PROJ852: Mecaruco

Release

Mar 30, 2018

Contents

CHAPTER 1

Tutorials:

1.1 Notebooks

1.1.1 Target

1.1.2 Data

Note: This notebook can be downloaded here: `aruco_basics_video.ipynb`

1.1.3 ARUCO markers: basics

1: Marker creation

```
import numpy as np
import cv2, PIL
from cv2 import aruco
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
%matplotlib nbagg
```

```
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)

fig = plt.figure()
nx = 4
ny = 3
for i in range(1, nx*ny+1):
    ax = fig.add_subplot(ny,nx, i)
    img = aruco.drawMarker(aruco_dict,i, 700)
```

```
    plt.imshow(img, cmap = mpl.cm.gray, interpolation = "nearest")
    ax.axis("off")

plt.savefig("_data/markers.jpeg")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

2: Print, cut, stick and take a picture

```
frame = cv2.imread("_data/marqueurs_chaise.jpg")
plt.figure()
plt.imshow(frame)
plt.show()
```

```
<IPython.core.display.Javascript object>
```

3: Post processing

```
%%time
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict,␣
↪parameters=parameters)
frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
```

```
Wall time: 178 ms
```

```
rejectedImgPoints[1]
```

```
array([[[ 1213.,  1229.],
        [ 1217.,  1221.],
        [ 1259.,  1224.],
        [ 1256.,  1229.]]], dtype=float32)
```

```
corners
```

```
[array([[[ 1339.,   951.],
         [ 1413.,   934.],
         [ 1434.,   981.],
         [ 1358.,   999.]]], dtype=float32), array([[[ 2247.,  1604.],
         [ 2306.,  1653.],
         [ 2263.,  1691.],
         [ 2203.,  1643.]]], dtype=float32), array([[[ 2071.,  1279.],
         [ 2101.,  1233.],
         [ 2162.,  1267.],
         [ 2132.,  1314.]]], dtype=float32), array([[[ 1209.,  1217.],
         [ 1297.,  1218.],
         [ 1290.,  1287.],
         [ 1201.,  1286.]]], dtype=float32), array([[[ 1507.,  1244.],
```

```
         [ 1510.,  1309.],
         [ 1421.,  1313.],
         [ 1419.,  1245.]]], dtype=float32), array([[[  940.,  1212.],
         [  933.,  1282.],
         [  840.,  1285.],
         [  849.,  1216.]]], dtype=float32), array([[[ 2736.,  1132.],
         [ 2764.,  1183.],
         [ 2723.,  1241.],
         [ 2701.,  1191.]]], dtype=float32), array([[[ 1140.,  1120.],
         [ 1129.,  1059.],
         [ 1214.,  1048.],
         [ 1226.,  1108.]]], dtype=float32), array([[[  990.,  1050.],
         [  906.,  1071.],
         [  885.,  1013.],
         [  968.,   993.]]], dtype=float32), array([[[ 1586.,   950.],
         [ 1513.,   929.],
         [ 1543.,   879.],
         [ 1616.,   899.]]], dtype=float32)]
```

Pretty fast processing !

4: Results

```
plt.figure()
plt.imshow(frame_markers, origin = "upper")
if ids is not None:
    for i in range(len(ids)):
        c = corners[i][0]
        plt.plot([c[:, 0].mean()], [c[:, 1].mean()], "+", label = "id={0}".
↪format(ids[i]))
"""for points in rejectedImgPoints:
    y = points[:, 0]
    x = points[:, 1]
    plt.plot(x, y, ".m-", linewidth = 1.)"""
plt.legend()
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
def quad_area(data):
    l = data.shape[0]//2
    corners = data[["c1", "c2", "c3", "c4"]].values.reshape(l, 2,4)
    c1 = corners[:, :, 0]
    c2 = corners[:, :, 1]
    c3 = corners[:, :, 2]
    c4 = corners[:, :, 3]
    e1 = c2-c1
    e2 = c3-c2
    e3 = c4-c3
    e4 = c1-c4
    a = -.5 * (np.cross(-e1, e2, axis = 1) + np.cross(-e3, e4, axis = 1))
    return a

corners2 = np.array([c[0] for c in corners])

data = pd.DataFrame({"x": corners2[:,:,0].flatten(), "y": corners2[:,:,1].flatten()},
```

```
                    index = pd.MultiIndex.from_product(
                            [ids.flatten(), ["c{0}".format(i )for i in np.
↪arange(4)+1]],
                        names = ["marker", ""] ))

data = data.unstack().swaplevel(0, 1, axis = 1).stack()
data["m1"] = data[["c1", "c2"]].mean(axis = 1)
data["m2"] = data[["c2", "c3"]].mean(axis = 1)
data["m3"] = data[["c3", "c4"]].mean(axis = 1)
data["m4"] = data[["c4", "c1"]].mean(axis = 1)
data["o"] = data[["m1", "m2", "m3", "m4"]].mean(axis = 1)
data
```

```
# Plante un peu...
"""cap = cv2.VideoCapture('_data/AeroTrain.mp4')
while(cap.isOpened()):
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()"""
```

```
"cap = cv2.VideoCapture('_data/AeroTrain.mp4')nwhile(cap.isOpened()):n    ret,
↪ frame = cap.read()nn    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)nn␣
↪ cv2.imshow('frame',gray)n    if cv2.waitKey(1) & 0xFF == ord('q'):n␣
↪breaknncap.release()ncv2.destroyAllWindows()"
```

```
cap = cv2.VideoCapture('_data/AeroTrain.mp4')
nframe = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))

print("nframe =", nframe)
cap.set(1, 300) # arguments: 1: laisser, 2: numéro du frame
ret, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
plt.figure()
plt.imshow(gray)
plt.show()
cap.release()
```

```
nframe = 712
```

```
<IPython.core.display.Javascript object>
```

```
%%time
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict,␣
↪parameters=parameters)
frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
```

```
Wall time: 31.3 ms
```

```
plt.figure()
plt.imshow(frame_markers, origin = "upper")
if ids is not None:
    for i in range(len(ids)):
        c = corners[i][0]
        plt.plot([c[:, 0].mean()], [c[:, 1].mean()], "+", label = "id={0}".
↪format(ids[i]))
"""for points in rejectedImgPoints:
    y = points[:, 0]
    x = points[:, 1]
    plt.plot(x, y, ".m-", linewidth = 1.)"""
plt.legend()
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
help(aruco.DetectorParameters_create)
```

```
Help on built-in function DetectorParameters_create:

DetectorParameters_create(...)
    DetectorParameters_create() -> retval
    .
```

Note: This notebook can be downloaded here: `Aruco_detection_direct.ipynb`

```
import numpy as np
import cv2
import cv2.aruco as aruco

aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
img = aruco.drawMarker(aruco_dict, 2, 700)
cv2.imwrite("test_marker.jpg", img)


cv2.waitKey(0)
cv2.destroyAllWindows()

cap = cv2.VideoCapture(0)

while(True):

    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
    parameters =  aruco.DetectorParameters_create()



    '''    detectMarkers(...)
        detectMarkers(image, dictionary[, corners[, ids[, parameters[, rejectedI
        mgPoints]]]]) -> corners, ids, rejectedImgPoints
```

```
        '''

    corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict,␣
↪parameters=parameters)

    frame = aruco.drawDetectedMarkers(frame, corners)

    cv2.imshow('frame',frame)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

Note: This notebook can be downloaded here: `video_to_image.ipynb`

1.1.4 Video to image

```
import numpy as np
import cv2, PIL, os
from cv2 import aruco
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
%matplotlib nbagg
```

```
workdir = "./data/"
name = "VID_20180314_141424.mp4"
rootname = name.split(".")[0]
cap = cv2.VideoCapture(workdir + name)
counter = 0
each = 5
length = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
for i in range(length):
    ret, frame = cap.read()
    if i % each == 0: cv2.imwrite(workdir + rootname + "_{0}".format(i) + ".png",␣
↪frame)

cap.release()
```

```
os.listdir("data/")
```

```
['IMG_20180307_091159.jpg',
 'VID_20180314_141424_290.png',
 'VID_20180314_141424_335.png',
 'VID_20180314_141424_175.png',
 'VID_20180314_141424_260.png',
 'VID_20180314_141424_370.png',
 'VID_20180314_141424_30.png',
```

```
'VID_20180314_141424_5.png',
'markers.pdf',
'VID_20180314_141424_320.png',
'VID_20180314_141424_150.png',
'VID_20180314_141424_265.png',
'VID_20180314_141424_210.png',
'VID_20180314_141424_85.png',
'VID_20180314_141424_250.png',
'VID_20180314_141424_165.png',
'VID_20180314_141424_255.png',
'VID_20180314_141424_55.png',
'VID_20180314_141424_195.png',
'VID_20180314_141424_200.png',
'VID_20180314_141424_60.png',
'IMG_20180307_091235.jpg',
'VID_20180314_141424_80.png',
'VID_20180314_141424_215.png',
'VID_20180314_141424_205.png',
'VID_20180314_141424_305.png',
'VID_20180314_141424_70.png',
'VID_20180314_141424_315.png',
'VID_20180314_141424_65.png',
'VID_20180314_141424_380.png',
'VID_20180314_141424_15.png',
'IMG_20180307_091210.jpg',
'VID_20180314_141424_45.png',
'VID_20180314_141424_240.png',
'VID_20180314_141424_35.png',
'VID_20180314_141424_330.png',
'IMG_20180307_091226.jpg',
'VID_20180314_141424_180.png',
'VID_20180314_141424_130.png',
'IMG_20180307_091203.jpg',
'VID_20180314_141424_390.png',
'VID_20180314_141424_120.png',
'VID_20180314_141424_400.png',
'VID_20180314_141424_155.png',
'VID_20180314_141424_220.png',
'VID_20180314_141424_360.png',
'chessboard.pdf',
'VID_20180314_141424_300.png',
'VID_20180314_141424_235.png',
'VID_20180314_141424_365.png',
'VID_20180314_141424_345.png',
'VID_20180314_141424_340.png',
'VID_20180314_141424_355.png',
'VID_20180314_141424_20.png',
'IMG_20180307_091217.jpg',
'VID_20180314_141424_115.png',
'VID_20180314_141424_185.png',
'VID_20180314_141424_245.png',
'VID_20180314_141424_105.png',
'VID_20180314_141424_310.png',
'IMG_20180307_091220.jpg',
'VID_20180314_141424.mp4',
'VID_20180314_141424_10.png',
'VID_20180314_141424_25.png',
'VID_20180314_141424_140.png',
```

```
 'VID_20180314_141424_40.png',
 'VID_20180314_141424_270.png',
 'VID_20180314_141424_100.png',
 'VID_20180314_141424_110.png',
 'VID_20180314_141424_295.png',
 'VID_20180314_141424_375.png',
 'IMG_20180307_091229.jpg',
 'VID_20180314_141424_160.png',
 'VID_20180314_141424_405.png',
 'VID_20180314_141424_135.png',
 'VID_20180314_141424_395.png',
 'VID_20180314_141424_50.png',
 'VID_20180314_141424_125.png',
 'VID_20180314_141424_275.png',
 'VID_20180314_141424_0.png',
 'VID_20180314_141424_285.png',
 'VID_20180314_141424_280.png',
 'VID_20180314_141424_95.png',
 'VID_20180314_141424_75.png',
 'VID_20180314_141424_90.png',
 'IMG_20180307_091213.jpg',
 'VID_20180314_141424_145.png',
 'VID_20180314_141424_350.png',
 'VID_20180314_141424_190.png',
 'VID_20180314_141424_230.png',
 'VID_20180314_141424_225.png',
 'VID_20180314_141424_385.png',
 'VID_20180314_141424_170.png',
 'VID_20180314_141424_325.png']
```

```
int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
```

```
408
```

Note: This notebook can be downloaded here: `aruco_calibration.ipynb`

1.1.5 Camera calibration using CHARUCO

```
import numpy as np
import cv2, PIL, os
from cv2 import aruco
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
%matplotlib nbagg
```

1. Marker dictionary creation

```
workdir = "data/"
```

```
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)

fig = plt.figure()
nx = 8
ny = 6
for i in range(1, nx*ny+1):
    ax = fig.add_subplot(ny,nx, i)
    img = aruco.drawMarker(aruco_dict,i, 700)
    plt.imshow(img, cmap = mpl.cm.gray, interpolation = "nearest")
    ax.axis("off")

plt.savefig(workdir + "markers.pdf")
plt.show()
#plt.close()
```

```
<IPython.core.display.Javascript object>
```

2. Camera pose estimation using CHARUCO chessboard

First, let's create the board.

```
board = aruco.CharucoBoard_create(11, 8, 10, 7, aruco_dict)
imboard = board.draw((500, 500))
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
plt.imshow(imboard, cmap = mpl.cm.gray, interpolation = "nearest")
ax.axis("off")
plt.savefig(workdir + "chessboard.pdf")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

And take photos of it from multiple angles, for example:

```
images = [workdir + f for f in os.listdir(workdir) if f.endswith(".png") and f.
↪startswith("VID_20180314_141424")]

im = PIL.Image.open(images[10])
fig = plt.figure()
ax = fig.add_subplot(1,1,1)
plt.imshow(im)
#ax.axis('off')
plt.show()
```

```
<IPython.core.display.Javascript object>
```

Now, the camera calibration can be done using all the images of the chessboard. Two functions are necessary:

- The first will detect markers on all the images and.
- The second will proceed the detected markers to estimage the camera calibration data.

```
def read_chessboards(images):
    """
    Charuco base pose estimation.
    """
```

```
    print("POSE ESTIMATION STARTS:")
    allCorners = []
    allIds = []
    decimator = 0
    # SUB PIXEL CORNER DETECTION CRITERION
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.0001)

    for im in images:
        print("=> Processing image {0}".format(im))
        frame = cv2.imread(im)
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners, ids, rejectedImgPoints = cv2.aruco.detectMarkers(gray, aruco_dict)

        if len(corners)>0:
            # SUB PIXEL DETECTION
            for corner in corners:
                cv2.cornerSubPix(gray, corner,
                                 winSize = (20,20),
                                 zeroZone = (-1,-1),
                                 criteria = criteria)
            res2 = cv2.aruco.interpolateCornersCharuco(corners,ids,gray,board)
            if res2[1] is not None and res2[2] is not None and len(res2[1])>3 and␣
↪decimator%1==0:
                allCorners.append(res2[1])
                allIds.append(res2[2])

        decimator+=1

    imsize = gray.shape
    return allCorners,allIds,imsize
```

```
#%%time

allCorners,allIds,imsize=read_chessboards(images)
```

```
POSE ESTIMATION STARTS:
=> Processing image data/VID_20180314_141424_290.png
=> Processing image data/VID_20180314_141424_335.png
=> Processing image data/VID_20180314_141424_175.png
=> Processing image data/VID_20180314_141424_260.png
=> Processing image data/VID_20180314_141424_370.png
=> Processing image data/VID_20180314_141424_30.png
=> Processing image data/VID_20180314_141424_5.png
=> Processing image data/VID_20180314_141424_320.png
=> Processing image data/VID_20180314_141424_150.png
=> Processing image data/VID_20180314_141424_265.png
=> Processing image data/VID_20180314_141424_210.png
=> Processing image data/VID_20180314_141424_85.png
=> Processing image data/VID_20180314_141424_250.png
=> Processing image data/VID_20180314_141424_165.png
=> Processing image data/VID_20180314_141424_255.png
=> Processing image data/VID_20180314_141424_55.png
=> Processing image data/VID_20180314_141424_195.png
=> Processing image data/VID_20180314_141424_200.png
=> Processing image data/VID_20180314_141424_60.png
=> Processing image data/VID_20180314_141424_80.png
=> Processing image data/VID_20180314_141424_215.png
```

```
=> Processing image data/VID_20180314_141424_205.png
=> Processing image data/VID_20180314_141424_305.png
=> Processing image data/VID_20180314_141424_70.png
=> Processing image data/VID_20180314_141424_315.png
=> Processing image data/VID_20180314_141424_65.png
=> Processing image data/VID_20180314_141424_380.png
=> Processing image data/VID_20180314_141424_15.png
=> Processing image data/VID_20180314_141424_45.png
=> Processing image data/VID_20180314_141424_240.png
=> Processing image data/VID_20180314_141424_35.png
=> Processing image data/VID_20180314_141424_330.png
=> Processing image data/VID_20180314_141424_180.png
=> Processing image data/VID_20180314_141424_130.png
=> Processing image data/VID_20180314_141424_390.png
=> Processing image data/VID_20180314_141424_120.png
=> Processing image data/VID_20180314_141424_400.png
=> Processing image data/VID_20180314_141424_155.png
=> Processing image data/VID_20180314_141424_220.png
=> Processing image data/VID_20180314_141424_360.png
=> Processing image data/VID_20180314_141424_300.png
=> Processing image data/VID_20180314_141424_235.png
=> Processing image data/VID_20180314_141424_365.png
=> Processing image data/VID_20180314_141424_345.png
=> Processing image data/VID_20180314_141424_340.png
=> Processing image data/VID_20180314_141424_355.png
=> Processing image data/VID_20180314_141424_20.png
=> Processing image data/VID_20180314_141424_115.png
=> Processing image data/VID_20180314_141424_185.png
=> Processing image data/VID_20180314_141424_245.png
=> Processing image data/VID_20180314_141424_105.png
=> Processing image data/VID_20180314_141424_310.png
=> Processing image data/VID_20180314_141424_10.png
=> Processing image data/VID_20180314_141424_25.png
=> Processing image data/VID_20180314_141424_140.png
=> Processing image data/VID_20180314_141424_40.png
=> Processing image data/VID_20180314_141424_270.png
=> Processing image data/VID_20180314_141424_100.png
=> Processing image data/VID_20180314_141424_110.png
=> Processing image data/VID_20180314_141424_295.png
=> Processing image data/VID_20180314_141424_375.png
=> Processing image data/VID_20180314_141424_160.png
=> Processing image data/VID_20180314_141424_405.png
=> Processing image data/VID_20180314_141424_135.png
=> Processing image data/VID_20180314_141424_395.png
=> Processing image data/VID_20180314_141424_50.png
=> Processing image data/VID_20180314_141424_125.png
=> Processing image data/VID_20180314_141424_275.png
=> Processing image data/VID_20180314_141424_0.png
=> Processing image data/VID_20180314_141424_285.png
=> Processing image data/VID_20180314_141424_280.png
=> Processing image data/VID_20180314_141424_95.png
=> Processing image data/VID_20180314_141424_75.png
=> Processing image data/VID_20180314_141424_90.png
=> Processing image data/VID_20180314_141424_145.png
=> Processing image data/VID_20180314_141424_350.png
=> Processing image data/VID_20180314_141424_190.png
=> Processing image data/VID_20180314_141424_230.png
=> Processing image data/VID_20180314_141424_225.png
```

```
=> Processing image data/VID_20180314_141424_385.png
=> Processing image data/VID_20180314_141424_170.png
=> Processing image data/VID_20180314_141424_325.png
```

```
def calibrate_camera(allCorners,allIds,imsize):
    """
    Calibrates the camera using the dected corners.
    """
    print("CAMERA CALIBRATION")

    cameraMatrixInit = np.array([[ 2000.,    0., imsize[0]/2.],
                                 [    0., 2000., imsize[1]/2.],
                                 [    0.,    0.,           1.]])

    distCoeffsInit = np.zeros((5,1))
    flags = (cv2.CALIB_USE_INTRINSIC_GUESS + cv2.CALIB_RATIONAL_MODEL)
    (ret, camera_matrix, distortion_coefficients0,
     rotation_vectors, translation_vectors,
     stdDeviationsIntrinsics, stdDeviationsExtrinsics,
     perViewErrors) = cv2.aruco.calibrateCameraCharucoExtended(
                      charucoCorners=allCorners,
                      charucoIds=allIds,
                      board=board,
                      imageSize=imsize,
                      cameraMatrix=cameraMatrixInit,
                      distCoeffs=distCoeffsInit,
                      flags=flags,
                      criteria=(cv2.TERM_CRITERIA_EPS & cv2.TERM_CRITERIA_COUNT,␣
↪10000, 1e-9))

    return ret, camera_matrix, distortion_coefficients0, rotation_vectors,␣
↪translation_vectors
```

```
%time ret, mtx, dist, rvecs, tvecs = calibrate_camera(allCorners,allIds,imsize)
```

```
CAMERA CALIBRATION
CPU times: user 11.3 s, sys: 10.9 s, total: 22.2 s
Wall time: 5.78 s
```

```
ret
```

```
10.507478602319868
```

```
mtx
```

```
array([[1.82907422e+03, 0.00000000e+00, 9.70018381e+02],
       [0.00000000e+00, 1.82396375e+03, 5.64679336e+02],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00]])
```

```
dist
```

```
array([[ 5.77647646e+00],
       [-1.19867510e+02],
       [ 2.81138209e-03],
```

```
       [ 2.04931051e-02],
       [ 1.36665086e+03],
       [ 5.10336618e+00],
       [-1.13575351e+02],
       [ 1.33870881e+03],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00],
       [ 0.00000000e+00]])
```

Check calibration results

```
i=3 # select image id
plt.figure()
frame = cv2.imread(images[i])
img_undist = cv2.undistort(frame,mtx,dist,None)
plt.subplot(1,2,1)
plt.imshow(frame)
plt.title("Raw image")
plt.axis("off")
plt.subplot(1,2,2)
plt.imshow(img_undist)
plt.title("Corrected image")
plt.axis("off")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

3 . Use of camera calibration to estimate 3D translation and rotation of each marker on a scene

```
frame = cv2.imread("data/VID_20180314_141424_335.png")
#frame = cv2.undistort(src = frame, cameraMatrix = mtx, distCoeffs = dist)
plt.figure()
plt.imshow(frame, interpolation = "nearest")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

Post processing

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict,
                                                      parameters=parameters)
# SUB PIXEL DETECTION
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.0001)
for corner in corners:
    cv2.cornerSubPix(gray, corner, winSize = (20,20), zeroZone = (-1,-1), criteria =␣
↪criteria)
```

```
frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)

corners
```

```
[array([[[ 621.     ,  972.     ],
         [ 701.     ,  969.     ],
         [ 709.82715, 1049.7007 ],
         [ 619.     , 1055.     ]]], dtype=float32),
 array([[[ 847.2396 ,  945.72266],
         [ 969.38995,  943.80817],
         [ 971.123  , 1068.5813 ],
         [ 846.4021 , 1070.483  ]]], dtype=float32),
 array([[[244.78148, 828.47906],
         [341.     , 845.     ],
         [352.81595, 951.9142 ],
         [233.71716, 952.73145]]], dtype=float32),
 array([[[726.8156 , 822.9357 ],
         [848.2925 , 820.77747],
         [847.2396 , 945.72266],
         [723.4202 , 948.4776 ]]], dtype=float32),
 array([[[1464.    ,  834.    ],
         [1549.7858,  817.7192],
         [1562.2096,  936.4745],
         [1474.44  ,  916.0094]]], dtype=float32),
 array([[[1202.7676 ,  817.2212 ],
         [1318.0548 ,  817.4417 ],
         [1326.7908 ,  939.2189 ],
         [1209.4231 ,  940.84424]]], dtype=float32),
 array([[[1191.0787 ,  587.5763 ],
         [1302.4027 ,  587.67444],
         [1309.4077 ,  696.56885],
         [1197.1561 ,  698.04144]]], dtype=float32),
 array([[[1093.    ,  500.    ],
         [1167.    ,  498.    ],
         [1191.0786,  587.5763],
         [1076.532 ,  588.4834]]], dtype=float32),
 array([[[284.72723, 374.40637],
         [392.8324 , 372.20898],
         [385.58038, 478.16342],
         [275.34912, 479.37204]]], dtype=float32),
 array([[[1423.     ,  388.     ],
         [1505.6632 ,  376.44638],
         [1517.4326 ,  474.12766],
         [1410.2745 ,  476.39386]]], dtype=float32),
 array([[[1067.2129 ,  269.61798],
         [1177.2119 ,  269.69016],
         [1181.5264 ,  374.11893],
         [1070.249  ,  374.3764 ]]], dtype=float32),
 array([[[849.6194 , 268.4913 ],
         [960.5578 , 269.36526],
         [962.0218 , 374.2559 ],
         [849.7227 , 373.83377]]], dtype=float32),
 array([[[627.9509 , 266.41553],
         [738.15485, 267.64926],
         [736.33246, 373.41757],
         [624.7238 , 372.56445]]], dtype=float32),
 array([[[ 961.74927,  173.74501],
```

```
        [1061.6177 ,  174.64995],
        [1067.2128 ,  269.61798],
        [ 960.5578 ,  269.36523]]], dtype=float32),
array([[[742.5307 , 172.02827],
        [846.7603 , 172.7219 ],
        [849.6194 , 268.49127],
        [738.15485, 267.64923]]], dtype=float32),
array([[[318.66827, 184.43867],
        [406.767  , 168.85188],
        [400.8201 , 264.7916 ],
        [293.67133, 265.81296]]], dtype=float32),
array([[[1087.626  ,  942.2079 ],
        [1209.4231 ,  940.84424],
        [1215.7225 , 1065.0857 ],
        [1092.1993 , 1066.6892 ]]], dtype=float32),
array([[[ 967.6742 ,  819.2523 ],
        [1083.6823 ,  818.21716],
        [1087.626  ,  942.2079 ],
        [ 969.38995,  943.80817]]], dtype=float32),
array([[[ 374.     ,  971.     ],
        [ 477.05597,  951.8135 ],
        [ 468.81622, 1075.2206 ],
        [ 371.48587, 1055.1322 ]]], dtype=float32),
array([[[1350.    ,  959.    ],
        [1448.153 ,  939.1731],
        [1457.3159, 1061.4172],
        [1360.1149, 1041.5824]]], dtype=float32),
array([[[483.96588, 826.2725 ],
        [607.66144, 824.7236 ],
        [602.3229 , 950.1209 ],
        [477.05603, 951.8134 ]]], dtype=float32),
array([[[393.4445, 730.7826],
        [471.    , 726.    ],
        [483.9658, 826.2725],
        [385.    , 809.    ]]], dtype=float32),
array([[[630.     , 725.     ],
        [711.     , 724.     ],
        [726.81555, 822.93567],
        [607.66144, 824.7235 ]]], dtype=float32),
array([[[867.     , 722.     ],
        [949.     , 722.     ],
        [967.67413, 819.2523 ],
        [848.2925 , 820.77747]]], dtype=float32),
array([[[1098.     ,  720.     ],
        [1189.615  ,  721.7594 ],
        [1202.7676 ,  817.2212 ],
        [1083.6823 ,  818.21716]]], dtype=float32),
array([[[1333.    ,  719.    ],
        [1418.965 ,  725.754 ],
        [1437.3319,  815.5953],
        [1343.6265,  809.7281]]], dtype=float32),
array([[[265.05103, 594.3574 ],
        [378.26273, 592.7059 ],
        [369.28116, 705.2608 ],
        [254.763  , 706.3264 ]]], dtype=float32),
array([[[498.10806, 592.1523 ],
        [617.7701 , 591.5943 ],
        [612.09393, 703.04694],
```

```
        [491.22415, 704.63715]]], dtype=float32),
array([[[732.6115 , 591.00323],
        [849.3975 , 589.73444],
        [848.42834, 700.7467 ],
        [730.0566 , 702.22296]]], dtype=float32),
array([[[ 964.55273,  589.08484],
        [1076.532  ,  588.4834 ],
        [1079.3727 ,  698.8115 ],
        [ 966.80176,  699.71594]]], dtype=float32),
array([[[1441.     ,  603.     ],
        [1528.1691 ,  588.53204],
        [1537.8229 ,  695.71027],
        [1458.9944 ,  685.63495]]], dtype=float32),
array([[[638.    , 500.    ],
        [717.    , 499.    ],
        [732.6115, 591.0032],
        [617.7701, 591.5943]]], dtype=float32),
array([[[867.     , 499.     ],
        [963.7391 , 478.3379 ],
        [964.5527 , 589.0847 ],
        [849.39746, 589.73444]]], dtype=float32),
array([[[406.     , 499.     ],
        [494.02493, 502.7837 ],
        [498.10797, 592.1522 ],
        [401.     , 575.     ]]], dtype=float32),
array([[[1317.     ,  497.     ],
        [1410.2745 ,  476.39386],
        [1418.0753 ,  585.7022 ],
        [1323.     ,  570.     ]]], dtype=float32),
array([[[1181.5265 ,  374.11902],
        [1288.7448 ,  375.0746 ],
        [1294.9125 ,  476.00323],
        [1186.5801 ,  477.40228]]], dtype=float32),
array([[[ 962.0219 ,  374.25592],
        [1070.2489 ,  374.37643],
        [1072.9188 ,  478.17056],
        [ 963.7391 ,  478.3379 ]]], dtype=float32),
array([[[736.33246, 373.41754],
        [849.7227 , 373.8338 ],
        [849.0077 , 478.46964],
        [735.2992 , 478.96436]]], dtype=float32),
array([[[508.6434 , 371.5545 ],
        [624.7238 , 372.56445],
        [620.7678 , 478.14   ],
        [503.92432, 477.97134]]], dtype=float32),
array([[[1304.     ,  285.     ],
        [1394.4258 ,  269.19562],
        [1400.6348 ,  372.73566],
        [1309.     ,  358.     ]]], dtype=float32),
array([[[421.     , 281.     ],
        [514.4226 , 265.67953],
        [508.6434 , 371.5545 ],
        [415.     , 355.     ]]], dtype=float32),
array([[[1407.     ,  185.     ],
        [1474.6414 ,  186.77351],
        [1497.5245 ,  266.7342 ],
        [1394.4258 ,  269.19562]]], dtype=float32),
array([[[523.3045 , 170.48518],
```

```
        [628.4742 , 171.3262 ],
        [627.9508 , 266.4155 ],
        [514.4226 , 265.67953]]], dtype=float32)]
```

Very fast processing !

Results

```
plt.figure()
plt.imshow(frame_markers, interpolation = "nearest")

plt.show()
```

```
<IPython.core.display.Javascript object>
```

Add local axis on each marker

```
size_of_marker =  0.015 # side lenght of the marker in meter
rvecs,tvecs = aruco.estimatePoseSingleMarkers(corners, size_of_marker , mtx, dist)
```

```
length_of_axis = 0.01
imaxis = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
for i in range(len(tvecs)):
    imaxis = aruco.drawAxis(imaxis, mtx, dist, rvecs[i], tvecs[i], length_of_axis)
```

```
plt.figure()
plt.imshow(imaxis)
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
data = pd.DataFrame(data = tvecs.reshape(len(tvecs),3), columns = ["tx", "ty", "tz"],
                    index = ids.flatten())
data.index.name = "marker"
data.sort_index(inplace= True)
data
```

```
v = data.loc[:6].values
((v[1:] - v[:-1])**2).sum(axis = 1)**.5
```

```
array([0.03353885, 0.03033577, 0.0326638 , 0.09532321, 0.16666463])
```

```
fig = plt.figure()
#ax = fig.add_subplot(111, projection='3d')
ax = fig.add_subplot(1,2,1)
ax.set_aspect("equal")
plt.plot(data.tx, data.ty, "or-")
plt.grid()
ax = fig.add_subplot(1,2,2)
plt.imshow(imaxis, origin = "lower")
```

```
plt.plot(np.array(corners)[:, 0, 0,0], np.array(corners)[:, 0, 0,1], "or")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
fig = plt.figure()


plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
a = np.arange(50)
a
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

```
import pickle
```

```
f = open("truc.pckl", "wb")
pickle.dump(a, f)
f.close()
```

```
f = open("truc.pckl", "rb")
b = pickle.load(f)
b == a
```

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True,  True,  True,  True,  True,
        True,  True,  True,  True,  True], dtype=bool)
```

```
corners = np.array(corners)
data2 = pd.DataFrame({"px": corners[:, 0, 0, 1],
                      "py": corners[:, 0, 0, 0]}, index = ids.flatten())
data2.sort_index(inplace=True)
data2
```

```
m0 = data2.loc[0]
m43 = data2.loc[43]
d01 = ((m0 - m43).values**2).sum()**.5
d = 42.5e-3 * (3.5**2 + 4.5**2)**.5
factor = d / d01
data2["x"] = data2.px * factor
data2["y"] = data2.py * factor
((data2[["x", "y"]].loc[11] - data2[["x", "y"]].loc[0]).values**2).sum()**.5
```

```
0.043476117957396747
```

```
c = np.array(corners).astype(np.float64).reshape(44,4,2)
(((c[:, 1:] - c[:, :-1])**2).sum(axis = 2)**.5).mean(axis =1)
```

```
array([ 138.33575835,  143.00113377,  142.012097  ,  140.69699432,
        146.66782406,  144.02442319,  138.67845434,  142.33812925,
        143.00229095,  140.33926025,  140.35356753,  146.66786569,
        139.34054504,  146.67222201,  140.03570454,  148.01939184,
        143.35647769,  142.67236143,  147.01931296,  148.02127735,
        137.67392157,  135.35308209,  141.00354688,  143.67946992,
        137.67149733,  138.67392207,  145.00112611,  142.33454105,
        138.3466791 ,  143.00234925,  139.0035972 ,  143.00115739,
        143.6865917 ,  144.67964727,  144.33446711,  141.67253496,
        143.67117097,  147.67232772,  150.35663387,  141.70034559,
        149.01342342,  146.01949591,  144.34013329,  150.35333222])
```

```
c
```

```
array([[[ 2406.,  1940.],
        [ 2546.,  1940.],
        [ 2545.,  2075.],
        [ 2405.,  2076.]],

       [[ 1991.,  1938.],
        [ 2138.,  1939.],
        [ 2138.,  2076.],
        [ 1993.,  2076.]],

       [[ 1584.,  1936.],
        [ 1728.,  1936.],
        [ 1731.,  2073.],
        [ 1586.,  2072.]],

       [[ 2619.,  1735.],
        [ 2759.,  1735.],
        [ 2754.,  1878.],
        [ 2615.,  1877.]],

       [[ 2198.,  1734.],
        [ 2347.,  1734.],
        [ 2346.,  1878.],
        [ 2199.,  1878.]],

       [[  973.,  1733.],
        [ 1117.,  1731.],
        [ 1121.,  1874.],
        [  976.,  1875.]],

       [[  572.,  1732.],
        [  710.,  1732.],
        [  713.,  1874.],
        [  577.,  1873.]],

       [[ 2410.,  1533.],
        [ 2554.,  1533.],
        [ 2552.,  1672.],
        [ 2408.,  1672.]],
```

```
       [[ 1373.,  1326.],
        [ 1519.,  1325.],
        [ 1519.,  1463.],
        [ 1374.,  1464.]],

       [[ 1785.,  1326.],
        [ 1926.,  1324.],
        [ 1927.,  1463.],
        [ 1786.,  1463.]],

       [[ 2627.,  1323.],
        [ 2767.,  1324.],
        [ 2763.,  1464.],
        [ 2622.,  1464.]],

       [[ 2200.,  1324.],
        [ 2350.,  1324.],
        [ 2349.,  1463.],
        [ 2198.,  1463.]],

       [[  760.,  1128.],
        [  901.,  1127.],
        [  903.,  1265.],
        [  764.,  1266.]],

       [[ 1988.,  1123.],
        [ 2138.,  1121.],
        [ 2138.,  1261.],
        [ 1988.,  1262.]],

       [[  547.,   920.],
        [  687.,   918.],
        [  692.,  1058.],
        [  552.,  1059.]],

       [[ 2203.,   910.],
        [ 2354.,   908.],
        [ 2351.,  1050.],
        [ 2200.,  1052.]],

       [[ 2631.,   908.],
        [ 2775.,   906.],
        [ 2771.,  1050.],
        [ 2629.,  1050.]],

       [[  750.,   708.],
        [  890.,   707.],
        [  892.,   855.],
        [  752.,   855.]],

       [[ 2419.,   695.],
        [ 2565.,   693.],
        [ 2563.,   842.],
        [ 2417.,   845.]],

       [[  946.,   494.],
        [ 1093.,   491.],
        [ 1096.,   642.],
```

```
        [  950.,   643.]],

       [[ 1181.,  1936.],
        [ 1319.,  1935.],
        [ 1321.,  2073.],
        [ 1184.,  2072.]],

       [[  780.,  1935.],
        [  916.,  1935.],
        [  920.,  2070.],
        [  785.,  2070.]],

       [[ 1788.,  1731.],
        [ 1928.,  1732.],
        [ 1929.,  1876.],
        [ 1790.,  1875.]],

       [[ 1378.,  1731.],
        [ 1521.,  1730.],
        [ 1524.,  1873.],
        [ 1379.,  1874.]],

       [[  771.,  1533.],
        [  909.,  1533.],
        [  911.,  1671.],
        [  774.,  1671.]],

       [[ 1176.,  1533.],
        [ 1315.,  1532.],
        [ 1317.,  1669.],
        [ 1177.,  1670.]],

       [[ 1989.,  1532.],
        [ 2137.,  1532.],
        [ 2137.,  1671.],
        [ 1989.,  1670.]],

       [[ 1581.,  1531.],
        [ 1726.,  1531.],
        [ 1727.,  1669.],
        [ 1583.,  1669.]],

       [[  560.,  1329.],
        [  700.,  1328.],
        [  703.,  1465.],
        [  565.,  1466.]],

       [[  966.,  1328.],
        [ 1112.,  1327.],
        [ 1113.,  1465.],
        [  968.,  1465.]],

       [[ 1169.,  1127.],
        [ 1309.,  1126.],
        [ 1310.,  1264.],
        [ 1171.,  1265.]],

       [[ 1579.,  1124.],
```

```
        [ 1723.,  1123.],
        [ 1723.,  1263.],
        [ 1578.,  1263.]],

       [[ 2415.,  1120.],
        [ 2560.,  1119.],
        [ 2556.,  1261.],
        [ 2412.,  1261.]],

       [[  956.,   919.],
        [ 1103.,   918.],
        [ 1106.,  1058.],
        [  959.,  1059.]],

       [[ 1367.,   917.],
        [ 1514.,   916.],
        [ 1514.,  1056.],
        [ 1368.,  1056.]],

       [[ 1784.,   914.],
        [ 1926.,   912.],
        [ 1926.,  1053.],
        [ 1784.,  1054.]],

       [[ 1160.,   706.],
        [ 1302.,   706.],
        [ 1304.,   854.],
        [ 1163.,   854.]],

       [[ 1574.,   703.],
        [ 1722.,   702.],
        [ 1722.,   850.],
        [ 1575.,   852.]],

       [[ 1991.,   699.],
        [ 2142.,   697.],
        [ 2138.,   847.],
        [ 1988.,   848.]],

       [[  539.,   499.],
        [  677.,   496.],
        [  681.,   644.],
        [  542.,   646.]],

       [[ 1360.,   490.],
        [ 1508.,   488.],
        [ 1510.,   639.],
        [ 1362.,   641.]],

       [[ 1784.,   486.],
        [ 1928.,   483.],
        [ 1926.,   635.],
        [ 1784.,   637.]],

       [[ 2637.,   479.],
        [ 2778.,   480.],
        [ 2776.,   630.],
        [ 2634.,   629.]],
```

```
       [[ 2207.,   481.],
        [ 2356.,   478.],
        [ 2356.,   629.],
        [ 2205.,   632.]]])
```

```
help(cv2.aruco.detectMarkers)
```

```
Help on built-in function detectMarkers:

detectMarkers(...)
    detectMarkers(image, dictionary[, corners[, ids[, parameters[,␣
↪rejectedImgPoints]]]]) -> corners, ids, rejectedImgPoints
```

Note: This notebook can be downloaded here: `aruco_basics.ipynb`

1.1.6 ARUCO markers: basics

1: Marker creation

```
import numpy as np
import cv2, PIL
from cv2 import aruco
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
%matplotlib nbagg
```

```
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)

fig = plt.figure()
nx = 4
ny = 3
for i in range(1, nx*ny+1):
    ax = fig.add_subplot(ny,nx, i)
    img = aruco.drawMarker(aruco_dict,i, 700)
    plt.imshow(img, cmap = mpl.cm.gray, interpolation = "nearest")
    ax.axis("off")

plt.savefig("_data/markers.pdf")
plt.show()
```

```
<IPython.core.display.Javascript object>
```

2: Print, cut, stick and take a picture

```
frame = cv2.imread("_data/aruco_photo.jpg")
plt.figure()
plt.imshow(frame)
plt.show()
```

```
<IPython.core.display.Javascript object>
```

3: Post processing

```
%%time
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
parameters =  aruco.DetectorParameters_create()
corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict,␣
↪parameters=parameters)
frame_markers = aruco.drawDetectedMarkers(frame.copy(), corners, ids)
```

```
CPU times: user 420 ms, sys: 20 ms, total: 440 ms
Wall time: 172 ms
```

Pretty fast processing !

4: Results

```
plt.figure()
plt.imshow(frame_markers)
for i in range(len(ids)):
    c = corners[i][0]
    plt.plot([c[:, 0].mean()], [c[:, 1].mean()], "o", label = "id={0}".format(ids[i]))
plt.legend()
plt.show()
```

```
<IPython.core.display.Javascript object>
```

```
def quad_area(data):
    l = data.shape[0]//2
    corners = data[["c1", "c2", "c3", "c4"]].values.reshape(l, 2,4)
    c1 = corners[:, :, 0]
    c2 = corners[:, :, 1]
    c3 = corners[:, :, 2]
    c4 = corners[:, :, 3]
    e1 = c2-c1
    e2 = c3-c2
    e3 = c4-c3
    e4 = c1-c4
    a = -.5 * (np.cross(-e1, e2, axis = 1) + np.cross(-e3, e4, axis = 1))
    return a

corners2 = np.array([c[0] for c in corners])

data = pd.DataFrame({"x": corners2[:,:,0].flatten(), "y": corners2[:,:,1].flatten()},
                    index = pd.MultiIndex.from_product(
                            [ids.flatten(), ["c{0}".format(i )for i in np.
↪arange(4)+1]],
                        names = ["marker", ""] ))

data = data.unstack().swaplevel(0, 1, axis = 1).stack()
data["m1"] = data[["c1", "c2"]].mean(axis = 1)
```

```
data["m2"] = data[["c2", "c3"]].mean(axis = 1)
data["m3"] = data[["c3", "c4"]].mean(axis = 1)
data["m4"] = data[["c4", "c1"]].mean(axis = 1)
data["o"] = data[["m1", "m2", "m3", "m4"]].mean(axis = 1)
data
```

CHAPTER 2

Package documentation: