçeklik uygulaması kullanılmıştır.

Hızlandırma işlemi, nesne takip uygulamalarında sıkça kullanılan ve işlem yoğunluğu yüksek olan Farkların Karelerinin Toplamı, FKT (Sum of Squared Differences, SSD) adımı üzerinde uygulanmıştır. FKT işlemi verilen iki görüntü parçasının benzerliğini ölçmek için kullanılır. Ardışık iki karedeki görüntü parçaları büyük değişim göstermediğinden bir karedeki bir noktaya ikinci kare içerisinde en çok benzeyen parça en düşük FKT değerine sahiptir. FKT hesaplanması görüntü parçaları üzerindeki pek çok parlaklık değerinin işlenmesini gerektirdiğinden işlem yoğunluğu yüksektir. Bu nedenle takip işleminin hızı FKT hesaplama hızıyla doğrudan ilintilidir.

Bildirinin amacı, gerçek zamanlı mobil uygulama geliştiren lisansüstü öğrenci, araştırmacı ve firma çalışanlarına NEON ve benzeri teknolojiler ile elde edilebilecek hız konusunda deneysel veri sunmak ve bu teknolojiler ile ilgili referans bilgi sağlamaktır. Bildiride önemli bir nesne takip adımı olan Farkların Karelerinin Toplamı'nın hesaplanması için NEON komutlarını kullanan bir yöntem geliştirilmiştir ve bir mobil arttırılmış gerçeklik uygulaması ile hız testleri gerçekleştirilmiştir.

II. İLGİLİ LİTERATÜR

Nesne takibi için kullanılan yöntemler genelde ardışık görüntülerde nesnenin konumunun bir kareden başka bir kareye taşınmasını gerektirir. Bazı nesne takip yöntemleri yinelemeli olarak nesnenin bir sonraki konumunu tahmin ederek çalışır. Bunlara en temel örnek, örüntüden bağımsız çalışabilen Kanade–Lucas–Tomasi yöntemidir [1], [2]. [3] ise ilk karedeki örüntüyü öğrenerek her noktaya özel doğrusal bir

Bu bildiride yer alan çalışmalar TÜBİTAK tarafından 113E496 numaralı araştırma projesi kapsamında desteklenmiştir.

978-1-5090-1679-2/16/$31.00

```
int sum = 0; int sqr_sum = 0; int sum_prod = 0;

for( int j = 0; j < 8; ++j ) {
    const uchar* p = reference_patch + j*8;
    const uchar* row = image_ptr(img, x-4, y-4 + j);
    sum += row[0]+row[1]+row[2]+row[3]
         +row[4]+row[5]+row[6]+row[7];
    sqr_sum += row[0]*row[0] + row[1]*row[1]
             + row[2]*row[2] + row[3]*row[3]
             + row[4]*row[4] + row[5]*row[5]
             + row[6]*row[6] + row[7]*row[7];
    sum_prod += row[0]*p[0] + row[1]*p[1]
              + row[2]*p[2] + row[3]*p[3]
              + row[4]*p[4] + row[5]*p[5]
              + row[6]*p[6] + row[7]*p[7];
}

int ssd = sqr_sum + patch_sqr_sum - 2*sum_product
        + (2*sum*patch_sum - sum*sum - patch_sum_sqr)
          / (8*8);
```

Şekil 1: Denklem 3 ile verilen SOFKT hesabını tek bir noktada $N = 8 \times 8$ boyutunda bir şablon üzerinde hesaplayan standart C++ kod parçası. `reference_patch` işaretçisi I görüntüsündeki şablonun ilk satır ve sütununa, `img` işaretçisi ise I' görüntüsünün ilk sütun ve satırına işaret etmektedir. Önceden hesaplanmış $\sum I$ terimi `patch_sum` ve $\sum I^2$ terimi `patch_sqr_sum` değişkenlerinde saklanmıştır. $\sum I'^2$ terimi `sqr_sum`, $\sum I'$ terimi `sum` ve $\sum II'$ terimi `sum_prod` değişkenlerinde tutulmaktadır.

takip yöntemi geliştirmiştir. [4] ise öğrenme adımını hızlandırarak bu yöntemi gerçek zamanlı uygulamalara adapte etmiştir.

Arama işlemi için bir başka yaklaşım da ikinci karede ilk karedekine benzer örüntülerin aranmasıdır. İlk karedeki nesne konumu bilindiğinden ikinci karede arama işlemi sınırlı bir hedef alanda gerçekleştirilir. Hedef alan üzerinde bir uyumluluk değeri hesaplanır ve bu değerin maksimum olduğu nokta nesne konumu olarak seçilir. Uyumluluk değeri olarak Farkların Kareleri Toplamı ya da Normallenmiş Çapraz İlişki (Normalized Cross Correlation) fonksiyonları seçilebilir. Bildiride FKT fonksiyonu kullanılmıştır. Bu yöntem özellikle birden fazla anahtar noktanın takip edildiği durumlarda Aynı Anda Konumlama ve Haritalama (Simultaneous Localization and Mapping) uygulamaları [5] ile mobil artırılmış gerçeklik için başarıyla uygulanmıştır [6].

Standard işlemci komutları bir yazmaç üzerindeki tek veriye tek bir işlem uygularken TİÇV komutları bir yazmaç üzerinde birden fazla veriyi saklar ve tek bir işlemi aynı anda her biri üzerinde uygulayabilir. Bu tip komutlar 1970'lerde vektör süper bilgisayarlarda ilk defa kullanıldıktan sonra 1996 yılında Intel'in tasarladığı MMX komut seti ile masaüstü işlemcilerde kullanılmaya başlamıştır. Intel uyumlu işlemciler için geliştirilen en yeni TİÇV komut setleri AVX2 ve AVX–512 [7] iken ARM işlemcilerde NEON [8] komut seti kullanılır.

TİÇV komutları C/C++ standart kütüphanelerinin parçası değildir. Ancak pek çok derleyici TİÇV eklentileri için standart başlıklar sunar. Bu bildiride de NEON komutları için GCC derleyicisi ve Android Yerel Geliştirme Kiti (Andorid NDK) içerisindeki NEON eklentileri kullanılmıştır.

```
const uchar* p   = reference_patch;
const uchar* row = image_ptr(img, x-4, y-4);

uint8x8_t vrow = vld1_u8(row);
uint8x8_t vp   = vld1_u8(p);

uint16x8_t vsqr  = vmull_u8(vrow, vrow);
uint16x8_t vprod = vmull_u8(vrow, vp);

uint16x8_t vsum      = vmovl_u8(vrow);
uint32x4_t vsqr_sum  = vdupq_n_u32(0);
uint32x4_t vsum_prod = vdupq_n_u32(0);

vsqr_sum  = vpadalq_u16(vsqr_sum, vsqr);
vsum_prod = vpadalq_u16(vsum_prod, vprod);

for( int i = 1; i < 8; ++i ) {
    p   += 8;
    row += image_row_step;
    vrow = vld1_u8(row);
    vp   = vld1_u8(p);

    vsqr  = vmull_u8(vrow, vrow);
    vprod = vmull_u8(vrow, vp);

    vsum      = vaddw_u8(vsum, vrow);
    vsqr_sum  = vpadalq_u16(vsqr_sum, vsqr);
    vsum_prod = vpadalq_u16(vsum_prod, vprod);
}

uint32_t arr[4];
vst1q_u32(arr, vsqr_sum);
int sqr_sum = arr[0] + arr[1] + arr[2] + arr[3];

vst1q_u32(arr, vsum_prod);
int sum_prod = arr[0] + arr[1] + arr[2] + arr[3];

uint32x4_t vsum4 = vpaddlq_u16(vsum);
vst1q_u32(arr, vsum4);

int sum = arr[0] + arr[1] + arr[2] + arr[3];
int ssd = sqr_sum + patch_sqr_sum - 2*sum_prod
    + (2*sum*patch_sum - sum*sum - patch_sum_sqr)
      / (8*8);
```

Şekil 2: Denklem 3 ile verilen SOFKT hesabını tek bir noktada $N = 8 \times 8$ boyutunda bir şablon üzerinde hesaplayan NEON komutları kullanan kod parçası. `uint8x8_t` ve benzeri veri tipleri doğrudan NEON yazmaçlarında tutulan değişkenler için kullanılır. Her satırdaki sekiz parlaklık değeri `vrow` ve `vp` sekizli sekiz bitlik yazmaçlara yüklenmekte ve ara değerler olan I'^2 ve $I'I$ değerleri `vsqr` ve `vprod` yazmaçlarında vektörel olarak hesaplanmaktadır. Parlaklık toplamları sekizli on altı bitlik `vsum` yazmaçında saklanırken, çarpım içeren toplamlar dörtlü otuz iki bitlik yazmaçlarda tutulmaktadır. NEON komutlarının açıklamaları Tablo I'de verilmiştir.

III. YÖNTEM

Nesne takip işlemi için I görüntüsündeki $\mathbf{x} = (x, y)$ noktası ile I' görüntüsündeki $\mathbf{x}' = (x', y')$ noktası arasındaki FKT değeri aşağıdaki gibi hesaplanır:

$$\text{FKT}(\mathbf{x}, \mathbf{x}') = \sum_{\delta \in R} \left(I'_{\mathbf{x}'+\delta} - I_{\mathbf{x}+\delta} \right)^2 \tag{1}$$

Burada $I_{\mathbf{x}}$, $\mathbf{x}$ noktasındaki parlaklık değerine, R ise FKT değerinin hesaplandığı şablonun alanına karşılık gelir. R

Tablo I: NEON komutları içeren kod parçasında kullanılan başlıca vektör komutlar ve girdi/çıktı tipleri

Dönüş Veri Tipi	NEON Komutu	Parametreler	Açıklama
uint8x8_t	vld1_u8	(uint8_t const *ptr)	Ardışık sekiz baytı sekizli sekiz bitlik bir vektöre yükle
uint16x8_t	vmull_u8	(uint8x8_t a, uint8x8_t b)	Sekizli sekiz bitlik iki vektörü çarp ve sonucu sekizli on altı bitlik bir vektöre yaz
uint16x8_t	vmovl_u8	(uint8x8_t a)	Sekizli sekiz bitlik veriyi sekizli on altı bitlik veriye dönüştür.
uint32x4_t	vdupq_n_u32	(uint32_t value)	Otuz iki bitlik bir değeri dörtlü otuz iki bitlik bir vektörün her elemanına yükle
uint32x4_t	vpadalq_u16	(uint32x4_t a, uint16x8_t b)	Sekizli on altı bitlik vektörün yan yana elemanlarını ikişerli olarak topla her bir sonucu dörtlü otuz iki bitlik vektörün elemanları ile toplayıp sonucu dörtlü otuz iki bitlik bir vektöre yaz
uint16x8_t	vaddw_u8	(uint16x8_t a, uint8x8_t b)	Sekizli sekiz bitlik vektörü sekizli on altı bitlik vektör ile geniş topla
void	vst1q_u32	(uint32_t * ptr, uint32x4_t val)	Dört otuz iki bitlik bir vektörün elemanlarını işaretcinin gösterdiği hafızaya ardışık yaz

genelde $\mathbf{x}$ merkezli ve $K \times K = N$ boyutlarında bir kare olarak alınır.

İki görüntü arasında pozlama süresi veya ışık değerlerinin değiştiği durumlarda FKT değerleri bundan çok etkilenmektedir. Bu yüzden FKT hesaplanırken her iki alan üzerindeki ortalama parlaklık çıkarılarak daha sağlıklı bir uygunluk değeri hesaplanabilir. Hesaplanan bu değere Sıfır Ortalamalı FKT (SOFKT) denir ve aşağıdaki gibi hesaplanır:

$$\text{SOFKT}(\mathbf{x}, \mathbf{x}') = \sum_{\delta \in R} \Big[\big((I'_{\mathbf{x}'+\delta} - \mu') - (I_{\mathbf{x}+\delta} - \mu) \big) \Big]^2 \quad (2)$$

Burada $\mu = \frac{1}{N} \sum_{\delta \in R} I_{\mathbf{x}+\delta}$ ve $\mu' = \frac{1}{N} \sum_{\delta \in R} I'_{\mathbf{x}'+\delta}$ iki görüntü için hesaplanan R alanı üzerindeki ortalama değerleri verir.

Nesne takip işlemi sırasında SOFKT değerleri ikinci görüntüdeki (I') hedef alan içerisinde pek çok nokta ($\mathbf{x}'$ değeri) için hesaplanır ve en düşük değere sahip nokta takip işleminin sonucunu verir. Bu sebeple SOFKT hesabı için gerekli ve $\mathbf{x}'$'den bağımsız terimlerin tek sefer hesaplanması ve depolanması yeterlidir. Bu amaçla Deklem 2 aşağıdaki şekilde açılabilir:

$$\begin{aligned}
\text{SOFKT} &= \sum \Big[\big((I' - \mu') - (I - \mu) \big) \Big]^2 \\
&= \sum \Big[\big((I' - I) + (\mu - \mu') \big) \Big]^2 \\
&= \sum I'^2 + \sum I^2 - 2 \sum II' \\
&\quad + \sum (\mu^2 - 2\mu\mu' + \mu'^2) + 2 \sum (I' - I)(\mu - \mu') \\
&= \sum I'^2 + \sum I^2 - 2 \sum II' \\
&\quad + N\mu^2 - 2N\mu\mu' + N\mu'^2 \\
&\quad + 2\mu \sum I' - 2\mu' \sum I' - 2\mu \sum I + 2\mu' \sum I \\
&= \sum I'^2 + \sum I^2 - 2 \sum II' \\
&\quad + \Big[2 \sum I \sum I' - (\sum I)^2 - (\sum I')^2 \Big] / N
\end{aligned} \quad (3)$$

Sadeleştirme amacıyla terimlerdeki $\mathbf{x} + \delta$ ve $\mathbf{x}' + \delta$ indeksleri çıkarılmıştır ve tüm toplamlar N noktadan oluşan R alanı üzerindendir. Denklem 3 ile SOFKT hesaplamasında $\sum I$ ve $\sum I^2$ terimleri $\mathbf{x}'$ içermediğinden bir kere hesaplanıp kaydedilebilir. Geriye hedef arama alanı üzerinde her $\mathbf{x}'$ için ayrı ayrı hesaplanması gereken üç terim kalır: $\sum I'^2$, $\sum I'$ ve $\sum II'$.

Şekil 1'de verilen C++ dilindeki kod parçası verilen tek bir $\mathbf{x}'$ için Denklem 3 ile $N = 8 \times 8$ boyutunda bir şablon üzerinde SOFKT hesaplanması için kullanılabilir. Bu kod parçası SOFKT hesaplamasını hızlandırmak için yapılabilecek temel iyileştirmeleri içerdiğinden mobil cihazlarda gerçek zamanlıya yakın nesne takibi için kullanılabilir. Ancak pek çok mobil işlemci içerisinde yer alan paralel donanımı kullanamamaktadır. Çünkü derleyiciler henüz karmaşık kod parçalarını vektör operasyonlara başarıyla dönüştürememektedir. Bu nedenle doğrudan vektör operasyonları kullanarak tekrar kodlanması nesne takip hızını arttırabilmektedir.

ARM işlemci içerisindeki NEON yazmaçları 128 bit uzunluğunda olduğundan gri tonlamalı bir resimdeki sekiz ardışık değerin ya da bunların çarpım veya karelerinin tek bir yazmaça paralel yüklenmesi ve kullanılması mümkündür. Bu sayede $\sum I'^2$, $\sum I'$ ve $\sum II'$ terimleri için yapılacak hesap, her bir görüntü satırı için tek işlemci komutuyla yapılabilir. Toplam sekiz satır olduğundan `for` döngüsünün sadece sekiz kere çalışması yeterlidir. Şekil 2, SOFKT hesaplanması için kullanılabilecek NEON komutlarını kullanan eşdeğer C++ kod parçasını içermektedir. Tablo I'de ise kullanılan NEON komutlarının açıklamaları yer almaktadır.

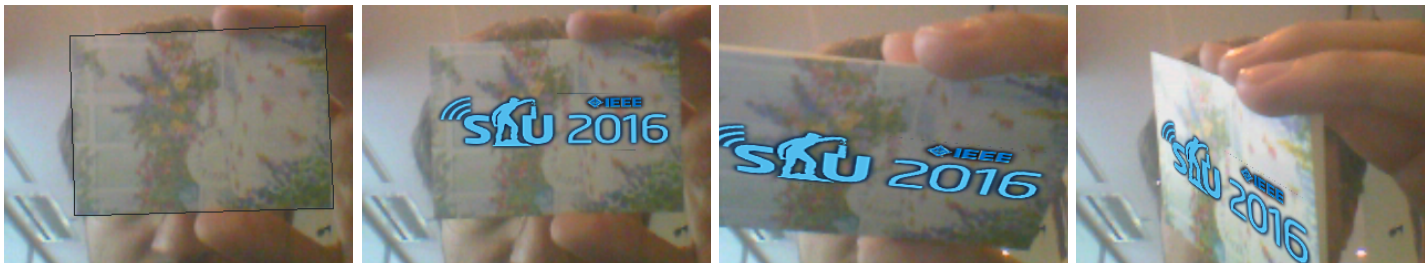
IV. DENEYLER

Geliştirilen kod parçalarının testleri için nesne takibi kullanan bir artırılmış gerçeklik uygulaması kullanılmıştır. Bu uygulama hem masaüstü hem de mobil cihazlarda çalışabilecek şekilde tasarlanmıştır.

Şekil 3'te uygulamanın çalışma adımları masaüstü sürümünde gösterilmektedir. Uygulama ile takip edilmek istenen nesne referans karede işaretlendikten sonra her karede betimleyici tabanlı (BRIEF [9] betimleyicisi kullanan) bir yöntem ile nesne aranmaktadır. Arama işlemi başarılı olunca daha hızlı olan nesne takip yöntemine geçilir. Nesne takibi, referans görüntü üzerindeki FAST [10] anahtar nokta konumlarının kamera görüntülerinde SOFKT yöntemiyle bulunması ile gerçekleştirilir. Takip edilen anahtar noktalar iki boyutlu bir perspektif dönüşümün PROSAC [11] yöntemi ile hesaplanması için kullanılır. Bu dönüşüm kullanılarak sanal nesne ve yazıların kamera görüntüsüne eklenmesi sağlanır.

Aynı yöntemin mobil cihaz sürümünün çıktıları Şekil 4'te verilmiştir. Uygulama Samsung Galaxy 4 cihazı üzerinde çalışmaktadır. Bu cihazın ARM Cortex–A15 işlemcisi NEON komut setini desteklemektedir. Bu sayede hem standart C++ hem de NEON komutları ile yazılan SOFKT kodlarının aynı anda test edilmesi mümkün olmuştur.

Yapılan hız testlerinin sonuçları Tablo II'de verilmiştir. Standart C++ ile elde edilen hız pek çok mobil uygulama için yeterlidir. Ancak çözünürlük, arama alanı ya da işlemci hızı



Şekil 3: Testlerin yapıldığı artırılmış gerçeklik uygulaması. İlk görüntüde seçilen alan takip edilecek nesneyi belirler. İstenilen grafikler takip işlemi sonucu kamera görüntüsüne aktarılır.

Şekil 4: Testlerin yapıldığı mobil cihaz sürümünün ekran çıktıları ve hız testleri sırasında kullanılan bazı görüntüler.

Tablo II: Standart C++ ve NEON sürümlerinin hız karşılaştırması. NEON komutlarıyla işlemler vektörel olarak yapıldığından saniyede işlenebilen kare sayısı yaklaşık iki katına kadar arttırılmıştır.

Farkların Kareleri Toplamı	Saniyedeki Kare Sayısı			
	Minimum	Ortalama	Medyan	Maksimum
C++	19.84	21.68	21.53	23.62
C++ ve NEON	33.09	39.91	41.03	45.41

değişirse yetersiz kalabilecek bir değerdedir. NEON komutları ile yaklaşık iki kat bir hızlanma elde edilmiştir. Bu da hem daha gerçekçi bir artırılmış gerçeklik deneyimi hem de donanım ve yöntem parametrelerinden görece bağımsız gerçek zamanda çalışma imkanı tanımaktadır.

V. SONUÇ

Bu bildiride nesne takibi için kullanılan bir yöntem olan Farkların Kareleri Toplamı hesaplanması için hem standart C++ ile hem de ARM işlemcilerin Tekil İşlem Çoklu Veri komut seti olan NEON komutlarıyla kod parçaları geliştirilmiştir. Bu kod parçalarının hız testleri mobil cihazlarda çalışan bir artırılmış gerçeklik uygulamasıyla test edilerek hız kazanımı niceliklendirilmiştir.

Bildiride NEON komutları kullanılmadan önce SOFKT hesaplanması için standart C++ ile verimli bir kod parçası geliştirilmiştir. Bir kere hesaplanarak saklanabilecek terimler toplanarak gereksiz işlemlerden kaçınılmıştır. NEON ve benzeri komut setleri kullanılmadan önce yapılacak hesaplama için en doğru yönteminin seçilmesi ve bu yöntem için verimli bir kod parçasının standart komutlarla geliştirilmesi önem taşır. Çünkü NEON ve benzeri komut setleriyle yazılan kod parçalarını anlamak ve hataları ayıklamak daha zordur. Tüm analizler sonucunda vektör hesaplamalardan yararlanabilecek bir kod parçası bulunduğunda, işlemci içerisindeki vektör komutlar kullanılarak donanımdan en iyi şekilde faydalanmak ve benzer uygulamalara göre fark yaratmak mümkündür.

KAYNAKÇA

[1] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision." in *International Joint Conference on Artificial Intelligence*, vol. 81, 1981, pp. 674–679.

[2] C. Tomasi and T. Kanade, *Detection and tracking of point features*. School of Computer Science, Carnegie Mellon Univ. Pittsburgh, 1991.

[3] F. Jurie and M. Dhome, "Real time robust template matching." in *British Machine Vision Conference*, 2002, pp. 1–10.

[4] S. Holzer, M. Pollefeys, S. Ilic, D. J. Tan, and N. Navab, "Online learning of linear predictors for real-time tracking," in *European Conference on Computer Vision*. Springer, 2012, pp. 470–483.

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *International Symposium on Mixed and Augmented Reality*, Nara, Japan, November 2007.

[6] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Pose tracking from natural features on mobile phones," in *International Symposium on Mixed and Augmented Reality*, Cambridge, UK, Sep. 2008.

[7] "Intel Intrinsics Guide," 2016. [Online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/

[8] "ARM NEON Intrinsics Reference," 2016. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.ihi0073a/IHI0073A_arm_neon_intrinsics_ref.pdf

[9] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a Local Binary Descriptor Very Fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, 2012.

[10] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 1, pp. 105–119, Jan 2010.

[11] O. Chum and J. Matas, "Matching with prosac - progressive sample consensus," in *Conference on Computer Vision and Pattern Recognition*, San Diego, CA, June 2005, pp. 220–226.