Rapport de Projet :

Reconnaissance d'objets via Traitement d'Image

Encadré par Vincent Coelen et Thomas Danel

Damien Narbais Jaureguy
Zoé Briois

IMA4
2017 - 2018

NARBAIS-JAUREGUY Damien
BRIOIS Zoé

Remerciements

Nous remercions toute l'équipe d'ARPL pour nous avoir permis de participer à l'Open German, à Magdeburg. Ce fut une expérience pleine de découvertes et très enrichissante.

Nous sommes particulièrement reconnaissants à nos tuteurs, Thomas Danel et Vincent Coelen, pour leur aide et leur soutien tout au long du projet.

SOMMAIRE

Remerciements **1**

SOMMAIRE **2**

Introduction **3**

I. Présentation du projet **4**
I.1. Objectif et cahier des charges 4
I.2. Choix techniques 5
I.2.a. Choix matériel 5
I.2.b. Choix logiciel - langage de programmation 9

II. Solution apportée **10**
II.1. Déroulement général 10
II.2. Récupération de l'image et communication 11
II.3. Reconnaissance d'image 14

III. Programmes annexes nécessaires **16**
III.1. Réalisation des sets d'images 16
III.2. En amont de la reconnaissance d'image 17

IV. Difficultées rencontrées **19**

V. Améliorations possibles **20**
V.1. Améliorations possibles 20
V.2. Ce qui pourrait être développé pour la logistic league en lien avec le projet 20

VI. Conclusion **21**

Annexes **22**
Annexe A 22
Annexe B 23
Annexe C 24

Introduction

Dans le cadre de notre deuxième année d'études dans la spécialité Informatique Microélectronique et Automatique à Polytech Lille nous avons eu l'opportunité de choisir comme projet : la reconnaissance d'objets via Traitement d'Image. Nous avons travaillé sur ce projet tout au long du deuxième semestre.

Ce projet nous a permis d'appliquer des notions étudiées en cours : que ce soit des notions techniques de programmation ou des notions organisationnelles. Il nous a également amené à en découvrir de nouvelles.
Après avoir établi plus en détail le cahier des charges et choisit de quelle façon y répondre au mieux, nous nous sommes réparti les tâches en fonction de nos affinités.

Nous avons choisi le sujet "Reconnaissance d'objet par traitement d'image", proposé par les membres de l'ARPL (Association de Robotique de Polytech Lille), celui-ci s'inscrit dans le cadre d'un challenge présenté à la Robocup pour la catégorie Logistic League.

Dans un premier temps nous présenterons le choix de nos outils matériels et logiciels pour répondre au mieux au cahier des charges. Puis, nous expliquerons ce que fait notre programme et le code nécessaire en amont de la compétition pour que le tout fonctionne (récupération des images, création de la banque d'images…). Nous mettrons par ailleurs en évidence les difficultés soulevées par ce projet. Enfin, nous vous détaillerons les pistes d'améliorations possibles.

I. Présentation du projet

I.1. Objectif et cahier des charges

Notre projet intervient dans une compétition de robotique : la Logistic League. Pendant cette compétition, un ou des robots doivent exécuter des actions sur un terrain contenant des machines de production appelées MPS.

Les matchs sont composés principalement de deux phases : une première phase d'exploration et une seconde de production. Nous participons à améliorer la première phase d'exploration, dans laquelle le robot sur le terrain doit reconnaître les différentes MPS ainsi que leurs orientations. Nous ne nous occupons pas de cette dernière exigence.

Pour l'instant, chacune des MPS (il en existe 5 types différents) est identifiée par un AR Tag (QR Code). Chaque année, les règles de la compétition évoluent, pour anticiper un éventuel changement, les membres de l'ARPL nous ont proposés sujet suivant : réussir à identifier ces machines sans l'aide des AR Tag, le plus rapidement possible et avec une fiabilité maximale pour pouvoir communiquer au Robotino un identifiant correspondant au type de la machine.

Pour répondre à ce besoin, le choix du moyen de vision nous est laissé libre, nos tuteurs étant ouverts aux moyens employés. Nous allons donc vous présenter à travers ce rapport, notre parcours pour arriver à la solution proposée.

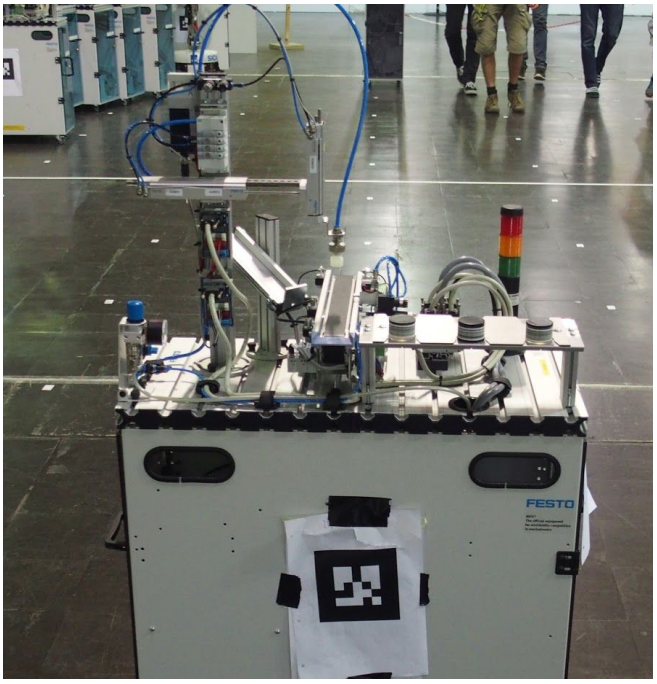


fig. 1: Exemple d'une MPS à reconnaître

I.2. Choix techniques

I.2.a. Choix matériel

Choix du mode de vision :

Pour commencer la stéréo vision cela nécessite l'acquisition d'au moins 2 images de l'objet à mesurer de 2 points de vues différents. Après la prise de vues de la MPS les coordonnées d'image des points à mesurer sont déterminées sur chacune des images puis mises en concordance.

Pour déterminer les dimensions de l'objet il est nécessaire d'étalonner le système de mesures. Cette solution ne nous semblait pas adéquate au vu de la largeur du robot par rapport à celle des MPS. En effet nous aurions eu des images trop ressemblantes en laissant le robot immobile face à la MPS avec 2 caméras placées sur des endroits différents du robot. De plus déplacer le robot aurait rendu le processus de reconnaissance plus long.

Ensuite la kinect possède une caméra RGB ainsi qu'une caméra infrarouge, cette caméra infrarouge permettrait de mesurer les différences de profondeur ce qui pourrait être avantageux pour notre projet. Par exemple cela permettrait de ne pas prendre en compte les éléments trop éloignés. Sur le même principe, il existe la caméra SR300 d'Intel. Malheureusement ces caméras nécessitent une connexion en USB 3.0, ce qui n'est pas possible avec la plupart des cartes présentes sur le marché à des prix abordables. C'est pour cela que nous nous sommes tournés vers le kit de développement d'Intel contenant une caméra SR300 et une carte de prototypage bénéficiant d'un port 3.0.

NARBAIS-JAUREGUY Damien
BRIOIS Zoé

Position de la caméra :

Une fois le matériel choisi, il a fallu positionner la caméra correctement. Pour cela nous avons procédé par étapes. Tout d'abord, pour vérifier le champ de vision de la caméra; nous nous sommes placés à 87 cm d'un mur, l'objectif parallèle à ce mur, et nous avons relevé les dimensions du quadrilatère observé. On a observé ceci :

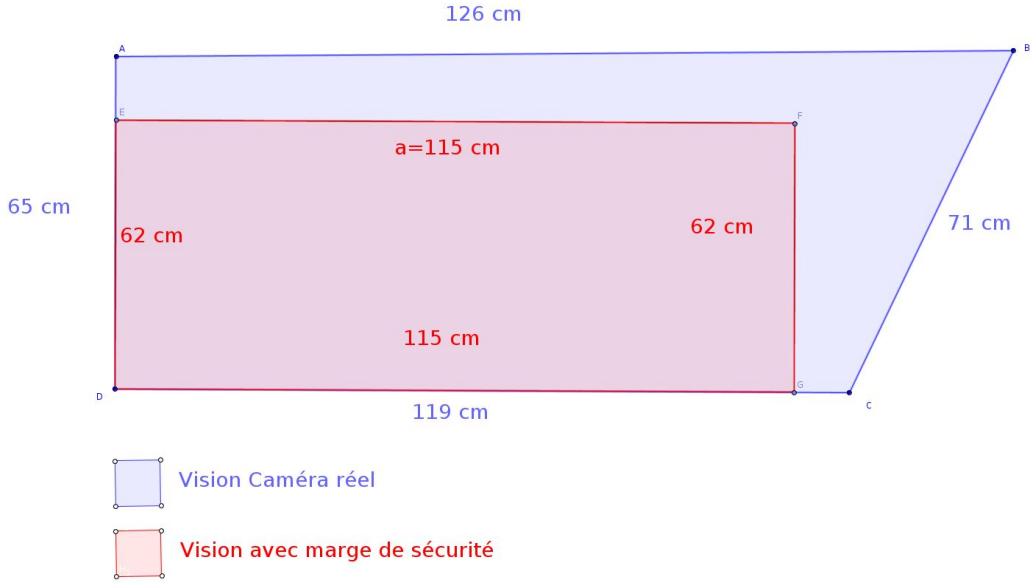


fig. 2: Schéma de la vision de la caméra réel (en bleu) et avec une marge de sécurité (en rouge)

En prenant une marge de sécurité de 5%, on a ensuite effectué nos calculs avec le quadrilatère rouge.
Vue du dessus, on obtient la vue suivante :

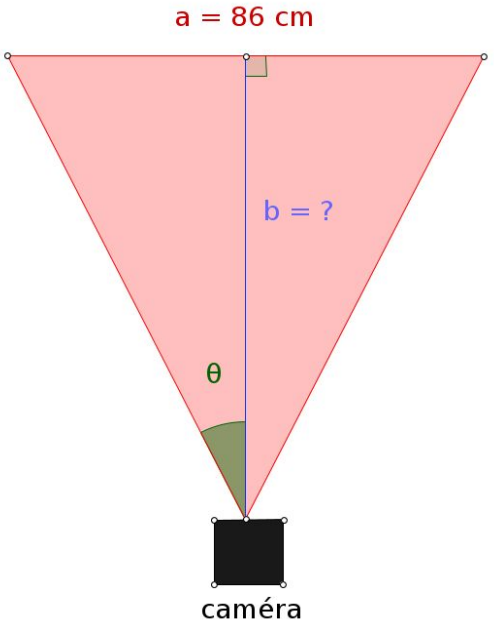


fig. 3: Schéma de la vision de la caméra vue du dessus

On doit donc trouver la longueur b pour savoir à quelle distance placer la caméra. On calcule d'abord Θ avec les données acquises avec la première étape, où b = 87 cm et a = 115 cm.

Donc on a :

$$\tan\Theta = \frac{a/2}{b}$$
$$\Theta = Arctan\left[\frac{a/2}{b}\right]$$
$$\Theta = Arctan\left[\frac{115/2}{87}\right]$$
$$\boxed{\Theta = 33,5°}$$

D'où, le b cherché vaut :

$$b = \frac{a}{2} \times \frac{1}{\tan\Theta}$$
$$b = \frac{86}{2} \times \frac{1}{\tan 33,5}$$
$$\boxed{b = 65\ cm}$$

On doit donc placer la caméra à 65 cm de la MPS. La précision est relative car nous avons travaillé avec une marge de sécurité.

Enfin, on a dû choisir la hauteur de la caméra par rapport à son plateau. En effet, la robotino porte un plateau sur lequel on peut placer les éléments qui interagissent avec lui. Pour éviter que les objets déjà en place ne gênent le champ de vision de la caméra on a donc placer la caméra sur le côté. C'est le placement du robot qui permettra d'avoir la caméra au centre de la MPS. Pour éviter que le plateau ne gêne la position de la caméra, on s'est basé sur le schéma suivant :

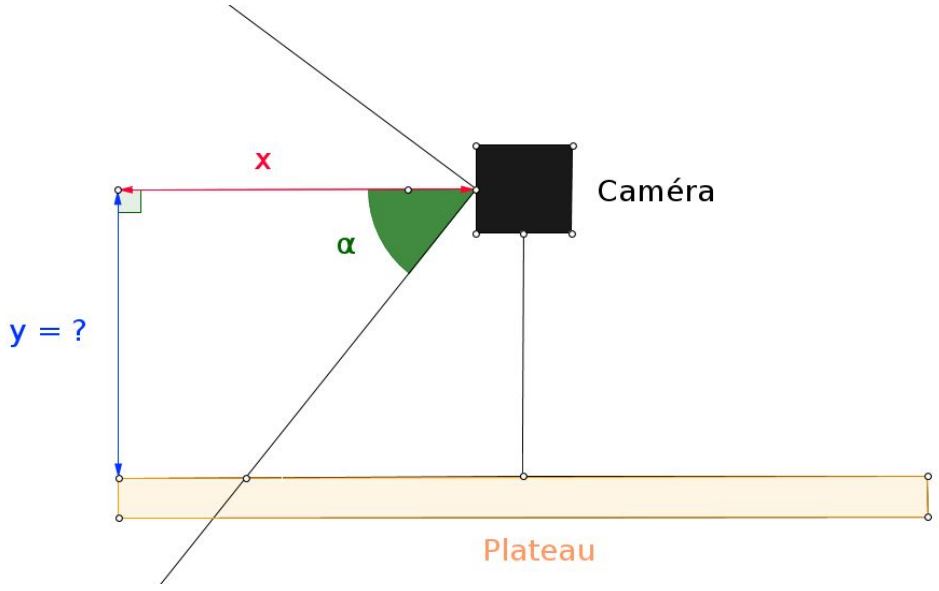


fig. 4: Schéma de la vision de la caméra vue de côté

La longueur x dépend des autres objets présents sur la plateau. Une fois que l'on connaît x, on détermine y de cet manière :

On calcul d'abord l'angle α grâce à l'aide des données de la première étape, où a' = 62 cm et b = 87 cm.

On a :

$$\tan \alpha = \frac{a'}{2b}$$
$$\alpha = Arctan(\frac{a'}{2b})$$
$$\alpha = Arctan(\frac{62}{2\times 87})$$
$$\alpha = 19,6°$$

$$b = 65\ cm$$

D'où $\tan \alpha = \frac{y}{x}$

$y = \frac{\tan \alpha}{x}$ et $y < 30\ cm$ (limite à ne pas dépasser dans le cahier des charges)

Pour la fixation, nous avons repris un modèle utilisé pour une autre caméra. Il n'y a pas de problème de tremblement car le robot à l'arrêt quand il prend une image et même si la caméra se décale par rapport au placement initial, nous avons pris une marge d'erreur assez grande.

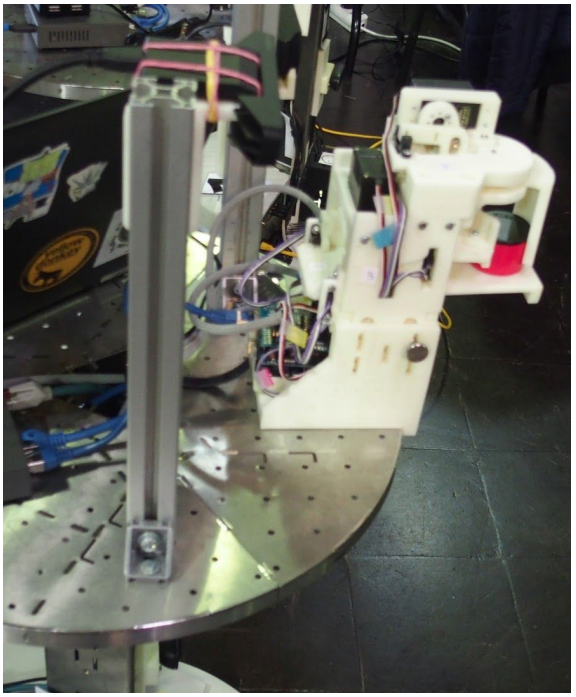


fig. 5: Fixation de la caméra

I.2.b. Choix logiciel - langage de programmation

Nous utiliserons l'ensemble d'outils informatiques **Robot Operating System** (ROS) pour récupérer les informations transmises par le moyen de vision mais également pour envoyer l'information au Robotino. Version employée: Kinetic.
Les langages privilégiés sont le **c++** et le **python**.

Pour la reconnaissance d'image, nous avons pensé à plusieurs solutions. Tout d'abord, comme nous n'avions que les plans 3Ds des MPS, nous voulions faire de la reconnaissance de contour. Mais l'inconvénient était que sur les MPS réelles, des câbles seraient présents et donc changeraient le contour des MPS.

On a aussi pensé à deux solutions cloud : l'API Cloud Vision de google et la solution Watson proposé par IBM mais alors, on aurait eu besoin d'une connection internet, ce qui n'est pas envisageable.

On s'est alors penché sur une solution utilisant la librairie **TensorFlow**, qui permet de faire du machine learning et du deep learning. De plus, cette librairie peut être utilisée dans des applications mobiles, donc sur la carte cela devait fonctionner. Or les instructions AVX ne sont pas supportées par la carte. Donc nos tuteurs nous ont suggéré de finir notre programme, en faisant tout fonctionner sur un ordinateur dans un premier temps.

Pour assurer la compatibilité entre les différents choix techniques on a travaillé avec Linux Ubuntu 16.04.

II. Solution apportée

La solution finale de notre projet est constituée principalement d'un fichier c++, un script python et un script bash. D'autres fichiers sont cependant nécessaires en amont.

II.1. Déroulement général

Notre solution se présente de la façon suivante:

1. Récupérer l'image prise par la caméra
2. Attendre que le Robotino soit correctement placé face à la MPS
3. Filtrer l'image reçue de façon à optimiser notre résultat
4. Enregistrer l'image résultante
5. Réaliser différents filtres sur cette image (sobel, canny, laplacien)
6. Donner cette image au programme de reconnaissance
7. Transmettre le résultat au Robotino

Pour cela il faudra au préalable:

1. Prendre différents set d'images
2. Entraîner des modèles
3. Établir quel set d'images utiliser (quels filtres) pour obtenir les résultats les plus probants

II.2. Récupération de l'image et communication

Tout d'abord on exécute le fichier de lancement sr300_nodelet_rgbd.launch du package realsense_camera proposé par ROS, celui-ci permet de récupérer les informations vues par les différents capteurs de la caméra.

Ensuite on exécute le fichier de lancement camTraitement_launch_file de notre paquet, celui-ci lance l'exécutable camTraitement de camTraitement.cpp et le script Python callTensorflow.py. Il fonctionne de la manière suivante:

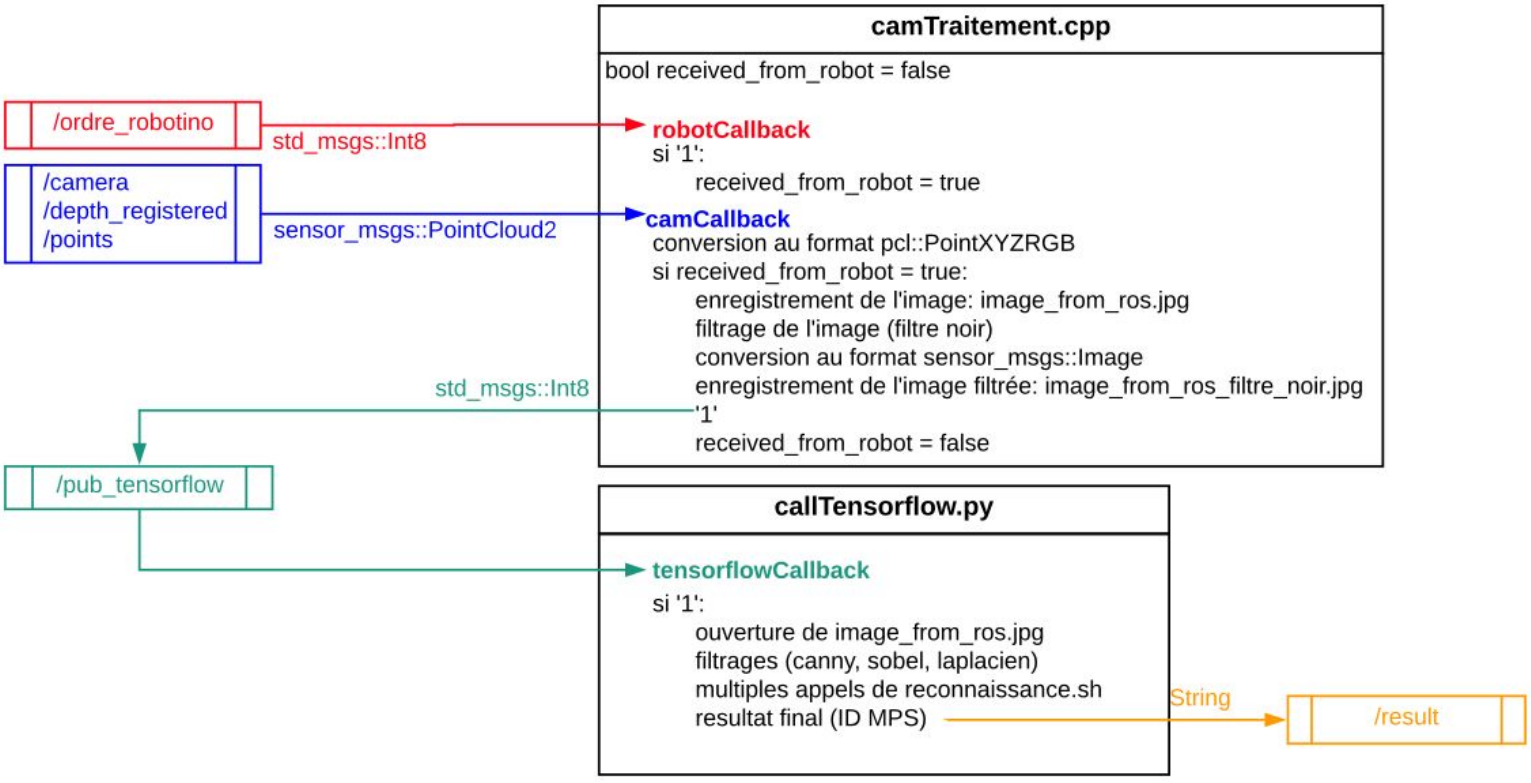


fig. 6: Schéma de fonctionnement de camTraitement_launch_file

Commençons par CamTraitement, celui-ci écoute deux topics:

- /ordre_robotino qui transmet des informations std_msgs::Int8. On considère que lorsque le robot est correctement placé face à la MPS il envoie 1 sur ce topic.
- /camera/depth_registered/points. Celui-ci permet de recevoir l'image vue par la caméra sous la forme d'un sensor_msgs::PointCoud2, il s'agit d'un nuage de points 3D. La fonction de callback camCallback sera détaillée juste après.

camTraitement.cpp possède une variable globale received_from_robot (un booléen initialisé à faux). Lorsque l'on reçoit 1 sur /ordre_robotino received_from_robot devient alors vrai par l'appel de la fonction robotCallback.

La fonction camCallback récupère donc un PointCloud2 depuis la caméra. Pour interagir avec ce nuage de points on utilisera la librairie PCL proposée par ROS. Notre

objectif est de récupérer un nuage en ayant pour chaque point ses coordonnées euclidiennes et la couleur du pixel correspondant au format RGB. Cela correspond au format pcl::PointCloudXYZRGB, on effectue la conversion en passant par l'intermédiaire du format pcl::PCLPointCloud2.

Si received_from_robot est vrai, c'est-à-dire que le Robotino est correctement placé face à la MPS nous allons maintenant filtrer l'image et l'enregistrer. On réalise un filtre de profondeur: pour tous les points du nuage, si le point est hors de portée de la caméra infrarouge (> 150 cm), c'est à dire que sa valeur sur l'axe z vaut NULL, on le transforme en pixel noir. (cf. Annexe A: *Exemple d'une image filtrée en profondeur*)

Ensuite pour enregistrer l'image il faut la convertir au format sensor_msgs::Image, en passant respectivement par les formats PCLPointCloud2 et PCLImage. Sensor_msgs::Image est un format propre à ROS, pour enregistrer l'image avec OpenCV on utilisera la librairie CVBridge qui fait l'interface entre les deux.
Une fois l'image enregistrée, la valeur 1 est envoyée sur le topic /pub_tensorflow au format Int8 et la variable globale received_from_robot est remise à faux.

En parallèle de ce fichier c++ on exécute le script callTensorflow.py, celui-ci écoute le topic /pub_tensorflow, si 1 est vu sur ce topic on ouvre alors l'image image_from_ros.jpg enregistrée juste avant, on lui applique différents filtres en utilisant OpenCv. Pour améliorer la détection des contours on a choisi les filtres:

- Sobel: son principe consiste à calculer le gradient de l'intensité de chaque pixel (c'est-à-dire la direction de la plus forte variation du clair au sombre et le taux de variation dans cette direction). Il est supposé que ces points de changement soudain de luminosité correspondent probablement à des bords. On applique un filtre de Sobel en dérivant selon x et un autre en dérivant selon y.
- Canny: il prend en arguments un seuil bas et un seuil haut. Comme pour sobel on considère l'intensité du gradient et sa direction mais cette fois on s'attarde plus sur le fait d'éliminer les faux contours. Pour cela il est admis que si un pixel pointe vers 2 valeurs supérieures au seuil maximum il faut l'éliminer car ce n'est pas un maximum local. Ensuite il effectue un seuillage par hystérésis. On commence par sélectionner les points qui dépassent le seuil haut et on applique ensuite le seuil bas en ne conservant que les composantes connexes qui contiennent un point au-dessus du seuil haut. On a choisi d'appliquer 2 filtres de canny aux arguments différents: 100/200 et 400/500.

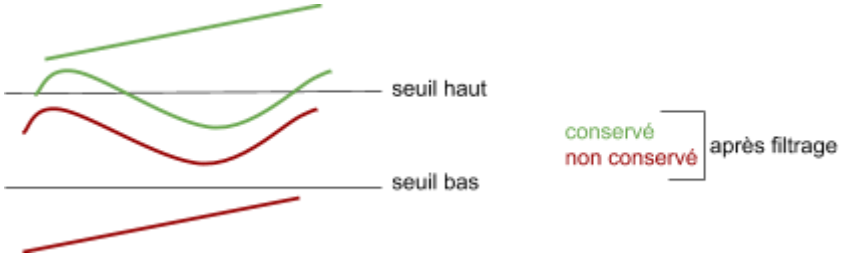


fig. 7: Schéma explicatif du filtre Canny

- Laplacien: il s'agit d'un filtre passe-haut dédié également à la détection de contours en mettant en valeur les détails qui ont un changement important de luminosité.

(cf Annexe B: *Visualisation des différents filtres appliqués sur une même image*)

On appelle ensuite plusieurs fois le fichier bash reconnaisance.sh expliqué dans la partie suivante, on publie le résultat final obtenu sur le topic /result au format String, c'est ce topic qui est utilisé pour communiquer avec le programme principal réalisé par les membres de l'ARPL. Ce résultat n'est autre qu'un identifiant correspondant au type de la MPS située face au Robotino.

II.3. Reconnaissance d'image

Pour la reconnaissance d'image nous utilisons la librairie TensorFlow avec un modèle pré-entraîné. C'est-à-dire qu'il a déjà été entraîné avec un problème similaire. En effet, ainsi on gagne du temps car le deep learning, utilisant des réseaux de neurones, peut prendre des jours à se modéliser mais ici, nous entraînons seulement la dernière couche de neurones et nous pouvons avoir un nouveau modèle en quelques minutes.

Le modèle utilisé initialement a été entraîné avec plus de 1000 classes différentes, allant d'une race de chiens au lave-vaisselle. Dans notre cas, nous allons le réutiliser en l'adaptant à notre problématique : reconnaître les 5 types de MPS. La partie la plus importante se fait en amont, par une analyse des modèles que l'on a créés. Mais nous verrons cela en détail dans la partie III. Pour le moment, on a donc plusieurs modèles, obtenus avec différents filtres. Par exemple, on a obtenu un modèle "canny1" en l'entraînant avec des images de MPS filtrées avec le filtre canny 100/200.

Pour la reconnaissance d'image, comme on le verra dans la partie III, on a utilisé des couples {image_filtrée_ou_non, modèle} pour obtenir des résultats significatifs. Ces couples seront appelés tests. Par exemple un test peut être la tentative de reconnaissance avec l'image obtenue filtrée en sobel x par le modèle entraîné avec les images canny 1.
Pour chaque type de MPS on a selon les cas :

- 16 tests pour reconnaître si c'est une base
- 15 tests pour reconnaître si c'est une cap
- 7 tests pour reconnaître si c'est une delivery
- 4 tests pour reconnaître si c'est une ring
- 9 tests pour reconnaître si c'est une storage

Tous ces tests sont réalisés grâce au programme reconnaissance.sh qui prend en paramètre: le nom de l'image à reconnaître et son modèle et qui retourne le type de la MPS reconnue avec son pourcentage de certitude.

Les probabilités pour chaque test sont toujours très hautes (toujours supérieures à 80% en moyenne). Donc on ne s'est pas fié aux probabilités mais plutôt au nombre de fois où un type de machine est reconnu. On fait ensuite un rapport entre le nombre de fois où le type de machine est reconnu et le nombre de tests réalisés. On renvoie alors le type de machine qui a le rapport le plus haut.

Par exemple, voici les valeurs obtenues avec un test sur une MPS de type base :

Type de MPS	Base	Cap	Delivery	Ring	Storage
Nombre de tests effectués	16	14	7	4	9
Nombre de tests validés	12	5	2	1	4
Fiabilité	*0,75*	0,36	0,29	0,25	0,44

fig.8 : résultat obtenu avec une image de type base, on renvoie ici le nom du type de la MPS, qui a sa fiabilité la plus élevée.

S’il y a une égalité alors on prend le résultat donné par le meilleur modèle global. Ce modèle est celui nommé “bad_original” qui donne un résultat correct 60% du temps. Il correspond au set d’images obtenu en prenant les photos des MPS à distance et hauteur aléatoires, sans filtrage.

III. Programmes annexes nécessaires

III.1. Réalisation des sets d'images

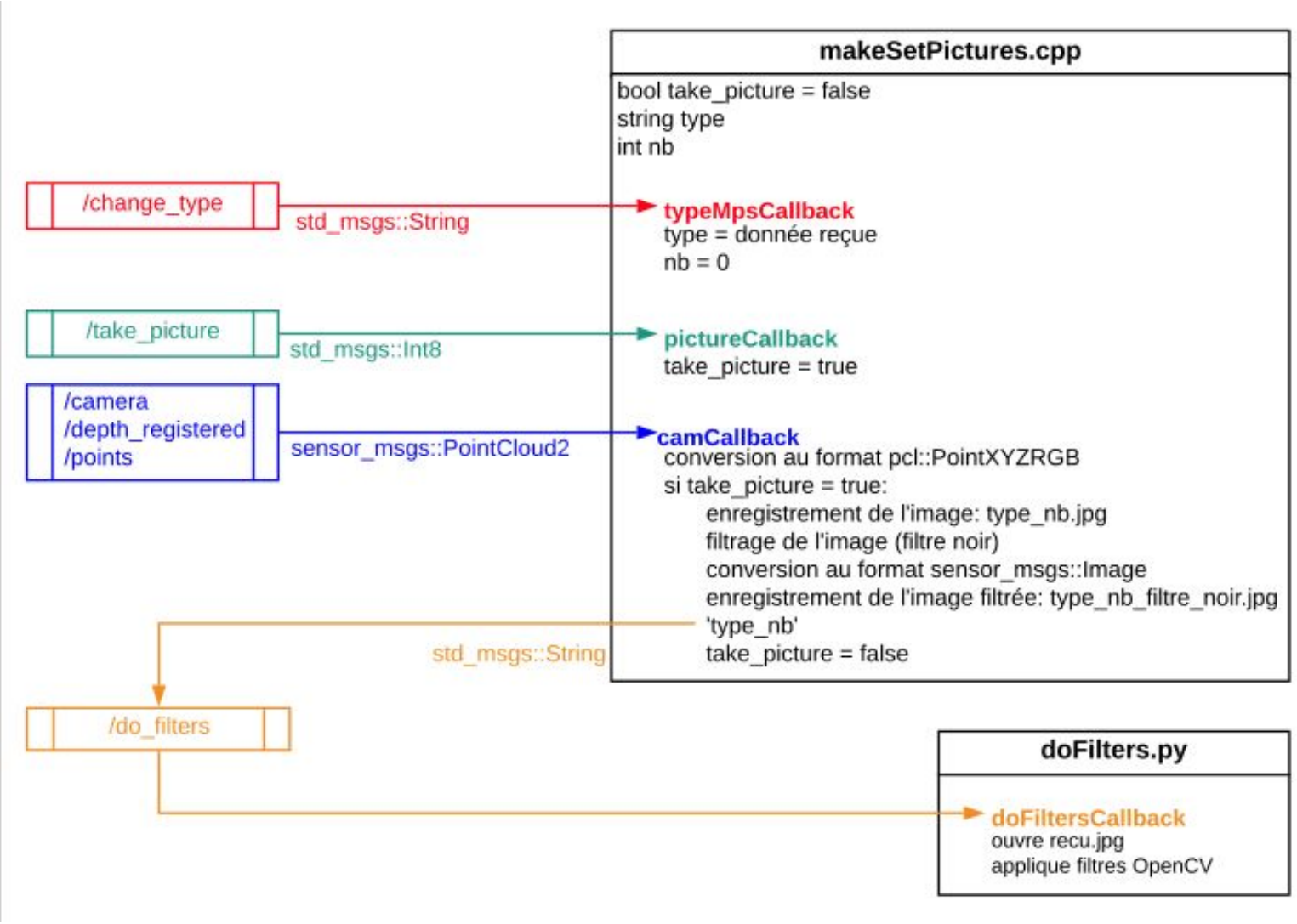


fig. 9: Schéma de fonctionnement de makeSetPictures_launch_file

Un second fichier de lancement est présent dans notre paquet: makeSetPictures_launch_file. Il permet de réaliser un set d'images avec tous les filtres appliqués automatiquement: que ce soit le filtre de profondeur (pixels noirs) ou les filtres réalisés sous OpenCV. Cela est fait rapidement par le biais de l'exécutable makeSetPictures du fichier c++ du même nom et du script doFilters.py.

Tout d'abord l'utilisateur envoie sur le topic /change_type le nom de la MPS qu'il va prendre en photo, ensuite à chaque fois qu'il souhaite prendre une photo de cette MPS il suffit d'envoyer 1 sur le topic /take_picture. Pour cela on utilise les commandes suivantes:

rostopic pub /change_type std_msgs/String ...

rostopic pub /ordre_picture std_msgs/Int8 1

L'image est alors enregistrée sans et avec le filtre de profondeur, le script doFIlters.py est appelé par la publication sur le topic /do_filters et réalise les filtres de canny, sobel et laplacien.

Pour passer à la photo suivante on publie sur /take_picture et lorsque l'on veut prendre en photo une autre MPS on publie sur /change_type. Par exemple pour la 10ème photo prise de la cap station on obtiendra les fichiers jpg suivants:

cap_10.jpg *cap_10_canny1.jpg* *cap_10_canny2.jpg* *cap_10_sobelx.jpg*
cap_10_sobely.jpg *cap_10_laplacien.jpg* *cap_10_filtre_noir.jpg*

notation nécessaire à connaître pour répartir correctement les fichiers lors de l'entraînement de des modèles

Le numéro de la photo s'incrémente à chaque envoi de 1 sur /take_picture et est remis à 0 à chaque nouvelle chaîne de caractère envoyée sur /change_type.

III.2. En amont de la reconnaissance d'image

L'utilisation du programme est très facile, ce qui nous a permis de nous pencher sur son optimisation. C'est la partie la plus importante car tout dépend de l'entraînement, et donc du modèle utilisé (un entraînement créant un modèle). En effet, un mauvais entraînement engendre de mauvais résultats. Par exemple si à la place de mettre des images de MPS de type base on met des photos de cap les résultats seront faussés.

Pour cela, on a utilisé un maximum d'options lors de l'entraînement (rotation de l'image, ajout ou diminution de luminosité…). Certes cela augmente le temps d'entraînement mais les résultats sont, le plus souvent, meilleurs en terme de précision.

On a d'abord commencé par créer un modèle avec les images originales, non filtrées. Mais on s'est rendu compte que la précision était à revoir (cf. tableau ci-dessous). En effet, en moyenne, la reconnaissance ne donnait que 62% de bonnes réponses. C'est comme cela que nous avons commencé à utiliser les filtres.

Type d'image testée	**nombre de résultats corrects sur 25 images testées**
base sans filtre	18
ring sans filtre	8
cap sans filtre	13
delivery sans filtre	11
storage sans filtre	12
moyenne de résultat bon (en %)	**62**

fig. 10: résultats obtenus avec le modèle sans filtres

On a fait 13 modèles au total, à chaque fois entraîné avec les images filtrées correspondantes au nom du modèle.

En parallèle, on a pris d'autres images du point de vue du robotino et on les a filtrées. On a pris 25 images de chaque type de MPS - donc on obtenu 25 images de base sans filtre, 25 images de base filtrées avec canny 100, [...], 25 images de cap sans filtre …

On a, par la suite, testé la reconnaissance d'image pour chaque couple {image_MPS_filtre,modèle}, 25 fois, avec les images prises précédemment. On a créé le script comparaison.sh pour tester les images et les filtres. On a tout reporté dans un tableur et on a ainsi pu déterminer les meilleurs couples {filtre, modèle}.
(cf. annexe C: *Résultats des différents modèles obtenus avec en couleur les couples retenus*).

Au vu des résultats, on s'est d'abord demandé si les modèles n'étaient pas spécialisés: s'ils ne renvoyaient pas toujours la même réponse, ce qui s'appelle du surapprentissage. Or, durant l'entraînement, il y a des tests pour éviter cela, et les résultats étaient toujours cohérents.

On a changé le script pour s'en rendre compte par nous-mêmes. On a alors remarqué que non, certains modèles sont seulement plus fiables que d'autres pour reconnaître certains types de MPS. On a gardé les couples qui donnaient de très bons résultats en conservant uniquement ceux qui ont reconnu au moins 17 images sur 25, c'est-à-dire précis à 68%. Cela nous a permis d'avoir assez de modèles: au moins 4 tests fiables pour chacun des types à reconnaître.

IV. Difficultées rencontrées

Nous avons rencontré quelques difficultés pour faire communiquer nos programmes entre eux. Au départ nous arrivions à exécuter le script réalisant l'appel à Tensorflow indépendamment, mais depuis ROS la librairie de Tensorflow était introuvable notamment à cause du multi-versions Python.

Nous souhaitions réaliser un filtre Passthrough proposé par la librairie PCL de ROS afin de tenter d'améliorer les résultats mais malgré un appel, des paramétrages correspondant aux tutoriels ROS et une compilation sans erreur celui-ci n'effectue aucun changement sur l'image. C'est pour cela que nous avons réalisé le filtre de profondeur avec les pixels noirs.

Ce dernier nous a également posé quelques soucis, par exemple si nous voulions mettre en noir tous les pixels situés à plus de 30 cm de la caméra nous obtenions une image avec en noir uniquement les pixels situés entre 30 et 150 cm (150 cm correspondant au champ de vision de la caméra en infrarouge). En effet la comparaison ne prenait pas en compte le fait qu'un point du nuage dont la distance est hors champ de vision infrarouge aura comme valeur NULL, c'est pour cela que nous effectuons le filtre de cette façon:

```
105     for (size_t i=0; i <pcl_trait_XYZRGB->points.size (); ++i)      {
106                     if (pcl_trait_XYZRGB->points[i].z!=pcl_trait_XYZRGB->points[i].z)    {
107                             pcl_trait_XYZRGB->points[i].rgb = rgb;
108                     }
109     }
```

Extrait de camTraitement.cpp

V. Améliorations possibles

V.1. Améliorations possibles

Des contraintes de temps ne nous ont pas permis de faire un set d'images filtrées en profondeur aussi important que nous l'avons espéré, il serait avantageux pour l'entraînement du modèle d'enrichir celui-ci lors d'une prochaine compétition Logistic League.

Pour l'instant lors de la reconnaissance des MPS le programme ne garde pas en mémoire celles déjà reconnues depuis le début de la phase d'exploration. Etant donné que nous connaissons le nombre de MPS à reconnaître pour chaque type par match, stocker les informations sur celles déjà reconnues pourrait permettre d'améliorer les résultats ou au moins éviter de perdre des points par exemple en évitant d'annoncer une deuxième storage station à la Referee Box. On pourrait stocker ces informations sous la forme d'un tableau:

ID MPS	C	D	R	S	B
Nombre restant de MPS à découvrir	1	0	2	1	1

Pour l'instant pour reconnaître une MPS le robotino doit l'approcher au centre et à cm, être capable de reconnaître les MPS de plus loin permettrait d'utiliser une caméra non 3D et de gagner du temps sur l'exploration. Cela permettrait également de minimiser le nombre de déplacements du robot et donc de perdre moins de précision au niveau de l'odométrie. Il faudrait alors être capable de déterminer la distance entre le robot et la MPS (c'est envisageable avec le laser) et de savoir quel côté de la MPS est observé par la caméra pour déterminer son orientation. Lors de la compétition 3 minutes maximum sont allouées à la phase d'exploration.

Enfin, il faudrait mettre le code, sur la carte Intel, pour pouvoir utiliser la solution en compétition réelle.

V.2. Ce qui pourrait être développé pour la logistic league en lien avec le projet

Il pourrait être utile d'utiliser Tensorflow pour déterminer les différents obstacles rencontrés par le robot (humain, robot) pour ajuster le moyen d'évitement. Cela devrait être assez rapide à réaliser et pourrait utiliser la caméra déjà présente sur le mât du robot.

VI. Conclusion

Notre projet consistait à reconnaître différents types de machines (MPS) grâce à de la reconnaissance d'image et sans utiliser d'AR tag. De plus, cette solution devait s'intégrer dans le cadre d'une compétition de robotique.

A travers ce rapport, nous avons montré par quelles étapes nous sommes passés pour répondre à l'objectif fixé, en passant par le choix matériel et logiciel. Nous avons donc réussi, d'un point de vue logiciel car nous arrivons bien à prendre une image suite à la réception d'un ordre sur ROS. Puis nous la traitons avec différents filtres pour ensuite reconnaître la MPS avec une meilleure précision. Les résultats ne sont pas encore complètement fiables, mais nous avons déjà des pistes d'améliorations.

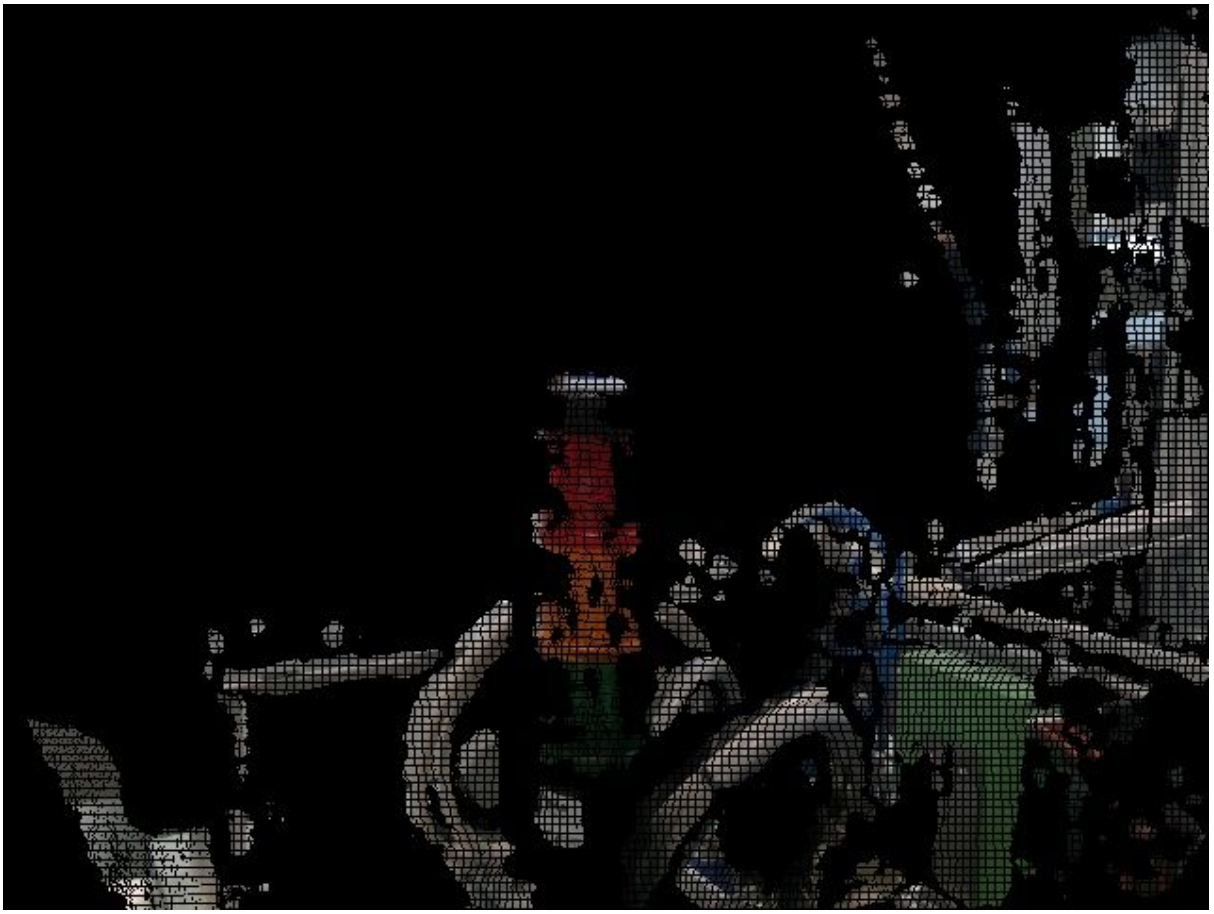
Néanmoins, d'un point de vue matériel, l'étude devait être réalisée directement sur une carte Intel, cela n'a pas été possible à cause des instructions AVX utilisées par la librairie TensorFlow. On s'est alors focalisé sur une solution utilisable sur un PC. Avec une meilleure gestion du temps, on aurait pu travailler sur la compilation lors de l'installation de cette librairie qui est open source.

Pour résumer, le projet a été réalisé avec un certain succès car en plus de l'objectif atteint, informatiquement, nous avons dû utiliser plusieurs outils que nous ne maîtrisons pas comme ROS, le C++, python, et des librairies comme OpenCV et TensorFlow.

Enfin, ce projet nous a permis de découvrir ce qu'était une compétition de robotique, un vrai hackathon de robotique, et pour cela nous remercions encore toute la PyroTeam, qui a réussi à susciter l'intérêt pour la Robocup, et nous convaincre d'être dans une filière adaptée.

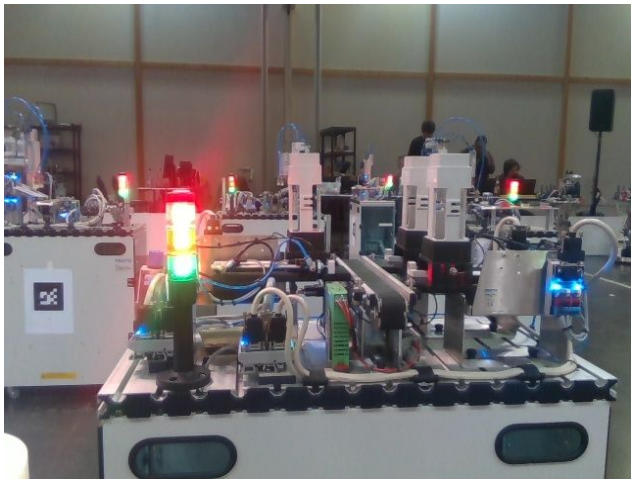
Annexes

Annexe A

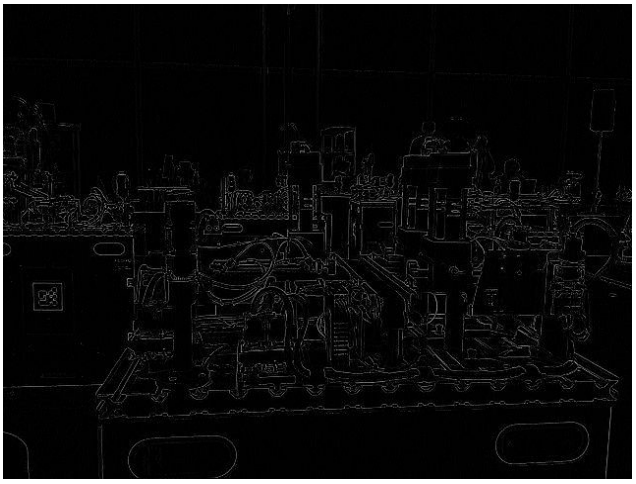


Exemple d'une image filtrée en profondeur (cap station)

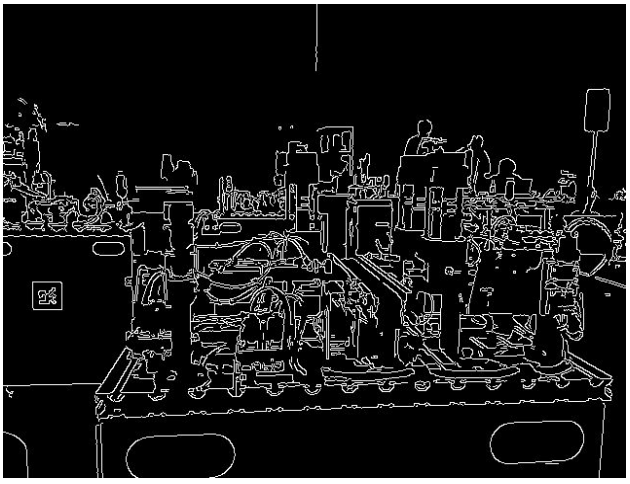
Annexe B



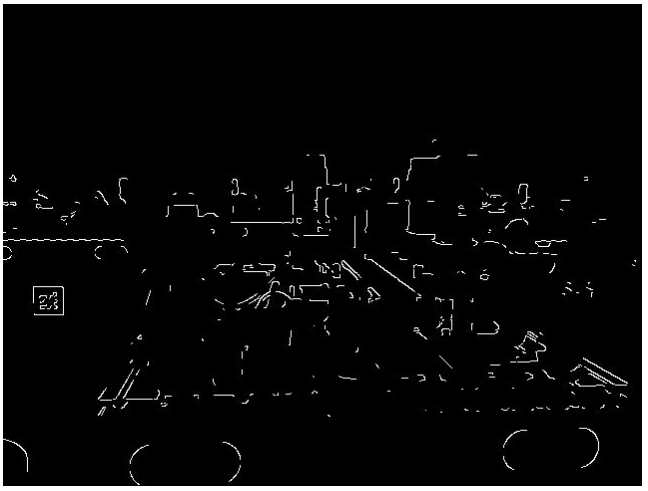
Original



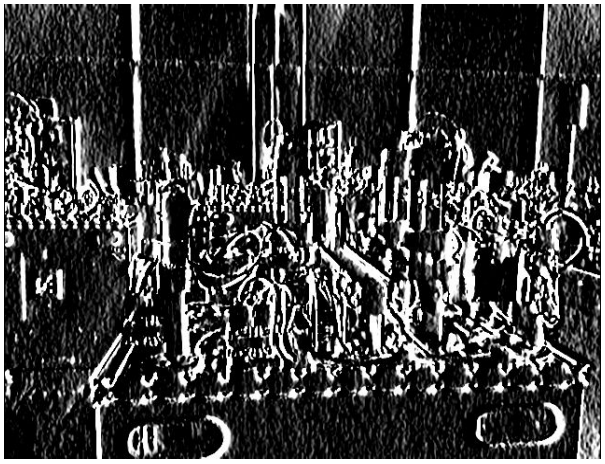
Laplacien



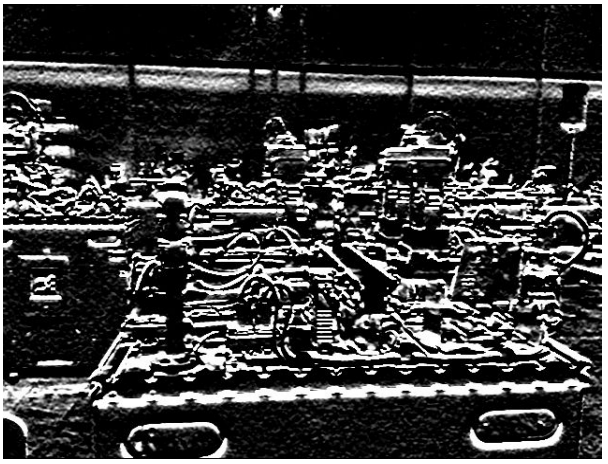
Canny 100/200



Canny 400/500



Sobel en x



Sobel en y

Visualisation des différents filtres appliqués sur une même image (base station)

NARBAIS-JAUREGUY Damien
BRIOIS Zoé

Annexe C

Modèle testé														
position caméra (*)		aléatoire											film	robot
filtre		canny 1		canny 2		laplacien	∅	∅	sobel x	sobel y	tous	tous	∅	pixels
nombre itérations		500	8000	500	8000	500	500	8000	500	500	500	8000	500	500
Image testée		nombre de résultats corrects sur 25 images testées												
MPS	filtre													
BASE	canny 1	10	14	1	6	14	19	18	20	18	1	17	0	1
	canny 2	6	5	3	3	19	25	25	25	16	1	10	2	0
	laplacien	1	1	4	8	13	5	5	6	1	0	1	0	0
	original	6	6	1	1	19	19	18	17	20	1	18	0	25
	sobel y	3	4	2	4	0	8	8	9	12	6	12	0	0
CAP	canny 1	14	17	5	4	5	0	0	0	1	11	4	20	0
	canny 2	0	0	5	4	0	0	0	0	0	5	0	19	1
	laplacien	24	20	4	4	10	0	0	0	21	13	19	25	0
	original	13	7	11	11	0	12	13	12	0	13	0	25	0
	sobel y	25	25	23	8	19	0	0	0	13	21	12	19	0
DELIVERY	canny 1	1	2	22	21	17	4	3	4	15	6	17	5	0
	canny 2	11	10	12	15	15	0	0	0	22	8	23	3	12
	laplacien	1	3	13	12	10	14	11	15	11	8	11	0	0
	original	0	1	5	9	8	9	11	10	1	13	1	3	0
	sobel y	1	1	3	7	5	17	14	17	10	6	9	11	5
RING	canny 1	6	8	6	10	0	12	14	10	6	11	5	0	0
	canny 2	4	4	12	14	0	0	0	0	2	19	2	0	0
	laplacien	0	0	5	5	1	18	20	17	7	4	5	0	0
	original	2	2	2	1	1	10	8	10	9	10	0	1	0
	sobel y	0	0	0	0	0	0	0	0	0	0	0	0	0
STORAGE	canny 1	7	11	0	0	4	0	0	0	0	5	0	0	25
	canny 2	9	12	0	0	2	0	0	0	0	3	0	6	18
	laplacien	11	17	4	10	18	13	1	7	0	18	0	0	25
	original	21	20	4	4	13	10	12	10	0	11	0	0	11
	sobel y	0	2	0	0	10	0	0	0	0	13	0	8	25

Résultats des différents modèles obtenus avec en couleurs les couples retenus

(*) aléatoire correspond aux photos prises des MPS à distance et hauteur variables
film correspond aux photos extraites d'une vidéo obtenue en faisant le tour des MPS
robot correspond aux photos avec la caméra correctement placée sur le robot