

# CALIBRACIÓN

---

Calibración de cámaras (e8.py)

Almacenamiento de datos de calibración JSON

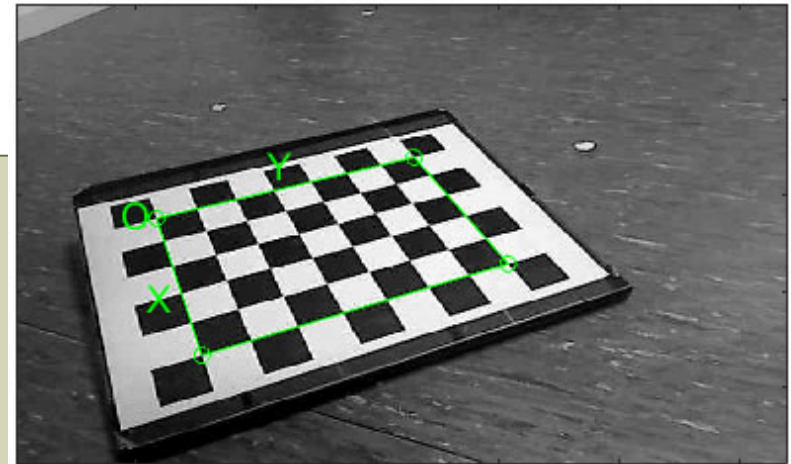
[https://docs.opencv.org/4.10.0/d9/d0c/group\\_\\_calib3d.html](https://docs.opencv.org/4.10.0/d9/d0c/group__calib3d.html)

[https://docs.opencv.org/4.10.0/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/4.10.0/dc/dbb/tutorial_py_calibration.html)

# Calibración de Cámaras

- **Ejemplo (e8.py)**: partiremos del código **e3.py** para la captura de imágenes de calibración
- Configurar datos patrón:

```
.....  
# Calibration pattern inner corners (cols Y-axis, rows X-axis)  
patternSize = (8, 6)  
squareSize = 27.0      # square pattern side in mm  
  
# 3D object points coordinates (x,y,z)  
objp3D = np.zeros((patternSize[1], patternSize[0], 3), np.float32)  
for x in range(patternSize[1]):  
    for y in range(patternSize[0]):  
        objp3D[x,y] = (x*squareSize, y*squareSize, 0)  
  
objp3D = objp3D.reshape(-1, 3) # transform in a row vector of (x,y,z) tuples  
  
objpoints = []    # 3D point in real world space  
imgpoints = []   # 2D points in image plane  
num_patterns = 0 # number of detected patterns. We need at lest 3  
  
.....
```



# Calibración de Cámaras

- Ejemplo (**e8.py**):
- Cargar ficheros de imagen de un directorio:
  - `os.listdir(path='.')` → files (tuple)

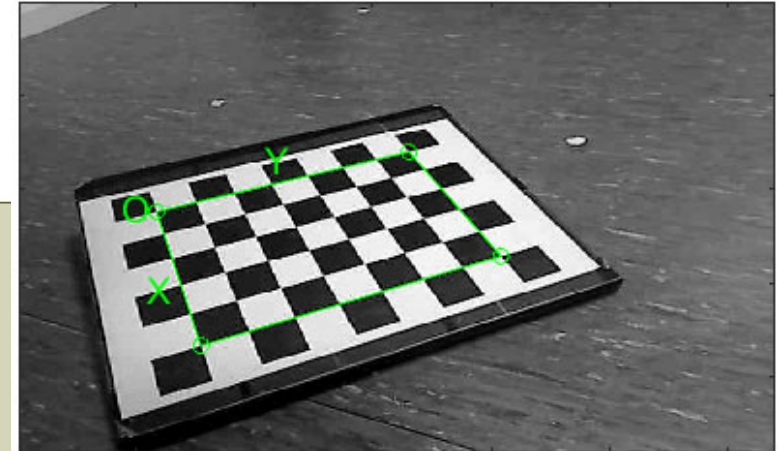
```
import os

.....

# reading directory files
prefix = 'Image'
valid_extension = ('jpg', 'jpeg', 'png', 'tiff', 'tif', 'bmp', 'pgm')
directoryList = sorted(os.listdir("."))
for file in directoryList:
    # Filter non image files
    if not file.startswith(prefix) or not file.endswith(valid_extension):
        continue

    print(f"Processing image file: {file}")
    image = cv.imread(file)
    gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY) # transforms to gray level

.....
```



# Calibración de Cámaras

---

- Ejemplo (**e8.py**):
- Detectar esquinas patrón calibración:
  - `cv.findChessboardCorners( image, patternSize[, corners[, flags]] )` → `retval, corners`  
**retval:** (bool) patternWasFound  
**corners:** `ndarray(nc,1,2)` Una fila por esquina, una columna que contiene una tupla (x,y)  
**patternSize:** (**cols** Y-axis, **rows** X-axis)  
**flags:** def: `cv.CALIB_CB_ADAPTIVE_THRESH, cv.CALIB_CB_NORMALIZE_IMAGE`  
`cv.CALIB_CB_FAST_CHECK , cv.CALIB_CB_FILTER_QUADS`
  - `cv.cornerSubPix( image, corners, winSize, zeroZone, criteria )` → `corners`  
**criteria:** tuple (`cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001`)  
**winSize:** tuple (11,11) Half of the side length of the search window  
**zeroZone:** tuple (-1,-1)
  - `cv.drawChessboardCorners( image, patternSize, corners, patternWasFound )` → `image`  
**patternSize:** (**cols** Y-axis, **rows** X-axis)  
**patternWasFound:** (bool). Devuelto por `cv.findChessboardCorners`



# Calibración de Cámaras

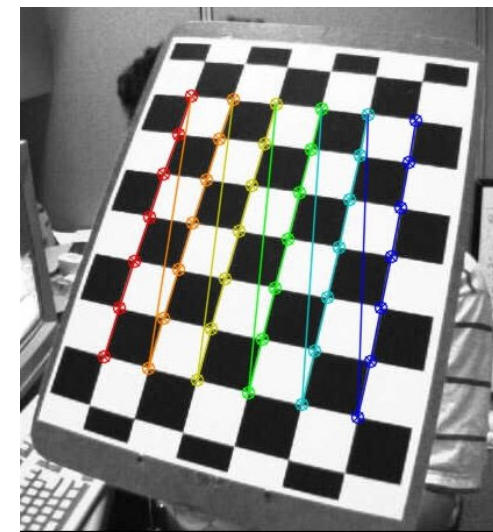
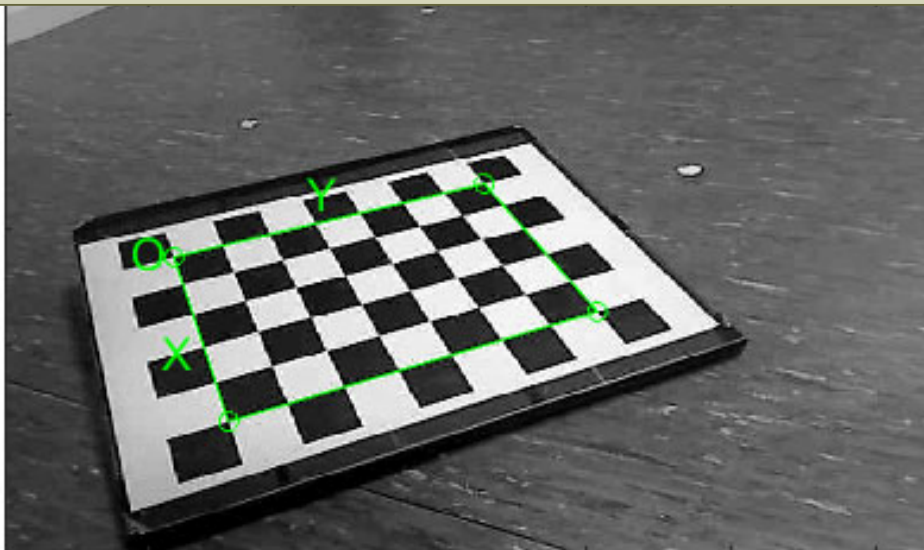
- Ejemplo (**e8.py**): Detectar esquinas patrón calibración:

```
patternWasFound, corners = cv.findChessboardCorners(gray_image, patternSize)

if patternWasFound:
    # Iterative algorithm termination criteria
    termCriteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
    corners = cv.cornerSubPix(gray_image, corners, winSize=(11,11), zeroZone=(-1,-1),
                              criteria=termCriteria)

    # Draw and display the corners
    cv.drawChessboardCorners(image, patternSize, corners, patternWasFound)

    cv.imshow(WINDOW_CAMERA1, image) # Display the resulting frame
    key = cv.waitKey(1000)          # update image and wait 1 second
    .....
```



# Calibración de Cámaras

## • Ejemplo (**e8.py**): Calibración

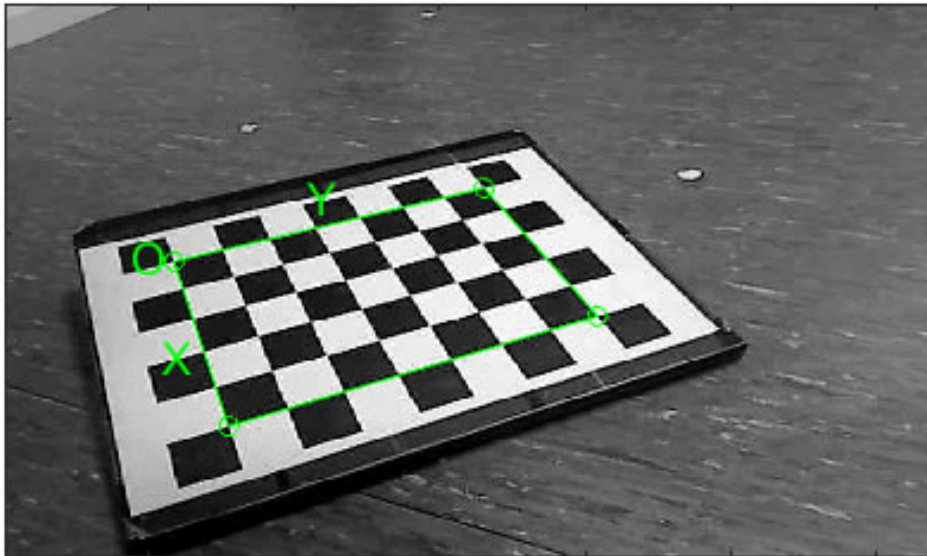
- `cv.calibrateCamera( objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs [, rvecs [, tvecs[, flags[, criteria]]]] )` → `retval, cameraMatrix, distCoeffs, rvecs, tvecs`
  - **retval**: (float) overall RMS reprojection error
  - **objectPoints**: tuple/list de puntos 3D tuple (x,y,z)
  - **imagePoints**: tuple/list de puntos 2D tuple (x,y)
  - **imageSize**: tuple (**cols**, **rows**)
  - **cameraMatrix, distCoeffs**: (ndarray)
  - **flags**: `cv.CALIB_USE_INTRINSIC_GUESS`, `cv.CALIB_ZERO_TANGENT_DIST` → (p1,p2) =0  
`cv.CALIB_FIX_XXX` (parámetro fijo)
  - **criteria**: (`cv.TERM_CRITERIA_EPS` + `cv.TERM_CRITERIA_MAX_ITER`, 30, 0.001)
- `cv.calibrateCameraExtended( objectPoints, imagePoints, imageSize, cameraMatrix, distCoeffs [, rvecs[, tvecs[, stdDeviationsIntrinsics[, stdDeviationsExtrinsics[, perViewErrors [, flags[, criteria]]]]]] ] )`  
→ `retval, cameraMatrix, distCoeffs, rvecs, tvecs, stdDeviationsIntrinsics, stdDeviationsExtrinsics, perViewErrors`

$$\text{camera matrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Distortion coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \\
 (k_1, k_2, p_1, p_2 [, k_3 [, k_4, k_5, k_6 [, s_1, s_2, s_3, s_4 [, \tau_x, \tau_y ]]])$$

# Calibración de Cámaras

- Ejemplo (**e8.py**): Calibración

```
if num_patterns >= 3:  
    # imageSize: (cols, rows)  
    rms, cameraMatrix, distCoeffs, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,  
                                                                    imageSize=gray_image.shape[:,-1],  
                                                                    cameraMatrix=None, distCoeffs=None )  
  
    print(f"RS reprojection error: {rms}")  
    .....
```



# Calibración de Cámaras: persistencia JSON

---

- **Lectura/escritura de datos de calibración en ficheros JSON:**
  - Permite almacenar/leer los tipos base y listas/tuplas/diccionarios/conjuntos de forma transparente
  - Otras estructuras: ndarray → convertirla en **lista** con el método **.tolist()**

```
import json
```

- Escribir fichero JSON:

- **json.dump( obj, fp)**
  - **obj**: pasaremos un diccionario { 'name': object }

```
# Writing calibration data to the file
with open('camera_calib.json', 'w') as file:
    json.dump( { 'camera_matrix': cameraMatrix.tolist(),
                 'distortion_coefficients': distCoeffs.tolist(),
                 'nx': imageSize[0], # width (x-axis)
                 'ny': imageSize[1], # height (y-axis)
                 'rms': rms },
               file, indent=4 )
```

# Calibración de Cámaras: persistencia JSON

---

- Lectura/escritura de datos de calibración en ficheros JSON:
- Leer fichero JSON:
  - `json.load(file)` → obj
    - obj: es un diccionario con los datos decodificados → { 'clave': valor }

```
# Reading calibration data from file
try:
    with open('camera_calib.json') as file:
        data = json.load( file )

    if 'camera_matrix' in data:
        camera_matrix = np.array(data['camera_matrix'])
    if 'distortion_coefficients' in data:
        distortion_coefficients = np.array(data['distortion_coefficients'])

    print('Camera matrix:', camera_matrix)
    print('Distortion coefficients:', distortion_coefficients)

except:
    print("File not valid")
```

# DETECCIÓN DE MARCAS

---

Librería ARUCO (Grupo AVA Univ. Córdoba) (e9.py)

Disponible en **opencv-contrib-python**

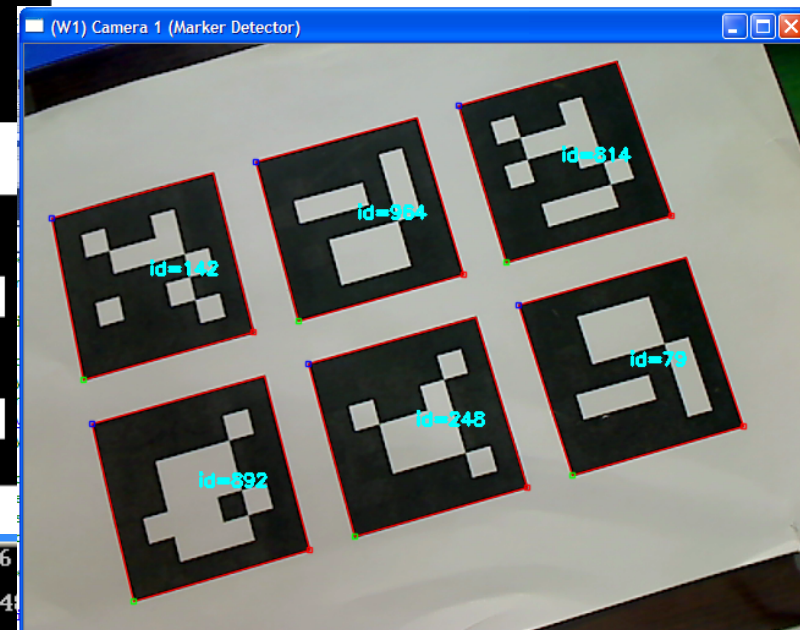
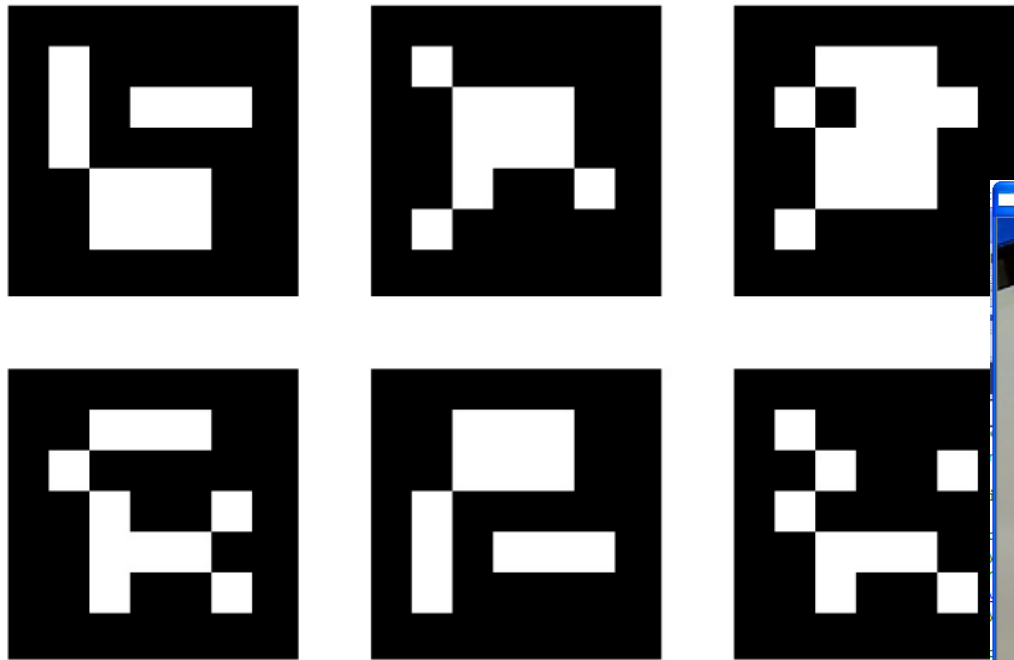
Instalación independiente: *pip install aruco*

Código fuente: <http://sourceforge.net/projects/aruco/>

<https://www.uco.es/investiga/grupos/ava/portfolio/aruco/>

# Detección de Marcas: ARUCO

- Ejemplo (**e9.py**) partiremos del código **e1.py**
  - Detectar y localizar marcadores codificados en la imagen
  - Cámara no calibrada
  - <http://umh1782.edu.umh.es/material/python/>
  - Fichero de tabla con marcadores: *markersBoard.pdf*, *marker\_id1.pdf*



```
Marker found id=142 Corners:[203.199, 241.962][62.3554, 279.877][36
541][170.196, 109.235]
Marker found id=248 Corners:[430.853, 373.421][286.081, 413.286][24
21][387.444, 230.51]
Marker found id=814 Corners:[550.907, 147.541][413.062, 185.569][373.05, 54.951
][504.638, 19.194]
```

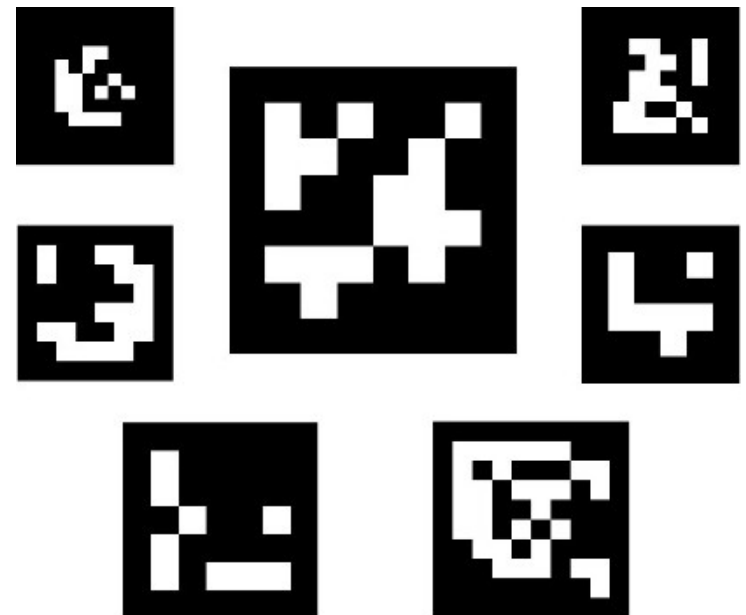
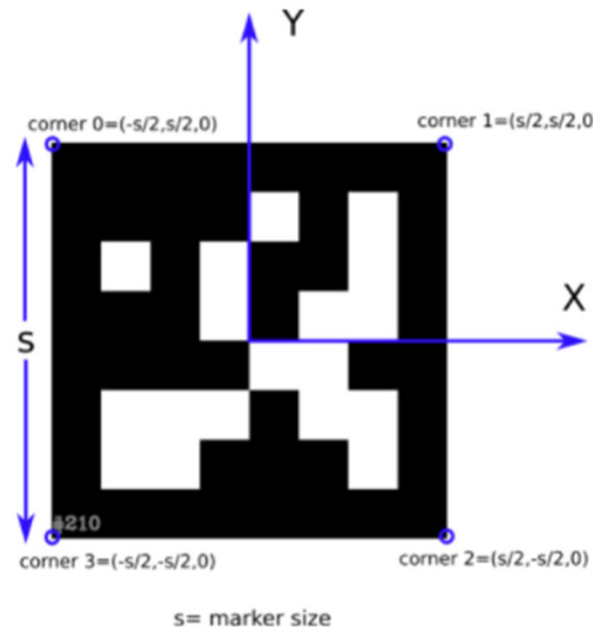
# Detección de Marcas: ARUCO

- Ejemplo (**e9.py**) módulo `cv.aruco` ([opencv-contrib-python](#))
  - Diccionarios de marcas disponibles: Clase `cv.aruco_Dictionary`
    - `cv.aruco.getPredefinedDictionary(dict) → dictionary` (`cv.aruco_Dictionary`)

**dict:** PREDEFINED\_DICTIONARY\_NAME

`cv.aruco.DICT_ARUCO_ORIGINAL` standard ArUco Library Markers. 1024 markers, 5x5 bits, 0 minimum distance  
`cv.aruco.DICT_APRIETAG_16h5` 4x4 bits, minimum hamming distance between any two codes = 5, 30 codes  
`cv.aruco.DICT_APRIETAG_25h9` 5x5 bits, minimum hamming distance between any two codes = 9, 35 codes  
`cv.aruco.DICT_APRIETAG_36h10` 6x6 bits, minimum hamming distance between any two codes = 10, 2320 codes  
`cv.aruco.DICT_APRIETAG_36h11` 6x6 bits, minimum hamming distance between any two codes = 11, 587 codes

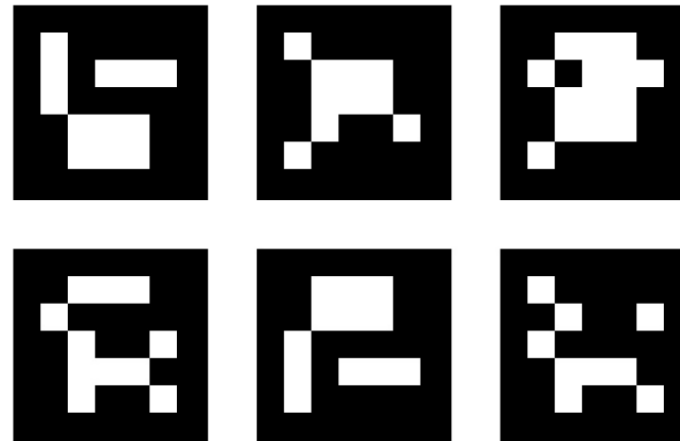
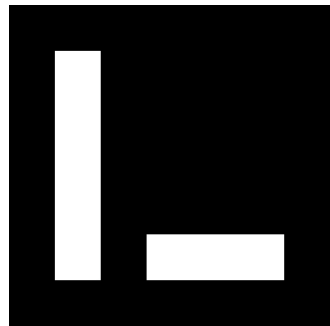
`cv.aruco.DICT_4X4_50`  
`cv.aruco.DICT_4X4_100`  
`cv.aruco.DICT_4X4_250`  
`cv.aruco.DICT_4X4_1000`  
`cv.aruco.DICT_5X5_50`  
`cv.aruco.DICT_5X5_100`  
`cv.aruco.DICT_5X5_250`  
`cv.aruco.DICT_5X5_1000`  
`cv.aruco.DICT_6X6_50`  
`cv.aruco.DICT_6X6_100`  
`cv.aruco.DICT_6X6_250`  
`cv.aruco.DICT_6X6_1000`  
`cv.aruco.DICT_7X7_50`  
`cv.aruco.DICT_7X7_100`  
`cv.aruco.DICT_7X7_250`  
`cv.aruco.DICT_7X7_1000`



# Detección de Marcas: ARUCO

---

- Ejemplo (**e9.py**)
- Clase: `cv.aruco.Board`
  - `cv.aruco.Board( objPoints, dictionary, ids )` → `board` (`cv.aruco.Board`)
    - objPoints**: array of object points of all the marker corners in the board `ndarray(nm,4,3)`
    - dictionary**: the dictionary of markers employed for this board
    - ids**: list of the identifiers of the markers in the board `ndarray(nm) / tuple()`
- Crear imagen marcas:
  - Método objeto `cv.aruco.Board`
    - `cv.aruco.Board.generateImage` (`outSize` [, `img` [, `marginSize` [, `borderBits`]]]) → `im`
      - outSize** size of the output image in pixels `tuple(w,h)`



# Detección de Marcas: ARUCO

- Ejemplo (**e9.py**)
- Funciones: módulo `cv.aruco` ([opencv-contrib-python](#))

Clase `cv.aruco.ArucoDetector`( [ , dictionary[, detectorParams[, refineParams]] )

Métodos:

- `cv.aruco.ArucoDetector.detectMarkers`( **image** [ , corners [ , ids [ , rejectedImgPoints]]])  
→ corners, ids, rejectedImgPoints

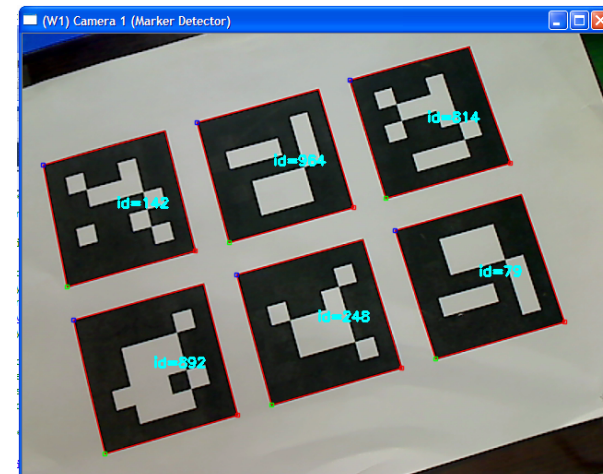
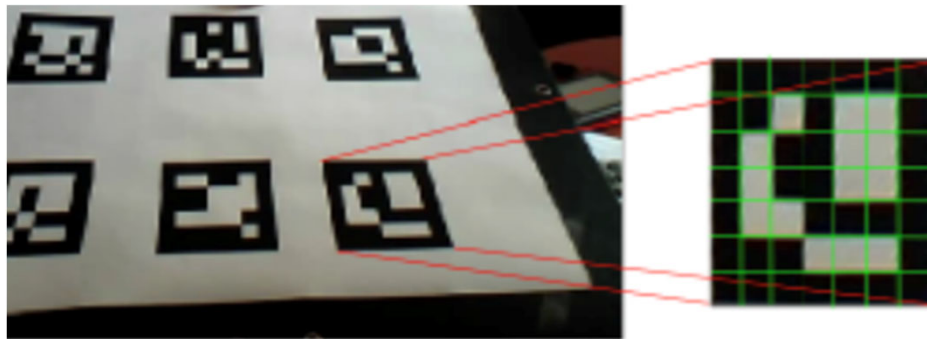
**corners**: tupla (nm) de `ndarray(1,4,2)`, 2º eje (corners) 3º eje tupla(x,y)

**ids**: `ndarray(nm,1)`. **None** si no detecta marcas

Clase: `cv.aruco.DetectorParameters` (configura parámetros de la detección)

Mostar marcas:

- `cv.aruco.drawDetectedMarkers`( **image**, **corners** [ , **ids** [ , **borderColor**]]) → image



# Detección de Marcas: ARUCO

---

- Ejemplo (**e9.py**)
  - Detectar y localizar marcadores codificados en la imagen
- Inicialización:

```
dictionary = cv.aruco.getPredefinedDictionary(cv.aruco.DICT_ARUCO_ORIGINAL)  
detector = cv.aruco.ArucoDetector(dictionary)
```

- Detección:

```
gray_image = cv.cvtColor(capture, cv.COLOR_BGR2GRAY) # transforms to gray level  
corners, ids, rejectedImgPoints = detector.detectMarkers(gray_image)  
dispimage = cv.aruco.drawDetectedMarkers(capture, corners, ids, borderColor=(0,0,255))
```

```
for item in corners: # display corner order  
    for i in range(4):  
        cv.putText(dispimage, str(i), item[0,i].astype(int), cv.FONT_HERSHEY_DUPLEX, 0.4,  
                  (0,255,255), 1, cv.LINE_AA)  
  
cv.imshow(WINDOW_CAMERA1, dispimage) # Display the resulting frame
```

# DETECCIÓN DE MARCAS

---

Librería **ARUCO** (Grupo AVA Univ. Córdoba) ([e9b.py](https://github.com/alejandrohdezcamacho/e9b.py))

- Leer datos calibración (JSON)
- Calcular la POSE 3D de cada marcador

# Detección de Marcas: ARUCO

---

- Ejemplo (**e9b.py**)
  - Detectar y localizar marcadores codificados en la imagen
  - Calcular la POSE y dibujar los ejes 3D proyectados en la imagen
  - Cámara Calibrada
  - <http://umh1782.edu.umh.es/python/>
  - Fichero de calibración de la cámara: **calib\_LogitechC300.json**
  - Fichero de configuración de marcas: **ARUCO\_board.json**

```
{
  "calibration_date": "28-04-2015 18:25",

  "camera_matrix": [
    [ 8.1651259051274019e+02, 0.0, 3.2889284380566903e+02 ],
    [ 0.0, 8.1639695331744053e+02, 2.3665325417424432e+02 ],
    [ 0.0, 0.0, 1.0 ]
  ],

  "distortion_coefficients": [ -3.3211338666840002e-03,
    -5.6912036471142000e-02, -1.0612607134607000e-02,
    5.8539220925489998e-03, 0.0 ],

  "nx": 640,
  "ny": 480,
}
```

```
{
  "aruco_bc_nmarkers": 6,
  "aruco_bc_mInfoType": 0,
  "aruco_bc_markers": [
    { "id":79, "corners":[[ 0.0, 0.0, 0.0 ], [ 0.0, 57.0, 0.0 ],
      [ 57.0, 57.0, 0.0 ], [ 57.0, 0.0, 0.0 ] ]},
    { "id":248, "corners":[[ 0.0, 71.0, 0.0 ], [ 0.0, 128.0, 0.0 ],
      [ 57.0, 128.0, 0.0 ], [ 57.0, 71.0, 0.0 ] ]},
    { "id":892, "corners":[[ 0.0, 143.0, 0.0 ], [ 0.0, 200.0, 0.0 ],
      [ 57.0, 200.0, 0.0 ], [ 57.0, 143.0, 0.0 ] ]},
    { "id":814, "corners": [[ 71.0, 0.0, 0.0 ], [ 71.0, 57.0, 0.0 ],
      [ 128.0, 57.0, 0.0 ], [ 128.0, 0.0, 0.0 ] ]},
    { "id":964, "corners":[[ 71.0, 71.0, 0.0 ], [ 71.0, 128.0, 0.0 ],
      [ 128.0, 128.0, 0.0 ], [ 128.0, 71.0, 0.0 ] ]},
    { "id":142, "corners":[[ 71.0, 143.0, 0.0 ], [ 71.0, 200.0, 0.0 ],
      [ 128.0, 200.0, 0.0 ], [ 128.0, 143.0, 0.0 ] ] }
  ]
}
```

# Detección de Marcas: ARUCO

---

- Ejemplo (**e9b.py**)
- Calculo POSE:

**NOTA: a partir de OpenCV 4.7 estas funciones han sido eliminadas**

- `cv.aruco.estimatePoseSingleMarkers( corners, markerLength, cameraMatrix, distCoeffs  
[, rvecs[, tvecs[, _objPoints]])` → `rvecs, tvecs, _objPoints`
  - **rvecs**: array of output rotation vectors (Rodrigues) `ndarray(nm,1,3)`
  - **tvecs**: array of output translation vectors. `ndarray(nm,1,3)`
  - **\_objPoints**: array of object points of all the marker corners
- `cv.aruco.estimatePoseBoard( corners, ids, board, cameraMatrix, distCoeffs, rvec, tvec  
[, useExtrinsicGuess])` → `retval, rvec, tvec`

- Funciones dibujo:

- `cv.drawFrameAxes( image, cameraMatrix, distCoeffs, rvec, tvec, length [, thickness] )`  
→ `image`

**OX** is drawn in **red**, **OY** in **green** and **OZ** in **blue**.

# Detección de Marcas: ARUCO

---

- Ejemplo (**e9b.py**)
- Calculo POSE:
  - `cv.solvePNP(objectPoints, imagePoints, cameraMatrix, distCoeffs [, rvec[, tvec[, useExtrinsicGuess[, flags]]])` → `retval, rvec, tvec,`
    - **retval**: Bool
    - **rvec**: array of output rotation vector (Rodrigues) **ndarray(1,3)**
    - **tvec**: array of output translation vector. **ndarray(1,3)**
  
    - **objectPoints**: np.array of object 3D points
    - **imagePoints**: np.array of object 2D image points
  
    - **flags**: Estimation method: `cv.SOLVEPNP_ITERATIVE, cv.SOLVEPNP_EPNP, cv.SOLVEPNP_P3P, cv.SOLVEPNP_AP3P, cv.SOLVEPNP_SQPNP, cv.SOLVEPNP_IPPE, cv.SOLVEPNP_IPPE_SQUARE`
- Funciones dibujo:
  - `cv.drawFrameAxes( image, cameraMatrix, distCoeffs, rvec, tvec, length [, thickness] )`  
→ `image`  
**OX** is drawn in **red**, **OY** in **green** and **OZ** in **blue**.

# Detección de Marcas: ARUCO

---

- **Ejemplo (e9b.py)**. Cargar datos de calibración y coordenadas marcador

```
import json
```

```
CALIB_FILE = calib_LogitechC300.json'      # default camera calibration file
```

```
try:
```

```
    with open('camera.json') as file:
```

```
        data = json.load(file)
```

```
except:
```

```
    print('Camera Calibration File not valid')
```

```
    exit()
```

```
cameraMatrix = np.array(data['camera_matrix'])
```

```
distCoeffs = np.array(data['distortion_coefficients'])
```

```
print(f"{cameraMatrix=}")
```

```
print(f"{distCoeffs=}")
```

```
# Marker data
```

```
marker_size = 93.0    # mm
```

```
marker3Dpoints = np.array([ [-marker_size/2, -marker_size/2, 0],    # (0,0,0) center of the marker  
                             [-marker_size/2, marker_size/2, 0],  
                             [ marker_size/2, marker_size/2, 0],  
                             [ marker_size/2, -marker_size/2, 0] ], dtype=np.float32)
```

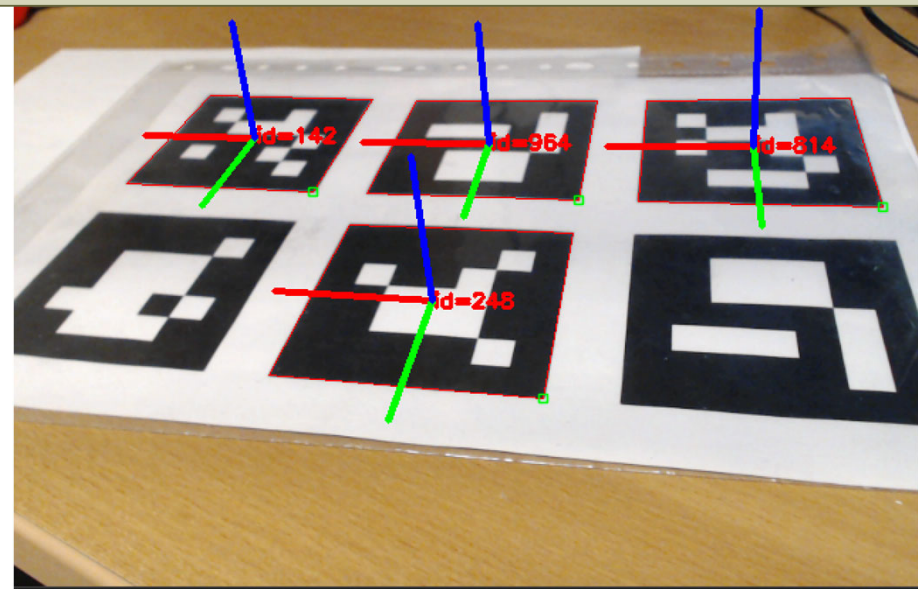
# Detección de Marcas: ARUCO

- Ejemplo (**e9b.py**). Bucle principal:

```
# Calculate POSE for each marker
rvecs, tvecs = [], []
for item in corners:
    ret, rvec, tvec = cv.solvePnP(marker3Dpoints, item, cameraMatrix, distCoeffs)
    rvecs.append(rvec)
    tvecs.append(tvec)

# Draw axis for each marker
for i in range(len(rvecs)):
    dispimage = cv.drawFrameAxes(dispimage, cameraMatrix, distCoeffs,
                                   rvecs[i], tvecs[i], length=50.0)

...
print(f"{rvecs=} \n {tvecs=}")
```



# DETECCIÓN DE MARCAS

---

Librería **ARUCO** (Grupo AVA Univ. Córdoba) ([e9c.py](https://github.com/alexisbarrat/e9c.py))

- Leer datos calibración (JSON)
- Calcular la POSE 3D de un conjunto de marcadores (MarkerMap)

# Detección de Marcas: ARUCO

- Ejemplo (**e9c.py**)
- Clase: `cv.aruco.Board`
  - `cv.aruco.Board( objPoints, dictionary, ids )` → `board` (`cv.aruco.Board`)

**objPoints**: array of object points of all the marker corners in the board `ndarray(nm,4,3)`

**dictionary**: the dictionary of markers employed for this board

**ids**: list of the identifiers of the markers in the board `ndarray(nm) / tuple()`

## Métodos:

- `cv.aruco.Board.matchImagePoints( detectedCorners, detectedIds[, objPoints[, imgPoints] ] )`  
→ `objPoints, imgPoints`

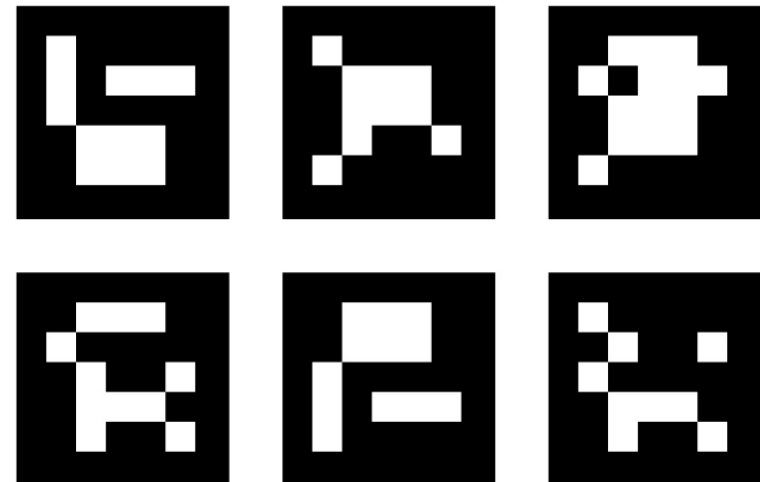
**detectedCorners** : List of detected marker corners of the board

**detectedIds** : List of identifiers for each marker

Devuelve dos listas con las esquinas de todos los marcadores detectados y casados:

**objPoints** : List of 3D coordinates for board corners

**imgPoints** : List of 2D coordinates for board corners



# Detección de Marcas: ARUCO

---

- Ejemplo (**e9c.py**) cálculo de POSE con múltiples marcas (Board)
  - Cargar datos de marcas (Board) clase: `cv.aruco_Board`

```
BOARD_FILE = 'ARUCO_board.json' # default ARUCO board data file

# Init ARUCO dictionary
dictionary = cv.aruco.getPredefinedDictionary(cv.aruco.DICT_ARUCO_ORIGINAL)

# define ARUCO detector object
detector = cv.aruco.ArucoDetector(dictionary)

# Load ARUCO Board data
try:
    with open(BOARD_FILE) as file:
        data = json.load(file)
except:
    print('Board Data File not valid')
    exit()

# data['aruco_bc_markers'] contains the marker's data (ids, 3D coordenates) as a dictionary
# split ids and objPoints (corners) in two separate lists
boardIds = [ item["id"] for item in data['aruco_bc_markers'] ]
boardPoints = [ item["corners"] for item in data['aruco_bc_markers'] ]

# Create cv.aruco.Board Object (id's and 3D corner's coordinates)
board = cv.aruco.Board(boardPoints, dictionary, boardIds)
```

# Detección de Marcas: ARUCO

- Ejemplo (**e9c.py**) cálculo de POSE con múltiples marcas (Board)
  - Bucle principal:

....

```
# Calculate POSE
```

```
if len(corners) > 0:
```

```
    # Find 3D coordinates for each corner of the markers detected using Board data
```

```
    # returns a flat coordinates array with all matching corners
```

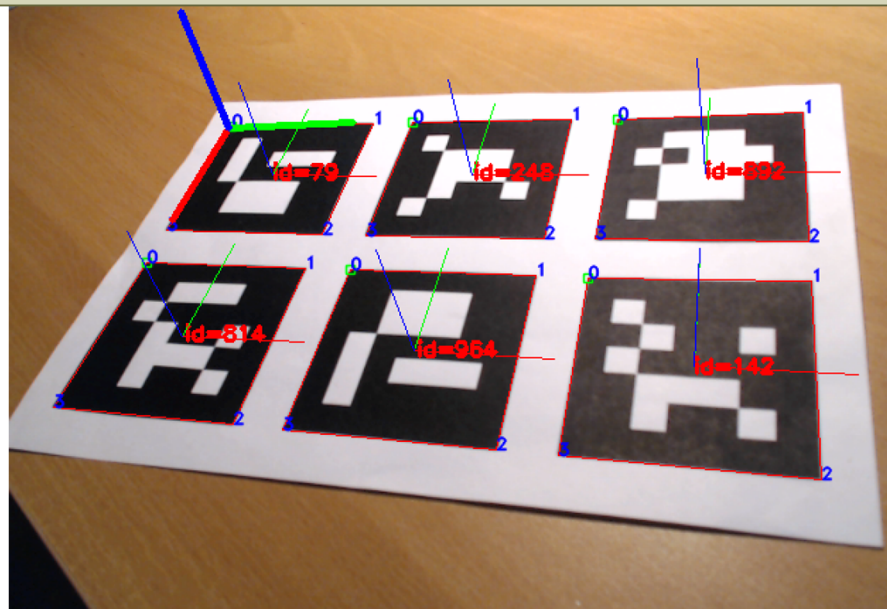
```
    objPoints, imgPoints = board.matchImagePoints(corners, ids)
```

```
    if objPoints is not None:
```

```
        retval, rvec, tvec = cv.solvePnP(objPoints, imgPoints, cameraMatrix, distCoeffs)
```

```
        # Draw board axis
```

```
        dispimage = cv.drawFrameAxes(dispimage, cameraMatrix, distCoeffs, rvec, tvec, length=50.0)
```



# Detección de Marcas: ARUCO

- Ejercicio:
  - Proyectar información 3D en la imagen: (módulo calib3D)
    - `cv.projectPoints( objectPoints, rvec, tvec, cameraMatrix, distCoeffs [, imagePoints [, jacobian[, aspectRatio]]])` → `imagePoints, jacobian`

**objectPoints:** Array of 3D object points (World reference RT) `ndarray(np,1,3)`

**imagePoints:** 2D image points `ndarray(np,1,2)`

