Übung 3: Detektion mit Histogrammen

Übung von: Remo Schwarzentruber

Einführung

Objekte in einem Bild zu finden oder zu vergleichen kommt bei vielen Bildverarbeitungsproblemen zum Einsatz. Wenn sich ein Objekt in einer Farbe ganz klar von den anderen Bildinhalten abhebt, genügt es möglicherweise die entsprechende Farbe zu detektieren. Wie sieht es nun aber aus, wenn das Objekt keine einheitliche Farbe hat sondern mehrere Farben? Und wenn das Objekt vielleicht nicht immer in der gleichen Ansicht sichtbar ist?

Einen Ansatz bietet der Vergleich mittels Histogrammen. Das Histogramm des gesuchten Objekts dient als dessen Beschreibung (Modell). An jeder Position im Bild kann nun das Histogramms des umliegenden Bereichs, mit dem Histogramm des Objekts verglichen werden. Sind die Histogramme ähnlich, könnte sich das Objekt dort befinden. Als Vergleichsfunktion zwischen Histogrammen dient die Histogramm-Intersection oder die Bhattacharyya Distanz.

Übung 1: Spieler Detektion

In den Beispielsbilder finden Sie Bilder von Fussballszenen. Sie sollen nun versuchen die Spieler einer Mannschaft zu detektieren.

Gehen Sie dazu wie folgt vor:

- Laden Sie das Bild und das Bild eines Spielers. Alternativ können sie später auch einen anderen Teil des Bilds ausschneiden. Sie haben nun 2 Bilder, eines das den Spieler enthält, den Sie suchen sollen (Modellbild) und das Bild auf dem Sie suchen sollen (Suchbild). Ihr Testprogramm soll diese beide Bilder als Input erhalten.
- Lesen Sie die Bilder ein und berechnen Sie das Histogramm des Modellbilds. Merken Sie sich auch die Grösse.
- Berechnen Sie nun an jedem Bildpunkt (oder an jedem n-ten Punkt, falls die Berechnung zu lange dauert) im Suchbild das Histogramm aus einem Bildausschnitt, der der Grösse des gesuchten Modells entspricht.
- Vergleichen Sie die beiden Histogramme mit einer geeigneten Funktion.
- Sie haben nun den Histogramm Vergleich an jedem Punkt berechnet. Wie sieht das Bild der Resultate des Histogram Vergleichs aus? Wo liegt das Maximum?
- Wo haben Sie Spieler gefunden? Zeichnen Sie entsprechende Rechtecke im Bild ein.

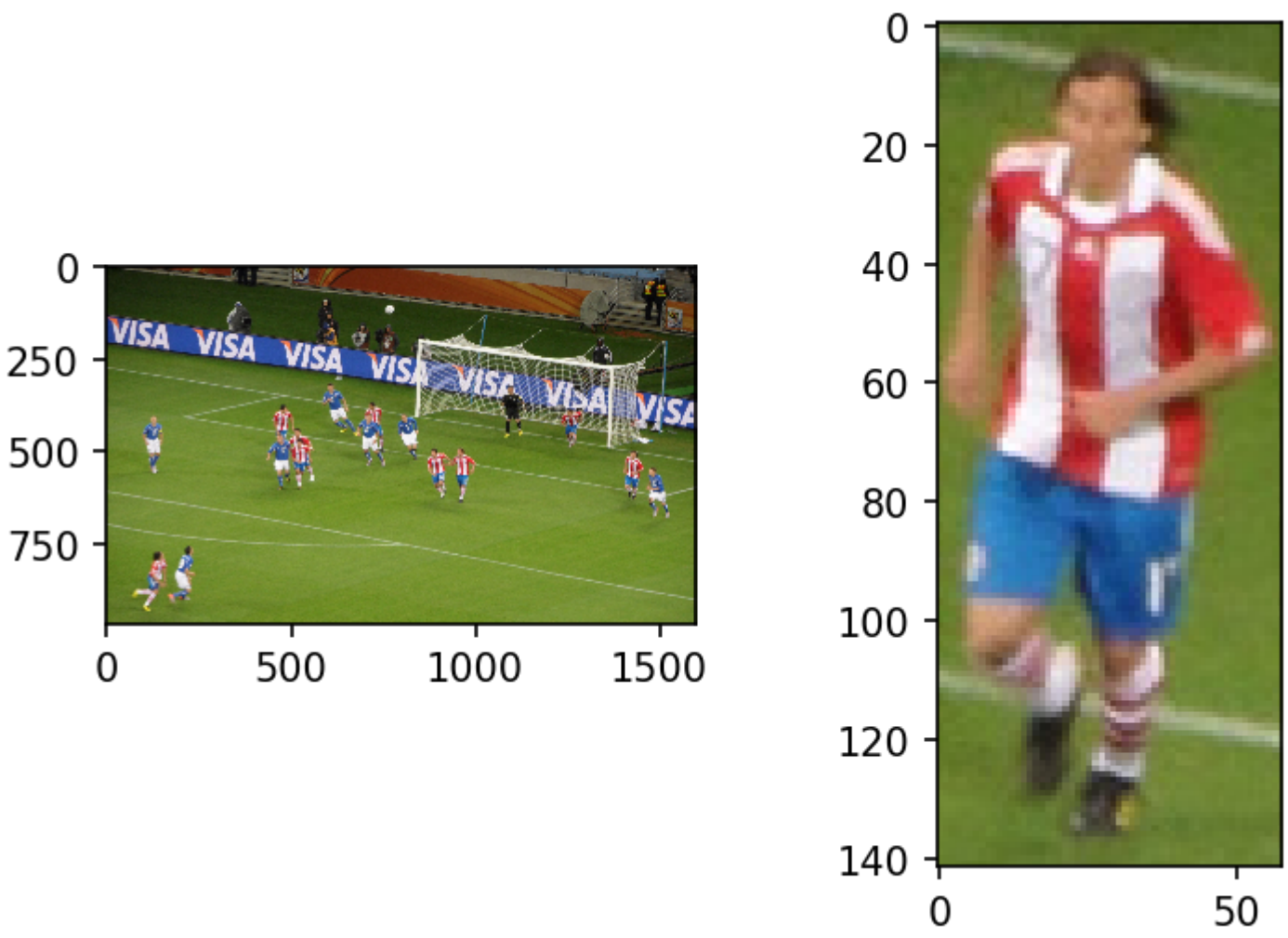
Wie könnten Sie die Maximumssuche verbessern?

Imports and settings

In [2]:
```
# OpenCV needs to be included first
import cv2
import numpy as np

# for displaying images in jupyter

import matplotlib as mpl
from matplotlib import pyplot as plt
%matplotlib inline
mpl.rcParams['figure.dpi']= 150
```

In [3]:
```
player  = cv2.imread('../data/player.jpg')
player = cv2.cvtColor(player, cv2.COLOR_BGR2RGB)
image = cv2.imread('../data/soccer-game.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.subplot(1, 2, 1)
plt.imshow(image)
plt.subplot(1, 2, 2)
plt.imshow(player)
```

Out[3]: <matplotlib.image.AxesImage at 0x125938668>



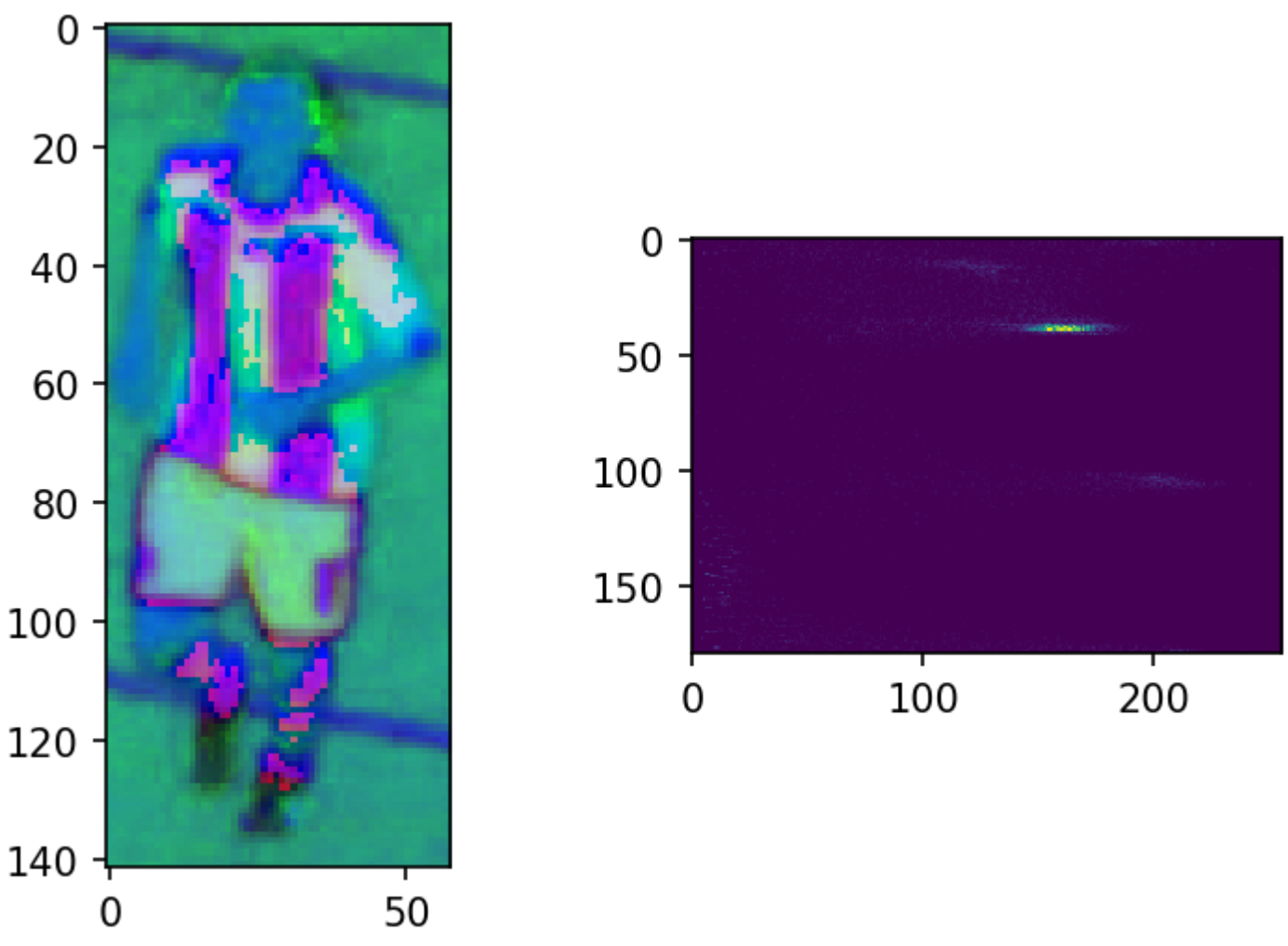
Histogramme berechnen

Berechnen Sie als erstes das Histogram des Spielers. Beachten Sie dabei die Dokumentation der Histogrammberechnung in OpenCV. Sie können das Histogramm über 1, 2 oder 3 Farbkanäle berechnen. Für den besseren Vergleich eignen sich vielleicht andere Farbmodelle besser als RGB.

Berechnen Sie als erstes ein 2 Kanal Histogram und stellen Sie dieses dar.

In [4]:
```
player_hsv = cv2.cvtColor(player, cv2.COLOR_RGB2HSV)
player_histogram = cv2.calcHist([player_hsv],[0,1], None, [180, 256], [0, 180, 0, 256])

plt.subplot(1, 2, 1)
plt.imshow(player_hsv)
plt.subplot(1, 2, 2)
plt.imshow(player_histogram, interpolation = 'nearest')
```

Out[4]: <matplotlib.image.AxesImage at 0x127a83e80>



Vergleich

Um Histogramme zu vergleichen, kann in OpenCV die Funktion `compareHist()` verwendet werden. Ein Parameter gibt dabei an, mit welcher Methode die Histogramme verglichen werden. Verbreitete und geeignete Methoden sind die Bhattacharyya Distanz oder die Kullback-Leibler Divergenz. Testen Sie beide Methoden in dem Sie das obige Histogramm mit sich selber vergleichen.

https://docs.opencv.org/4.0.1/d6/dc7/group__imgproc__hist.html#gaf4190090efa5c47cb367cf97a9a519bd (https://docs.opencv.org/4.0.1/d6/dc7/group__imgproc__hist.html#gaf4190090efa5c47cb367cf97a9a519bd)

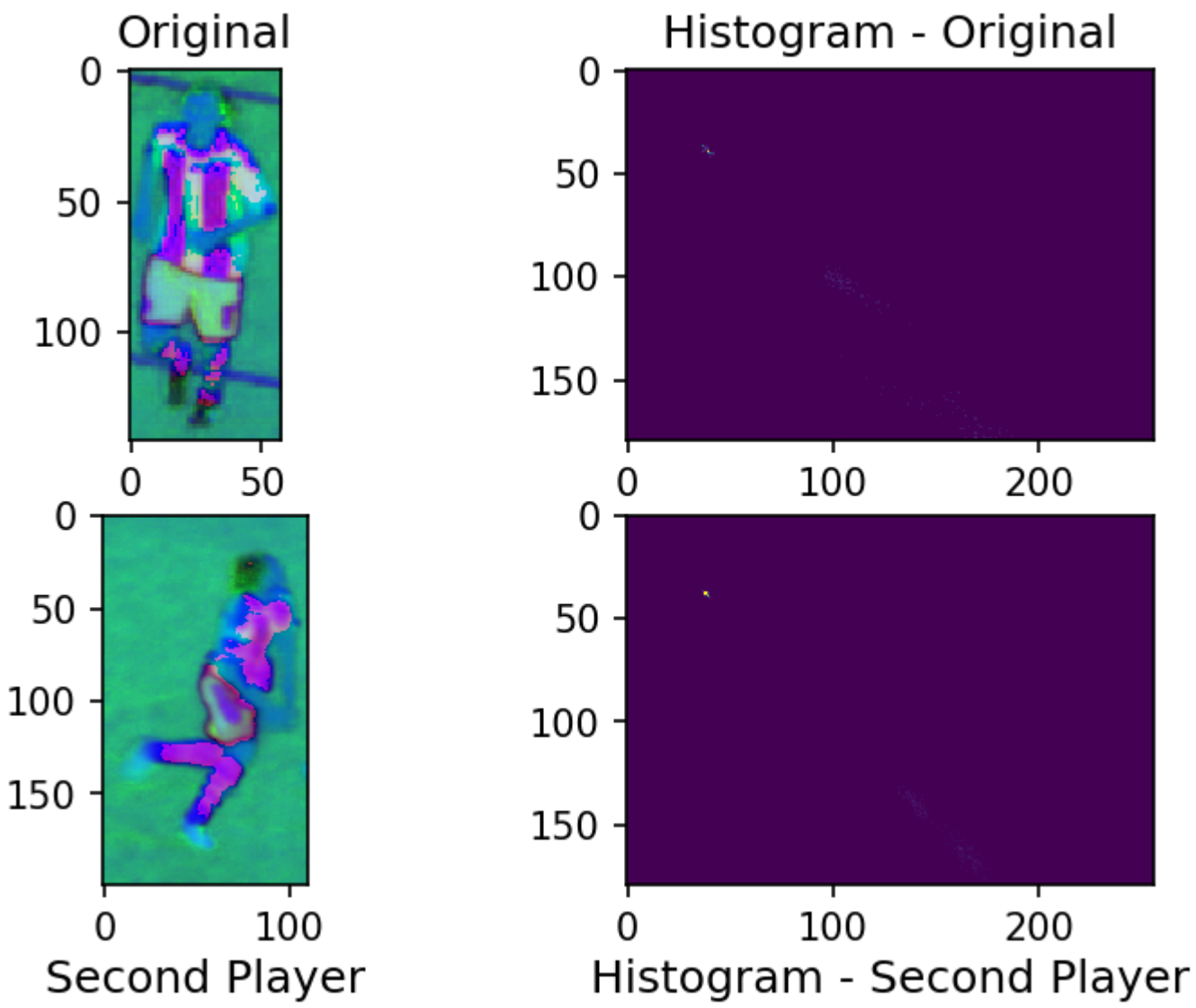
In [5]:
```
dist_bhatt = cv2.compareHist(player_histogram, player_histogram, method=cv2.HISTCMP_BHATTACHARYYA)
dist_kl = cv2.compareHist(player_histogram, player_histogram, method=cv2.HISTCMP_KL_DIV)
print('Bhat: {}, KL: {}'.format(dist_bhatt, dist_kl))
```

```
Bhat: 0.0, KL: 0.0
```

Extrahieren Sie nun einen anderen (beliebigen) Bereich im Bild und berechnen Sie darauf das Histogramm, stellen Sie es dar und vergleichen Sie es mit demjenigen des Spielers.

In [6]:
```
# Load images
image_hsv = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
player_hsv = cv2.cvtColor(player, cv2.COLOR_RGB2HSV)

# height and width
player_height, player_width, _ = player.shape

# edge = [493, 1399] # [y, x] Point of origin: Top-Left-Corner
edge = [750,60] # other player
sub_image = image_hsv[edge[0]:950, edge[1]:170]

# Calculate Histogramm
player_histogram = cv2.calcHist(player_hsv, [0, 1], None, [180, 256], [0, 180, 0, 256])
sub_image_histogram = cv2.calcHist(sub_image, [0, 1], None, [180, 256], [0, 180, 0, 256])

# Plotting images
plt.subplot(2, 2, 1)
plt.title("Original")
plt.imshow(player_hsv)
plt.subplot(2, 2, 2)
plt.title("Histogram - Original")
plt.imshow(player_histogram, interpolation = 'nearest')
plt.subplot(2, 2, 3)
plt.title("Second Player", y=-0.35)
plt.imshow(sub_image)
plt.subplot(2, 2, 4)
plt.title("Histogram - Second Player", y=-0.35)
plt.imshow(sub_image_histogram, interpolation = 'nearest')

# Calulate Bhattacharyya distance and Kullback-Leibler divergence
dist_bhatt = cv2.compareHist(player_histogram, sub_image_histogram, method=cv2.HISTCMP_BHATTACHARYYA)
dist_kl = cv2.compareHist(player_histogram, sub_image_histogram, method=cv2.HISTCMP_KL_DIV)
print('Bhat:\t\t\t{},\tKL:\t\t{}'.format(dist_bhatt, dist_kl))

dist_bhatt_kommutativ = cv2.compareHist(sub_image_histogram, player_histogram, method=cv2.HISTCMP_BHATTACHARYYA)
dist_kl_kommutativ = cv2.compareHist(sub_image_histogram, player_histogram, method=cv2.HISTCMP_KL_DIV)
print('Bhat_kommutativ:\t{},\tKL_kommutativ:\t{}'.format(dist_bhatt_kommutativ, dist_kl_kommutativ))

# Bhattacharyya distance: Histogramm-Vergleich ist kommutativ!
# Kullback-Leibler divergence: Histogramm-Vergleich ist nicht kommutativ!
```

```
Bhat:                   0.8584695412846294,     KL:             2685.87747868665
Bhat_kommutativ:        0.8584695412846294,     KL_kommutativ:  5262.78432772113
```



Sind die Histogrammvergleiche kommutativ?

- `Bhattacharyya distance`: Histogram compare is communicative!
- `Kullback-Leibler divergence`: Histogram compare is NOT communicative!

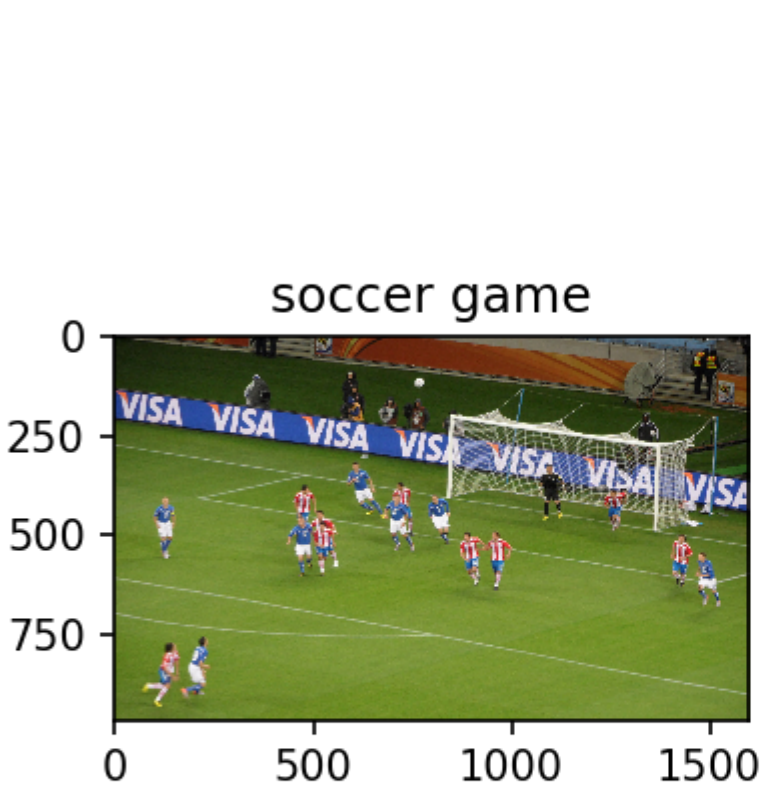
Player Detection

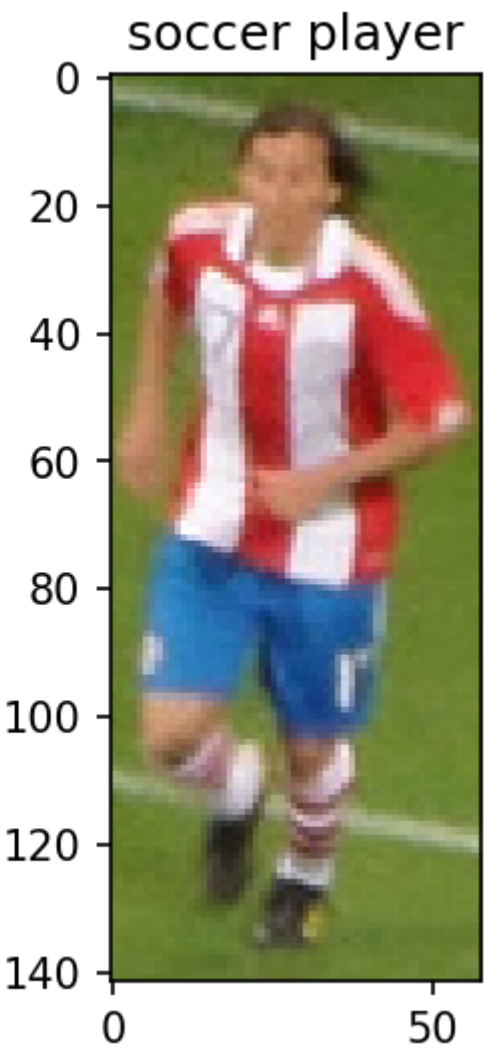
Iterieren Sie nun über das Bild und finden Sie Spieler in dem Sie das Histogramm des Spielers mit den Bildauschnitten vergleichen und den resultierenden Wert in einem Array festhalten. Der Vergleich an jedem Punkt ist wahrscheinlich (für die ersten Tests) zu aufwendig, sie könnten aber zum Beispiel jeden n-ten Punkt anschauen.

Wandeln Sie das resultierende Array in ein Bild um und stellen Sie dieses dar. Falls dazu die Auflösung im Browser zu klein ist, können Sie das Bild auch speichern (mittels `cv2.imwrite()`).

In [7]:
```
# load image
player  = cv2.imread('../data/player.jpg')
player_hsv = cv2.cvtColor(player, cv2.COLOR_BGR2HSV)
player_rgb = cv2.cvtColor(player, cv2.COLOR_BGR2RGB)
image = cv2.imread('../data/soccer-game.jpg')
image_hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# show image
plt.subplot(1, 2, 1)
plt.title("soccer game")
plt.imshow(image_rgb)
plt.subplot(1, 2, 2)
plt.title("soccer player")
plt.imshow(player_rgb)

# settings
bhat_threshold = 0.60
hit_smaller = 0
hit_bigger = 0
accuracy = 1

# list with all matchs
hit_list = list()

# size from image and player
player_height, player_width, _ = player.shape
image_height, image_width, _ = image.shape

# scanned range
range_height, range_width = 150, 50

# Calculate histogram from player
player_hsv_histogram = cv2.calcHist([player_hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])

# loop through the picture
#for y in np.arange(0, image_height, range_height):
for y in range(0, (image_height - player_height + accuracy), accuracy):
    #for x in np.arange(0, image_width, range_width):
    for x in range(0, (image_width - player_width + accuracy), accuracy): #width
        # Cut off sub_image
        sample_image = image_hsv[y:y+player_height, x:x+player_width]
        sample_histogram = cv2.calcHist([sample_image], [0,1], None, [180, 256], [0, 180, 0, 256])
        dist_bhatt = cv2.compareHist(sample_histogram, player_hsv_histogram, method=cv2.HISTCMP_BHATTACHARYYA)

        if dist_bhatt < bhat_threshold:
            hit_smaller += 1
            hit_list.append((y, y+player_height, x, x+player_width, dist_bhatt))
        else:
            hit_bigger += 1

print('smaller={}, bigger={}'.format(hit_smaller, hit_bigger))
```

```
smaller=570, bigger=1272405
```





Resultat einzeichnen

Finden Sie nun die Stelle im Bild wo das Histogramm am besten passt (`cv2.minMaxLoc(...)`)und markieren Sie diese. Dazu können Sie zum Beispiel mit `cv2.circle(...)` einen Kreis im Bild zeichnen.

In [8]:
```
image_result = image_rgb.copy()
for hit in hit_list:
    # calculate center of circle
    y = int(hit[0] + (hit[1]-hit[0])/2)
    x = int(hit[2] + (hit[3]-hit[2])/2)

    cv2.circle(image_result, (x,y), 100, (255,255,255), 10)

plt.imshow(image_result)
```

Out[8]: `<matplotlib.image.AxesImage at 0x1296060f0>`