Système de tracking temps réels pour la robotique en essaim

L'objectif est de mettre en place un système de tracking visuel permettant d'obtenir en temps réel les positions de chaque robot présent dans l'arène. A cette fin, on disposera de plusieurs caméras fixées au plafond au-dessus de l'arène, ainsi que d'un moyen de distinguer les robots. Chaque robot sera identifié par un code visuel (p.ex. une variation de QR-code) qui sera imprimé sur une feuille d'environ 10cm par 10 cm, laquelle sera placée sur la tête du robot (donc visible par la caméra). Il s'agit donc d'un problème de tracking sans occlusion.

Etude bibliographique

Etat de l'art en matière de marqueurs robustes aux changements

Robust Marker-Based Tracking for Measuring Crowd Dynamics, Mono-spectrum marker : an AR marker robust to image blur and defocus, Automatic detection and tracking of planar markers in a stereo vision system

Travail Réalisé

Configuration des machines de travail avec l'intégration des librairies OpenCV, Tkinter, Python Imaging Library et la librairie nécessaire au fonctionnement d'une Kinect, OpenKinect.

Développement d'un algorithme de détection multi-camera et d'une interface utilisateur permettant la visualisation de la détection et de la position des robots. Le programme est multi-support (testé sur Windows et Linux), est développé en Python et fait intervenir les librairies citées plus haut.

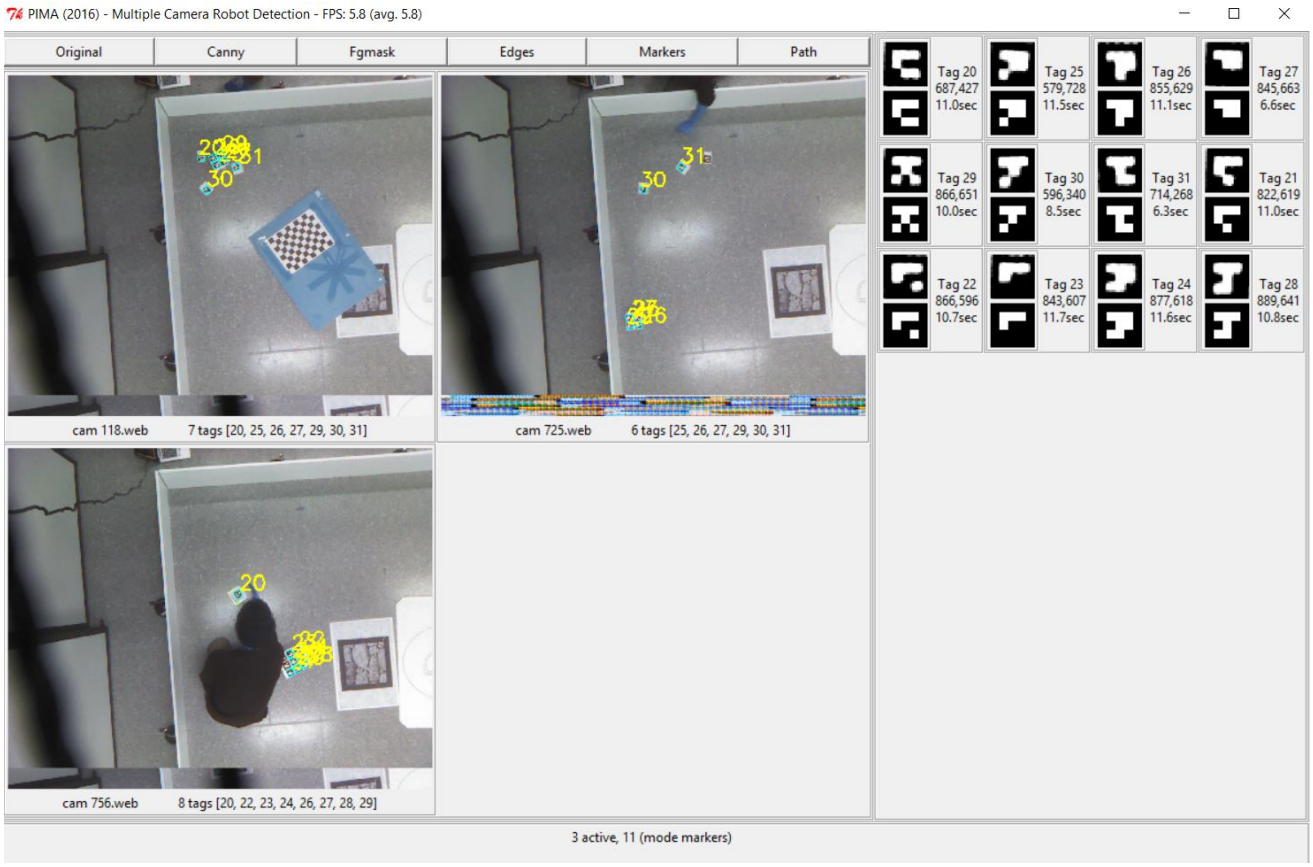


Figure 1 Fenêtre utilisateur avec à gauche les différentes caméras et à droite les marqueurs détectés

Interface, évaluation et options

Une interface avancée avec analyse en temps-réel a été développé avec présentation (console uniquement, pour l'instant) du framerate en fonction du nombre de robot, le nombre de code visuel détecté par rapport au nombre de quadrilatères détecté, deux graphes qui montre la détection au cours du temps d'un code visuel (à sélectionner), avec en X le temps, et en Y la position x du robot pour le premier graphe et en y pour le second.

Une option de visualisation du chemin parcouru des robots avec indication sur le type de mouvement (chemin réel ou interpolation si perte de la détection au cours d'un laps de temps) est aussi présent.

Ajouter un plot pour visualiser le nombre de marqueur détecté sur une caméra (en fonction de la distance) pour comparer avec précision les solutions.

Choix du code visuel et similarité

Le marqueur utilisé est un tag de taille 3x3 plus la bordure de taille 1 : l'encodage se fait sur 2^9 bits moins les symétries de rotation. Différents types de code visuel ont été essayés dont un type de marqueur qui permet la correction d'erreur *[1]*, un autre code où l'on réserve 4 cases pour détecter la rotation et ainsi augmenter la rapidité de la détection (pas de nécessité de faire des rotations sur le tag).

Enfin, afin de choisir les codes il faudrait permettre à l'utilisateur de choisir ceux qui sont les moins ambiguës et ainsi calculer une distance de similarité entre marqueurs, par exemple si l'utilisateur n'a besoin que de 5 marqueurs, l'algorithme va lui fournir les 5 marqueurs qui ont une similarité la plus faible entre eux. Le choix des tags peut maintenant se faire en fonction de ce critère

Il y a eu deux options et implémentations possible du choix d'un nombre de tag. L'un est semi-exact mais de complexité combinatoire et l'autre est de type Monte-Carlo.

Pour savoir si deux tags sont similaires il me faut définir une métrique adaptée, la distance euclidienne n'est pas très informative car elle ne tient pas compte de la position dans la matrice et il me fallait une distance qui soit représentative de la distance qu'il y aura entre les tags dans la video et non pas seulement entre les matrices de tag.

Algorithme du calcul de la similarité

Distances essayées : Euclidienne, Hausdorff (qui donne une information bien plus utile que l'euclidienne)

Distance courante : Je pars du fait que deux tags qui peuvent être confondu ont très souvent plusieurs blocs de même couleur en commun disposés à la même place.

Je propose donc la métrique suivante :
- Entre deux tags, je compte les blocs de même couleur consécutifs en X, dès qu'un bloc n'est pas de la même couleur j'ajoute le dernier compte dans une liste, je recommence jusqu'à la fin de la matrice.
- Je fais la même chose mais en Y.
- Ensuite je mets au carré ces deux vecteurs pour fortement punir le nombre de série consécutive.
- Je somme les deux vecteurs
- Je fais la somme du vecteur obtenu afin d'avoir un score entier.

Algorithme du choix d'un sous-groupe de tag avec la plus faible similarité

Option 1 :

- Calculer l'ensemble des combinaisons possibles d'une taille demandé (par l'utilisateur).
- Calculer la similarité inter-tag sur l'ensemble des tags du sous-groupe.
- Faire la moyenne des similarités trouvés entre chaque tag (sachant que la distance que j'ai proposé se fait entre deux tags seulement).
- Trouver la combinaison qui minimise cette moyenne.

Avantages/Désavantages : Trouve le sous-groupe optimal mais est très long à calculer (complexité exponentielle O(n^k))

Option 2 :

Choisir un tag aléatoirement.
Choisir un seuil de tolérance sur la similarité du groupe.
Tant que le groupe de tag à rendre n'est pas de taille demandée :
    Choisir un tag aléatoirement parmi ceux non essayés ou dans le groupe de tag à rendre
    Calculer la similarité du groupe
    Si la similarité est inférieure au seuil
        Ajouter le tag au groupe
        Réinitialiser les tags déjà essayés et le nombre de tour de boucle fait
    Si le nombre de tour de boucle est supérieur au nombre de tags restant
        Augmenter la tolérance (et donc le seuil de similarité)
    Sinon ajouter le tag en cours à la liste des tags déjà essayé

Avantages/Désavantages : Beaucoup plus rapide, résultats satisfaisants

Première itération du système de détection

Dispositif de capture vidéo : Utilisation de la caméra *Génius WideCam F100*

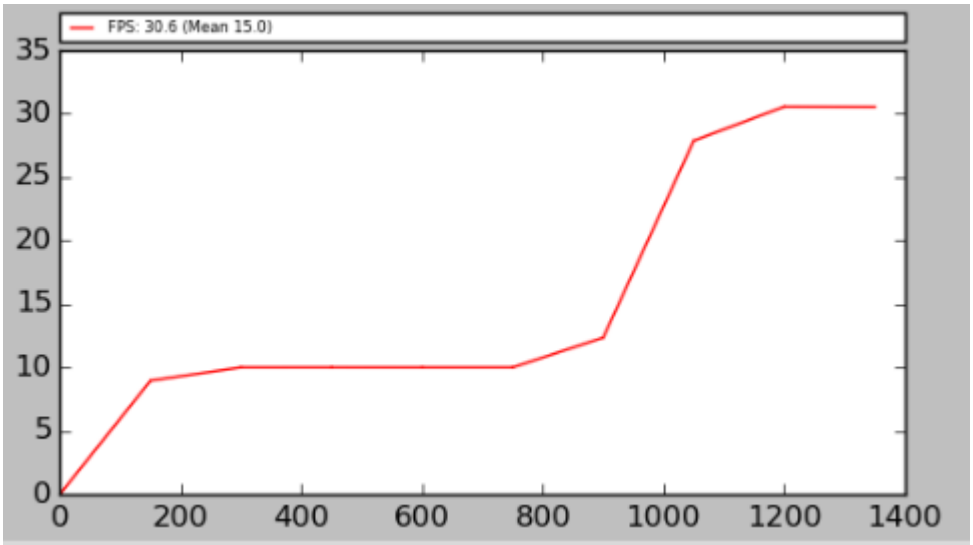
Image Sensor	1080p Full HD pixel CMOS
Video resolution	CIF/VGA: Up to 30fps 720P HD: Up to 30fps 1080p FHD: Up to 30fps

Algortihme de détection :

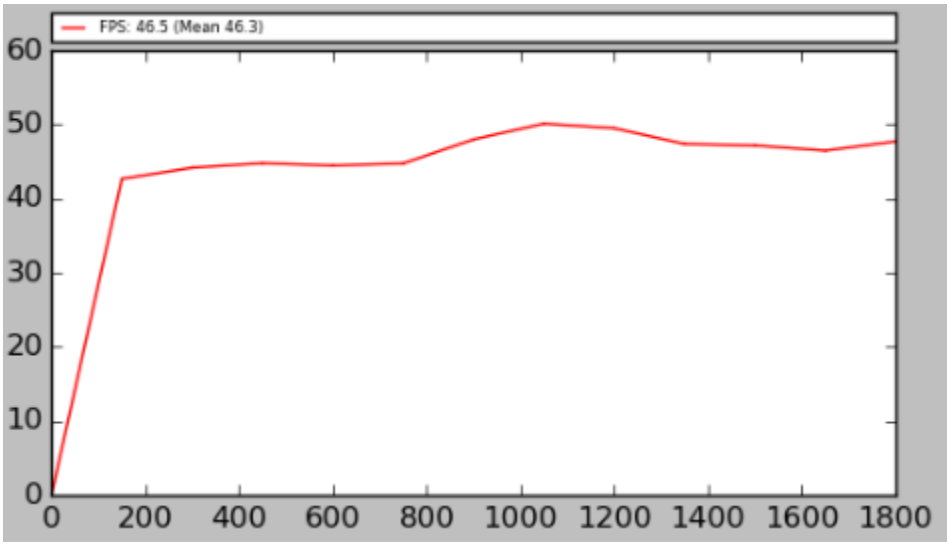
L'image est dilatée/érodée afin de faciliter la détection des contours de nos marqueurs (wiki)
L'image est convoluée avec un filtre gaussien de taille 3x3 pour réduire le bruit
Un filtre Canny est appliqué pour la détection de contour
L'algorithme de Suzuki (84) est utilisé via la fonction findContours d'OpenCV afin de récupérer les contours
On discrimine les contours selon leur géométrie (on cherche des parallélogrammes)
On effectue une homothétie de perspective sur les parallélogrammes et on regarde si une de leur rotation égale un de nos marqueurs dans la base de donnée

Résultats :

Framerate sur deux vidéos jouées simultanément (60fps) en illumination de jour : environ 46 traitements/secondes



Framerate sur une caméra en temps réel (30fps max.)
- en faible illumination environ 10 traitements/secondes, [0 à 900 tours]
- en forte illumination (de jour) environ 30 traitements/secondes. [900 à 1400 tours]



La distance de détection dépend aussi de l'illumination de la pièce, pour l'instant le programme est capable dans de bonnes conditions d'éclairage de détecter des marqueurs à moins de 30 à 40cm.

L'objectif en cours est donc de minimiser l'impact de l'illumination sur la détection et sur le taux de rafraichissement et d'augmenter significativement la distance de détection.

Seconde itération du système de détection

Dispositif de capture vidéo : Utilisation de la caméra Kinect version 1

Image Sensor	480p CMOS
Video resolution	480p : Up to 30fps

- Changement d'un filtre passe-bas median par un filtre gaussien avec un kernel (5,5) avant l'application du Canny
- Divers changements dans les étapes de la détection.
- Normalisation des intensités du quad candidat avant de décider s'il s'agit bien d'un code visuel et duquel.
- Dépot du programme sur le GitHub du projet (nekonaute/thymioPYPI/robocologieimage/P-ima_2016
- Détection à des intervalles de temps et non plus en continu (en support du tracking)
- Méthode d'apprentissage pour la détection afin d'éviter les faux-positifs et accélérer le programme.

Résultats :

KINECT_ISIR
8381 (Total) 8129 (Correct 96.99%) 252 (Error 3.01%) 0 (Maybe 0.0%)

Dispositif de capture vidéo : Utilisation de la caméra Kinect version 2

KINECT_V2_ISIR
15607 (Total) 15310 (Correct 98.1%) 297 (Error 1.9%) 0 (Maybe 0.0%)

Je considère une détection comme correct si le tag détecté est bien un tag que j'ai mis dans la vidéo. Généralement les tags mis dans la vidéo ont une similarité la plus faible possible et une faible quantité de tags sont utilisés afin de garantir une bonne représentativité de l'évaluation (la probabilité qu'un tag ou qu'un quad soit confondu avec un tag présent à tord devrait être faible).

Méthodes de décision

1. Méthodes exactes
    a. Méthode d'analyse locale d'intensité
    b. Méthode d'analyse de forme

2. Méthodes par apprentissage et prédiction
    a. Machines à vecteurs de support (Machines de Restricted Boltzmann)
    b. Réseaux de neurones profonds

3. Méthode de tracking
    a. Optical Flow
    b. Block-matching
    c. Soustraction d'images

Méthode par apprentissage et prédiction

Cette méthode me permet de prédire si le tag candidat que je regarde est vraiment un tag et lui affecter son identifiant correspondant.

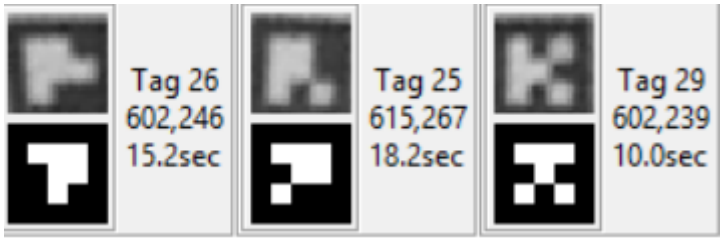
Dans mon jeu qui servira pour l'apprentissage j'aurai en label les identifiants de tags et en donnée les images que j'ai collecté via mon algorithme de détection.

Par exemple voici un exemple train:

labels = [26, 25, 29]
data = [[image26_1, ..., image26_n], [image25_1, ..., image25_m], [image29_1, ..., image29_l]]

Une fois le modèle appris, je pourrai directement utiliser ma fonction de prédiction pour classer mon tag ou le rejeter, ce qui en termes de temps devrait être plus rapide et fiable que de la détection.

Et voici la tête des images des tags candidats (voir les images du haut, l'homothétie vers un carré a déjà été appliqué).



J'ai filtré un maximum possible les faux-positifs afin d'avoir un apprentissage cohérent.

Méthodes exactes

Algorithme de la méthode d'analyse locale d'intensité

Je détecte les contours de l'enveloppe du tag, je dois faire ça pour ensuite décider s'il s'agit bien d'un tag et trouver l'identifiant associé,
et c'est cette étape d'identification que je souhaite améliorer tant au niveau de la fiabilité que de la rapidité.
Dans le cas présent afin de décider si mon candidat est un tag, je prends l'image de ma projection 3d vers un carré qui ressemble à la figure ci-dessous.

Puis je la standardise (l'image est normalisée, je soustrais à l'image sa moyenne et je la divise par son écart type).
Et ensuite je divise l'image en bloc et je prends la valeur médiane de l'intensité dans ce bloc pour ensuite faire un threshold et le comparer à mes tags de référence. De ce fait si un seul des blocs est incorrect le tag entier l'est aussi. Afin de trouver une solutions plus robuste ou en tout cas complémentaire, j'ai pensé qu'utiliser une approche statistique peut être un bénéfice (temps + adaptabilité).

Intégration de ces méthodes décision

Le système courant utilise à la fois une méthode exacte et une méthode par apprentissage et prédiction afin d'assurer une précision plus grande dans les résultats. De plus avoir ces deux méthodes permet d'avoir une évaluation d'une méthode par rapport à une autre et est ainsi utile pour l'amélioration.

Plan de travail

- Fusion des images et des informations des détecteurs et calibration pour correction de la déformation induite par le capteur et sa position dans l'espace.

- Intégrer une méthode de tracking afin de réduire davantage le coût de l'algorithme.

- Créer des jeux de données pour l'apprentissage propres, complets et représentatif de plusieurs scénarii et configurations (faible et forte luminosité, angles très grands, etc…). Les jeux de données actuels sont issus de la méthode de détection avec des contraintes très fortes sur l'attente du résultat, de ce fait il peut y avoir des faux positifs dans le jeu, de plus l'ensemble des tags n'a pas été appris. Les paramètres sont différents de ceux utilisés lors d'une détection en temps réel où l'on cherche le plus de positions possibles.

- Je compte aussi ajouter un label avec uniquement des faux-positifs, afin qu'il apprenne à reconnaitre ce qui n'est pas un tag. En ce moment, à l'étape de prédiction, ceux qui ne sont pas détectés comme tag sont ceux dont la probabilité d'appartenir à un tag est inférieur à 0.80 pour chaque tag.

Je tiens à remercier Séverine Debuisson, Nicolas Bredèche et Arthur Bernard pour leurs précieuses aides et conseils ainsi que Emilie, Daphné et Yann avec qui on s'entraide toute les semaines.

Elias Rhouzlane