École Normale Supérieure, SAPHIRE

Projet Cobra

Localisation - Équipe 2

DEGRAEVE Maxime, GOUPILLON Mathis
Encadrant : *JUTON Anthony*

11 juin 2024

Table des matières

1 Les AprilTags

1.1 État de l'art

Nous avons choisi d'utiliser les AprilTags pour différentes raisons :
Tout d'abord, le coût de cette technologie est très faible à part l'encre et les quelques euros de la caméra utilisée il n'y a pas de coût.
De plus, l'utilisation des AprilTags pour ce qui est de la localisation du dirigeable est sans doute la solution permettant le mieux de s'adapter à un environnement quelconque. En effet, il nous suffira de replacer les AprilTags dans le nouvel environnement lorsque l'on décide de le changer.

1.2 Présentation des AprilTags

L'AprilTag est une technologie de marquage visuel similaire aux codes QR, utilisée principalement en robotique. Elle permet une détection et une identification précises d'objets ou de points d'intérêt dans un environnement donné. Grâce à sa capacité à être repérée facilement par des caméras et à fournir des informations de positionnement précises, l'AprilTag est devenu un outil essentiel pour la navigation autonome des robots.
Les AprilTags se présentent comme de simple QR code, il en existe différentes familles, celle que nous utiliserons se nomme tag36h11.

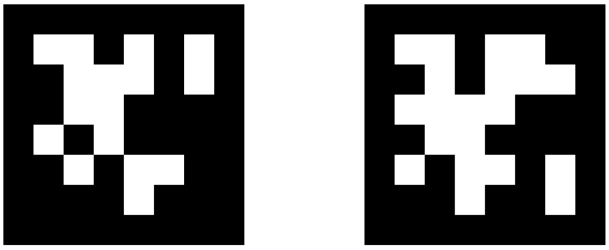

FIGURE 1 – AprilTags 36h11

1.3 Protocole

L'idée est de placer une caméra sur la nacelle du dirigeable, celle ci devra être pointé vers le sol afin d'avoir dans son champ de vision un maximum d'AprilTags.
Chaque AprilTag devra être associé à une position connue dans le **Repère Réel**, ainsi nous pourrons déduire la position de la caméra par rapport à l'origine de ce **Repère Réel**.

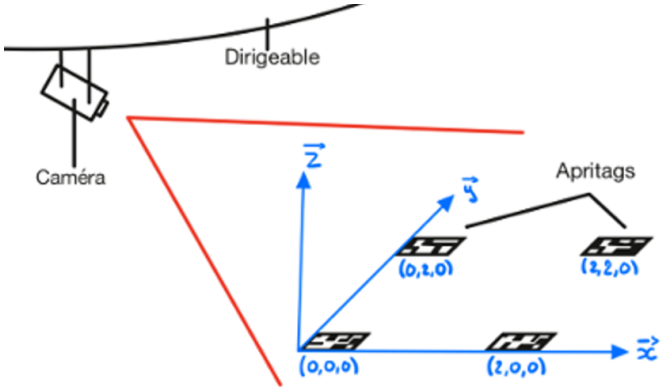

FIGURE 2 – Schéma du protocole

Nous avons alors dû recouvrir l'atrium d'AprilTags, en veillant à les placer minutieusement tout en relevant leurs coordonnés dans un repère que nous avons pris soin de définir.

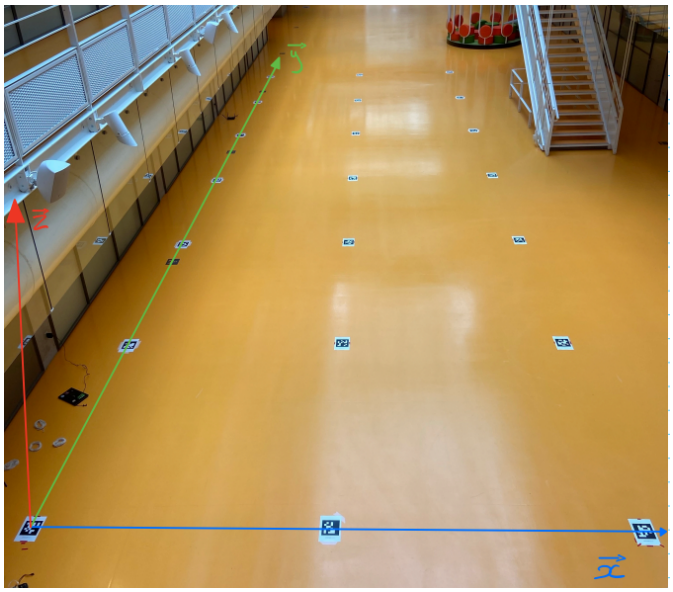

FIGURE 3 – AprilTags dans l'atrium

Pour la suite de l'implémentation en python nous avons décider d'utiliser la famille de tag "tag36h11" et de les imprimer sur des feuilles A4 de sorte à ce que les tags fassent environ 17cm de coté (cette donnée est mesurée précisément à chaque nouvelle impression d'AprilTags et est renseignée dans le code du détecteur à chaque fois).

1.4 Mise en place sur python

La mise en place de la détection des AprilTags sur python fut légèrement laborieuse mais après différents essais nous avons décidés de nous baser sur le module **dt-apriltags** de duckietown. Pour la capture d'images nous utilisons la bibliothèque **picamera2** associé à une caméra "Raspberry Pi Camera Module 2".

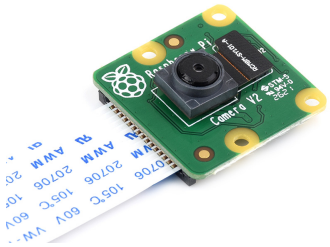

FIGURE 4 – Raspberry Pi Camera Module 2

Dans un premier temps il nous a fallu calibrer la caméra à l'aide de cv2 pour trouver la matrice intrinsèque de cette dernière :

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

FIGURE 5 – Matrice intrinsèque de la caméra

Dans cette matrice sont présentes les données :

$$\begin{cases} \text{(fx,fy) : longueur focale (horizontale,verticale) en pixels} \\ \text{(cx,cy) : position du centre optique en pixels} \end{cases}$$

Le détecteur d'AprilTags est donné sous forme de classe python du module dt-apriltags et est créé une seule fois au début de l'algorithme, en renseignant quelques paramètres (que nous ne développeront pas ici) ainsi que la famille de tags utilisée. Ensuite la fonction "detect" de l'objet Détecteur est utilisé après chaque nouvelle prise de vue pour détecter les AprilTags visibles et les analyser, c'est dans cette fonction que l'on doit préciser à chaque fois les paramètres intrinsèques de la caméra, la taille des tags ainsi que l'image.

Une image est donc prise en échelle de gris puis envoyé dans le détecteur d'AprilTags préalablement créé pour obtenir en sortie l'id de l'AprilTag ainsi que différentes valeurs mais les seules dont nous nous serviront sont "pose_R" ainsi que "pose_t" qui sont respectivement la position de l'AprilTag par rapport à la caméra dans le Repère de la caméra et la matrice d'orientation de l'AprilTag par rapport au repère de la caméra.

1.5 Exploitation des résultats du code

Après avoir obtenu la position des AprilTags par rapport à la caméra dans le **Repère caméra** (pose_t) ainsi que la matrice d'orientation du **Repère de l'AprilTag** vers le **Repère caméra** noté (pose_R) pour chaque AprilTags. Il nous faut déterminer la position de la caméra par rapport à l'origine du **Repère Réel**, pour cela il faut tout d'abord déterminer la position de la caméra par rapport à chaque ApriTags dans le repère réel ainsi :

On note *pose* la position de la caméra par rapport à un AprilTag dans le repère réel :

$$pose = -(pose\_R)^T \cdot (pose\_t)$$

Ce vecteur de position est ensuite ajouté aux coordonnées réelle de l'AprilTag par rapport au quel la caméra se repère, on a donc :

$$Position = pose + position_{\text{apriltag}}$$

Cependant, nous avons remarqué expérimentalement que dans le repère choisit que les coordonnées de *pose* ne correspondaient pas à celle attendues nous y avons alors appliqué une matrice de rotation de la forme :

$$\text{matrice} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

On obtient donc l'expression de la position de la caméra par rapport à l'origine dans le **Repère réel** suivante :

$$Position = matrice \cdot pose + position_{\text{apriltag}}$$

Ainsi ce processus est répété pour chaque AprilTags visibles, on moyenne chaque position obtenues pour gagner un maximum en précision.
Tout ceci est enfin répété à chaque image de la caméra ce qui nous permet d'avoir environ 15 résultats de positions par seconde.

1.6 Résultats

Ce système est alors fonctionnel et nous permet alors d'obtenir des résultats cohérents et précis vis à vis de la position du dirigeable.
Nous avons alors pu mener différents essais nous menant à estimer une erreur de position maximale de moins de 10cm.

2 Bibliographie

— Librairie dt-apriltags utilisée : https://github.com/duckietown/lib-dt-apriltags
— Calibration de la caméra : https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html