Localisation via la vision avec la bibliothèque Apriltags

La Revue 3E.I
Culture Sciences de l'Ingénieur

Gilles Arthur FADE[1] – Anthony JUTON[2]

Édité le 18/11/2024

[1] *Elève de 4ème année au DER Nikola Telsa de l'ENS Paris-Saclay.*
[2] *Professeur agrégé de physique appliquée au département N. Tesla de l'ENS Paris-Saclay.*

Cette ressource fait partie du N°114 de La Revue 3EI du 1er trimestre 2025.

La localisation précise est une problématique importante de la robotique mobile (robots agricoles, voitures autonomes, robots de logistique, …). En extérieur, l'ajout de balises au sol permet au GPS RTK d'assurer une localisation au centimètre. En intérieur, une solution possible est d'utiliser la vision pour repérer des étiquettes (tags) dont la forme et la position est connue. Apriltag [1] est une bibliothèque basée sur ce concept, exploitée à l'ENS Paris-Saclay pour la localisation de drone-dirigeable (en intérieur) et pour ajuster l'arrivée d'un robot autonome sur son socle de recharge.

Cette ressource présente pas à pas comment, à partir d'une caméra embarquée, il est possible d'atteindre une précision de localisation au centimètre ainsi que les angles de rotation de la caméra.

Les fichiers .py des codes pour raspberry pi avec caméra raspberry pi v2 sont fournis [10]. Il est assez aisé de passer à une autre cible. Le fichier pdf des images des tags est fourni également [10].

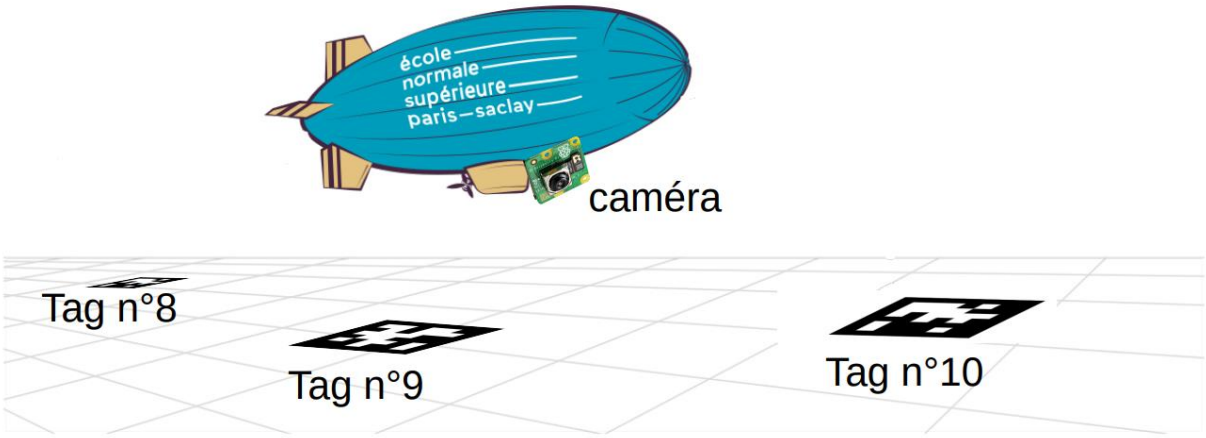


Figure 1 : Schéma de principe de la localisation à base d'Apriltags

Si la zone de déplacement est limitée, il est également possible de mettre le tag sur l'objet mobile et d'installer la caméra de manière fixe, localisée précisément.

1 - Apriltags parmi les autres solutions de localisation

Lors des deux sujets évoqués ci-dessus (dirigeable autonome et guidage vers un socle de recharge), plusieurs solutions ont été étudiées.

Technologie	Principe	Avantages	Inconvénients	Applications
FingerPrint	On établit une carte des niveaux de réception des émetteurs fixes bluetooth et wifi. En fonction des réseaux qu'il reçoit et de l'intensité du signal, le robot peut alors se placer sur cette carte.	Il n'y a rien à installer, on profite des réseaux existants (des fabricants comme Siemens Building Technologie propose des émetteurs bluetooth dans les détecteurs de présence pour augmenter la densité d'émetteurs).	La précision est de l'ordre du mètre au mieux en intérieur.	Les voitures en charge de la cartographie googleStreetMap ont aussi cartographié les réseaux wifi. Le récepteur wifi d'un smartphone participe ainsi à la localisation, en cas de signaux GPS faibles. C'est utilisé également pour localiser les audioguides dans les musées.
Ultra Wide Band	Des émetteurs UWB sont installés et on mesure les durées de réception.	Les balises et le récepteur sont peu chers, légers et à basse consommation. C'est relativement robuste au masquage.	Précision de l'ordre de 50 cm. Les balises, bien que frugales, nécessitent une alimentation.	Qorvo propose notamment une solution DWM3000 avec 30 cm de précision.
GPS Indoor (Marvelmind)	Les balises sont des émetteurs ultrasons. La mesure du temps de vol des ultrasons est couplée à une centrale inertielle.	Système éprouvé, livré avec son logiciel de configuration et ses bibliothèques.	Les ultrasons sont sensibles au masquage. Le système a un coût certain pour des surfaces importantes (500 euros pour 200 m^2). Taille et poids du récepteur. Balises nécessitant une alimentation.	Utilisé pour la robotique indoor. Marvelmind annonce 2 cm de précision.
Apriltag	Une caméra sur le robot se localise grâce à la vue de tags dont la taille et la position sont connues.	Système très bon marché (15 euros pour la caméra et simple impression pour les tags). Précision de l'ordre du centimètre. Les tags sont passifs.	Il faut positionner de nombreux tags, la caméra devant toujours en avoir au moins un dans son champ de vision. Le traitement de l'image nécessite un système informatique conséquent pour avoir un rafraichissement fréquent de la position.	Localisation indoor

Tableau 1 : Comparaison de quatre solutions de localisation intérieure

1.1 - Matériel utilisé

Le matériel utilisé pour les exemples fournis dans cette ressource est celui du robot se localisant pour sa recharge.

Figure 2 : Équipements utilisés pour la localisation via Apriltags

Pour tenir compte des contraintes de poids et de consommation, le dirigeable est, quant à lui, équipé d'un nano-ordinateur Raspberry pi zéro 2 W, associé à la même caméra. Il est possible de faire le même travail sur un PC équipé d'une webcam. Par la suite, le système mobile sur lequel se situe la caméra sera nommé robot.

1.2 - Démarche suivie

Les 4 étapes menant à la localisation sont expliquées dans la suite de cette ressource :

- Acquisition de l'image
- Calibration de la caméra
- Détection des tags
- Localisation

2 - Acquisition de l'image

L'installation du système d'exploitation sur la carte raspberry pi, en mode Headless (sans clavier ni écran) est expliqué dans le guide Installation de Raspberry OS sur Raspberry Pi 4 [2].

Les modules et bibliothèques suivants doivent être installés, depuis un terminal de la Raspberry Pi 4 (accès en SSH ou depuis un client VNC) :

- La bibliothèque pour la caméra raspberry [3] :

```
$ sudo apt install -y python3-picamera2
```

- La bibliothèque openCV [4]

```
$ sudo apt install -y python3-opencv
```

- La bibliothèque numpy

```
$ sudo apt install -y python3-numpy
```

Comme indiqué dans la documentation, il est possible de tester le bon fonctionnement de la caméra et de connaître ses résolutions possibles grâce à la commande suivante :

```
$ rpicam-hello
```

```
opcuaserver@opcuaserver:~ $ rpicam-hello
[4:14:00.487322463] [55436]  INFO Camera camera_manager.cpp:325 libcamera v0.3.2+27-7330f29b
[4:14:00.554521081] [55440]  WARN RPiSdn sdn.cpp:40 Using legacy SDN tuning - please conside
[4:14:00.556430835] [55440]  WARN RPI vc4.cpp:393 Mismatch between Unicam and CamHelper for
[4:14:00.557316694] [55440]  INFO RPI vc4.cpp:447 Registered camera /base/soc/i2c0mux/i2c@1/
/dev/media0
[4:14:00.557396953] [55440]  INFO RPI pipeline_base.cpp:1126 Using configuration file '/usr/
Made X/EGL preview window
Mode selection for 1640:1232:12:P
    SRGGB10_CSI2P,640x480/0 - Score: 4504.81
    SRGGB10_CSI2P,1640x1232/0 - Score: 1000
    SRGGB10_CSI2P,1920x1080/0 - Score: 1541.48
    SRGGB10_CSI2P,3280x2464/0 - Score: 1718
    SRGGB8,640x480/0 - Score: 5504.81
    SRGGB8,1640x1232/0 - Score: 2000
    SRGGB8,1920x1080/0 - Score: 2541.48
    SRGGB8,3280x2464/0 - Score: 2718
[4:14:02.021216662] [55436]  INFO Camera camera.cpp:1197 configuring streams: (0) 1640x1232-
[4:14:02.022263186] [55440]  INFO RPI vc4.cpp:622 Sensor: /base/soc/i2c0mux/i2c@1/imx219@10
ected unicam format: 1640x1232-pBAA
```

Figure 3 : Résultat de l'exécution de la commande rpicam-hello

Le programme suivant permet de faire l'acquisition d'une image depuis un programme python :

```
import cv2
from picamera2 import Picamera2
import time

picam2 = Picamera2()
picam2.configure(picam2.create_preview_configuration(main={"format": 'XRGB8888',
"size": (640, 480)}))
#picam2.configure(picam2.create_preview_configuration(main={"format":
'XRGB8888', "size": (3280, 2464)}))
picam2.start()

while True:
    t1 = time.time()
    image=cv2.cvtColor(picam2.capture_array(),cv2.COLOR_BGR2GRAY)
    t2 = time.time()
    cv2.imshow("Camera", image)
    t3 = time.time()
    print("duree acquisition ", (t2-t1), "duree affichage", (t3-t2))
    cv2.waitKey(1)
```

Figure 4 : Programme acquisitionCameraPython.py pour l'acquisition d'une image

Sur un PC avec une webcam, la bibliothèque OpenCV permet de faire l'acquisition de l'image, sans besoin de picamera2.

3 - Calibration de la caméra

Pour obtenir la position d'un tag dans le repère de la caméra, il est indispensable de connaître la distance focale de la caméra et, pour une bonne précision, de connaître la matrice de distorsion due à la lentille de la caméra. L'opération de calibration de la caméra permet d'obtenir ces paramètres intrinsèques de la caméra. La production industrielle ayant une bonne répétabilité, les paramètres sont semblables d'une caméra raspberry pi v2 à une autre. Toutefois, ils dépendent de la résolution choisie et l'étape n'est pas très longue.

Le site web d'OpenCV présente en détail le principe de la calibration [5]. Cette ressource se contente de présenter la démarche expliquée sur la page suivante d'OpenCV [6] :

3.1 - Imprimer un damier

Pour cette ressource, c'est un damier 9x9 avec des cases de 3 cm qui a été choisi et collé sur une boite rigide. Un damier non carré est préférable car il évite la confusion entre les axes x et y par le programme de calibration

3.2 - Acquérir 10 images

Il faut au moins 10 acquisitions de ce damier par la caméra, au format jpg. La fonction libcamera-still est bien utile pour cela :

```
$ libcamera-still -o nomdufichier.jpg
```

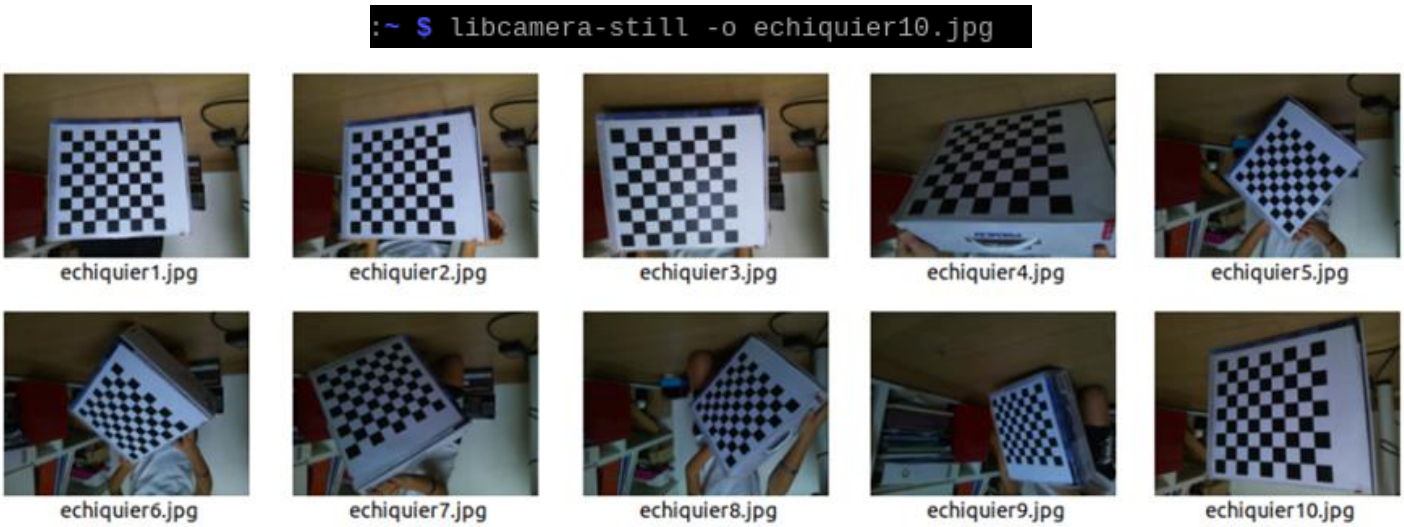


Figure 5 : images acquises pour la calibration de la caméra

3.3 - Lancer le programme de calibration

Une fois les images acquises sur la caméra du robot, il est possible de lancer le programme de calibration sur une machine plus puissante. Le robot utilisé ici étant doté d'un nano-ordinateur Raspberry Pi 4, la calibration est lancée directement dessus.

Le programme, repris de la page d'OpenCV [6] et adapté au damier 9x9 (avec donc 8x8 coins internes) de cases 3 cm est le suivant. Il doit être placé dans le dossier où sont stockées les 10 images d'échiquier.

```
import numpy as np
import cv2 as cv
import glob

# termination criteria carrés de 30 mm
criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(6,5,0)
objp = np.zeros((8*8,3), np.float32)
objp[:,:2] = np.mgrid[0:8,0:8].T.reshape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.
images = glob.glob('*.jpg')
for fname in images:
    img = cv.imread(fname)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    # Find the chess board corners
    ret, corners = cv.findChessboardCorners(gray, (8,8), None)
    # If found, add object points, image points (after refining them)
    if ret == True:
        objpoints.append(objp)
        corners2 = cv.cornerSubPix(gray,corners, (11,11), (-1,-1), criteria)
        imgpoints.append(corners2)
        # Draw and display the corners
        cv.drawChessboardCorners(img, (8,8), corners2, ret)
        cv.imshow('img', img)
        cv.waitKey(500)

cv.destroyAllWindows()
ret, mtx, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
print("mtx : ",mtx)
print("dist : ",dist)
```

Figure 6 : Programme calibrationCamera.py

Le programme renvoie alors la matrice *mtx* de la caméra et la matrice *dist* de distorsion.

```
>>> %Run calibrationCamera.py

  qt.qpa.plugin: Could not find the Qt platform plugin "wayland" in "/home/o
  v2/qt/plugins"
  mtx :  [[2.53069519e+03 0.00000000e+00 1.75866732e+03]
   [0.00000000e+00 2.52204812e+03 1.21835539e+03]
   [0.00000000e+00 0.00000000e+00 1.00000000e+00]]
  dist :  [[ 0.13432391 -0.13190146  0.00487085  0.01105209 -0.30871937]]
```

3.4 - Détection des tags

Avant de détecter les tags, il faut les imprimer. Apriltag propose un dépôt avec plusieurs familles de tags [8]. La famille par défaut 36h11 convient bien aux conditions de cette étude (587 tags, avec une résolution peu élevée). Le fichier pdf avec des tags de 17,3 cm est fournie avec cette ressource.

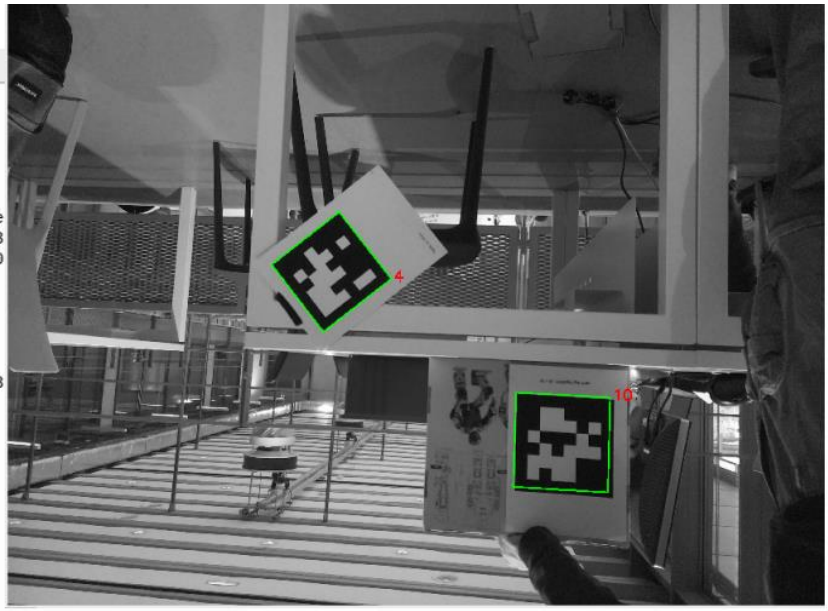
Pour détecter les tags, on utilise un portage python, pyapriltags [7], de la bibliothèque Aptriltag, écrite en langage C. Sur le robot, le nano-ordinateur n'a pas d'autres usages que le contrôle du robot, donc les environnements virtuels ne sont pas utilisés, ce qui explique l'option *--break-system-packages* :

```
$ pip install pyapriltags --break-system-packages
```

Le programme de détection des tags est adapté de celui proposé par le dépôt git de pyaprilags [9]. Remarquer l'indication des paramètres de la caméra et de la taille du tag (0,173 m).

```
from pyapriltags import Detector
import cv2
import numpy
from picamera2 import Picamera2
import os

#Configuration de l'acquisition et paramètres de la calibration
picam2 = Picamera2()
WIDTH,HEIGH=3280,2464
picam2.configure(picam2.create_preview_configuration({'size':(WIDTH,HEIGH)}))
picam2.start()
fx=2530.69519
fy=2522.04812
cx=1758.66732
cy=1218.35539
mtx = numpy.array([[fx,0,cx],[0,fy,cy],[0,0,1]])
dist = numpy.array([[0.13432391,-0.13190146,0.00487085,0.01105209,-0.30871937]])

#Acquisition de l'image, correction et détection des tags présents sur l'image
img=cv2.cvtColor(picam2.capture_array(),cv2.COLOR_BGR2GRAY) #prise d'une photo
at_detector = Detector()
img_undistorded = cv2.undistort(img, mtx, dist, None, newCameraMatrix=mtx)
tags=at_detector.detect(img_undistorded, estimate_tag_pose=True,
camera_params=[fx,fy,cx,cy], tag_size=0.173)
print(tags)

#Affichage de l'image et pour chaque tags, des lignes reliant les coins et du
numéro
color_img = cv2.cvtColor(img, cv2.COLOR_GRAY2RGB)
for tag in tags:
    for idx in range(len(tag.corners)):
        cv2.line(color_img, tuple(tag.corners[idx-1, :].astype(int)),
tuple(tag.corners[idx, :].astype(int)),
                 (0, 255, 0),5)

    cv2.putText(color_img, str(tag.tag_id), org=(tag.corners[0,
0].astype(int)+10,tag.corners[0, 1].astype(int)+10),
            fontFace=cv2.FONT_HERSHEY_SIMPLEX, fontScale=2, color=(0, 0,
255),thickness=5)

cv2.imshow('Detected tags', color_img)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
```

Figure 7 : programme detectionApriltags.py



```
Shell
[Detection object:
tag_family = b'tag36h11'
tag_id = 4
tag_size = 0.173
hamming = 0
decision_margin = 60.11915969848633
homography = [[-9.92707762e+01  1.35397916e
 [-1.29654511e+02 -1.05726395e+02  1.130073
 [ 3.64948577e-03  1.41159575e-04  1.000000
center = [1296.17899103 1130.07395578]
corners = [[1536.23730469 1158.06494141]
 [1327.27490234  891.31439209]
 [1057.79919434 1102.27868652]
 [1264.84643555 1370.6505127 ]]
pose_R = [[-0.61741651  0.78586926  0.03473
 [-0.78410775 -0.61836905  0.05286538]
 [ 0.06302291  0.00540578  0.99799744]]
pose_t = [[-0.23292996]
 [-0.04466603]
 [ 1.27447714]]
pose_err = 3.10671914452935e-08
, Detection object:
tag_family = b'tag36h11'
tag_id = 10
tag_size = 0.173
hamming = 0
decision_margin = 43.55595397949219
homography = [[-1.82607975e+02  1.22162398e+01  2.21903762e+03]
 [-5.05076794e+00 -1.83720223e+02  1.81194393e+03]
 [ 2.93146774e-03  3.07906089e-03  1.00000000e+00]]
center = [2219.03761751 1811.94392706]
corners = [[2413.50561523 1633.03344727]
 [2036.40600586 1613.47509766]
 [2024.51220703 1990.90722656]
 [2403.87792969 2012.81298828]]
pose_R = [[-0.99736217  0.04269254  0.05870299]
 [-0.03873319 -0.99700045  0.06700625]
 [ 0.06138757  0.06455575  0.99602416]]
pose_t = [[0.21215382]
 [0.27450297]
 [1.16698485]]
pose_err = 4.720902499790753e-07
]
```

Figure 8 : Résultat obtenu par l'exécution du programme detectionApriltags.py

Le programme renvoie le contenu de l'objet tag, pour chaque tag détecté, avec notamment

- Le numéro du tag,
- La position en pixels du centre du tag et de ses coins, dans le repère (u,v) de l'image,
- Une matrice de translation pose_t, coordonnées du centre du tag dans le repère (Xc,Yc,Zc) de la caméra dont le centre est le centre du capteur de la caméra.
- Une matrice de rotation pose_R, coordonnées des axes du tag dans le repère des axes de la caméra.

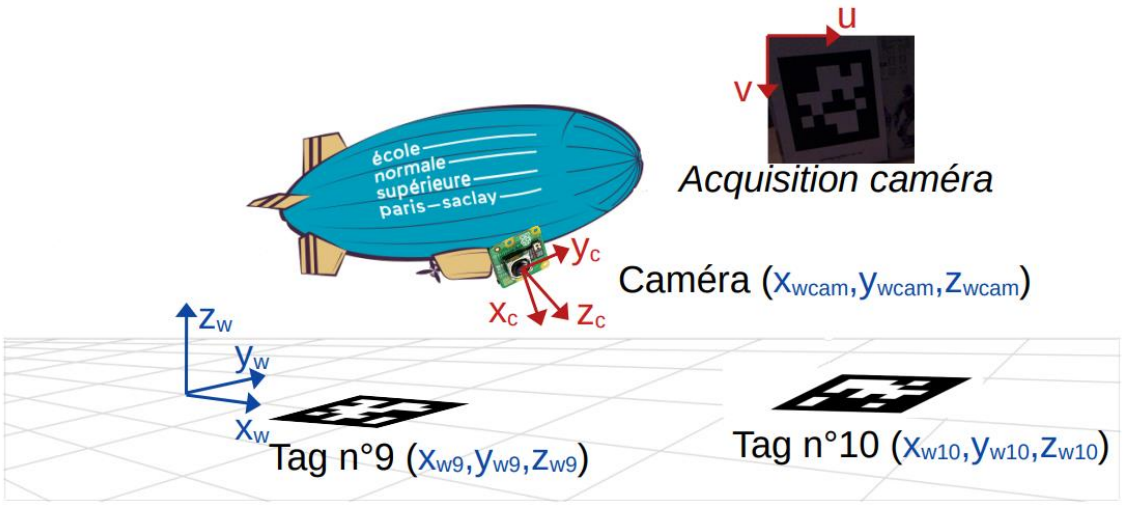
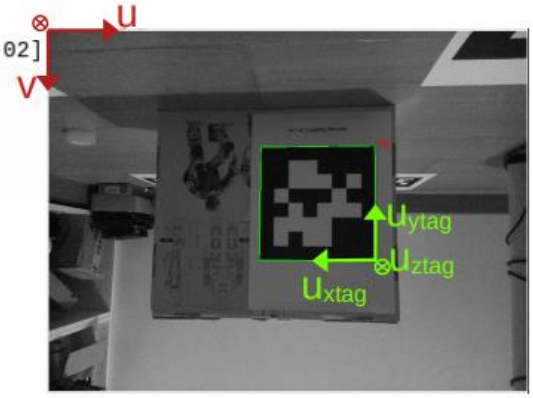


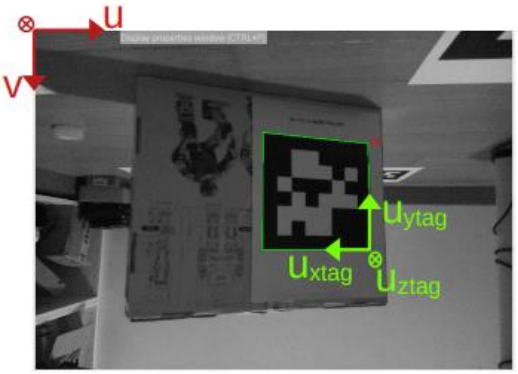
Figure 9 : Axes de l'image, de la caméra et du référentiel terrestre

Quelques détections permettent de bien comprendre la signification des coordonnées des axes du tags dans le référentiel de la caméra :

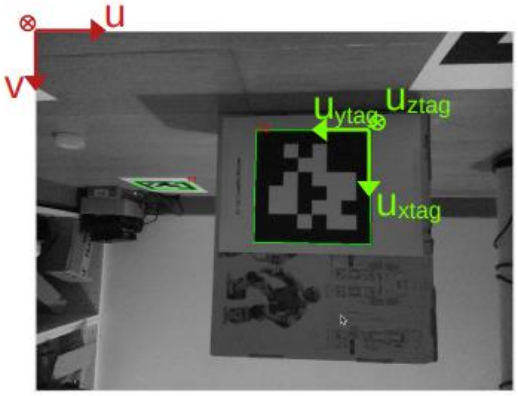
```
pose_R = [[-9.96885448e-01  6.51301391e-03  7.85937908e-02]
 [ 6.74304299e-05 -9.96513164e-01  8.34356563e-02]
 [ 7.88631648e-02  8.31810912e-02  9.93409033e-01]]
pose_t = [[ 0.01811647]
 [-0.00917807]
 [ 0.55366039]]
```




```
pose_R = [[-0.92367668  0.00126739 -0.38317069]
 [-0.04245575 -0.99417574  0.09905607]
 [-0.38081346  0.10776358  0.91835076]]
pose_t = [[ 0.04097382]
 [-0.00696748]
 [ 0.53778519]]
```




```
pose_R = [[-0.01377302 -0.99860436  0.05098666]
 [ 0.99379263 -0.0080414   0.11095742]
 [-0.11039256  0.05219839  0.9925164 ]]
pose_t = [[ 0.03070509]
 [-0.0348385 ]
 [ 0.55431973]]
```




```
pose_R = [[-0.01298112 -0.99973005  0.01926975]
 [ 0.9960856  -0.01124385  0.08767589]
 [-0.08743555  0.02033245  0.99596266]]
pose_t = [[-0.13856745]
 [-0.03145634]
 [ 0.57234996]]
```

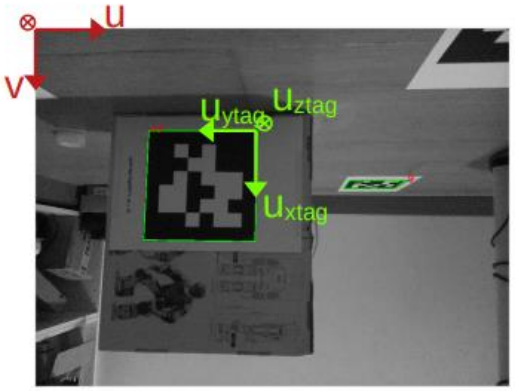


Figure 10 : Différentes détection avec des angles du tag 10 différents

3.5 - Localisation

Une fois la détection des tags fonctionnelle, on s'intéresse à la localisation, à partir des matrices pos_t et pose-R des tags détectés. À partir des positions connues des tags dans le référentiel terrestre (Xw,Yw ,Zw) et de leur position dans le repère de la caméra (Xc,Yc,Zc), l'algorithme donne la position et l'orientation de la caméra dans le référentiel terrestre (Xw,Yw ,Zw).

```
import cv2
from picamera2 import Picamera2
import time
from pyapriltags import Detector
import numpy as np

picam2 = Picamera2()
WIDTH,HEIGH=3280,2464
picam2.configure(picam2.create_preview_configuration({'size':(WIDTH,HEIGH)}))
picam2.start()

at_detector = Detector(families="tag36h11",nthreads=1,quad_sigma=0.0,refine_edges=1,\
decode_sharpening=0.25,debug=0)

fx=2.54535633e+03
fy=2.54786131e+03
cx=1.56751977e+03
cy=1.27877378e+03
#coefficients de distorsion de la camera
dist=np.array([ 0.16388869,-0.21043407,0.00532856,-0.00619909,-0.15718788])
mtx=np.array([[fx,0,cx],[0,fy,cy],[0,0,1]]) #matrice de la camera

#Positions des tags dans l'environnement
listePoints3D = {9:(0,0,0),11:(0.70,0,0),12:(0,-0.90,0),13:(0.70,-0.90,0)}

def capture():
    img=cv2.cvtColor(picam2.capture_array(),cv2.COLOR_BGR2GRAY) #prise d'une photo puis correction
    img_undistorded = cv2.undistort(img, mtx, dist, None, newCameraMatrix=mtx)
    #indication de la taille des tags, lancement de la detection
    tags=at_detector.detect(img_undistorded,estimate_tag_pose=True,camera_params=[fx,fy,cx,cy],\
tag_size=0.173)
    return tags

def calculAngles(R):
    sy = np.sqrt(R[0, 0] * R[0, 0] + R[1, 0] * R[1, 0])
    singular = sy < 1e-6
    if not singular:
        x = np.arctan2(R[2, 1], R[2, 2])
        y = np.arctan2(-R[2, 0], sy)
        z = np.arctan2(R[1, 0], R[0, 0])
    else:
        x = np.arctan2(-R[1, 2], R[1, 1])
        y = np.arctan2(-R[2, 0], sy)
        z = 0
    return np.degrees(np.array([x, y, z]))

t0=time.time()
matrice=np.array([[-1,0,0],[0,1,0],[0,0,1]])
while True:
    tags=capture()
    positions=[]
    positionMoyenne=np.array([0,0,0],dtype='float64')
    angles=[]
    angleMoyen=np.array([0,0,0],dtype='float64')
    for tag in tags:
        #calcul des angles suivant Xw, Yw et Zw
        angles.append(np.array(calculAngles(tag.pose_R)))

        pose=np.dot(np.transpose(tag.pose_R),tag.pose_t)
        #formule pour trouver la position  partir du resultat apriltag
        try :
           positions.append(np.dot(matrice,np.transpose(pose)[0]+\
np.array(listePoints3D[tag.tag_id]))
        except :
           print("tag inconnu detecte : ",tag.tag_id)
    #Calcul de la moyenne des différentes positions mesurées
    for position in positions:
        positionMoyenne+=position
    for angle in angles:
        angleMoyen+=angle
    n=len(positions)
    if n!=0:
```

```
        positionMoyenne=positionMoyenne/n
        print("position et nb tags : ", positionMoyenne,n)
        angleMoyen=angleMoyen/n
        print("angle : ", angleMoyen)

    t1=time.time()
    print("temps acquisition + traitement =",str(t1-t0),"s")
    t0=t1
```

Voici un exemple de ce que renvoie le programme, lors d'essais au-dessus de 4 tags posés au sol. On remarque le temps {acquisition/traitement} supérieur à 1s.

```
position et nb tags :  [ 0.12183679 -1.10036623  1.09074508] 2
angle :  [-39.07364088  -4.633135    -0.0574566 ]
temps acquisition + traitement = 1.1130125522613525 s
position et nb tags :  [ 0.11667911 -1.11900408  1.08101764] 4
angle :  [-32.11642046  -5.49557211  89.53985238]
temps acquisition + traitement = 1.0861926078796387 s
position et nb tags :  [ 0.11579877 -1.12883579  1.05652832] 3
angle :  [-33.9097842   -5.51460797  59.61656629]
temps acquisition + traitement = 1.1049537658691406 s
```

Figure 11 : Résultats renvoyés par le programme localisationApriltags

4 - Validation expérimentale

En plus des essais de localisation sur un dirigeable, des essais ont été menés pour utiliser cette technologie pour le repérage précis d'un robot par rapport à son chargeur.

Les tags sont fixés au mur, au-dessus de l'emplacement auquel doit se rendre le robot. Ils sont tous orientés de la même manière de sorte d'avoir les axes orientés de façon identique, pour faciliter l'exploitation des matrices de rotation des axes renvoyées par le détecteur Aprlitag.

Le robot étant stable et le sol horizontal, l'angle intéressant pour le guidage est alors l'azimut.

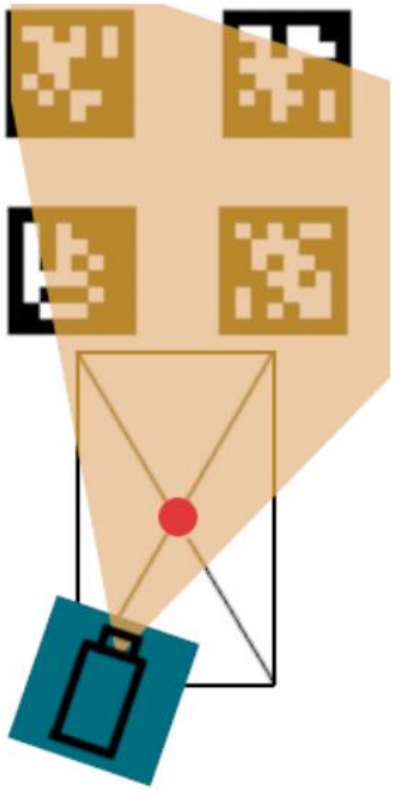


Figure 12 : Schéma du montage expérimental robot, emplacement recharge et tags

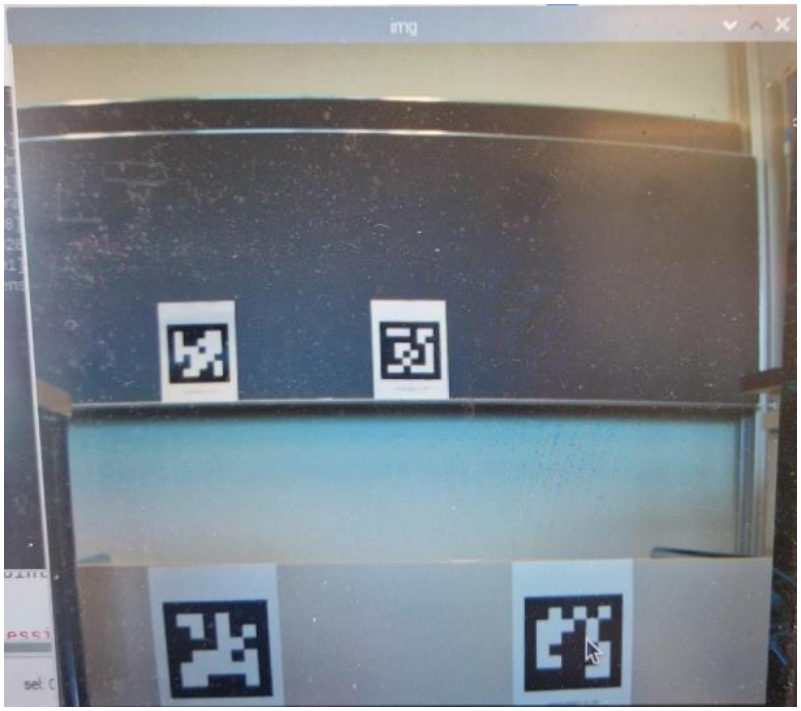


Figure 13 : affichage d'une acquisition caméra

On relie une caméra à notre Raspberry pi pour effectuer la détection des Apriltags. Comme on peut spécifier les coordonnées des Apriltags, on les placera verticalement pour qu'ils soient visibles de loin. Il aurait été même possible de les placer au plafond. Comme la matrice de rotation renvoyée est celle des Apriltags par rapport à la caméra, et qu'on les place tous avec la même orientation, on obtient une moyenne de la matrice de rotation du mur par rapport à la caméra : l'angle qui nous intéresse pour le guidage est alors l'azimut [Y,Z] que l'on obtient par : Azimut = tan-1 (e/b). Et on appliquera alors un angle opposé à cet azimut pour s'aligner avec le mur.

On mesure alors à la fois la précision de détection, si les valeurs de la caméra sont justes et la précision de suivi, si le robot arrive correctement au centre de la place :

Distance réelle(cm)	Détection(cm)
100 ; 0	102,1 ; 1,7
100 ; 30	101,7 ; 34
100 ; -30	102,6 ; -28
50 ; 0	48,3 ; 1,5
50 ; 30	47,7 ; 33
50 ; -30	50,2 ; -26

Figure 14 : Résultat des mesures de position

5 - Conclusion

Cette ressource s'est intéressée à la problématique de localisation dans l'espace à partir d'une unique caméra et des étiquettes Apriltags.

L'objet *detector* d'Apriltag utilise les paramètres intrinsèques de la caméra, obtenus lors de la calibration, pour obtenir le vecteur de translation et la matrice de rotation du tag dans le repère de la caméra. La réciproque permet d'obtenir la position de la caméra dans le repère terrestre.

La précision s'améliore avec le nombre de tags visibles mais il est possible d'effectuer une localisation avec un seul tag.

Le temps d'acquisition et calcul est très important (plus de 1 seconde). Il est possible de le réduire en diminuant la résolution de la caméra. Il doit être possible de limiter la liste des tags détectables, mais cela n'a pas été testé.

6 - Bibliographie

[1] site web official du laboratoire de robotique April :
https://april.eecs.umich.edu/software/apriltag.html

[2] Dépôt Git de la course de voitures autonomes avec notamment le guide d'installation de Raspberry OS sur la Raspberry Pi 4.
https://github.com/ajuton-ens/CourseVoituresAutonomesSaclay/blob/main/Bibliotheques_logicielles/

[3] Documentation de la bibliothèque picamera2 :
https://www.raspberrypi.com/documentation/accessories/camera.html

[4] Site web du projet OpenCV : https://docs.opencv.org

[5] Page web OpenCV d'explication détaillée de la calibration :
OpenCV: Camera Calibration and 3D Reconstruction

[6] Démarche à suivre pour calibrer la caméra :
https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html

[7] Page de présentation du module pyapriltags : https://pypi.org/project/pyapriltags/

[8] Dépôt des images des différents tags apriltag : https://github.com/AprilRobotics/apriltag-imgs

[9] Programme de test de pyapriltags : https://github.com/WillB97/pyapriltags/tree/master/test

[10] Annexes : Localisation via la vision avec la bibliothèque Apriltags, G.-A. Fade, A. Juton,
https://eduscol.education.fr/sti/si-ens-paris-saclay/ressources_pedagogiques/localisation-via-la-vision-avecla-bibliotheque-apriltags

Ressource publiée sur Culture Sciences de l'Ingénieur : https://eduscol.education.fr/sti/si-ens-paris-saclay