RESUMEN

El término robótica procede de la palabra robot. La robótica es, por lo tanto, la ciencia o rama de la ciencia que se ocupa del estudio, desarrollo y aplicaciones de los robots. De esta definición podemos concluir que en la robótica se aúnan para un mismo fin varias disciplinas confluyentes, pero diferentes, como puede ser la Mecánica, la Electrónica, la Automática, la Informática, etc.

Existen varias definiciones del término robot algunas de ellas se acercan más a la ciencia ficción que al mundo real. Una definición del robot actual podría ser la de máquina autónoma multifuncional y reprogramable diseñada para realizar funciones que normalmente son ejecutadas por el hombre. Dentro de esta definición caben diferentes tipos de clasificación de un robot pero la que interesa destacar para este proyecto es la de robot móvil.

Los robots móviles están provistos de patas, ruedas u orugas que los capacitan para desplazarse de acuerdo a su programación. Elaboran la información que reciben a través de sus propios sistemas de sensores y se emplean en determinado tipo de instalaciones industriales, sobre todo para el transporte de mercancías en cadenas de producción y almacenes. También se utilizan robots de este tipo para la investigación en lugares de difícil acceso o muy distantes,

como es el caso de la exploración espacial y las investigaciones o rescates submarinos.

Pekee es un robot móvil desarrollado por la empresa Wany Robotics capaz de realizar movimientos y de tener un comportamiento autónomo. La variedad de sensores y dispositivos que incluye así como la posibilidad de comunicación inalámbrica mediante un módulo wireless y la recepción de imágenes mediante la cámara incorporada al módulo PC permite un gran abanico de posibles aplicaciones.

En el capítulo I se hace una descripción general del robot tanto a nivel hardware explicando sus características más importantes como a nivel software explicando las funciones de bajo nivel utilizadas y las diferentes librerías aportadas por Wany para facilitar la programación de aplicaciones sobre el robot.

Pekee es un robot propulsado por dos ruedas motrices independientes delante y sostenido por una rueda "loca" detrás. Esta disposición ofrece una gran movilidad permitiendo, por ejemplo, de hacer un giro completo sobre su eje.

El desplazamiento del robot Pekee es totalmente autónomo gracias a sus 15 sensores de distancia infrarrojos repartidos a su alrededor.

En la parte superior del chasis del robot se encuentran 3 emplazamientos directamente accesibles que se utilizan para conectar los módulos o tarjetas hijas suministradas por la empresa Wany o diseñadas por el propio usuario.

A nivel hardware el robot incluye las siguientes características:

- 15 sensores de distancia infrarrojos
- 2 odómetros (180 impulsos por giro de rueda)
- 1 detector de choque
- 2 girómetros
- 1 sensor de luz
- 2 sensores de temperatura
- Baterías: 2 x 12V (NiMH).
- 1 buzzer a frecuencia modulable
- 1 enlace infrarrojo para una comunicación entre robots y periféricos
- 1 enlace serie infrarrojo para la transferencia de datos entre Pekee y una estación de recarga o un PC
- 1 microcontrolador Mitsubishi 16MHz (16 bits), con 256KB de Flash ROM y 20KB de RAM.
- 4 conectores para accesorios con bus I2C.
- 5 conectores para accesorios con bus OPP

En este proyecto se han utilizado dos módulos o tarjetas hijas. El módulo PC embebido y el módulo de comunicación wireless.

El módulo PC embebido esta formado por un procesador compatible Intel 486 , memoria RAM, disco duro, salida para monitor, entradas para ratón y teclado y tarjeta de Red Ethernet. También incorpora sus propias baterías que le permiten funcionar autónomamente sin tener que estar conectado al robot.

Además incorpora una tarjeta de adquisición de video y una cámara. A través del bus OPP, la tarjeta se puede comunicar con el robot o cualquier otro módulo conectado.

Pekee incorpora un módulo Wireless que añade capacidad de comunicación inalámbrica al robot. En el capítulo II se hace una descripción del Standard 802.11 en que se basa la comunicación Wireless así como de sus características principales y sus modos de funcionamiento.

Ya en el capítulo III se hace una descripción funcional de los programas desarrollados. Para facilitar la comprensión se explican sus características y funciones usando esquemas de bloques y diagramas de flujo. Por lo tanto, el código generado en VC++ no se incluirá en este capítulo sino que está disponible en un anexo al proyecto.

PekeeControl es el nombre de la interfaz gráfica desarrollada en este proyecto. Permite el seguimiento y control del robot a distancia. Además, también se ha creado el programa *ServerCapture* que realiza las funciones de captura de

imágenes y servidor de video y que se ejecuta en el módulo PC de Pekee. Los dos programas se han desarrollado utilizando el lenguaje de programación Visual C++ 6.0 y las Microsoft Foundation Classes (MFC).

Visual C++ es un lenguaje de programación orientado a objetos y a entornos gráficos como Windows. Mientras que en lenguaje C la unidad de programación es la función, en C++ es la clase. Los objetos son creados a partir de estas clases.

MFC son un conjunto de clases desarrolladas por Microsoft y que facilitan enormemente la programación de una aplicación en Visual C++.

Para el desarrollo de *PekeeControl* y *ServerCapture* se han creado clases nuevas y se han utilizado clases de MFC o generadas por la empresa Wany.

Además de estos dos programas también se utiliza el programa OppToNetwork desarrollado por Wany y que permite traducir los mensajes enviados vía TCP/IP a mensajes OPP para que sean entendidos por Pekee. Este programa se ejecuta junto con *ServerCapture* en el módulo PC del robot.

PekeeControl se ejecuta en el ordenador del usuario y mediante una ventana se pueden observar los parámetros básicos del robot móvil (sensores de distancia, de luz, de temperatura,...) y de las imágenes de video. Además incorpora tres funciones que modifican el comportamiento del robot:

- Función Keyboard: Esta función permite al usuario controlar el movimiento del robot desde el teclado de su ordenador.

- Función Reflex: Esta función permite a Pekee el desplazamiento autónomo así como evitar los obstáculos que se encuentre en su camino.

- Función Recognize: Esta función permite a Pekee reconocer objetos situados delante de la cámara que tengan la característica que sean de color rojo.

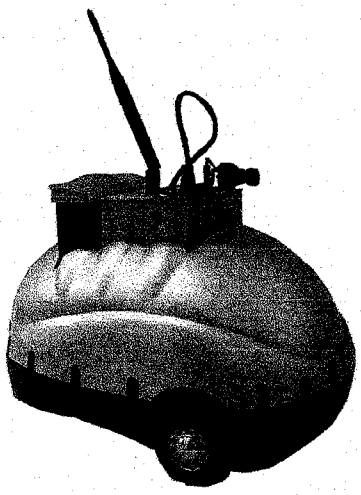
ECOLE CENTRALE D'ELECTRONIQUE

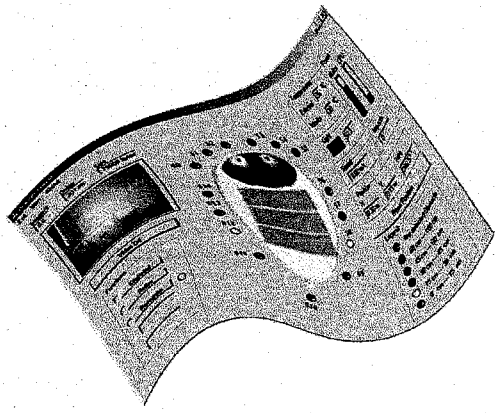
ece

GROUPE ECE

Projet de fin d'études

Développement d'une interface graphique pour le suivi et le contrôle à distance du robot Pekee





Sergio López
Erasmus 02-03

SOMMAIRE

INTRODUCTION

Le terme robotique trouve ses racines dans le mot robot. La robotique est donc la science ou une des branches de la science qui s'occupe de l'étude, du déroulement et des applications des robots. De cette définition nous pouvons en déduire que la robotique touche à plusieurs domaines tels que la Mécanique, l'Électronique, l'Automatique, l'Informatique, etc.

Ils existent plusieurs définitions du terme robot dont certaines s'approchent plus à la science fiction qu'au monde réel. Une définition du robot actuel pourrait être celle d'une machine autonome multifonctionnelle et reprogrammable destinée à réaliser des fonctions qui sont normalement exécutées par l'homme. Dans cette définition, on trouve différents types de robots. Par exemple, notre robot est du type robot mobile.

Les robots mobiles sont fournis de pattes, des roues ou des chenilles qui les permettent se déplacer suivant leur programme. Ce type de robots élaborent l'information qu'ils reçoivent à travers leurs propres systèmes de capteurs et sont souvent utilisés dans des installations industrielles, comme par exemple le transport de marchandises dans les chaînes de production et les dépôts. De même, ces robots sont aussi utilisés dans l'investigation des lieux d'accès difficile ou très distants, comme c'est le cas dans l'exploration spatiale et dans les recherches ou les sauvetages sous marins.

Pekee est un robot mobile développé par l'entreprise Wany Robotics capable de réaliser des mouvements et d'avoir un comportement autonome. La variété de capteurs et des dispositifs inclus ainsi que la possibilité de communiquer sans fils moyennant un module wireless et la réception d'images moyennant la camera incorporée au module PC embarqué permettent un grand éventail d'applications possibles.

PekeeControl et *ServerCapture* sont les programmes qui composent l'interface graphique développée dans ce projet. Ils permettent le suivi et le contrôle du robot à distance. *ServerCapture* est un programme serveur qui s'exécute dans le module PC embarqué de Pekee et qui se charge de la transmission des images capturées par la camera. *PekeeControl* s'exécute dans l'ordinateur de l'usager et moyennant une boîte de dialogue on peut observer les paramètres de base du robot mobile (capteurs de distance, de lumière, de température,...) et les images de la vidéo. En outre, il incorpore trois fonctions qui modifient le comportement du robot:

- Fonction Keyboard: Cette fonction permet à l'usager de contrôler les mouvements du robot depuis le clavier de son ordinateur.

- Fonction Reflex: Cette fonction permet à Pekee de se déplacer de manière autonome en évitant les obstacles qu'il rencontre sur son chemin.

- Fonction Recognize: Cette fonction permet à Pekee de reconnaître des objets situés devant la camera qui ont la caractéristique d'être rouges.

Le chapitre I décrit de manière générale le robot tant au niveau hardware, expliquant ses caractéristiques les plus importantes, qu'au niveau software, expliquant les fonctions de niveau bas utilisées et les différentes librairies fournies par Wany pour faciliter la programmation d'applications sur le robot.

Pekee incorpore un module Wireless qui ajoute la capacité de communication sans fils au robot. Le chapitre II décrit le Standard 802.11 qui est la norme utilisée pour la communication Wireless ainsi que les caractéristiques principales et les modes de fonctionnement.

Le chapitre III décrit de manière fonctionnelle les programmes développés. Pour faciliter la compréhension, l'explication des caractéristiques et des fonctions est basée sur des schémas blocs et des organigrammes. Le chapitre III ne comporte donc aucun code VC++, mais on pourra trouver le code des programmes dans l'annexe.

CHAPITRE I

LE ROBOT PEKEE

I.1 Description générale.

Le robot Pekee est un système complètement autonome et programmable fabriqué par l'entreprise Wany Robotics avec une architecture extensible basée sur le bus OPP (breveté par eux).

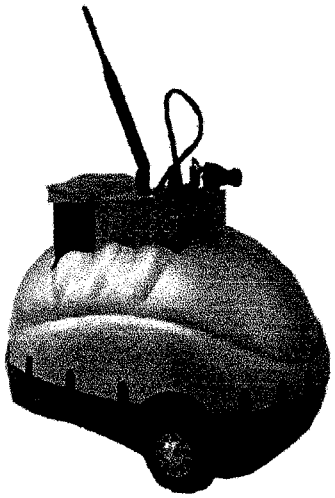


Fig. I. 1 Photo Pekee

C'est un robot propulsé par deux roues motrices indépendantes à l'avant et soutenu par une roue folle à l'arrière. Cette disposition lui offre une très grande mobilité, lui permettant par exemple de faire un demi-tour tour sur place.

Le déplacement du robot Pekee est entièrement autonome grâce aux capteurs de distance répartis tout autour et dont la technologie novatrice a été breveté par Wany.

Le châssis de Pekee est composé d'une structure autoporteuse en ABS, constituée de deux parties rigides assemblées, et d'un élément de carrosserie interchangeable souple.

On trouve ainsi trois emplacements directement accessibles sur la partie supérieure de l'unité qui sont utilisés pour connecter les cartouches fournies par Wany. Si on retire le châssis supérieure, on accède à deux emplacements supplémentaires, un à l'avant et un à l'arrière.

Il y a également quatre connecteurs pour de petits périphériques compatibles I2C, et un connecteur qui permet, grâce à un accessoire prévu à cet effet, de reprogrammer Pekee.

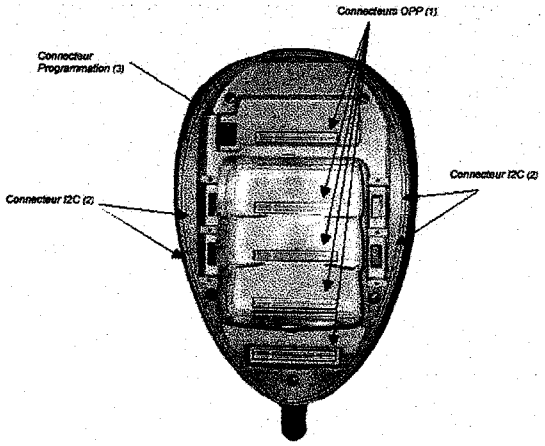


Fig. I. 2 Photo connecteurs

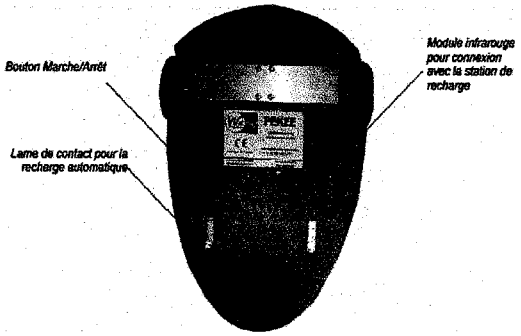


Fig. I. 3 Vue de dessous

I.1.1 Caractéristiques générales

Dimensions

- Longueur hors tout : 40 cm.
- Longueur sans la queue : 37 cm.
- Largeur hors tout : 25.5 cm.
- Largeur entre roues motrices : 22.5 cm.
- Hauteur sans accessoires : 21 cm.

Poids

- 2.9 Kg sans accessoires.

Vitesse maximale

- 1 mètre/seconde.

Autonomie

- 15 heures maximum avec des déplacements limités (et sans accessoires).

- Environ 1 heure 30 minutes avec déplacements ininterrompus.

Rotation

- 360 ° dans un cercle de 70 cm de diamètre.

I.2 Description Hardware

Le robot Pekee incorpore une grande variété de capteurs, détecteurs et tout une série de dispositifs qui lui permet une large fonctionnalité dans les différentes applications :

- 15 télémètres infrarouges (10 mesures / seconde en utilisant la technologie brevetée Wany).
- 2 odomètres (180 impulsions par tour de roue).
- 1 détecteur de choc.
- 2 gyromètres (axes lacet et tangage).
- 1 capteur de lumière.
- 2 capteurs de température.
- Batteries d'accumulateurs : 2 x 12V (NiMH).
- 1 buzzer à fréquence modulable.
- 1 voyant de charge lente (LED rouge).
- 1 voyant de fonctionnement (LED rouge).
- 1 liaison infrarouge pour une communication entre robots et périphériques.
- 1 liaison série infrarouge pour le transfert des données entre Pekee et sa station de recharge ou un PC.
- 1 microcontrôleur Mitsubishi 16MHz (16 bits), avec 256Ko de Flash ROM et 20Ko de RAM.
- 4 emplacements pour accessoires avec bus I2C.
- 5 emplacements pour accessoires bus OPP avec transfert multi-maîtres vers multi-esclaves haut débit.

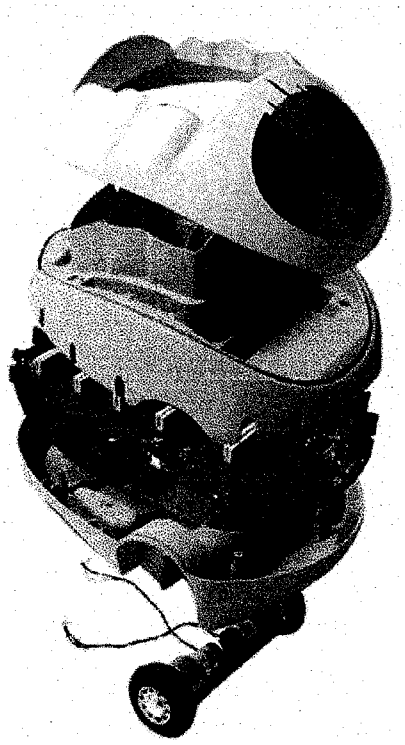


Fig. I. 4 Pekee désassemblé

De tous les capteurs du robot, les capteurs infrarouges sont les plus importants pour connaître l'environnement extérieur. Les mesures de distance entre le corps du robot et l'environnement permettent à Pekee de répondre devant des obstacles grâce à l'algorithme de comportements réflexes développé dans ce project.

Les distances de détection des capteurs sont de 0 à 10 mètres avec une précision de 1 à 5 cm. Le cône de vision est de 10 degrés environ.

Pekee incorpore aussi un capteur de choc qui sera utile dans les situations où les capteurs infrarouges sont insuffisants. En effet, les capteurs infrarouges peuvent être imprécis ou inutiles devant des surfaces qui absorbent ce type de rayons (surfaces principalement noires) ou des surfaces transparentes (verre).

I.2.1 Accessoires. Cartes filles

Le robot Pekee accueille des cartes filles que l'on branche sur le bus OPP de la carte mère. Ces cartes accessoires sont compatibles avec ce bus et permettent la gestion de l'énergie des batteries (si elles en ont une). Les cartes filles sont équipées d'un détrompeur pour garantir un sens correct d'insertion.

Dans ce projet, deux cartes filles ont été utilisés, la carte PC embarqué plus vidéo et la carte wireless.

I.2.1.1 Carte PC embarqué plus Vidéo (Carte fille vidéo)

Il s'agit d'un ordinateur embarqué autonome, pouvant accueillir n'importe quel système d'exploitation compatible Intel x86 (en ce cas Windows 98).

Basé sur un processeur compatible Intel 486, avec RAM, disque dur, sortie SVGA, clavier / souris et réseau Ethernet 10/100 Mbps. Elle dispose de son propre pack d'accumulateurs qui peut être rechargé directement en utilisant un connecteur spécifique, ou en utilisant le chargeur Pekee lorsqu'il est connecté au bus OPP.

De plus, elle inclut une carte d'acquisition de vidéo et une caméra. A travers l'interface OPP, la carte peut communiquer avec la plate-forme Pekee et avec toute autre carte connectée.

La carte fille vidéo comporte tous les connecteurs standard que l'on trouve classiquement sur un PC. La caméra est utilisée via un connecteur coaxial et un câble d'alimentation 5V.

Les caractéristiques principales sont :

- Microprocesseur 485 à 75MHz.
- 32 Mo PC100 SDRAM.
- 128 Mo disque dur (extensible).
- Contrôleur SVGA 4 MB (mémoire partagée).
- Carte d'acquisition de vidéo (signal CVBS).
- Contrôleur Ethernet 10/100 Mbps (IEEE 802.3).
- Contrôleur USB V1.1 à 12Mbps.
- Contrôleur PS/2 clavier/souris.
- Compatible bus OPP.
- Voyant de marche.
- LED marche/arrêt.
- 1 interrupteur de mise sous tension.

Et les spécifications de la caméra de vidéo :

- Dimensions (22 x 22 x 28 mm).
- Faible consommation.
- Lentille avec filtre infrarouge intégré.
- Contrôle automatique de l'exposition.
- Auto-calibration du niveau de noir.
- Balance automatique des blancs.
- Résolution : de 160 x 120 pixels à 640 x 480 pixels.
- Capteur CMOS 1/3 de pouce.
- Alimentation de 5V.
- Sortie de vidéo composite.

Cette carte ajoute des nouvelles caractéristiques au robot Pekee et étend le champ d'applications que l'on peut développer.

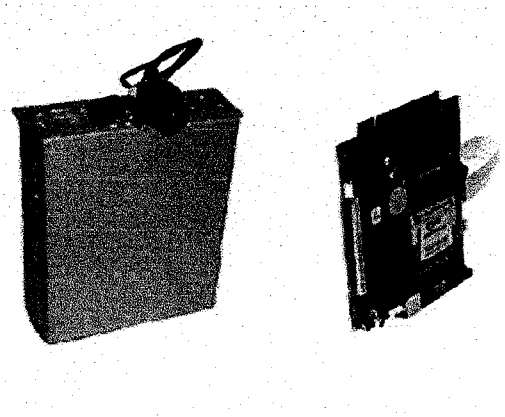


Fig. I. 5 Module PC embarqué + Vidéo

I.2.1.2 Carte Wireless

Cette carte ajoute des capacités de connexion sans-fil au robot Pekee.

La carte wireless inclut une passerelle Ethernet vers un réseau sans fil standard Wi-Fi de l'entreprise Linksys. Cette carte utilise la batterie principale du Pekee via le bus OPP.

Leurs caractéristiques principales :

- Administrable depuis le web.
- Pas de driver spécifique
- Ne consomme aucune ressource du PC embarqué.
- Portée d'utilisation :

  Intérieur
  Jusqu'à 50 m à 11 Mbps.
  Jusqu'à 80 m à 5.5 Mbps.
  Extérieur
  Jusqu'à 150 m à 11 Mbps.
  Jusqu'à 300 m à 5.5 Mbps.

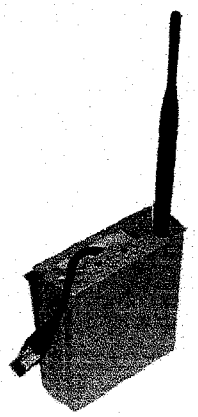


Fig. I. 6 Module Wireless

Dans le chapitre II, on explique avec un peu plus de soin le standard IEEE 802.11 (LAN Sans fil).

I.3 Description Software.

L'entreprise Wany a développé plusieurs librairies et fonctions en langage C++ lesquelles aident à la communication avec le robot Pekee via le bus OPP, à connaître l'état de ses différents capteurs ou à la capture des images de vidéo par exemple.

Dans cette section on explique, brièvement les fonctions utilisées pour la réalisation de l'interface graphique.

I.3.1 Le Bus OPP

Le bus OPP est un bus parallèle de largeur 32 bits native, utilisé en 16 bits actuellement. Il est conçu pour pouvoir aussi fonctionner en largeur 8 bits pour des transferts plus simples.

Le mode d'adressage de ce bus est particulier car il permet d'envoyer une information à plusieurs récepteurs simultanément. Un tri électronique est opéré automatiquement au niveau des éléments connectés afin de ne recevoir que les informations auxquelles ils sont sensibles.

Les transferts classiques séparés « Adresse / Données » sont remplacés par un multiplexage « Type / Données ». Le type correspond à un descriptif des données transmises.

L'élément qui transmet sur le bus est temporairement émetteur du bus, lorsqu'il a terminé son envoi, il redevient récepteur. La synchronisation des éléments raccordés sur le bus n'est pas nécessaire, sachant que c'est l'émetteur qui fixera le débit de la trame qu'il émet. Il est donc tout à fait possible de faire cohabiter des processeurs rapides (de type DSP par exemple) avec des microcontrôleurs peu rapides.

I.3.2 Librairie OppAccess

La librairie OppAccess est utilisée pour accéder au bus OPP. Le but de cette DLL (librairie dynamique) est de décharger le programmeur des problèmes inhérents à l'accès local / distant de l'OPP. OppAccess permet de travailler aussi bien en local qu'à distance, sans se soucier du type de cible.

I.3.2.1 Modes de fonctionnement

OppAccess peut travailler suivant 2 modes :

- Mode Direct : L'accès à l'OPP se fait en direct. Ce mode de fonctionnement n'est valable qu'en local sur le bus physique.
- Mode REMOTE : L'accès à l'OPP se fait via TCP/IP avec le programme OppToNetwork sur le port 12000. Ce programme s'exécute dans la carte PC embarqué. Les requêtes sont transmises sur le réseau IP.

Les fonctions utilisées[1] de cette librairie sont les suivantes :

- *oppaccessStartup :* Cette fonction permet d'effectuer l'initialisation de la dll et doit être appelée une seule fois au début du programme.
- *oppaccessStop :* Cette fonction doit être appelée à la fin de l'exécution du programme pour terminer la dll en fin de programme.
- *oppaccessChooseAccessMode :* Cette fonction permet de choisir le mode d'accès de la dll. Une boîte de dialogue propose les différents modes de fonctionnement.

[1] Le « Manuel DLL OPP Access » de Wany explique en détail toutes les fonctions.

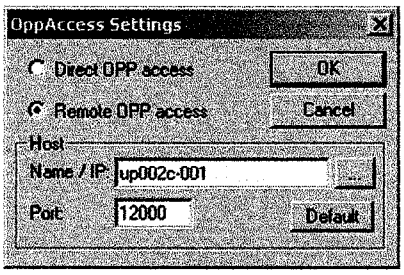


Fig. I. 7 Boîte de dialogue OppAccess

- *oppaccessOppStartup :* Cette fonction permet d'effectuer l'initialisation du moteur OPP. Ses paramètres sont le serveur sur lequel on désire se connecter, le port TCP utilisé pour la connexion et le type d'accès demandé (Direct ou Remote).

- *oppaccessOppStop :* Cette fonction doit être appelée à la fin de l'utilisation du moteur OPP.

- *oppaccessSendFrame :* Cette fonction permet d'envoyer une trame sur l'OPP suivant le mode de fonctionnement en cours. Ses paramètres sont le Function Key à écrire, la signature de la carte émettrice, le type de la trame, la taille des données et les données.

- *oppaccessReceiveFrame :* Cette fonction permet de recevoir une trame OPP. L'appel est bloquant jusqu'à ce que se produise une réception d'une trame valide, une erreur de réception ou la fermeture de la connexion.

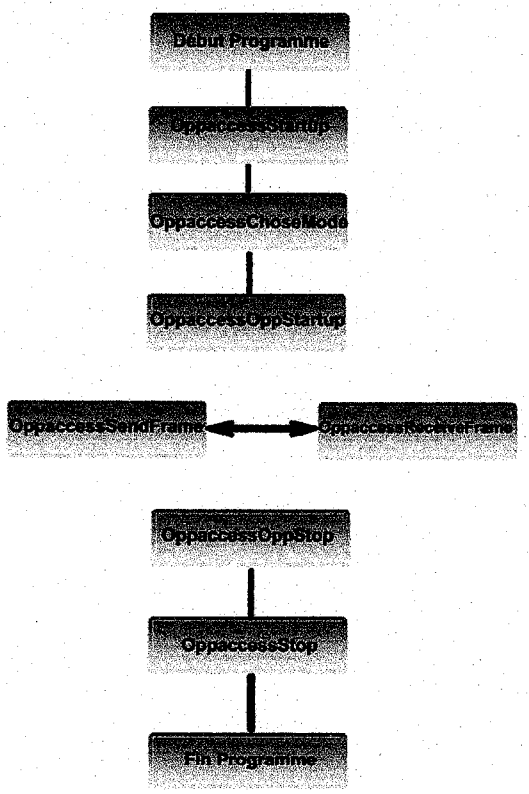


Fig. I. 8 Ordre fonctions OppAccess

I.3.3 OPP Function Keys

Les Function Keys (FK) sont des fonctions implémentées par Wany pour faciliter le control et la supervision du robot Pekee. Ces fonctions envoient et reçoivent grâce au bus OPP et à la libraire OppAccess (oppaccessSendFrame / oppaccessReceiveFrame).

I.3.3.1 Structures utilisées

Certaines FK ont besoin d'informations structurées dont voici le détail écrit en C++ :

- *WPOSITIONSTRUCT :* position absolue de la base mobile. La position absolue est exprimée en mètres et est mise à jour par la base mobile.

```
typedef struct
{
double dTETA; //rotation de radian
double dXPos; //position X en m
double dYPos; //position Y en m
} WPOSITIONSTRUCT;
```

- *WSENSORSSTRUCT:* mesures de variables proprioceptives. Il est possible de demander à la base mobile de renvoyer différentes mesures en une seule trame ce qui permet de simplifier le développement et de réduire le trafic sur l'OPP.

```
typedef struct
{
UINT16 wIRSensors[15]; //Capteurs infra rouge WANY [0 25]. Les valeurs peuvent être à 99si rien n'est vu.
UINT16 wGyros[2]; //Valeur instantanée des gyromètres X et Y
UINT16 wTemperatures[2]; //Température interne et externe [0 1024]
UINT16 wImpact; //Capteur de choc [0 1]
UINT16 wLight; //Lumière extérieure [0 1024]
} WSENSORSSTRUCT;
```

- *WIMAGEINFOS :* informations sur les images caméra. Il est possible d'obtenir des informations sur les images qui transitent sur le bus.

```
#define COMPRESSIONTYPE_BMP 1
#define COMPRESSIONTYPE_JPG 2
typedef struct
{
UINT32 dwDx; //taille horizontale de l'image en pixels
UINT32 dwDy; //taille verticale de l'image en pixels
UINT32 dwBitsPerPixels; //nombre de bits par pixels (pour COMPRESSIONTYPE_BMP)
UINT32 dwImageType; // COMPRESSIONTYPE_BMP ou COMPRESSIONTYPE_JPG
UINT32 dwImageDataSize; //taille en octets de l'image
} WIMAGEINFOS;
```

- *WPRODUCT :* information sur le produit.

```
#define WPRODUCT_STRINGSSIZE 20
typedef struct WPRODUCT
{
INT8 szProduct [WPRODUCT_STRINGSSIZE]; //nom du produit
INT8 szCard [WPRODUCT_STRINGSSIZE]; //nom de la carte
INT8 szVendor [WPRODUCT_STRINGSSIZE]; //vendeur
INT8 szUserInfo[WPRODUCT_STRINGSSIZE]; //informations utilisateur
}WPRODUCT;
```

- *WRUNNINVERSION :* Cette structure contient des informations sur la version logicielle qui est exécutée dans le module.

```
#define WRUNNINGVERSION_STRINGSSIZE 20
typedef struct
{
UINT16 wMajor; //numéro majeur de version
UINT16 wMinor; //numéro mineur de version
INT8 szName [WRUNNINGVERSION_STRINGSSIZE]; //nom de la version
INT8 szDesc [WRUNNINGVERSION_STRINGSSIZE]; //description
INT8 szUserInfo [WRUNNINGVERSION_STRINGSSIZE]; //informations utilisateur
} WRUNNINGVERSION;
```

- *WSPEEDSTEERINGINFOS :* Cette structure contient le déplacement instantané en vitesse et direction.

```
typedef struct
{
INT16 iSpeed; //Vitesse (voir FK_SET_MOVE pour bornes)
INT16 iSteering; //Direction
} WSPEEDSTEERINGINFOS;
```

- *WODOMETERS :* Cette structure contient le numéro de tours de chaque moteur en un incrément de temps.

```
typedef struct
{
UINT16 wOdometer0; //valeur de l'odometre 0 [0 65535]
UINT16 wOdometer1; //valeur de l'odometre 1 [0 65535]
UINT32 dwTime; //numéro d'incrément de temps [0 65535]
} WODOMETERS;
```

- *ROBOTINTERNALSTATES :* Cette structure permet de transporter en une seule trame OPP les valeurs les plus couramment utilisées.

```
typedef struct
{
WSENSORSSTRUCT Sensors;
WPOSITIONSTRUCT Position;
WODOMETERS Odometers;
UINT32 dwTime; //incrément de temps courrant
UINT16 wCPUPowerLevel;
UINT16 wMotorsPowerLevel;
}ROBOTINTERNALSTATES;
```

I.3.3.2 Functions Keys utilisées[2]

Cette section liste l'ensemble des Functions Keys utilisées avec leurs types de requêtes.

- *FK_GET_INFORMATIONS :* Permet de demander des informations sur les modules connectés à l'OPP.
    - *Type de requête (FK_INFORMATIONS_TYPE_) :*
        - *_RUNNINGVERSION :* Demande informations dur la version actuellement exécutée dans le module.
        - *_PRODUCT :* Demande informations générales sur le produit.
- *FK_ANS_INFORMATIONS :* Réponse à la demande d'information.
    - Type de requête (FK_INFORMATIONS_TYPE) :
        - *_RUNNINGVERSION :* Informations sur la version actuellement exécutée dans le module.
        - *_PRODUCT :* Informations générales sur le produit.
- *FK_SET_HARDFUNCTIONS :* Permet de modifier l'état de variables internes.
    - *Type de requête (FK_HARDFUNCTIONS_TYPE) :*
        - *_REFLEX :* Positionne le reflex (OFF/ON). Reflex est une fonction pour éviter des obstacles implémentée à la base mobile du robot.

[2] Le « Manuel OPP Function Keys » de Wany explique toutes les fonctions et ses requêtes en détail.

- *_ENSLAVEMENT :* Positionne l'asservissement moteurs (ON/OFF).

- *_WANYIRSENSIBILITY :* Positionne la rampe de puissance des capteurs IR.

- *_MOVECOMMANDPROTECTION :* Positionne la protection de commande moteurs (OFF/ON). Si la protection est active, une consigne de déplacement est valable par défaut pour 10 fois la base de temps du Pekee (63 ms). Si aucune autre commande de déplacement n'est intervenue dans cet intervalle, la base mobile stoppe les moteurs et enclenche l'asservissement moteur.

  Si la protection n'est pas active, la base applique les commandes de déplacement telles qu'elles sont.

- *_MOVECOMMANDPROTECTIONTIME :* Positionne le délai avant stop suite à une commande de déplacement.

- *FK_GET_HARDFUNCTIONS :* Permet de demander l'état de variables internes.

  - *Type de requête (FK_HARDFUNCTIONS_TYPE) :*

    - *_REFLEX :* Demande l'état de marche du Reflex (OFF/ON).

    - *_ENSLAVEMENT :* Demande l'état de marche de l'asservissement moteurs (OFF/ON).

    - *_MOVECOMMANDPROTECCION :* Demande information sur état de la protection des commandes moteurs.

    - *_MOVECOMMANDPROTECCIONTIME :* Demande information sur le délai avant stop pour la protection des commandes moteurs.

- *FK_ANS_HARDFUNCTIONS :* Réponse sur une demande d'état de variables internes.

  - *Type de requête (FK_HARDFUNTIONS_TYPE) :*

- *_REFLEX :* Retourne l'état de marche du Reflex (OFF/ON).
- *_ENSLAVEMENT :* Retourne l'état de marche de l'asservissement moteurs (OFF/ON).
- *_MOVECOMMANDPROTECCION :* Information sur état de la protection des commandes moteurs.
- *_MOVECOMMANDPROTECCIONTIME :* Information sur le délai avant stop pour la protection des commandes moteurs.

- *FK_SET_MOVE :* Permet de commander un déplacement ou affecter la position absolue.
  - *Type de requête (FK_SETMOVE_TYPE) :*
    - *_SPEEDANDSTEERING :* Commande un déplacement en vitesse / direction. Les valeurs maximums de vitesse sont -22 pour arrière à fond, 0 pour stop et +22 pour avant à fond. Les valeurs maximums pour la direction sont -22 pour virage à gauche à fond, 0 pour droit devant et +22 pour virage à droite à fond.
    - *_SETPOSITIONOFFSET :* Initialise la position absolue de la base.
- *FK_GET_MOVE :* Permet de demander des informations sur le déplacement de la base.
  - *Type de requête (FK_GETMOVE_TYPE) :*
    - *_POSITION :* Demande la position absolue de la base.
- *FK_ANSGETMOVE :* Permet d'obtenir des informations sur le déplacement de la base.
  - *Type de requête (FK_GETMOVE_TYPE) :*
    - *_POSITION :* Position absolue de la base.
- *FK_GET_MEASURE :* Permet de demander des mesures proprioceptives.
  - *Type de requête (FK_MEASURES_TYPE) :*

- *_INTERNALSTATES :* Demande d'informations structurées complètes (ROBOTINTERNALSTATES). On peu obtenir une seule trame de réponse ou obtenir des réponses automatiques.

- *FK_ANS_MEASURES :* Permet d'obtenir des informations structurées de la base.

  - *Type de requête (FK_MEASURES_TYPE) :*

    - *_INTERNALSTATES :* Informations structurées complètes (ROBOTINTERNALSTATES).

- *FK_ANS_IMPACTSTATE :* Permet d'obtenir l'état du capteur de choc.

  - *Type de requête (FK_MEASURES_TYPE) :*

    - *_SPONTANEOUSIMPACTNOTIFY :* Trame spontanée de choc. Contient en plus le nombre de chocs depuis le démarrage.

- *FK_SET_SOUND :* Permet de faire un son.

  - *Type de requête (FK_SOUND_TYPE) :*

    - *_BUZZER :* Son du buzzer.

I.3.4 Librairie Graphique (Codecsaa7111a)

Cette librairie permet de travailler avec le numériseur SAA711A compris dans le PC embarqué. Celle-ci met à disposition un ensemble de fonctionnalités simples d'utilisations pour faire de l'acquisition vidéo.

Dans cette section, on explique brièvement seule les fonctions utilisées[3] pour la réalisation de l'interface graphique.

- *Codecsaa711aStartup :* Cette fonction permet d'effectuer l'initialisation de la dll et droit être appelée une seule fois au début du programme.

- *Codecsaa7111aStop :* Cette fonction permet de stopper le module et restituer l'espace mémoire alloué. L'appel à cette fonction doit se faire une seule fois à la fin du programme utilisateur.

[3] Plus information à le document « KIT VISION codecsaa7111a » de Wany.

- *Codecsaa7111aGetNativeInformation :* Pour pouvoir démarrer la capture vidéo, une structure contenant les informations de capture doit être fournie. Cette fonction permet d'obtenir les informations natives du système de capture. L'utilisateur peut modifier certains champs (résolution, luminosité,.....).

- *Codecsaa7111aAdjustImageInfo :* Cette fonction permet de corriger une structure d'information modifiée ou créée par l'utilisateur.

- *Codecsaa7111aStartCapture :* Cette fonction permet de démarrer le système de capture vidéo. Une fois démarré, le système de capteur est prêt à appeler la fonction CALLBACK.

- *Codecsaa7111aStopCapture :* Cette fonction permet de stopper la capture vidéo et restitue les ressources allouées pour cette tache.

- *Codecsaa7111aSetVideoCallback :* Cette fonction permet de définir une fonction utilisateur que le système de capture appellera chaque fois qu'une image vidéo sera prête.

- *Codecsaa7111aSetParameter :* Cette fonction permet de modifier les paramètres d'affinage d'image (luminosité, saturation,....).

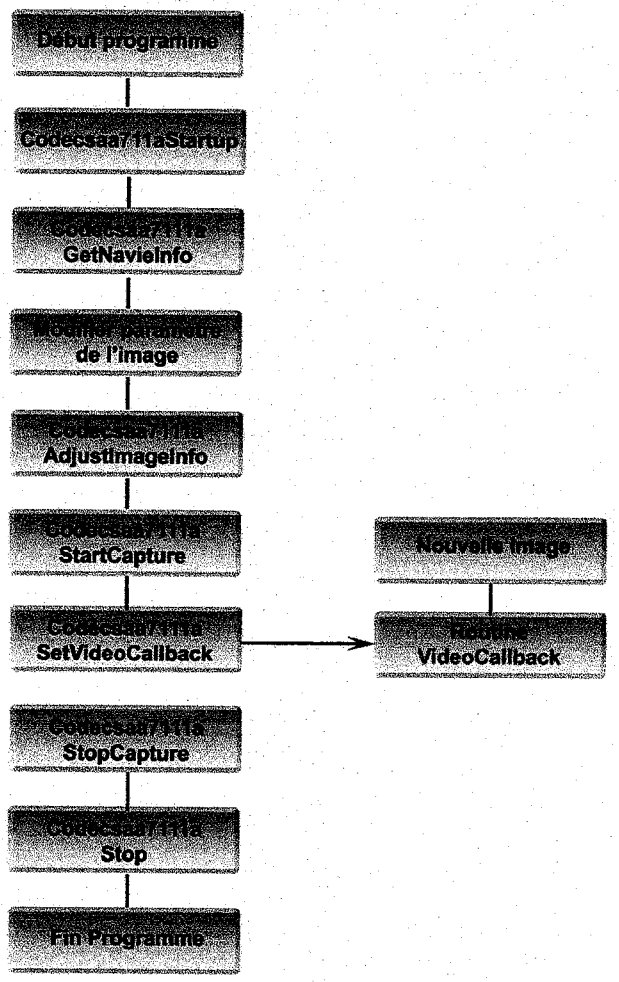


Fig. I. 9 Ordre fonctions Codecsaa711a

CHAPITRE II

LE STANDARD 802.11 WIRELESS LAN

II.1 Définition

Le standard IEEE 802.11 a été élaboré en 1997 et il a été développé pour favoriser l'interopérabilité du matériel entre les différents fabricants. L'implantation du standard fut un pas important pour le développement rapide de la communication sans fil. Les réseaux 802.11 permettent la connexion et l'accès aux réseaux traditionnels câblés (Ethernet, Token Ring,..) comme si c'était une extension de cette dernières mais avec la flexibilité et la mobilité qu'offrent les réseaux sans fil.

Le standard définit un choix de différentes couches physiques. Celles-ci sont au choix DSSS (Direct Sequence Spread Spectrum), ou FHSS (Frequency Hopping Spread Spectrum).

La norme 802.11 connaît différentes versions suivant les modifications et les améliorations. De cette manière, nous avons les spécifications suivantes :

- *802.11* : Spécification pour 1-2 Mbps en la bande des 2.4GHz, utilisant FHSS ou DSSS.
- *802.11b* : Extension de 802.11 pour offrir 11Mbps utilisant DSSS.
    - *Wi-Fi (Wireless Fidelity)* : Promulgué par le WECA pour certifier produits 802.11b capables d'interoperer avec les autres fabricants.
- *802.11a* : Extension de 802.11 pour offrir 54Mbps utilisant OFDM.
- *802.11g* : Extension de 802.11 pour offrir 20-54 Mbps utilisant DSSS et OFDM. C'est compatible avec 802.11b. Il a majeur porté et mineur consommation.

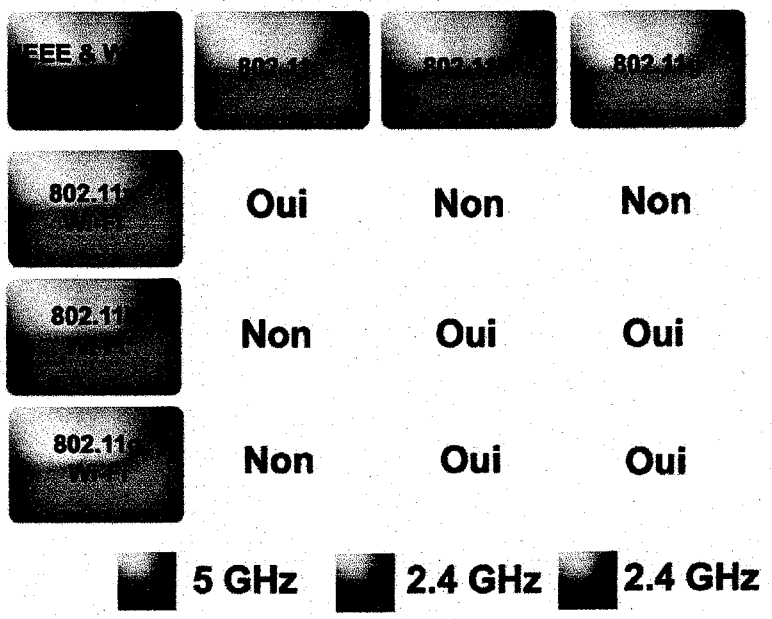


Fig. II. 1 Tableau de compatibilité entre Standards Wireless

II.2 Caractéristiques principales

Le standard 802.11 est très robuste et plein de fonctionnalités. Il présente de nombreux avantages permettant de minimiser les interférences et de maximaliser la bande passante sur les canaux. Il peut travailler de manière transparente avec l'ethernet à travers un pont, ou un point d'accès, de manière à ce que tous les éléments avec et sans fils puissent interagir. Ses principales caractéristiques sont :

- *Roaming-Itinérance :* Le standard 802.11 permet de « roamer » entre plusieurs points d'accès étant sur le même ou sur différents canaux. Le client en roaming peut utiliser ce signal pour déterminer la puissance de sa connexion avec la station de base. Si ce signal est jugé insuffisant ou faible le client en roaming peut décider de s'associer à une nouvelle station.

- *W.E.P (Wired Equivalent Privacy) :* Le standard définit un mécanisme par lequel le WEP peut être atteint. Celle-ci correspond à une encryption 40bit RC4. Si le WEP est mise en œuvre alors toutes les données transmises sont encryptées.

- *Interopérabilité :* Avec le modification 802.11b (High Rate), l'un des plus grands avantages du standard est la capacité pour des produits venant de fabricants différents à opérer entre eux. Ceci permet à l'utilisateur de trouver le matériel répondant spécifiquement à ses besoins.

II.3 Technologie DSSS (Direct Sequence Spread Spectrum)

La technologie DSSS consiste à diffuser le signal de l'information en utilisant la bande passante disponible, c'est-à-dire, au lieu de concentrer l'énergie des signaux autour d'une porteuse concrète, DSSS se reparte sur toute la bande disponible. Cette bande passante totale se partage entre les usagers qui travaillent sur la même bande fréquentielle.

Dans le cas des États Unis et de l'Europe, la technologie DSSS opère dans la bande 2.4 GHZ - 2.4835 GHz, c'est-à-dire, avec une bande passante totale disponible de 83.5 MHz.

Cette bande se divise en 14 canaux avec une bande passante par canal de 5 MHZ, chaque pays utilise un ensemble de canaux choisis selon les normes régulatrices pour chaque cas particulier.

Dans le cas de la France, les canaux 10, 11, 12, 13 sont utilisés dont les fréquences centrale sont 2.457, 2.462, 2.467, 2.472 GHz.

II.4 Types de connexions

Les cartes 802.11b ont été conçues pour fonctionner selon deux modes :

- Mode « peer to peer » (P2P) ou AdHoc.
- Mode infrastructure ou mode centralisé.

II.4.1 Mode P2P ou AdHoc

Le mode peer-to-peer est une connexion directe de machine à machine, toute machine est à la fois serveur et client. Ce mode est adapté à des situations où les machines restent proches les unes des autres, et qu'aucune organisation n'est nécessaire. C'est une organisation anarchique qui convient quand il y a peu de machines.

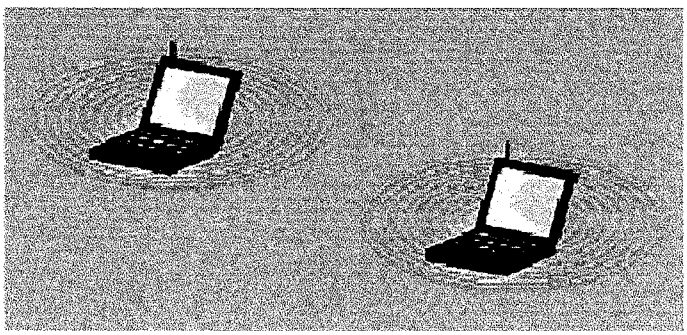


Fig. II. 2 Mode AdHoc

II.4.2 Mode Infrastructure ou Centralisé

Le mode centralisé est organisé, comme sont nom l'indique, selon une architecture serveur-client : le serveur s'occupe de router le trafic entre les différentes machines. Pour augmenter la portée du réseau, on utilise un point d'accès. Avec cet élément, on double la portée du réseau (la distance maximum permise n'est pas entre les stations, mais entre chaque station et point d'accès).

Entre autre, les points d'accès peuvent être connectés à d'autres réseaux et, en particulier, à un réseau fixe à partir duquel un usager peut avoir accès aux ressources depuis sa machine mobile.

Pour permettre une couverture totale d'une zone déterminée, il faudra installer plusieurs points d'accès de manière à couvrir la superficie nécessaire en tenant compte de la couverture qui proportionne chaque point d'accès. Il faudra aussi que chaque zone soit juxtaposée pour permettre le passage d'une cellule à une autre sans perdre la communication.

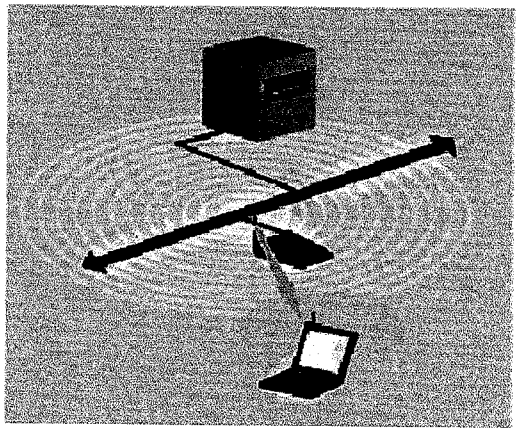


Fig. II. 3 Mode Infrastructure

II.5 Contrôle à distance. Wireless en Pekee

Le robot Pekee utilise un module wireless 802.11b de l'entreprise LynkSys pour en permettre le contrôle à distance. Le taux de transfert maximal est donc de 11 Mbps.

La carte wireless inclut une passerelle (bridge) Ethernet, laquelle permet de travailler de manière transparente avec le réseau.

II.5.1 Configuration de la connexion

La connexion sans fil entre le robot Pekee et un ordinateur se réalise grâce à deux modules wireless, un à chaque extrême de la connexion. Chaque module se connecte à la carte réseau Ethernet correspondante. Les quatre dispositifs

doivent être configurer avec une adresse IP valide. De plus, on doit sélectionner le canal de travail et le mode de connexion (AdHoc ou Infrastructure).

Dans le cas de ce projet on travaille en mode AdHoc, généralement utilisé avec Pekee, puisque il existe seulement une machine qui contrôle Pekee. Le canal est le 10 et les adresses IP sont ainsi définies :

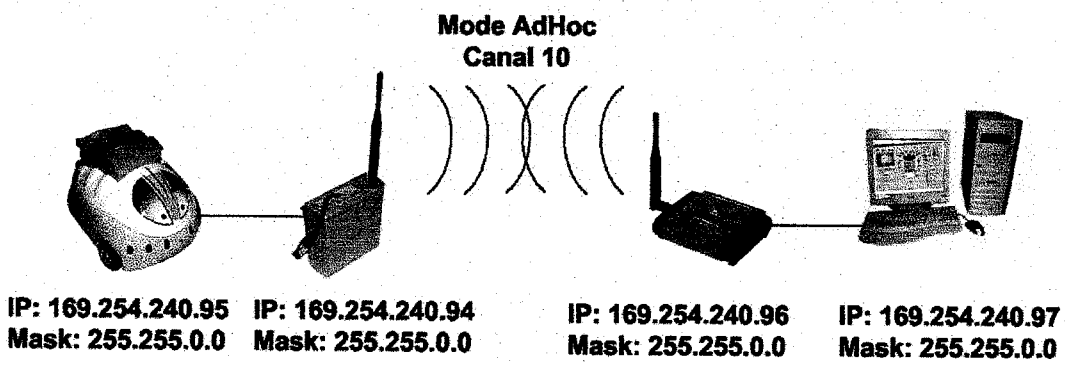


Fig. II. 4 Connexion Pekee - Ordinateur

On peut observer qu'au niveau de la programmation, que la connexion soit sans fil ou câblée, cela n'apporte aucune difficulté en plus.

CHAPITRE III

L'INTERFACE GRAPHIQUE LES PROGRAMMES PEKEECONTROL ET SERVERCAPTURE

III.1 Description général.

PekeeControl est le nom de l'interface graphique développée dans ce projet. De même, il a été réalisé le programme ServerCapture qui s'occupe des fonctions de capture d'images et serveur vidéo. ServerCapture s'exécute dans le module PC embarqué de Pekee. Les deux programmes sont développés en *Visual C++ 6.0* avec *Microsoft Foundation Classes* (MFC).

Visual C++ est un langage de programmation orientée objets intégrant l'environnements graphiques comme Windows. Tandis qu'en langage C, l'unité de programmation est la fonction, en C++, c'est la classe. Les objets sont créés à partir de ces classes.

MFC est un ensemble de classes développées par Microsoft qui facilitent énormément la programmation d'une application en Visual C++.

Pour le développement de PekeeControl et ServerCapture, des classes neuves ont été développées. Ils utilisent des classes de MFC et des classes générées par l'entreprise Wany. Le chapitre III ne comporte donc aucun code VC++, mais vous pourrez trouver le code des programmes dans l'annexe. Pour faciliter la compréhension, l'explication des caractéristiques et des fonctions est basée sur des schémas blocs et des organigrammes.

Entre autre de ces deux programmes, on utilise aussi le programme OppToNetwork développé par Wany et qui permet de traduire les messages envoyés via TCP/IP en messages OPP compris par Pekee. Ce programme s'exécute ensemble avec ServerCapture dans le module PC embarqué dur robot.

III.1.2 Caractéristiques de PekeeControl

L'interface graphique PekeeControl permet le suivi et le contrôle à distance du robot Pekee. Les caractéristiques et les fonctions les plus importantes sont les suivantes :

- Visualisation et conversion des données générées par les capteurs présents dans le robot : Capteurs de distance, température, lumière, impacts, vitesse, position ou niveau de batteries.

- Réception d'images capturées par la camera située dans le robot mesurant la vitesse de transfert (images par seconde) et son retard (Client TCP / IP).

- Fonction pour le contrôle par clavier du mouvement du robot.

- Fonction pour la détection d'obstacles et rectification automatique de trajectoire.

- Fonction pour la reconnaissance et suivi d'objets de couleur rouge.

III.1.3 Caractéristiques de ServerCapture

ServerCapture s'exécute dans le module PC embarqué du robot Pekee. Ce programme réalise la capture d'images de la camera et aussi de serveur de vidéo TCP / IP pour la distribution de ces images à un client (PekeeControl). Les caractéristiques les plus importantes sont :

- Changement des paramètres de l'image à capturer : Image en couleur (24 bits) ou Image en blanc et noir (8 bits), saturation, luminosité, contraste, couleur.

- Compression de l'image en JPG et sélection du niveau de qualité de 0 à 100 %.

- Fonction Preview pour observer les images avec les paramètres sélectionnés.

- Fonction AutoStart qui permet le démarrage automatique du programme.

- Sauvegarde des paramètres sélectionnés pour être utilisés la fois prochaine en initiant le programme.

III.2 Descriptions fonctionnelles

Le robot Pekee à l'initialisation met en marche le module PC embarqué et le module wireless. Le module PC embarqué est sous Windows 98 qui exécute le programme ServerCapture et OppToNetwork. Les deux programmes restent à l'écoute d'un ordre du client, dans ce cas, du programme PekeeControl.

Les demandes d'images de la part du programme PekeeControl sont contrôlées par ServerCapture. Une fois reçue, l'image est traitée et visualisée à l'écran.

La demande de données des capteurs et les ordres de mouvement sont contrôlées par OppToNetwork. Les données reçues des capteurs sont gardées pour être traités.

Pendant que le processus principal du programme *PekeeControl* attend un événement quelconque généré par l'usager (change d'options, fin de programme,...), ils se réalisent 3 processus en parallèle:

- Un thread (processus) de réception et traitement de l'image.
- Un thread de réception de données envoyées par Pekee.
- Une interruption exécutée périodiquement (chaque 300ms) pour la gestion des données.
- Un thread (programme principal) qui reste en attente d'un événement généré par l'usager.

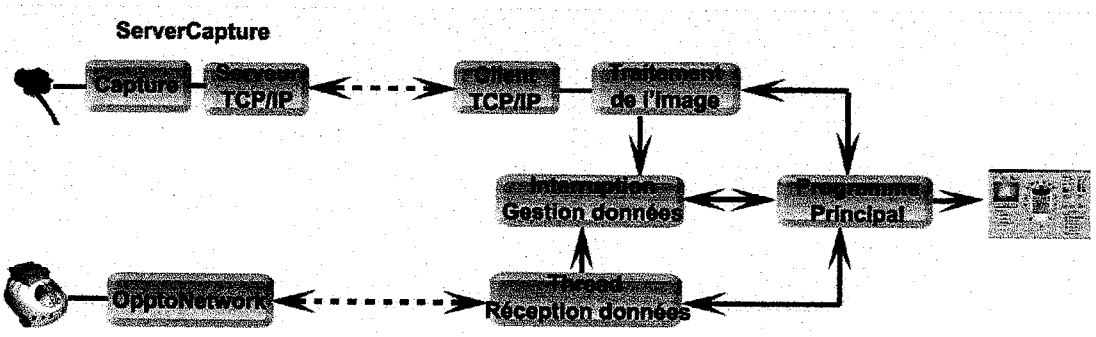


Fig. III.1 Schéma général de l'application

III.2.1 Thread de réception et traitement des images

Ce processus réalise la tâche de client TCP/IP en communiquant avec le serveur vidéo. Au début, le client demande au serveur des informations sur l'image (taille, résolution, couleur ou blanc et noir,...) et si elle est compressée en JPG ou au format BMP.

Une fois connues les propriétés des images qui vont être reçues, commence une boucle continue sollicitant et recevant les images capturées par la camera. Pendant que se finalise la connexion avec le serveur, ce processus reçoit une image et on la visualise à l'écran. En outre, si l'usager a activé la fonction

Recognize qui permet la détection d'objets de couleur rouge, ils se réalisent ainsi les tâches de traitement d'image.

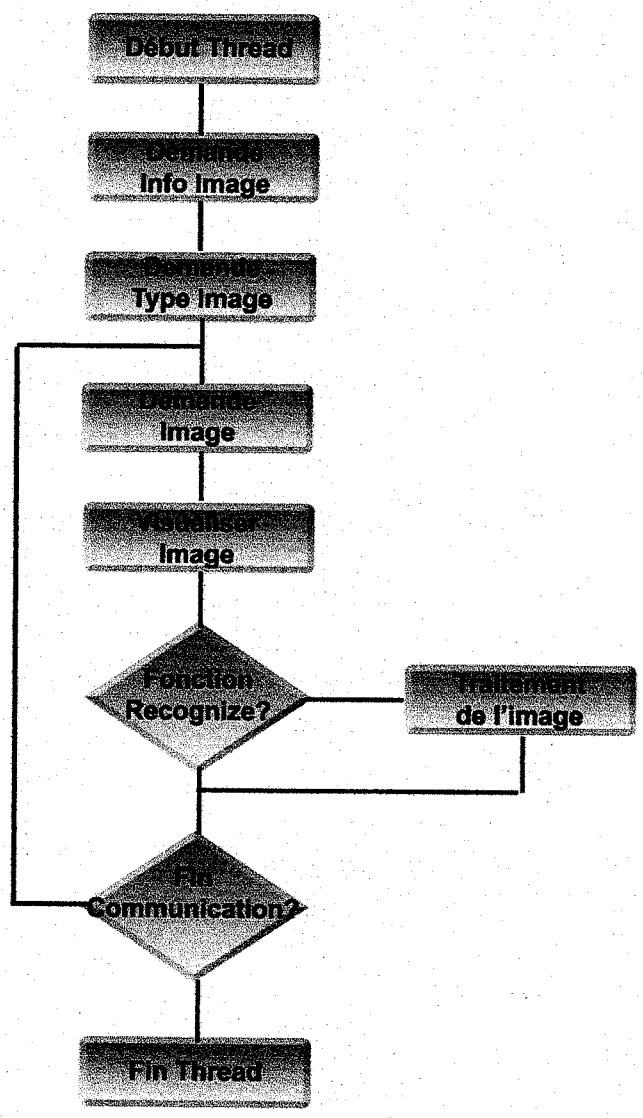


Fig. III.2 Organigramme Thread réception de l'image

III.2.2 Thread de réception de données

Ce processus reçoit périodiquement les données envoyées par le robot Pekee. Ces données peuvent être de 3 types:

- Information générale sur le robot. L'usager a sollicité des informations sur la version du robot.

- Information sur les chocs. Lorsqu'un choc se produit, il se génère un message spontané indiquant le nombre de chocs depuis le démarrage du robot.

- Information sur les capteurs. Ces informations sont reçues tous les 126ms. On y trouve les valeurs de tous les capteurs du robot dans un seul paquet d'information (ROBOTINTERNALSTATES).

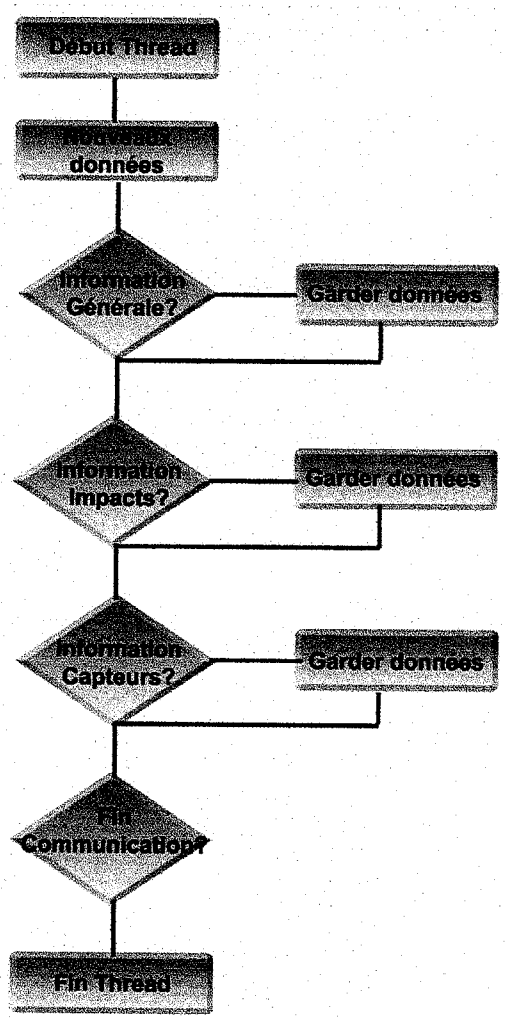


Fig. III.3 Ordinogramme Thread réceptions données

III.2.3 Interruption de gestion de données

Cette interruption se charge de montrer à l'écran les informations des capteurs reçus par le thread de réception de données. Ces informations sont représentées de manière graphique (niveau de batteries, capteurs de distance, capteurs de lumière) ou numérique (température, vitesse, position,...).

Entre autre, il réalise un appel aux fonctions sélectionnées par l'usager:

- Fonction *Keyboard*. Si cette fonction est activée, le mouvement du robot est contrôlé grâce au clavier.

- Fonction *Reflex*. Si cette fonction est activée, le robot bouge tout seul en évitant les obstacles.

- Fonction *Recognize*. Les fonctions de traitement nécessaires pour reconnaître les objets rouges sont réalisées dans le thread de réception d'images. Si cette fonction est activée, l'interruption montre le résultat.

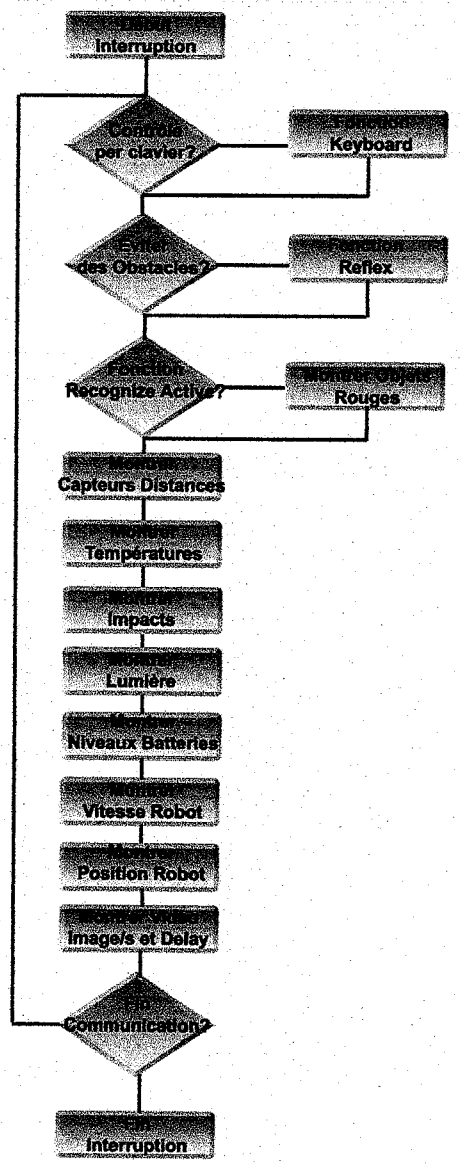


Fig. III.4 Organigramme Interruption gestion des données

III.3 La transmission de vidéo. Le programme ServerCapture

La transmission des images générées par la camera se réalise grâce au programme ServerCapture. L'organigramme de la figure II.9 montre le fonctionnement interne du programme. Fondamentalement, le programme travaille avec deux processus en parallèle pendant que le processus principale attend un événement généré par l'usager (changement d'options, fin de programme,....).

Une fois initié, le programme récupère la configuration de la dernière exécution. Si l'option *AutoStart* est sélectionnée, le programme initie la capture et le serveur de vidéo ; sinon il attend que l'usager génère un ordre.

La capture d'images utilise les fonctions de la librairie graphique *Codecsaa7111* expliquée dans le chapitre 1. Ces fonctions permettent de travailler avec le processeur graphique du module PC embarqué du robot. Chaque fois que le processeur obtient une image, il s'exécute la routine VideoCall. Cette routine garde l'image et la convertit en JPG et on peut la visualiser à écran si cette option est sélectionnée.

En outre, il active un flag (événement) indiquant au thread qui réalise la fonction de serveur de vidéo qu'il existe une image neuve. Avec ce flag, on évite les erreurs de partage des objets. L'image capturée est gardée dans un objet créé à partir de la classe *CSImage* (expliquée plus bas).

Cet objet est utilisé tant par la routine *VideoCall* que par le thread du serveur de vidéo, une pour garder l'image neuve et l'autre pour l'envoyer au client. En travaillant en parallèle, les deux processus tentant d'accéder à un objet en même temps produisent une erreur de programme.

Avec l'utilisation de ce flag, on évite l'accès simultané à l'objet car le thread du serveur de vidéo attend une nouvelle image pour y accéder. Lorsqu'il a réalisé une copie de l'image, il libère le flag pour que la routine *VideoCall* puisse garder l'image suivante.

Le thread qui réalise la fonction de serveur de vidéo attend la connexion d'un client par le port IP 12003. La communication se réalise en utilisant un objet créé à partir de la classe *TCPCS* (expliquée plus bas).

Lorsqu'un client s'est connecté au serveur (le client est le programme *PekeeControl*), il peut réaliser trois demandes :

- Demande d'information de l'image : Des dimensions en píxels, taille, couleur, etc.
- Demande du type d'image : JPG ou BMP.
- Demande de l'image.

Lorsque le serveur détecte qu'il y a une image neuve capturée, il réalise une copie d'elle, libère le flag de contrôle et il envoie la réponse de la demande du client.

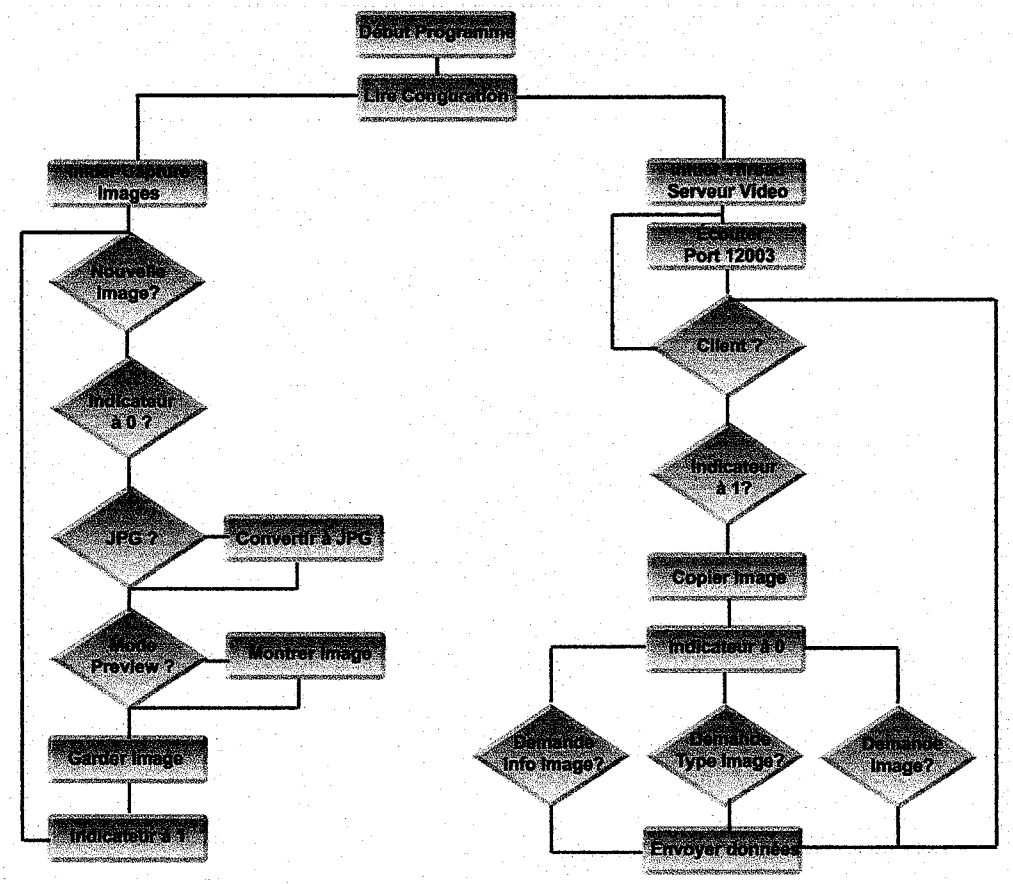


Fig. III.5 Organigramme de ServerCapture

III.3.1 Serveur / Client. La classe TCPCS

Une connexion est le point final d'une communication entre réseaux. En d'autre terme, les communications de réseaux impliquent à deux ordinateurs ou à deux processus qu'il se passe des données de l'un vers l'autre à travers d'un réseau.

Pour qu'une communication ait lieu entre deux programmes, il faut qu'il s'établisse un dialogue chez les deux points extrêmes de la liaison.

La classe *TCPCS* a été développée pour permettre la communication entre les deux extrêmes. Comme il sera expliqué après, elle incorpore des fonctions tant pour le serveur que pour le client.

La classe *TCPCS* utilise la classe *CSocket.* Cette classe est incluse dans *Microsoft Foundation Classes.* Sa fonction est de faire l'intermédiaire entre les

fonctions de niveau bas utilisées par Windows pour la communication et les fonctions de niveau haut de la classe *TCPCS.*

La classe *CSocket* inclut des fonctions qui facilitent les tâches suivantes :

- Création de connexions.
- Établissement d'une connexion.
- Écouter au travers d'une connexion.
- Écriture et lecture de données.

La classe *TCPCS* inclut les méthodes suivantes :

- *ConnectServer :* Cette méthode permet au client (PekeeControl) la connexion au serveur de ServerCapture.
- *ListenPort :* Cette méthode permet au serveur d'écouter par le port 12003 la demande d'une connexion.
- *SendBuffer :* Cette méthode permet au client comme au serveur l'envoi de données.
- *ReceiveBuffer :* Cette méthode permet au client comme au serveur la réception de données.

Une considération d'importance : la fonction *ConnectServer* peut utiliser une certaine quantité de temps avant de faire son travail, c'est-à-dire que l'application peut se mettre en pause pendant l'établissement de la connexion.

Ce type de fonctions qui se charge d'établir une connexion en mettant en pause l'application s'appelle *appels de blocage.*

Les fonctions *ListenPort* et *ReceiveBuffer* sont aussi « appels de blocage ». Le mieux est d'établir les appels à ces fonctions dans un thread comme il est réalisé dans les programmes *ServerCapture* et *PekeeControl.*

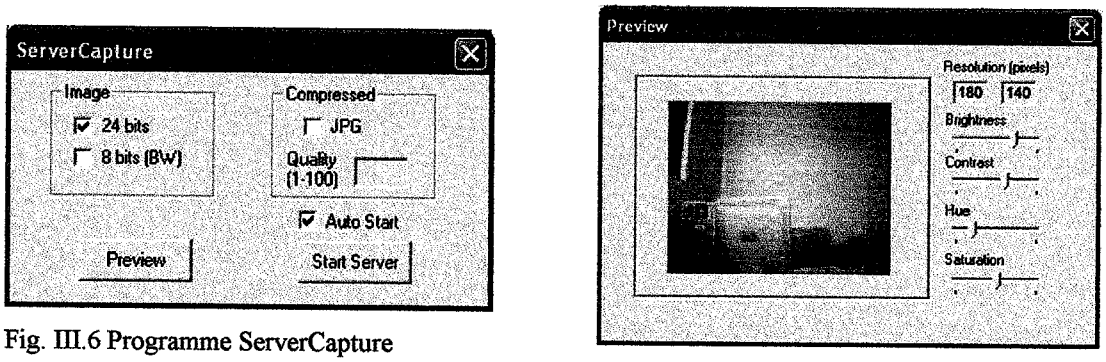


Fig. III.6 Programme ServerCapture

III.4 Traitements de l'image. Fonction Recognize

La fonction Recognize permet la reconnaissance d'objets de couleur rouge dans l'image capturée par la camera du robot. Pour cela, il utilise différentes techniques de traitement d'image.

Une fois l'image acquise, pour séparer les éléments qui contiennent la couleur rouge, on utilise un processus de segmentation. Puis, pour reconnaître la position où l'objet se trouve, on utilise l'algorithme Optical Flow qui permet de calculer le vecteur de vitesses de l'image et de pouvoir déterminer le mouvement que doit réaliser le robot pour se centrer sur l'objet.

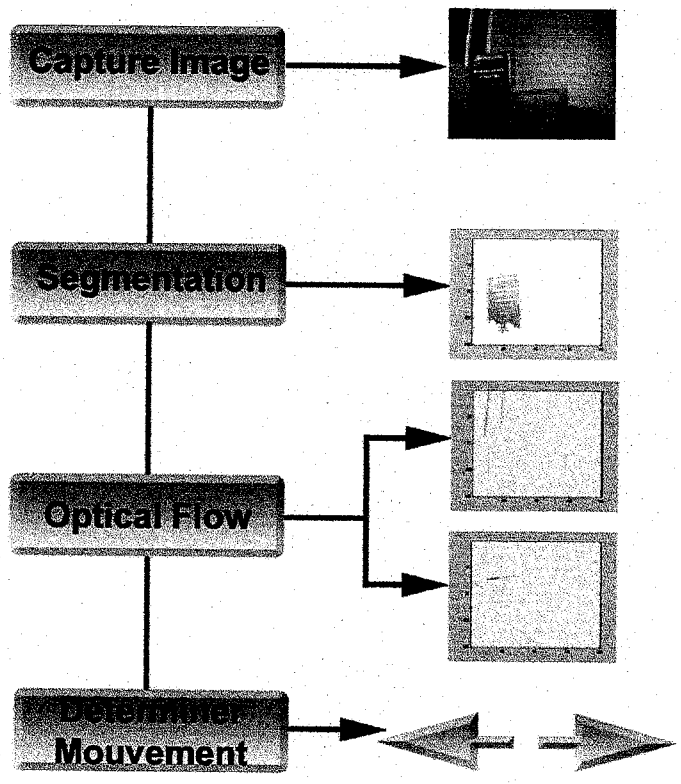


Fig. III.7 Fonction Recognize

L'objectif basique de cette fonction est de s'exécuter avec la fonction Reflex. Si les deux fonctions sont actives, pendant que le robot bouge, tout en évitant les obstacles, il tente de reconnaître les objets de couleur rouge. S'il en trouve un dans son champ de vision, il se dirigera vers lui et tentera de le centrer dans l'image.

Pour cela, l'objet doit aussi « être vu » par les capteurs infrarouges pour que la fonction Reflex puisse connaître la distance à l'objet. Ceci implique que l'objet doit être situé dans une zone de l'image où les capteurs peuvent le détecter.

Fig. III.8 Zone de détection

III.4.1 Représentations de l'image

Le programme *ServerCapture* permet d'acquérir une image en bitmap (carte de bits) et codifiée en 8 ou 24 bits. L'image est représentée en échelle de gris si elle est codifiée en 8 bits (256 niveaux) ou en couleur en mode RGB si elle est codifiée en 24bits.

La fonction *Recognize* nécessite que l'image soit représentée en mode RGB. Cette mode représente les pixels comme une combinaison de trois images monochromes rouges, vertes et bleues (Red-Green-Blue).

Pour réduire le coût du traitement de l'image au niveau du processeur et, en tenant compte que les images sont envoyées depuis le robot à l'ordinateur par la liaison wireless (maximum bande passante de 11Mpbs), après différents tests, on a établi une résolution de l'image de 180 x 140 pixels (75600 bytes par image). Une résolution majeure réduirait sensiblement le flux d'images par second (inférieur à 5).

III.4.2 Segmentations de l'image

Segmenter une image consiste à la décomposer en plusieurs composants pour en extraire des informations importantes pour une analyse postérieure. Ces segmentations dépendent de l'application ; elles peuvent être par niveaux de gris ou de couleur, par régions, texture ou forme.

La fonction *Recognize* réalise une segmentation simple qui consiste à éliminer les pixels qui ne sont pas de niveau rouge. Pour cela, il réalise les opérations suivantes :

- Premier, calcule de la luminance dans le pixel défini :

$$\text{Luminance } Y = R + G + B$$

- Si la luminance du pixel est supérieur à 13% (en dessous de ce niveau, on considère que le pixel est de couleur noire), on calcule le pourcentage de niveau de rouge en ce pixel défini :

$$r = R/Y$$

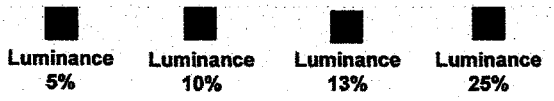


Fig. III.9 Types de luminance

- Si le pourcentage est supérieur à 0.4, on garde son composant R sinon on le garde comme pixel blanc.

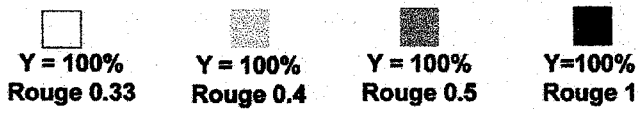


Fig. III.10 Pourcentage de rouge (r).

L'algorithme de segmentation parcourt pixel à pixel l'image, gardant le résultat dans une matrice de la même taille que l'image (180 x 140).

Pour réaliser les exemples suivants, on a implémenté l'algorithme de segmentation dans Matlab :

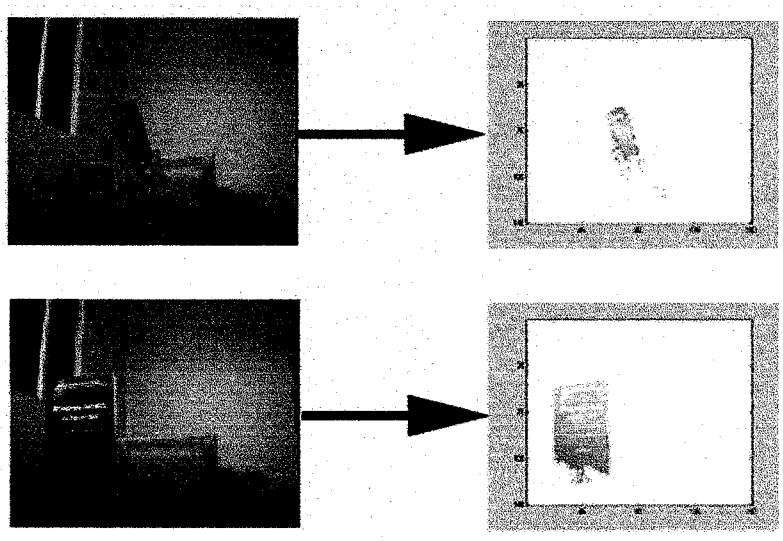


Fig. III.11 Exemples de segmentation

III.4.3 Optical Flow

Une fois éliminée de l'image les pixels qui ne contenaient pas un niveau de couleur rouge, on applique l'algorithme Optical Flow.

Lorsque les objets bougent devant la camera, ou lorsque la camera bouge à travers d'un environnement fixe, il y a des changements dans l'image. Ces changements permettent de reconnaître les mouvements relatifs et les formes des objets.

Le mouvement apparent des points de l'image, qui se produit lorsque la camera bouge par rapport aux objets centrés, se dénomme Optical Flow (flux optique). L'Optical Flow est basé sur l'Équation de Contention :

$$I_x \cdot u + I_y \cdot v + I_T = 0$$

$$I_x = \frac{dI}{dx},\ I_y = \frac{dI}{dy},\ I_T = \frac{dI}{dt} \qquad u = \frac{dx}{dt},\ v = \frac{dy}{dt}$$

Où I(x, y, t) est l'intensité du point (x, y) de l'image à l'instant t. Après un petit intervalle de temps dt, ce point sera en (x+u·dt, y+v·dt) conservant son intensité.

$$I(x,y,t) = I(x+dx, y+dy, t+dt)$$

Le champ de vitesses assigne un vecteur de vitesse à chaque point de l'image. A un instant déterminé, un point p_i, de vitesse v_i, dans l'image correspond à un point p_0, avec vitesse v_0, de la superficie d'un objet.

Le champ de vitesse ne peut pas se calculer directement, sinon à travers l'Optical Flow.

Ils existent différentes méthodes et algorithmes pour calculer l'Optical Flow (Horn & Schunck, Luc & Kanade, Anandan,..) plus ou moins complexes. Dans la bibliographie et références de ce projet vous pouvez trouver plusieurs sites Internet sur Optical Flow.

Dans ce projet, on utilise un algorithme Optical Flow très simplifié mais qui donne des résultats acceptables. Pour cela, on se base sur les suppositions suivantes:

- En premier lieu, on considère l'illumination constante sur toute l'image.
- Les points proches dans le plan de l'image bougent d'une manière similaire (restriction à des vitesses lentes).

De cette manière, on s'assure que l'image est différenciable d'une autre.

La détermination du Optical Flow est basée sur les méthodes d'itération Gauss-Seidel utilisant des paires (consécutifs) d'images. Le vecteur de vitesse dans un point se définit comme :

$$u^{k+1} = u^k - \frac{I_x\left[I_x \cdot u^k + I_y \cdot v^k + I_T\right]}{\alpha^2 + I_x^2 + I_y^2}$$

$$v^{k+1} = v^k - \frac{I_y\left[Ix \cdot u^k + I_y \cdot v^k + I_T\right]}{\alpha^2 + I_x^2 + I_y^2}$$

Où k indique le nombre d'itérations. Ces formules peuvent être simplifié si on réalise une itération pour chaque paire d'images.

$$v \cong u = \frac{-I_T}{I_x + I_y}, \qquad \text{si } I_x \cong I_y$$

III.4.3.1 Différenciations. Calcul du gradient

La méthode la plus commune de différenciation dans les applications de traitement de l'image est le gradient. Pour une fonction *f(x, y)*, le gradient de f de point de coordonnées *(x, y)*, on définit le vecteur :

$$\nabla F = \begin{bmatrix} \frac{dF}{dx} \\ \frac{dF}{dy} \end{bmatrix}$$

, et le module de ce veteur,

$$\nabla F = \max(\nabla F) = \left[\left(\frac{dF}{dx}\right)^2 + \left(\frac{dF}{dy}\right)^2\right]^{\frac{1}{2}}$$

Soit une image, où les différents sous indices d'une variable z indiquent les différentes valeurs des niveaux de gris :

Z1	Z2	Z3
Z4	Z5	Z6
Z7	Z8	Z9

L'équation permet de faire une approximation autour du point *Z5* de plusieurs manières. Les masques avec un nombre pair d'emplacements sont plus compliqués à implémenter. Une approximation à l'équation antérieure dans le point *Z5*, utilisant une matrice 3x3, est :

$$\frac{dF}{dx} = (Z_7 + Z_8 + Z_9) - (Z_1 + Z_2 + Z_3)$$

$$\frac{dF}{dy} = (Z_3 + Z_6 + Z_9) - (Z_1 + Z_4 + Z_7)$$

La différence entre la troisième et la première ligne de la région 3x3 donne une approximation de la dérivée dans la direction *x*, et la différence entre la troisième et la première colonne donne une approximation de la dérivée dans la direction *y*.

Pour implémenter lesdites dérivées, on peut utiliser différents masques connus. Dans ce cas, nous utiliserons les masques de Sobel, la première implémente la dérivée dans l'axe *x* en fonction du pixel central, et la seconde, la dérivée dans l'axe *y*.

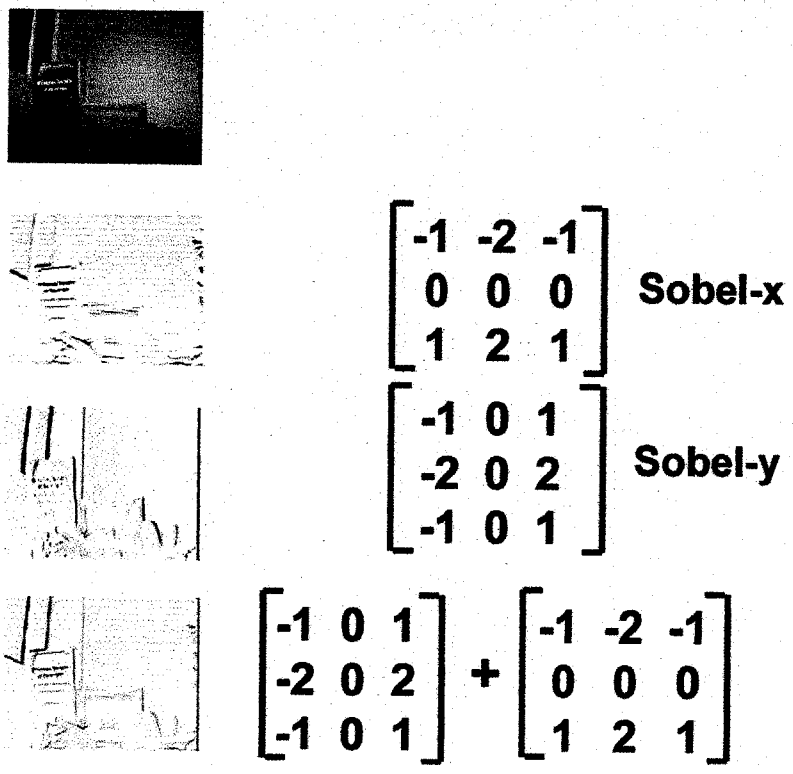


Fig. III.12 Exemples masques Sobel

Le gradient dans le temps s'obtient facilement en réalisant une soustraction entre les deux images.

III.4.4 Déplacements de Pekee

Une fois obtenue les vecteurs de vitesses, on doit déterminer comment calculer le vecteur de la vitesse nette de mouvement, vecteur qui servira d'entrée pour détecter où l'objet se trouve.

Pour détecter l'objet, on obtient deux vecteurs nets, un pour les vitesses en *x* et l'autre pour les vitesses en *y*. Ces vecteurs de vitesses nettes sont obtenus au travers d'un processus de quadrillage de l'image en plusieurs quadrants (5 x 5).

Après avoir réalisée le quadrillage, on procède au calcul de la moyenne de ses vitesses dans l'axe *x* et *y*.

De cette manière, on peut connaître le quadrant où se trouve la moyenne majeure de la vitesse et donc l'objet à suivre. Connaissant la zone de l'image, on peut indiquer à la fonction Reflex vers où il doit faire déplacer le robot.

Comme la camera est fixe sur le robot, le mouvement est un peu limité. Dans ce cas seulement, on réalise un déplacement pour centrer l'objet si celui-ci se trouve dans une zone concrète de l'image (un quadrant concret de l'image est un quadrant à la porté des capteurs infrarouges).

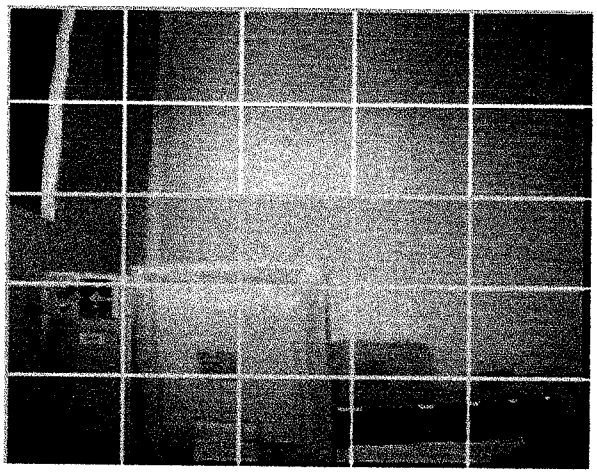


Fig. III.13 Quadrillage de l'image

III.4.5 La classe CRecognize

Le contenu de cette partie est basé sur la classe *CRecognize*. Un objet de cette classe permet l'analyse de l'image et l'extraction de l'information nécessaire pour permettre la détection des objets de couleur rouge et permettre à Pekee de centrer cet objet dans l'image.

La classe *CRecognize* inclut les méthodes suivantes :

- *Startup* : Cette méthode initialise la fonction Recognize.
- *Finalize* : Cette méthode arrête la fonction Recognize.
- *Threshold_r* : Cette méthode extrait les pixels rouges de l'image et il crée une couche qui démarque les pixels trouvés dans l'image.
- *Gradient* : Cette méthode calcule les contours verticaux et horizontaux d'une image utilisant l'opérateur Sobel.
- *Substract* : Cette méthode calcule le gradient en fonction du temps de deux images consécutives.
- *Op_flow* : Cette méthode calcule le vecteur de vitesses à partir des gradients de contours et du temps.
- *V_seg* : Cette méthode calcule les vecteurs de vitesses moyennes par quadrant divisant l'image en une matrice de 5 x 5.
- *Calcul_max* : Cette méthode retourne une valeur indiquant dans quel quadrant se trouve le vecteur de vitesses moyennes majeur.
- *Detect_d* : Cette méthode calcule vers quelle direction (droite ou gauche) doit bouger le robot à partir du quadrant qui contient le vecteur de vitesses moyennes majeur.
- *Detect_r* : Cette méthode indique si on a trouvé une surface minime de pixels de couleur rouge dans de la zone de détection.

III.4.6 La classe CSImage

Cette classe est basée à partir de la classe *WImage* réalisée par l'entreprise Wany. On a utilisé quelques méthodes et quelques variables de celle-ci et on a conçu d'autres méthodes nécessaires pour l'application.

Un objet créé à partir de cette classe permet de garder une image capturée par la camera et toute ses informations (taille, résolution, type,...), et de réaliser une série d'opérations avec elle. Les méthodes inclues dans la classe sont:

- *Display* : Cette méthode permet de visualises à l'écran l'image capturée.
- *ToBMP24* : Cette méthode convertit une image JPG en BMP avec 24bits de résolution.

- *ToJPG* : Cette méthode convertit une image BMP en JPG avec le niveau de qualité sélectionnée.

- *GetDy* : Cette méthode retourne les dimensions en pixels sur l'axe *Y* (haut) de l'image.

- *GetDx* : Cette méthode retourne les dimensions en pixels sur l'axe *X* (largeur) de l'image.

- *AddLayer* : Cette méthode ajoute à l'image capturée une couche qui démarque les pixels trouvés par la fonction Recognize de couleur rouge.
- *AddGrid* : Cette méthode ajoute à l'image un grillage qui divise l'image dans une matrice de 5 x 5.

III.5 Détections d'obstacles. La fonction Reflex

La fonction Reflex permet au robot de bouger en toute liberté. S'il trouve sur son chemin un obstacle, il tente de l'éviter suivant un algorithme concret. Cet algorithme contrôle aussi bien la vitesse que la direction à suivre par le robot. Le comportement est différent si la fonction *Recognize* est active. En détectant un objet de couleur rouge dans de la zone de détection, la fonction *Reflex* tentera de centrer l'objet dans l'image.

Les 15 capteurs infrarouges situés autour du robot Pekee sont basiques pour réaliser la détection des obstacles.

Ils existent des situations où les capteurs infrarouges ne sont pas efficients (des superficies qui absorbent ce type de signal). Par cela le robot incorpore un détecteur de collision. L'algorithme de la fonction *Reflex* prend en compte ce détecteur dans le cas où il se produit une collision avec un objet.

III.5.1 Configurations de la sensibilité des capteurs des infrarouges

La rampe de puissance d'émission des capteurs est composée de 26 valeurs. Chaque élément de cette rampe peut varier de 0 à 255. Cette rampe est totalement configurable, permettant ainsi d'ajuster la sensibilité des capteurs au type d'environnement où le robot se trouve.

Les capteurs envoient une information de tout ou rien indiquant la présence ou non d'un objet. La puissance d'émission dépendra de la valeur configurée dans la rampe de puissances. Mais s'il ne détecte aucun objet, le capteur retourne la valeur 99.

Dans ce cas, on suppose que le robot bouge dans une pièce où il se trouve une grande variété d'obstacles. Suite à différents testes, on a modifié la rampe de

puissance en configurant la portée maximale à 1 mètres divisés en 5 zones de détection.

Position	Valeur	Distance	Zone
0	4	0 - 15 cm	0
1	4		
2	4		
3	4		
4	4		
5	8	15 - 30 cm	1
6	8		
7	8		
8	8		
9	8		
10	13	30 - 45 cm	2
11	13		
12	13		
13	13		
14	13		
15	20	45 - 60 cm	3
16	20		
17	20		
18	20		
19	20		
20	30	60 - 100 cm	4
21	30		
22	30		
23	30		
24	30		
25	30		
99		100 - ∞	5

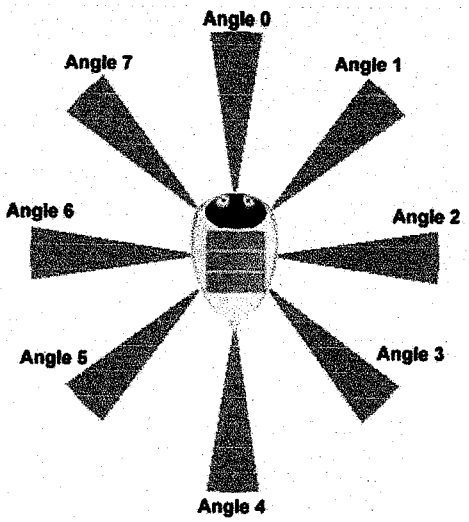
Fig. III.14 Configuration rampe des capteurs infrarouges

III.5.2 Algorithme

Les 15 capteurs s'ont groupé en 8 sections ou angles pour détecter d'une manière plus efficiente l'objet. Le robot réalisera des tours multiples de 45° permettant 8 manœuvres différentes.

La vitesse de déplacement dépendra de la distance à laquelle se trouve l'obstacle.

Les 8 angles de détection et les capteurs assignés sont placés suivant cette figure :



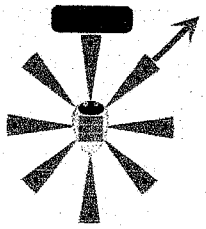
Angle	Capteurs
0	2,3,4,5
1	6,7
2	7,8
3	9,10
4	11
5	12,13
6	14,15
7	14,1

Fig. III.15 Angles de détections

Donc, on dispose de 6 zones de détection de distances et de 8 angles de détection.

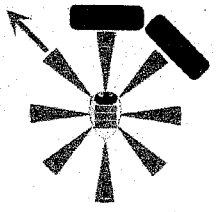
III.5.2.1 Contrôles de direction

Le robot bouge en ligne droite jusqu'à ce qu'il détecte un obstacle frontal. A ce moment si l'obstacle se trouve dans une zone supérieure à 1(distance supérieure à 30cms) tente de trouver quel est le meilleur manoeuvre de contournement. 8 cas distincts peuvent se présenter:



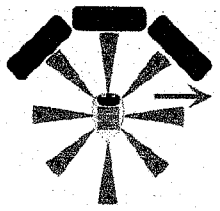
Cas 1

Seulement un obstacle frontal est détecté ou les autres obstacles sont à plus de 30cms (zone 2 ou majeur). L'algorithme décide de tourner 45° à la droite.



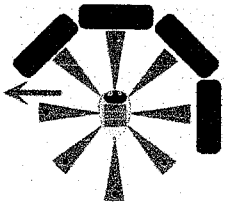
Cas 2

Entre autre de l'obstacle frontal, Pekee détecte un autre dans l'angle 1. L'algorithme décide de tourner 45 ° à la gauche.



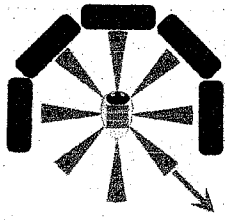
Cas 3

Il détecte des obstacles dans les angles 7, 0 et 1. L'algorithme décide de tourner 90° à la droite.



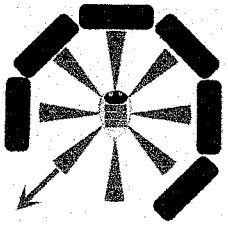
Cas 4

Il détecte des obstacles dans les angles 7, 0, 1 et 2. L'algorithme décide de tourner 90° à la gauche.



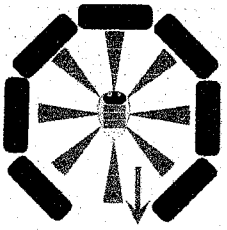
Cas 5

Il détecte des obstacles dans les angles 6, 7, 0, 1 et 2. L'algorithme décide de tourner 135° à la droite.



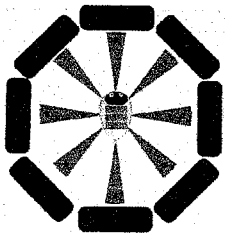
Cas 6

Il détecte des obstacles dans les angles 6, 7, 0, 1, 2 et 3. L'algorithme décide de tourner 135° à la gauche.



Cas 7

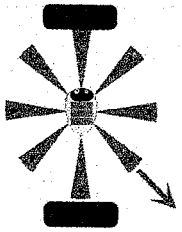
Il détecte des obstacles dans tous les angles hormis en 4. L'algorithme décide de tourner 180° à la droite.



Cas 8

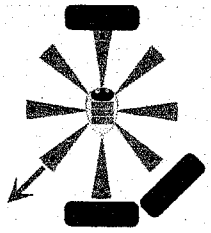
Situation de blocage, il détecte des obstacles dans toutes les directions. L'algorithme décide d'arrêter le robot.

Si l'obstacle se trouve dans la zone 0 ou 1 (moins de 30cms) le robot doit faire marche arrière puisqu'il n'aurait pas suffisant d'espace pour réaliser un demi-tour. Si en reculant, il détecte un objet derrière lui qui se trouve dans une zone supérieure à 1 (plus de 30cms), il doit réaliser un tour en reculant. L'algorithme prévoit donc 5 cas :



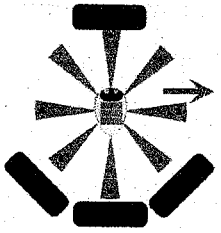
Cas 1

Il détecte seulement un obstacle. L'algorithme décide de tourner en reculant 45° à la droite.



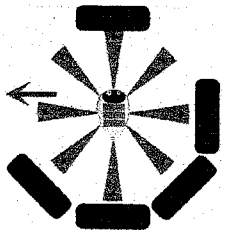
Cas 2

Il détecte des obstacles dans les angles 4 et 3. L'algorithme décide de tourner en recul 45° à la gauche.



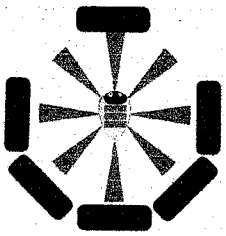
Cas 3

Il détecte des obstacles dans les angles 5, 4 et 3. L'algorithme décide de tourner en reculant 90° à la droite.



Cas 4

Il détecte des obstacles dans les angles 5, 4, 3 et 2. L'algorithme décide de tourner en reculant 90° à la gauche.



Cas 5

Situation de blocage : il détecte des obstacles dans toutes les directions. L'algorithme décide d'arrêter le robot.

Si, en reculant, il détecte un objet derrière lui à une distance inférieure à 30cms : le robot se trouve dans une situation où il détecte un obstacle frontal et un autre derrière lui à très peu distance. L'algorithme décide de faire un tour de 90° sans vitesse dans le sens où il ne détecte pas obstacle.

III.5.2.2 Contrôles de vitesse

La vitesse du robot est déterminée en fonction de la distance des obstacles détectés. S'il ne détecte aucun objet ou s'il en trouve un à plus de 45cms (zone

3 ou supérieur), il augmente la vitesse. Plus la distance entre le robot et l'objet sera grande, plus cet incrément de vitesse sera important.

Si un obstacle est détecté à une distance inférieure à 45cms, la vitesse est réduite. Dans le cas où l'objet est détecté à une distance inférieure à 15cms, l'algorithme arrête au robot.

On a limité la vitesse à 0.3 m/s, la limite de vitesse peut être choisi suivant 15 valeurs différentes.

Donc, le robot varie sa vitesse en fonction de sa distance à un objet. Après différents tests d'ajustage, l'algorithme de vitesse répond de la manière suivante :

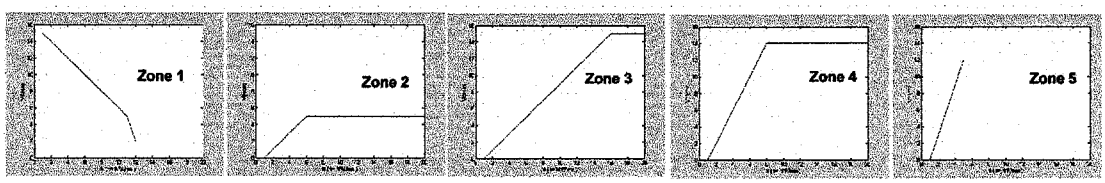


Fig. III.17 Réponse en fonction de la distance au l'objet

III.5.3 Mouvements de Pekee. Les fonctions de niveau bas

Les fonctions de niveau haut implémentées pour obtenir un mouvement du robot utilisent les fonctions de niveau bas que agissent directement sur les moteurs.

Fondamentalement, la fonction de niveau bas utilisé est l'ordre *FK_SETMOVE_SPEEDANDSTEERING*, commentée dans le chapitre 1.

Les paramètres de cet ordre sont la vitesse de déplacement et la vitesse de tour. Les deux paramètres peuvent varier entre -22 et +22. Wany recommande d'utiliser des valeurs entre -15 et 15 pour la vitesse de déplacement et entre -10 et 10 pour la vitesse de tour.

Le robot inclut un système de protection des moteurs. Si la protection est activée, une consigne de déplacement (consigne vitesse/direction) est valable par défaut pendant 10 fois la base de temps du Pekee (63 ms). Si aucune autre commande de déplacement n'est intervenue dans cet intervalle, la base mobile stoppe les moteurs et enclenche l'asservissement moteur.

Si la protection n'est pas active, la base applique les commandes de déplacement telles qu'elles sont.

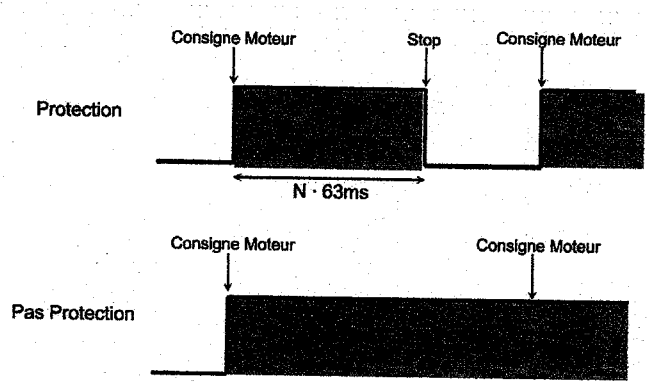


Fig. III.18 Modes de fonctionnement

Dans l'application développée pour ce projet, on utilise la protection des moteurs. La raison principale est que, dans le cas d'une erreur de connexion entre l'application et le robot, celui-ci doit s'arrêter et, donc, ne pas continuer à se déplacer.

Le délai avant STOP peut être modifié par l'ordre *FK_HARDFUNCTIONS_TYPE_MOVECOMMANDPROTECTIONTIME*.

Pour pouvoir réaliser les tours dans des multiples de 45°, l'intervalle d'exécution varie, conservant la vitesse de tour constante à 5. Après plusieurs tests, les valeurs appropriées pour réaliser le mouvement sont les suivants :

Angle	Vitesse	Intervalle = N · 63ms
45°	± 5	N = 14
90°	± 5	N = 28
135°	± 5	N = 42
180°	± 5	N = 56
225°	± 5	N = 70
270°	± 5	N = 84
315°	± 5	N = 98
360°	± 5	N = 112

Fig.III.19 Tableau d'angles

III.5.4 La classe CReflex

Cette partie est axée sur la classe CReflex. Un objet de cette classe permet de contrôler le déplacement du robot et la détection des obstacles qui se trouvent sur sa trajectoire.

La classe CReflex inclut les méthodes suivantes :

- *Startup :* Cette méthode va appeler, au début, la fonction Reflex.

- *Finalize :* Cette méthode arrête le robot et arrête la fonction Reflex.

- *Front :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 0.

- *F_Right :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 1.

- *F_Left :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 7.

- *Right :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 2.

- *Left :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 6.

- *B_Right* : Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 3.

- *B_Left :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 5.

- *Back :* Cette méthode retourne dans quelle zone de détection (0 à 5) se trouve un objet situé dans l'angle 4.

- *Control* : Cette méthode génère la consigne de mouvement (vitesse et tour) en fonction de l'algorithme précédemment commenté.

- *MoveR* : Cette méthode fait appels aux fonctions de niveau bas qui agissent sur les moteurs.

- *Stop :* Cette méthode arrête le robot.

III.6 Mouvements de Pekee par clavier. La fonction Keyboard

La fonction Keyboard permet à l'usager de contrôler le déplacement du robot. La fonction observe s'il a appuyé sur les touches assignées au contrôle et il ordonne le mouvement. Ils existent 6 touches assignées :

- Curseur en haut : Si elle est pressée, le robot se déplace vers le devant.

- Curseur en bas : Si elle est pressée, le robot se déplace vers l'arrière.

- Curseur droit : Si elle est pressée, le robot réalise un tour à droite.

- Curseur Gauche : Si elle est pressée, le robot réalise un tour à gauche.

- Touche 'A' : Si elle est pressée, le robot augmente sa vitesse.

- Touche 'Z' : Si elle est pressée, le robot diminue sa vitesse.

III.6.1 La classe CKeyboard

La classe CKeyboard est destinée à implémenter la fonction Keyboard. Un objet créé à partir de cette classe permet de reconnaître qu'elle touche a été pressée et donc, quel mouvement doit réaliser le robot. Une méthode est suffisante pour réaliser cette fonction :

- *Get_Speed_Steering* : Cette méthode retourne la consigne de direction et de tour qui doit être envoyée au robot en fonction des touches pressées.

III.7 Visualisations de l'information générées par Pekee

En plus des fonctions précédemment expliquées, l'application *PekeeControl* montre à l'écran les informations des capteurs, des détecteurs ou des indicateurs incorporés en Pekee. Pour faciliter la compréhension de ces informations par l'usager, en fonction de chaque dispositif il a été réalisé une conversion numérique ou graphique des informations.

III.7.1 Vitesse du robot

L'information de vitesse de déplacement de Pekee est extraite à partir de différentes données. Les odomètres incorporés dans le robot donne le nombre de tours réalisés par les roues et, connaissant le diamètre de celles-ci, on peut trouver une formule qui détermine la vitesse du robot.

Selon les spécifications de Wany, les odomètres comptent 180 impulsions par tour. On obtient l'information du nombre d'impulsions et le temps associé à ce nombre d'impulsions en même temps (dans des paquets de 63ms).

Avec ces données, on peut en déduire l'expression suivante :

Si le diamètre de la roue est de 7.2 cm,

$$p = \pi \cdot d = 0.226m$$

Sachant que un tour équivaut à 180 impulsions :

$$V = \frac{e}{t}$$

Pour trouver l'espace parcouru, on connaît le nombre d'impulsions (moyenne des deux roues),

$$e = \frac{N}{180} \cdot 0.226m$$

Le temps passé est aussi connu,

$$t = x \cdot 63ms$$

x étant le nombre de paquets 63ms.

Donc,

$$V = \frac{NP \cdot 0.226}{180 \cdot 63ms} = NP \cdot 0.0199 \ (\mathrm{m/s})$$

Avec

$$NP = \frac{N}{x}$$

III.7.2 Température extérieur et intérieur

Pekee incorpore un capteur de température extérieure et un capteur de température intérieure. Wany fournit dans ses spécifications un tableau de correspondances entre la valeur calculée par le convertisseur analogique - numérique du microcontrôleur du module PC embarqué et la température en degrés Celsius.

Le tableau comprend des valeurs de -40 °C à 85 °C par pas de 1°C (136 valeurs).

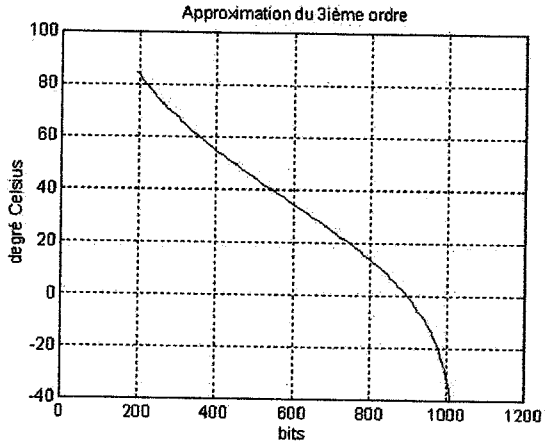


Fig. III.20 Graphique de température

Pour réaliser la conversion en degrés Celsius sans avoir à inclure le tableau de correspondances dans le programme *PekeeControl*, on a calculé moyennant Matlab une fonction qui répond aux valeurs du tableau.

Selon les spécifications de Wany, même si les valeurs du tableau des capteurs sont comprises entre -40 °C et 85 °C, le robot est prévu pour fonctionner entre +10 °C et 40 °C. Donc pour le calcul de la fonction, on peut s'appuyer sur une marge mineure des températures du tableau.

La fonction trouvée grâce à Matlab prenant en compte les températures de 0°C à 50°C est la suivante :

$$T(x) = -2.2637 \cdot 10^{-7} x^3 + 4.0392 \cdot 10^{-4} x^2 - 0.3358x + 139.6308$$

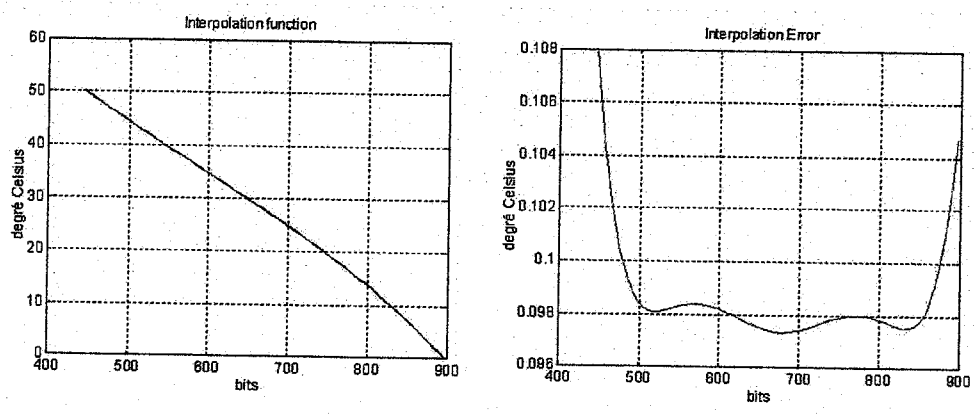


Fig. III. 21 Graphiques Fonction interpolation de température

III.7.3 Vitesse et retard des images

Le programme *PekeeControl* montre à l'écran le nombre d'images par second et le retard existant enter chaque images.

Pour calculer le nombre d'images par seconde, il existe un compteur d'images qui va en incrémentant. L'interruption de gestion de données se charge de diviser la valeur du compteur par le temps passé pour connaître le nombre d'images par seconde.

Le retard est défini comme le temps passé entre la demande de l'image de la part du programme *PekeeControl* et la réception de celle-ci. Le retard montré à l'écran est la moyenne de l'intervalle utilisée pour calculer le nombre d'images par seconde.

A partir de ces données, on a réalisé différentes tests pour connaître la vitesse et le retard des images en fonction du type de connexion (LAN ou Wireless) et du type d'images.

Le tableau suivant en montre le résultat.

Résolution (pixels)	Image	Wireless (11Mbps)		LAN (100Mbps)		Size Bytes
		Image/s	Delay (ms)	Image/s	Delay (ms)	
180 x 140	BMP 24 bits	6	170	6	170	75600
180 x 140	BMP 8 bits (B&W)	13	75	13	74	25200
180 x 140	JPG (24bits) 10%	4	267	4	264	1600
180 x 140	JPG (24bits) 20%	4	267	4	268	2100
180 x 140	JPG (24bits) 50%	3,7	275	3,7	274	3000
180 x 140	JPG (24bits) 80%	3,7	294	3,7	290	5000
180 x 140	JPG(24bits) 100%	2,3	480	2,3	430	23000

Fig.III.22 Tableau de résultats. La vitesse et le retard en fonction du type de connexion.

Bien que la bande passante est majeure dans une connexion LAN (100Mbps) que dans une connexion Wireless (11Mbps), la vitesse et le retard des images sont très proches dans les deux types de connexion. Cette circonstance est due à la faible capacité du processeur du module PC embarqué (486). Également, lorsque les images sont compressées à JPG le processeur doit consommer plus temps d'exécution et donc il diminue le taux de vitesse et il augmente le retard.

III.7.4 Capteurs infrarouges

Ces capteurs sont chargés de mesurer la distance qui existe entre le robot et l'objet détecté. Comme expliqué précédemment, les capteurs infrarouges s'ont configurés pour détecter 6 zones distinctes. Le programme *PekeeControl* montre les 15 capteurs autour de Pekee. La zone détectée par le capteur est indiquée par une couleur distincte.

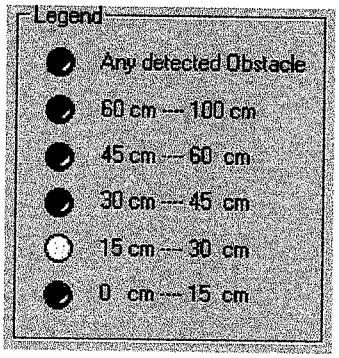


Fig. III.23 Légende des capteurs

III.7.5 Capteur de Lumière

Le capteur de lumière est situé juste au milieu des deux capteurs infrarouges frontaux. *PekeeControl* montre le valeur envoyée par le microcontrôleur (de 0 à 1024) via un icône rectangle dont la couleur varie en fonction de la luminosité.

III.7.6 Niveaux de Batteries

Ils existent deux systèmes de batteries différents. Une située dans le module PC embarqué et qui alimente ce dernier, et l'autre située dans le robot même et qui alimente la base mobile (moteurs, capteurs, microcontrôleur,...) et le module wireless.

Les niveaux de batteries des deux systèmes sont montrés dans *PekeeControl* de manière graphique par des barres verticales. La couleur de celles-ci varie en fonction de l'état des batteries : bleu si elles sont pleines ou moyennement pleines et rouge lorsque les batteries sont basses (coïncidant avec l'avertissement sonore du robot).

III.7.7 Position du Robot

La position est envoyée et actualisée directement par la base mobile, en mètres pour les axes X et Y et en radians pour la composante Teta (convertie en degrés par le programme pour être visualisée). L'usager peut initialiser la position du robot grâce au programme.

III.8 Aspect final de l'interface graphique

L'image suivante montre l'aspect final du programme *PekeeControl*.

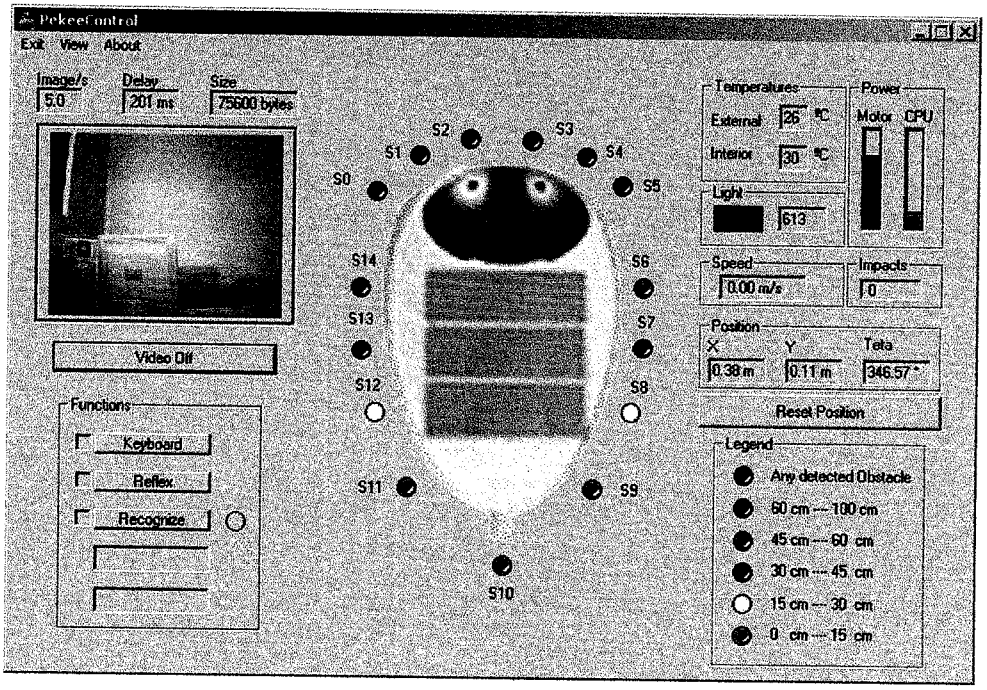


Fig. III.24 Boîte principal du programme PekeeControl

III.8.1 Information sur Pekee

Le programme inclut une option pour connaître les informations générales sur le robot (version du software, nommez du produit...) et sur le programme.

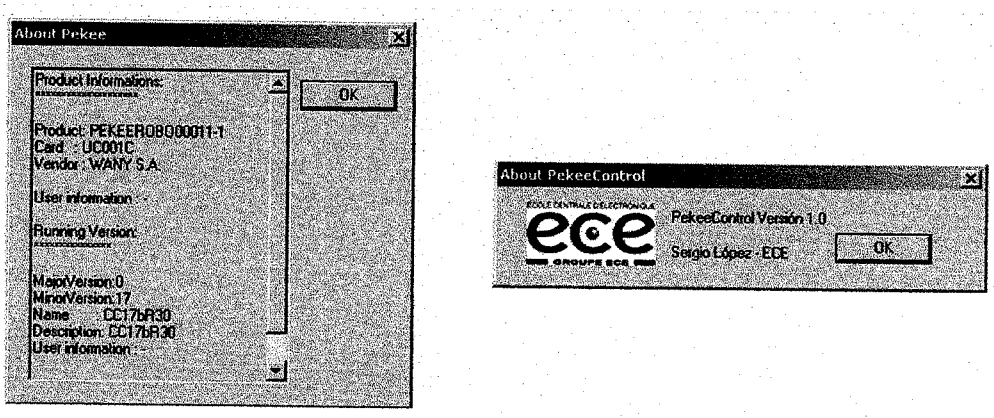


Fig. III.25 Boîtes « About Pekee » et « About PekeeControl »

CONCLUSIONS

Le robot autonome Pekee développé par l'entreprise Wany Robotics est un excellent outil pour la réalisation de différentes applications dans le domaine de la robotique. La variété des capteurs et des dispositifs inclus, ainsi que la possibilité de communiquer sans fils moyennant le module wireless et la réception d'images moyennant la camera incorporée au module PC ouvrent un grand éventail de possibilités.

L'interface graphique développée pour ce projet m'a permis d'approfondir, avancer ou commencer à connaître les différentes matières tel que la programmation orientée objets et la gestion de processus en temps réel (Visual C++), la communication sans fils ou le traitement de l'image. Chacune de ces matières peut être le thème d'un projet dû à leur énorme complexité et contenu.

Cependant, l'objectif principal, au niveau personnel, n'a pas été le développement de l'application sinon le chemin parcouru jusqu'à l'aboutissement du projet : on a obtenu un programme pratique et utile pour le contrôle, le suivi et la supervision du robot Pekee.

Néanmoins, le programme peut être amélioré. Grâce à la modularité que permet la programmation orientée objets et l'utilisation des classes, les modifications pour augmenter ou améliorer les fonctions développées peuvent se faire d'une manière simple. Toutes les fonctions importantes réalisées (Reflex, Keyboard, Recognize ou Serveur / Client) sont générées à partir d'une classe indépendante. De cette manière, la majorité des modifications, que se soient aux niveaux des méthodes ou des fonctions qui sont basées sur des classes indépendante, n'affectera pas le programme principal.

Une des voies à explorer pour améliorer le programme pourrait être l'algorithme de reconnaissance d'objets. L'algorithme développé dans ce projet travaille seulement avec des objets de couleur rouge et à un niveau simple de traitement de l'image. Il existe une multitude d'information tant dans des livres que sur Internet concernant le traitement d'images et ses applications possibles : reconnaissance de la peau humaine, de textures, de contours, suite, etc.

Une autre voie serait l'incorporation d'une fonction à plus haut niveau que la fonction Reflex qui permettrait la planification de la trajectoire du robot. La fonction Reflex permet au robot d'éviter les obstacles présents sur son chemin mais ce chemin n'est pas déterminé. Une nouvelle fonction qui supervise la position du robot et qui trouve le chemin optimal est une amélioration possible qui pourrait être incorporé à l'application.

Au niveau de la communication, on peut exploiter les possibilités de la technologie wireless. Une application possible serait la communication entre plusieurs robots Pekee pour développer une unique tâche et pouvoir échanger des informations (par exemple la reconnaissance d'un terrain étendu ou la surveillance d'un lieu).

Une autre application serait de disposer d'un robot et de plusieurs ordinateurs pour le contrôler. De cette manière, on pourrait augmenter la zone de communication en disposant de différents points de connexion. Logiquement, les applications ne se travaillerait pas en mode AdHoc (connexion pointe à point) mais en mode Infrastructure (connexion Serveur - Cliente).

A partir des tests réalisés, on peut déduire qu'une amélioration possible serait au niveau hardware du robot par l'incorporation de capteurs infrarouges à différents niveaux de hauteur pour améliorer la détection d'obstacles qui empêchent le déplacement du robot. Aussi la possibilité de pouvoir faire tourner la camera de 360º à volonté serait un plus au robot Pekee.

BIBLIOGRAPHIE

(1) Richard C.Leinecker et Tom Archer, *Visual C++ 6 Biblie*, Anaya, (1999)

(2) *Manuel d'utilisation Pekee,* Wany Robotics, (2002)

(3) *Manuel OPP Function Keys,* Wany Robotics, (2002)

(4) *Manuel DLL OPP Access,* Wany Robotics, (2002)

(5) *User's Manual Video Developement Kit*, Wany Robotics, (2002)

Webs d'intérêt

Sur robot Pekee

www.wanyrobotics.com

www.pekee.fr

Sur Wireless

www.wireless-fr.org

www.wireless-info.org

Sur Optical Flow

www.cs.otago.ac.nz/research/vision/Research/OpticalFlow/opticalflow.html

www.ai.mit.edu/people/lpk/mars/temizer_2001/Optical_Flow

www.inf.utfsm.cl/~romina/Bajar_Romina/Proyecto_WebCam.pdf (espagnole)

Sur Traitement d'images

www.ccr.jussieu.fr/urfist/image_numerique/Image_numerique1.htm

www.ing.ula.ve/~abravo/document/tutorial/imagenes/indice.html#c43 (espagnole)