

**THE IMAGINATION UNIVERSITY PROGRAMME**

**RVfpga Deney 5**

**Görüntü İşleme: C & Çevirici**

# GİRİŞ

# Bu deneyde görüntü işleme yordamlarını yapan RISC-V programlama projeleri geliştireceksin. Projeler, kimisi C’de kimisi çeviricide yazılmış, birden çok kaynak dosyaları içerecek. C işlevlerinin nasıl çevirici yordamlarını ya da çevirici işlevlerinin nasıl C yordamlarını çağırabileceğini göstereceğiz.

# GÖRÜNTÜ İŞLEME ÖĞRETİCİSİ

Bu deneye RGB görüntüyü (Figür 1’in sol yanı) işleyip grilisini (Figür 1’in sağ yanı) oluşturan, bitmiş bir programı inceleyerek başlayacağız. Bu program C ile RISC-V çevirici dillerinde yazılıp PlatformIO ortamında çalışmak üzere yapılandırılmıştır. Şuradadır:

*[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing*

Kaynak kodu *src* altdizinindedir.

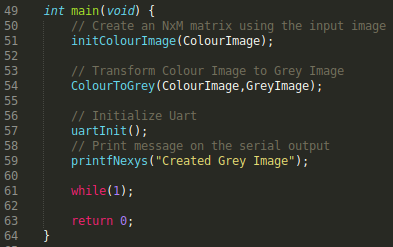


Figür . RGB görüntünün Grili görüntüye dönüşümü.

1. **Proje yapısı ile *main* işlevi**

Program şu kaynak dosyalarından oluşur: **main.c**, **VanGogh\_128.c, assemblySubroutines.S**. *.c* dosyaları işlevler (görüntü dönüşümlerini gerçekleştiren işlevler gibi) ile değişken bildirilerini (karakter dizisi olarak bildirilmiş görüntü gibi) barındırır. **assemblySubroutines.S** dosyası görseli rgb’den griliye dönüştüren şu çevirici dilinde gerçekleştirilmiş işlevi barındırır: *ColourToGrey\_Pixel*.

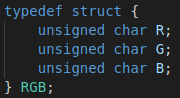
Figür 2 projenin main işlevini gösterir. İlk girdi görüntü verisiyle N x M matris oluşturan *initColourImage* işlevini çalıştırır. Ardından renkli görüntüyü grili görüntüye dönüştürür (işlev *ColourToGrey*). Son olarak işlev bir ileti yazdırıp sonsuz döngüye girer (while (1);).



Figür . ImageProcessing projesindeki *main* işlevi

1. **RGBli, grili görüntüler**

Bir görüntü, elemanlarının belirli bir ölçekte piksel değerlerini gösterdiği bir piksel matrisinden oluşur. RGB’de bütün pikseller üç değere karşılık gelir, bu da kırmızı (**R**), yeşil (**G**), mavi (**B**) bileşen parlaklıklarıdır. Dolayısıyla renkli bir görüntünün pikselleri üç-bileşenli vektörler olacaktır. Bu projede RGB piksel türü için şu tanımı kullanıyoruz:



Kod *RGB* adında bir yapı tanımlar. C’de, struct veri türü, bir adla belirlenen bir değişkenler, yüksek olasılıkla değişik türlerde, koleksiyonudur. Bu yapı aynı türden (unsigned char) *R*, *G,* *B* diye üç alan barındırır. Yani renk kanalları 8 bitle gösterilir, dolayısıyla piksel başına 24 bit olacak (24bpp) biçimde 256 ayrı yoğunluk düzeyi algılayabiliriz. Bu günümüzün dijital görüntü işlemesinde yaygın bir biçimdir.

Grili bir görüntüyü göstermek için 0 ile 255 arasında bir değer (bir kanal) piksellerin parlaklığını belirtir. Bu ImageProcessing projesinde grili görüntüyü bir 2-boyutlu karakter dizisiyle gösteriyoruz:



1. **Renkli bir görüntüyü grili bir görüntüye dönüştürme**

İki renk uzayı (RGB ile Grili) arasındaki dönüşüm şu ağırlık toplamayla yapılır:

grey = 0.299\*R + 0.587\*G + 0.114\*B

Bu denklemin tabanı şurada tanımlanan algoritmalardır <https://www.mathworks.com/help/matlab/ref/rgb2gray.html>.

Pikselleri renk kanallarını denklemdeki ağırlıkla çarparak grili değerlerini hesaplıyoruz. Ağırlıkların toplamı (0.299+0.587+0.114) birdir, dolayısıyla sonuçtaki grili değer 0-255 aralığında olacaktır, yani bir bayt ile gösterilebilir.

Denklemde verilen ağırlıkları kullanmak için gerçek sayılarla işlememiz gerekir ancak **SweRV EH1** işlemcisi kayan noktalı sayı desteği içermez. İlk Kullanım Kılavuzunun Bölüm 5.H’indeki DotProduct programında gösterildiği gibi kayan nokta emülasyonu da kullanılabilir ancak bu deneyde tam sayı aritmetiği tabanlı bir yaklaşım kullanıyoruz. Ağırlıklar önce tam sayılara çevrilir, toplam ise bir ikinin üssü olur (burada 210). Ağırlıkları tam sayıya dönüştürmek için bütün kayan nokta ağırlıklarını 210 ile çarpıp en yakın tam sayıya yuvarlıyoruz:

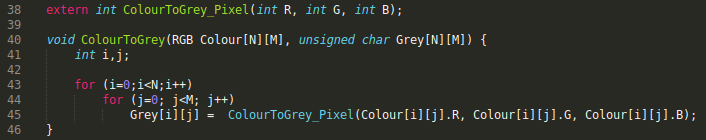
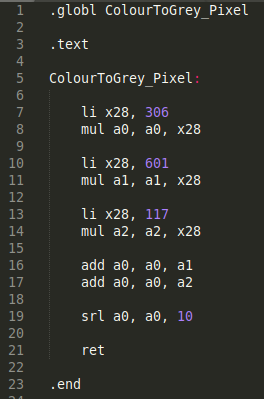
* 0.299\*210 = 306.176 ≈ **306 (weight for R)**
* 0.587\*210 = 601.088 ≈ **601 (weight for G)**
* 0.114\*210 = 116.736 ≈ **117 (weight for B)**

Son olarak grili değeri 0-255 aralığına indirgemek için toplamı 210 ile bölüyoruz, ki bu değeri 10 bit sağa kaydırarak kolayca yapılabilir. Dolayısıyla son dönüşüm şu formülle elde edilir:

grey = (306\*R + 601\*G + 117\*B) >> 10

Önemli olarak sabitlerin toplamı (306+601+117) 1024 olduğu göz önünde bulundurulursa son grili değer yine 0-255 aralığında olacaktır.

Figür 3 *ColourToGrey* işleviyle (sol yan) *ColourToGrey* çalıştırdığı *ColourToGrey\_Pixel* alt yordamının (sağ yan) görselleştirmesini yapar.

Figür . *ColourToGrey* işlevi (*main.c* dosyasında gerçekleştirilmiştir) ile *ColourToGrey\_Pixel* alt yordamı (*assemblySubroutines.S* dosyasında gerçekleştirilmiştir).

Çevirici dilinde semboller (değişkenler, işlevler, alt yordamlar) normalde yereldir, bir diğer deyişle diğer dosyalara görünmezdir. Bu yerel sembolleri genel sembollere dönüştürmek için .globl çevirici program yönlendirmesiyle dışa aktarmamız gerekir. Figür 3'ün sağ yanındaki ilk satır (.globl ColourToGrey\_Pixel) *ColourToGrey\_Pixel* işlevini dışa aktararak farklı bir dosyada bulunan *ColourToGrey* işlevi tarafından kullanılabilir *(main.c).* Figür 3'ün sol yanındaki ilk satır (extern int ColourToGrey\_Pixel(int R, int G, int B)) ColourToGrey\_Pixel işlevini bu dosyanın dış işlevi olarak bildirir.

1. **Programın yürütmesi, sonuçların görselleştirilmesi**

Griliye dönüştürme tamamlandıktan sonra, programın yürütmesi sona ermeden önce, kimi bellek bölgelerinin içeriğini dosyalara dökebiliriz. Bunun için, GDB ayıklayıcısının dump komutunu kullanırız. Proje kodunu çalıştırmak ve görüntü sonuçlarını elde etmek için sonraki adımları izle:

1. VSCode ile PlatformIO’yu aç.
2. Üst menü çubuğunda, *File* (Dosya) → *Open Folder* (Klasörü Aç) tıklayıp ardından *[RVfpgaPath]/RVfpga/Labs/Lab5* dizinine git. *ImageProcessing* dizinini seç (açma, yalnızca seç)ardından pencerenin üstünde OK’a tıkla. PlatformIO şimdi projeyi açacaktır.
3. platformio.ini açıp board\_build.bitstream\_file’ın yorumluğunu kaldırıp veri dosyasının dizinini gir. Örneğin Deney 1’de oluşturduğun veri dosyasını kullanabilirsin.

board\_build.bitstream\_file = [RVfpgaPath]/RVfpga/Labs/Lab1/Project1/Project1.runs/impl\_1/rvfpga.bit

1. src dizinindeki bütün kaynak dosyalarını açıp (*main.c*, *assemblySubroutines.S*) çözümle ki programın nasıl çalıştığını anla.
2. RVfpga’i Nexys A7 kartına sol menü şeridindenki PlatformIO ikonuna tıklayarak indir, ardından Project Tasks → env:swervolf\_nexys → Platform genişletip Upload Bitstream tıkla.
3. PlatformuIO'da programı yürütün. Bunu kartta (bu durumda önce Nexys A7'ye, önceki adımda olduğu gibi RVfpga yüklemeniz gerekir) veya Whisper simülatörü (RVfpga İlk Kullanım Kılavuzunda açıklandığı gibi) kullanarak yapabilirsiniz. İki türlü de PlatformIO’nun sol yan çubuğundaki “Run”  butonuna tıkla, ardından oynat butonuna  basarak ayıklayıcıyı başlat.

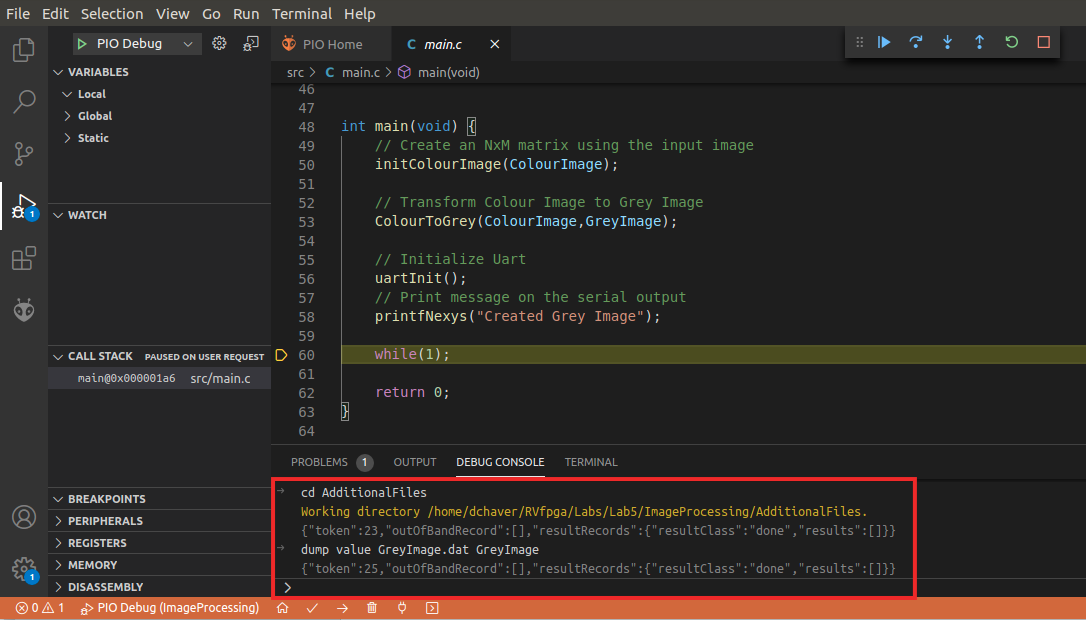
Yürütme *main* işlevinin yürütmesinin başında duracaktır, “Continue” butonuna basarak sürdür.

Kısra bir süre sonra (yaklaşık 1 saniye), program yukarıda açıklanan grili görüntü dönüşümlerini bitirip sondaki sonsuz (while(1);) döngüye erişecektir (Figür 2’ye göz at). *Pause* butonuna basarak  yürütmeyi beklet.

1. Şu komutları Debug Console’da çalıştırarak grili görüntüyü (GreyImage) dışa aktar (Figür 4’e göz at, bu iki komutun yürütmesini gösterir):

cd AdditionalFiles

dump value GreyImage.dat GreyImage



Figür . Grili görüntüyü dosya olarak dışa aktar

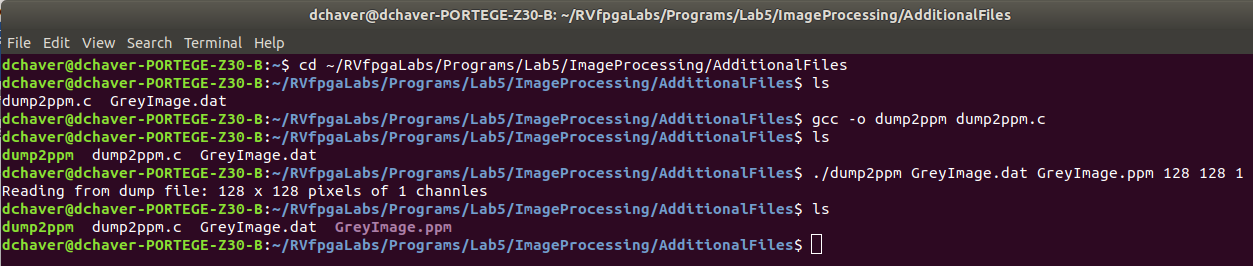
1. .dat dosyasını sisteminde görüntüleyebileceğin.ppm dosyasına dönüştür.

**LINUX içerisinde:** terminali açıp şu komutları yazarak yap (Figür 5’e göz at):

cd *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles*

gcc -o dump2ppm dump2ppm.c

./dump2ppm GreyImage.dat GreyImage.ppm 128 128 1



Figür . Görseli *.ppm* biçimine dönüştür

**WINDOWS’ta:** ya şöyle yap:

1. dump2ppm.exe yürütülebilirini şuradan kullanarak *[RVfpgaPath]\RVfpga\Labs\Lab5\ImageProcessing\AdditionalFiles*. Bir komut istemcisi aç, o klasöre git, yürütülebiliri yukarıdaki argümanlarla çalıştır:

dump2ppm.exe GreyImage.dat GreyImage.ppm 128 128 1

Ya da şöyle

2. dump2ppm.c programını derlemek için Cygwin’i (RVfpga İlk Kullanım Kılavuzunda tanımlandığı gibi kurduysan) kullanarak. Ardından programı (dump2ppm.exe) Cygwin terminalinde ya da yukarıdaki birinci seçenekteki gibi komut istemcisinde çalıştır.

1. .ppm dosyasını GIMP, GNU Image Manipulation Program, kullanarak aç. Eğer program kurulu değilse kurucuyu indirmek için şu websitesine git:

https://www.gimp.org/downloads/

Grili görüntü Figür 1’in sağ yanındaki gibi görünmeli (renkli girdi görüntüsüne şuradan da erişebilirsin *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/VanGogh\_128.ppm*, Figür 1’in sol yanındaki gibi görünmeli).

# Alıştırmalar

**Alıştırma 1.** Programı başka bir girdi görüntü üzerinde yürüt. Şurada sağlanan görüntüyü kullanabilrsin: *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/src/TheScream\_256.c* (Karşılık gelen .ppm görüntüsüne şuradan bakabilirsin: *[RVfpgaPath]/RVfpga/Labs/Lab5/ImageProcessing/AdditionalFiles/TheScream\_256.ppm.* Bu görseli sen de dat2ppm programını çalıştırarak oluşturacaksın, önceden tanımlandığı gibi.)

**Alıştırma 2.** *VanGogh* grili görüntüsündeki beyaza yakın (>235) ile siyaha yakın (<2) ögelerin sayısını sayan bir C işlevi oluştur. İki sayıyı da Western Digital’in PSP ile BSP kütüphanelerini Deney 2’nin Bölüm 3’ünde açıklandığı gibi dizisel konsolda yazdır.

**Alıştırma 3. ColourToGrey\_Pixel** çevirici altyordamını bir C işlevine, **ColourToGrey** C işlevini, **ColourToGrey\_Pixel** C işlevini çalıştıran bir çevirici altyordamına dönüştür.

* + C’de bütün işlevlerle genel değişkenler varsayılan olarak dışarıya genel sembol olarak aktarılır, dolayısıyla *ColourToGrey\_Pixel* işlevini *ColourToGrey* altyordamında kullanabilirsin.
  + Çevirici dilinde bir matrise erişmek için bir ögenin adresini (*i*,*j*) dizinin başlangıç adresine göre hesaplamalısın. ANSI C standardına göre iki-boyutlu diziler bellekte satır satır depolanır. Dolayısıyla satır i, sütun j’deki bir pikselin adresi başlangıç adresiyle boşluk (i\*M + j)\*B toplanarak bulunur, burada M sütun sayısı, B ise piksel başına kaplanan bayt sayısıdır: RGB’de üç bayt, grilide bir bayt.

**Alıştırma 4.** VanGogh renkli görüntüsüne bir **Bulanıklaştırma Filtersi** uygula (internette çok bilgi bulabilirsin; örneğin şuradaki bilgiyi kullanabilirsin: [https://lodev.org/cgtutor/filtering.html#Find\_Edges](https://lodev.org/cgtutor/filtering.html#Find_Edges_)).

.dat görüntüsünü bir .ppm görüntüsüne dönüştürmek için dump2ppm komut çalıştırmasında 1 kanal yerine 3 kanalın değerlendirilmesi için bir yeri değiştirmen gerek:

./dump2ppm FilterColourImage.dat FilterColourImage.ppm 128 128 **3**

Dahası, filtreli görseli orijinaliyle karşılaştırabilirsin, şuradan erişilebilir *[RVfpgaLabsPath]/RVfpgaLabs/Programs/Lab5/ImageProcessing/AdditionalFiles/VanGogh\_128.ppm*