

HERAIZ-BEKKIS Daphné & FARJAS Emilie

Robocologie Rapport

**Paris
9 Mai 2016**

Table des matières

I- Système d'identification	
II- Détection de Robots	
III- Description de la fouille d'une image	
IV- Luminosité dynamique	
V- Performances du programme	
VI- Distance de détection	
ANNEXE 1 : Documentation	

Introduction

Implémentation d'une solution qui permet à un robot d'identifier en temps réel les robots qui se trouvent dans son champ de vision et leur disposition.

Cette application permet donc, à partir d'un Thymio-2 connecté à une Raspberry Pi possédant une caméra, de détecter un robot dans une image et d'y lire ses informations.

Grâce au Thymio-2, constituant les Robots, ceux-ci sont mobiles. Ils se déplacent donc dans un environnement précis. Ici, une arène 3mx3m est utilisée.

Ce système ne doit pas dépendre des interférences extérieures de l'environnement telles que la luminosité ou le décor; il ne doit pas non plus dépendre du nombre de Robots présents autour de lui. Des Robots peuvent être ajoutés ou retirés à tout moment de l'expérimentation dans l'arène.

A partir de cela, le Robot a une vision restreinte, il n'a donc pas une connaissance globale de son environnement.

L'application finale ne fonctionne que sur **sol plat** avec des tags **à la même hauteur**.

Nous avons tout d'abord élaboré un système d'identification du robot.

I- Système d'identification

1. Contraintes du système

Le système d'identification des robots doit transporter le numéro du Robot, il doit être portatif, léger et remarquable. Il doit aussi être unique à chaque Robot et facilement analysable par un homme. Il doit être possible d'en générer plusieurs simplement et que cela ne bloque en rien l'analyse des Robots déjà en exécution.

2. Modélisation choisie

À partir de ces contraintes, deux zones principales du système nous sont apparues comme évidentes :

- une zone de nommage : **l'identifiant** du Robot (id)
- la face du robot sur laquelle le système est apposé : la **direction** (ou orientation) du Robot

Ce système, permettant alors de tagger un Robot, est donc considéré comme un tag.

Cependant, n'étant pas à l'abris d'une occlusion certaine de ce tag, il nous est donc apparu comme nécessaire d'y incorporer un système anti-occlusion : une bordure. Celle-ci permettra de marquer les contours délimitants un tag, entourant ainsi les zones d'identification et de direction de ce dernier. Par la même occasion, cela spécifiera la définition d'un tag, arborant ainsi un design propre à celui-ci.

3. Élaboration d'une palette de tags

Une option aurait été l'utilisation du QRCode. Cependant, l'encodage visuel de l'information y est trop petit. De plus, la représentation des données contenues dans cette image est trop complexe pour nos besoins.

À partir des critères définis ci-dessus, nous nous sommes inspirées du principe d'encodage binaire de l'information du QRCode et avons conçu plusieurs maquettes de tags, constitué des trois zones principales suivantes :

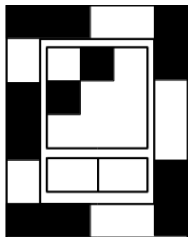
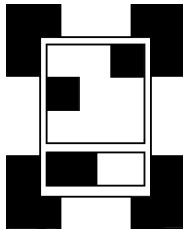
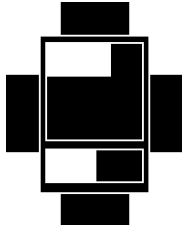
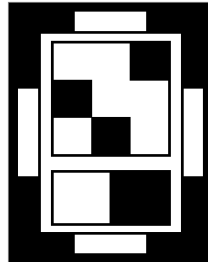
- ◆ un contour extérieur avec un schéma particulier : la ***bordure anti-occlusion***
- ◆ une zone centrale haute : l'***identifiant du robot***
- ◆ une zone centrale basse : l'***orientation du robot***

La direction peut prendre quatre valeurs:

- [blanc;blanc] → "00" → face avant
- [blanc;noir] → "01" → côté gauche
- [noir;blanc] → "10" → côté droit
- [noir;noir] → "11" → face arrière

L'identifiant, quant à lui, est constitué de **NB_LIGN_CASE** lignes et de **NB_COL_CASE** colonnes, donc de **NB_LIGN_CASE x NB_COL_CASE** cases. Ce qui lui permet de représenter de 0 à $2^{(\text{NB_LIGN_CASE} \times \text{NB_COL_CASE})-1}$ numéros possibles de Robot. Chaque case représente une puissance de 2, une case noire correspondant à 1×2^X et une case blanche correspondant à 0×2^X .

Le calcul se fait alors en partant de la case en bas à droite - correspondant à la puissance 0 (2^0), s'incrémentant de droite à gauche sur ligne puis de bas en haut. L'identifiant du Robot est alors la somme des puissances représentées, comme le montrent les tag ci-dessous (exception faite du tag inversé, tag3) :

			
<p>TAG1 (n°160)</p> <p>Tag abandonné dû à la connotation nazie de la bordure.</p>	<p>TAG2 (n°96)</p> <p>Modification du schéma de la bordure extérieure.</p> <p>→ ne permet pas de gérer l'occlusion correctement (contours du tag trop peu marqués)</p>	<p>TAG3 (n°384)</p> <p>Inversion binaire des couleurs du TAG2 (<i>blanc</i> ↔ <i>noir</i>)</p> <p>→ <i>meilleur résultat que le TAG2 dans l'interprétation des données.</i></p> <p>→ <i>ne permet pas de gérer l'occlusion correctement (les coins sont blancs)</i></p>	<p>TAG4 (n°98)</p> <p>TAG2 avec ajout d'une bande noire externe dans la zone de contour</p> <p>→ meilleur identification du tag</p> <p>→ bonne gestion de l'occlusion grâce à son contour</p>

Le **TAG4** a donc été retenu.

4- Etalonnage du tag à différentes distances

Une fois ce tag définit, nous avons créé plusieurs images de ce tag à des distances différentes et l'avons mesuré dans l'image. Ces données nous servent d'étalonnage des dimensions de ce tag à 20 cm, 30 cm, 40 cm, 50 cm, 65 cm et 80 cm.

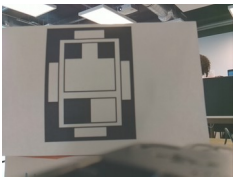
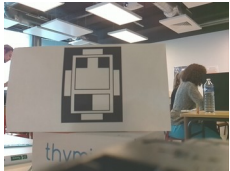
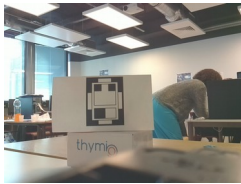
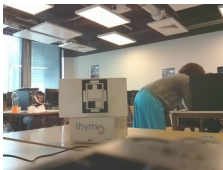
Les résultats et images qui vont suivre ont été obtenus avec les paramètres suivants:

Résolution : 640x480

Luminosité : 60

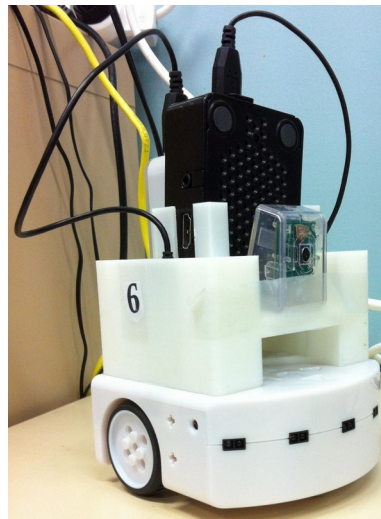
Taille de référence d'un tag sur papier : 7.8 cm x 10.4 cm

Taille de référence d'un tag en pixels : 225 x 296

Distance caméra-tag	Largeur dans l'image (pixels)	Hauteur dans l'image (pixels)	Image (exemple)
20 cm	270 pixels	350 pixels	
30 cm	175 pixels	225 pixels	
40 cm	125 pixels	170 pixels	Non sauvée
50 cm	100 pixels	130 pixels	
65 cm	75 pixels	95 pixels	
80 cm	55 pixels	75 pixels	Non sauvée

5- Création des chapeaux de Robots

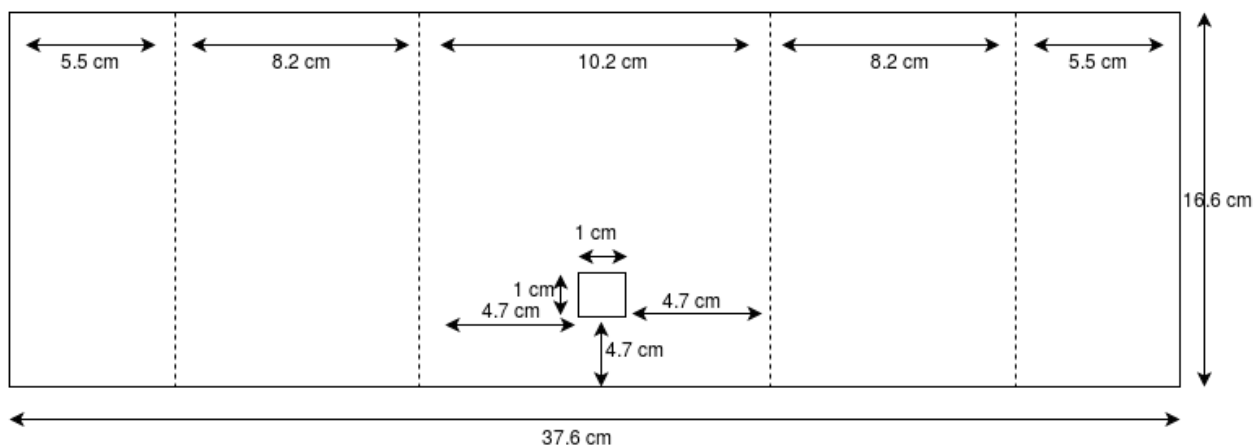
Pour des raisons esthétiques et pratiques, nous avons été créé un habillage pour les Robots. En effet, ces derniers étaient un peu dévêtus :



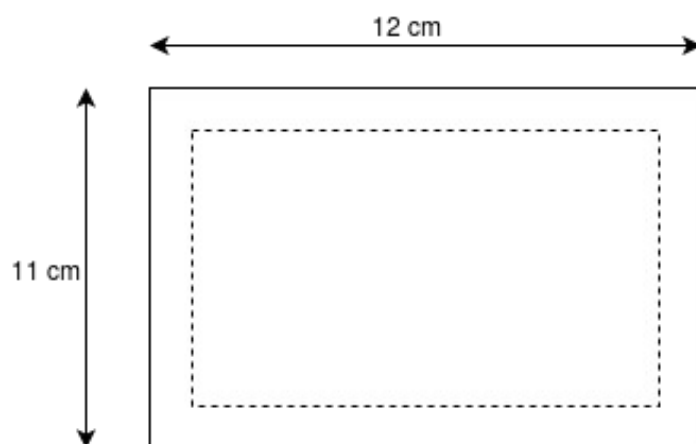
Nous avons créé des coiffes sur-mesure pour les Robot. Ces élégants chapeaux sont agrémentés des tags créés, sur chaque face. Le chapeau est supporté par la structure du Thymio-2 et englobe l'ensemble câbles - Raspberry Pi - batterie PowerBank et caméra. La caméra permettant la visualisation de l'environnement extérieur au Robot, un emplacement dédié est prévu à cet effet : une ouverture est prévue centrée sur la face avant du Robot.

Ci-dessous les patrons de conception :

Patron principal :



Patron du couvercle :



Celui-ci pourra être utilisé pour y placer un système de reconnaissance aérien.

II- Détection de Robots

Le flux vidéo pris par la Raspi-Camera, caméra de la Raspberry Pi, est considéré image par image. Pour détecter des Robots dans son entourage proche, le Robot doit donc analyser chaque image. Nous traiterons ces photos afin de déterminer la présence ou non d'un ou plusieurs tags et d'en extraire les informations nécessaires.

Pour ce qui est du traitement des images prises, des opérations sur celles-ci doivent être effectuées (filtres, crop, ...).

1- Grandes étapes du programmes

- Capture d'une image
- Traitement de l'image
- Détection et reconnaissance des tags
- Obtention des informations à partir de chacun de tags

2- Traitement de l'image

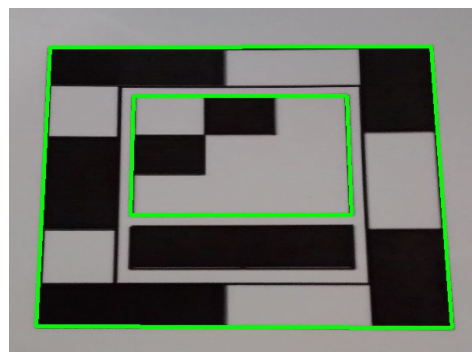
Pour simplifier tout traitement et limiter les calculs, les manipulations se font sur des images converties en nuances de gris.

a) Algorithmes pré-existants

En traitement d'images, les algorithmes de détection de schémas, tel qu'un tag comme nous l'avons décrit, sont généralisés à la détection des contours. Dans notre cas, ceci peut s'avérer une bonne chose du fait de la forme rectangulaire du tag et de ses éléments internes.

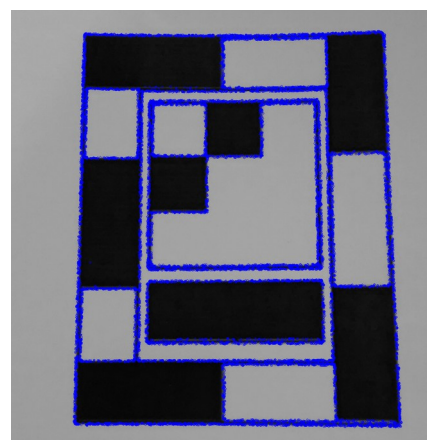
SIFT (Scale-Invariant Feature Transform) Algorithm : Un algorithme de recherche de points d'intérêts indifférents à l'échelle de l'objet dans l'image.

Temps d'exécution sur Raspberry Pi :
15 secondes/image



FAST (Features from Accelerated Segment Test) Algorithm : Une amélioration de SIFT qui est plus rapide et qui donne de bons résultats. Mais qui malgré tout n'est pas assez rapide pour être utilisé sur Raspberry Pi pour un traitement en temps réel.

Temps d'exécution sur Raspberry Pi :
10 secondes/image



Cependant, comme dit précédemment, ceux-ci ne sont pas assez spécifiques et demandent alors un temps d'exécution relativement long compte tenu de nos besoins.

De ce fait, nous avons été contraintes de créer par une suite de manipulation des images notre propre algorithme qui répond spécifiquement et uniquement à nos besoins : équilibre entre précision et rapidité.

b) Algorithmes personnalisés

En imagerie, un algorithme est défini par une combinaison de filtres, qui sont eux-mêmes des opérations transformant les images. Les algorithmes ainsi construits sont basés sur cette définition. L'ensemble des combinaisons possibles étant imposant, il nous a fallu plusieurs tests, en voici trois, dont celui choisi.

Our Algorithm :

- *égalisation d'histogramme* : marquage des contrastes de gris, différentiation des niveaux de gris
- *seuillage binaire automatique des valeurs* : récupération d'une image noir & blanc
- *gradient Sobel* : marquage des contours en lignes (y) et colonnes (x)

Elias Algorithm :

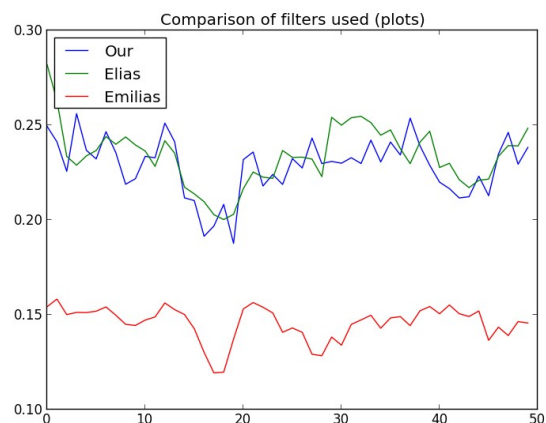
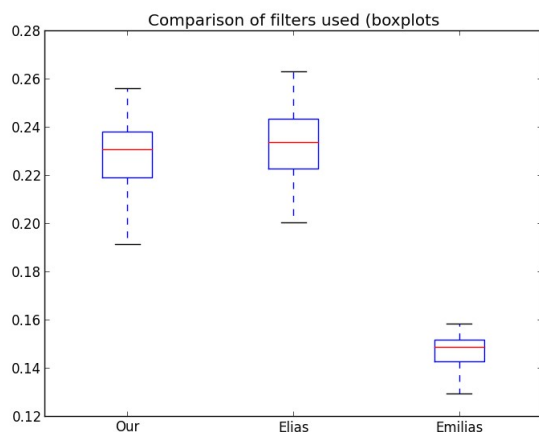
- *floutage gaussien* : marquage des contrastes, renforcement des séparations des objets
- *calcul de la valeur médiane de l'image*
- *calcul de l'intervalle de gris* dont les bornes sont des pourcentages de la médiane
- *utilisation de l'intervalle de gris* calculé pour marquer les contours

Emilias Algorithm :

- *floutage gaussien* : marquage des contrastes, renforcement des séparations des objets
- *seuillage binaire automatique des valeurs* : récupération du seuil, calculé automatiquement, comme borne supérieure de l'intervalle de gris
- *calcul de la borne inférieure de l'intervalle de gris* qui est un pourcentage de la borne supérieure précédemment calculée
- *utilisation de l'intervalle de gris calculé* pour marquer les contours

Une des contraintes définies dans le cahier des charges est la capacité du Robot à traiter un minimum d'images en une seconde (au moins 5). De ce fait, la rapidité est un des critères de choix d'un algorithme; le second étant la précision.

c) Choix d'un meilleur algorithme et optimisations



Des graphes ci-dessus, on peut observer que deux des algorithmes décrits plus haut sont en concurrence (Our et Elias), tandis que le troisième, Emilias, s'exécute en un temps presque deux fois plus court. De cela, on peut en déduire quel est, pour l'application, l'algorithme le plus approprié. Ces écarts sont dus à la simple utilisation de modules différents et à des valeurs fixées ou dynamiques.

Ainsi, on peut aussi comparer les diverses manières de réaliser un seuillage en traitement d'images - opération réalisée très souvent pour l'application.

Threshold (ou **seuillage d'image**) est la méthode la plus simple de segmentation d'image. À partir d'une image en niveaux de gris, le seuillage d'image remplace un à un les pixels d'une image à l'aide d'une valeur **SEUIL** fixée. Il peut donc créer une image ne contenant que deux valeurs de pixels : le noir ou le blanc (binarisation monochrome).

Il existe, dans notre cas, différentes méthodes de définition du seuil :

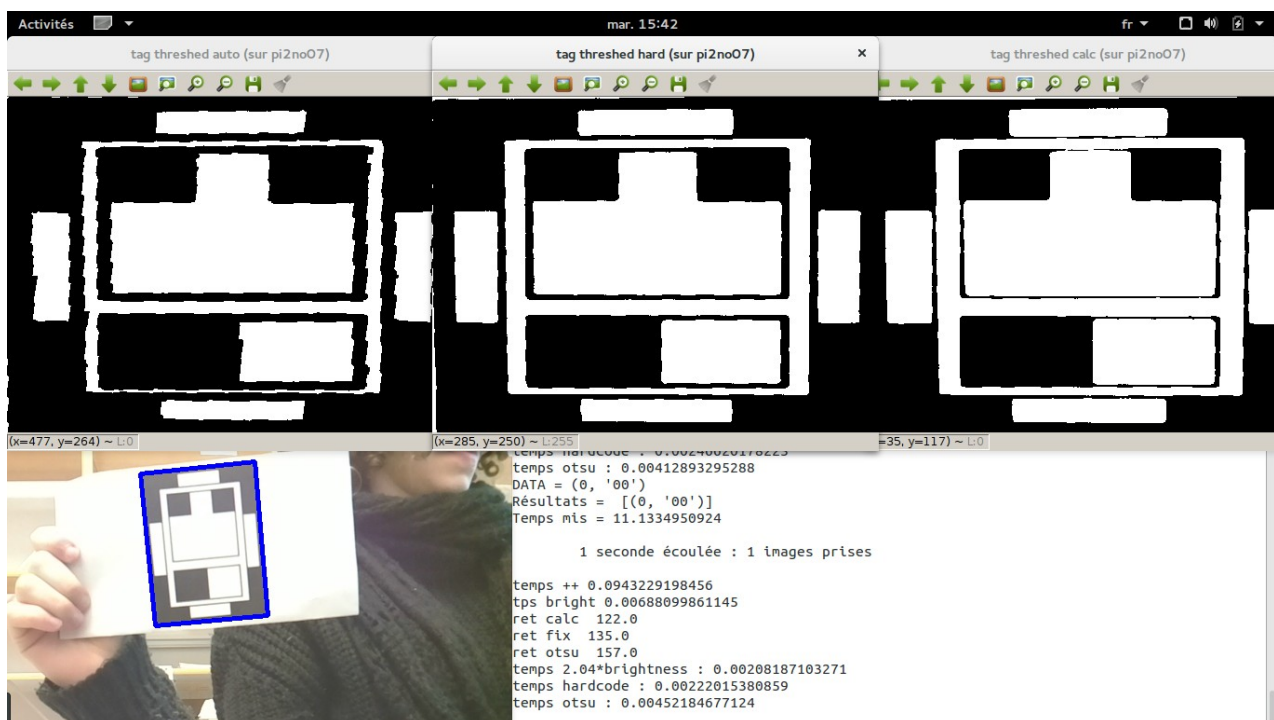
- **Binarisation d'OTSU** : calcul automatiquement le seuil, bon résultat, exécution ralentie (peu adaptée à nos contraintes)
- **Seuil fixé (valeur arbitraire)** : bon résultat en général, très rapide, peu de résultats incohérents
- **Seuil dynamique (fonction de la luminosité : $2.04 * \text{luminosité}$)** : calcul prouvé par des articles scientifiques, très rapide, mauvais résultats, beaucoup d'incohérences

Ci-dessous des résultats comparatifs des trois valeurs méthodes décrites :

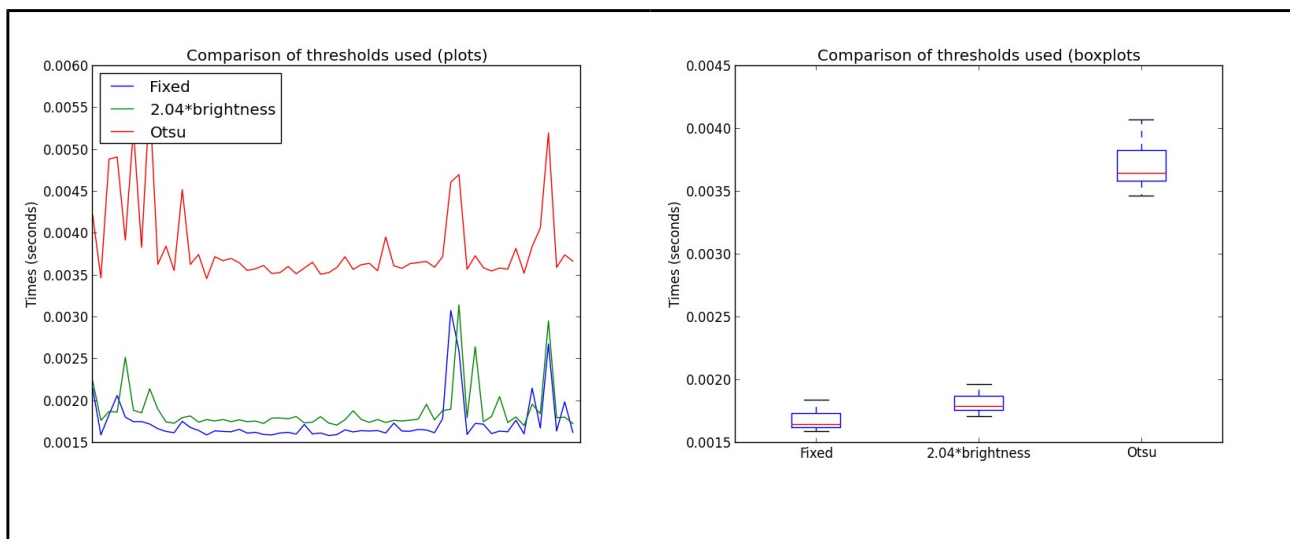
tag gauche : binarisation d'OTSU

tag centre : valeur fixe à 135

tag droit : valeur calculée en fonction de la luminosité ($2.04 * \text{luminosité}$)



Résultats graphiques:



III- Description de la fouille d'une image:

- ➔ Application de l'algorithme retenu : **Emilias**
- ➔ Récupération des parties de l'image pouvant être intéressantes
- ➔ Vérification des formes obtenues : quatre coins, parallélogramme
- ➔ Vérification des zones trouvées : dans un intervalle de confiance (20cm;80cm)
- ➔ Assurance de trouver des parallélogrammes debouts
- ➔ Validation des tags : ceux-ci sont conformes à la bordure attendue
- ➔ Séparation des zones d'information de chaque tag
- ➔ Lecture de celles-ci
- ➔ Calcul des distances aux tags
- ➔ Calcul des rotations des Robots aperçus

1. Définition d'un tag

Une partie majeure de l'algorithme est basée sur le filtrage des parties pouvant être intéressantes de l'image avec des suites de tests basés sur la définition d'un tag (sa forme, son orientation, ses dimensions).

Ainsi, un tag est défini comme un quadrilatère -une forme mathématique à quatre angles- qui est convexe, de tel sorte que sa base soit plus petite que ses hauteurs. Son périmètre et son aire sont compris dans l'intervalle de confiance (20cm; 80cm), calculés à l'avance avec des calibrations. Un tag contient aussi au moins quatre sous-parties (quatre enfants).

L'intervalle de confiance est délimité en borne inférieure par la distance minimale de visibilité d'un tag et en borne supérieure par la distance maximale d'acceptation pour les filtres utilisés.

2. Bordure et validation

La bordure du tag est la partie importante pour trier parmi des objets intéressants trouvés dans l'image ceux qui sont assurément des tags. Celle-ci se reconnaît par une alternance particulière en trois parties de noir et de blanc pour chaque côté.

3. Séparation des zones d'informations

L'identifiant du Robot et la direction du tag sont deux informations transportées par ce dernier. Leur obtention détermine la performance de l'algorithme.

La zone haute, l'**identifiant**, et la zone basse, la **direction**, possède chacune des caractéristiques et des dimensions pouvant varier. Leur identification est donc basée sur des approximations.

Ainsi, l'identifiant a un périmètre équivalent à 70% de son parent direct et une aire à environ 50%. Tandis que la direction a un périmètre équivalent à 50% de son parent direct et une aire à environ 23%.

4. Calcul de la distance

La distance entre le Robot visualisant et le tag visualisé est proportionnelle à toute autre distance possible du tag. Cette proportion se retrouve dans les dimensions du tag. Une simple application du théorème de Thalès peut être utilisée.

Nous utilisons comme tag de référence le tag correspondant à celui de la borne inférieure de l'intervalle de confiance (20cm; 80cm). Le rapport de proportionnalité est fait selon la hauteur du tag. Le choix de la hauteur en comparaison est arbitraire car celui-ci donne les meilleurs résultats jusque là, et offre plus de précision: plus de précision que la largeur car plus grand, plus de précision que le périmètre car ce dernier dépend de la largeur, il en est de même pour l'aire.

Ainsi, le calcul obtenu ressemble à:

Distance_Chchée = Distance_Min*Hauteur_Min/Hauteur_Trouvée cm

Où

Distance_Min correspond à la borne inférieure de l'intervalle de confiance (en cm)

Hauteur_Min correspond à la hauteur (en pixels) du tag dans l'image de référence

Hauteur_Trouvée correspond à la hauteur (en pixels) du tag trouvé et pour lequel on cherche la distance à laquelle il se trouve

5. Détermination de l'angle de rotation du Robot

L'angle de rotation d'un Robot est l'angle du Robot dans son référentiel, c'est-à-dire relativement à celui qui le voit, c'est l'angle duquel ce dernier devrait tourner pour avoir une direction parallèle à celle du Robot vu.

Ainsi, lorsqu'un tag arrière est reconnu juste en face, l'angle du Robot est de 0°. Tandis que s'il s'agit du tag avant qui est reconnu juste en face, le Robot est tourné à 180°; pour le tag du côté droit, il s'agit de 90°, et pour celui du côté gauche, 270°.

De ce fait, la rotation est calculée dans le sens anti-horaire: le sens trigonométrique .

(Angles visibles sur le couvercle dans l'image ci-contre)



IV- Luminosité dynamique

La gestion de la luminosité des images est une partie très importante de l'amélioration de nos résultats. Équilibrer les variations de luminosité et toujours rester dans une zone acceptable pour faciliter la phase de reconnaissance.

L'idée est que l'image ne contienne pas trop de niveaux de gris, mais qu'elle ne soit pas trop claire non plus. En effet, les images sur-exposées (trop de blanc) donnent souvent des images difficilement exploitables, il en est de même pour les images sous-exposées (trop de noir). Il faut donc conserver tout au long de l'expérimentation une luminosité médiane.

Algorithme :

luminosite initialement à INIT_BRIGHTNESS

Pour chaque image prise :

 Ajout de l'image à la liste des BRIGHTNESS_COMP dernières images prises LAST_IMGS_MEAN

 Suppression de la sixième si elle existe

Toutes les BRIGHTNESS_CHECK images :

Test de la luminosité

 S'il y a plus de BRIGHT_MIN_WRONG_IMG images parmi les BRIGHTNESS_COMP dernières contenues dans LAST_IMGS_MEAN au dessus du niveau d'acceptation (BRIGHT_MEAN_MAX):

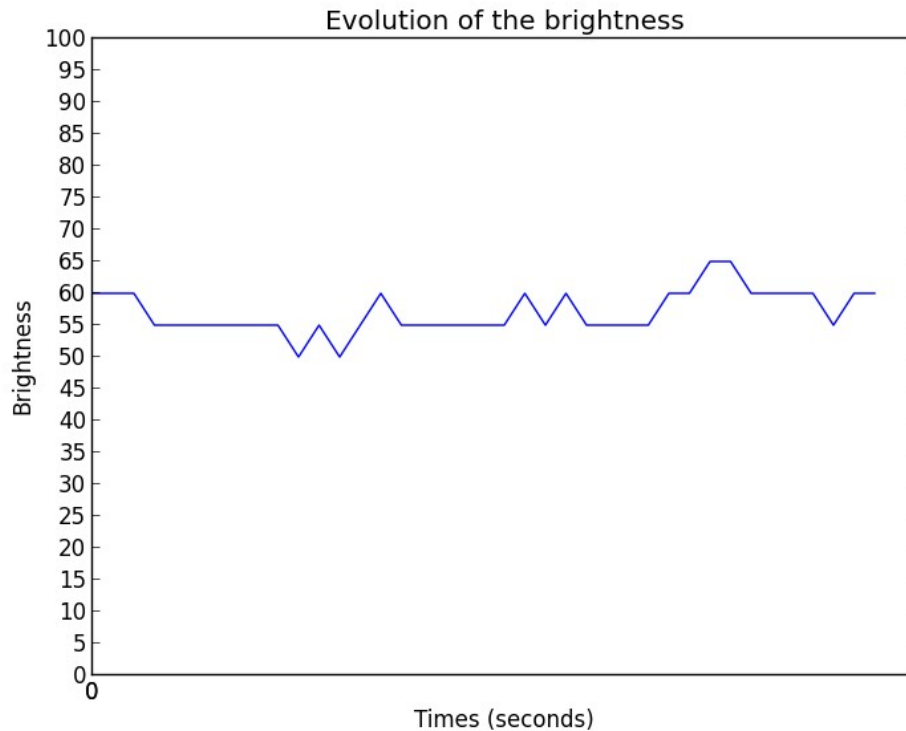
 Décrémenter la luminosité de BRIGHTNESS_STEP

 Sinon s'il y a plus de BRIGHT_MIN_WRONG_IMG images en-dessous du niveau d'acceptation (BRIGHT_MEAN_MIN):

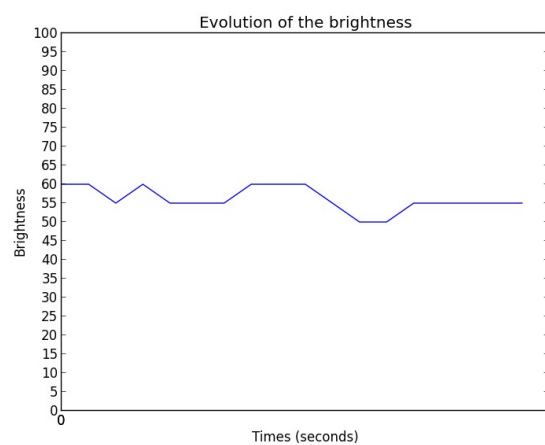
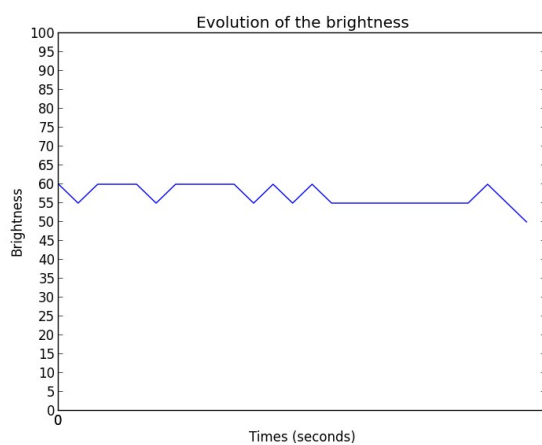
 Incrémenter la luminosité de BRIGHTNESS_STEP

 Sinon ne rien faire

À partir de cet algorithme, la luminosité s'adapte entièrement et en continu à son environnement extérieur, comme on peut le voir sur le graphe d'évolution de la luminosité en fonction du temps ci-dessous.

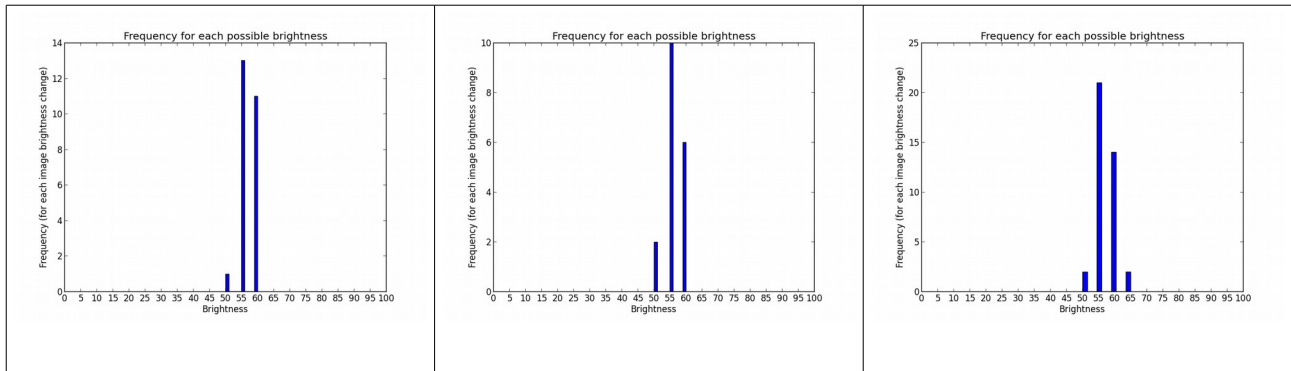


On peut voir que le programme s'adapte parfaitement à ce qu'il voit, car dans les deux exemples ci-dessous et l'exemple du dessus, il s'agit de trois Raspberry Pi ayant été démarrées en même temps et au même endroit. On observe, cependant, une gestion différente pour un même instant :

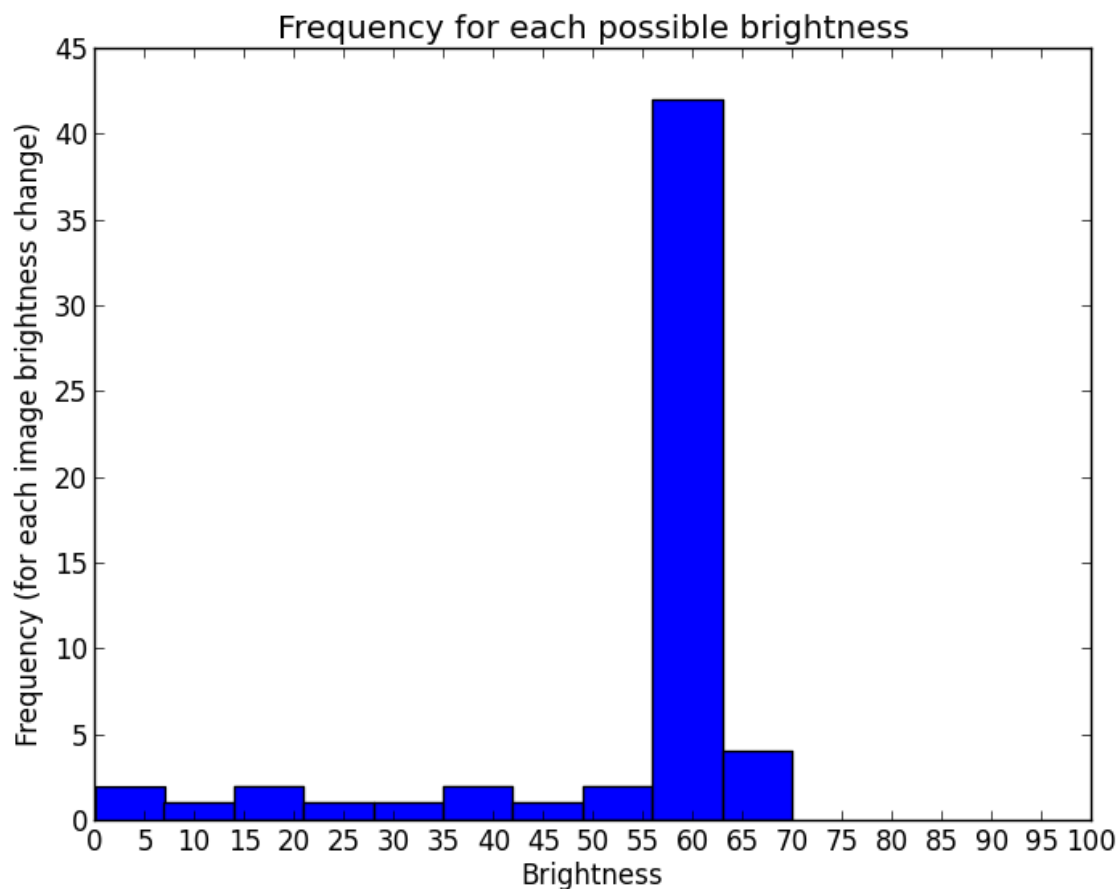


On peut aussi observer les fréquences d'utilisation des luminosités par les Robots. Ceci nous permet alors de déterminer la valeur la plus adaptée à un environnement spécifique.

Ci-dessous, trois exécutions sur trois Robots distincts en simultané et évoluant dans le même environnement:

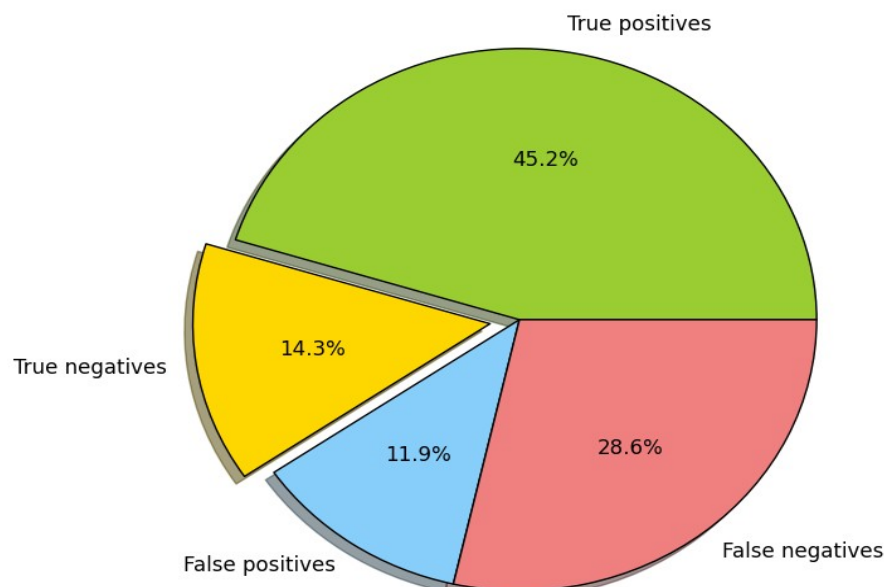


Suite à un test fait partant d'une luminosité nulle (image noire), nous observons sur l'histogramme ci-dessus un réajustement progressif de la valeur de la luminosité de la caméra.

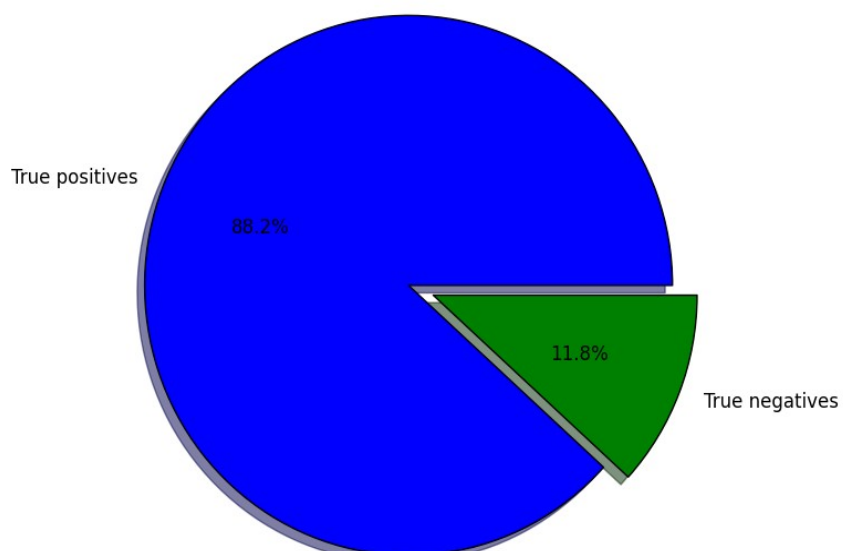


V- Performances du programme

Ci-dessous les résultats de détection obtenus par classification manuelle :



Ci-dessous les résultats attendus pour cette même classification :



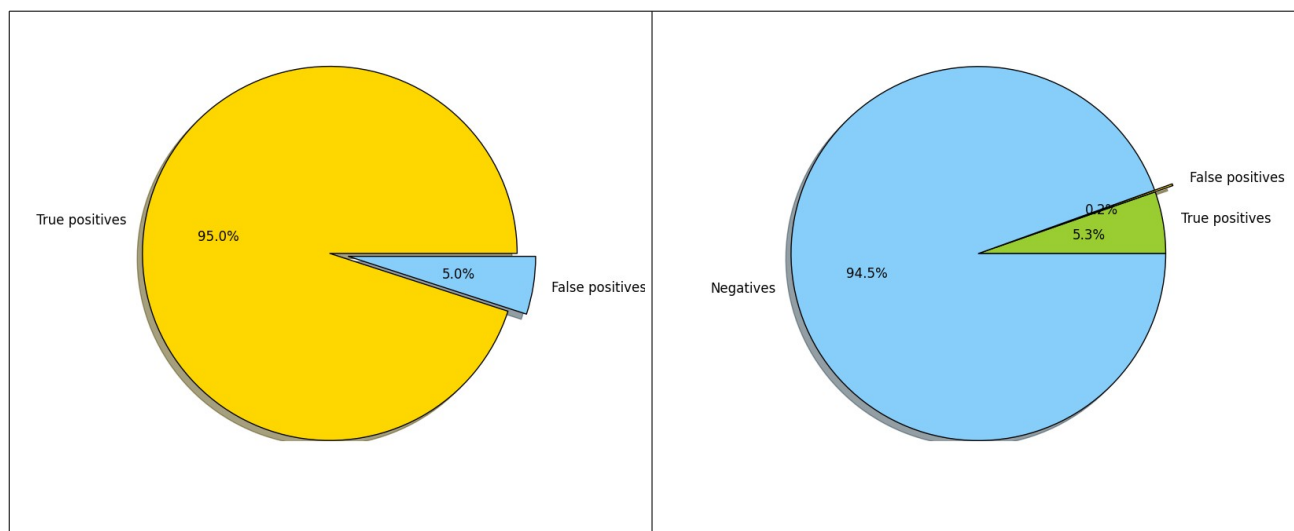
Nous pouvons remarquer que nous devrions atteindre 86% de vrais positifs, or nous n'atteignons que 45%.

Ce mauvais résultat peut s'expliquer par la mauvaise détection du tag du fait des filtres utilisés qui ne sont pas optimaux. Les contrastes obtenus à la suite de ceux-ci ne sont donc pas assez marqués.

Pas d'automatisation possible : L'environnement dans lequel les Robots évoluent est un environnement ouvert.

En effet, les caméras représentent le champs de vision des Robots, qui sont en mouvement. Cette vision a une portée locale au Robot et ne couvre pas toute l'arène. On ne peut donc pas savoir à un moment donné quel robot est dans le champs de vision d'un autre (contrairement à l'Optitrack - où les caméras sont fixes et couvrent la toute la surface de l'arène, ou encore la classification manuelle).

En tentant, une automatisation de la classification en temps réel, plusieurs données ne peuvent être interprétées correctement, car il n'y a alors aucun moyen de distinguer True Negatives et False Negatives. Nous n'avons aucun moyen de comparer dynamiquement quels Robots la caméra aurait dû voir. Nous obtenons alors:



VI- Distance de détection

Pour déterminer au mieux la distance à laquelle se trouve un tag ou encore pour filtrer les tags selon leurs dimensions, l'estimation de la distance de détection retournant ainsi l'intervalle de confiance est primordiale.

Ci-dessous, quelques exemples de détection de tag sur un Robot statique positionné à différentes distances, face au sujet. Nous pouvons évaluer l'efficacité du programme de détection de tag.

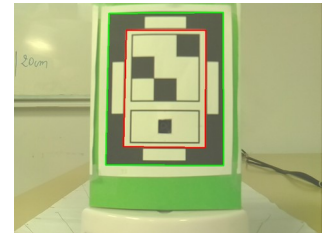
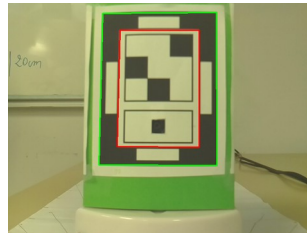
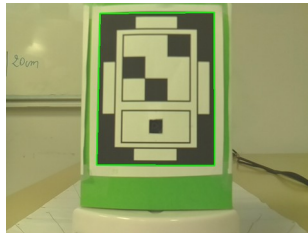
**Distances
(en cm)**

Prise 1

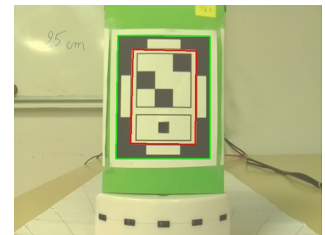
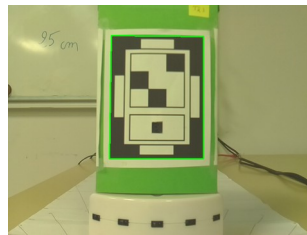
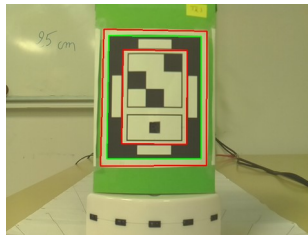
Prise 2

Prise 3

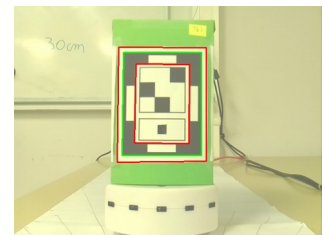
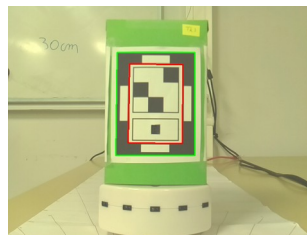
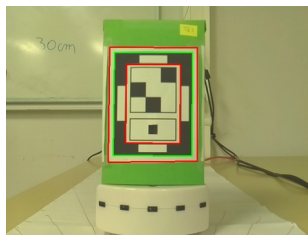
20 cm



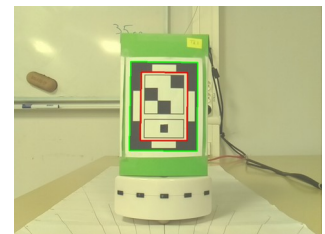
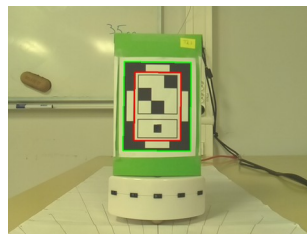
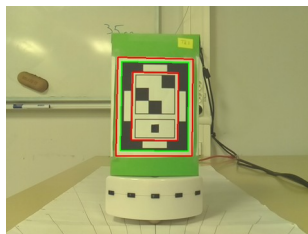
25 cm

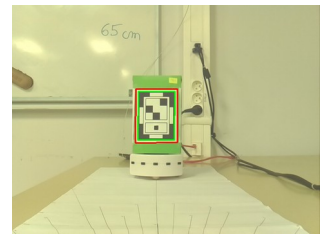
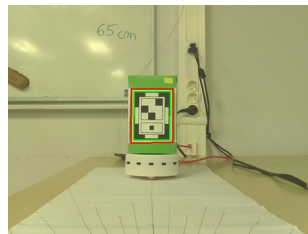
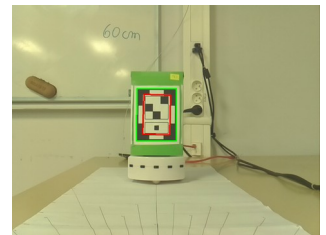
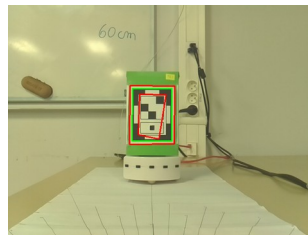

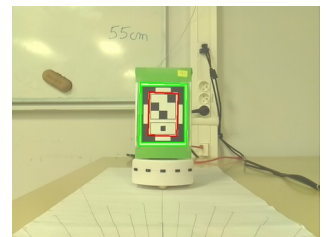
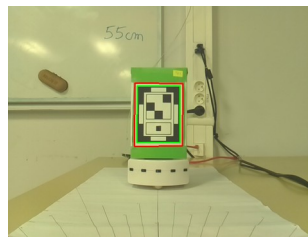
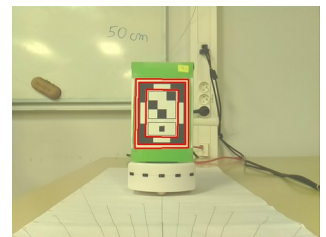
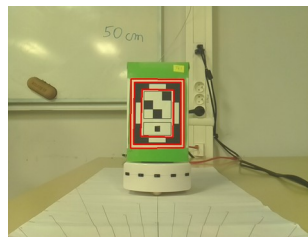
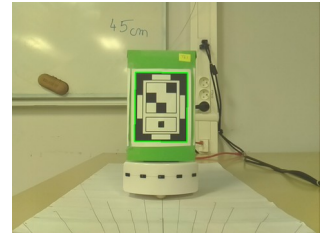
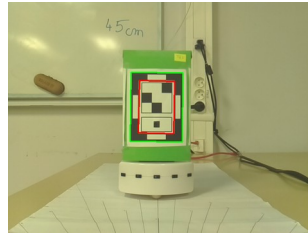
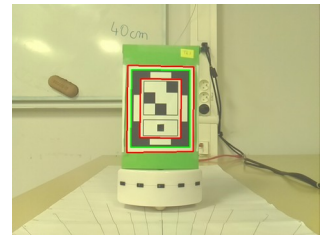
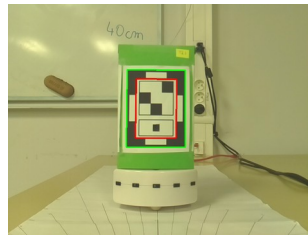


30cm

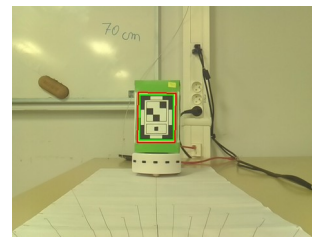
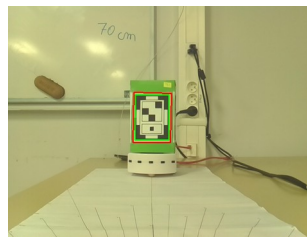
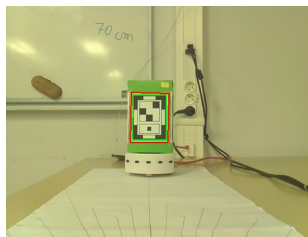


35 cm

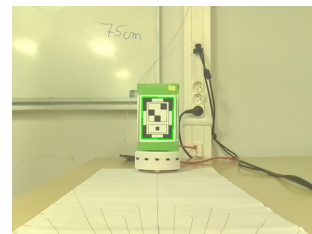
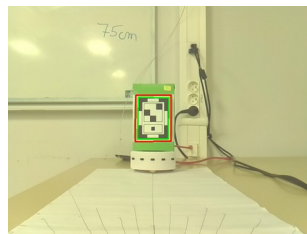
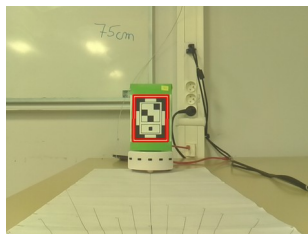




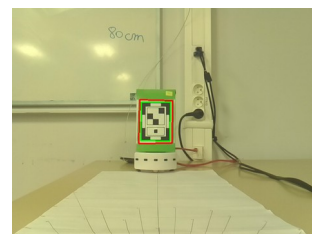
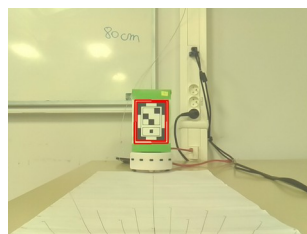
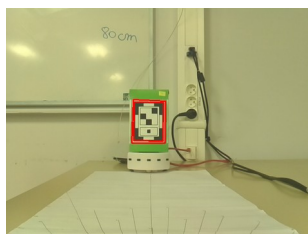
70 cm



75 cm



80 cm



ANNEXE 1 : Documentation

<https://hal.archives-ouvertes.fr/hal-01161837/document>

<https://www.thymio.org/en:asebamedulla>

<http://opencv-python-tutroals.readthedocs.org/>

https://fr.wikipedia.org/wiki/Filtre_de_Sobel

<http://docs.opencv.org/>

<http://stackoverflow.com>

<http://opencvpython.blogspot.fr>