Unit 6 Computer Vision mit RVision

# Vorbereitungen

## Clear workspace

Löschen Sie die alte Umgebung, um in einem “neuen” R zu starten.

rm(list=ls())

# Rvision

## Installation

### Devtools

Zur Installation der Rvision Library wird Devtools benötigt.

#if (!require(devtools))  
# install.packages("devtools")

### ROpenCVLite

Im Anschluss muss ROpenCVLite installiert werden.

#install.packages("ROpenCVLite")

#library(ROpenCVLite)

#devtools::install\_github("swarm-lab/Rvision")  
library(Rvision)

##   
## Attache Paket: 'Rvision'

## Die folgenden Objekte sind maskiert von 'package:graphics':  
##   
## frame, image

## Die folgenden Objekte sind maskiert von 'package:base':  
##   
## exp, log, merge, split, sqrt, sum

# 3. Rvision als Computer Vision Bibliothek für R

## Installation

### Devtools

Zur Installation der Rvision Library wird Devtools benötigt.

#if (!require(devtools))  
# install.packages("devtools")

### ROpenCVLite

Im Anschluss muss ROpenCVLite installiert werden.

#install.packages("ROpenCVLite")

#library(ROpenCVLite)

#devtools::install\_github("swarm-lab/Rvision")  
library(Rvision)

# Find the path to the balloon1.png image provided with Rvision  
path\_to\_image <- system.file("sample\_img", "balloon1.png", package = "Rvision")  
  
# Load the image in memory  
my\_image <- image(filename = path\_to\_image)

plot(my\_image)



## Input/output Operations

### Bildobjekt aus einer Datei

my\_file\_img <- image("../images/Riedlingen.jpg")

### Bilddatei schreiben

write.Image(my\_file\_img, paste0(tempdir(), "/Riedlingen\_export.png"))

## Image saved successfully.

### Bildeigenschaften aufrufen

isImage(my\_file\_img)

## [1] TRUE

dim(my\_file\_img)

## [1] 144 256 3

nrow(my\_file\_img)

## [1] 144

ncol(my\_file\_img)

## [1] 256

bitdepth(my\_file\_img)

## [1] "8U"

colorspace(my\_file\_img)

## [1] "BGR"

### Bild anzeigen

plot(my\_file\_img)



### Erweiterte Operationen an Bildern

#### KOnvertierung in Grayscale

my\_file\_img\_gray <- changeColorSpace(my\_file\_img, "GRAY")  
plot(my\_file\_img\_gray)



#### Aufsplittung der Farbkanäle in einzelne Bilder

my\_channels <- split(my\_file\_img) # Order of channels is Blue, Green, Red, Alpha  
  
layout(matrix(1:4, 2, 2, byrow = TRUE))  
invisible(lapply(my\_channels, plot)) # 'invisible' is used to hide the console output. It is not necessary otherwise.



#### Separate Channels mergen

my\_merged\_channels <- merge(my\_channels)  
plot(my\_merged\_channels)



#### Filter: Blurs

Image von <https://pixabay.com/de/photos/ara-papagei-vogel-hybrid-rot-grün-943228/>

img <- image("../images/macaw-943228\_1280.jpg")  
library(dplyr)

## Warning: Paket 'dplyr' wurde unter R Version 4.4.3 erstellt

##   
## Attache Paket: 'dplyr'

## Die folgenden Objekte sind maskiert von 'package:stats':  
##   
## filter, lag

## Die folgenden Objekte sind maskiert von 'package:base':  
##   
## intersect, setdiff, setequal, union

boxFilter(img, k\_height = 25, k\_width = 25) %>% plot()



gaussianBlur(img, k\_height = 25, k\_width = 25, sigma\_x = 5, sigma\_y = 5) %>% plot()



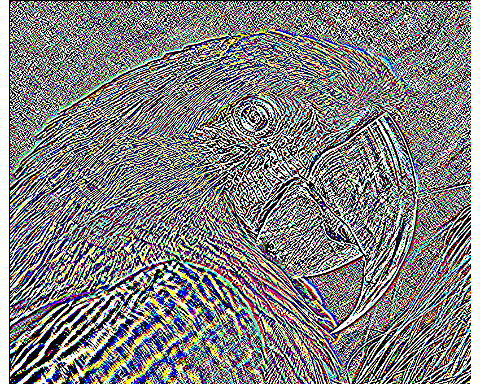
medianBlur(img, k\_size = 25) %>% plot()



sqrBoxFilter(img, k\_height = 25, k\_width = 25) %>% plot()

 #### Edge Detection Anwenden von Edge detection Algorithmen wie sobel und laplacian

sobel(img) %>% plot()



#### Draws

img1 <- cloneImage(img)  
drawCircle(img1, x = 750, y = 750, radius = 200, color = "blue",   
 thickness = 10)  
plot(img1)



#### Blob Detektion

Bis jetzt haben wir uns eine Reihe verschiedener Funktionen angeschaut, aber alle wurden einzeln verwendet. Jetzt wollen wir sie kombinieren, um etwas innerhalb des Bildes zu erkennen.

Bild laden

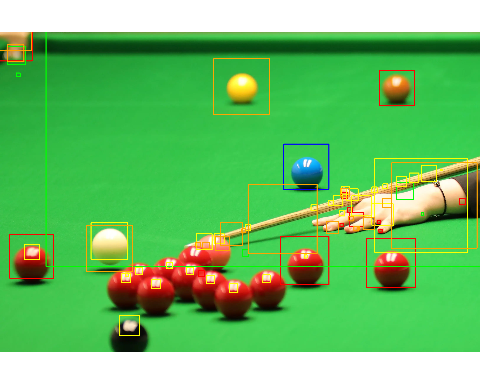
img <- image("../images/snooker2.jpg")  
plot(img)

 Funktion für die Blob Detektion

blob\_fun <- function(img, fun, color = character()) {  
 img %>%  
 split() %>%  
 do.call(fun, .) %>%  
 medianBlur(15) %>%  
 simpleBlobDetector(max\_area = Inf, min\_area = 10, blob\_color = 255,  
 filter\_by\_convexity = FALSE,   
 filter\_by\_inertia = FALSE, min\_threshold = 0) %>%  
 mutate(color = color)  
}   
  
multi\_draw <- function(img, blobs) {  
 if (nrow(blobs) > 0) {  
 for (i in 1:nrow(blobs)) {  
 drawRectangle(img,   
 blobs$x[i] - 1 + blobs$size[i],   
 blobs$y[i] - 1 + blobs$size[i],  
 blobs$x[i] - 1 - blobs$size[i],   
 blobs$y[i] - 1 - blobs$size[i],   
 thickness = 5, color = blobs$color[1])  
 }  
 }  
}

blue <- function(B, G, R) B > 150 & R < 200 & G < 200  
red <- function(B, G, R) R > 150 & B < 200 & G < 150  
green <- function(B, G, R) G > 150 & B < 200 & R < 200  
yellow <- function(B, G, R) G > 150 & B < 200 & B > 150 & R > 150  
orange <- function(B, G, R) G > 150 & B < 150 & R > 150

blue\_mms <- blob\_fun(img, blue, "blue")  
red\_mms <- blob\_fun(img, red, "red")  
green\_mms <- blob\_fun(img, green, "green")  
yellow\_mms <- blob\_fun(img, yellow, "yellow")  
orange\_mms <- blob\_fun(img, orange, "orange")  
  
multi\_draw(img, blue\_mms)  
multi\_draw(img, red\_mms)  
multi\_draw(img, green\_mms)  
multi\_draw(img, yellow\_mms)  
multi\_draw(img, orange\_mms)  
  
plot(img)

 Übung: Passen Sie den Blob-Filter so an, dass er bessere Resultate liefert.