HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

RAPPORT DE TRAVAIL DE BACHELOR

Démonstrateur pour un robot d'arrosage

Robotique autonome et agriculture durable

Diplomant :
Marie Pascale Nzinke LEMDJO

Professeur :
Etienne MESSERLI

Mandant :
Christian BOVIGNY

Expert :
Philippe AMEZ-DROZ

HEIG-VD
Département TIC - Informatique Embarquée
Institut REDS
(Reconfigurable & Embedded Digital Systems)

25 juillet 2019

Table des matières

Remerciements

Je profite de l'occasion qui m'est offerte pour dire un grand merci à toutes les personnes qui de près ou de loin ont contribué à ma formation et à la rédaction de ce rapport.

"Toute grâce excellente et tout don parfait descendent d'en haut, du Père des lumières, chez lequel il n'y a ni changement ni ombre de variation". Je souhaite avant tout à exprimer toute ma reconnaissance à **Dieu tout puissant** de qui j'ai reçu tout le nécessaire afin de suivre cette formation jusqu'à son terme.

Je veux dire toute ma gratitude à mon encadreur, **Mr Etienne MESSERLI**, pour le temps qu'il a consacré à m'apporter les outils méthodologiques indispensables à la réalisation ce travail. Je le remercie de m'avoir orienté, aidé et conseillé.

Je suis reconnaissante à tous les enseignants ainsi qu'aux assistants du département des Technologies de l'information et de la communication, en particulier ceux de la filière Informatique Embarquée.

J'adresse ma profonde gratitude à **Mr** et **Mme Pierre-Alain** et **Dorinda BRUGGER** pour m'avoir apporté leur aide et leurs conseils.

J'adresse mes sincères remerciements à mes très chers parents, **Mr** et **Mme Samuel** et **Odette LEMDJO** qui m'ont soutenu et encouragé malgré les milliers de kilomètres qui nous séparent. Je dis un grand merci à **Mr** et **Mme Guy-Roger** et **Nadine KAMSU** qui m'ont soutenu et encadré depuis mon arrivée en Suisse. Je remercie ma grande soeur **Gaëlle NDJANMOU** et mon tuteur **Théophile NDJANMOU** ainsi que ma petite soeur **Cassandra LEMDJO** pour leurs encouragements.

Enfin je remercie mon fiancé **Kevin GUEMTO**, qui a toujours été là pour moi. Son soutien inconditionnel et ses encouragements ont été d'une grande aide.

HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

Département **TIC**
Filière **Informatique**
Orientation **Informatique embarquée**
Candidate **Lemdjo Nzinke Marie Pascale**
Responsable **Messerli Etienne**

TRAVAIL DE BACHELOR 2018 - 2019

Démonstrateur pour un robot d'arrosage

Domaine de recherche : Robotique autonome, agriculture durable.

Entreprise Ferme bio du Moulin (Bovigny Christian)

Énoncé

Nous souhaitons réaliser un robot d'arrosage pour des plantons dans le but de soutenir une agriculture écologique et durable. Durant les premiers jours, voir les premières semaines, les plantons sont très sensibles à un manque d'eau par le faible volume et profondeur de leurs racines. Ce stress peut engendrer des pertes de production assez conséquente. En 2018, un premier projet a permis d'identifier différentes problématiques à résoudre.

L'objectif de ce projet est de concevoir et réaliser un démonstrateur pour le robot d'arrosage. Un précédent travail de Bachelor a permis de réaliser un traitement d'image permettant d'identifier des palntons de salades. Durant ce projet, il s'agira de réaliser un robot à échelle réduite démontrant la fonctionnalité d'arrosage.cette démonstration doit permettre de positionnner le robot au début d'une bande de plantons et de simuler l'arrosage des palntons pendant l'avance de celui-ci.

Les principales étapes du projets sont:

- Disposer d'un robot à échelle réduite permettant des mouvements et des vitesse de déplacement nécessaire pour l'arrosage. Il devra disposer d'une tête d'arrosage. Une solution sera proposée pour simuler l'arrosage.
- Développer et réaliser une commande manuel du robot.
- Concevoir et réaliser la commande de la tête d'arrosage. Il doit être prévu de pouvoir paramétrer l'arrosage.
- Développer et réaliser la commande du robot permettant de démontrer la phase d'arrosage du robot.

Mots clés : robotique, positionnement, vision, système embarqué, prog C

Cahier des charges

Disposer d'un robot à échelle réduite permettant des mouvements et des vitesses de déplacement nécessaires pour l'arrosage. Le robot se déplacera sur une surface simulant un champ avec des plantons. Il dispose d'un bras mobile pour simuler l'arrosage.

Les principales étapes du projets sont:

- Développer et réaliser une commande manuel du robot utilisant une console sans fils.
- Concevoir et réaliser la commande du bras mobile de la tête d'arrosage. Il doit être prévu de pouvoir paramétrer l'arrosage.
- Développer et réaliser la commande du robot permettant de démontrer la phase d'arrosage des plantons pendant l'avance du robot. Une interface interactive, utilisant l'écran tactile, sera développée pour la configuration des différents paramètres du robots.
- Rédiger une documentation sur le développement de la commande réalisée ainsi qu'un manuel d'utilisation du démonstrateur du robot d'arrosage.

Bibliographie

Liens: https://bovigny.ch/ferme-bio-du-moulin/

Candidate

Lemdjo Nzinke Marie Pascale Date: _____ Signature: _____

Responsable

Messerli Etienne Date: _____ Signature: _____

Responsable de la filière Informatique

Donini Pier Date: _____ Signature: _____

Ferme bio du Moulin

Bovigny Christian Date: _____ Signature: _____

Résumé

Démonstrateur pour un robot d'arrosage

Robotique autonome et agriculture durable

Marie Pascale Nzinke LEMDJO

Le but de ce rapport est de présenter le travail effectué pour l'implémentation d'un démonstrateur pour un robot d'arrosage en vue de soutenir l'agriculture durable. Pendant leurs premières semaines après leur plantation, les plantons sont sensibles à un manque d'eau à cause de leur faible volume et profondeur des racines dans le sol. Il est donc nécessaire de les arroser régulièrement avec une quantité d'eau bien précise afin d'éviter une perte de productivité et un gaspillage d'eau. C'est dans cet optique que le mandant qui est un propriétaire d'une ferme biologique a proposé de développer un robot d'arrosage pour ses plantons. En 2018, un travail de diplôme a été réalisé et a permis d'implémenter un prototype montrant les fonctionnalités de détection des plantons et de commande des moteurs du robot.
Le présent projet quant à lui permettra à un utilisateur de configurer la durée d'arrosage. Ensuite le robot démarre au début d'une ligne de plantons du champ. Il sera dès lors capable de suivre la ligne de plantons en les arrosant à l'aide de son bras d'arrosage, sans s'arrêter et de façon autonome. Il est à noter que dans ce projet, une Led est utilisée pour marquer la fonction d'arrosage. Mais dans le projet futur, la commande d'une vanne est prévue. Le travail consiste à développer la commande automatique du mouvement du robot, du bras d'arrosage et à contrôler le suivi d'une ligne de plantons. Ce rapport montre aussi les difficultés rencontrées et les compétences acquises durant ce travail.

MOTS-CLES :
Robotique ; Positionnement ; Vision ; Système embarqué ; Programmation C.

Chapitre 1

Introduction

1.1 Contexte du projet

Un robot autonome est un robot conçu pour accomplir certaines tâches sans conducteur, ou opérateur [**?**]. La plupart d'entre eux sont équipés de systèmes de localisation qui leur permettent de se répérer et de naviguer dans leurs champs de travail tous seuls [**?**] afin d'exercer leurs tâches. Cette technologie pourrait être utilsée dans le domaine de l'agriculture pour réaliser les tâches de récolte, de désherbage ou encore d'arrosage. Il existe en Suisse plusieurs projets de robotique agricole, à l'instar d'Ecorobotix, un robot autonome pour le déserbage [**?**]. Ces technologies présentent de nombreux avantages ; économie de coûts, de ressources et limitation d'impact environnemental des exploitations.

Le présent projet dont le mandant est un propriétaire d'une ferme biologique, désire un robot autonome pour l'arrosage de ses plantons. Rappelons que durant les premières semaines, les plantons sont très sensibles à un manque d'eau par le faible volume et profondeur de leurs racines. Ce stress peu engendrer des pertes de production assez conséquentes. Il est donc nécessaire de les arroser régulièrement. La gestion de la ressource eau devient alors critique.

1.2 Objectifs

Le but de ce travail est de réaliser un démonstrateur pour un robot d'arrosage. Un travail a été effectué en 2018 et a permis de réaliser les fonctions de détection des plantons par un traitement d'images et de commande des moteurs [**?**]. Ce démonstrateur quant à lui, utilise le premier travail pour démonter le fonctionnement du robot avec la gestion de l'arrosage. Cette commande permettra d'assurer l'arrosage planton par planton avec la quantité d'eau appropriée pendant l'avance du robot. L'avantage qu'apporte un tel robot c'est donc de permettre une utilisation efficiente de la ressource eau.

1.3 Structure du rapport

Avant que nous puissions entrer dans les détails d'informations pratiques, plusieurs fondements doivent être posés. A cette fin, la structure du rapport se présente comme suit :

Chapitre 2 - Etat initial du projet : Il s'agit ici de présenter le design, le matériel et la configuration de l'environnemt de travail et la mise en route du robot. Ce chapitre parle aussi des fonctionnalités implémentées du robot.

Chapitre 3 - Commande des moteurs : Dans ce chapitre, il s'agit de décrire les tests effectués pour le réglage des roues. Il présente aussi certaines corrections qui ont été nécessaires d'apporter au niveau logiciel pour un meilleur rendu de leur fonctionnement.

Chapitre 4 - Conception du système d'arrosage : Ce chapitre présente l'analyse faite, les différents choix conceptuels proposées et leurs justifications pour réaliser le système d'arrosage. Il décrit les différentes fonctionnallités de base retenues pour mener à bien cette tâche.

Chapitre 5 - Commande automatique de l'arrosage : Ce chapitre décrit l'algorithme utilisé pour commander automatiquement le bras d'arrosage et simuler l'arrosage des plantons.

Chapitre 6 - Configuration de la durée d'arrosage : Ce chapitre décrit les outils utilisés pour l'implémentation de l'interface utilisateur, ainsi que son intégration dans le projet entier. Il présente les tests effectués sur cette interface et les résultats obtenus.

Chapitre 7 - Tests et résultats : Comme son nom l'indique, ce chapitre décrit les différents tests effectués et leurs conditions et discute des résultats obtenus.

Chapitre 8 - Conclusion : Ce chapitre constitue la synthèse du projet. Il présente aussi les commentaires personnels, les difficultés rencontrées, et parle des améliorations possibles pour le projet futur.

Chapitre 2

Etat initial du projet

Au début du projet, il a fallu mettre en marche le robot. Pour ce faire, il était nécessaire de comprendre les spécificités du design et du logiciel du travail précédent. Pour arriver à cette fin, une lecture minutieuse du rapport de Ludovic Richard [**?**] a été d'une très grande utilité.

2.1 Design

Le corps du robot est un ancien robot qui a été conçu par le REDS et utilisé lors des portes-ouvertes de l'école. Ce robot a été adapté en conservant le châssis, les moteurs et la carte d'alimentation. L'ensemble du système embarqué a été refait pour servir de prototype du robot d'arrosage lors du projet de 2018. Le fonctionnement de ces différents composants électroniques seront décris dans la suite du rapport. Ce robot baptisé Wateringbot, servira de démonstrateur pour le projet actuel.

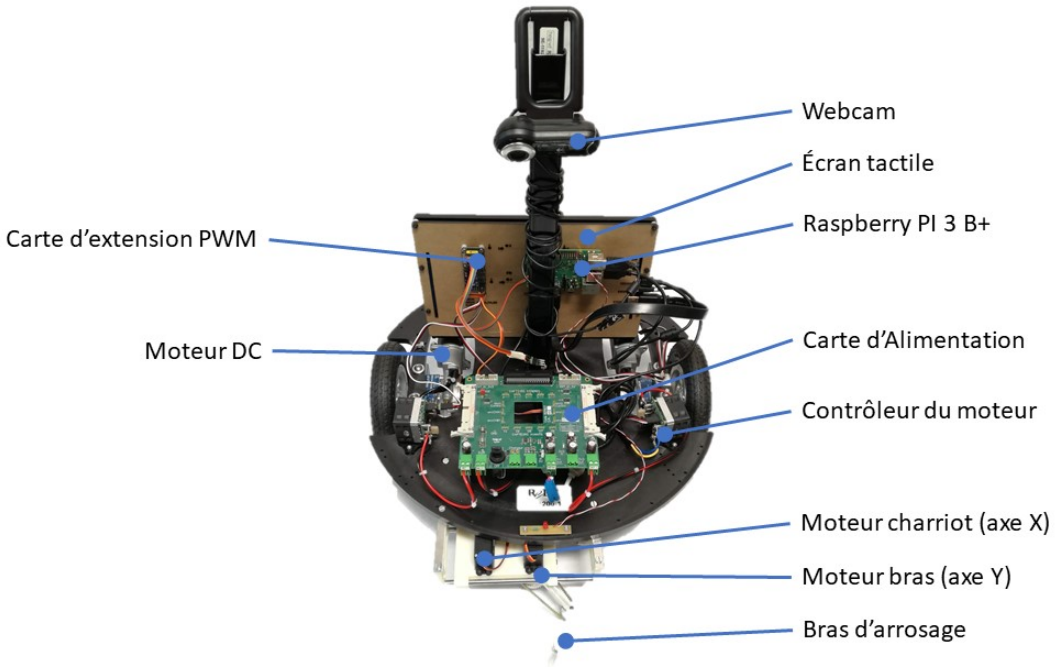

FIGURE 2.1 – Schéma descriptif du Wateringbot

Le schéma ci-dessous montre les élements choisis pour cette partie système embarqué.

- Une Raspberry Pi 3 B+, carte à processeur pour servir de contrôleur pour tout le système.
- Une carte d'extension PWM de chez Adafruit pour la commande des moteurs.
- Une caméra de chez Logitech pour la détection des plantons
- Un écran tactile pour montrer les résultats des tests de la commande du robot

La Raspberry Pi envoie une consigne PWM (Pulse Width Modulation) à la carte d'extension via une ligne série I2C. La carte d'extension génère la valeur de PWM correspondant à cette consigne. Pour un moteur DC, cette valeur de PWM est traduite par le contrôleur de moteur en tension afin de définir la vitesse de la roue. Pour le moteur du bras, cette valeur PWM définit sa position.

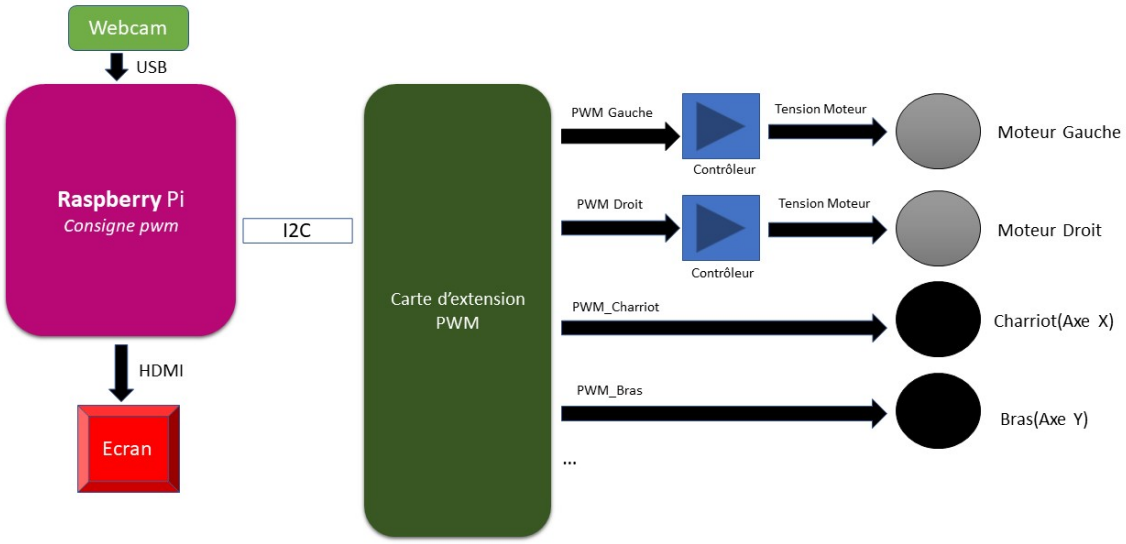

FIGURE 2.2 – Schéma de la plateforme embarquée

Raspberry PI 3B+[?] La carte Raspberry PI 3 modèle B+ est le modèle sorti en mars 2018. Elle a été choisie pour sa capacité à pouvoir effectuer du traitement d'image et pour son grand réseau de développeurs.
Cette carte embarquée a un microcontrôleur Broadcom BCM2837B0 comportant 4 CPU ARM Cortex-A53 tournant à 1,4GHz et possède 1 GB de SDRAM LPDDR2. Outre son microcontrôleur, elle dispose de :

- 4 ports USB
- 1 port micro USB 5v DC
- 40 pins GPIOs
- 1 port Ethernet
- 1 bus SD
- 1 port camera
- Une sortie Audio
- Une sortie HDMI

Elle dispose de tous les bus et les interfaces nécessaires(UART, SPI, I2C...). Il est possible de monter plusieurs cartes d'extensions pour ajouter des fonctionnalités supplémentaires (par exemple une carte d'alimentation pour moteurs).

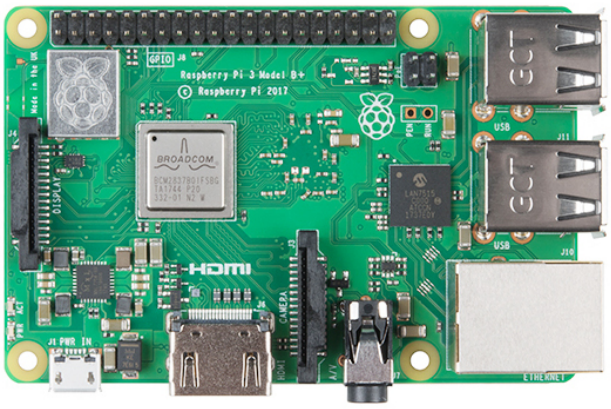

FIGURE 2.3 – Carte Raspberry PI 3 B+[**?**]

La carte d'extension de chez Adafruit est utilisée pour commander les moteurs. Cette carte d'extension a été ajoutée à la Raspberry Pi afin de disposer suffisamment de sorties PWMs pour commander les moteurs de ce projet. Elle est capable de générer 16 sorties PWMs et se commande par le bus I2C. Un PWM est une valeur de pulse permettant de définir la vitesse du moteur. Cette carte possède un compteur 12-bits, dont les valeurs de pulse sont comprise entre 0 et 4095. Lorsque cette pulse est grande, le moteur rallentit et à l'inverse(valeur de pulse petite), le moteur augmente sa vitesse. La fréquence PWM ne peut être paramétrée, elle est fixée à 200 Hz, ce qui est compatible avec les moteurs de modélisme du prototype. Sa fiche technique est donnée en annexe .

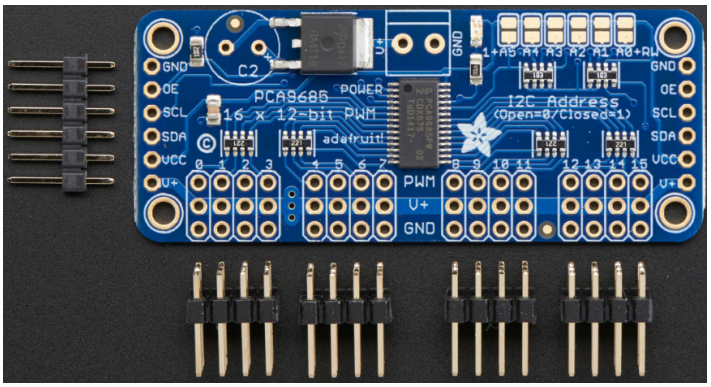

FIGURE 2.4 – Carte d'extension PWM de chez Adafruit[**?**]

La webcam de chez Logitech est utilisée pour la détection des plantons. Elle a été choisie sur la base du rendu de couleurs.

FIGURE 2.5 – Camera Logitech Quickcam Pro 9000 [**?**]

L'écran tactile de chez Waveshare [?] est utilisé pour montrer les résultats des tests et paramétrer l'arrosage. Il a été choisi pour des questions de flexibité. Il ne nécessite pas de configuration, il suffit de le connecter à la Raspberry Pi pour qu'il fonctionne.

FIGURE 2.6 – Ecran tactile Waveshare 10,1' [**?**]

HB-25 Motor Controller[?] Ce contrôleur de moteur pour les roues de chez Parallax Inc permet de régler la vitesse du moteur en faisant varier la tension à la sortie de celui-ci. Il se pilote avec deux fils d'alimentation et un fil de commande (le Ground, le power et le signal PWM).
Attention : si l'on branche dans le sens inverse les entrées du contrôleur de moteur, il ne se passera rien. Le fichier "dépannage" contient la description des files et du branchement. Il existe plusieurs modes d'opération pour ce contrôleur.

1. Le mode 1 ou single mode est celui dans lequel un seul HB-25 est relié à la ligne d'entrée-sortie du microcontroleur. Dans ce mode, une seule valeur de pulse envoyée suffit pour contrôler le HB-25. Il est aussi important de souligner qu'une valeur PWM

valide est comprise entre 0.8ms et 2.2 ms. En dehors de cet intervalle, le moteur est au repos. Le tableau suivant montre les spécifications des valeurs de PWM valides.

Full Reverse	Neutral(off)	Full Forward
1.0 ms	1.5 ms	2.0 ms

TABLE 2.1 – Valeurs de la pulse d'entrée

2. Le mode 2 ou dual mode est celui pour lequel un deuxième HB-25 est relié au premier HB-25 plutôt qu'au microcontroleur. Ceci permet de contrôler avec la même ligne d'entrée-sortie du microcontrôleur deux HB-25.

Dans le cadre de ce projet, le mode 1 est utilisé. En effet, deux lignes indépendantes du microcontrôleur ont été utilisées pour contrôler chaque HB-25 parce que c'est le mode qui convient pour diriger le robot correctement ; avancer, reculer et tourner. Sa fiche technique est donnée en annexe.

Les moteurs du bras[?] sont des servos moteurs. Un servo moteur est un type de moteur qui permet un positoinnement par rotation continue. Il se pilote aussi avec trois fils ; deux fils d'alimentation et un fil de commande. Il ne permet pas de parcourir un tour complet. Sa rotation maximale est de 120 degrés dont 60 degrés de chaque direction. Malheureusement sa documentation est assez pauvre pour en dire plus sur son fonctionnement. Nous procéderons plus par tests pour mieux comprendre comment il marche. Cependant, la même remarque est valable pour ce moteur comme pour les HB-25, en ce qui concerne les branchements filaires. Sa fiche technique est donnée en annexe.

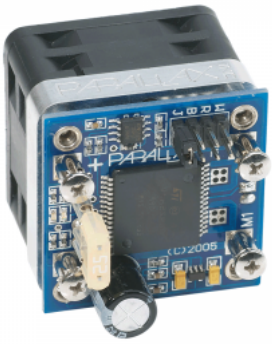

FIGURE 2.7 – Contrôleur de moteur HB-25[?]

FIGURE 2.8 – Moteur MG995[?]

2.2 Principe de fonctionnement

Une fois ces composants électroniques présentés, il faut faire une revue des fonctionnalités informatiques implémentées.

2.2.1 Fonctionnalités existantes

Détection des plantons : La détection des plantons consiste à découvrir leurs emplacements sur l'image de la caméra. La méthode utilisée pour y parvenir est la méthode de différentiation par les couleurs. Cette méthode se base sur le principe que tous les plantons sont de la même couleur. Donc tous les objets de l'image étant dans une plage vert clair - vert foncé seront détectés comme planton. Ensuite il faut appliquer des opérations d'analyse d'image suivantes :

1. La binairisation qui permet de transformer l'image initiale en image noir/blanc. Après la binairisation, les objects ayant une couleur de la plage vert clair - vert seront de couleur blanche et le reste de l'image en noir.
2. L'analyse des pixels pour déterminer la couleur du pixel ; noir ou blanc.
3. Les opérations morphologiques qui sont l'érosion et la dilatation. Elle permettent de reconstruire l'image initiale, et sont surtout très utiles pour éliminer les bruits. En effet, les bruits induisent des changements aléatoires dans l'image.

Suivie d'une ligne dans le champ : Cette fonctionnalité intègre la commande des moteurs et la reconnaissance de la direction à suivre.

1. La commande des moteurs du robot se fait par PWM. En effet, un driver PWM a été écrit. Il permet d'envoyer périodiquement une valeur de pulse d'entrée comprise entre 1 ms et 2 ms. Une pulse de 1ms fait tourner le moteur en arrière à sa vitesse maximale, une pulse 1,5 ms le fait stopper. Une pulse de 2ms le fait tourner en avant à sa vitesse maximale.
2. La reconnaissance de la direction à prendre. Elle est indispensable puisque le but étant que le robot se déplace tout seul. Le robot est mis au début d'une ligne de plantons. La trajectoire du robot est calculée grâce à la reconnaissance des plantons détectés. Le robot avance en corrigeant automatiquement sa trajectoire. Le principe de cet algorithme est de rechercher la première ligne de plantons horizontale devant le robot et de s'assurer que la caméra lui soit perpendiculaire. On s'intéresse ici, aux coordonnées en Y. Ces coordonnées sont triées et stockées dans un tableau lors de la reconnaissance. Les trois dernières coordonnées du tableau représentents celles des plantons de la ligne horizontale devant le robot. Pour tester qu'elles appartiennent à la même ligne, on vérifie qu'elles soient le plus proches les unes des autres(coordonnées Y assez similaires). La fin d'une ligne est détectée lorsque le robot détecte moins de trois plantons.

2.2.2 Structure logicielle

La structure logicielle permet de montrer les différentes parties du programme et leurs interactions. Ce programme comprends deux threads. Le premier thread est le thread principal. Il est chargé de parser les arguments d'entrée, d'initialiser le joystick, le driver I2C, le driver PWM ainsi que d'activer le deuxième thread. Le deuxième thread se charge du pilotage automatique du robot. Il initialise la capture vidéo et les options de détection. Il crée les fenêtres de visionnage des plantons détectés, lance l'algorithme de détection des plantons et détecte les plantons dans l'image. La partie grisée n'a pas été implémentée. Elle est prévue pour le paramétrage des options de détection et d'arrosage via l'écran tactile.

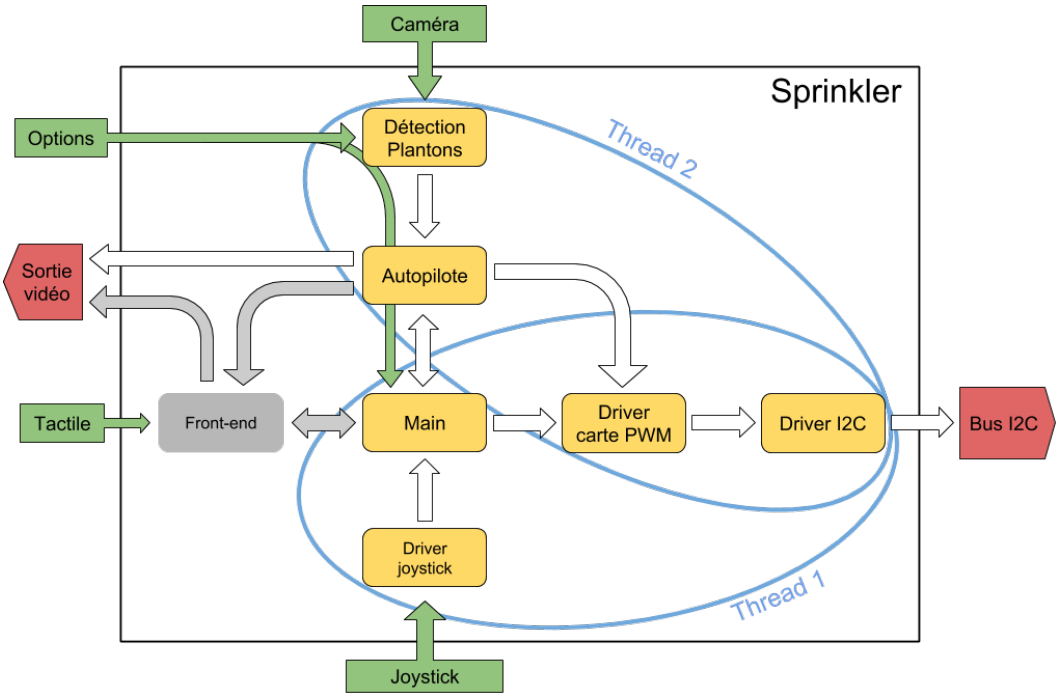

FIGURE 2.9 – Schéma de la structure logicielle [?], page 51

2.2.3 Banc de test

Le banc de test est une grande feuille de papier imprimée représentant le champ avec plusieurs images de plantons accolées.

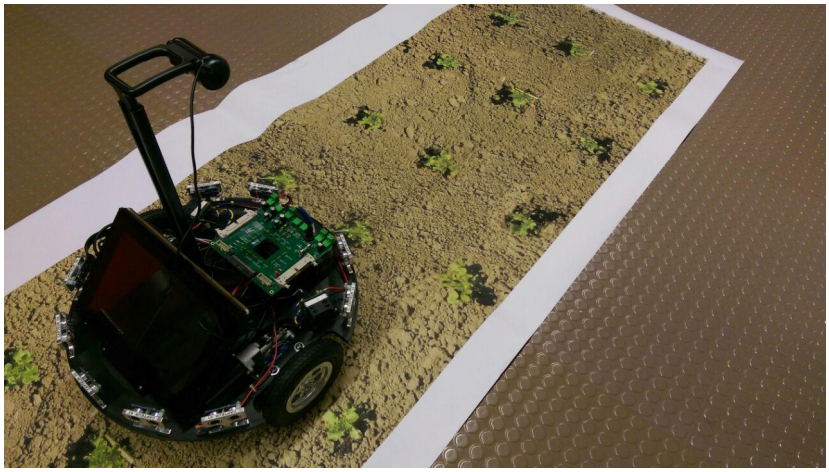

FIGURE 2.10 – Banc de tests [?], page 57

2.2.4 Langage de programmation et Librairie

Le projet a été implémenté en **C/C++**. Au tout début du projet, il était entièrement implémenté en C. Le programme s'est vu migré en partie dans le langage C++ à cause de l'utilisation de la librairie OpenCV.

OpenCV est la principale librairie utilisée dans ce projet. C'est une bibliothèque open-source conçue par Intel pour le traitement d'images en temps réel et officiellement rendu disponible en 1999 [**?**].

2.3 Mise en route du robot

Au début du projet, il m'a été confié deux carte SD, une contenant le projet et une deuxième carte ne contenant que l'OS noobs [**?**] préinstallé. Il m'a parru bon d'utiliser la deuxième carte pour réinstaller le projet, le recompiler et travailler dessus. Après avoir mis en marche cette carte, fort a été de constater qu'elle ne démarrait pas normalement. Il a fallut réinstaller l'OS avant d'installer et configurer OpenCV. La version 3.2.0 a été installée sur la carte sous recommandation de mon prédécesseur. Une difficulté s'est produite lors de l'installation. Cette difficulté était due à l'absence de la librairie TBB. Il a donc fallut la retrouver et l'installer. Toutefois, il faut faire attention de désactiver aussi l'option BUILD_TBB au risque de passer du temps à devoir débugger cela.

Il existe au total 25 paramètres disponibles au lancement du logiciel. Ces paramètres sont choisis en ligne de commande. L'option **-r** permet d'afficher cette liste de paramètres. Pour plus de détails concernant cette liste de paramètres, voir le manuel de l'utilisateur.

Le programme peut se lancer automatiquement en deux modes possibles. Le mode utilisateur est celui qui utilise le fichier script **/etc/xdg/lxsession/LXDE-pi/autosatart**. En effet, à la fin du démarrage du bureau GUI, ce fichier est exécuté. Il suffit donc de modifier ce fichier et d'ajouter une ligne correspondant au chemin vers le fichier exécutable du programme à démarrer. Le mode système est celui qui utilise le fichier script **/etc/rc.local**. En effet, lors du démarrage du système, ce dernier exécute ce fichier script. Il suffit de modifier ce fichier pour ajouter un programme à exécuter au démarrage.

Chose faite, au démarrage du robot, celui-ci avance tout seul et la détection des plantons se fait sur l'image. Le robot s'arrête à la fin d'une ligne de plantons. Il est aussi possible de le piloter manuellement.

Les détails d'installation reportés ici ont été mis à jour dans le document annexe nommé "Installation et configuration"

Chapitre 3

Commande des moteurs

3.1 Tests sur le fonctionnement des moteurs

Etant donné que les deux roues sont indépendantes, les moteurs sont pilotées par deux variateurs indépendants. Il a été constaté que le robot ne se déplaçait pas de façon linéaire. En effet, on observait une déviation accez conséquente au niveau de sa trajectoire à l'intervalle d'une très courte distance. S'il faut chiffrer cette déviation, on dira que sur une distance de moins d'un mètre, elle était inclinée d'une dizaine de degré. Plusieurs hypothèses ont été posées sur les raisons de ce disfonctionnement :

- Les valeurs de PWM ne sont pas correctes
- Les amplificateurs sont bas de gamme, et donc ils n'ont pas une bonne symétrie entre les moteurs
- Les roues ont un jeu
- Les moteurs DC n'ont pas une très bonne précision contrairement à des moteurs pas à pas

A l'issue de ces réflexions, nous avons procédé à des tests manuels au moyen d'un joystick et d'un oscilloscope. Nous voulions savoir si les valeurs de PWM étaient générées correctement.

3.1.1 Joystick Xbox 360 de chez Microsoft

Pour faire les tests sur les moteurs, il a fallu choisir et configurer une manette pour PC Linux compatible avec une Raspberry pi. Le joystick choisi est un joystick de chez Microsoft, le xbox 360. Il a été choisi parce qu'il fonctionne en wifi, très pratique pour effectuer les tests au niveau du pilotage manuel ; même si le robot est en mouvement, pas besoin de lui courir après. De plus il est compatible pour la Raspberry Pi. Il n'est pas nécessaire de reécrire un driver spécialement pour lui, son constructeur fournit une couche d'abstraction pour qu'il soit compatible avec le driver linux conventionnel. Il suffit donc d'installer ce driver et de le

lancer en ligne de commande [**?**]. Pour plus de détails sur Son installation et son utilisation, consulter son "manuel utilisateur" [**?**].

FIGURE 3.1 – Joystick xbox 360 [**?**]

Les tests consistaient à faire varier les valeurs de pulse d'entrée pour voir si les bonnes valeurs de PWM étaient générées en sortie. Les valeurs de PWM générées ont été affichées à l'oscilloscope. Nous avons pu constater que les valeurs mesurées étaient bien celles attendues. Il nous est donc venu à l'idée de faire une mesure des tensions aux bornes des amplificateurs. Ces mesures sont contenues dans le tableau suivant :

moteurs des roues				
PWM	TimeOn RG [ms]	TimeOn RD [ms]	Tension RG [V]	Tension RD [V]
819	0,992	0,992	-11,4	-11,5
1024	1,24	1,24	-5	-5,3
1106	1,34	1,34	-2,9	-3,2
1147	1,388	1,388	-1,8	-2,1
1188	1,44	1,44	-0,6	-1
1229	1,488	1,488	0	0
1270	1,588	1,588	0,8	0,4
1311	1,63	1,36	1,9	1,6
1352	1,636	1,636	3	2,6
1434	1,74	1,74	5	4,6
1638	1,984	1,984	9,6	9,6

TABLE 3.1 – Tension des moteurs en fonction des valeurs de PWM

Ce tableau présente un léger décalage entre les valeurs des tensions du moteur gauche par rapport aux valeurs des tensions du moteur droit. En effet, la roue droite va beaucoup plus vite que la roue gauche. C'est dons la raison pour laquelle le robot ne se déplace pas en ligne droite. Ceci pourrait être lié au type de moteur ou bien les roues auraient un jeu.

Pour bien se rendre compte de ce fait, il a été relevé deux courbes d'évolution des tensions. Il s'agit de deux fonctions croissantes. La courbe bleue représente celle du moteur gauche tandis que la courbe orange celle du moteur droit.

	819	1024	1106	1147	1188	1229	1270	1311	1352	1434	1638
V_LW	-11.4	-5	-2.9	-1.8	-0.6	0	0.8	1.9	3	5	9.6
V_RW	-11.5	-5.3	-3.2	-2.1	-1	0	0.4	1.5	2.6	4.7	9.6

FIGURE 3.2 – Courbe des tensions des moteurs des roues

3.2 Calibrage des moteurs

Au vu de ce qui précède il était donc nécessaire de procéder au calibrage des moteurs.

3.2.1 Calibrage des moteurs des roues

Pour calibrer ces moteurs, le moteur gauche a été pris comme moteur de référence. Il a été pris comme référence parce qu'il faut que le robot avance le moins vite possible afin qu'on puisse avoir le temps de voir la démonstration plus tard. Seulement, dans le champ réel, le robot devra aller le plus vite possible. Donc, il a fallu calibrer le moteur droit par rapport à celui de gauche.

La première idée de solution a été de trouver une formule mathématique qui permette de définir une valeur constante de correction à appliquer au moteur droit afin de rétablir les tensions aux sorties des amplificateurs. Cette formule consistait à prendre deux points au hasard, calculer la pente de la courbe à partir de ces point-là et ensuite en multiplier la valeur obtenue par chaque valeur de pulse d'entrée. On obtient ainsi la valeur du correcteur à ajouter à chaque abscisse pour augmenter les valeurs des tensions aux bornes de l'amplificateur du moteur droit. Mais cette solution s'est avérée sans succès. On observait encore et même en plus grand effet, le décalage.

Ensuite, une deuxième solution a été de calculer la pente point par point, et de trouver ainsi la valeur du correcteur à appliquer à chaque point indépendamment des autres. Là encore le résultat a été peu satisfaisant.

Finalement, la solution qui a bien marché a été de procéder point par point, en testant des valeurs PWM proches de celle initiale(roue gauche). Et la valeur PWM qui correspondait toujours à la tension attendue a été mémorisée pour la roue droite.

Le programme a été modifié en conséquence au niveau du fichier d'entête pour la définition de nouvelles valeurs de pulse d'entrée du PWM pour la commande de ces moteurs. Elle se fait au moyen de variable globales comme le montre l'extrait de code ci-dessous. Il a été définit la vitesse maximale (PWM_MAX_VALUE_LW) et la vitesse minimale (PWM_MIN_VALUE_LW), la vitesse maximale d'avancée (PWM_FORWARD_MAX_LW) et la vitesse minimale d'avancé (PWM_FORWARD_MIN_LW), ainsi que la vitesse maximale (PWM_BACK_MAX_LW) de recul et la vitesse minimale de recul (PWM_BACK_MIN_LW).

Listing 3.1 – Calibrage des moteurs pour les roues

```
// Left wheel
#define ADDR_REG_MOT_LW  ADDR_REG_MOT_0
#define PWM_MIN_VALUE_LW    655.0   /* 0,8ms */
#define PWM_BACK_MAX_LW     1024.0 /* 1,24ms */
#define PWM_BACK_MIN_LW     1147.0 /* 1,38ms */
#define PWN_STOP_LW         1229.0 /* 1,48ms */
#define PWM_FORWARD_MIN_LW  1311.0 /* 1,58ms */
#define PWM_FORWARD_MAX_LW  1434.0 /* 1,74ms */
#define PWM_MAX_VALUE_LW    1803.0 /* 2,2ms */


// Right wheel
#define ADDR_REG_MOT_RW  ADDR_REG_MOT_1
#define PWM_MIN_VALUE_RW   655.0    /* 0,80ms */
#define PWM_BACK_MAX_RW    1036.0  /* 1,25ms */
#define PWM_BACK_MIN_RW    1159.0  /* 1,40ms */
#define PWN_STOP_RW        1229.0  /* 1,38ms */
#define PWM_FORWARD_MIN_RW 1320.0  /* 1,59ms */
#define PWM_FORWARD_MAX_RW 1443.0  /* 1,75ms */
#define PWM_MAX_VALUE_RW   1803.0   /* 2,19ms */
```

3.2.2 Calibrage des moteurs du bras

Le principal objectif de ce travail étant l'arrosage des plantons, il était nécessaire de calibrer le bras d'arrosage par rapport aux millimètres. Ce calibrage permettra de déterminer le rapport entre une valeur de PWM correspondant à une distance en millimètre. Ceci permettra l'autopilotage du bras. Pour ce faire, des mesures de distance correspondantes à un ensemble de valeurs de PWM ont été prises. Mais avant, comme pour les moteurs des roues, il a d'abord fallut définir les valeurs de pulse d'entrée dans le logicielle de la manière suivante :

Listing 3.2 – Calibrage des moteurs pour les roues

```
// Arm axe X
#define ADDR_REG_MOT_ARMX  ADDR_REG_MOT_2
#define PWM_LEFT_MAX_ARM  800.0  /* 0.98ms */
#define PWM_RIGHT_MAX_ARM 1638.0 /* 2.00ms */

// Arm axe Y
#define ADDR_REG_MOT_ARMY  ADDR_REG_MOT_3
#define PWM_FORWARD_MAX_ARM 819.0  /* 1.0ms */
#define PWM_BACK_MAX_ARM  1638.0   /* 2.0ms */
```

Pour ce qui est des mesures, la figure ci-dessous présente une courbe d'évolution des points mesurés permettant d'analyser ce rapport PWM et millimètre. Elle a été construite à partir des résultats des mesures prises et reportées dans un tableau de valeurs comme le montre cette même figure suivante :

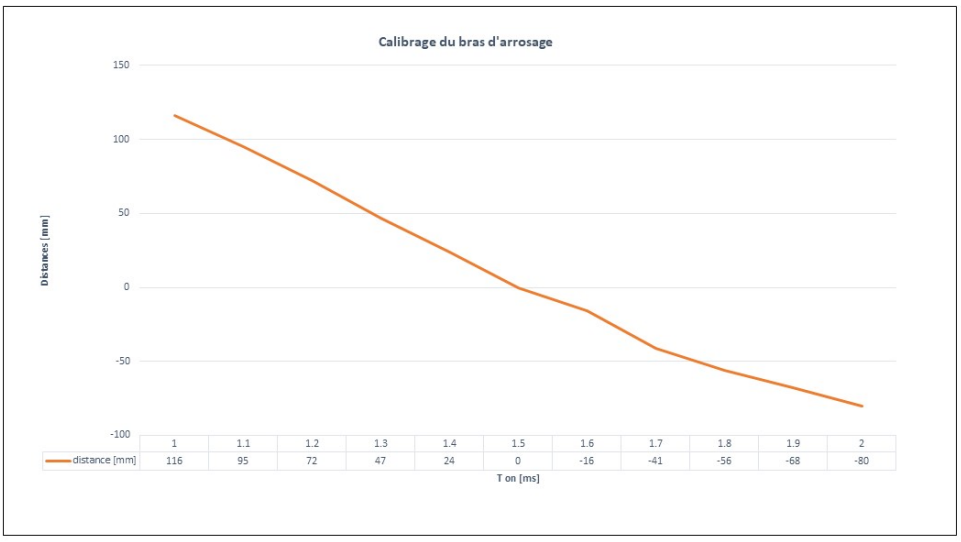

FIGURE 3.3 – Courbe d'évolution des posotions du bras sur l'axe Y en fonction du pwm

On remarque que les valeurs de distance mesurées ne sont pas symétriques. En effet, le bras s'étend plus en avant qu'en arrière par rapport à sa position neutre qui est (1.5, 0). On observe aussi que la courbe n'est pas tout à fait linéaire. Celà est du à la structure mécanique du bras, composée de deux roues dentées et d'un segment associant ces deux roues. Notons que ces segments doivent avoir la même longueur pour que le mouvement du bras soit parfaitement linéaire. La méthode par interpolation linéaire a été utilisée afin de trouver cette relation PWM et distance en mm.

L'Interpolation linéaire est une opération mathématique qui consiste à estimer la valeur prise par une fonction continue entre deux points déterminés. La méthode consiste à utiliser

la fonction affine définie par **y = ax + b** entre ces deux points déterminés [**?**]. Supposons que l'on connaisse les valeurs prises par notre fonction f en deux points (X1, Y1) et (X3, Y3) et que l'on veille calculer la valeur Y2 correspondant à X2 comprise entre (X1, Y1) et (X3, Y3). L'équation est la suivante :

$$Y_2 = Y_3 + \frac{(X_2 - X_3)(Y_3 - Y_1)}{(X_3 - X_1)}$$

Il est important de mentionner que lors des prises de mesures, on a pu remarquer qu'il existe une valeur d'offset d'environ 4 mm lorsque le bras se déplace de l'arrière vers l'avant par petit pas jusqu'à sa position maximale. Ce qui n'est pas bien grâve puisque le programme a besoin de déplacer le bras une fois vers l'avant et n fois vers l'arrière.

Au niveau de l' implémentation, une lookup table a été déclarée. C'est en faite un tableau constant avec des valeurs prédéfinies basé sur le tableau de valeurs distance et PWM de la Figure 3.3. Une fonction qui implémente la technique d'interpolation linéaire se sert de ce tableau afin de calculer la position du bras dans la fourchette de PWM de 1 ms à 2 ms qui correspond à la fourchette de distances -80 mm à 116 mm.

3.3 Difficultés rencontrées

Nous avons accusé plusieurs jours de retard à cause d'une panne qui est survenue au niveau d'un câble défectueux. La carte Raspberry Pi rebootait toute seule et nous ne savions pas pourquoi. Puisque les tensions correspondaient à celles escomptés aux bornes de la carte d'alimentation. Après plusieurs tentatives de dépannage, il a finalement fallu faire appel au technicien qui a scruté chaque câble. Il se trouve que c'était la faute au câble qui connecte la carte d'alimentation avec la Raspberry.

Ce problème résolu, une autre panne est survenue quelques jours plus tard. Et celle-ci n'était pas facile à deviner puisqu'il s'agissait d'une panne mécanique. Une pièce du bras d'arrosage était défectueuse, et il y'avait un jeu au niveau du mouvement du bras. Ce qui impliquait que les résultats du calibrage du bras étaient toujours faux. C'est lors de la démonstration intermédiaire du projet, que l'expertise du professuer qui encadre le projet de mécanique nous a été d'une grande aide. Ce dernier nous a proposé de collaborer avec son étudiante pour trouver rapidement une solution. Aussitôt chose dite, chose faite. Heureusement pour nous, cette dernière a réagit promptement et a pu corriger la panne le même après-midi que nous l'avions contacté.

Chapitre 4

Conception du système d'arrosage

Rappelons que le système à implémenter doit pouvoir détecter le début d'une ligne de plantons et commander automatiquement le bras d'arrosage pour arroser les plantons. Pour celà, il se sert des fonctionnalités déjà existantes, de détection des plantons et de suivi d'une ligne de plantons du champ.

4.1 Spécifications fonctionnelles

Pour bien comprendre le fonctionnement de ce système d'arrosage, il a été représenté les actions du mouvement du bras et du robot dans le champ sous forme graphique comme le montre le schéma ci-dessous. Le premier graphique représente le champ. Sur l'axe des ordonnées est représenté les positions des plantons Pi de coordonnées (Xi, Yi). La courbe bleue/brune représente la position du bras dans le champ. Et la courbe verte quant à elle, la position du robot dans le champ. Cette courbe est une droite passant par les points constitués des coordonnées des plantons détectées. Sa pente est donc connue et vaut :

$$m = \frac{Yi - Yi - 1}{Xi - Xi - 1}$$

Donc la pente étant une constante. La vitesse du robot qui est en faite la dérivée de cette droite (fonction linéaire) est donc constante. Dès lors, on dit que le robot avance à vitesse constante.

Le deusième graphe est représente les mouvements du bras dans le temps. Dès qu'un planton est détecté, le bras est déplacé vers l'avant au dessus du planton détecté. Pendant l'arrosage, le bras est déplacé dans le sens inverse du robot de manière à ce qu'il soit maintenu au dessus du planton qui est en train d'être arrosé. Lorsque la durée d'arrosage est écoulée, le bras est déplacé le plus rapidement possible vers le prochain planton détecté et le processus recommence.

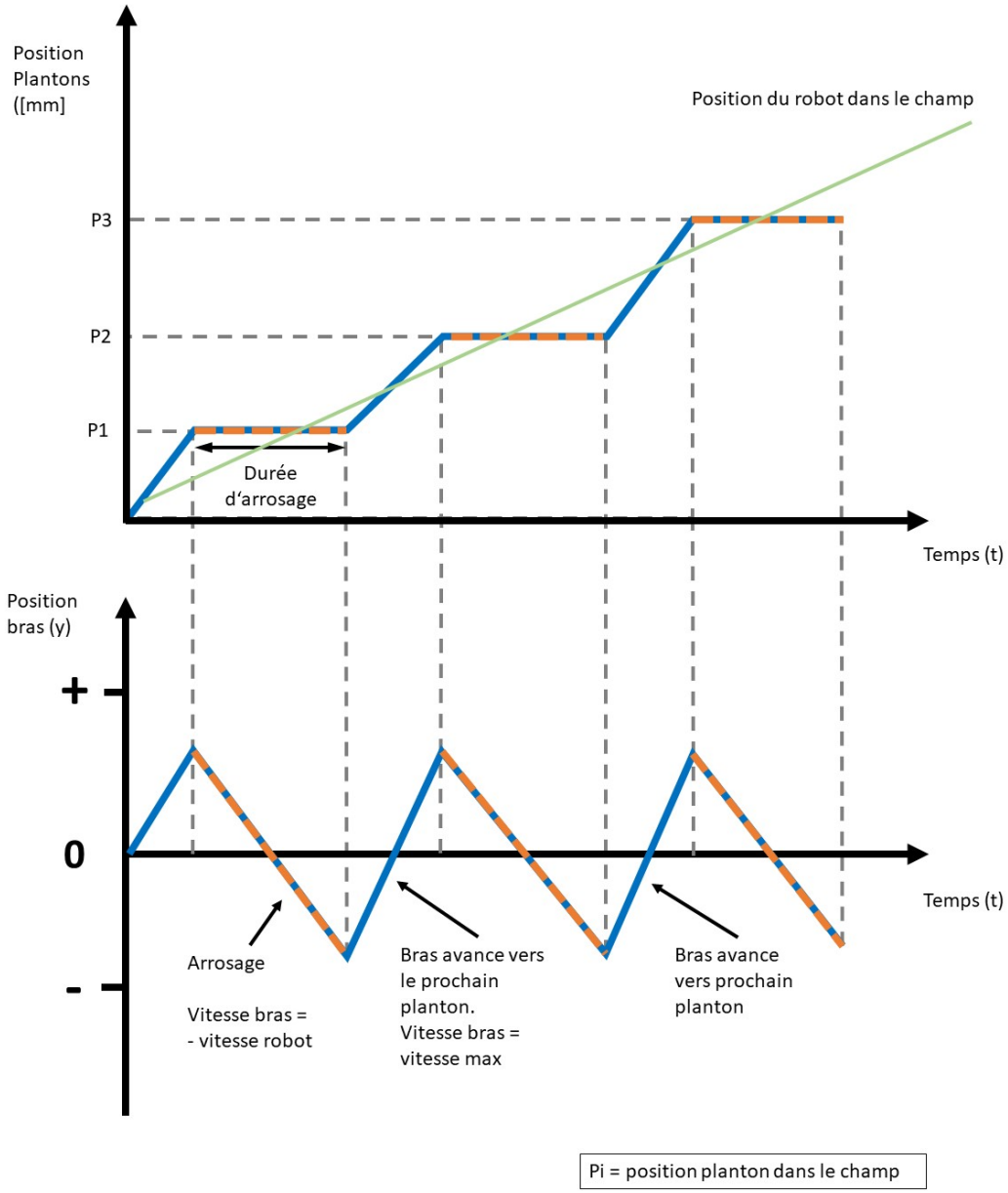

FIGURE 4.1 – Schéma du mouvement du bras d'arrosage et du robot dans le champ en fonction du temps

4.1.1 Fonctionnement du système d'arrosage

Le schéma ci-dessous représente le structogramme du logiciel à implémenter pour faire marcher le robot arroseur.

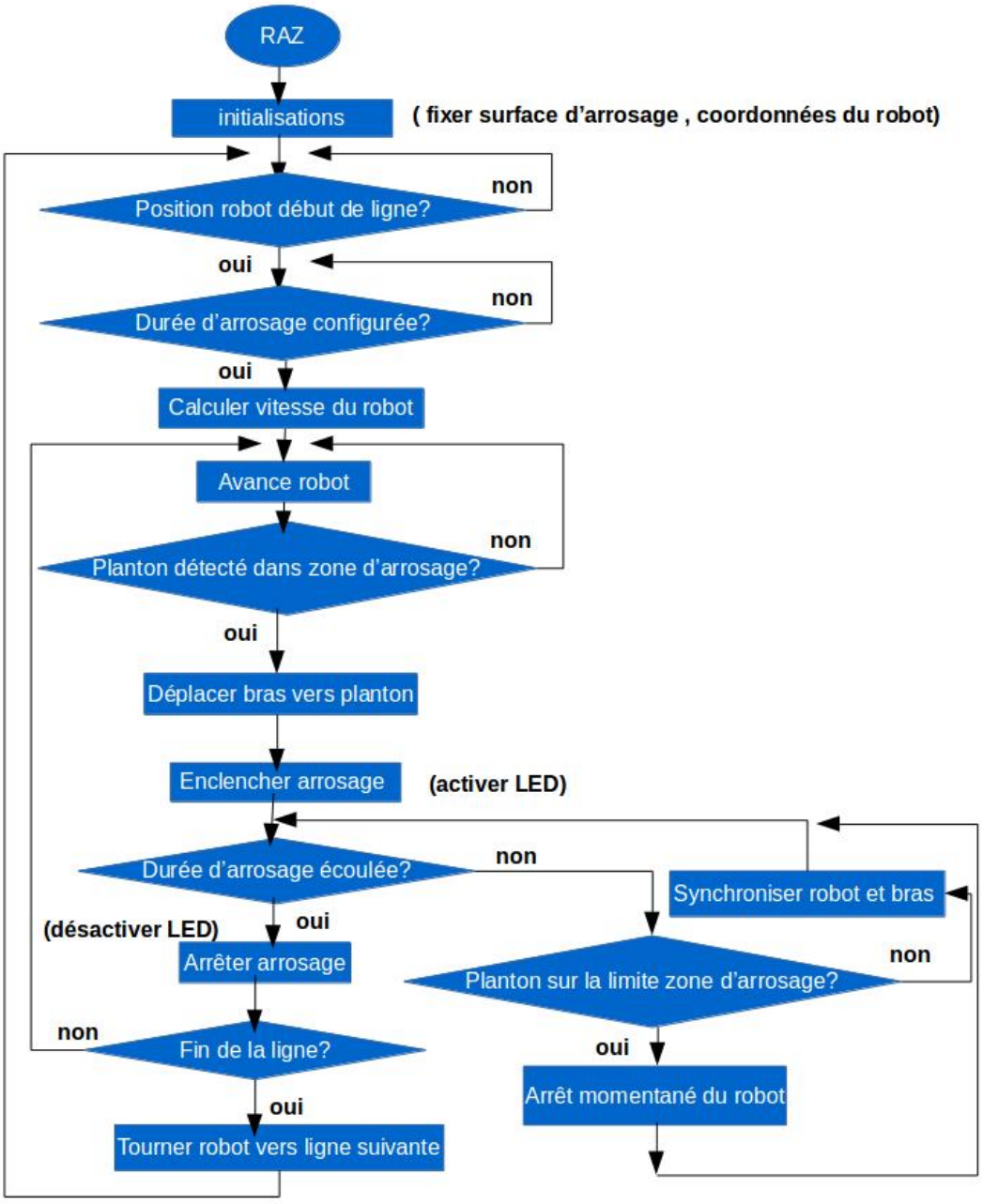

FIGURE 4.2 – diagramme du fonctionnement du système d'arrosage

Initialisations : Au début, le programme effectue toutes les initialisations. Ces initialisations consistent à définir la zone d'arrosage et le point de référence du robot (coordonnées du robot dans le champ). Ce point de référence est fixe par rapport à la caméra et donc permet d'avoir toujours la position robot dans le champ à tout instant t.

Détection du début d'une ligne de plantons : Il doit ensuite pouvoir vérifier que le robot se trouve au début d'une ligne de plantons, sinon le programme demande à l'utilisateur de veiller à bien positionner le robot au debut de la ligne de plantons. Le début d'une ligne est détecté à l'aide de la caméra. Il existe des algorithmes pour la détection des contours. On peut imaginer de déssiner une forme géométrique (triangle, cercle,...) au debut de chaque ligne de plantons. Le programme détectera grâce à l'algorithme de détection des contours approprié, la ligne formée par l'ensemble des figures géométriques. Il vérifira que la droite formée par les centres de ces figures géométriques est perpendiculaire à la caméra. Il se dirigera vers la figure géométrique la plus proche, qui correspond en fait à la ligne de plantons à traiter (arrosage des plantons de la ligne).

Configuration de la durée d'arrosage : Le programme se met en attente que l'utilisateur configure les paramètres de l'arrosage(la qauntité/volume d'eau par planton). On peut imaginer plusieurs paramètres supplémentaires (type de plantons). Ces paramètres doivent être des valeurs positives et non nulles. Une interface graphique sera implémentée pour permettre cette configuration.

Calcul de la vitesse du robot : Il calcule la vitesse du robot en fonction du débit d'eau donné en paramètre. Cette vitesse est utile pour configuer l'arrosage (l'accelérer ou le décélérer). La formule de la vitesse en [mm/s] est donnée par la formule suivante :

$$Vrobot = \frac{D(Pi, Pi-1)}{TmouvementBras}$$

$$TmouvementBras = Tarrosage + TretourBras$$

Cette vitesse est obtenue en faisant le rapport de la distance parcourue par le robot entre deux plantons (300 mm) et la durée totale du mouvement deu bras. Cette durée totale du mouvement du bras représente la somme entre la durée d'arrosage et le temps mis par le bras à la fin d'un cycle d'arrosage pour se déplacer vers le prochain planton.

Une durée d'arrosage trop grande diminue la vitesse du robot et inversément une durée courte augmente sa vitesse. Il sera nécessaire de connaître les vitesses minimale et maximale du robot(à mesurer à l'aide du **tacomètre**). En effet, nous sommes limités par les caractéristiques des moteurs des roues utilisés, qui ne fonctionnent que dans une tranche de valeurs précise(voir Figure 3.2).

Faire avancer le robot : Le programme démarre le robot en lançant le thread 2 en charge de gérer la détection et le suivi d'une ligne de plantons.

Détection de planton dans la zone d'arrosage : La zone d'arrosage est délimitée par la distance entre la position maximale du bras vers l'avant et sa position maximale vers l'arrière par rapport à sa position initiale. Sa longueur est de 195 mm.

Pour détecter si un planton est dans la zone d'arrosage, il faut calculer le delta entre la position du robot et la position du planton détecté dans le champ. Si ce delta est inférieur ou égal à 116 mm, alors le planton est dans la partie supérieure de la zone d'arrosage. Si par contre le planton détecté est situé au-dessous du robot, alors delta est calculé différemment puisque le planton en question n'est plus présent dans le champ de vision du robot.

Dans ce cas, il faut additionner la position du planton détecté avec la distance entre deux plantons. Ensuite il faut soustraire à cette valeur, la position du robot dans le champ. Le planton sera considéré dans la zone d'arrosage si et seulement si ce delta est inférieur ou égal à 80 mm.

Lorsqu'un planton est détecté dans la zone d'arrosage du robot, le programme déplace la tête d'arrosage au-dessus de celui-ci pour qu'il soit arrosé.

Déplacer le bras au-dessus du planton : Pour déplacer le bras d'arrosage au-dessus du planton détecté, il faut au préalable connaître la distance qui sépare le robot et du planton détecté. Cette distance sera obtenue au moyen de la caméra. Il faudra trouver le rapport entre le pixel(unité de mesure sur l'image) et le millimètre(unité de mesure dans le champ). Cette distance (delta) connue, la consigne PWM correspondant à cette distance est alors calculée. À partir de cette consigne le bras est commandé pour être placé au-dessus du planton détecté. Il faut prévoir une valeur de correction puisque nous sommes en temps réel et qu'il exite un temps de latence entre l'acquisition de l'image et son affichage sur l'écran tactile. Ce temps vaut 2 [s] environs. Cette correction, est une valeur de distance estimée en fonction de la durée de latence d'affichage de l'image et la vitesse du robot.

Enclenchement de l'arrosage : Par la suite, l'arrosage est enclenché(activation de la LED) et dure tant que la durée d'arrosage n'est pas écoulée.

Durée d'arrosage écoulée ? : Ce contrôle est fait au moyen d'un compteur de temps. Il commence à 0 et s'incrémente jusqu'à ce qu'il atteigne une valeur égale à la durée d'arrosage, puis il est reinitialisé.

Synchronisation robot et bras : Puisque le robot continue d'avancer simutanémement pendant l'arrosage, le programme ajuste la position de la tête d'arrosage en fonction de la vitesse d'avance du robot, afin de le maintenir au-dessus du planton en train d'être arrosé. Pour cela, la nouvelle position du bras sera calculée au moyen de la position courante du robot, de la position du prochain planton détecté et de la distance entre deux planton(valeur fixe égale à 300 mm). Ceci poserait un problème dans le cas où la distance entre les planton n'est pas toujours fixe.

Arrêt de l'arrosage : Lorsque cette durée d'arrosage expire, l'arrosage est arrêtée(désactivation de la LED).

Planton sur la limite de la zone d'arrosage ? Si la durée d'arrosage est trop grande et que le planton atteint la limite de la zone d'arrosage du robot sans être complètement arrosé, le programme autorise un arrêt momentané du robot(envoie de la consigne stop au moteurs des roues). Le robot reste immobile jusqu'à la fin de la durée d'arrosage restante.

Détection de la fin d'une ligne de plantons : Lorsque le programme détecte la fin d'une ligne(moins de trois plantons détectés), il fait tourner le robot vers la ligne suivante.

Faire tourner le robot faire la ligne suivante : c'est faire une rotation sur lui même de 90 degrés, puis le faire avancer de 400 mm (distance entre deux ligne de plantons) et enfin, faire une autre rotation de 90 degrés.

4.1.2 Fonctionnalités nécessaires

De ce qui précède, les principales tâches à effectuer pour réaliser ce système sont les suivantes :

- Définir la zone d'arrosage du robot
- Calculer la vitesse du robot
- Déterminer les coordonnées du robot dans le champ
- Calculer les coordonnées des plantons détectés dans le champ
- Calculer la distance entre le robot et le planton détecté
- Ajuster la position du bras par rapport à la vitesse du robot
- Arroser les plantons détectés
- Détecter le début d'une ligne
- Faire tourner le robot vers la ligne suivante
- Concevoir une interface graphique pour permettre à un utilisateur de paramétrer l'arrosage

Toutes ces tâches peuvent se résumer en cinq fonctionnalités mimimnales :

1. Calculer la Vitesse du robot
2. Calculer la Distance entre le robot et le planton détecté (Traitement planton détecté)
3. Réguler le mouvement du bras (avancer, reculer)
4. Arroser les plantons (commande de la pomme à eau)
5. Front-end (paramètres de configuration de l'arrosage)

Le détail de chacun de ces modules est donné ci-dessous :

Vitesse du robot : Il est nécessaire d'écrire une fonction pour le calcul de la vitesse du robot. Elle prendra en paramètre la durée d'arrosage. La distance entre deux plantons est fixe (300 mm), idem pour le temps de retour du bras.

Traitement planton détecté : L'acquisition des coordonnées sur l'image des plantons détectés se fait par le module de détection. Ces coordonnées sont envoyeés au module de traitement des plantons détectés qui calcule les coordonnées des plantons dans le champ et la distance entre le robot et le planton détecté sur la ligne.

Régulation du bras : Une fois la distance calculée, le module de régulation du bras s'en sert pour commander le bras (avancer ou reculer) afin de le positionner au-dessus du planton détecté.

Arrosage : Lorsque le bras est bien positionner au-dessus du planton, l'arrosage est enclenchée (si la décision a été d'avancer) et ou l'arrosage est arrêtée (si la quantité d'eau pour ce planton est couvert).

Front-end : C'est une interface graphique qui permettra de configurer les paramètres de l'arrosage (quantité d'eau, type de planton par exemple...)

4.1.3 Structure logicielle

Le schéma de la structure logicielle ci-dessous présente les différents modules qui ont été cités plus haut et leurs interactions. Comme mentionné précédemment, ces modules sont le traitement du planton détecté, la régulation du mouvement du bras d'arrosage, l'arrosage des plantons et le front end. L'idée première était que cette structure du logiciel conserve les deux threads déjà existants. Le premier est le thread principal. Son rôle est d'invoquer le deuxième thread, de parser les options et les paramètres d'entrée, ainsi que la gestion du joystick pour la commande manuelle. Outre cest tâches, il permet aussi le calcul de la vitesse du robot. Le deuxième thread quant à lui s'occupe de l'autopilotage du bras et des roues, de la détection et de l'arrosage des plantons.

Pour rappel, un thread est une partie d'un programme en cours d'exécution. Un programme peut être constitué de plusieurs threads qui s'exécutent quasi-simultanément ou simultanément sur des processueurs à plusieurs coeurs. Les threads partagent le même espace mémoire et les même ressources contrairement aux programmes en cours d'exécution (processus). Il est donc nécessaire de protéger lors de la lecture ou l'écriture, ces ressources pour garder leur intégrité tout au long de l'exécution du programme (exclusion mutuelle). Plusieurs mécanismes de protection existent, à l'instar des sémaphores et des mutex, pour ne citer que ceux là. Nous opterons pour une solution avec mutex, car simple d'implémentation.

Pour ce qui est des couleurs, les rectangles gris représentent les fonctionnalités existantes et les rectangles jaunes sont les fonctionnalités à implémenter. Les rectangles verts sont les entrées et les rectangles rouges sont les sorties au système.

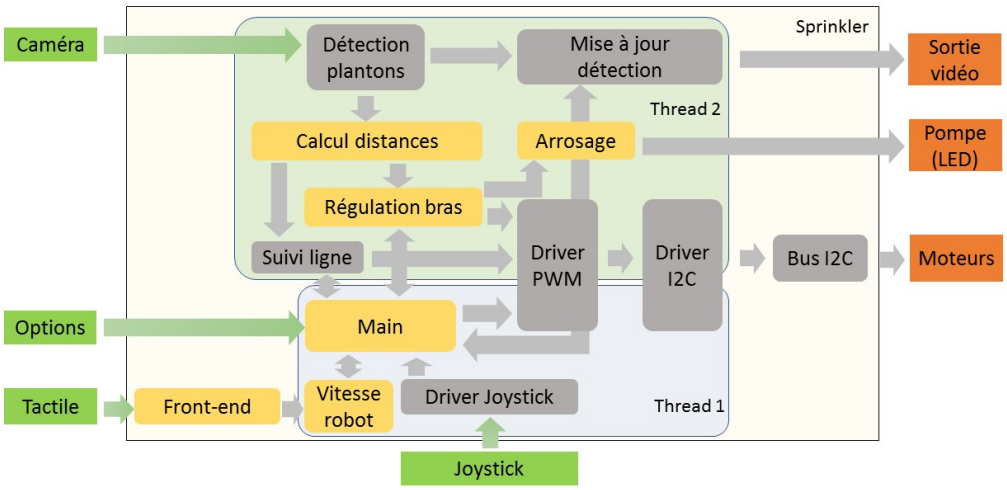

FIGURE 4.3 – Structure logicielle du robot d'arrosage

4.2 Implémentation du système d'arrosage

L'implémentation du logiciel se fait en plusieurs étapes qui sont les suivantes :

4.2.1 Initialisations

1. La définition d'une zone d'arrosage. Les valeurs 116 mm et 80 mm seront définis dans le programme comme des constantes globales, pour servir de limite pour la zone d'arrosage

2. La définition d'un point de référence (origine) du robot. Ce point sera aussi défini comme constante globale dans le programme à partir des mesures faites à la règle depuis le point d'origine du repère du champ jusqu'au point d'origine du robot dans le champ. Il a été choisi que le point d'origine du robot est aussi le point d'origine du bras.

4.2.2 Régulation du mouvement du bras

La régulation du bras comprend deux actions, l'avancée du bras au-dessus du planton, et l'ajustement de la position du bras en fonction de la vitesse d'avancée du robot pendant l'arrosage, pour le maintenir au-dessus du planton pendant la durée d'arrosage.

Pour déplacer le bras vers le planton détecté, il faut calculer la distance qui le sépare de celui-ci. Pour se faire, il faut acquérir la coordonnée du planton détecté, la convertir en coordonnée du planton dans le champ, aquérir la position du robot dans le champ, calculer le delta entre le robot et le planton et enfin envoyer la consigne au moteur pour déplacer le bras à la position souhaitée. Les tâches à effectuer pour la régulation du bras sont les suivantes :

1. Calcul de la distance entre le planton détecté et le robot
    - La définition de l'angle de la caméra
    - Le calibrage de la caméra (calcul de la matrice de convertion).
    - La convertion des coordonnées des plantons de l'image avec distorsion (pixel) en coordonnées des plantons de l'image sans distorsion (pixel)
    - La définition du rapport entre les pixels et les millimètres.
    - Le calcul des coordonnées du robot dans le champ
    - Le calcul du delta entre le robot et le planton détecté
2. Calcul de la consigne PWM correspondant à distance trouvée comme expliquée précédemment.

4.2.3 Arrosage des plantons

Arroser revient à :

1. L'enclenchement de l'arrosage ou ouverture de la vanne (activation de la LED)
2. Le contrôle du débit d'eau (La gestion d'un compteur de temps pour contrôler la durée d'arrosage)
3. Le désenclenchement de l'arrosage ou fermeture de la vanne (La désactivation de la LED)

4.2.4 Vitesse du robot

Cette fonctionnalité sert à régler la vitesse du robot en fonction du volume d'eau de la pompe. Un grand volume d'eau implique une faible vitesse du robot et inversément, un faible volume implique une plus grande vitesse du robot. Dans le cas de ce démonstrateur, la vitesse est configurée à partir de la durée d'arrosage. Pas exemple si on double la durée d'arrosage, alors la vitesse est divisée par deux. Une fois cette vitesse calculée, elle est constante tout au long de l'exécution du programme.

4.2.5 Frontend

Il s'agit de permettre à l'utilisateur de choisir une valeur pour la débit d'eau au moyen d'une interface graphique sur l'écran tactile. Cette valeur permet de configurer la vitesse du robot.

L'outil choisi pour l'implémentation de l'interface graphique est Qt. C'est un set de librairies qui fournit un environnement de developpement complet pour le développment d'applications utilisateurs. Il très connu et déployable pour divers types de devices comme les tablettes, téléphones... De plus il est basé sur le langage c/c++ utilisé pour notre logiciel. Pour arriver à l'utiliser il faut :

1. Cross-compiler et déployer Qt pour la Raspberry Pi
2. Implémenter l'interface graphique avec Qt
3. Consolider l'interface avec le reste du projet

4.2.6 Détection du début d'une ligne

Pour détecter la début d'une ligne, nous utiliserons l'algorithme de détection des contours. Cet algorithme permettra au programme de détecter une forme géométrique sur l'image de la caméra, par exemple un triangle. Ces triangles seront dessinés au debut de chaque ligne de plantons, de sorte qu'ils soient bien alignés horizontalement. Ensuite, il faudra calculer la moyenne des centres de gravité de ces triangles. Soient le centre de gravité G(gx, gy) du triangle ABC et A(ax,by), B(bx,by) et C(cx,cy) dans un repère orthonormé (O, x,y).

$$gx = \frac{ax + bx + cx}{3}$$

$$gy = \frac{ay + by + cy}{3}$$

Si cette moyenne diffère trop de la moitié de la largeur de l'image, le robot n'est pas en face d'une ligne de plantons. Alors le logiciel terminera la capture et affichera un message d'erreur à l'utilisateur pour lui signifier de bien positionner le robot devant la ligne. Autrement, il détectera qu'il s'agit bien du début d'une ligne. Il dirigera le robot vers la ligne dont la position de la forme géométrique est la plus proche (triangle avec le plus petit abscisse). Pour effectuer cette détection, il faudra :

1. Dessiner les formes géométriques en face de chaque ligne de plantons
2. Implémentation de l'algorithme de détection de la forme géométrique (triangle)
3. Calculer la moyenne des positons (centres) des triangles détectés.

4.2.7 Faire tourner le robot vers la ligne suivante

Supposons que la première ligne du champ est à gauche et que le reste du champ est à droite de cette ligne, alors il faudra :

1. Effectuer une rotation de 90 degrés à droite
2. Avancer de 400 mm
3. Effectuer une rotation à droite de 90 degrés

4.3 Le planning

Le planning intégrant les différentes étapes et leurs répartitions dans le temps figure en annexe "planningFinal". Ce planning a subi plusieurs modifications. En effet, certaines tâches ont pris plus de temps que prévu et pour d'autres beaucoup moins. Ceci est dû à leurs niveaux de complexité respectives qui ont varié plus la compréhension du sujet devenait claire.

Chapitre 5

Commande automatique de l'arrosage

5.1 Initialisations

5.1.1 Définition de la zone d'arrosage

Un planton détecté ne peut être arrosé que s'il est présent dans la zone d'arrosage. Cette zone est définie par la longueur de la course du bras d'arrosage (la distance qui sépare la position maximale du bras vers l'arrière (80 mm) et sa position maximale vers l'avant (116 mm)). La mesure de cette zone a été prise à la règle est de 195 mm de long. Elle a été défini dans le logiciel comme le montre l'extrait de code suivant :

Listing 5.1 – Définition de la zone d'arrosage

```
#define FORWARD_MAX_ARM_DISTANCE 116.0
#define BACKWARD_MAX_ARM_DISTANCE 80.0
```

5.1.2 Point de référence du robot

Pour définir les coordonnées du robot dans le champ, il a fallu mettre des marqueurs sur le champ afin de définir les limites du champ en fonction de l'angle de vision de la caméra. Le schéma ci-dessous montre l'image du champ capturée par la caméra avec les marqueurs posés dessus. Ces marqueurs sont indiqués au moyen des flèches jaunes sur l'image.

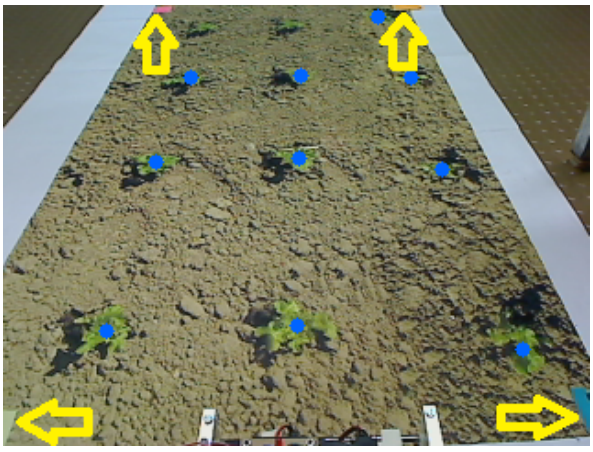

FIGURE 5.1 – Image capturée par la caméra avec les marqueurs sur le champ

À partir de ces marqueurs il a été mesurées à la règle les limites du champ et la position du robot dans ce champ. La position du robot est fixe par rapport au repère du champ. La figure suivante montre le repère du champ qui en découle.

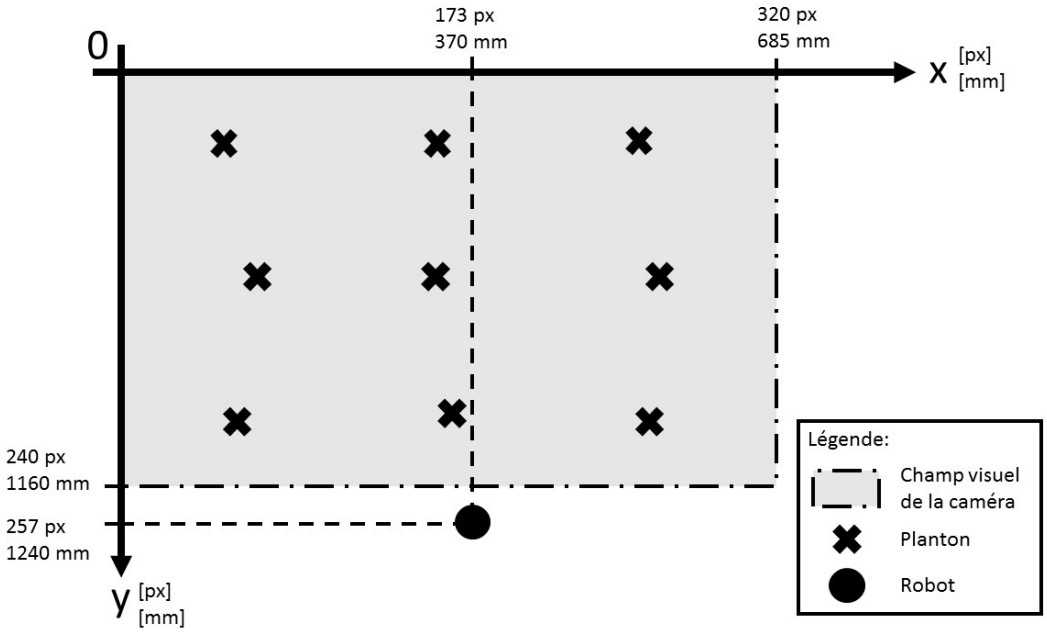

FIGURE 5.2 – Schéma du repère du champ avec le point de référence du robot

Au niveau du logiciel, cela se traduit par les lignes de code suivantes :

Listing 5.2 – Limites du champ et point de référence du robot

```
// ground size [mm]
#define GROUND_WIDTH 685.0
#define GROUND_HEIGHT 1160.0

// Robot ground coordinates
#define ROBOT_X 370.0
#define ROBOT_Y 1240.0
```

5.2 Régulation du bras d'arrosage

Deux actions définissent la régulation du bras, avancer et reculer. Avancer pour positionner le bras au-dessus du planon et reculer pour maintenir le bras au-dessus du planton pendant la durée d'arrosage. Mais pour ce faire, il faut connaître de quelle distance déplacer le bras dans l'un ou dans l'autre sens.

5.2.1 Calcul de la distance entre le planton détecté et le bras d'arrosage

Le calcul de la distance s'est faite en plusieurs étapes. Il a fallu fixer l'angle de la caméra, calibrer la caméra, trouver le rapport entre les pixels et les millimètres et calculer le consigne PWM en fonction d'une valeur de distance donnée.

Fixer l'angle de la caméra : Puisque le pilotage du bras dépend des captures d'images faites par la caméra, il est important de fixer l'angle de la caméra. Pour définir cet angle, un marquage sur le robot, au niveau du joint de la tige de la caméra avec la caméra. Cette marque permet de positionner la caméra avec le même angle pour que l'utilisation du robot d'arrosage se fasse toujours avec les mêmes conditions initiales, pour des résultats cohérents.

Calibrage de la caméra : Cette tâche est utile parce que la plupart du temps, les problèmes de distorsion d'images peuvent se produire lors de la capture d'images au moyen d'une caméra. Cette situation est due à l'angle de vision de la caméra qui n'est pas perpendiculaire par rapport au sol. L'image suivante illustre bien ce problème. Elle se présente avec une déformation en perspective. En effet, l'image semble se retressir plus on va en profondeur d'elle.

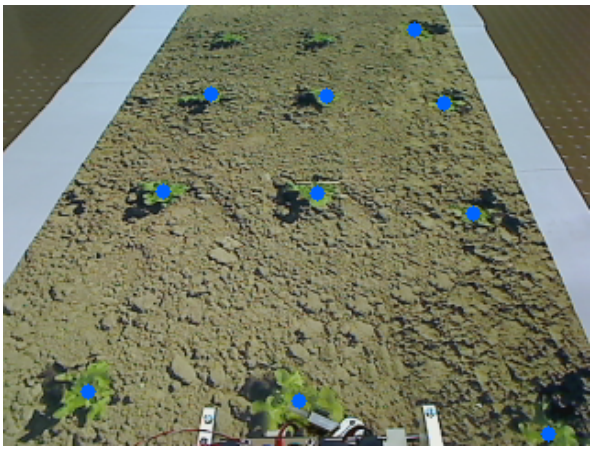

FIGURE 5.3 – Image capturée par la caméra

Il était donc nécessaire d'éliminer cette distorsion. Ces problèmes de distorsion peuvent être réglés en calculant le vecteur de distorsion de l'image et en l'appliquant à tous les pixels de l'image déformée pour obtenir une image plus uniforme. Ce vecteur de distorsion n'est rien d'autre qu'une matrice M 3 par 3 [**?**]
Considérons le pixel x de coordonnée (u, v, w) dans une image source et le pixel x' de coordonnée (ui, vi, wi) dans une image destination, le vecteur de distorsion de l'image est obtenue à partir de l'équation suivante :

$$\begin{bmatrix} ui_{11} \\ vi_{12} \\ wi_{13} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \begin{bmatrix} u_{11} \\ v_{12} \\ w_{13} \end{bmatrix}$$

Ce vecteur relie les coordonnées du pixel dans les deux images tel que :

$$x' = M \times x \tag{5.1}$$

Lorsqu'elle est appliquée à tous les pixels, la nouvelle image est une image transformée de l'image originale, ayant la même résolution. Dans notre cas, l'image est capturée en perspective. OpenCV a prévu l'implémentation du vecteur de distorsion par la fonction **getPerspectiveTransform()**. Cette fonction prend en entrée quatre points de l'image source et quatre points de l'image destination et calcule la matrix de distorsion de l'image en perspective [**?**]. Le résultat de la transformation est donné par l'image de droite. On voit bien que

l'effet perspective a disparu. Il y'a aussi deux bouts de l'image transformée au coin gauche et droit, qui sont sombres, ceci est normal puisque la caméra ne capture pas l'entièreté du champ.

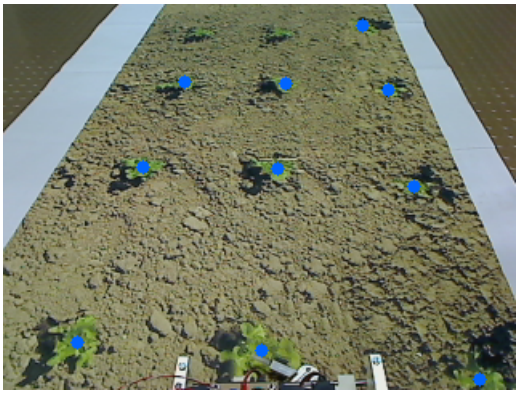

FIGURE 5.4 – Image avec distorsion

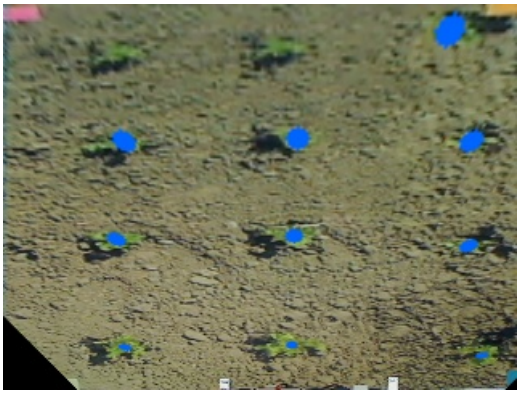

FIGURE 5.5 – Image sans distorsion

Pour calculer la matrice de convertion, il faut donc choisir quatre points de référence de l'image source et quatre points de référence de l'image destination.

Listing 5.3 – Choix des points de référence pour la matrice de converstion

```
// Input Quadilateral or Image plane coordinates
Point2f cameraQuad[4];
// Output Quadilateral or World plane coordinates
Point2f worldQuad[4];


// calibration camera init
/* The 4 points that select quadilateral on the input ,
 * from top-left in clockwise order
 * These four pts are the sides of the rect box used as input
 */
cameraQuad[0] = Point2f(83, 0);
cameraQuad[1] = Point2f(220, 0);
cameraQuad[2] = Point2f(0, 240);
cameraQuad[3] = Point2f(318, 240);

/* The 4 points where the mapping is to be done ,
 * from top-left in clockwise order
 */
worldQuad[0] = Point2f(0, 0);
worldQuad[1] = Point2f(320, 0);
worldQuad[2] = Point2f(47, 240);
worldQuad[3] = Point2f(310, 240);
```

Les quatre points pixels de l'image source de la caméra (cameraQuad) sont représentés par les coordonnées des quatre marqueurs sur l'image capturée avec les marqueurs. A l'aide de l'éditeur d'image **Gimp Image Editor** de Linux, on a pu acquérir la valeur des coordonnées en pixel de chaque marqueur sur l'image capturée. Ils sont pris de la manière suivante : en haut à gauche, en haut à droite, en bas à gauche et en bas à droite. Il est recommendé de respecter cet ordre pendant l'implémentation. Il est aussi important de préciser qu'il faut prendre les points de sorte que l'image soit la plus grande possible afin de limiter les erreurs liées à la transformation.

Les quatres points en pixels du champ (worldQuad) sont pris à partir des coordonnées des limites du champ (hauteur = 1160 mm et largeur = 685 mm) . Nous avons pu calculer en pixel les coordonnées des deux marqueurs du bas de l'image (La caméra ne capture pas l'entièreté du champ) comme le montre les équations ci-dessous. De ceci, découle aussi la relation entre les pixels et les millimètre.

$$position_{[pixels]} = position_{[mm]} \div 685[mm] \times 320[pixels] \tag{5.2}$$

$$\begin{aligned} wordQuad[2].x &= 100 \div 685 \times 320 && (5.3)\\ &= 47_{[pixels]} && (5.4) \end{aligned}$$

$$\begin{aligned} wordQuad[3].x &= 662 \div 685 \times 320 && (5.5)\\ &= 310_{[pixels]} && (5.6) \end{aligned}$$

À l'issu du choix de ces points de référence, il ne restait plus qu'à calculer la matrice de convertion. Cela s'est fait de la manière suivante :

Listing 5.4 – Initialisation de la matrice de convertion

```
//Load the image
src = imread("results/imageResult.png", 1);
// Check for invalid input
if(! src.data )  {
    //cout <<  "Could not open or find the image" << std::endl ;
    PRINT_ERROR("Could not open or find the image");
    exit(-1);
}
/* Initialize the convertion matrix with the same type
 * and size as input source
 */
autopilotArgs.convertionMatrix =
    Mat::zeros(src.rows, src.cols, src.type());
// Calculates the 3 x 3 matrix of a perspective transform
autopilotArgs.convertionMatrix =
```

```
getPerspectiveTransform(cameraQuad, worldQuad);
```

Finalement, la matrice de convertion obtenue est la suivante :
ConvertionMatrix =

$$\begin{bmatrix} 2.335766423357655 & 1.360865180909033 & -193.8686131386859 \\ -3.663735981263017e-15 & 2.82423468679748 & 9.947598300641403e-14 \\ -2.211772431870429e-17 & 0.007600977861656174 & 1 \end{bmatrix}$$

Convertion des coordonnées des plantons de l'image avec distorsion(pixels) en coordonnées des plantons sur l'image sans distorsion(pixels) : Cette étape intermédiaire est cruciale. En effet, c'est seulement après s'être débarassé de la vision en perspective que l'on peut avoir une relation entre l'image et le sol(terre). Une fois la matrice de convertion obtenue, pour passer d'une coordonnée d'un pixel de l'image capturée en une coordonnée d'un pixel de l'image transformée, il faut appliquer l'équation (5.1). Par exemple pour le pixel de coordonnée(318,240) dans l'image source, on obtient le pixel de coordonnée (310,240) par la matrice M de convertion. On peut aussi effectuer le calcul en sens inverse, autrement dit, passer de l'image d'une coordonnées dans l'image sans distorsion pour retrouver son image sur l'image avec distorsion.

Au niveau du logiciel, la fonction map3DTo2DCoordinate() permet d'effectuer ce calcul. Elle prend en paramètre la position du planton détecté sur l'image avec distorsion et retourne la position du planton sur l'image sans distorsion.

Listing 5.5 – Convertion intermédiaire

```
 /*
 * map3DTo2DCoordinate
 * convert 3D coordinate to 2D coordinate
 * with the perspective transform matrix.
 *
 * detectedObj : 3D coordinate from image camera
 * row : X axis coordinate get by convertion (in pixel)
 * col : Y axis coordinate get by convertion (in pixel)
*/
void map3DTo2DCoordinate(Mat mapMatrix, Object detectedObj,
                                        double* row, double* col)
{

  /* Calculates the equation to get the image
   * of the image coordinates so that:
   * (t*x', t*y', t) = mapMatrix*(x, y,1)
   */
  double _t = mapMatrix.at<double>(2,0)*detectedObj.pos.x
                        + mapMatrix.at<double>(2,1)*detectedObj.pos.y
                        + mapMatrix.at<double>(2,2);
```

```
  double _x = (mapMatrix.at<double>(0,0)*detectedObj.pos.x
                        + mapMatrix.at<double>(0,1)*detectedObj.pos.y
                        + mapMatrix.at<double>(0,2)) / _t;
  double _y = (mapMatrix.at<double>(1,0)*detectedObj.pos.x
                        + mapMatrix.at<double>(1,1)*detectedObj.pos.y
                        + mapMatrix.at<double>(1,2)) / _t;

  *row = _x;
  *col = _y;

  return;
}
```

Rapport pixels et millimètres : Une fois la caméra calibrée, pour savoir le rapport qui existe entre les pixels et les millimères est connu, il faut se référer à la formule (5.2). On peut aussi passer des millimètres aux pixels. N'importe quel point pris dans le repère des pixels trouve son image dans le repère des millimètres. Il faut préciser que les deux repère ont leurs origines en haut à gauche, principe standard dans le traitement d'image. La fonction map2DToGroundCoordinate() a été définie pour à cet effet.

Listing 5.6 – Convertion finale

```
/*
 * map2DToGroundCoordinate
 * convert 2D coordinate [pixel] to plan coordinate [mm]
 *
 * row : 2D coordinate width axis
 * col : 2D coordinate height axis
 * x : plan coordinate
 * y : plan coordinate
*/
void map2DToGroundCoordinate(const double row, const double col,
                                                    double* x, double* y)
{
  // xgroung = xpixel / WIDTH * GROUND_WIDTH
  // ygroung = ypixel / HEIGHT * GROUND_HEIGHT
  *x = row / ROWS * GROUND_WIDTH;
  *y = col / COLS * GROUND_HEIGHT;
  return;
}
```

ROWS et COLS représentent respectivement la largeur et la hauteur de l'image. Tandis que GROUNDWIDTH et GROUNDHEIGHT représentent respectivement la largeur et la hauteur du champ.

Faire avancer le bras au-dessus du planton : Pour faire avancer le planton au-dessus du planton détecté, il faut calculer la consigne en mm du bras. Cette consigne, que nous avons appelé deltaY représente la distance entre la position du robot et la position du palnton détecté. Notons que cette distance n'est pas seulement en Y mais aussi en X. Seulement, l'axe X du bras n'a pas encore été calibré.

Listing 5.7 – Distance planton et robot

```
/* calculate delta.
 * Because of the image acquisition latency,
 * we remove some offset
 */
deltaY = ROBOT_Y
       - groundY
       - speed * TIME_PRINT_IMAGE_LATENCE;
```

Il est important de souligner qu'il s'est posé le problème de latence d'affichage de l'image à l'écran. Le déplacement du bras vers le planton à arroser pourrait donc ne pas être très précisément au-dessus du planton. Pour corriger ce défaut, il a fallu calculer un offset et d'ajuster la valeur de delta. Cet offset est calculée au moyen de la vitesse du robot et de la valeur du temps de latence d'affichage de l'image à l'écran. On sait que :

$$distance = vitesse \times temps \tag{5.7}$$

Cette distance représente l'offset recherché. Il est ensuite soustrait à la valeur de delta de base (position du planton soustrait à la position du bras). D'ou la formule des Listings (5.7) et (5.8)

5.2.2 Synchronisation du robot avec le bras :

La synchronisation du bras avec le robot implique de faire reculer le robot selon la vitesse d'avance du robot.

Faire reculer le bras pour le maintenir au-dessus du planton : Puisque le robot continue d'avancer alors que l'arrosage est en cours, il a fallu faire reculer le bras d'un certain delta pour le maintenir au-dessus du planton pendant l'arrosage. Ce dont nous disposons, c'est la position du robot, la position du prochain planton et la distance entre deux plantons consécutifs. Alors pour calculer ce delta, il faut additionner à la position position du prochain planton la distance entre les deux plantons consécutifs et soustraire à ça la position initiale du bras. Concrètement, la formule a été appliquée dans le programme de manière suivante :

Listing 5.8 – Distance planton et robot

```
/* calculate delta.
 * Because of the image acquisition latency,
 * we remove some offset
 */
deltaY =  groundY
       + MAX_PLANTS_GAP_Y
```

```
- ROBOT_Y
+ speed * TIME_PRINT_IMAGE_LATENCE;
```

5.3 Arrosage des plantons

Cette fonctionnalité revient à effectuer l'activation et/ou le désactivation de l'arrosage (activation de la LED et Désactivation de la LED). Rappelons que pour le futur projet, cette gestion de la LED sera remplacée par la commande d'une vanne.

Mais pour le présent projet, il a fallu faire une mise à jour de l'état de la LED pendant (activé) et après (désactivé) la durée d'arrosage. La fonction updateLED a été implémentée pour permettre au thread autopilote de commander d'enclencher ou de désenclencher l'arrosage.

Listing 5.9 – Mise à jour de l'arrosage

```
if(ledState) {
    gpioledOn();

 } else {
    gpioledOff();
 }
```

Pour contrôler la durée d'arrosage, un compteur de durée a été implémenté à partir de l'horloge interne du microcontrôleur. Il part de 0 et dès qu'il atteint la valeur de la durée d'arrosage, l'arrosage est désenclenché (désactivation de la LED) et le compteur reinitialisé.

Listing 5.10 – Mise à jour de l'arrosage

```
// enable watering
ledState = 1;
updateLed(ledState, i2cMutex);
// set a counter to count watering duration
if( clock_gettime(CLOCK_REALTIME, &start) == -1 )
{
    perror( "error: clock gettime" );
    exit( EXIT_FAILURE );
}

...

if( clock_gettime( CLOCK_REALTIME, &stop) == -1 )
{
    perror( "error: clock gettime" );
    exit( EXIT_FAILURE );
}
```

```
res = diff(start, stop);
// calculates duration in seconds
duration += res.tv_sec + 1e-9*res.tv_nsec;

if (duration >= opts->wateringDuration)
{ // is watering duration ended?
    // disable watering
    ledState = 0;
    updateLed(ledState, i2cMutex);
    // disable counter duration
    duration = 0.0;
 }
 ...
```

5.4 Détection du début d'une ligne

Pour détecter le début d'une ligne, une fonction de détection des contours a été écrite. Elle détecte un si un triangle existe dans l'image au début d'une ligne de plantons. Cette fonction lit une image, charge l'ensemble des contours qui existe dans l'image. Un contour est stocké comme un vecteur. Elle parcourt l'ensemble des contours et vérifie s'il existe un contour formé de trois points (sommets du triangle). En effet, un triangle est cette forme géométrique formée de trois sommets(points). Opencv met à disposition une fonction pour celà, la fonction findContours() et plusieurs exemples de codes en guise d'exemple d'utilisation [**?**]. En date d'écriture de ce rapport, cette fonction a été implémentée et les centres de gravités aussi. Il reste à calculer la moyenne des coordonnées (axe X) des centres de gravité des triangles.

5.5 Difficultés rencontrées

Une panne est survenue lors des tests du système d'arrosage sur le robot. De nouveau la carte Raspberry Pi rebootait toute seule. Il a été vérifié au multimètre toutes les tensions d'entrée (5 V et 12 V), tout avait l'air normal. Nous avons pensé à changer le câble d'alimentation, et là encore le problème persistait. Puis, il nous a semblé bon d'isoler les tests, d'abord les moteurs des roues, ensuite les moteurs du bras d'arrosage. Ainsi nous avons pu nous rendre compte que Chaque fois que le programme sollicitait le bus I2C pour commander le moteur du bras, la carte rebootait exactement à ce moment là. En regardant plus concrètement, il se passait que le bras en essayant d'avancer, coinçait au niveau de la grande poulie. En fait, cette panne était du au serre-fil qui avait été mis pour soutenir la LED nouvellement installée sur le bras d'arrosage. Le serre-fil a été coupé et remplacé par du scotch et le problème s'est résolu.

Chapitre 6

Configuration de la durée d'arrosage

Qt est un ensemble de librairies, basé sur le langage C++, qui fournit un environnement de développement complet, principalement pour concevoir aisément des interfaces graphiques [**?**]. Il a été choisi comme outil pour la création de l'interface graphique qui permettra la configuration des paramètres de l'arrosage des plantons. Les raisons étant qu'il est basé sur le langage C++, et donc compatible avec notre projet. Il possede aussi une documentation qui est assez fournie avec des exemples de projets à portée. Il prend en charge l'exécution et le déploiement d'applications Qt conçues pour différentes plates-formes cibles à l'aide de différents compilateurs, débuggueurs.

6.1 Cross-compilation et déploiement de Qt pour la Raspberry Pi

Cross-compiler signifie pour un compilateur d'être capable de créer du code exécutable pour une plateforme cible autre que celle sur laquelle le compilateur est exécuté [**?**]. Un avantage c'est que le code est portable sur plusieurs plateformes différentes. Il suffit de renseigner la toolchain apporpriée. Une toolchain est un ensemble d'outils constitué de compilateur, linker, tout ce qui est nécessaire afin de produire un exécutable pour une cible donnée.

Plusieurs étapes sont nécessaires pour cross-compiler Qt et le déployer pour la Raspberry Pi. Consulter le site officiel [**?**].

6.2 Difficultés rencontrées

Il nous a fallu jusqu'à deux jours pour essayer de mettre en place l'environnement Qt pour la Raspberry Pi. Les étapes ont été suivis jusqu'à la numéro 10. Une erreur s'est produite portant sur les tests de fonctionnalité de OpenGL. Une précondition qui ne serait pas respectée pour terminer correctement l'intallation d'OpenGL.

Il a été vérifié que la librairie ait bien été installée. Pour ce faire, il a fallu regarder au niveau de fichiers "includes" et "librairies", ce qui s'avérait juste. Pourtant OpenGL est resté activé mais pas installé si on s'en tient à la liste des configurations suivante.

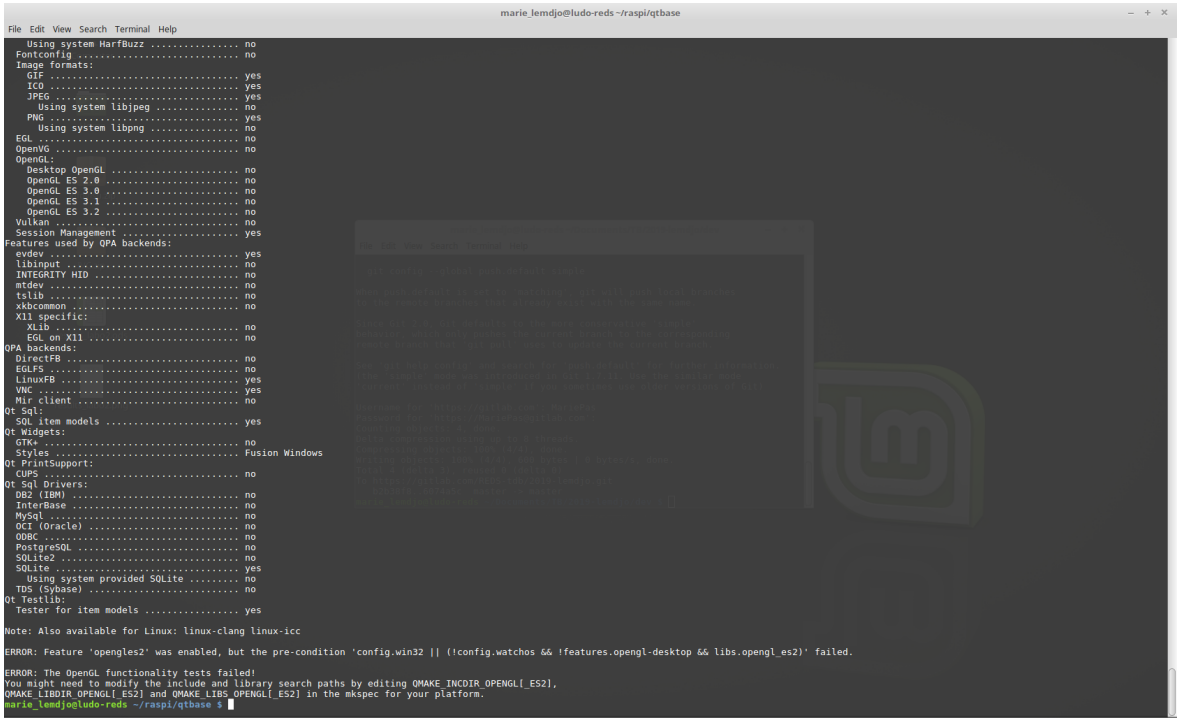

FIGURE 6.1 – Erreur de configuration d'OpenGL

Pluisieurs autres hypothèses ont été posées sur les raisons de ce disfonctionnement. La version actuelle de l'OS de la Raspberry Pi qui serait obsolète ? Le librairies seraient incompatible avec la carte Raspberry Pi 3 B+ ?

Après plusieurs tentatives (mise à jour de Raspbian par exemple), en regardant dans les forums et en discutant avec un des assistants de l'institut REDS, triste a été de constaté que les recherches sont restées infructueuses. Il a donc fallu trouver rapidement une autre solution.

6.3 Solutions proposées

6.3.1 En ligne de commande

Ls solution qui a été implémentée est d'ajouter une option au programme. Pour configurer la durée d'arrosage, il faut préciser avec l'option " -D " suivie de la valeur de cette durée au démarrage du programme.

6.3.2 Un autre framework - GTK

GTK est une alternative à Qt. C'est un set de librairies Open-source pour la création d'interfaces graphiques. Il a par exemple été utilisé pour gérer les composants (les menus, les boutons et les champs d'entrée) de l'interface graphique pour l'éditeur d'image GIMP de Linux.

Pour pouvoir l'utiliser, il est nécessaire d'installer le package "libgtk-3-dev". Il a une sysntaxe assez particulière. Il sera possible dans les prochains jours de le tester et voir les possibilités de l'intégrer à ce projet.

Chapitre 7

Tests et résultats

7.1 Tests et résultats

Ce chapitre présente les tests qui ont été effectués et les résultats obtenus.

7.1.1 Calibration des moteurs

Calibrage des roues : Les tests ont pu être fait de trois manières.

- Au multimètre : On arrive à avoir plus précisément les valeurs identiques de tensions d'entrée des deux moteurs.
- Sur la planche à essaie avec du papier posé sur les roues au même point. : on observe que les roues tournent plus précisément à la meme vitesse
- Sur un sol lisse dans le couloir de l'école : le robot suit de manière plus précise une ligne droite

Conclusion des tests : Le calibrage des roues a permis de corriger la trajectoire du robot. L'angle d'inclinaison a pu être diminuée au moins de moitié. Et la vitesse minimale du robot a pu être diminuée de peu (valeur de PWM légèrement supérieure à 1.38 ms mais garde les moteurs alimentés). Notons qu'il serait impossible d'aller plus haut que ça. Parce que pour une valeur de 1.48 ms les moteurs sont au repos.

Pour ce qui est du bras, il possible d'acquérir la valeur de PWM correspondant à une distance en mm donnée avec un offset de 3 mm. Sachant que la largeur du planton sur le sol est de 4 à 5 mm, cela n'est pas trop contraignant.

7.1.2 Convertion des coordonnées de l'image en coordonnées sur le sol

Conditions des tests : Les tests ont été effectués dans les conditions montrées ci-dessous.
Image : (Figure 5.3)
Résolution de l'image : 320x240
Angle de la caméra : possibilité de voir jusqu'à quatre plantons sur une même ligne
Origine du bras : (370, 1240)
Temps moyen des opérations pour le système d'arrosage est de : 0.45 secondes
Robot statique

Test1 : Comparaison Coordonnées estimées et coordonnées réelles des plantons

num	avant calibration [px]	après calibration [px]	calculées [mm]	réelles [mm]
1	(169, 45)	(195, 95)	(415, 458)	(390, 420)
2	(107, 46)	(88, 96)	(187, 456)	(160, 420)
3	(234, 49)	(306, 101)	(649, 487)	(620, 420)
4	(83, 98)	(76, 159)	(162, 767)	(160, 710)
5	(167, 97)	(189, 158)	(401, 762)	(380, 750)
6	(251, 106)	(297, 166)	(632, 801)	(600, 760)
7	(48, 204)	(77, 226)	(163, 1091)	(160, 1040)
8	(161, 207)	(180, 227)	(383, 1098)	(380, 1040)
9	(297, 222)	(298, 233)	(634, 1127)	(600, 1050)

TABLE 7.1 – Résultat1 : Tableau de comparaison des coordonnées estimées avec les coordonnées réelles

Test2 : Comparaison Coordonnées estimés et coordonnées réelles des plantons

num	avant calibration [px]	après calibration [px]	calculées [mm]	réelles [mm]
2	(107, 46)	(88, 96)	(187, 456)	(160, 420)
1	(169, 45)	(195, 95)	(415, 458)	(390, 420)
3	(234, 48)	(306, 101)	(649, 487)	(620, 420)
4	(84, 98)	(76, 159)	(162, 767)	(160, 710)
5	(169, 97)	(189, 158)	(401, 762)	(380, 750)
6	(252, 106)	(297, 166)	(632, 801)	(600, 760)
7	(48, 204)	(77, 226)	(163, 1091)	(160, 1040)
8	(161, 207)	(180, 227)	(383, 1098)	(380, 1040)
9	(298, 223)	(298, 233)	(634, 1127)	(600, 1050)

TABLE 7.2 – Résultat2 : Tableau de comparaison des coordonnées estimées avec les coordonnées réelles

Conclusion des tests : On remarque par rapport au premier test que la première ligne et la deuxième ligne se sont interchangées. Ceci n'est pas bien grave puisque ce qui nous intéresse c'est le dernier planton et lui, il est fixe et correspond toujours au planton le plus proche du robot. On observe aussi de légères variations au niveau des valeurs en pixel, de l'ordre de 1 ou 2 unités. Ceci est du au fait que les coordonnées du centre du planton varient de très peu sur l'image. On sais aussi que le planton a une épaisseur de 4 à 5 mm, donc on peut vivre avec cette variation.

7.1.3 Arrosage des plantons détectés

En date d'écriture de ce rapport, le logiciel a été testé avec l'utilisation d'un seul thread qui gère à la fois la détection, le suivi d'une ligne de plantons et et l'arrosage.

Il en ressort que l'arrosage se fait, mais le robot arrose sur une ligne de 7 plantons détectés seulement 5, soit 71%. De plus, il corrige d'un cran en plus sa trajectoire par rapport à ce qu'avait mon prédécesseur. Néanmoins on arrive à observer l'arrosage.

7.2 Améliorations possibles

Comme le témoigne la section (4.1.3), nous étions partie sur une solution d'implémentation qui utilise un seul thread qui s'occupe à la fois de la détection des plantons, le suivi de la ligne de plantons et l'arrosage. Mais fort a été de constater que cette solution présentait plusieurs inconvenients. Outre le code surchargé et donc pas lisible, le rendu au final n'est pas pouvait être encore amélioré. En effet, le système était lent, alors pendant le suivi de la ligne de plantons par le robot, il effectue un cran en plus une correction de sa trajectoire. De plus, pas tous les plantons sont arrosés (soit 71%).

7.2.1 Structure logicielle adaptée

Il a été décidé malgré le peu de temps qui restait, d'explorer une solution utilisant deux threads supplémentaires. Le thread existant se charge de la détection des plantons et celui-ci mémorise la position des plantons détectés dans une variable globale partagée par tous les threads. Le deuxième s'occupe tout seul du suivi d'une ligne de plantons. Et le troisième thread lui, se charge de l'arrosage. Ces deux derniers accèderons en lecture et de manière concurrente à la variable mémorisée (partagée) pour acquérir les coordonnées des plantons détectés par le premier. Biensûr le Main, qui est le thread principal garde sa fonction de gestion et de traitement des options du programme et de la commande manuelle. Par rapport à la précédente structure, les tâches ont été séparées et réorganisées. L'avantage de cette solution c'est que le parallélisme est mieux exploité et le système est plus rapide. En effet, la Raspberry possède 4 coeurs de processeurs. Le code est aussi plus lisible. Le reste de la structure est inchangé.

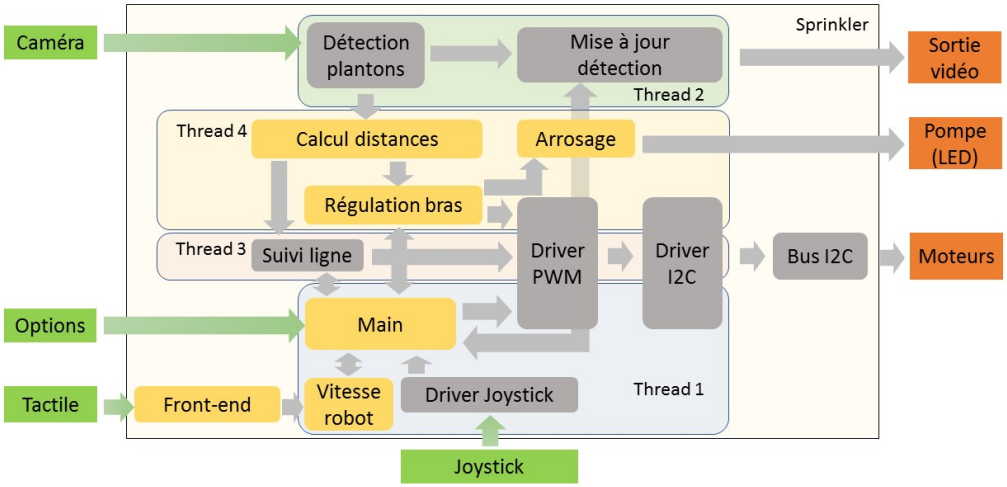

FIGURE 7.1 – Nouvelle structure logicielle du robot d'arrosage

Au niveau du code, il a été créé deux threads supplémentaires comme le montre les extraits de code suivants.

Le mécanisme d'exclusion mutuelle a été mis en place. C'est un mécanisme pour garantir que deux threads n'effectuent des actions simultanément sur la même ressource partagée. Rapellons qu'un mutex est une sorte de drapeau. Lorsqu'un thread a besoin de ressource partagée, il interroge ce drapeau. S'il est libre, alors le thread change son état de libre à

occupé et entre en possession de la ressource. Dans ce cas, aucun autre thread ne peut avoir accès à la ressource jusqu'à ce que le mutex soit relâché.

Il est aussi à noter que les variables globales ont été déclarées avec le mot clé "extern" afin qu'elles soient définies qu'une seule fois et à un seul endroit pour tous les fichiers qui l'utilisent.

Listing 7.1 – Gestion des nouveaux threads

```
// thread declaration
pthread_t wateringThread;
pthread_t followLineThread;
...
// create and start threads
if(pthread_create(&followLineThread,
                  NULL, followLine, &autopilotArgs)) {
    PRINT_ERROR("Can't create follow line thread\n");
}

if(pthread_create(&wateringThread,
                  NULL, watering, &autopilotArgs)) {
  PRINT_ERROR("Can't create watering thread\n");
}
...

 // terminate threads
 if(pthread_join(followLineThread, NULL))
 {
     perror("pthread_join");
 }
 PRINT_GENERAL_DEBUG("followLine thread terminate\n");

 if(pthread_join(wateringThread, NULL))
 {
     perror("pthread_join");
 }
 PRINT_GENERAL_DEBUG("watering autoPilot thread terminate\n");
```

Listing 7.2 – Création du mutex pour garantir l'exclusion mutuelle des variables partagées

```
// mutex for global variables
pthread_mutex_t globalVariableMutex;

// Init Mutex
pthread_mutex_init(&globalVariableMutex, NULL);
PRINT_THREADS_DEBUG("globalVariableMutex init\n");
```

```
    ...

    // destroy mutex
    pthread_mutex_destroy(&globalVariableMutex);
```

Listing 7.3 – Création des variables partagées et des fonctions pour les tâches

```
// Objects
extern Object obj[N_OBJ_MAX];
extern size_t nObj;

/* Functions Headers */
void* followLine(void* args);
void* watering(void* args);
```

7.2.2 Difficultés rencontrées

Le programme avec la nouvelle structure est fonctionnel à 90%. En effet, tous les threads se lancent et se terminent. Chaque thread effectue sa tâche. L'exclusion mutuel se fait correctement et les valeurs des variables globales sont intègrent durant l'exécusion. Il ne reste que quelques heures de travail pour que l'arrosage soit précis.

Cependant, les difficultés n'ont pas manquées. La mise en place de cette solution a nécessité une réorganisation du code. Il fallait faire attention de bien manier tous les éléments cités plus haut ; threads, accès aux ressources partagées. Il a fallu activer l'option debug pour les threads afin de se rassurer que tous les threads se lançaient et se terminaient correctement, ainsi que tous les mutex acquis étaient relâchés.

Chapitre 8

Conclusion

8.1 Synthèse

Le but de ce rapport était de montrer le travail effectué dans le cadre du projet de fin de Bachelor pour la conception et la réalisation d'un démonstrateur pour robot d'arrosage. Au début de ce projet, il m'a été remis les travaux effectués par l'étudiant Ludovic Richard. Ses travaux ont permis de réaliser les fonctions de détection des plantons et de suivi d'une ligne dans le champ. Il a été question de prendre en main le projet pour la bonne marche de sa suite. Ensuite il a fallu se pencher sur l'objectif principal de ce travail, la réalisation du système d'arrosage par la commande automatique du bras d'arrosage. Par rapport au cahier des charges, les points suivant ont pu être fait :

La commande manuelle du robot arroseur : Elle comprend la commande des roues (avancer, reculer, tourner) et du bras d'arrosage (avancer, reculer) à l'aide du joystick configuré. Il est possibles de vérifier son bon fonctionnement avec le joystick .

La commande automatique du bras d'arrosage : Nous pouvons déjà procéder à l'arrosage des plantons de façon automatique. Une version single thread a été implémentée en premier. Puis l'idée est venue de le rendre plus performant en séparant mieux les tâches pour rendre l'exécution du programme encore plus parallèle et donc de le rendre plus performant. Pour celà, il a été utilisé deux threads supplémentaires dont l'un s'occupe de l'arrosage et l'autre du suivi d'une ligne de plantons. Il ne nous reste que quelques heures de travail pour des tests plus approfondis. En date d'écriture de ce rapport, il n'a pas été possible de faire toutes les vérifications pour cette version du code.

La partie front-end : Il avait été choisi d'implémenter l'interface graphique à l'aide de Qt. Cependant sa configuration s'est montré plus difficile que prévu et a pris beaucoup de temps sans que nous y parvenions effectivement. Vu le temps qui restait, nous nous avons pris la décision de laisser tomber cette ideée et d'implémenter la version en ligne de commande.

Détection du début d'une ligne de plantons : Cette fonctionnalité n'a pas pu être achevée, pour cause de la gestion de nombreux imprévus comme par exemple les pannes qui sont survenues de temps en temps. Néanmoins, une fonction permettant la détection des contours a été implémentée. L'idée était de mettre une marque (une forme géométrique comme le triangle), de détecter ce triangle et par ce triangle détecter le début de la ligne.

8.2 Futur du projet

Pour la suite du projet, il faudrait :

1. Pour des projets comme ceux-ci se procurer des alimentations pour des tests du robot en statique. ça éviterait de perdre du temps à cause des batteries qui se déchargent incessemment.
2. Calibrer l'axe X du bras d'arrosage
3. Implémenter une inetrface graphique pour rendre les tests plus pratiques, avec une autre librairie comme par exemple GTK.
4. S'il est aussi possible de se procurer un bras d'arrosage plus pratique que celui qui est actuellement là.
5. Migrer tout le projet en C++. Définir des classes C++ pour une meilleure structuration du code
6. Améliorer la détection des plantons avec un algorithme plus intelligent. Il serait aussi intéressant de prévoir des plantons de d'autres types, autres couleurs possibles.

8.3 Commentaires personnels

Le travail s'est avéré challengeant au départ. En effet, il n'est toujours pas évident de reprendre un projet initié par quelqu'un d'autre. Néanmoins, ce qui était un défi au départ est à présent une compétence nouvellement acquise. Egalement, comprendre le fonctionnement des composantes électroniques et manier les appareils de mesures sont une expérience nouvelle et enrichissante.

Même si des erreurs d'apprentissage ont été constatées tout au long du projet, par exemple dans la manière de trouver des mécanismes pour ne pas perdre du temps lors de l'implémentation, je suis assez contente d'être arrivée à prendre conscience de ces lacunes. Je suis particulièrement contente de la manière dont j'ai pu gérer la résolution des imprévues et des pannes, choses importantes pour la suite de ma carrière.

Je me réjouis d'achever ce qui reste à faire dans les prochains jours afin de voir ce projet abouti.

Bibliographie

[1] Wikipedia. Robot agricole. https://fr.wikipedia.org/wiki/Robot_agricole, 2019.

[2] I-Hsum Li Ming-Chang Chen Wei-Yen Wang Shun-Feng Su and To-Wen Lai. Mobile robot self-localization system using single webcam distance measurement technology in indoor environments. https://www.researchgate.net/profile/I_Hsum_Li/publication/, 2014.

[3] Ecorobotix SA. Ecorobotix. https://www.ecorobotix.com/fr/, 2019.

[4] Ludovic RICHARD. Schema complet de la structure du programme, 2018.

[5] Raspberry PI. Raspberry PI modèle 3b+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/, 2018.

[6] Sparkfun. Image de la raspberry 3 modèle b+. https://cdn.sparkfun.com//assets/parts/1/2/8/2/8/14643-Raspberry_Pi_3_B_-05.jpg, 2018.

[7] Adafruit. 16-CHANNEL 12-BIT PWM/SERVO DRIVER - i2c INTERFACE. https://www.adafruit.com/product/815, 2015.

[8] Logitech. Image de la webcam quickcam pro 9000 de chez logitech. https://secure.logitech.com/assets/18745/18745.png, 2007.

[9] Waveshare. 10.1inch hdmi lcd (b) (with case). https://www.waveshare.com/wiki/10.1inch_HDMI_LCD_(B)_(with_case), 2016.

[10] Waveshare. Image de l'écran 10.1inch hdmi lcd (b) (with case). https://www.waveshare.com/media/catalog/product/cache/1/image/800x800/9df78eab33525d08d6e5fb8d27136e95/1/0/10.1inch-hdmi-lcd-b-with-holder-front_1.jpg, 2016.

[11] inc parallax. HB-25 motor controller, 2014.

[12] electronic os caldas. MG955 high speed, 2014.

[13] generation robots. Image de HB-25 motor controller. https://www.generationrobots.com/en/401563-hb-25-motor-controller.html, 2014.

[14] TowerPro. Image du moteur MG955. https://www.towerpro.com.tw/product/mg995/, 2014.

[15] OpenCV. About opencv. https://opencv.org/about/, 2019.

[16] Raspberry Pi Foundation. Noobs os, 2018.

[17] Raspberry Pi Turoriel. Control your raspberry pi by using a wireless xbox 360 controller. https://tutorials-raspberrypi.com/raspberry-pi-xbox-360-controller-wireless/, 2019.

[18] Ingo Ruhnke. xboxdrv — a xbox/xbox360 gamepad driver that works in userspace. https://xboxdrv.gitlab.io/xboxdrv.html, 2011.

[19] Ingo Ruhnkee. Pilote de manette de jeu xbox / xbox360 utilisateur pour linux. https://xboxdrv.gitlab.io/, 2015.

[20] Wikipedia. Interpolation linéaire. https://fr.wikipedia.org/wiki/, 2019.

[21] Wikipedia. Homography (computer vision). https://en.wikipedia.org/wiki/, 2019.

[22] opencv dev team. Geometric image transformations. https://docs.opencv.org/2.4/, 2019.

[23] OPenCV. Structural analysis and shape descriptors. https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#findcontours, 2019.

[24] The Qt Company. Complete software development framework – qt. https://www.qt.io/what-is-qt/, 2019.

[25] Wikipedia. Cross compiler. https://en.wikipedia.org/wiki/Cross_compiler, 2019.

[26] The Qt Company. Raspberrypi2eglfs. https://wiki.qt.io/RaspberryPi2EGLFS, 2019.

Authentication

Par la présente, je soussigné, Marie Pascale Nzinke LEMDJO, déclare avoir réalisé seul ce travail et ne pas avoir utilisé d'autres sources que celles citées dans la bibliographie.

Date	Marie Pascale Nzinke LEMDJO
	Signature

Glossary

Abréviation	Terme	Définition
REDS	Reconfigurable Embedded Digital System	Institut de prestige de l'HEIG
RAZ	Remise à zéro	noeud initial du diagramme de flow.
PWM	Pulse Width Modulation	technique qui permet de générer un signal carré avec un rapport cyclique modulé en fonction d'un signal de commande voir ici
OpenCV	Open Source Computer Vision	Librairie utilisée pour le traitement d'images.

Table des figures

Liste des tableaux