2008-2009
Arts et Métiers ParisTech
Paris

Accès libre

Fusion de données : Application à un robot mobile. Le robot HAMMI

Etudiant :

SÁEZ CARDADOR Javier

Professeur :

VERGÉ Michel

ANNEE : 2009

Groupe du PJE : FISE

Numéro du PJE : PA-F09145

CENTRE DE RATTACHEMENT PJE : Paris

AUTEUR : SÁEZ CARDADOR Javier

TITRE : Fusion de données : Application à un robot mobile. Le robot HAMMI.

ENCADREMENT DU PJE : VERGÉ Michel ; LMSP

PARTENAIRE DU PJE : HanditecAM

NOMBRE DE PAGES : 55

NOMBRE DE REFERENCES BIBLIOGRAPHIQUES : 13

RESUME : Le robot mobile et autonome d'assistance aux personnes handicapées appelé HAMMI est conçu pour pousser n'importe quel fauteuil roulant. Après plusieurs études, l'objectif est maintenant d'intégrer les données fournies par des capteurs de distance infrarouges et ultrasons pour obtenir des renseignements utiles agissant sur les trajectoires. On a réalisé plusieurs essais pour conditionner les capteurs. La programmation a été effectuée avec Visual C++ de Microsoft et avec les bibliothèques des cartes National Instrument. Le but est d'obtenir des données de distances propres et leur estimation à partir de la trajectoire que le robot suit et de l'environnement. A la fin du projet, tous les outils et données doivent être mis en place pour l'implantation d'un futur filtre de Kalman.

MOTS-CLES : Robotique mobile, personnes handicapées, Visual C++, fusion données, acquisition de données et filtre de Kalman.

PARTIE A REMPLIR PAR LE PROFESSEUR RESPONSABLE DU PROJET

ACCESSIBILITE DE CE RAPPORT (entourer la mention choisie) :

Classe 0 = accès libre

Classe 1 = Confidentiel jusqu'au _ _ _ _ _ _ _ _ _

Classe 2 = Hautement confidentiel jusqu'au _ _ _ _ _ _ _ _ _ (date de fin de confidentialité)

Date : Nom du signataire : Signature :

REMERCIEMENTS

Je voudrais commencer par remercier toute ma famille pour son soutien inconditionnel. Dans le lointain, elle a toujours été près de moi, de mes bonheurs et de mes malheurs. Sans son effort, je n'aurais jamais réussi dans mes études.

Je remercie également M. Vergé pour son encadrement dans ce projet. Il a montré une grande patience, une passion pour transmettre ses connaissances et une proximité qui m'ont permis de développer mon projet dans une ambiance très propice et agréable.

Je ne pourrai jamais oublier Mme. Cliton qui a été toujours disponible et qui m'a aidé à m'intégrer dans l'école et dans la vie parisienne. De même, elle a dédié tout son effort et son temps à m'offrir un enseignement du français le plus complet possible.

Dans cette ligne, je tiens aussi à remercier tant d'autres professeurs et professionnels de l'ENSAM Paris qui ont contribué à que mon séjour devienne plus facile.

Finalement, je remercie tous les élèves du laboratoire qui ont travaillé avec moi cette année, en m'apportant des bons moments et leur aide en ce qui concerne le français et mon intégration dans l'école.

SYNTHÈSE EN ESPAGNOL

ÍNDICE

ÍNDICE DE FIGURAS

1. INTRODUCCIÓN.

Desde siempre, la tecnología ha estado enfocada al servicio de la humanidad y está destinada a satisfacer sus necesidades. Por lo tanto, es de esperar que a medida que la tecnología evolucione, será puesta a disposición de las personas discapacitadas. Este es el fin de este proyecto.

En el año 2007, la población francesa con una deficiencia motriz sobrepasaba el 10%, llegando al 21% si se consideraban otras deficiencias. En el caso de la comunidad europea, los datos son parecidos. Queda el presente proyecto inscrito en este marco. Si se quiere profundizar más en estos datos se puede consultar la referencia bibliográfica [HDP07].

De tal forma, el LMSP (Laboratorio de Mecánica de Sistemas y Procesos), perteneciente al ENSAM e interesado en hacer una contribución a este campo, propuso el proyecto llamado Robótica Móvil Para Personas Discapacitadas. Siempre con la colaboración de la sociedad HandiTecAM, el fin de este proyecto es el de diseñar una plataforma de desarrollo que permita ayudar a las personas con deficiencias motrices o sensoriales.

Dentro de este gran proyecto desde hace tres años, se desarrolla el robot HAMMI (HandiTec Arts et Métiers Motorización Inteligente). Se trata de un prototipo de robot autónomo que, acoplado detrás de una silla de ruedas, será capaz de seguir las instrucciones del ocupante o desplazarse automáticamente. Gracias a un brazo robótico, el ocupante podrá también manipular objetos.

1.1. El robot HAMMI.

En este epígrafe se presentará el estado del robot HAMMI al comienzo de este proyecto. Dicha presentación se estructurará según dos partes: primero los componentes mecánicos y eléctricos y seguidamente, los componentes electrónicos.

La primera parte es necesaria para tener una concepción global sobre el robot. La segunda parte es la más relacionada con el presente proyecto y por lo tanto, es imprescindible para entender el trabajo realizado.

Parte mecánica y eléctrica.

Gracias a los proyectos realizados en los últimos años por varios equipos de trabajo sobre el robot HAMMI, se dispone de una estructura autónoma móvil capaz de unir y portar los componentes necesarios para el funcionamiento adecuado del sistema.

Los elementos más importantes y sus funciones se exponen en la siguiente tabla:

COMPONENTE	DESCRIPCIÓN	FUNCIÓN
Estructura de soporte	Conjunto de segmentos de perfil estructural	Sirve de soporte para los otros componentes y asegura la estabilidad
Ruedas motrices	Ruedas laterales con motores eléctricos y encoders	Desplazamiento del robot y control de las trayectorias del mismo
Fuente de alimentación	Alimentación regulada 24 v.	Alimentar toda la electrónica excepto el ordenador embarcado
Ventosas magnéticas	Electroimanes cilíndricos	Transmisión de fuerzas a la silla
Cintura para sensores	Soporte metálico mecanizado	Albergar los sensores de distancia
Soporte para cámara	Vástago y base mecanizados	Albergar la cámara de video

Fig. 1.1: Tabla Resumen de Componentes Mecánicos y Eléctricos

A continuación se muestra una figura donde se pueden identificar los elementos anteriormente mencionados:

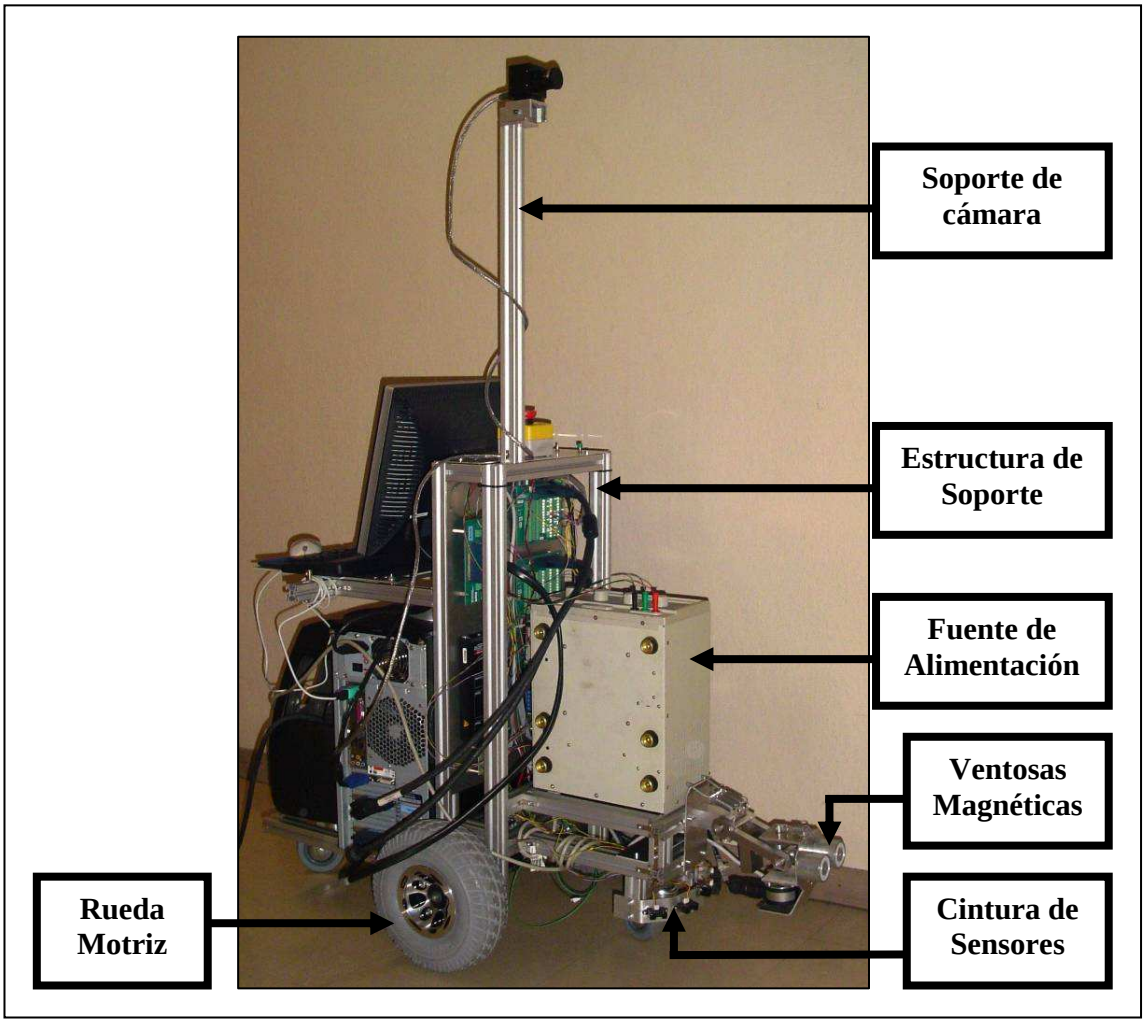

Fig. 1.2: Vista General de los Componentes Mecánicos y Eléctricos

En esta imagen ya se pueden apreciar los elementos electrónicos que se comentarán a continuación, entre los que resalta el ordenador embarcado. Pero antes de llegar a ese punto, nótense algunos detalles como las ruedas no motrices, que ayudan a soportar la estructura o la alimentación general a través de una línea de 220 v. habitual.

Parte electrónica.

Igualmente que en el apartado anterior, del trabajo de los años precedentes, al comienzo de este proyecto el robot disponía de los elementos resumidos en la siguiente tabla:

COMPONENTE	DESCRIPCIÓN	FUNCIÓN
Ordenador	PC de propósito general con todos sus periféricos	Alberga las tarjetas de adquisición y ejecuta los programas de gobierno
Tarjetas de adquisición de datos	Componentes informáticos que digitalizan información	Obtención y generación de todas las señales correspondientes al robot
Variadores de frecuencia	Componentes que actúan como drivers de los motores	Regulan la velocidad de los motores de las ruedas motrices
Cámara de video	Sensor de imágenes	Captura de imágenes para conocer el entorno del robot
Sensores de distancia	Elemento capaz de obtener la distancia a un objeto	Obtener la distancia a los diferentes obstáculos del entorno del robot
Sensor angular	Elemento que obtiene el ángulo que describe su eje	Medición del ángulo que conforma el robot con respecto a la silla de ruedas
Mando a distancia	Emisora RF de 6 canales	Manejo del robot a distancia
Borneros	Conjunto de elementos de conexionado eléctrico	Alimentar y conectar componentes y tarjetas de adquisición

Fig. 1.3: Tabla Resumen de Componentes Electrónicos

La mayor parte de estos componentes pueden ser observados en la figura 1.2. Cabe comentar también que, la cintura de sensores, está compuesta por dos tipos de sensores según su principio de medida: 4 sensores infrarrojos y un ultrasonidos (a partir de ahora IR y US).

Sin embargo, lo más importante de cara a este proyecto es conocer cuáles son las características más relevantes de las tarjetas de adquisición de datos, del fabricante *National Instruments* (a partir de ahora NI), montadas en el ordenador. Es por esto que se presenta el siguiente cuadro:

TARJETA	6221	6602.1	6602.2
Modelo	PCI 6221	PCI 6602	PCI 6602
Entradas Analógicas	16	—	—
Salidas Analógicas	2	—	—
E/S Digitales	24	40	40
Timers	—	8	8
Conexión Serie/Par.	—	✓	✓
Puertos USB	—	✓	✓
Conexiones del robot	Motores	Mando a Dist.	Sensor US
	Encoders Ruedas	Sensor Angular	
	Sensores IR	Ventosas	
E/S Libres	8A / 24D	23D	39D

Fig. 1.4: Tabla Resumen de Características de las Tarjetas de Adquisición

Para ampliar cualquier información referente al desarrollo del robot HAMMI anterior a este proyecto, se pueden consultar las referencias bibliográficas [VEN07], [SIX07], [EVE08] y [GEN08]. En cuanto a los manuales de usuario de las tarjetas de adquisición de datos, pueden encontrarse en las referencias [MNI99] y [MNI08].

Terminando este breve repaso sobre los trabajos acometidos los años anteriores, es necesario decir que existe una serie de programas, estudios y modelos para el gobierno del robot. No presentados aquí, estos trabajos están referenciados en bibliografía y son utilizados para continuar el desarrollo en el presente proyecto.

1.2. Objetivos del proyecto.

Después de explicar los elementos y conceptos que deben ser manejados durante el desarrollo de este proyecto, es el momento de presentar más claramente de los objetivos que este proyecto pretende cubrir.

La primera tarea a realizar es obtener los datos de distancia del entorno. Para ello hay que tratar los datos de los sensores, mediante las tarjetas NI, para obtener una medida estable y en la magnitud adecuada. Seguidamente, hay que estimar estas mismas distancias a través de los datos de las trayectorias del robot y de su entorno.

Este es el punto en el que se pueden fusionar los distintos datos para poder implantar un filtro de Kalman. El objetivo principal que debe alcanzar este proyecto es facilitar los datos de medidas reales y estimadas para que la futura implantación de dicho filtro sea inmediata. Igualmente, se manipularán los programas existentes para detectar obstáculos y reaccionar ante ellos.

En los siguientes capítulos, se explicarán por lo tanto los estudios realizados sobre los sensores para obtener datos explotables. Después, se comentará el modelo diseñado para estimar las distancias. Finalmente, se comentará todo lo aprendido sobre el entorno de programación para pasar a describir las funciones implementadas en el robot.

Se debe comentar que, otros dos alumnos han realizado sendos proyectos sobre la ley de gobierno del robot HAMMI y sobre su brazo robótico, existiendo una coordinación entre las diferentes partes, entre otros, para respetar la nomenclatura utilizada.

2. ACONDICIONAMIENTO DE SENSORES.

Los sensores anteriormente mencionados, infrarrojos o ultrasonidos, no aportan una medida estable, homogénea ni libre de ruido. La medida primaria puede ser una tensión o una señal digital de una longitud de pulso variable; por lo tanto, dicha medida no podrá ser utilizada directamente.

De tal modo, se debe realizar un tratamiento de los datos de estos sensores mediante las tarjetas de adquisición NI. Gracias a la programación, se implantará un filtro digital y una conversión entre estas medidas primarias y la distancia deseada, en centímetros (a lo que se llamará calibración a partir de ahora). Sin embargo, habrá que diseñar y construir también un filtro analógico interpuesto entre sensores y tarjetas.

2.1. Programa de banco de ensayos para los sensores.

Aunque años atrás se trabajó con diferentes medidas de los encoders, nunca se ha explotado la cintura de sensores. Para obtener dichos datos y estudiar el tratamiento y soluciones adecuadas en cada momento, se ha desarrollado el programa que se explica en este epígrafe.

Este programa debe obtener los datos de los sensores y tratarlos en tiempo real, pudiéndolos transferir a un fichero. El lenguaje de programación de la aplicación es el *Visual C++*, el entorno de desarrollo el *Microsoft Visual Studio 2005* junto con los módulos suministrados por NI. La arquitectura seguida se explica en el capítulo 4 de esta memoria.

El nombre elegido para la aplicación es *Enregistrer* y su código más relevante puede ser consultado en el anexo A7. El programa ha sido desarrollado hasta la versión 2.4 que, siendo estable y funcional, es a la que pertenece el interfaz que se presenta a continuación:

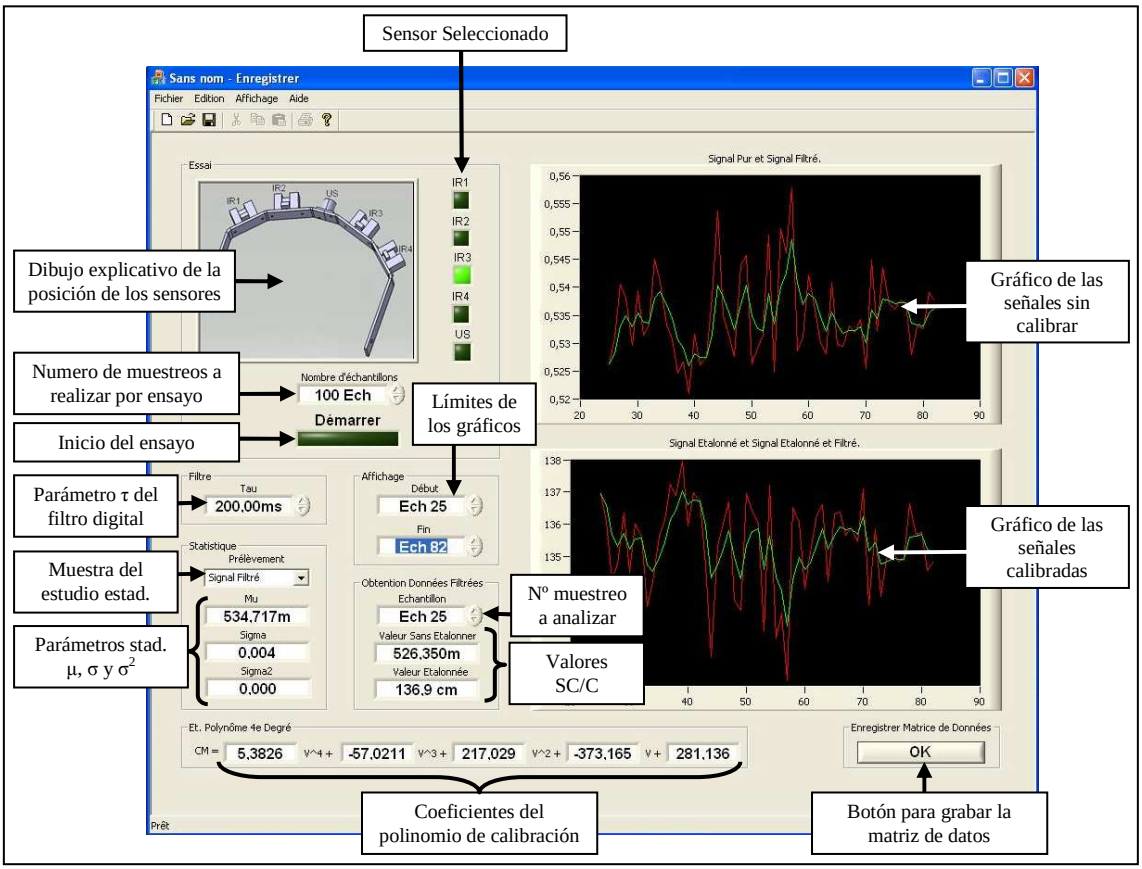

Fig. 2.1: Interfaz de usuario de Enregistrer v2.4

A continuación se describen brevemente los componentes de esta interfaz:

COMPONENTE	DESCRIPCIÓN
Número de muestreos	Permite seleccionar la duración del ensayo (1 muestreo cada 100ms)
Filtro digital de 1er orden	Permite cambiar dinámicamente la constante de tiempo **τ** del filtro
Gráficos	Se dibuja la evolución temporal de la señal del sensor seleccionado y el tratamiento de datos de la misma. Las líneas rojas son la señal sin filtrar y las verdes, filtrada. Se pueden fijar dinámicamente los límites de los gráficos para ampliar una zona de especial interés
Análisis estadístico	Permite obtener los valores de μ, σ et σ^2 de la muestra escogida. Esta muestra puede ser señal pura, filtrada, calibrada o filtrada-calibrada
Análisis de muestreo	Permite obtener el valor de un sensor para un muestreo exacto, siempre filtrado, sea calibrado o sin calibrar
Calibración	Permite calibrar la señal pura o filtrada de un sensor para convertir la medida a centímetros mediante un polinomio de hasta 4º grado
Grabado de datos	Permite actualizar la matriz de datos con los cambios efectuados en el interfaz. La matriz puede ser grabada en un fichero

Fig. 2.2: Tabla Resumen de los Componentes de la Interfaz de usuario de Enregistrer v2.4

Para terminar, cabe comentar que, aunque este programa encuentra su aplicación directa en este proyecto, puede servir de modelo para un banco de ensayos de cualquier otro sistema de adquisición de datos.

2.2. Tratamiento de datos de los sensores.

Habiendo explicado la necesidad del tratamiento de datos de las medidas tomadas por los sensores de distancia, es necesario ahora presentar los elementos concretos de dicho tratamiento y los estudios realizados para implantarlos.

Cada problema percibido ha encontrado una solución después de la realización de estudios diseñados para ese fin concreto. Estos estudios cuentan con un análisis de la problemática, una proposición de soluciones y la puesta en funcionamiento de la solución escogida con el resultado final. Se distingue, en cada momento, si la solución elegida es de carácter eléctrico o a nivel programado.

Parte eléctrica.

El primero de los inconvenientes del sistema de adquisición existente es una señal con un nivel de ruido muy elevado en los sensores infrarrojos. Dicho ruido no permitirá disponer de una medida con suficiente precisión y por tanto, explotables.

Para paliar este problema y según lo dicho, se ha llevado a cabo un estudio para conocer a fondo el fenómeno: magnitud del ruido y origen del mismo. Variando el número de sensores alimentados al mismo tiempo se pretende conocer si el ruido proviene de un acople óptico o eléctrico.

Tomando siempre los datos del sensor denominado IR4 (ver figura 2.3) se proponen tres escenarios: sólo el sensor IR4 conectado, dos sensores completamente opuestos conectados (IR1 e IR4) y todos los sensores conectados. Obsérvese la disposición de los sensores en la cintura:

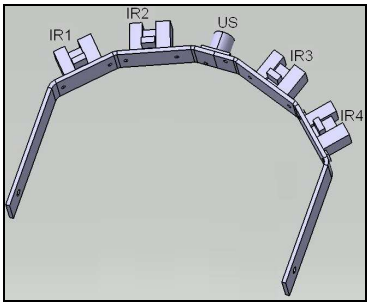

Fig. 2.3: Disposición de los sensores

Por cada escenario se realizan 25 ensayos con un total de 100 muestreos cada uno. Posteriormente se realiza un análisis estadístico con la ayuda de la aplicación en versión de evaluación *STATGRAPHICS Centurion XV* (ver bibliografía [STG09]).

Si el ruido se produce al conectar más de un sensor en el sistema, las varianzas de los tres escenarios serán distintas. Si el ruido se produce por un acople óptico, las varianzas no debería variar al pasar de un sensor conectado a dos sensores opuestos conectados (sin interferencia óptica posible).

Tras realizar un contraste de hipótesis al 95% se llegó a la conclusión de que el ruido aumentaba a medida que se conectaban más sensores y que éste era de carácter eléctrico.

De tal modo, se decidió que la solución más viable era la de implantar un filtro analógico en combinación con otro digital. El filtro analógico es de tipo pasivo de primer orden y sintonizado en 1,45 Hz, tras analizar la dinámica del robot. Dicho filtro

se dispone entre los sensores y las tarjetas de adquisición de datos y sigue el siguiente esquema:

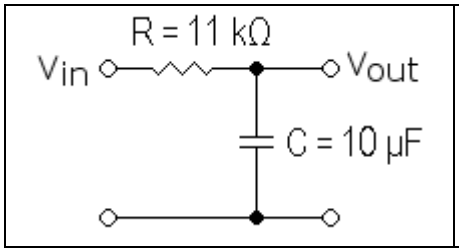

Fig. 2.4: Configuración del Filtro

$$f_C = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 11k\Omega \cdot 10\mu F} = 1{,}45\,Hz$$

Fig. 2.5: Frecuencia de Corte del Filtro

Siguiendo las recomendaciones del fabricante, también se ha situado un condensador de 10µF junto a la alimentación de todos los sensores. Para ello se han hecho unos nuevos cables que unen los sensores a una nueva tarjeta diseñada para albergar los 4 filtros analógicos según la foto:

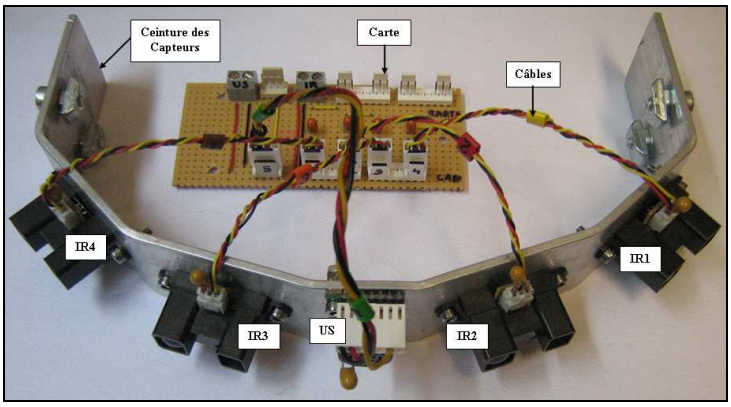

Fig. 2.6: Conjunto del Sistema de Sensores

Esta tarjeta se atornilla a la estructura del robot junto a la cintura de sensores, sustituyendo a una tarjeta más antigua que solo servía de bornero. Para mayor detalle sobre el sistema de sensores anterior, consultar las referencias bibliográficas [VEN07] y [SIX07].

Tras poner en servicio el nuevo sistema y realizar una nueva prueba, el nivel de ruido se redujo considerablemente. Un análisis para cuantificar la precisión del sensor IR4 dio como resultado un error de 1,0674 cm, lo cual se considera más que aceptable para la aplicación.

Parte programada.

Después que la señal está libre de ruido, se puede proceder a un tratamiento informático de los datos para finalizar teniendo medidas explotables. Al igual que el banco de ensayos de los sensores, estos sistemas son programados en *Visual C++* y ejecutados en el ordenador embarcado.

Primeramente, se ha implementado un filtro digital de primer orden en combinación con el filtro analógico. Dicho filtro es de tipo RII o Respuesta Impulsional Infinita (en bibliografía [BEL93]), y siendo *y(n)* la salida actual del filtro, *y(n-1)* la salida anterior y *x(n)* la entrada; el filtro sigue la siguiente ecuación recursiva:

$$y(n) = (1-\alpha)\cdot x(n) + \alpha \cdot y(n-1); \quad \alpha \in (0,1), \alpha = e^{-0{,}1/TAU}$$

Fig. 2.7: Ecuación del Filtro Digital de 1er Orden

Respetando la dinámica sistema de adquisición de datos y del robot y después de haber realizado diversos ensayos, se fijó la constante de tiempo τ (TAU en la ecuación y el interfaz de la figura 2.1) a 0,1 segundos. La implementación de este filtro se puede consultar en los anexos A7 y A8.

Posteriormente se realiza un nuevo estudio para calibrar los sensores. El fin de dicho estudio es obtener los polinomios que conviertan lo mejor posible las magnitudes primarias de los sensores (voltios y segundos) en centímetros.

Para ello se tomarán las medias del conjunto de 5 ensayos de 50 muestreos cada uno a unas distancias conocidas y fijas. Dichas distancias son de 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 110 et 120 centímetros para los infrarrojos y de 20, 40, 60, 80, 100, 120, 140 et 160 centímetros para el sensor de ultrasonidos. Así, se conseguirá una tabla que relacione voltios o segundos y centímetros. Véase la tabla del sensor IR4:

Conjunto	Media [V]	Distancia [cm]
1	0.6336134	120
2	0.7007718	110
3	0.767979	100
4	0.7867718	95
5	0.8064412	90
6	0.8454564	85
7	0.933906	80
8	0.9531074	75
9	1.0094	70
10	1.0478	65
11	1.166	60
12	1.2566	55
13	1.4082	50
14	1.538	45
15	1.7042	40
16	1.908	35
17	2.1828	30
18	2.4658	25
19	2.7212	20

Fig. 2.8: Datos para la Calibración del Sensor IR4

Una vez se tenga esta tabla, los datos serán pasados a dos vectores en *Matlab* (uno para las tensiones y otro para las distancias) y gracias a una función llamada *interpoler.m* especialmente diseñado para esta calibración, se obtendrán polinomios de diferentes grados y el error que comete cada uno. La implementación de dicha función se puede encontrar en el anexo A4.

De esta manera, se seleccionan polinomios de 4º grado, en función del voltaje de salida (con diferentes coeficientes para cada sensor), para los sensores de infrarrojos y una función lineal, en función de los milisegundos de retorno de eco, para el sensor de ultrasonidos. Véase a continuación los polinomios seleccionados para el sensor IR4 y para el sensor de ultrasonidos:

$$cm = 17{,}0095 \cdot volt^4 - 137{,}5219 \cdot volt^3 + 413{,}2059 \cdot volt^2 - 568{,}9687 \cdot volt + 347{,}416$$

Fig. 2.9: Polinomio de Calibración del Sensor IR4

$$cm = 17{,}5884 \cdot ms + 2{,}094$$

Fig. 2.10: Polinomio de Calibración del Sensor de Ultrasonidos

Es posible consultar la implementación de estas funciones en el código de la aplicación de gobierno del robot, dentro del anexo A8.

3. MODELIZACIÓN DE LOS SENSORES.

Para continuar con los requisitos para implantar un filtro de Kalman, es necesario diseñar un modelo cinemático que estime las distancias que miden los sensores.

Se sabe que los puntos que el centro de gravedad del robot describe en su trayectoria son datos disponibles en tiempo real, dentro de la aplicación de gobierno que hay implantada actualmente.

Sean ω1 y ω2 las velocidades angulares de las ruedas motrices del robot, existe ya un modelo cinemático del robot y varios bancos de ensayos desarrollado en *Matlab*; cada uno describiendo una trayectoria y cuyo fichero de ejecución *Matlab* siempre se llama « programme_principal.m ». Por lo tanto y según la figura 3.1, en este punto se diseñará un modelo cinemático para los sensores en Matlab, integrado en el banco de pruebas existente:

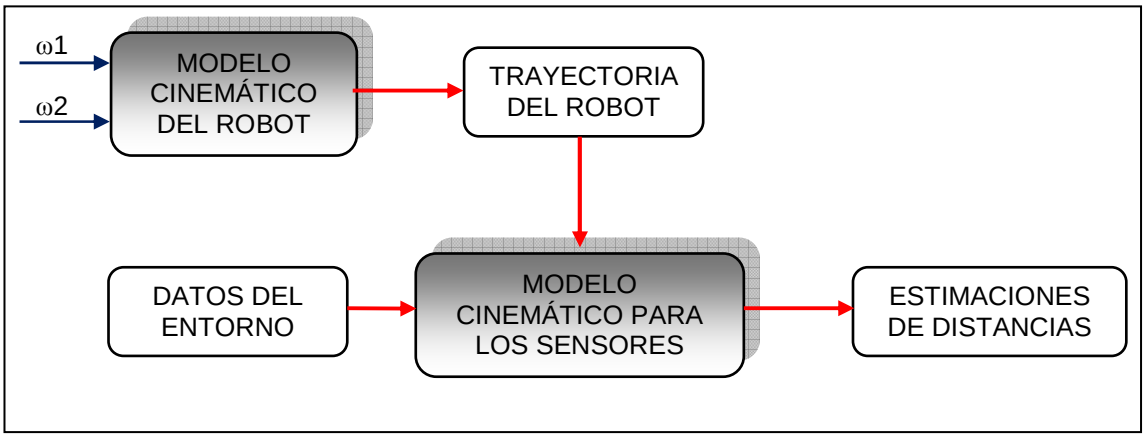

Fig. 3.1 : Schéma pour la Conception du Modèle pour les Capteurs

De tal modo, el nuevo código desarrollado en el fichero « modele_capteurs.m » se incluirá al final del banco de ensayos que define la trayectoria. De este último recibe los datos de los tiempos de discretización ***t*** y una matriz ***x*** de la cual se obtendrá la posición (***Xr***, ***Yr***) y orientación (***θr***) del centro de gravedad del robot. Para saber más sobre las trayectorias desarrolladas en años anteriores, consultar la referencia bibliográfica [RUB07].

A parte de estos datos, es necesario conocer la configuración de la cintura de sensores con respecto al centro de gravedad. Para ello, se presenta la siguiente figura:

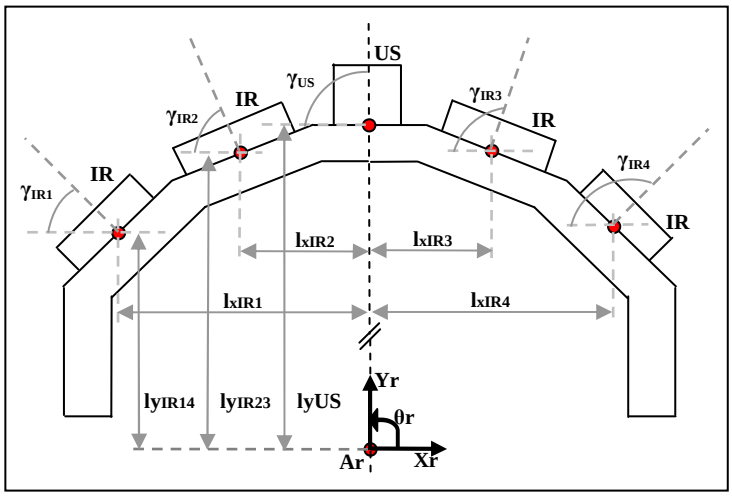

Fig. 3.2: Parámetros Geométricos de los Sensores

El valor numérico de todos estos parámetros ha sido obtenido y se puede consultar en la asignación realizada al comienzo del código del fichero « modele_capteurs.m », en el anexo A5. Para más precisión sobre la construcción de la cintura de sensores, diríjase a las referencias [VEN07] y [GEN08].

Con respecto al entorno del robot, cabe especificar que el robot se inscribirá dentro de un rectángulo delimitado por 4 paredes que son: ***ymur2***, ***ymur1***, ***xmur2*** et ***xmur1***; según su orientación con el eje *x* o el *y*. Las coordenadas de estas paredes se expresan siempre con respecto al origen del sistema (Xo, Yo), común al de la trayectoria.

Cabe especificar que, este año, sólo se ha tomado en cuenta el robot sin unirse a la silla de ruedas. En el futuro, para tener este punto en cuenta, habrá que manejar una ley de gobierno que genera la trayectoria para todo el conjunto pero de cara al modelo de los sensores no habrá diferencias significativas, si bien algún sensor puede quedar anulado.

3.1. Método de cálculo.

Tras analizar las restricciones y datos disponibles para la realización de este modelo, se ha llegado a la conclusión de que lo más conveniente es realizar un modelo de tipo geométrico; basándose en el análisis del entorno y la posición del robot.

La base de este método se observa para el sensor IR4 en el siguiente esquema:

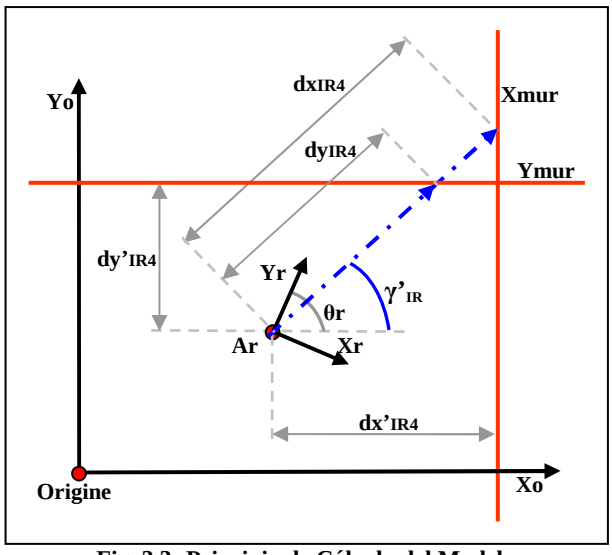

Fig. 3.3: Principio de Cálculo del Modelo

Como se aprecia en la figura, el haz de percepción de los sensores es susceptible de incidir en dos de los muros definidos, siempre dependiendo de la orientación del robot. El principio de este modelo geométrico consiste en detectar en qué muros impactará el haz y calcular las distancias denominadas en la figura como *dxIR4* y *dyIR4*. Después se escogerá la menor distancia entre las dos posibles, que será la distancia correcta.

La ecuación para obtener la distancia denominada cómo *dxIR4* es:

$$dxIR4 = \frac{dx'\,IR4}{\cos(\gamma'_{IR4})} = \frac{Xr - Xmur - lyIR14 \cdot \sin(pi/2 - \theta r) + lxIR4 \cdot \cos(pi/2 - \theta r)}{\cos((pi/2 + \theta r) - \gamma_{IR4})}$$

Fig. 3.4: Ecuación de Cálculo

Para conocer los muros donde impactará el haz del sensor y por tanto, las ecuaciones que deben utilizarse, se analizarán los intervalos del ángulo θr una vez reducido al rango [0, 360]. Así, el siguiente gráfico muestra los intervalos obtenidos para el sensor IR4:

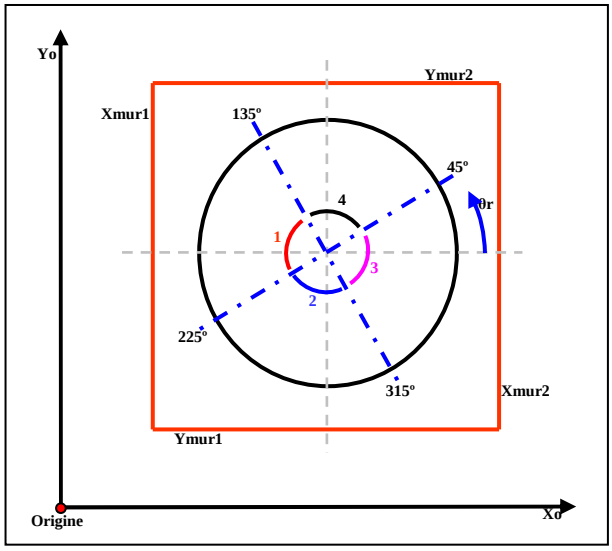

Fig. 3.5: Intervalos para las Ecuaciones del sensor IR4

Para consultar el resto de ecuaciones para todos los sensores así como sus intervalos de aplicación, diríjase al anexo A5 donde se encuentra el código de este fichero.

3.2. Validación del modelo.

Para poder verificar que la construcción del modelo es correcta, se le ha sometido a diversas trayectorias y pruebas. Sin embargo, aquí se presentarán los resultados del escenario más exigente que se muestra a continuación:

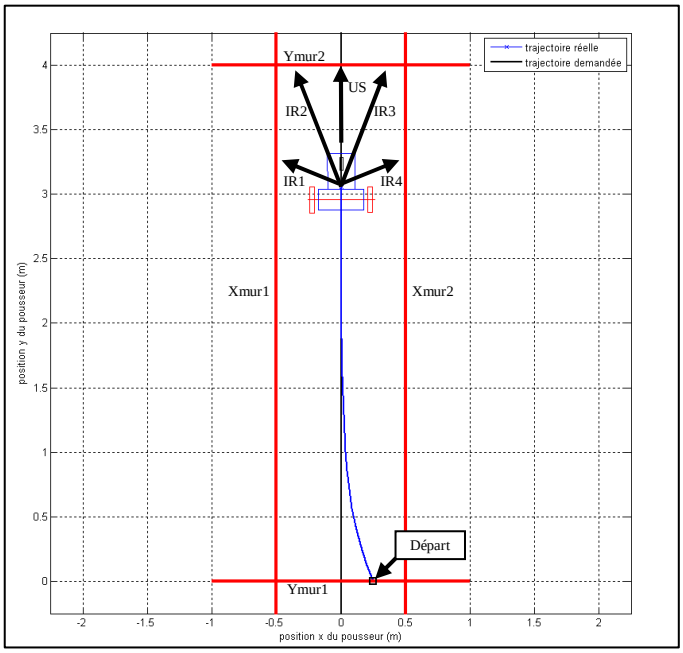

Fig. 3.6: Trayectoria « Suivi de Ligne Droite »

Tras aplicar el modelo, se obtuvo la siguiente gráfica donde se puede observar la evolución de las distancias en el tiempo y por tanto, verificar la validez de los resultados:

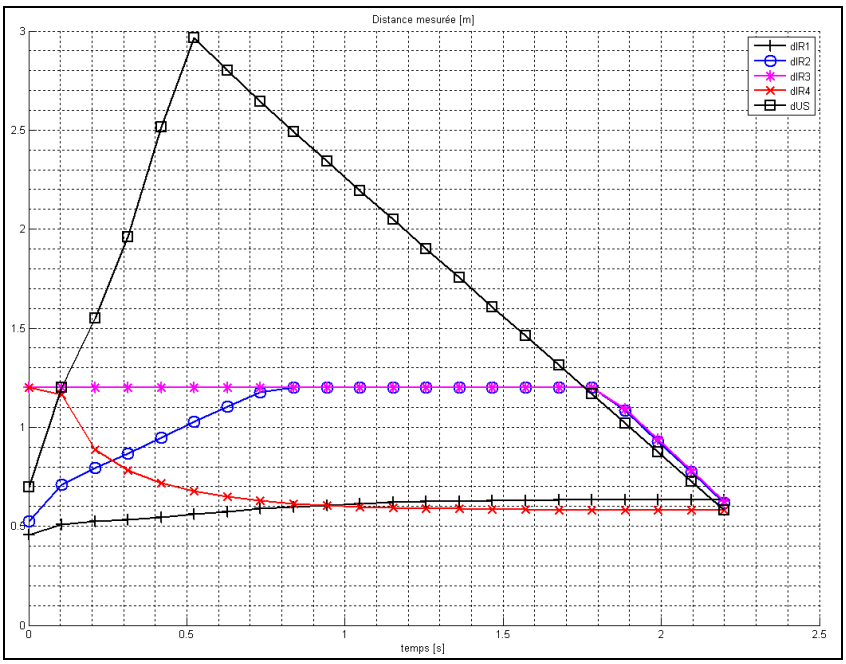

Fig. 3.7: Representación Gráfica de Distancias

Cabe comentar que se han modelado los límites de percepción de los sensores que son de 20 a 120 centímetros para los infrarrojos y de 20 a 600 centímetros para el ultrasonidos. Debido a estos límites y como se observará en la práctica, los sensores IR2 e IR3 no son muy funcionales en este caso aunque podrán usarse para detectar obstáculos diferentes a las paredes.

4. PROGRAMATION DEL ROBOT.

Anteriormente, se ha comentado que el gobierno del robot se llevará a cabo a través de aplicaciones desarrolladas en lenguaje *Visual C++* y con el entorno de desarrollo el *Microsoft Visual Studio 2005* y bibliotecas de NI.

La aplicación para el ensayo de los sensores, realizada bajo estas condiciones, ha sido ya expuesta. Se presentará ahora la arquitectura de todas estas aplicaciones y los nuevos programas desarrollados para incluir el tratamiento de datos y el modelo de sensores en un programa de gobierno.

4.1. Arquitectura Document-View, Visual Studio 2005 y NI.

Entre las diversas arquitecturas que ofrece el lenguaje y entorno de programación utilizados, se ha elegido la Document-View. Las justificaciones de esta elección son: la herencia de los anteriores proyectos, las recomendaciones de NI para el desarrollo de aplicaciones y la mejor estructuración del programa según dos clases principales.

Estas clases son el documento y la vista, de ahí el nombre de esta arquitectura. En general, en el documento se gestionarán todos los datos del programa mientras que

la vista se encargará de todo lo referente al interfaz de usuario. Ambas clases se comunican e incluyen diversos ficheros según lo muestra la siguiente figura:

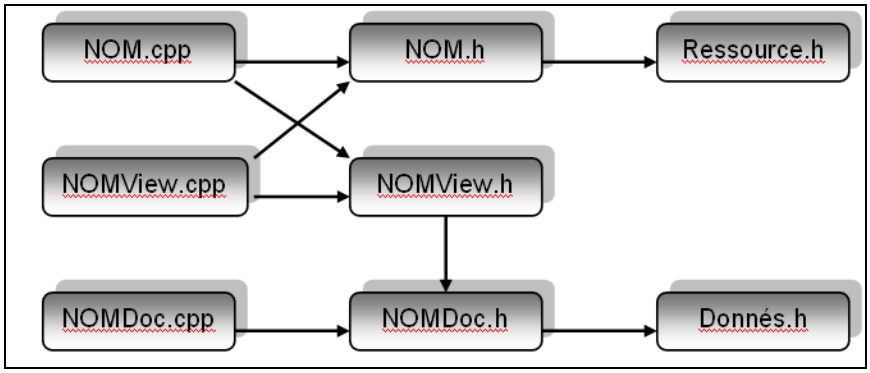

Fig. 4.1: Arquitectura Document-View en Visual Studio 2005

Esta estructura de archivos es creada automáticamente al generar un proyecto adecuadamente en el entorno de desarrollo *Visual Studio 2005*. Se observan así los ficheros de código *.cpp* que contienen todas las funciones y manejadores y los ficheros *.h* de declaración de funciones y variables. El distintivo *NOM* hace referencia a un nombre genérico elegido por el programador al que se le añadirá el final indicado.

Una vez generado un proyecto Document-View, aparece el entorno de desarrollo de la aplicación como se muestra en la siguiente figura:

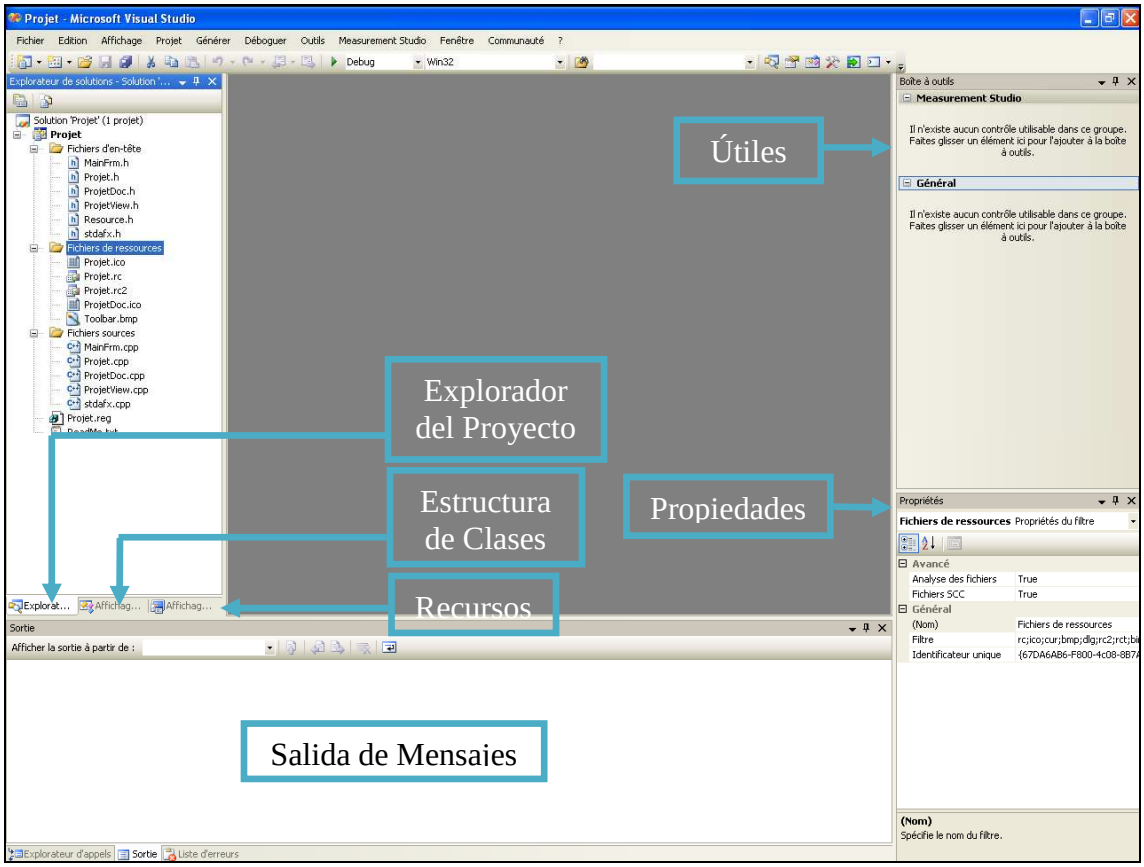

Fig. 4.2: Entorno de Desarrollo de Visual Studio 2005

Añadido al entorno de desarrollo, NI incluye una serie de bibliotecas que permiten utilizar las tarjetas de adquisición de datos y una serie de recursos exclusivos, tales como clases, elementos gráficos, etc.

En la siguiente imagen se presenta el interfaz de usuario de una aplicación de ejemplo, explicado en el anexo A6, desarrollado con elementos de NI:

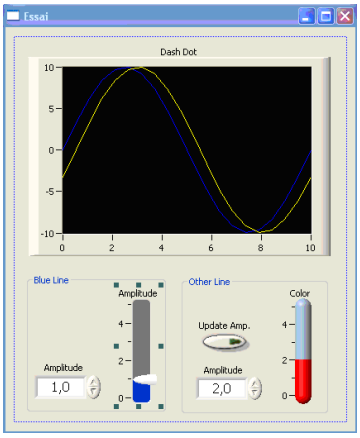

Fig. 4.3: Aplicación de Ejemplo

Para cualquier duda sobre la arquitectura Document-View y el entorno de desarrollo *Visual Studio 2005*, consúltese la referencia bibliográfica [MIC00]. Si se pretende profundizar en los recursos de NI y su herramienta principal llamada *Measurement Studio* diríjase a la referencia [TNI09].

4.2. Aplicación para el robot.

Después de desarrollar la aplicación para el ensayo de los sensores, ahora se pretende modificar el código de una aplicación para el gobierno del robot para incluir el tratamiento de datos, el modelo de los sensores y un sistema de paro automático. Así, como evolución de la versión 1.0, se ha llegado al programa *Robot_HAMMI v2.2*, Su código se adjunta en el anexo A8 y su interfaz, totalmente renovado, se muestra a continuación:

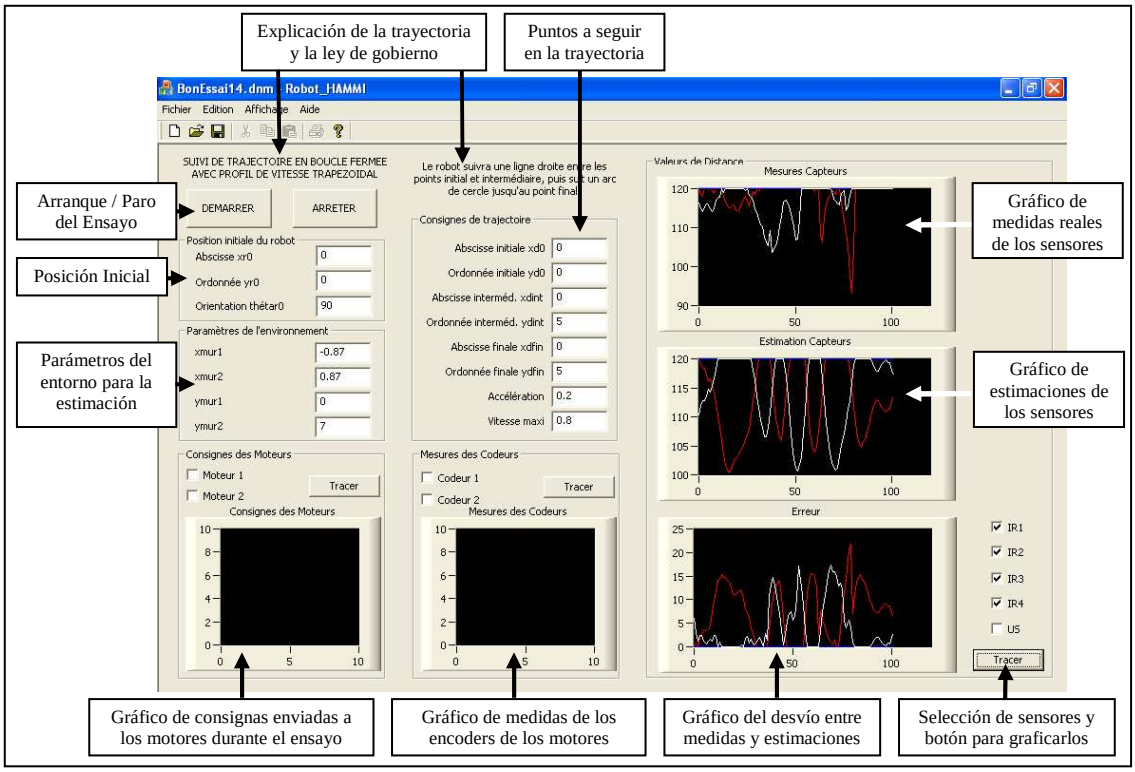

Fig. 4.4: Interfaz de Usuario de Robot_HAMMI v2.2

A través de ciertos pasos, la información de una matriz de datos, que contiene toda la información acerca del ensayo del robot en una trayectoria recta en un pasillo, ha sido pasada a *Matlab* y analizada con un fichero llamado *comparer.m*; cuyo desarrollo se explica en el anexo A9. De tal manera y entre otras, se representa la trayectoria realizada por el robot y su entorno en este ensayo:

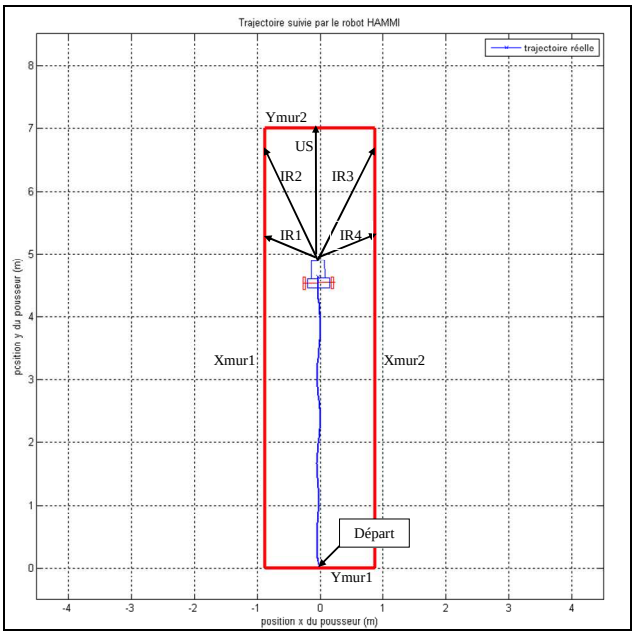

Fig. 4.5: Figura 4 del Fichero comparer.m (Trayectoria)

Con el mismo fichero, se ha comprobado si la traducción del modelo de estimación de las distancias de los sensores ha sido correctamente realizada al lenguaje *Visual C++*. El resultado ha sido totalmente satisfactorio. Este fichero también permite el análisis y comparación de las distancias reales y comparadas en un entorno *Matlab* con todas sus ventajas. Así se han apreciado las gráficas del interfaz de la figura 4.4 y se han analizado los resultados.

Para los sensores infrarrojos, aunque el resultado es satisfactorio, existen diferencias de hasta 15 centímetros entre medida y estimación debidas al ruido estructural, desfase del sistema de adquisición de datos y datos de la trayectoria no reales. Esto se corregirá con el futuro filtro de Kalman. El sensor de ultrasonidos ha mostrado problemas de reflexión y de amplitud de haz en el pasillo.

Por último, se comprobó el funcionamiento del sistema de paro automático (SPA) con unos resultados satisfactorios, detectando el obstáculo a 50 centímetros y parando suavemente a 20 centímetros del mismo. Véase el cuadro de diálogo mostrado en este caso:

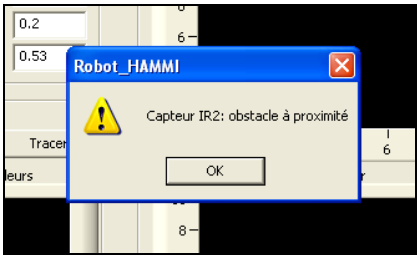

Fig. 4.6 Mensaje de Aviso del SPA

5. CONCLUSIÓN Y TRABAJOS PREVISTOS.

El resultado global de este proyecto es el de desarrollar e implantar en el robot HAMMI los sistemas necesarios para que la futura puesta en marcha de un filtro de Kalman, que corrija las trayectorias del robot, sea inmediata.

Para ello, se ha trabajado en el diseño y construcción de un sistema de adquisición de datos, en la realización de un modelo de estimación de las medidas de los sensores de distancia e implantación de ambas en una aplicación de gobierno.

Mediante este trabajo, se ha llegado a disponer de medidas de distancia adecuadas y estimaciones de estas medidas. Esto se ha llevado a una aplicación que permitió obtener la diferencia entre estos dos datos, es decir, el error estimado; que es la base de un filtro de Kalman. Se puede afirmar, por tanto, que el proyecto ha sido un éxito y se han cumplido todos sus objetivos.

En cuanto a los trabajos futuros, la implantación de un filtro de Kalman es inminente. Sin embargo, se puede trabajar también en otro tipo de modelo de estimación, probar nuevas leyes de gobierno e incluso, en un nuevo emplazamiento de sensores. También se debe contar con las restricciones que supondrá incluir la silla de ruedas en todos los trabajos realizados puesto que un primer ensayo demuestra que la silla de ruedas inutiliza la cintura de sensores actual.

Personalmente, este proyecto ha sido satisfactorio por sus resultados y por los conocimientos adquiridos en cuanto a automática, sistemas de adquisición de datos y programación de robots. En resumen, muy interesante a nivel académico, este proyecto me ha permitido también integrarme en el laboratorio LMSP y trabajar en colaboración con otros alumnos; compartiendo los conocimientos adquiridos.

RAPPORT EN FRANÇAIS

SOMMAIRE

SOMMAIRE DES FIGURES

1. INTRODUCTION.

Depuis longtemps la technologie a été développée au service de l'humanité et en satisfaisant une partie de ses besoins. Donc, ce n'est pas par hasard que lorsque cette technologie a évolué, elle ait été mise à la disposition des personnes handicapées. C'est dans esprit que ce projet a été défini.

En 2007, la population française avec une déficience motrice quelconque est de plus de 10 %, en atteignant à environ 21% si l'on parle d'autres incapacités. Dans le cas de la communauté européenne on parle aussi de 10% de personnes handicapées sur la totalité de sa population. C'est ici que se situe ce projet. Pour plus d'information sur ces données voir la référence [HDP07].

Ainsi, le LMSP (Laboratoire de Mécanique des Systèmes et des Procédées), appartenant à l'ENSAM et soucieux de faire une contribution remarquable en ce champ, a proposé le projet appelé : Robotique Mobile Pour Personnes Handicapées. Tout cela avec la collaboration de la société HandiTecAM. Le but de ce projet est de concevoir une plateforme de développement qui permettra d'aider les personnes ayant ces difficultés motrices ou sensorielles.

Inscrit dans ce projet et depuis trois ans, on développe le robot HAMMI, qui signifie HandiTec Arts et Métiers Motorisation Intelligente. Il s'agit d'un prototype de robot pousseur autonome qui, placé derrière un fauteuil pour handicapés, sera capable de suivre les instructions d'un pilote comme, par exemple, le déplacer automatiquement. De même et grâce à un bras robotique, il pourra l'aider à manipuler des objets de la vie quotidienne.

1.1. État d'avancement du robot HAMMI au début du projet.

Dans ce paragraphe on va commenter l'état du robot HAMMI au début de ce projet-ci. Pour mieux structurer cette présentation, on a groupé le travail développé selon deux parties : d'abord les composants mécaniques et électriques et puis, les composants électroniques.

La première partie est nécessaire pour avoir une conception globale. La deuxième partie, étant la plus importante de ce projet, va être détaillée afin d'améliorer au maximum la compréhension des travaux réalisés.

1.1.1. Côté mécanique et électrique.

Depuis les projets faits pendant les dernières années par les diverses équipes de travail sur le robot pousseur HAMMI, on dispose d'une structure autonome mobile qui porte les éléments nécessaires pour déplacer le fauteuil.

Les composants les plus remarquables sont exposés ci-dessous :

- **Structure Support :** Il s'agit d'un ensemble de segments du profil structural dûment reliés qui hébergent le reste des éléments tels que le calculateur, les roues et moteurs, etc. Elle assure aussi le déplacement et stabilité de l'ensemble du pousseur à travers de deux roues folles. Voir la figure 1.1 :

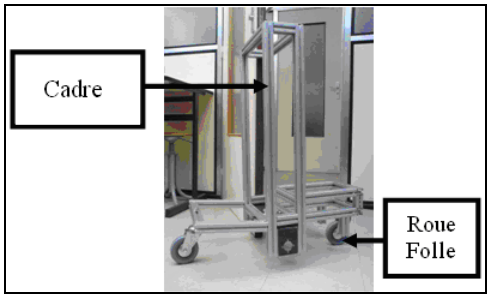

Fig. 1.1 : Structure Support

- **Roues Motrices :** Ce sont les composants qui permettent au robot de se déplacer. Le robot comprend deux roues latérales unies d'axes avec les moteurs correspondants, qui sont reliés à la structure. On peut noter que les moteurs sont accouplés à des encodeurs pour piloter et mesurer le mouvement. La figure 1.2 présente cet ensemble :

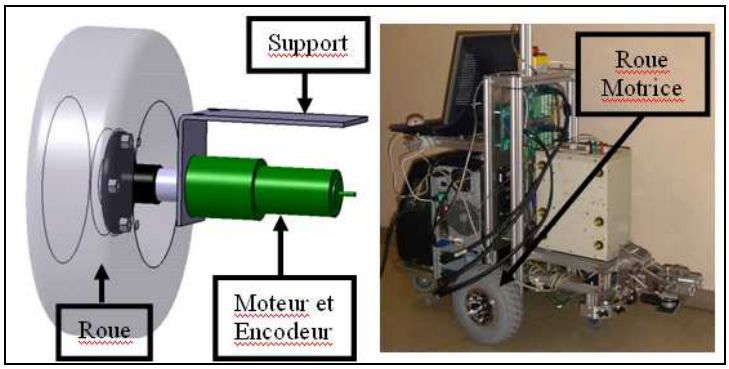

Fig. 1.2 : Roue Motrice

- **Ventouses Electromagnétiques :** Elles réalisent la jonction physique entre le fauteuil et le robot pousseur. Elles sont composées de deux aimants commandés par un signal électrique. Elles sont placées devant le robot et un ensemble d'articulations permettent supporter les efforts. Son orientation et mesurée à travers d'un capteur angulaire absolu. Voir la figure 1.3 :

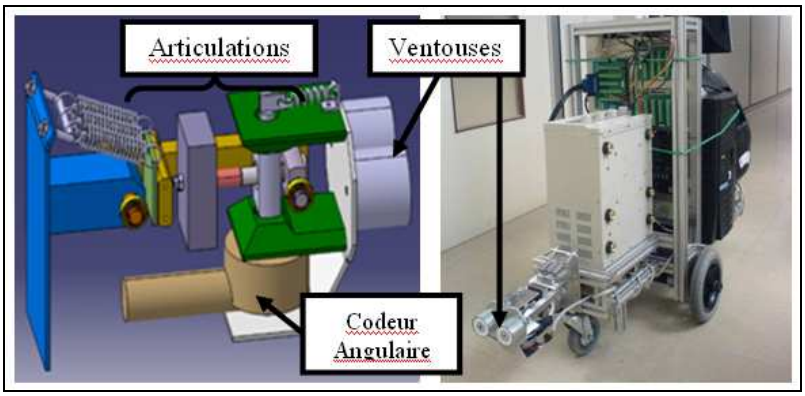

Fig. 1.3 : Ventouses Electromagnétiques

- **Source d'Alimentation :** Embarquée devant du robot on peut trouver la source d'alimentation de celui-ci. Elle alimente toute la partie électronique sauf le calculateur, qui a sa propre source. Cette source est montrée à la figure 1.4 :

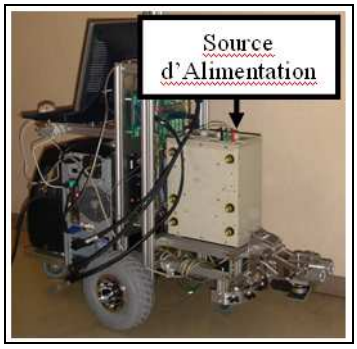

Fig. 1.4 : Source d'Alimentation

- **D'autres Supports :** En plus des composants déjà présentés, on termine en exposant les supports des capteurs. On parle du support de la caméra ainsi que du support de la ceinture de capteurs IR et Ultrasons. Ils sont montrés à la figure 1.5 :

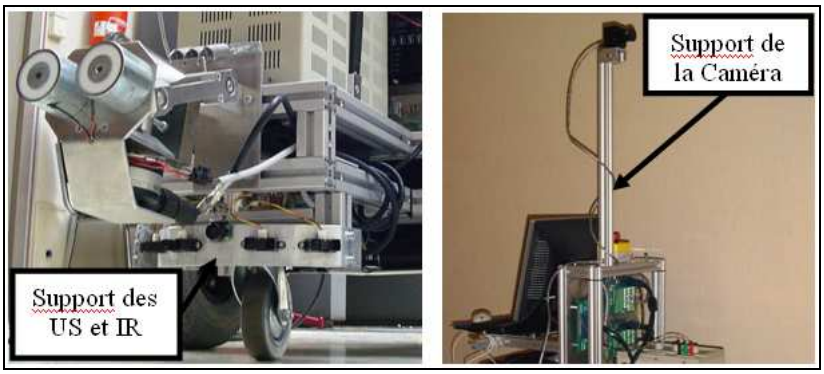

Fig. 1.5 : D'autres Supports

Tous ces renseignements ont été obtenus en utilisant les projets des dernières années. Pour plus de précision, on peut consulter ces rapports référencés en bibliographie [VEN07], [SIX07] et [GEN08].

1.1.2. Côté calculateur, électronique et capteurs.

Auparavant, on a réalisé un développement qui va servir comme base pour apporter des nouvelles fonctionnalités au robot. En raison des objectifs qu'il faut atteindre, cette partie est très importante pour ce projet car elle fournit toute l'intelligence au robot.

Ainsi, on présente tous ces composants :

- **Calculateur :** Il s'agit d'un ordinateur PC de type général qui facilitera le travail de programmation, l'acquisition des données, la commande des actionneurs, etc. Les logiciels dont on a besoin pour contrôler toutes les fonctionnalités du robot sont, tout simplement, le système d'exploitation *Windows XP* et l'outil *Visual Studio 2005*, enrichi avec les bibliothèques des cartes électroniques installées. On parlera plus tard de ces cartes.

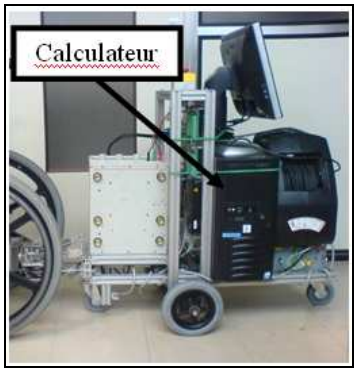

Fig. 1.6 : Calculateur

- **Cartes National Instruments :** Le calculateur possède trois cartes électroniques du fabriquant *National Instruments* (désormais NI) qui servent à communiquer avec les capteurs et actionneurs du robot. Ces trois cartes ont des fonctionnalités différentes selon le tableau suivant :

CARTES	6221	6602.1	6602.2
Modèle	PCI 6221	PCI 6602	PCI 6602
Entrées Analogiques	16	—	—
Sorties Analogiques	2	—	—
E/S Numériques	24	40	40
Timers	—	8	8
Liaison Série/Par.	—	✓	✓
Ports USB	—	✓	✓
Liaisons	Moteurs	Télécommande	Capteur US
	Codeurs Roues	Encodeur	
	Capteurs IR	Ventouses	
E/S Libres	8A / 24N	23N	39N

Fig. 1.7 : Tableau des Cartes NI

- **Variateurs :** Les cartes précédentes n'ont pas la puissance suffisante pour agir directement sur les moteurs des roues motrices. C'est pour cela qu'on a utilisé deux variateurs de chez *Maxon*, un pour chaque moteur. Il s'agit donc d'une interface entre les sorties des cartes NI et les moteurs mais il transporte aussi le signal des encodeurs des roues.

- **Borniers :** Pour connecter les cartes NI aux autres actionneurs et capteurs on dispose des borniers fournis par NI. De cette manière, on assure la connexion entre les dispositifs de façon fiable et claire. Tous les deux derniers éléments sont fixés à une plaque métallique, qui est également embarquée sur le robot, comme on peut voir à la figure 1.8:

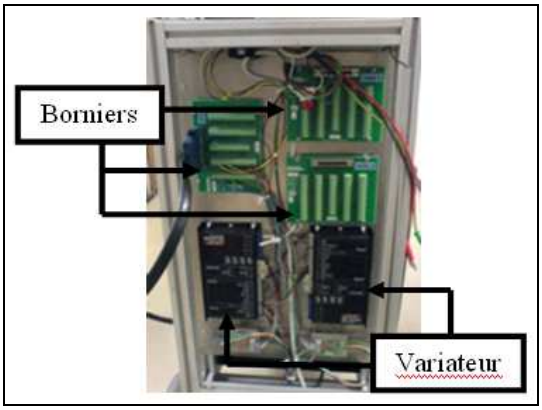

Fig. 1.8 : Borniers et Variateurs

- **Caméra :** Ce robot possède aussi une caméra couleur pour la reconnaissance de l'image et pour avoir la possibilité d'interagir avec l'environnement. Bien entendu, cette caméra est bien sûr un capteur. Néanmoins, elle n'est pas connectée aux cartes NI mais au port série du calculateur. On peut voir cette caméra et son support à la figure 1.5.
- **Capteurs :** Bien que l'objectif soit d'augmenter le nombre des capteurs dans le périmètre du robot, pour l'instant il n'y a que quatre capteurs infrarouges et un capteur ultrasons installés sur une ceinture à l'avant du robot. Ces capteurs permettent de mesurer les distances aux obstacles que le robot peut trouver. Le travail présent et futur de ce projet est ciblé sur ces capteurs et leur pilotage à travers des cartes NI. On peut voir ces capteurs et leur support à la figure 1.5.
- **Codeur Angulaire :** Comme cela a été déjà dit, lié aux ventouses électromagnétiques, il y a un codeur angulaire qui mesure la rotation du fauteuil par rapport au robot. Cette information sera très importante pour calculer les trajectoires du pousseur. On peut voir ce capteur à la figure 1.3.
- **Télécommande :** Une télécommande a été prévue et installée pour agir sur le pousseur à distance et de façon immédiate. De telle façon, les personnes handicapées pourront elles mêmes faire bouger le fauteuil. Cette télécommande fonctionne à travers d'une application déjà développée sous Visual C++ et des cartes NI.

Tous ces renseignements ont été obtenus à partir des rapports des projets des dernières années. Pour plus de précision, on peut consulter ces rapports référencés en bibliographie [VEN07], [SIX07], [EVE08] et [GEN08]. En outre, dans l'annexe A1 on présente des tables qui résument les connexions entre les capteurs, les actionneurs, les borniers et les cartes NI.

1.2. Organisation et objectifs du projet.

Après avoir expliqué les éléments à piloter et ceux dont on dispose, on peut parler plus clairement de l'organisation et des objectifs que ce projet doit couvrir. Dans ce paragraphe on présentera aussi des autres projets concernant en ce moment le robot HAMMI.

Tout d'abord, le travail que l'on doit faire est d'obtenir des données stables et correctement traitées issues des capteurs de distance, grâce aux cartes NI. Ensuite, il faut estimer les distances que les capteurs mesurent à travers des trajectoires suivies par le robot dans un environnement défini. C'est ainsi qu'on peut fusionner les différents données pour réaliser les calculs nécessaires et implanter un filtre de Kalman. L'objectif essentiel à la fin de cette année est de laisser disponibles tous ces données mesurées et estimées pour que l'implantation de ce filtre devienne immédiate. De même, on doit manipuler les codes existants pour détecter les obstacles éventuels et agir sur le mouvement du robot.

Dans les prochains paragraphes, on commencera donc à parler des études réalisées sur les capteurs pour obtenir des données exploitables. Ensuite, on expliquera le modèle conçu pour estimer les distances mesurées par les différents capteurs. Juste après, on parlera de tout ce qui est nécessaire sur le langage de programmation *Visual C++*, le logiciel de développement *Microsoft Visual Studio 2005* et les possibilités des

cartes NI. Ainsi, on assimilera toutes les informations nécessaires pour expliquer les fonctions implémentées sur le robot.

Quant aux autres projets étant réalisés sur ce même robot cette année, on peut en commenter deux : le calcul des trajectoires selon nouvelles lois de commande et avec les nouvelles données, et la modélisation du bras robotique qui permettra aux usagers du robot manipuler des objets.

Le premier de ces projets a pour titre « Robot mobile pour l'assistance aux personnes handicapées : partie commande » et il est réalisé par Thomas Stenkiste ; tandis que le deuxième d'entre eux est intitulé « Modélisation d'un bras robotique » et il fut accompli par Madeleine Hellqvist.

2. CONDITIONNEMENT DES CAPTEURS.

Les différents capteurs, soit infrarouges, soit ultrasons, n'apportent pas une mesure ni stable ni homogène ni libre de bruit. Dans certains cas la mesure est un voltage et dans d'autres un code numérique, mais elle n'est jamais une donnée qu'on puisse utiliser directement.

Ainsi, il faut toujours réaliser un traitement des données des capteurs. Grâce aux cartes d'acquisition NI, on peut implanter des algorithmes programmés sur langage *Visual C++* tels qu'un filtre numérique de premier ordre ou une fonction de conversion entre les données primaires (volts, secondes, etc.) et les données secondaires (mètres) ; ce qu'on appelle désormais étalonnage. Mais, il faudra aussi mettre en place un filtre analogique entre les capteurs et les cartes d'acquisition.

La prise des données et l'analyse de la solution nécessaire et des résultats après l'implantation des mécanismes de traitement des données sont réalisées à l'aide d'un logiciel, développé sur *C++*, exprès pour essayer les capteurs.

Le logiciel d'essai des capteurs et les traitements des données qui s'avèrent nécessaires, sont largement expliqués plus bas dans ce rapport.

2.1. Logiciel d'essai des capteurs.

Dans les années précédentes on était arrivé à prendre des mesures des encodeurs des moteurs et d'autres périphériques et à travailler avec elles. Pourtant, personne n'avait pas pris les données des capteurs de la ceinture, déjà construite et embarquée sur le robot d'ailleurs.

D'abord, il faut donc un logiciel qui obtienne les données des capteurs et qui soit capable de les traiter en temps réel de différentes façons et de les enregistrer et les charger. Pour tout cela, on a conçu une application sur le langage de programmation *Visual C++*, à l'aide de *Microsoft Visual Studio 2005* et des modules de cartes d'acquisition NI.

Cette application s'appelle *Enregistrer* et elle a été développée jusqu'à la version 2.4, qui reste très fonctionnelle et stable. Le code de ce logiciel peut être consulté dans l'annexe A7.

On présente ici son interface d'usager :

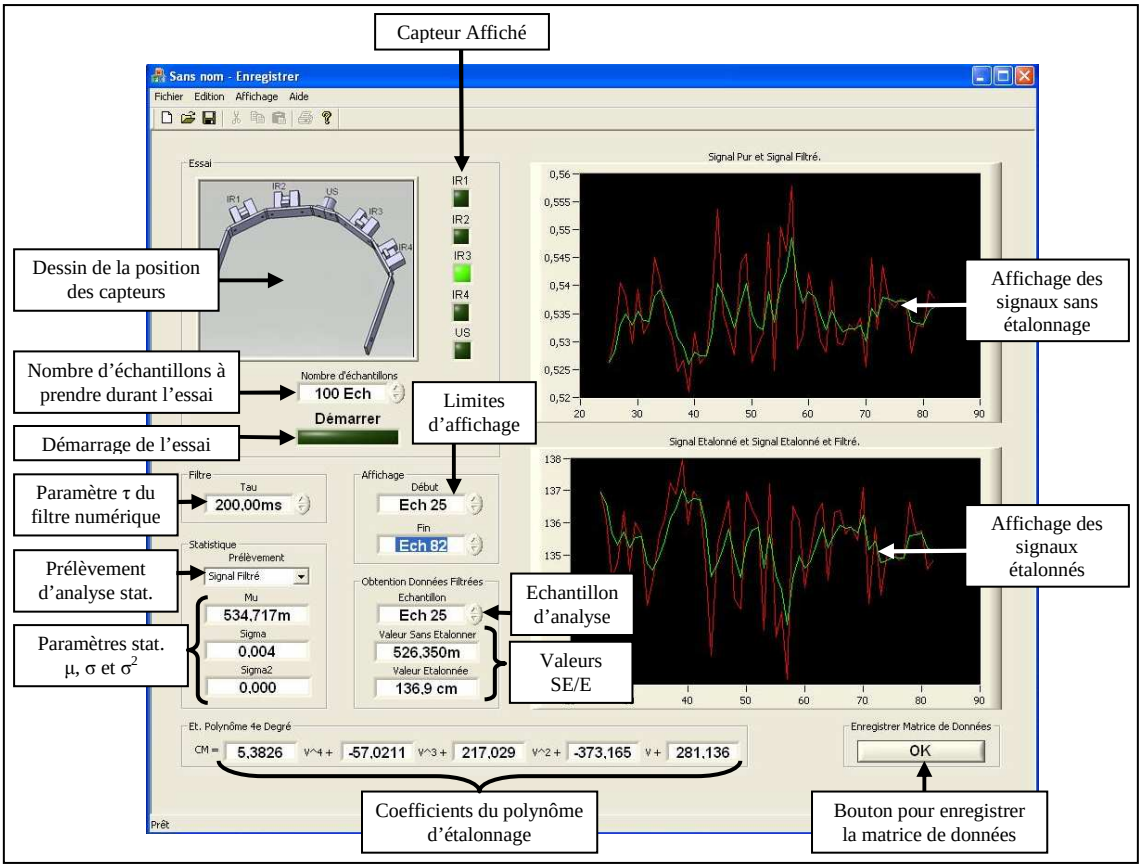

Fig. 2.1 : Interface d'usager d'Enregistrer v2.4

On décrit ci-dessous ses fonctions plus remarquables du logiciel et leur utilité :

- **<u>Nombre d'échantillons</u> :** Il faut le sélectionner avant d'appuyer sur le bouton de démarrage et permet de choisir la longueur de l'essai, en sachant que la fréquence d'échantillonnage est d'un échantillon chaque 100 millisecondes. Tous les essais ne demandent pas le même temps d'observation et de cette façon on peut le réduire aux besoins.

- **<u>Paramètre τ</u> :** Il permet de sélectionner de façon dynamique la valeur de la constante de temps d'un filtre numérique de premier ordre. Les effets sont montrés instantanément et ainsi l'on peut choisir la meilleure valeur.

- **<u>Affichage des valeurs</u> :** Il y a beaucoup d'éléments rapportés à cette fonction principalement graphique. On peut voir deux graphiques où on montre l'évolution des valeurs des échantillons : celui en haut les valeurs sans étalonner et celui en bas les valeurs étalonnées. Les lignes rouges dessinent toujours les valeurs sans filtrer et celles en vert les valeurs filtrées. Plusieurs boutons à côté du dessin de la ceinture, dont seulement un peut être actif, permettent de choisir le capteur à afficher bien que tous sont enregistrés. Enfin, on peut sélectionner les limites d'affichage pour mieux apercevoir une certaine partie du graphique. Tout cela permet d'analyser visuellement la réponse temporelle du signal, le bruit, etc.

- **<u>Statistique</u> :** Cette boîte apporte les valeurs statistiques de μ, σ et σ^2 du prélèvement choisi. De même, pour sélectionner le prélèvement il existe une liste déroulante où on peut choisir parmi le signal pur, le signal filtré, le signal

étalonné et le signal filtré et étalonné (en cet ordre). Cette analyse permettra de pondérer le niveau de bruit d'une mesure, en testant l'effet des filtres et de l'étalonnage.

- **Obtention données filtrées :** Ici, on peut sélectionner un échantillon exact et obtenir la valeur numérique correspondante étalonnée et sans étalonner. Cela sert à connaître rapidement la valeur des mesures pour réaliser une analyse numérique ; par exemple des pics et même de l'étalonnage.
- **Étalonnage :** En fixant ces coefficients on va constituer un polynôme de 4^{e} degré au maximum pour transformer les données primaires des capteurs en distances en centimètres (utiles pour l'utilisation ultérieure). Les changements sont actualisés en temps réel sur le graphique en bas.
- **Enregistrement :** Pour enregistrer tous les changements réalisés sur les données initiales de l'essai il faudra appuyer ce bouton avant d'enregistrer définitivement. Ainsi, la matrice de données, où résident dynamiquement les mesures, est actualisée avec les modifications.

Voilà la description de l'application développée. Pour finir, on doit dire que ce logiciel marche toujours lié aux cartes NI et que, soit tel qu'il l'est, soit pris comme modèle, cette application servira pour essayer des futurs capteurs et même pour un autre système.

2.2. Traitement des mesures des capteurs.

On a déjà expliqué la nécessité d'un traitement correct des mesures puisque il est impossible de les utiliser directement à cause du bruit, de la nature de la mesure, etc.

Il a fallu donc réaliser une analyse pour chaque problème aperçu. Cette analyse comprendra une estimation du problème pour mieux le connaître, une proposition des solutions possibles, la mise en place de la solution choisie et la présentation du résultat obtenu.

On fera une distinction selon la solution soit implanté avec des circuits électriques ou à travers de la programmation et du logiciel dans le calculateur embarqué. Voici les différents traitements accomplis pour le bon fonctionnement des capteurs du robot HAMMI.

2.2.1. Partie électrique.

Le premier des inconvénients rencontrés dans le système d'acquisition des données à travers des capteurs infrarouges du robot est la présence d'un bruit trop élevé. Ce bruit ne permettra pas d'avoir précision dans les données obtenues et elles ne seront donc pas exploitables.

D'après ce qui a été dit auparavant, on a réalisé une étude complète pour connaître l'ampleur et l'origine du bruit à travers d'un essai et puis pour expliquer la solution prise et montrer les conclusions finales. Voici cette étude.

Conception de l'essai.

Cette étude a pour but de connaître l'origine des interférences qui provoquent le bruit dans les mesures des capteurs du robot HAMMI. Pour cela on va considérer trois situations différentes :

- 1er Cas : seul le capteur 4 du robot connecté (Voir la figure 2.2).
- 2e Cas : deux capteurs complètement opposés connectés (Le capteur 4 prend les mesures).
- 3e Cas : tous les capteurs sont connectés (Le capteur 4 prend les mesures).

De cette manière, on peut connaître si le bruit est la résultante des interférences à travers l'alimentation ou bien des couplages optiques.

La disposition et nomenclature des capteurs dans la ceinture du robot est désormais établie telle qu'elle est montrée à la figure 1.1 :

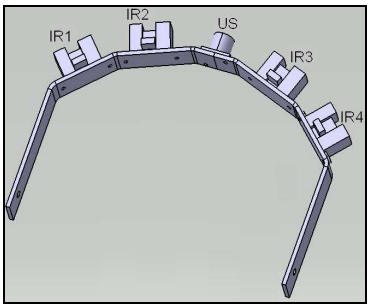

Fig. 2.2 : Disposition des Capteurs

Pour chacune de ces situations, on va faire 25 essais sans modifier ni la distance à mesurer ni l'objet à détecter (même couleur, même orientation, etc.) par le capteur 4 du robot. Dans chaque essai on prend 100 échantillons, un échantillon chaque 100 millisecondes et on calcule la moyenne et la variance. Cela met un total de 10 secondes par essai.

Finalement, on aura trois groupes de 25 essais et avec deux données chacun, la moyenne des 100 échantillons et leur variance. Grâce à l'analyse statistique, on compare les trois situations avec les moyennes et les variances.

Tout ce procès a été réalisé à l'aide de la première version du logiciel auparavant présenté (*Enregistrer v1.0*). Comme on a déjà dit, il a été développé exprès pour tester les capteurs en affichant les résultats des essais dans des graphiques ou bien en les enregistrant. Cette première version permettra aussi de fixer n'importe quel nombre d'échantillons pour chaque essai, la constante τ d'un filtre numérique de premier ordre et les limites d'affichage (pour zoomer sur quelques échantillons). De même, elle calculera la moyenne des échantillons affichés, l'écart type et la variance. Enfin, on pourra obtenir les valeurs numériques exactes qui correspondent à un échantillon souhaité.

Voici l'interface d'usager de cette application :

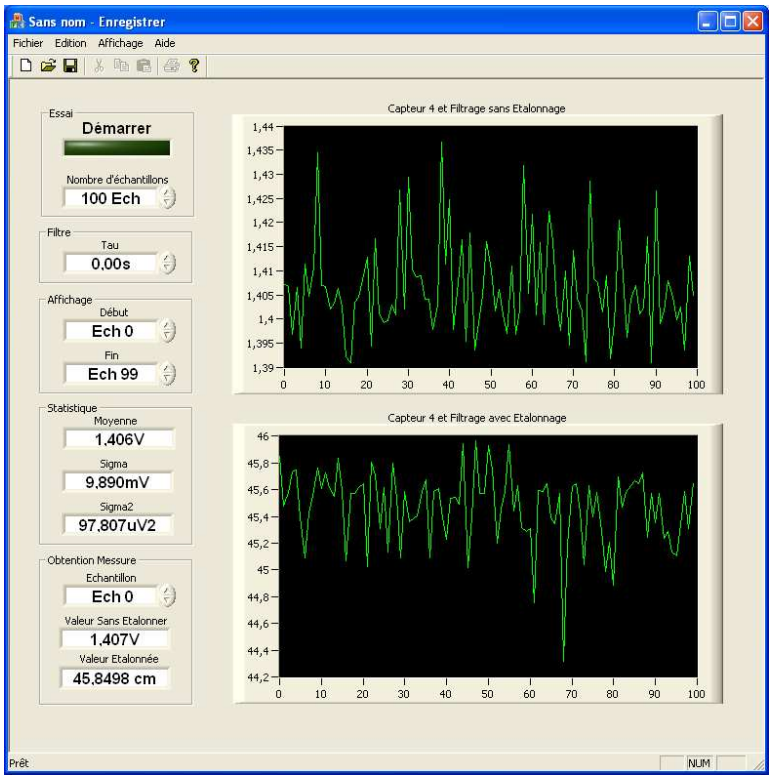

Fig. 2.3 : Interface d'usager d'Enregistrer v1.0

Résultats de l'essai.

On a rempli trois tableaux dont chacun correspond à un prélèvement ou cas auparavant décrit. Chaque prélèvement ou tableau comprend donc 25 échantillons où on a pris la moyenne et la variance des 100 données qui composent chaque échantillon de l'étude. Ces tableaux et leurs données, qui sont la source de l'analyse réalisée plus bas, sont exposés dans l'annexe A2.

Analyse de l'essai.

Cette analyse-ci sera réalisée grâce à un logiciel dédié à cet effet qui s'appelle *STATGRAPHICS Centurion XV*, (voir bibliographie [STG09]) obtenu en version d'évaluation.

Ainsi, on a obtenu le diagramme de boîte appliqué sur la variance du 1er cas décrit, « Seul le capteur 4 connecté ». Il est montré à la figure 2.4 :

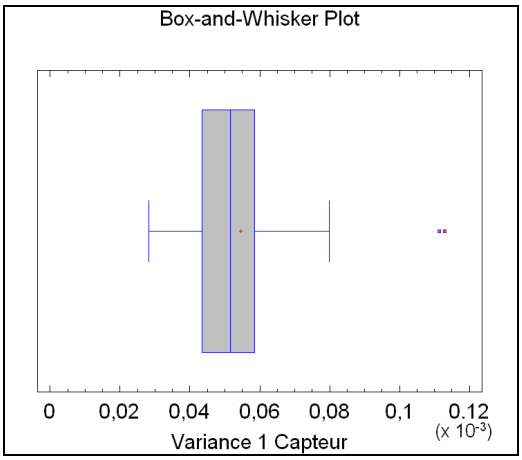

Fig. 2.4 : Diagramme de Boîte du 1er Cas

Remarquons que les centiles 25% et 75% sont les latéraux de la boîte tandis que la moyenne est représentée avec la ligne verticale dans la boîte et elle se situe vers $0{,}0545377 \cdot 10^{-3}$ V^2. Notons aussi qu'il y a deux données atypiques.

De même, on montre ensuite les diagrammes correspondants aux cas « Deux capteurs opposés connectés » et « Tous les capteurs connectés » :

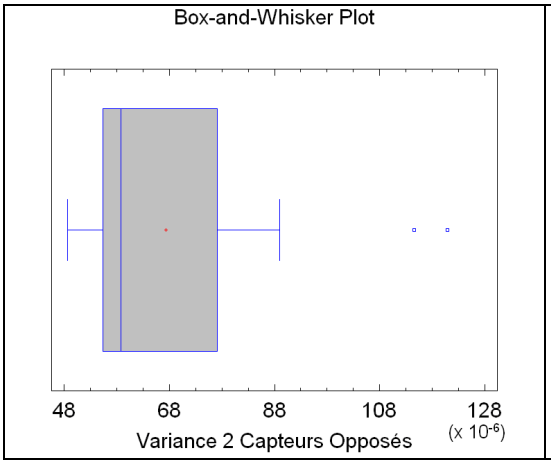

Fig. 2.5 : Diagramme de Boîte du 2e Cas

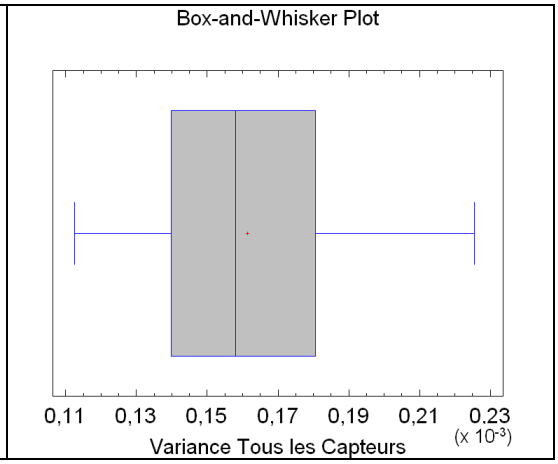

Fig. 2.6 : Diagramme de Boîte du 3e Cas

Notons que la moyenne du premier diagramme est de $0{,}067321 \cdot 10^{-3}$ V^2 et que celle du deuxième diagramme est de $0{,}16148 \cdot 10^{-3}$ V^2. C'est-à-dire, qu'on peut observer déjà des différences significatives entre les trois cas.

Pour confirmer ce fait, on fera des comparaisons entre les différents cas et on obtiendra un résultat à travers des contrastes d'hypothèses. L'hypothèse nulle signifiera que les deux moyennes des variances sont égales et alors qu'il n'y a pas de différences entre les variations (ou bruit) des cas en étude. Par contre, l'hypothèse alternative comportera que le bruit des cas est d'une amplitude différente dans chaque cas, soit que les différentes configurations ont des niveaux de bruit différents.

Voici les résultats :

- « Seul le capteur 4 connecté » vs. « Tous les capteurs connectés » : On rejette l'hypothèse nulle avec un 95% de confiance.
- « Seul le capteur 4 connecté » vs. « Deux capteurs opposés connectés » : On rejette l'hypothèse nulle avec un 95% de confiance.
- « Deux capteurs opposés connectés » vs. « Tous les capteurs connectés » : On rejette l'hypothèse nulle avec un 95% de confiance.

En conclusion, tous les niveaux de bruits sont différents puisque les trois cas montrent des différences significatives entre les variances obtenues à partir des échantillons pris. Dans le 1er cas le niveau du bruit est le plus petit, un peu plus grand pour le 2e cas et le plus grand pour le 3e cas.

D'autres renseignements.

Pour aider à trouver une solution au problème et pour donner une idée des différents types de bruit qui ont été mesurés, on apporte à la suite quelques graphiques des essais réalisés :

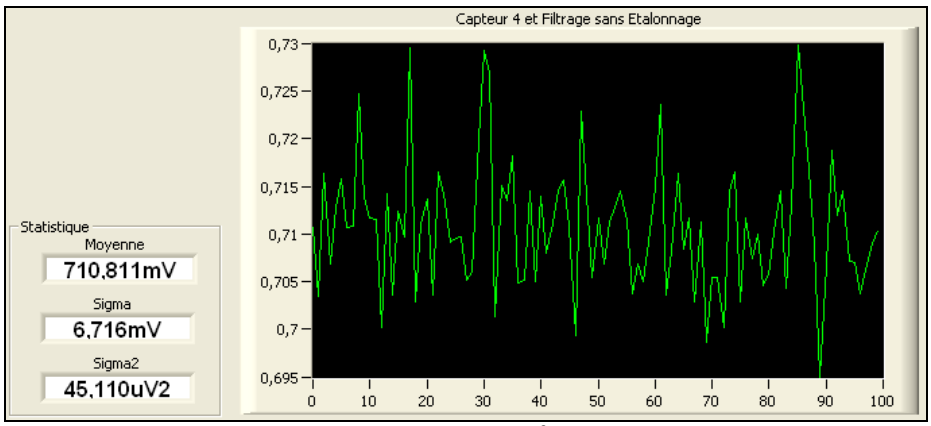

Fig. 2.7 : Essai du 1e Cas

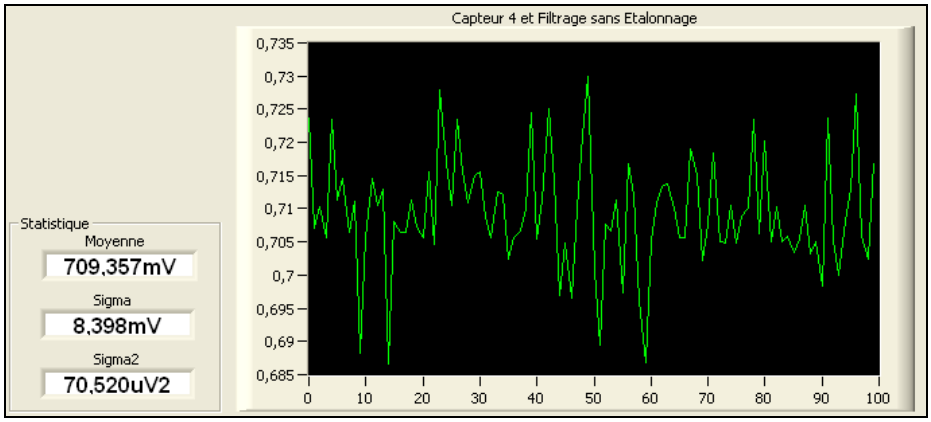

Fig. 2.8 : Essai 2 2e Cas

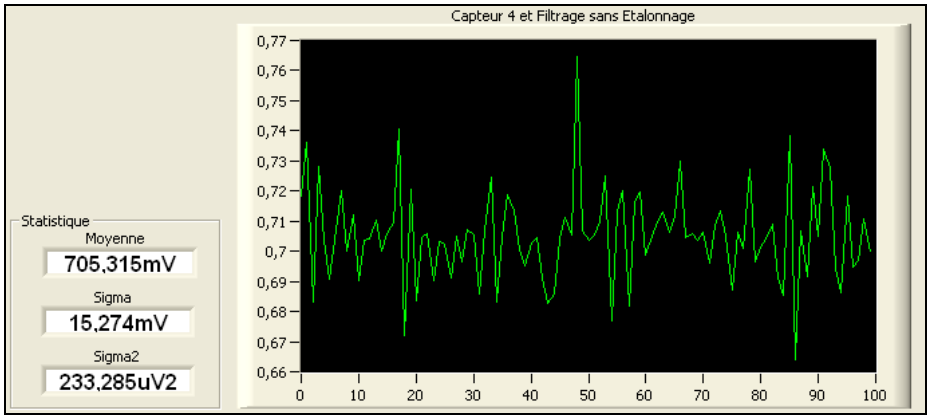

Fig. 2.9 : Essai 1 3e Cas

Conclusions de l'essai.

Répondons à la question qu'on posait au début de cette étude où l'on cherchait à déterminer si le bruit des capteurs provient d'une interférence électrique ou d'un couplage optique.

Si l'on fait une analyse logique des résultats, on peut dire que le bruit est d'origine électrique et qu'il s'amplifie lors de l'ajout de plusieurs capteurs.

Si le bruit était optique, il y aurait une différence entre les moyennes du cas « Tous les capteurs connectés » et du cas « Deux capteurs opposés connectés », mais il n'y aurait pas de différences entre le bruit du cas « Deux capteurs opposés connectés » et celui du cas « Seul le capteur 4 connecté ». La différence entre la première comparaison se justifie grâce à que le bruit par couplage électrique augmente si l'on connecte tous les capteurs.

Il faut donc corriger le bruit électrique induit entre les capteurs en essayant d'obtenir le bruit du cas « Seul le capteur 4 connecté ». De plus, ce bruit sera éliminé grâce au filtre numérique de premier ordre (pareil à celui qui est implanté dans l'application *Enregistrer*).

Correction du problème.

On a déjà vu le type de bruit et sa source et maintenant, on va essayer de l'enlever. Tout d'abord on fera une petite analyse des solutions possibles et ensuite on détaillera celle qui a été prise.

On a prouvé qu'il existe un rapport entre le nombre des capteurs connectés et le bruit qu'ils génèrent. Donc, une première approche serait de mettre en place une alimentation séparée pour chaque capteur. Pourtant, cela est lourd et coûteux et l'année dernière on a déjà séparé l'alimentation des capteurs infrarouges du reste des alimentations.

D'autre côté, il y a déjà un filtre numérique développé qui peut être augmenté pour essayer d'éliminer ce bruit (voir plus bas). Mais aussi, il faut penser à la réponse dynamique du système et ne pas trop la ralentir. C'est pour cela que si on met une constante du temps trop élevée, le retard introduit n'est pas négligeable. Une bonne dynamique serait assurée avec un filtre de constante du temps d'environ 100 millisecondes mais, bien qu'il soit utile, il n'enlève pas suffisamment le bruit.

L'option d'essayer des capteurs d'un autre type est toujours possible mais on veut utiliser ceux qui sont déjà adaptés et mis en place sur la ceinture du robot. En plus, les nouveaux capteurs devraient avoir toujours sortie analogique pour ne pas épuiser et respecter les entrées des cartes numériques, donc le problème risque de continuer avec d'autres capteurs.

Il semble que l'option la plus sensée soit d'ajouter un filtre analogique en combinaison avec le filtre numérique, mais en tenant toujours compte de la dynamique du système. Le filtre analogique sera assez efficace mais pourtant, il va ralentir aussi la réponse du système et donc, il faudra sélectionner une bonne constante du temps et fonction de transfert.

Ensuite, il reste beaucoup de paramètres à définir, soit le type, soit l'ordre du filtre. Quant au type, il faut dire qu'il serait ennuyeux de mettre en place un filtre actif, en introduisant des amplificateurs opérationnels qui même pourraient comporter plus de bruit. Si on choisi donc un filtre passif, il sera beaucoup plus simple et il est possible qu'il suffise d'un 1er ordre.

Etant donné le filtre passif de 1er ordre, il faut encore fixer la configuration et la constante τ du filtre en fonction des valeurs des composants. Quant à elles, on va suivre la figure 2.10 dont les valeurs sont spécifiées, résultant cette fréquence de coupure:

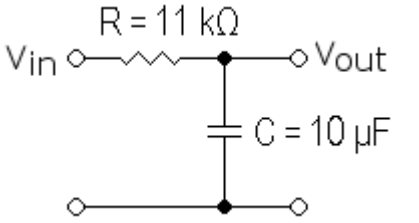

Fig. 2.10 : Configuration du Filtre

$$f_C = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot 11k\Omega \cdot 10\mu F} = 1{,}45\,Hz$$

Fig. 2.11 : Fréquence de Coupure du Filtre

On assure donc qu'on enlève la plupart du bruit en respectant la dynamique avec une constante $\tau = R \cdot C = 11k\Omega \cdot 10\mu F = 110ms$. En comptant aussi sur la constante du filtre numérique et en sachant que la vitesse linaire du robot n'atteint qu'au maximum à 1 m/s, ce qui veut dire que le robot avance environ 21 cm dès qu'un obstacle est interposé jusqu'à ce qu'il soit détecté. Cet écart n'est pas trop grave en sachant que la plus part des obstacles ne sont pas instantanément interposés dans les rayons des capteurs.

Pour mettre en place tous ces filtres on a refait une nouvelle carte de liaison des capteurs avec les borniers NI. Sa localisation ainsi que sa photographie est exposé ci-dessous :

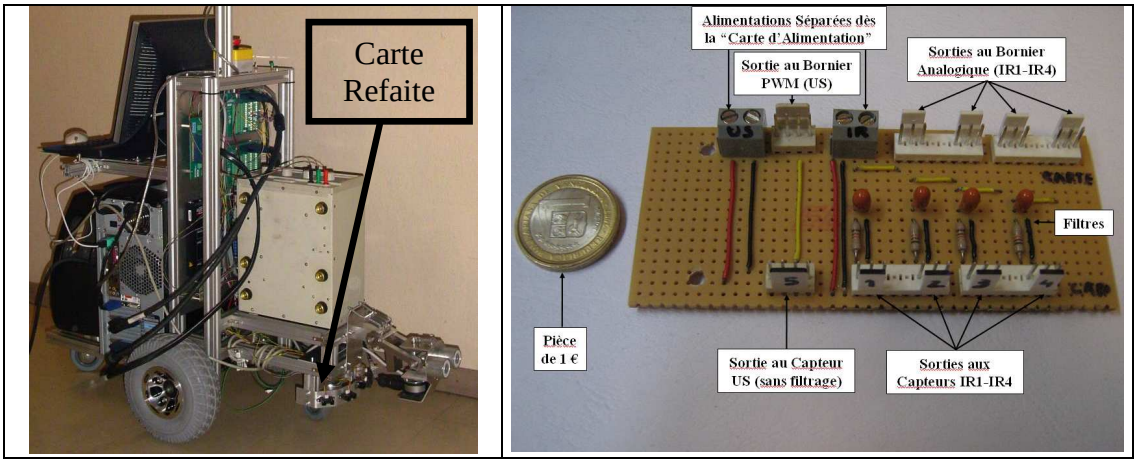

Fig. 2.12 : Localisation et Photographie de la Carte Mise en Place

En plus de ce filtre et en suivant les indications du fabricant, une capacité de 10 µF a été mise le plus proche possible à l'alimentation des capteurs. Pour cela, il a fallu refaire tous les câbles des capteurs comme est montré à la figure suivante :

Fig. 2.13 : Câbles vers les Capteurs

Finalement, on expose la photographie de tout l'ensemble du nouveau système des capteurs avant d'être remonté sur le robot :

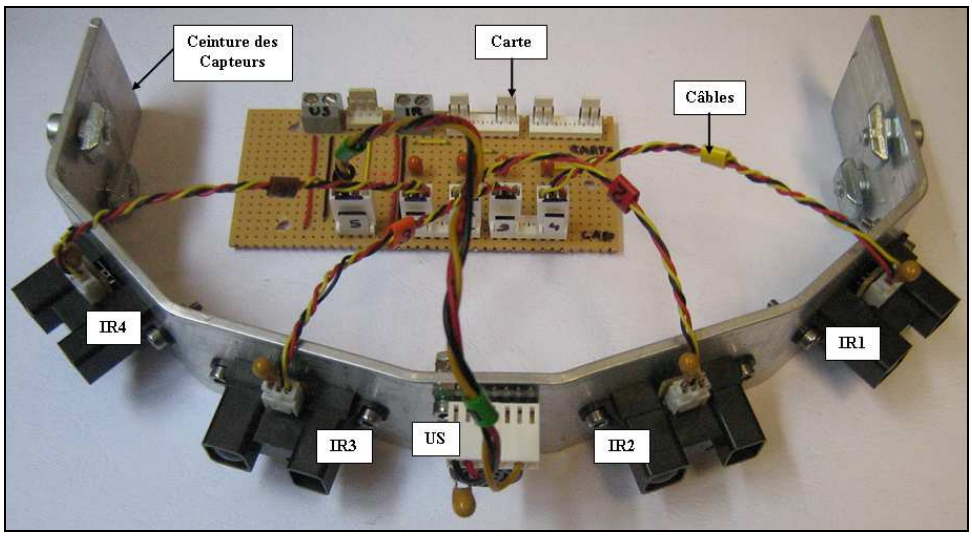

Fig. 2.14 : Ensemble du Système des Capteurs

Pour plus de détail sur le système des capteurs auparavant en service ainsi que d'autres sujets électroniques, on peut consulter les rapports référencés en bibliographie [VEN07] et [SIX07].

Résultats et conclusions finales.

Pour vérifier que la solution entreprise est satisfaisante, on va répéter un essai identique à ceux présentés dans cette étude ; plus exactement celui où tous les capteurs sont connectés, toujours en mesurant avec le capteur IR4 (le cas le plus critique côté bruit).

Les paramètres de cet essai sont restés invariables sauf le logiciel utilisé. Comme il a été dit, ce logiciel a été développé et conçu pour avoir plus de fonctionnalités jusqu'à la version actuelle (*Enregistrer v2.4*). Les mesures statistiques de cette version du logiciel sont arrondies. De même que les autres résultats de l'étude, le tableau qui contient toutes les données de ce nouveau prélèvement est présenté dans l'annexe A2.

Comme auparavant, on calcule le diagramme de boîte appliqué sur la variance à travers du logiciel *STATGRAPHICS Centurion XV* en version d'évaluation. Une fois les résultats obtenus, on compare les résultats avec le cas antérieur où il n'y avait pas de filtre.

Voici le diagramme de boîte qui synthétise et permet de comparer les données des tableaux de la figure A2.3 et de celle A2.4 :

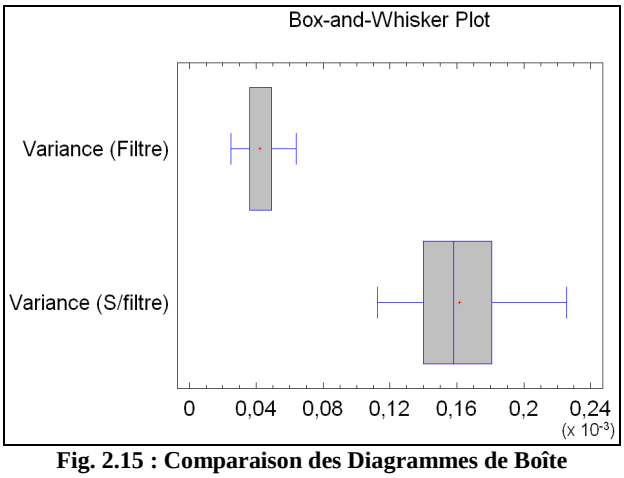

Fig. 2.15 : Comparaison des Diagrammes de Boîte

D'un coup d'œil, on peut confirmer que les filtres ont réduit considérablement le bruit et par conséquent, la variance du dernier essai. La moyenne des variances est passée de $0{,}16148 \cdot 10^{-3}\ V^2$ à $0{,}0424 \cdot 10^{-3}\ V^2$. On observe aussi que les variances sont plus stables qu'auparavant puisque la variation maximale des variances et plus petite maintenant.

Pour confirmer ce fait, on fera un contraste d'hypothèses. L'hypothèse nulle signifiera que les deux moyennes des variances sont égales et alors, que les filtres mis en place ne sont pas suffisamment effectifs. D'ailleurs, l'hypothèse alternative signifiera que le bruit des cas est différent et vérifiera le succès du travail. Voici le résultat : on rejette l'hypothèse nulle avec un 95% de confiance. Si l'on va plus loin, effectivement grâce au filtre conçu, on a réduit la moyenne des variances et donc, le bruit ; même au-dessous de la moyenne qu'on a obtenue dans l'essai du 1er cas (uniquement le capteur 4 connecté), qui était de $0{,}0545377 \cdot 10^{-3}\ V^2$.

Pour estimer l'erreur typique en termes de distance, on applique un étalonnage polynomial de 4e degré tel qu'on obtiendra un peu plus bas dans ce rapport. Le polynôme appliqué sera évalué dans la moyenne des moyennes du dernier essai et dans ce point plus l'écart type moyen ; c'est-à-dire dans 0,712328 V et dans $0{,}712318 + \sqrt{(0{,}0424 \cdot 10^{-3})}$ V. Voici le polynôme :

$$cm = 17{,}0095 \cdot volt^4 - 137{,}5219 \cdot volt^3 + 413{,}2059 \cdot volt^2 - 568{,}9687 \cdot volt + 347{,}4160$$

Fig. 2.16 : Polynôme d'étalonnage du capteur étudié

L'expression qui permettra de calculer cette erreur est montrée ci-dessous :

$$erreur = pol(0{,}712328) - pol\left(0{,}712328 + \sqrt{0{,}0424 \cdot 10^{-3}}\right)$$

Fig. 2.17 : Formule pour Obtenir l'Erreur Typique

La formule de la figure 8.4 fournit : 1,0674 cm. Soit une erreur au peu près d'un centimètre, cet écart est bien acceptable pour l'application qu'on va développer. De plus, cette erreur est identique pour tous les autres capteurs et donc le problème du bruit est résolu.

2.2.2. Partie programmée.

Une fois le signal est presque débarrassé de bruit, on est en prêt à réaliser un traitement informatique des mesures pour aboutir à des données exploitables. Ces processus sont toujours programmés sur *Visual C++* et dans le calculateur embarqué sur le robot.

2.2.2.1. Filtre numérique de 1er ordre.

Le filtre ici décrit fut mis en place en combinaison avec le filtre analogique expliqué dans le paragraphe 2.2.1. En outre, en tant qu'une partie de la solution entreprise, les résultats finals comptent aussi sur l'effet de ce filtre. Voyons maintenant comment il a été construit.

La fonction de transfert d'un filtre numérique de 1er ordre est montrée à la figure 2.18 :

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1-\alpha}{1-\alpha \cdot z^{-1}}; \quad \alpha \in (0,1), \alpha = e^{-0{,}1/TAU}$$

Fig. 2.18 : Fonction de Transfer du Filtre Numérique de 1er Ordre

De cette fonction, en opérant on arrive jusqu'à celle autre :

$$y(n) = (1-\alpha)\cdot x(n) + \alpha \cdot y(n-1); \quad \alpha \in (0,1), \alpha = e^{-0,1/TAU}$$

Fig. 2.19 : Equation du Filtre Numérique de 1er Ordre

A la vue du résultat, remarquons que ce filtre est récursif et qu'il faudra toujours avoir, au moins, la dernière donnée calculée pour obtenir la sortie dans un moment donné. Il faut aussi calculer le paramètre α du filtre, ce qui est fait à travers de la constante du temps τ ; dont sa valeur a été fixée à 0,1 secondes auparavant.

Effectivement, pour tester l'effet sur la dynamique du système, ce filtre a été déjà mis en place dans les applications *Enregistrer v1.0 et v2.4* et on peut consulter son implémentation dans l'annexe A7.

Pour finir, spécifions que ce filtre est du type *RII* ou à réponse impulsionnelle infinie. C'est pour cela que celui-ci a une nature récursive. Pour plus de renseignements par rapport à ces filtres consulter la référence bibliographique [BEL93].

2.2.2.2. Etalonnage.

On a déjà parlé du but de l'étalonnage, soit d'obtenir une conversion, la plus précise possible, d'une mesure des capteurs, acquise par la carte NI en volts, à une donnée traitée en centimètres.

De même, on a mentionné que, pour les capteurs infrarouges, cet étalonnage est réalisé à travers d'un polynôme de 4e degré. Cependant, pour le capteur à ultrasons, l'étalonnage suit une fonction linéaire, comme il fallait espérer. Ensuite, il ne manque que connaître les coefficients des fonctions et pour y arriver, une autre étude a été conçue et réalisée.

Conception de l'essai.

Dans cet essai, on va recueillir des différentes mesures pour des distances établies et connues. Ainsi, pour chaque distance on va réaliser 5 essais, de 50 échantillons chacun, et on va prendre leurs moyennes. Finalement, on fera la moyenne des 5 essais à la même distance (moyenne des moyennes) et on obtiendra un tableau en rapportant les volts aux centimètres.

Les distances qui vont être essayées pour les capteurs infrarouges sont les suivantes : 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 110 et 120 centimètres. Pour le choix de ces distances on a pris en compte la sensibilité et le rang de distance des capteurs infrarouges. De même, les distances essayées pour le capteur à ultrasons sont : 20, 40, 60, 80, 100, 120, 140 et 160 centimètres. Notons que pour une fonction linéaire ces mesures suffissent et qu'au-delà de 160 centimètres le cône de perception du capteur (déjà de 85 centimètres de diamètre) empêche de prendre des mesures dans un couloir.

Le protocole d'essais et de mesures est réalisé à l'aide du logiciel déjà présenté qui s'appelle *Enregistrer v2.4* et son code est expliqué dans l'annexe A7.

Une fois on ait le tableau qui rapporte les volts aux centimètres, celui-ci sera introduit à travers de deux vecteurs dans le logiciel de calcul technique appelé *Matlab* (l'un des vecteurs pour les volts et l'autre pour les centimètres). A l'aide d'un fichier de type *.m* construit exprès pour ce but, on fera un ajustement des mesures à polynômes de différents degrés et on présentera les résultats. On parlera de cela plus précisément lors

de son utilisation dans le paragraphe suivant. Le code de ce fichier, appelé *interpoler.m*, est fourni dans l'annexe A4.

Résultats de l'essai.

Bien que tous les tableaux de données et d'autres résultats sont présentés dans l'annexe A3, on va expliquer ici et comme exemple, le procédé suivi pour étalonner le capteur IR4. Ainsi on montre le tableau avec les données intermédiaires de l'essai de ce capteur :

Essai	Distance [cm]	Mesure 1 [mV]	Mesure 2 [mV]	Mesure 3 [mV]	Mesure 4 [mV]	Mesure 5 [mV]	Moyenne [mV]
1	20	2721	2721	2722	2721	2721	2721,2
2	25	2466	2465	2466	2466	2466	2465,8
3	30	2183	2183	2183	2182	2183	2182,8
4	35	1908	1907	1909	1908	1908	1908
5	40	1703	1705	1704	1704	1705	1704,2
6	45	1539	1538	1538	1537	1538	1538
7	50	1408	1408	1409	1408	1408	1408,2
8	55	1256	1257	1257	1257	1256	1256,6
9	60	1167	1166	1166	1166	1165	1166
10	65	1048	1048	1047	1048	1048	1047,8
11	70	1010	1009	1009	1009	1010	1009,4
12	75	953,164	953,177	953,319	952,372	953,505	953,1074
13	80	933,387	933,483	933,598	934,635	934,427	933,906
14	85	845,914	845,056	845,409	844,711	846,192	845,4564
15	90	806,493	806,176	806,649	805,963	806,925	806,4412
16	95	786,713	786,758	787,002	787,209	786,177	786,7718
17	100	767,718	768,054	768,491	768,023	767,609	767,979
18	110	701,27	701,143	701,181	699,916	700,349	700,7718
19	120	633,714	634,144	633,363	633,877	632,969	633,6134

Fig. 2.20 : Tableau de Mesures Brutes pour l'Etalonnage du Capteur IR4

De telle façon, on introduira dans le fichier *.m* de *matlab* les vecteurs suivants, obtenus du tableau précédant :

Essai	Moyenne [V]	Distance [cm]
1	0.6336134	120
2	0.7007718	110
3	0.767979	100
4	0.7867718	95
5	0.8064412	90
6	0.8454564	85
7	0.933906	80
8	0.9531074	75
9	1.0094	70
10	1.0478	65
11	1.166	60
12	1.2566	55
13	1.4082	50
14	1.538	45
15	1.7042	40
16	1.908	35
17	2.1828	30
18	2.4658	25
19	2.7212	20

Fig. 2.21 : Données pour l'étalonnage du Capteur IR4

Notons, dans la figure 2.21, que la première colonne présentée est celle des moyennes des voltages et que la deuxième est celle des distances. De même, peut être remarqué que les valeurs ont été inversées, en rangeant les données par les moyennes des voltages et de mineur à majeur. Il n'est pas fait pour les calculs mais pour améliorer la compréhension du procès. Il ne faut pas oublier que le résultat espéré est une fonction tel que *distance = f(voltage)*. Finalement, les virgules des décimales ont été remplacées par des points pour pouvoir coller directement sur la fenêtre de *Matlab*.

Pour passer toutes ces donnés à *Matlab*, on va coller les vecteurs dans la fenêtre de commandes:

```
>> volt=['coller ici'
]

volt =

    0.6336
    0.7008
    0.7680
      ...
    2.7212

>>
```

```
>> cm=['coller ici'
]

cm =

   120
   110
   100
   ...
    20

>>
```

Fig. 2.22 : Code pour l'Analyse sur Matlab

Remarquons qu'à la figure 2.22 le vecteur *volt* correspond à la copie de la colonne *Moyenne [V]* et qu'il est lié au vecteur *cm*, la copie de la colonne *Distance [cm]*. Aussi, on peut dire que la figure 2.22 est une représentation et que, si l'on suivait les pas donnés, ce qu'on verrait dans la fenêtre de *Matlab* serait une extension de cette figure avec toutes les données.

Ensuite, on utilise le fichier *interpoler.m* qui, à partir des vecteurs précédents, calcule les coefficients des polynômes, de 5^e, 4^e et 3^e degré, qui s'ajustent le mieux aux mesures prises. De plus, il calcule la moyenne des écarts types estimés pour chaque polynôme ainsi que l'erreur relative moyenne.

Si c'était le cas du capteur à ultrasons, il ne calculerait qu'une fonction linéaire. Pour cela il faudra le lui indiquer en lui passant la valeur 1 dans le dernier des arguments ; autrement cet argument doit être 0.

Tout cela peut être remarqué dans la figure suivante :

```
>> [p5,p4,p3,p1]=interpoler(volt,dist,0)

ecart_type_moyen5 =
    1.8419

ecart_type_moyen4 =
    1.8473

ecart_type_moyen3 =
    2.7608

erreur_relative5 =
    1.2779

erreur_relative4 =
    1.9147

erreur_relative3 =
    3.8961

p5 =
   -9.0780   91.8606 -371.9307  759.4570 -808.9741  409.7753

p4 =
   17.0095 -137.5219  413.2059 -568.9687  347.4160

p3 =
  -25.6051  155.1927 -324.3017  267.1904

p1 =
     0

>>
```

Fig. 2.23 : Sorties du Fichier Interpoler.m sur Matlab

L'écart type qui est obtenu dans la figure 2.23 est calculé à partir des fonctions de *Matlab polyfit* et *polyval*. La fonction *polyfit* est celle qui fait l'interpolation des polynômes, mais elle donne aussi une structure qui contient les champs de l'élément triangulaire à partir d'un QR décomposition de la matrice de Vandermonde, les degrés de liberté et la norme de résidus (Pour plus précision voir l'aide de *Matlab* en tapant *help polyfit* dans la fenêtre de commande).

Postérieurement, cette structure est introduite dans la fonction *polyval* avec les coefficients des polynômes et leurs points d'évaluation. Cela donne un écart type estimé en supposant une future observation (Egalement, on peut voir l'aide de *Matlab* en tapant *help polyval* dans la fenêtre de commande).

L'écart type ici calculé est la moyenne pour chaque polynôme de tous les écarts types estimés à chaque point d'évaluation ou voltage de l'essai. Ces écarts types sont individuellement représentés à droite dans la figure 2.25 pour chaque polynôme.

Par ailleurs, l'erreur relative est calculée à travers de quelques opérations simples. Pour l'obtenir il faut suivre l'équation suivante :

$$\varepsilon^{X}_{RELATIVE}(\%) = \frac{\left| Y^{X}_{RÉELLE} - Y^{X}_{INTERPOLÉE} \right|}{Y^{X}_{RÉELLE}} \cdot 100$$

Fig. 2.24 : Equation pour Calculer l'Erreur Relative

Selon la figure 2.24, il faut obtenir la valeur absolue de la soustraction entre une distance réelle et celle qu'on aurait en appliquant le polynôme correspondant, toujours pour le même voltage X ; ainsi on a l'erreur commise en un terme non relatif. Ensuite, cette erreur est divisée par la même distance réelle utilisée dans la soustraction pour la relativiser et multipliée par 100 pour obtenir finalement un pourcentage.

L'erreur relative montrée dans la figure 2.23 est la moyenne pour chaque polynôme de toutes les erreurs relatives calculées à chaque voltage de l'essai.

En fin, on obtient une figure qui montre un graphique des mesures réelles et les résultats de chaque interpolation. Elle montre aussi l'estimation de l'écart typique pour chaque polynôme. Voici la figure 2.25 :

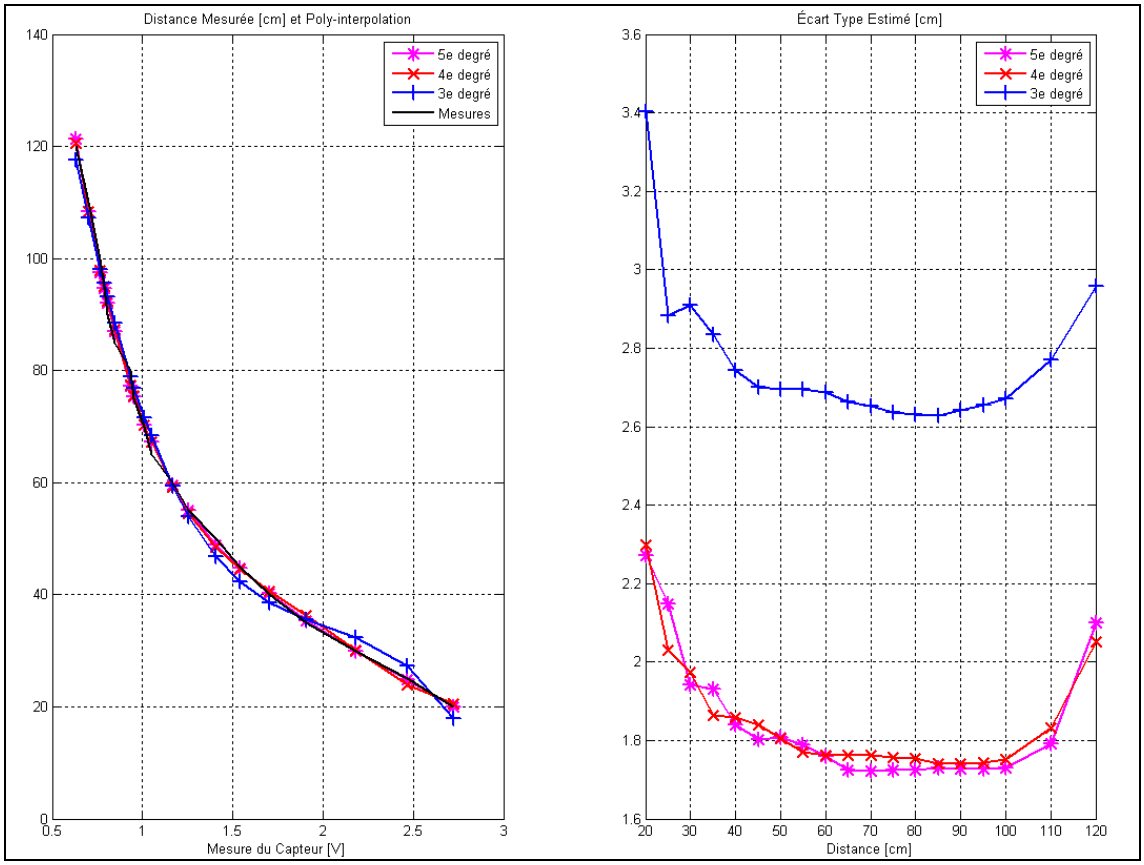

Fig. 2.25 : Graphique des Interpolations sur Matlab

Analyse de l'essai.

Cette analyse prend les données que le fichier *interpoler.m* a calculées et a pour but d'arriver à une solution par rapport à quel étalonnage choisir, soit degré du polynôme, soit coefficients de celui-ci.

C'est ainsi qu'on apprécie que le polynôme de 5^{e} degré s'ajuste assez bien à la courbe des mesures, tandis que le polynôme de 3^{e} degré présente une forte erreur et qu'avec le polynôme de 4^{e} degré les performances sont moyennes. L'estimation démontre que le polynôme de 5^{e} degré est celui qui suit le mieux la courbe des mesures.

Les données de la figure 2.23 renforcent ces remarques. Pourtant il faut dire que les écarts types moyens ainsi que les erreurs relatives des polynômes de 5^{e} et 4^{e} degré sont très semblables.

Conclusions et résultats de l'essai.

Selon cette analyse, on estime que le polynôme de 5^{e} degré suit le mieux aux mesures du capteur IR4. Pourtant, il est un peu plus lourd travailler avec lui et de même, le polynôme de 4^{e} degré a des performances presque pareilles.

C'est pour cela qu'en un compromis entre complexité, vitesse et précision et selon le déjà dit, on va choisir le polynôme de 4^{e} degré pour étalonner tous les capteurs

infrarouges. De telle façon, on est arrivé à obtenir une fonction facile à implémenter en *Visual C++* et qui convertit les volts ou millisecondes obtenus (mesure réelle à travers des capteurs) en centimètres (mesure souhaitée).

Pour finir, en répétant les pas indiqués dans ce paragraphe pour tous les capteurs on obtient les fonctions d'étalonnage. Ensuite, présentons les différents polynômes choisis pour l'étalonnage des capteurs :

$$cm = 13{,}6409 \cdot volt^4 - 104{,}6611 \cdot volt^3 + 300{,}0149 \cdot volt^2 - 402{,}2714 \cdot volt + 254{,}2031$$

Fig. 2.26 : Polynôme d'étalonnage du capteur IR1

$$cm = 12{,}6296 \cdot volt^4 - 98{,}2016 \cdot volt^3 + 284{,}8906 \cdot volt^2 - 388{,}02 \cdot volt + 252{,}8794$$

Fig. 2.27 : Polynôme d'étalonnage du capteur IR2

$$cm = 13{,}5935 \cdot volt^4 - 103{,}9796 \cdot volt^3 + 297{,}3103 \cdot volt^2 - 397{,}7897 \cdot volt + 250{,}7731$$

Fig. 2.28 : Polynôme d'étalonnage du capteur IR3

$$cm = 17{,}0095 \cdot volt^4 - 137{,}5219 \cdot volt^3 + 413{,}2059 \cdot volt^2 - 568{,}9687 \cdot volt + 347{,}416$$

Fig. 2.29 : Polynôme d'étalonnage du capteur IR4

$$cm = 17{,}5884 \cdot ms + 2{,}094$$

Fig. 2.30 : Polynôme d'étalonnage du capteur à ultrasons

3. MODELISATION DES CAPTEURS.

On a déjà des données exploitables obtenus grâce au système d'acquisition qui a été amélioré et développé. On aura donc un ensemble de mesures de distance (données en centimètres) associé à chaque capteur en provenant des capteurs.

Pourtant, pour aboutir à fusionner des différentes données et implanter ultérieurement un filtre de Kalman, il faudra disposer d'un modèle cinématique qui soit capable d'estimer les distances des capteurs à travers des données du mouvement du robot. Voici le schéma où on doit arriver :

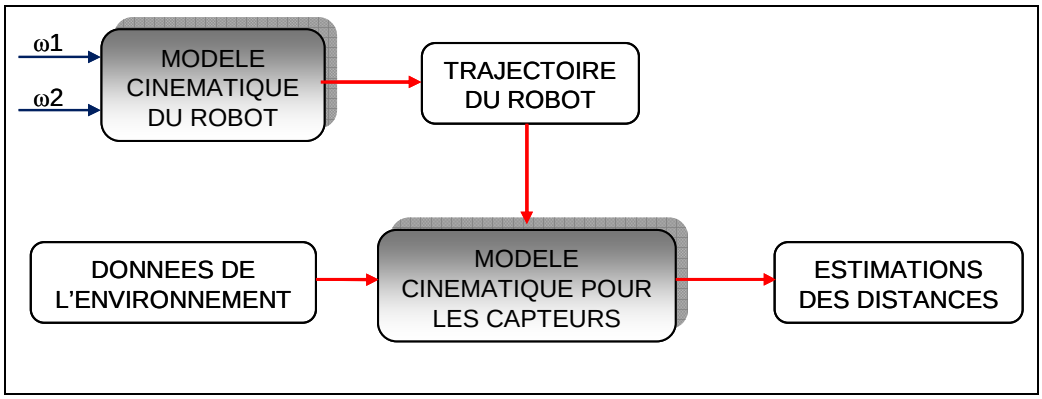

Fig. 3.1 : Schéma pour la Conception du Modèle pour les Capteurs

Soient ω1 et ω2 les vitesses angulaires des roues du robot, un modèle cinématique du robot est déjà développé dans les années précédentes. Ce modèle apporte les paramètres nécessaires pour connaître la trajectoire suivie par le robot.

En ce point, il s'agit de réaliser un autre modèle pour les capteurs qui soit capable de prendre les paramètres de la trajectoire pour obtenir une estimation des distances que les capteurs devraient mesurer étant donné un environnement. Cette année on n'a pris en compte que le cas du robot seul, sans fauteuil accroché. On estime que

tout le travail ici fait est valide pour le fauteuil accroché avec une adaptation minimale. Voici le développement de ce modèle.

3.1. Spécifications et caractéristiques du modèle.

Vu le motif par lequel le modèle des capteurs s'avère nécessaire, on va décrire ce qu'on espère de lui. Tout d'abord il faut dire que pour construire le modèle avant de l'implanter sur le robot, il a été conçu sous le logiciel *Matlab*, ce qui correspond a une parfaite plateforme d'essais. Voyons les besoins qui sont attendus.

Ce modèle doit être capable de construire des vecteurs de distances avec les mesures théoriques des capteurs du robot et de les représenter à partir d'un environnement et d'une trajectoire suivie par le robot ; au cas où cette trajectoire serait réalisée idéalement (sans glissement des roues sur le sol et d'autres déviations). Une fois ce modèle sera implanté sur le logiciel du robot, celui-ci aura une estimation des mesures des capteurs en fonction de son mouvement et enfin on pourra réaliser un filtre de Kalman.

Le développement du modèle a été réalisé grâce au logiciel *Matlab*. Cette application permet de programmer des codes de façon directe, sans se soucier des considérations habituelles de la programmation. C'est ainsi qu'on a construit le code appelé « modele_capteurs.m ».

Ce modèle s'intègre et reprend le travail réalisé les années précédentes. Nous avons utilisé les simulations et les modèles cinématiques du robot *HAMMI* lorsqu'il est seul, développés l'année dernière. Pour s'adapter aux différents fichiers existants, comprenant chacun des trajectoires ou caractéristiques propres, le modèle est complètement indépendant de ces trajectoires. Il suffit d'inclure l'instruction fournie à la ligne 110 à la fin du fichier « programme_principal.m » de n'importe quelle trajectoire :

```
105
106      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
107      %%%%%% Simulation Cinématique des Capteurs %%%%%
108      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
109
110 -    modele_capteurs(t,x);
111
```

Fig. 3.2 : Code pour les Fichiers « programme _principal »

Du coup, on a créé un modèle qui s'intègre à différentes conditions en changeant certains paramètres de l'essai. Ces paramètres sont inclus au début du code et comportent les données géométriques des capteurs ainsi que celles de l'environnement.

D'un autre côté, le modèle reçoit du fichier « programme_principal.m » un vecteur de temps ***t*** où on observe la discrétisation de la trajectoire du robot et une matrice ***x*** qu'apporte principalement la position (***Xr***, ***Yr***) et orientation (***θr***) du centre de gravité du robot (point ***Ar***) à chaque instant de la discrétisation. Pour plus de précision des trajectoires développées les années précédentes, on peut consulter le rapport de S.Rubrecht, référencé en bibliographie [RUB07].

La figure suivante définit les paramètres géométriques des capteurs :

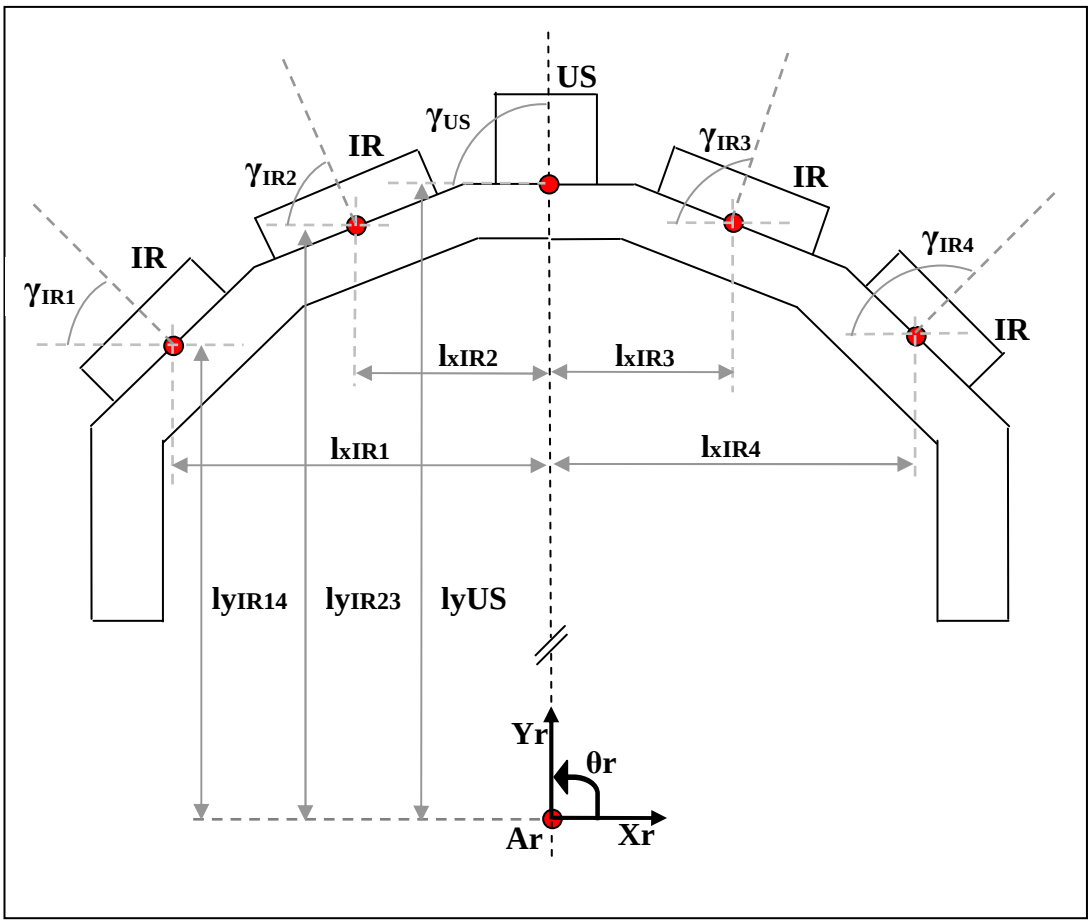

Fig. 3.3 : Paramètres Géométriques des Capteurs

Les valeurs numériques de tous ces paramètres sont listées dans le tableau de la figure 3.4 :

Paramètre	Valeur
γ_{IR1}	50º
γ_{IR2}	72º
γ_{IR3}	110º
γ_{IR4}	135º
γ_{US}	90º
l_{yIR14}	0,329 m
l_{yIR23}	0,355 m
l_{yUS}	0,363 m
l_{xIR1}	-0,091 m
l_{xIR2}	-0,044 m
l_{xIR3}	0,042 m
l_{xIR4}	0,090 m

Fig. 3.4 : Tableau des Valeurs des Paramètres Géométriques

Pour plus d'informations sur la construction de la ceinture de capteurs on peut consulter les rapports référencés en bibliographie [VEN07] et [GEN08].

Quant aux paramètres de l'environnement, on va fixer quatre lignes droites représentant des murs où les rayons des capteurs vont se réfléchir. Ces lignes dessineront toujours un rectangle grâce à leurs coordonnées par rapport à l'origine (Xo, Yo), commune à celle du fichier « programme_principal.m ». Les lignes sont ***ymur2***, ***ymur1***, ***xmur2*** et ***xmur1*** ; selon leur orientation par rapport à l'axe *x* ou à l'axe *y*. Les

paramètres de définition de ces murs doivent aussi être spécifiés à la fin du code « modele_capteurs.m ».

D'un autre côté, le fichier « modele_capteurs.m » va recevoir, des différents fichiers « programme_principal.m » où il s'intègre, le vecteur du temps de simulation et le vecteur de coordonnées de la trajectoire du robot, à travers du code de la figure 3.2 déjà vu. Pour ce modèle, les données utilisées sont celles des points ***Ar*** (***Xr*** matrice ***x*** colonne 1, ***Yr*** matrice ***x*** colonne 2 et ***θr*** matrice ***x*** colonne 3) et le temps de chacun d'eux.

Grâce à toute cette information et d'une façon automatique et opaque, « modele_capteurs.m » va calculer la distance de tous les capteurs de la ceinture vue dans la figure 3.3. Ce code va toujours distinguer par lui-même si les rayons impactent tel ou tel mur en fonction des conditions concrètes. Après avoir rempli et montré une matrice avec les résultats et après avoir dessiné les murs dans la figure de la trajectoire du robot, le code va dessiner l'évolution les différentes mesures au cours du temps.

Enfin, on a également intégré dans le code les limites de perception des capteurs. Ces limites varient de 20 à 120 centimètres pour les capteurs infrarouges et de 10 à 600 centimètres pour le capteur à ultrasons.

3.2. Méthode de calcul.

Pour calculer toutes les distances des capteurs il a fallu développer plusieurs formules que l'on présente plus bas.

Tout d'abord, pour le calcul de la distance en fonction des paramètres géométriques des capteurs et de la position du robot, il faut prendre en compte la figure suivante :

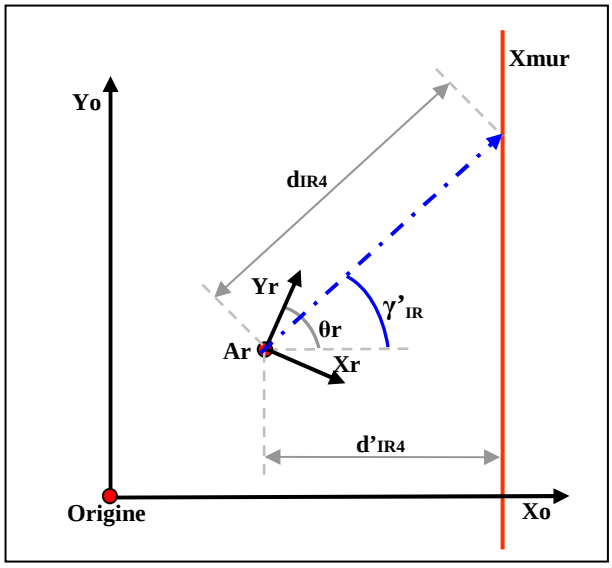

Fig. 3.5 : Principe de Calcul

Ainsi, pour obtenir la distance ***dIR4***, il faudra calculer d'abord la distance ***d'IR4*** et la diviser par le cosinus de l'angle ***γ'IR4***. La distance ***d'IR4*** est calculée à travers de la coordonnée ***Xr*** du robot et des paramètres ***lyIR14*** et ***lxIR4*** ; ceux derniers multipliés par sinus et cosinus pour obtenir la projection sur l'axe *X*. L'angle ***γ'IR4*** est l'aboutissement de la combinaison de l'angle ***γIR4*** et de ***θr***. Voici la formule qui permet de calculer la distance souhaitée :

$$d_{IR4} = \frac{d'_{IR4}}{\cos(\gamma'_{IR4})} = \frac{Xr - Xmur - lyIR14 \cdot \sin(pi/2 - \theta r) + lxIR4 \cdot \cos(pi/2 - \theta r)}{\cos((pi/2 + \theta r) - \gamma_{IR4})}$$

Fig. 3.6 : Equation de Calcul

Les différentes équations qui permettraient de calculer toutes les autres distances suivent la même logique et dynamique et peuvent s'apprécier dans le code qui est présenté plus bas.

Pourtant, il faut s'arrêter ici pour montrer un cas qui doit être étudié et pris en compte. Ce cas est celui de la figure 3.7 :

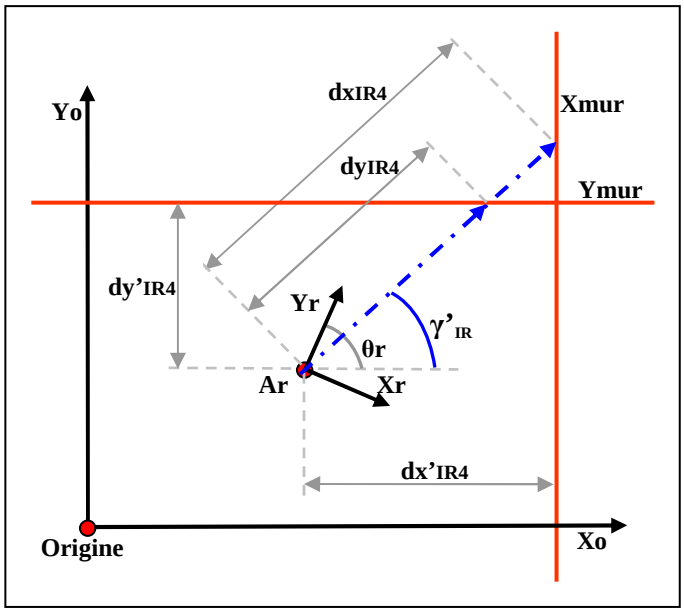

Fig. 3.7 : Principe de Calcul : Cas Spécial

Voici ce qu'il peut se passer si le rayon impacte sur l'un des murs qui composent le rectangle défini par les quatre lignes et qui est parallèle à l'axe ***Xo*** (***ymur***). Le principe qui fut montré dans la figure 3.5 prendrait comme mesure valide la distance ***dxIR4***, tandis que la mesure réelle serait la ***dyIR4***. Cette dernière peut néanmoins être calculée selon le même principe à travers de la distance ***dy'IR4*** au lieu de la ***dx'IR4***. Le code qui est conçu calculera, dans ce cas, toutes les deux distances (***dxIR4*** et ***dyIR4***) et prendra la plus petite d'elles, ce qui correspond à la mesure correcte.

Mais il reste encore une autre des hypothèses à considérer. En remarquant l'équation de la figure 3.6 on peut apprécier que le dénominateur de l'expression est une fonction cosinus qui va être nulle lorsque son angle vaut ***pi/2*** radians et tous les multiples de la forme ***pi/2***, ***3pi/2***, ***5pi/2***… Ce cosinus va donc provoquer une division par zéro qu'il faut éviter.

Pour éviter ce phénomène, on remarque, par exemple, la valeur du paramètre ***γIR4*** et on analyse les intervalles de ***θr*** où chaque équation est valable. De cette façon, cette étude est arrivée au résultat de la figure suivante, dans laquelle on peut voir les intervalles, numérotés de l'un au quatre, pour le capteur IR4 :

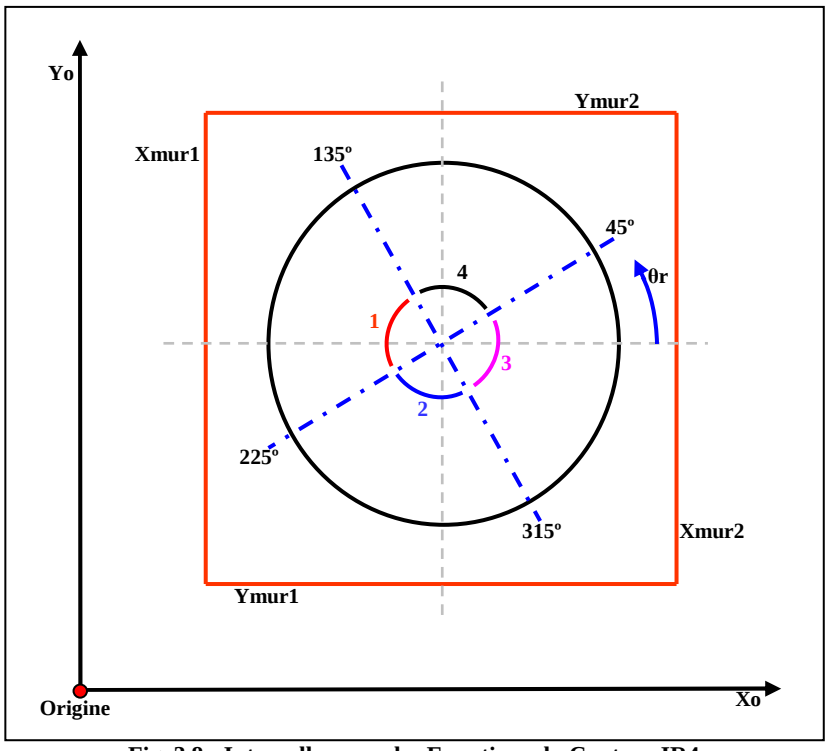

Fig. 3.8 : Intervalles pour les Equations du Capteur IR4

La figure 3.8 montre ces quatre intervalles qui indiquent que les rayons du capteur ***IR4*** vont impacter sur des différents murs. Par exemple, dans l'intervalle 1, il faudra prendre en compte les murs ***xmur1*** et ***ymur2*** tandis que, dans l'intervalle 3, ce seront les murs ***xmur2*** et ***ymur1*** où le rayon sera susceptible de se réfléchir.

Ensuite, on présente le résultat numérique de l'analyse précédente pour tous les capteurs sous forme de tableau :

Intervalle	Murs	IR1	IR2	IR3	IR4	US
1	***xmur1, ymur2***	(50º, 140º]	(72º, 162º]	(110º, 200º]	(135º, 225º]	(90º, 180º]
2	***xmur1, ymur1***	(140º, 230º]	(162º, 252º]	(200º, 290º]	(225º, 315º]	(180º, 270º]
3	***xmur2, ymur1***	(230º, 320º]	(252º, 342º]	(290º, 20º]	(315º, 45º]	(270º, 360º]
4	***xmur2, ymur2***	(320º, 50º]	(342º, 72º]	(20º, 110º]	(45º, 135º]	(0º, 90º]

Fig. 3.9 : Intervalles pour Toutes les Equations

Ici, il faut faire quelques remarques importantes. Tout d'abord, il faut remarquer que, pour que le modèle soit neutre et valable pour n'importe quelle loi de commande, les intervalles doivent comprendre aussi des valeurs de $\boldsymbol{\theta r}$ négatives. Pour les établir, il n'y a qu'à soustraire 360º à tous les angles des intervalles de la figure 3.9. En raison de simplicité et facilité de compréhension, ces intervalles n'ont pas été indiqués dans le tableau mais ils ont été pris en compte et peuvent être consultés directement sur le code fourni un peu plus bas.

Ensuite, pour interpréter le code, il faut préciser que les angles n'ont pas une valeur numérique telle qu'elles ont été écrites dans le tableau de la figure 3.9. Au lieu de cela, on va trouver une expression qui dépend de la valeur des différents **γIRx** ou **γUS**. De cette façon le modèle est flexible pour accueillir une nouvelle configuration de la ceinture des capteurs.

De même, on doit savoir que, au cas où la trajectoire du robot serait très compliquée, la valeur de $\boldsymbol{\theta r}$ dépassera les ±360º. C'est pour cela que plus bas, dans la présentation du code, on trouvera quelques lignes qui ont pour but de réduire cette angle au rang de 0 à ±360º.

3.3. Validation du modèle.

Dans ce paragraphe, on va tester deux types de trajectoires, toujours avec des lois de commande de l'année dernière, et on va justifier les résultats pour aboutir à valider le modèle qu'on vient de construire.

La première de ces trajectoires est celle qu'on peut observer à la figure 3.10. Il s'agit du suivi d'une ligne droite sans fauteuil qui s'étend jusqu'à l'infini et qui est située en $\boldsymbol{x} = 0$. Pour savoir plus de cette trajectoire, on peut s'adresser au rapport de S.Rubrecht, référencé en bibliographie [RUB07]. Le robot part des coordonnées $\boldsymbol{Xr}$ = 0,25, $\boldsymbol{Yr}$ = 0 et $\boldsymbol{\theta r}$ = 135º. Les murs sont situés en $\boldsymbol{xmur1}$ = -0,5, $\boldsymbol{xmur2}$ = 0,5, $\boldsymbol{ymur1}$ = 0 et $\boldsymbol{ymur2}$ = 4.

On présente alors la simulation cinématique du robot où, éventuellement, on a dessiné les rayons des capteurs et étiqueté les murs :

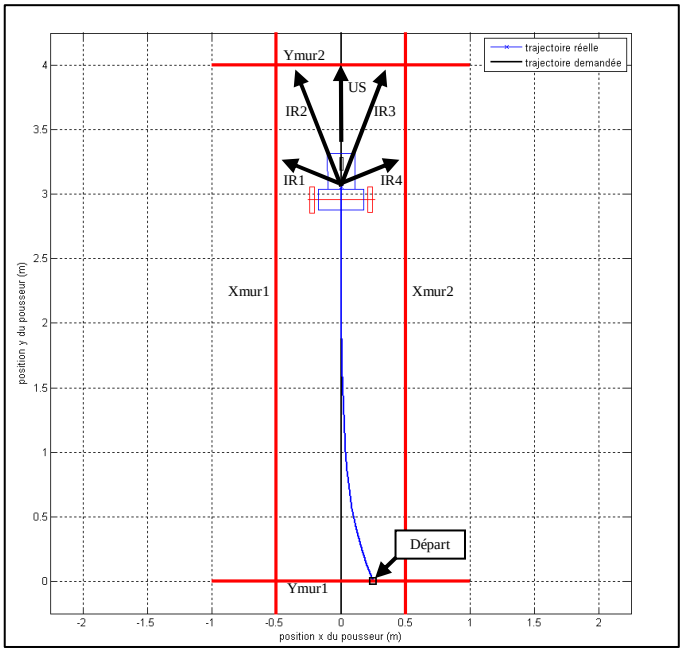

Fig. 3.10 : Trajectoire « Suivi de Ligne Droite »

Les coordonnées, résultat de la cinématique du robot et du suivi de la trajectoire, que le code de l'année dernière rend sont introduites dans le fichier « modele_capteurs.m », selon l'instruction de la figure 3.2. Ainsi, on obtient une série de distances correspondant à la situation du robot à chaque moment. Voici le résultat obtenu à la fin de l'exécution du code :

t	*Xr*	*Yr*	*θr*	IR1 [m]	IR2 [m]	IR3 [m]	IR4 [m]	US [m]
0	0.2500	0	2.3562	0.7529	0.8275	1.7747	4.0153	1.0607
0.1048	0.1959	0.1409	2.0325	0.7591	0.9601	3.8837	1.2983	1.5622
0.2095	0.1495	0.2805	1.9173	0.7511	1.0135	3.7196	1.0176	1.9125
0.3143	0.1130	0.4227	1.8379	0.7456	1.0607	3.5893	0.9263	2.3224
0.4190	0.0850	0.5668	1.7756	0.7451	1.1122	2.8863	0.8764	2.8767
0.5238	0.0637	0.7120	1.7264	0.7478	1.1658	2.2696	0.8451	3.3282
0.6286	0.0476	0.8578	1.6882	0.7521	1.2176	1.9706	0.8246	3.1640
0.7333	0.0356	1.0040	1.6590	0.7567	1.2647	1.8008	0.8108	3.0077
0.8381	0.0265	1.1504	1.6369	0.7610	1.3056	1.6955	0.8014	2.8558
0.9429	0.0198	1.2969	1.6202	0.7646	1.3397	1.6266	0.7948	2.7064
1.0476	0.0148	1.4435	1.6077	0.7676	1.3675	1.5798	0.7901	2.5582
1.1524	0.0110	1.5901	1.5983	0.7700	1.3896	1.5473	0.7868	2.4108
1.2571	0.0082	1.7368	1.5913	0.7719	1.4068	1.5242	0.7844	2.2637
1.3619	0.0061	1.8834	1.5861	0.7734	1.4202	1.5076	0.7827	2.1168
1.4667	0.0046	2.0301	1.5822	0.7745	1.4305	1.4956	0.7814	1.9701
1.5714	0.0034	2.1767	1.5793	0.7753	1.4382	1.4868	0.7805	1.8233
1.6762	0.0025	2.3234	1.5772	0.7760	1.4441	1.4804	0.7798	1.6766
1.7810	0.0019	2.4701	1.5755	0.7764	1.4486	1.4756	0.7793	1.5299
1.8857	0.0014	2.6167	1.5743	0.7768	1.4519	1.4702	0.7789	1.3833
1.9905	0.0011	2.7634	1.5734	0.7771	1.3172	1.3147	0.7787	1.2366
2.0952	0.0008	2.9101	1.5728	0.7773	1.1607	1.1591	0.7785	1.0899
2.2000	0.0006	3.0567	1.5723	0.7774	1.0043	1.0033	0.7783	0.9433

Fig. 3.11 : Résultats du Modèle

Dans ces résultats on peut faire trois remarques. D'abord, les ***θr*** sont exprimés en radians pour avoir une cohérence avec le code déjà développé. Ensuite, il faut dire que tous les paramètres géométriques du robot (tableau de la figure 3.4) ont été sélectionnés tels que les capteurs sont situés sur le point ***Ar*** du robot et la ceinture est parfaitement symétrique. Ainsi, c'est plus facile d'analyser les résultats obtenus. Enfin, la modélisation des limites de perception des capteurs a été enlevée pour cet essai ; ainsi on peut analyser l'évolution des différentes distances mesurées avec des variations plus grandes.

Une analyse plus rapide va être réalisée à partir de l'étude de la représentation graphique des distances au cours du temps. Cette représentation est affichée ci-dessous :

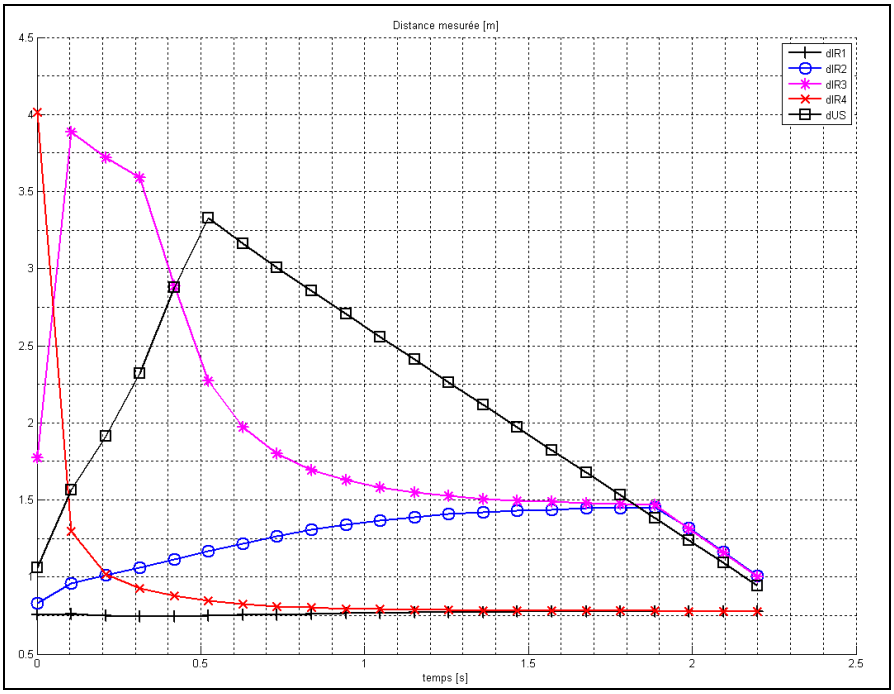

Fig. 3.12 : Représentation Graphique des Distances (Ligne Droite)

Pour analyser les résultats et valider le modèle, on présente les phénomènes remarquables et leurs interprétations :

- La distance mesurée par le capteur IR1 ne varie presque pas. Bien entendu, on peut remarquer que deux événements arrivent en même temps : le robot s'approche au mur gauche (la distance doit diminuer) et le robot s'aligne avec la trajectoire à suivre (***θr*** → 90º, la distance doit augmenter). La contribution des deux effets est semblable, donc la distance ne change pas d'une manière significative.
- Les capteurs symétriques (***IR1-IR4*** et ***IR2-IR3***) mesurent la même distance lorsque le robot s'aligne avec la trajectoire. Cela est prévisible puisque la ceinture et la géométrie sont supposées symétriques.
- La distance mesurée par le capteur ultrasons et les capteurs ***IR2*** et ***IR3*** diminue d'une façon linéaire à la fin de la simulation. Cela arrive lorsque les rayons impactent sur le mur du bout (***ymur2***) et le robot est presque aligné avec la ligne droite. A ce moment-là, la variation de la distance dépend uniquement de l'avancement du robot.
- Les capteurs ***IR3*** et ***IR4*** semblent avoir des tronçons différents. Pour le capteur ***IR4*** la séquence d'impact du rayon est la suivante : ***ymur2*** (diminution linéale) et ***xmur2*** (diminution non linéale). Pour le capteur ***IR3*** la séquence est la suivante : ***xmur1*** (augmentation linéale), ***ymur2*** (diminution linéale), ***xmur2*** (diminution non linéale) et ***ymur2*** (diminution linéale). C'est pour cela qu'on remarque ces tronçons.
- La distance du capteur ***US*** est moindre que celles des capteurs ***IR2*** et ***IR3*** à la fin de l'essai. Cela arrive parce que le rayon du capteur ***US*** est dirigé presque perpendiculaire au mur ***ymur2***. Pourtant, les autres capteurs ont un certain angle d'incidence qui augmente la distance parcourue des rayons.

De même, on peut prouver mathématiquement que les mesures du modèle coïncident avec celles des calculs.

Maintenant, on réalise une trajectoire circulaire qui va être suivie par le robot pendant environ un tour et un quart. La cinématique de l'année dernière, qui peut être consultée en bibliographie [RUB07], montre le parcours suivant :

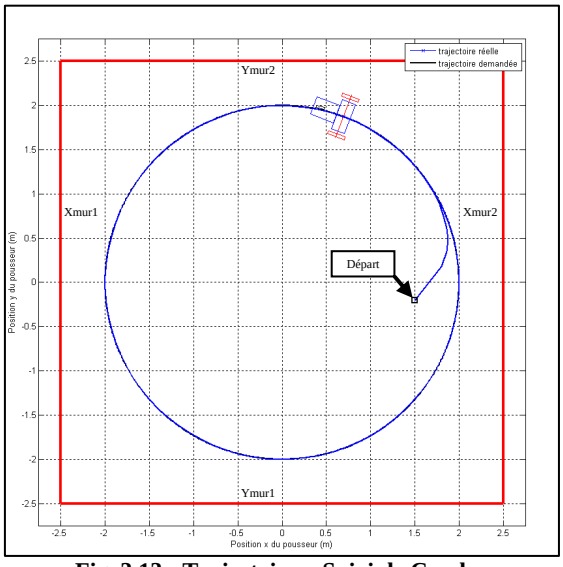

Fig. 3.13 : Trajectoire « Suivi de Cercle»

Tel qu'on voit, cette trajectoire nécessite des murs suffisamment éloignés. Si tous les capteurs sont situés sur le point ***Ar*** et la ceinture est supposée symétrique, voici la représentation des distances correspondante :

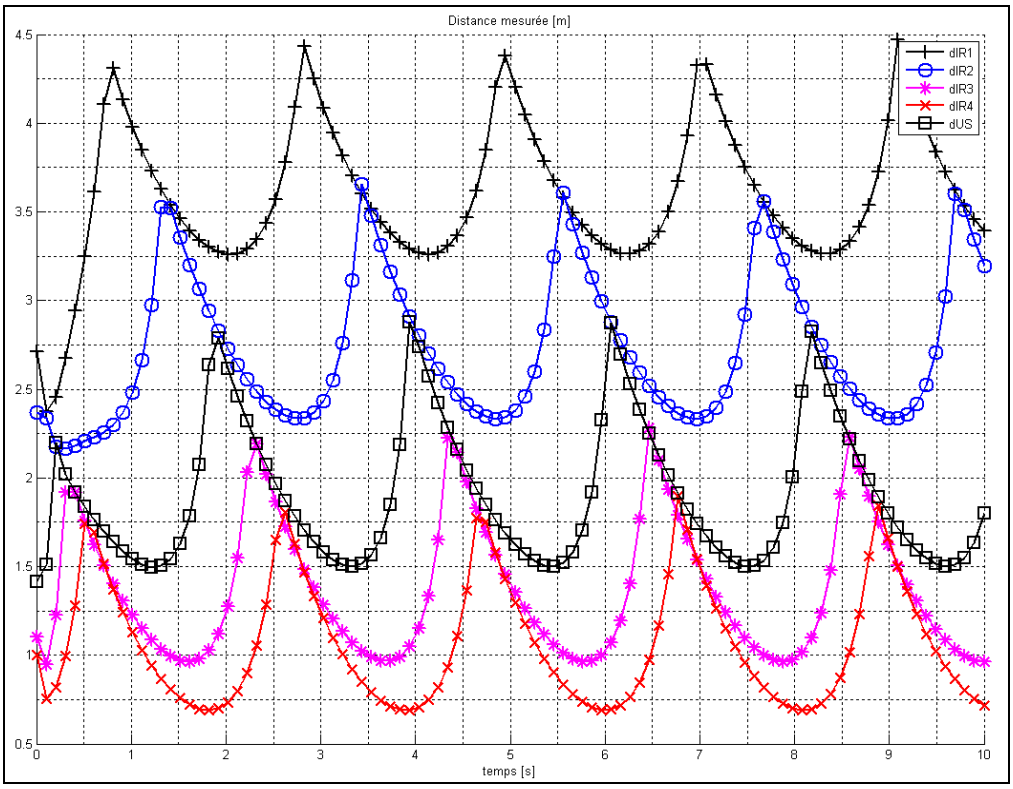

Fig. 3.14 : Représentation Graphique des Distances (Cercle)

Ensuite, on remarque les phénomènes les plus significatifs :

- Certaine périodicité des mesures. Effectivement, lorsque le robot prend le rayon du cercle de la trajectoire, les mesures décrivent toujours un même dessin. Cela se passe puisque, à ce moment-là, les distances mesurées dans un quadrant de la trajectoire (par exemple de $\boldsymbol{\theta r}$ = 90º à $\boldsymbol{\theta r}$ = 180º) sont pareilles à celles du reste des quadrants. Les pics sont le résultat de l'impact des rayons des capteurs dans les coins du carré, c'est-à-dire, les points les plus éloignés de chaque capteur.

- Chaque mesure de chaque capteur s'encadre entre un maximum et un minimum. A cause de la périodicité et l'angle de chaque capteur, les mesures sont périodiques et déplacées dans le temps selon le mouvement du robot.

- Le comportement des signaux près des pics est parfois irrégulier. Cela est dû au pas de calcul de la trajectoire, c'est-à dire, à sa discrétisation. Au voisinage des pics, le pas de calcul conduit donc à des courbes discontinues. Cependant, loin des pics, la distance calculée est suffisamment précise pour la trajectoire suivie. Ce phénomène ne peut être corrigé qu'en diminuant le pas de calcul mais il n'est pas grave, il faudra tout simplement en tenir compte.

En bref, on peut dire que, sauf le cas remarqué des pics, le modèle est satisfaisant pour ce type de trajectoire.

Dans ce paragraphe, on introduit les paramètres géométriques réels des capteurs vis-à-vis du point ***Ar*** et indiqués dans le tableau de la figure 3.4 (auparavant pris comme

idéaux). Ainsi, on va essayer la trajectoire de la figure 3.10 et on va comparer les résultats. Prenant ces considérations, voici le graphique des mesures :

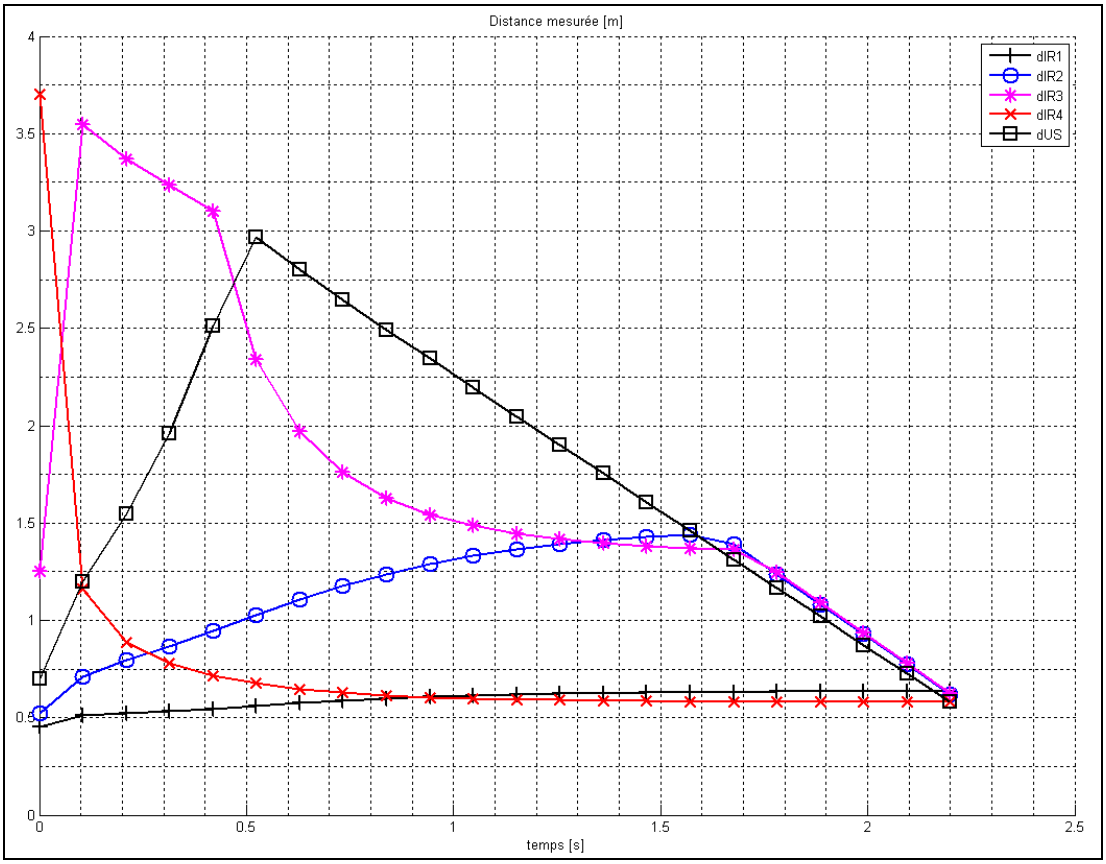

Fig. 3.15 : Représentation Graphique des Distances (Ligne Droite Modèle Semi-réel)

Il n'est pas difficile de souligner les différences suivantes :

- Les distances finales ont diminué. Maintenant les capteurs sont plus proches des murs donc les mesures sont plus petites.
- Les instants où les tronçons varient ou lorsque les rayons changent de mur d'impact sont différents. Cela est provoqué par la nouvelle position et par le changement d'inclinaison des capteurs.
- Les mesures des capteurs, théoriquement symétriques (***IR1-IR4*** et ***IR2-IR3***), lorsque l'angle $\boldsymbol{\theta r} \approx 90^{o}$ ne coïncident pas. Le robot est aligné avec la trajectoire mais la ceinture n'est plus symétrique.

Pourtant, on n'a pas encore vu l'effet de la prise en compte des limites de perception des capteurs. C'est ce qu'on peut apprécier dans la figure suivante :

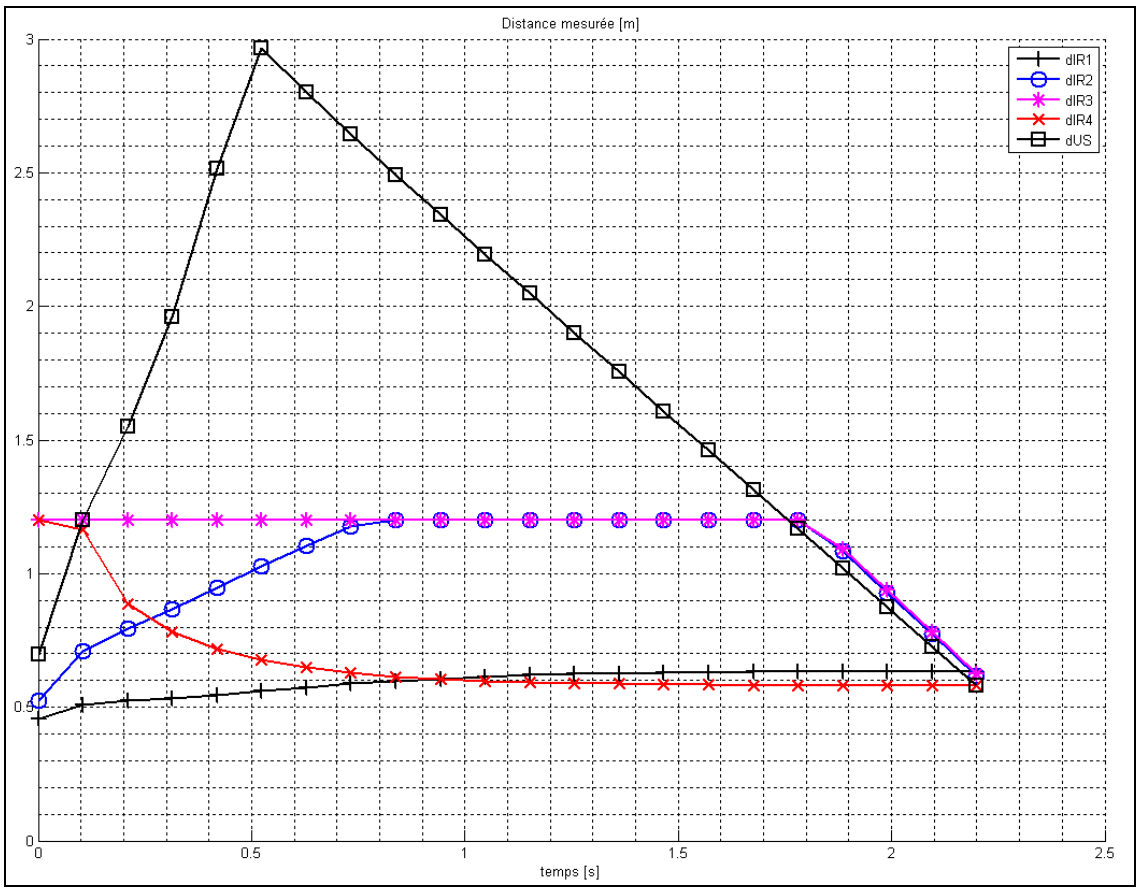

Fig. 3.16 : Représentation Graphique des Distances (Ligne Droite Modèle Réel)

Finalement, il faut remarquer que, dans le cas d'un couloir, les capteurs les plus utiles sont les capteurs centraux, qui sont : IR2, IR3 et US.

Les simulations différentes qui ont été précédemment présentées montrent que notre modélisation de l'ensemble des capteurs est tout à fait correcte. Il reste à utiliser ce code, en tant que travail futur, dans le logiciel de commande en temps réel du robot.

4. PROGRAMMATION DU ROBOT.

Dans ce paragraphe on résume tout ce qui concerne à la programmation du calculateur embarqué sur le robot. Les renseignements vont d'un point de vue général à la particularité des applications ou fonctionnalités développées.

Ainsi, on commence en expliquant le type d'architecture qu'on va suivre. Puis, on continue en présentant certains renseignements sur l'utilisation du logiciel du développement (*Visual Studio 2005*) et en montrant les fonctionnalités additionnelles des bibliothèques fournies par *National Instruments*. On termine par présenter une application pour détecter obstacles qui est capable d'arrêter le robot, de prendre les mesures des capteurs et de les comparer avec des estimations.

4.1. Architecture Document-Vue.

Parmi les différentes architectures possibles, que l'outil de programmation et développement *Visual Studio 2005* et le langage *Visual C++* offrent, on a choisi l'architecture *Document-Vue*.

Il existe beaucoup de justifications importantes de ce choix. Parmi celles-ci, on trouve l'héritage des projets antérieurs ; ils ont utilisé cette architecture et logiquement, le plus sensé est de la suivre. Une autre explication pourrait être la recommandation du fabriquant *National Instruments* de programmer sur cette organisation logique.

Pourtant, au-delà de ces motifs, on peut trouver la véritable raison d'être du choix, c'est un choix fonctionnel et pratique. L'alternative la plus proche serait l'architecture *Dialog-Based*, qui ne permet pas trop de complexité et qui manque de certaines utilités, de barres et de boîtes d'outils, etc. En plus, l'architecture choisie, organise et structure toute la programmation selon deux classes : le document et la vue. Pour plus de précision sur ce sujet, on peut consulter la référence [MIC00] de la bibliographie.

D'un côté, dans la classe document on peut trouver toutes les sous-classes et données qu'on va utiliser dans l'application. Il y aura aussi des fonctions membres et d'autres fonctions intégrées destinées à la manipulation des données, c'est-à-dire l'autorisation et l'accès à ces données.

D'un autre côté, la classe vue est tout à fait dédiée à l'interface avec l'utilisateur. Elle va gérer les boîtes de dialogue, les barres d'outils, les menus, les graphiques et tous les autres composants de l'application qui vont la rendre plus conviviale.

La classe vue peut contenir des boîtes de dialogues dans d'autres classes avec lesquelles on communique. De plus, on peut trouver plusieurs documents dans une seule application, c'est le *Multiple Document Interface* ou *MDI* ; c'est le cas de *Microsoft Word*, par exemple. Pourtant, cette option n'est pas intéressante pour ce projet. On utilise donc le *Single Document Interface* ou *SDI*.

Il ne faut pas oublier qu'à la fin les deux classes présentées travaillent ensembles. Cela veut dire qu'il est nécessaire de les faire communiquer entre elles. C'est pour cela que l'on fait l'intégration document-vue, ce qui permet modifier les données représentées à l'écran lorsqu'elles changent dans le document et vice versa.

Il y a trois procédés différents pour faire cette intégration document-vue : c'est le document qui demande l'actualisation des vues, c'est la vue qui demande la nouvelle valeur de la donnée au document ou bien on établit un protocole spécifique document-vue. Le procédé le plus recommandé est le premier bien que dans ce projet on utilisera souvent le deuxième, qui est le plus adapté aux besoins. On peut voir cette intégration dans le schème ci-dessous :

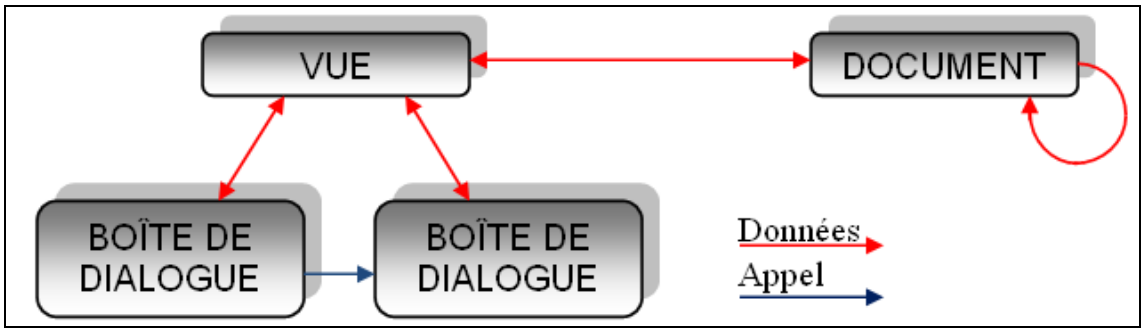

Fig. 4.1 : Intégration Document-Vue

4.2. Visual Studio 2005.

On a déjà dit que le logiciel, qui sert au développement des applications qui sont exécutées dans le calculateur du robot, est le *Visual Studio 2005*. Dans la suite, on donne quelques notions sur l'utilisation de cet outil ainsi que sur l'adaptation de l'architecture *Document-Vue*.

4.2.1. Usage basique.

Créer un projet

On expose les étapes les plus importantes pour créer un nouveau projet dans Visual Studio 2005. Tout d'abord, il faut démarrer le menu de création d'un projet à travers de *Fichier→Nouveau→Projet* ou en appuyant le raccourci *Ctrl+Maj+N*. En ce moment-là, on ouvrira une fenêtre appelée *Nouveau Projet*, telle qu'on peut observer dans la figure 4.2 :

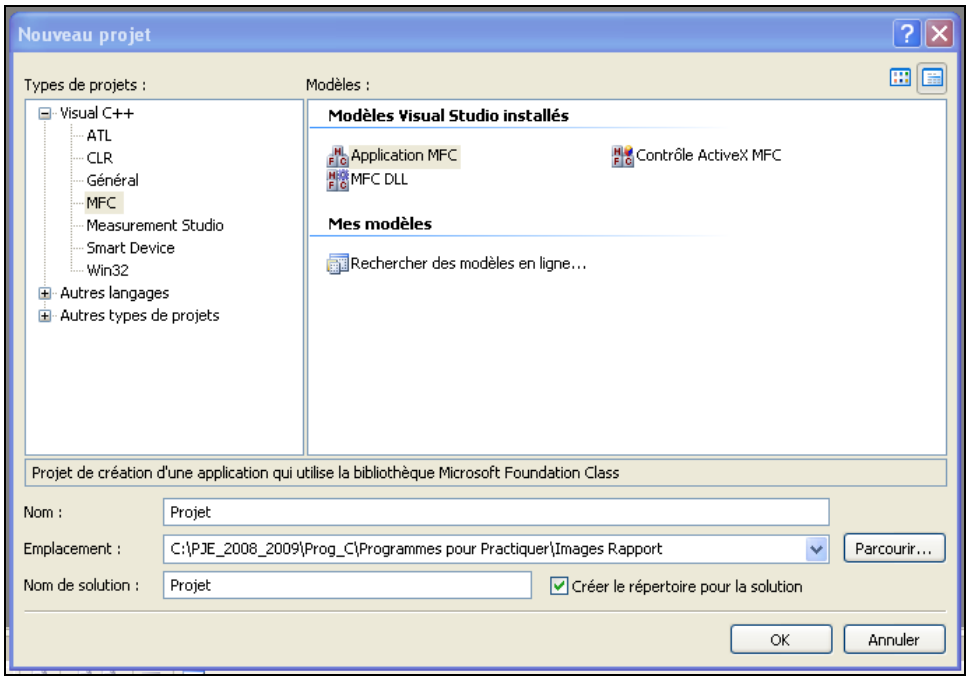

Fig. 4.2 : Fenêtre Nouveau Projet

Comme visible à la figure 4.2, pour créer une application exécutable dans le calculateur, il faut choisir *Measurement Studio* (*Types de projets*), *Application MFC* (*Modèles Visual Studio Installés*) et sélectionner un nom et un emplacement ; ensuite on appuie *OK*. Le motif de ce choix est que, grâce aux MFC (*Microsoft Fondation Class*), le procédé de programmation deviendra beaucoup plus facile.

Dès que le bouton *OK* est validé, il apparaît la fenêtre qui va guider le reste de la création du nouveau projet. A gauche, on voit les étapes qui doivent être suivies et à droite, on trouve les différents choix qui vont définir la configuration base du projet. Cette fenêtre est présentée à la suite :

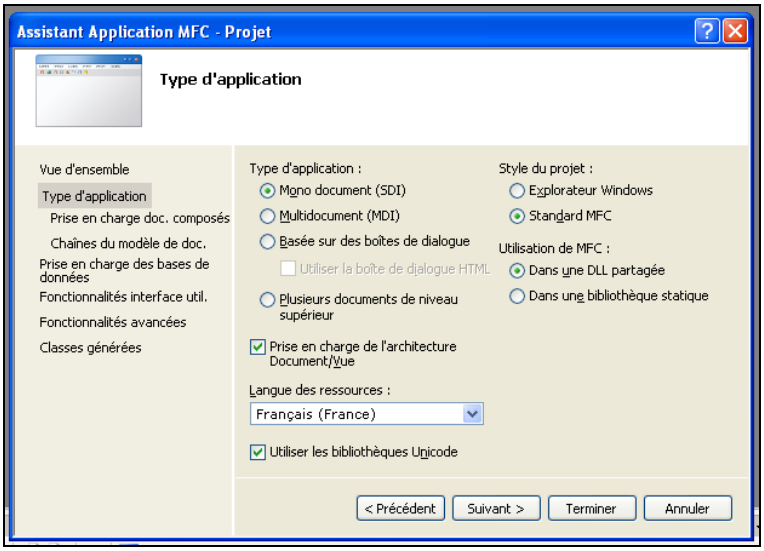

Fig. 4.3 : Fenêtre Assistant Application MFC

Dans la figure 4.3 on peut observer les choix qu'il faut retenir pour obtenir un projet valable pour l'application souhaitée. L'élément le plus important est le choix mono-document, c'est-à-dire, SDI dont on a déjà parlé. Les autres choix de la figure 4.3 étant validés, il faut cliquer *Suivant >*.

Ensuite, on va parcourir les sujets à gauche en pouvant choisir l'extension des fichiers enregistrés par l'application future mais en laissant tous les autres choix par défaut. Finalement, on arrive à la fenêtre suivante :

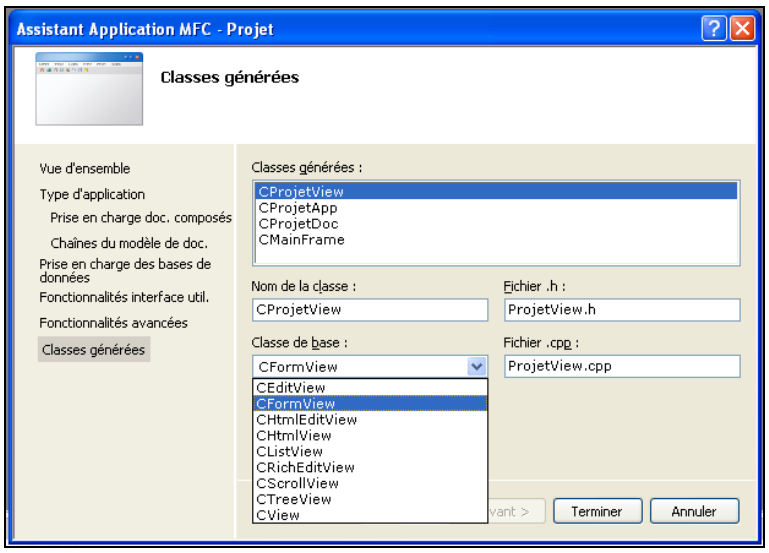

Fig. 4.4 : Fenêtre Assistant Application MFC (Classes Générées)

A ce moment, il faut choisir la classe de base *CFormView* pour la classe vue, comme le montre la figure 4.4. De cette façon, l'application résultant va avoir une boîte de dialogue dans une fenêtre cadre principale, où l'on pourra placer tous les contrôles nécessaires pour la vue. En parcourant les classes qui vont être générées, on pourra remarquer le nom des fichiers générés et associés aux classes ; il vaut mieux conserver les noms proposés.

La figure 4.5 montre le résultat obtenu après avoir suivi les étapes précédentes. De même, on va commenter les utilisations de chaque partie étant dans la fenêtre du logiciel *Visual Studio 2005* :

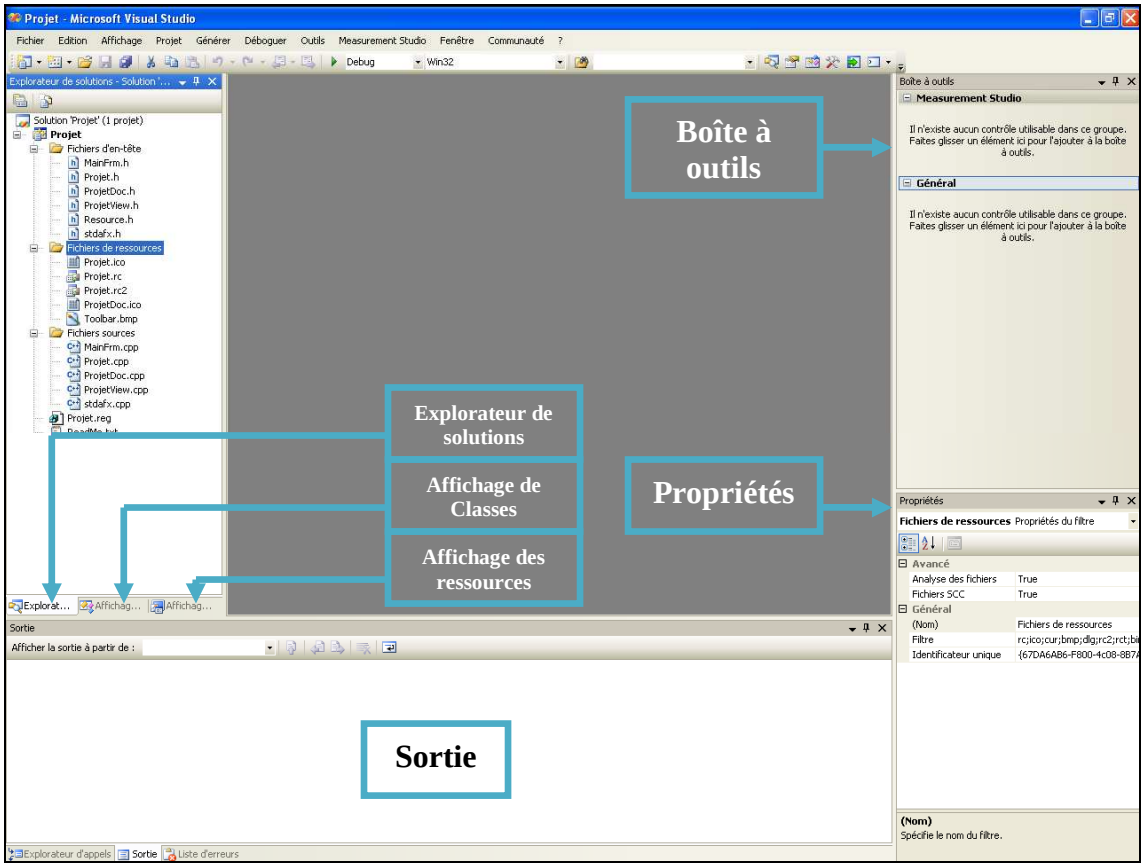

Fig. 4.5 : Environnement de Visual Studio 2005

Voici la fenêtre de travail de *Visual Studio 2005*, créé par le projet avec l'architecture Document vue. On peut facilement remarquer la barre de menu habituelle dans les applications *Microsoft*. On y apprécie les menus fichier, édition, fenêtre et même le *Measurement Studio*, puisque le logiciel pour la intégration entre les cartes NI et le Visual C++ est déjà installé. Un peu plus bas on trouve la barre d'outils avec des éléments d'édition.

Juste à gauche, placée à mode de colonne, on voit l'explorateur de solutions joint avec l'affichage de classes et celui de ressources. Les plus utilisées sont le premier, avec lequel on ouvrira les différents fichiers inclus dans le projet, et le dernier, avec lequel on verra et ouvrira les ressources qui appartiennent au projet.

Tout en bas, on trouve l'affichage des sorties. C'est ici le lieu où on va voir et consulter tous les renseignements apportés par rapport aux procédés réalisés sur le projet. L'utilisation la plus importante est celle de l'analyse du résultat de la génération puisque toutes les erreurs y apparaîtront et, en faisant un double click ou en appuyant F1 sur ces erreurs, on dirigera à la ligne où ils se trouvent ou l'on donnera toute l'information nécessaire pour corriger cette ligne.

Enfin, à droite on voit une autre colonne divisée en deux. En haut, on trouve la boîte à outils, principalement utilisée lors du développement des ressources car c'est là où les composants graphiques se trouvent. En bas, c'est la fenêtre des propriétés, l'une des plus importants aussi lors de développement de la vue. Chaque fois qu'on sélectionne une ressource, soit un bouton, soit un menu, soit un graphique, toutes ses propriétés y apparaissent avec leurs identificateurs (*IDx_NOM*). De même, si l'on peut joindre un événement de contrôle, on le fera en appuyant sur un icône avec une foudre.

Concevoir une interface d'usager

On indique ici comment manipuler l'interface de l'application et comment lui donner la fonctionnalité souhaitée. Tout cela est fait grâce à l'addition des ressources de diverses origines. Par exemple : un menu et ses options accrochées dans la barre de menu, un bouton sur la barre d'outils, un bouton animé sur le dialogue ou même une barre de contrôle également animé sur ce dialogue.

On peut voir, à la figure 4.6, tous les genres de ressources qu'on peut trouver d'habitude dans une architecture document-vue :

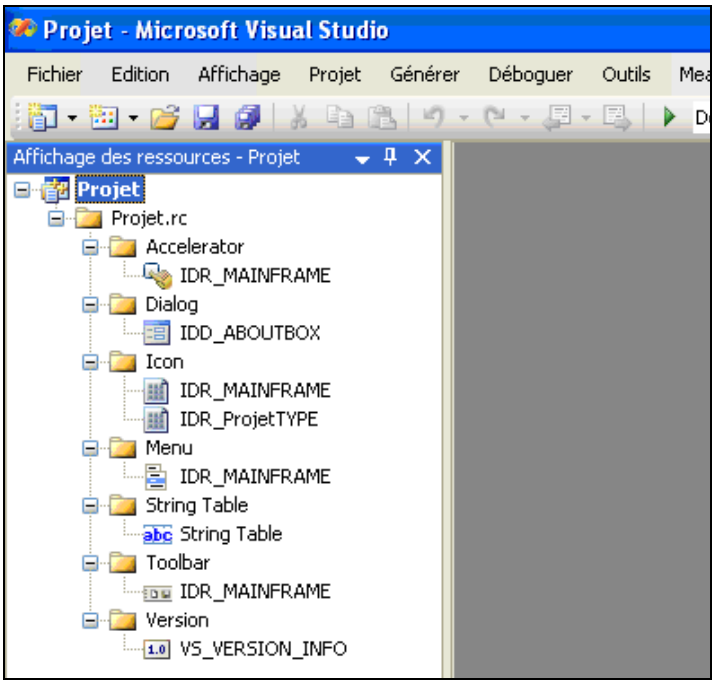

Fig. 4.6 : Affichage des Ressources

Cette représentation graphique permet de voir les composants du fichier *NOM.rc*, inclus dans le projet. Ainsi, en dépliant les objets, on trouve les accélérateurs, les dialogues, les icônes, le menu, la table des chaînes, les barres d'outils et la fenêtre de la version.

La liste des accélérateurs comprend les identificateurs des ressources associés à une combinaison des touches du clavier ; cela permet de raccourcir le temps d'accès à une option ou fonctionnalité. Les icônes sont justement les petits dessins qui identifient l'application. La table des chaînes montre les identificateurs des ressources, leurs valeurs numériques et leurs légendes (l'étiquète dans le menu joint avec l'explication qui apparaît lorsque la souris devient immobile). Pourtant, en raison de leurs intérêts, on va s'arrêter un peu plus sur les autres éléments.

D'abord, on commence en expliquant la barre de menu. Elle possède toujours les fonctionnalités typiques des applications comme le menu fichier, édition, etc. Pourtant, grâce à l'édition de cette ressource, on pourra modifier ces applications et ajouter de nouvelles. Pour plus de détail, on montre cette édition ci-dessous :

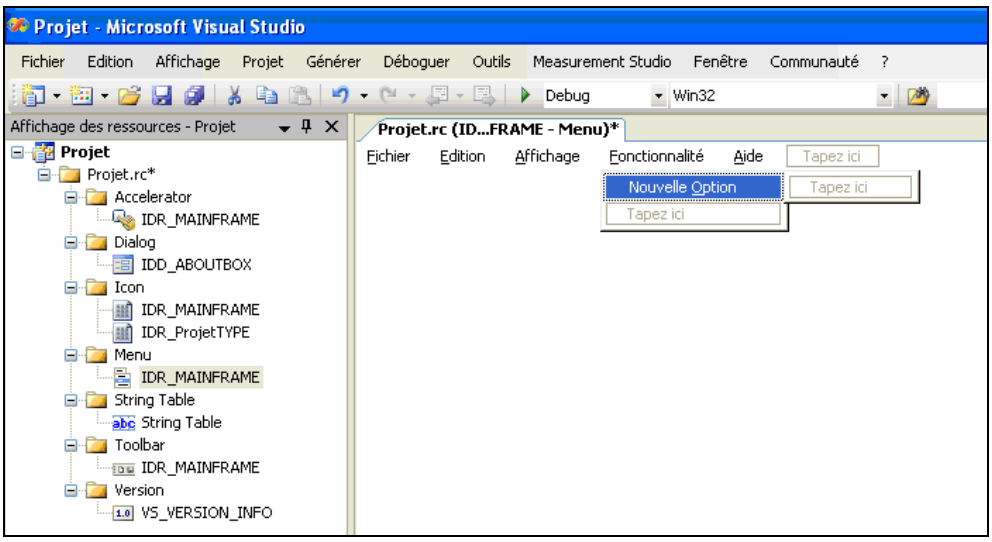

Fig. 4.7 : Edition de la Barre de Menu

On peut remarquer, dans la figure 4.7, qu'un nouveau menu a été a ajouté ainsi que dans celui-ci une nouvelle option. Tout cela peut être fait tout simplement en tapant les noms dans les boîtes où on indique *Tapez ici*. Pour rappeler les raccourcis on peut inclure & avant la lettre qui l'on veut souligner.

A ce moment, on peut voir dans la boîte de propriétés tous les paramètres en rapport avec la nouvelle option qui est marquée. Le paramètre le plus important est l'identificateur, qui peut être modifié et à travers lequel on va donner la fonctionnalité à l'option.

Un cas semblable à la barre de menu, c'est la barre d'outils. Dans ce cas, la nouveauté est qu'il y a des boutons graphiques dessinés par l'utilisateur au lieu d'un menu de mots. Voici cette édition :

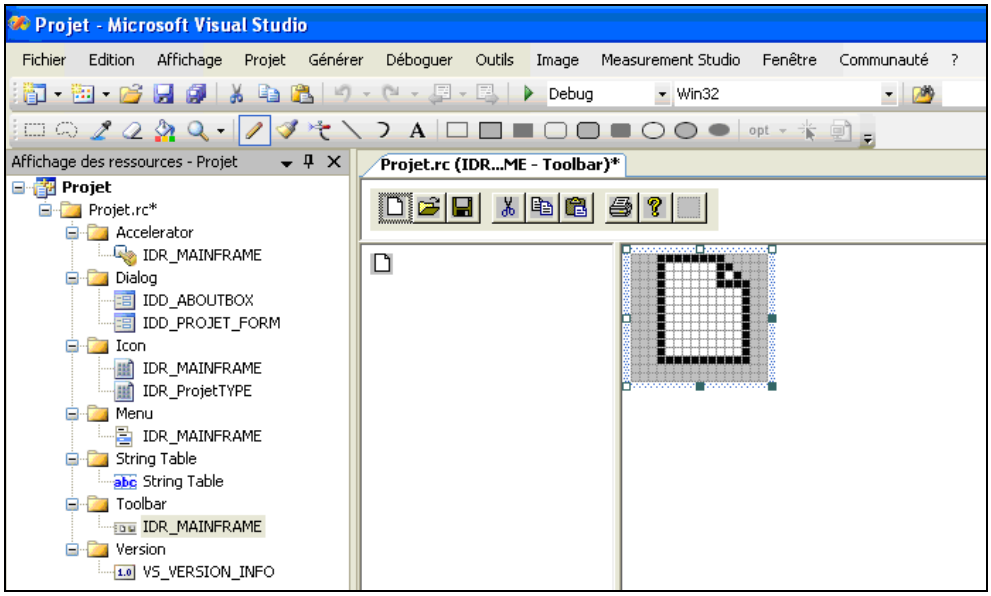

Fig. 4.8 : Edition de la Barre d'Outils

On peut remarquer qu'il existe une boîte à outils spéciale pour dessiner chacun des icônes. Lors de l'édition d'un icône, dans la boîte de propriétés on montre, parmi d'autres, l'identificateur du bouton ; rien n'empêche d'avoir le même identificateur qu'une option de la barre de menu et logiquement, la même fonctionnalité.

Pour finir, on va présenter la conception d'une boîte de dialogue principale, qui sera encadré dans la fenêtre cadre de l'application. C'est ici où l'on trouvera les composants les plus importants de l'application. Pour sa construction, dans la boîte à outils on montre les composants spécifiques. Parmi ces éléments, il y aura quelques uns ajoutés pour l'application *Measurement Studio* de NI, puisque le logiciel d'intégration a été déjà installé.

Voici un exemple d'une boîte de dialogue à laquelle ont été ajoutés plusieurs composants graphiques et numériques pour montrer quelques possibilités de cette édition. On pourra remarquer aussi d'autres composants dans la boîte à outils :

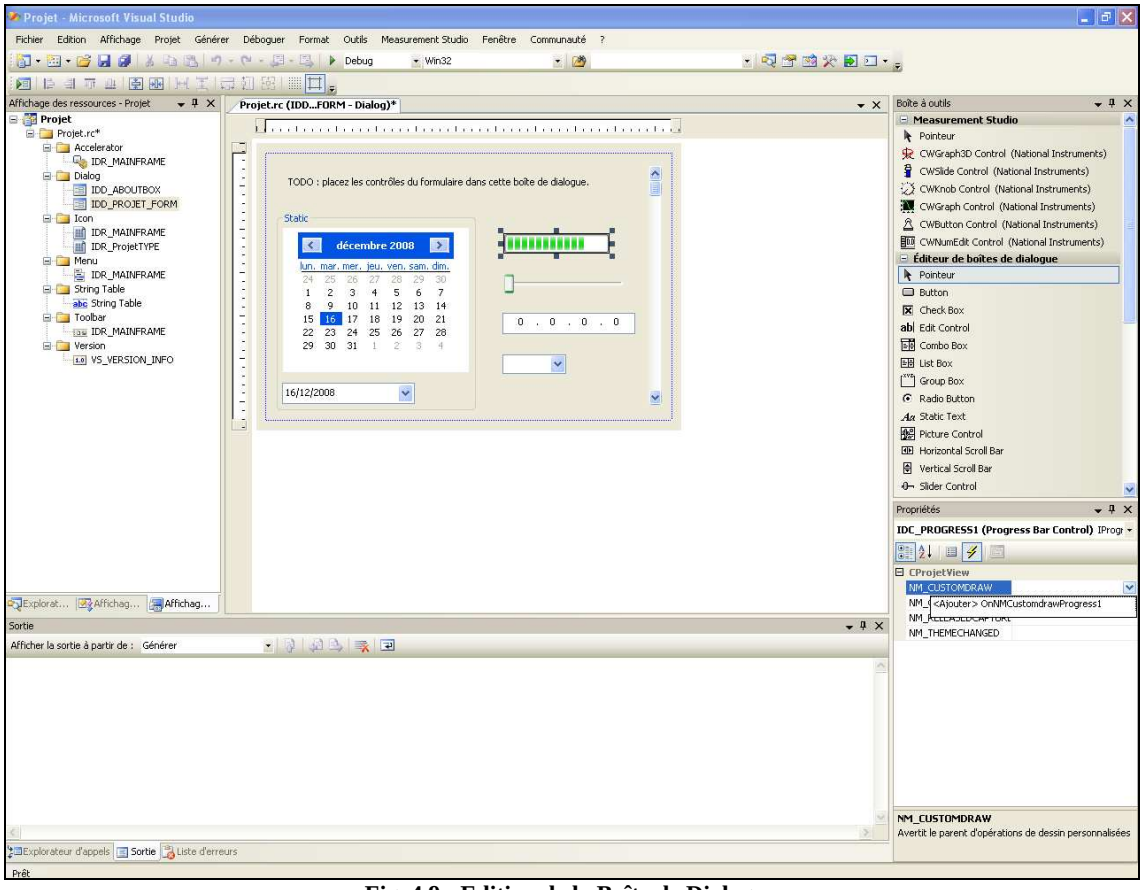

Fig. 4.9 : Edition de la Boîte de Dialogue

De même, il faut dire que dans la boîte de propriétés il y aura l'identificateur du composant de la vue sélectionnée, mais aussi on pourra ajouter un gestionnaire d'événements pour le gérer. Pour cela, il suffit d'appuyer sur le foudre, choisir l'événement souhaité et appuyer à droite sur *<Ajouter> OnXXXXX*.

Une fois on a fait ce procédé, *Visual Studio* va se diriger directement vers le segment de la classe vue où on peut taper le code nécessaire pour faire fonctionner le composant (fichier *NOMView.cpp*). Le code prédéfini peut être vu ci-dessous :

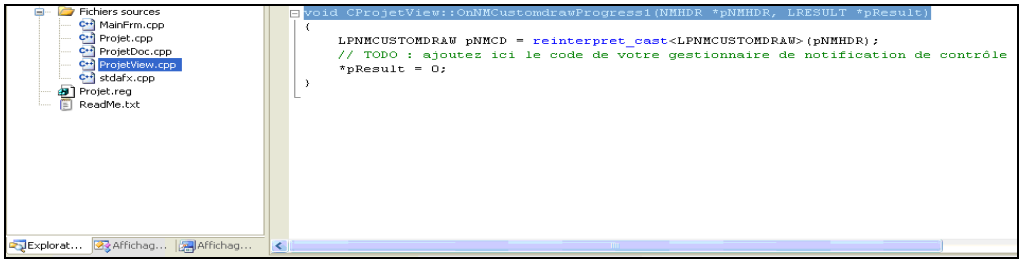

Fig. 4.10 : Ajoute d'un Gestionnaire

On termine ainsi le bref résumé sur l'utilisation du *Visual Studio 2005*. On peut trouver beaucoup de bibliographie par rapport à ce sujet, même sur internet. Pour obtenir des informations complémentaires, on conseille ici les références [MIC00] et [NUN07] de la bibliographie.

4.2.2. Architecture Document-Vue en Visual Studio.

On a déjà vu que l'architecture dans laquelle on va développer les applications de ce projet est celle de *Document-Vue* et on a indiqué quelques unes de ses avantages et les motifs de ce choix. Egalement, on a expliqué ce qu'il faut faire pour créer et modifier une application au-dessous cette architecture dans le logiciel *Visual Studio 2005*. On va étudier maintenant les fichiers que ce logiciel génère et ses liaisons.

On rappelle que, dans l'architecture choisie, il y a deux classes fondamentales ; ce sont le document et la vue. Visual Studio gère ce sujet en générant plusieurs fichiers d'extensions différentes, parmi lesquelles on trouve : *.cpp* (code des fonctions), *.h* (déclarations), *.rc* (ressources), etc. Mais les fichiers qui vraiment définissent cette organisation sont *NOMDoc.cpp/.h* et *NOMView.cpp/.h*. Chaque couple représente une classe composant le document ou la vue. En plus, il y d'autres fichiers comme *MainFrm.cpp/.h* qui gèrent la fenêtre cadre de l'application *NOM.cpp/.h*.

A part de cela, l'architecture est suivie selon les règles générales présentées auparavant. C'est-à-dire, le fichier *NOMDoc.cpp* contiendra le code des fonctions dédiées au traitement des données tandis que le fichier *NOMDoc.h* contiendra les déclarations de ces fonctions ainsi que des variables ou données à utiliser (inclues les classes de données). De même, le fichier *NOMView.cpp* aura les fonctions membres et gestionnaires nécessaires pour gérer l'interface de l'utilisateur et le fichier *NOMView.h* aura toutes les déclarations des variables et classes utilisées à ce but ; et tout cela par rapport aux ressources développés.

Pour que toute cette distribution des déclarations des fonctions et variables des fichiers *.h* avec leurs utilisations et déroulement des fichiers *.cpp* soit cohérent, il y a un rapport administré à travers du mot réservé du langage C *#include <NOM.EXT>*. On montre, à la suite, les liaisons qui permettent la bonne démarche de cette technique :

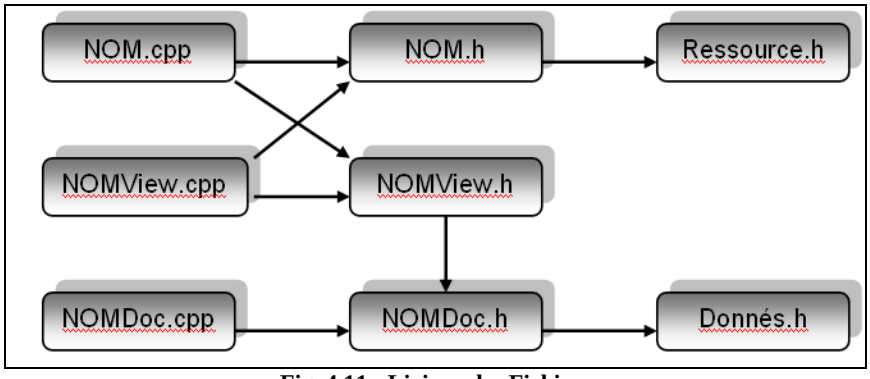

Fig. 4.11 : Liaison des Fichiers

En bref, on peut remarquer que chaque fichier *.cpp* est lié à un fichier *.h*, que les fichiers généraux de l'application sont reliés tant aux ressources qu'à la vue et finalement, que le document contient indirectement toutes les déclarations. Pour finir, on doit préciser que le fichier *Données.h* et le fichier *Données.cpp* sont associés et qu'ils définiront ensemble une classe de données générique.

4.3. National Instruments.

Jusqu'au présent on a parlé des cartes NI et de leurs fonctions dans le projet aussi qu'on a commenté le besoin d'installer le logiciel *Measurement Studio* de *National Instruments* pour intégrer ces cartes et leur pilotage dans Visual Studio et l'application à programmer. On verra maintenant comment utiliser quelques des fonctionnalités auxquelles on a accès grâce au logiciel de NI.

On commence par présenter un petit exemple, détaillé en l'annexe A6, où l'on peut remarquer l'édition des ressources d'un dialogue avec composants du *Measurement Studio* :

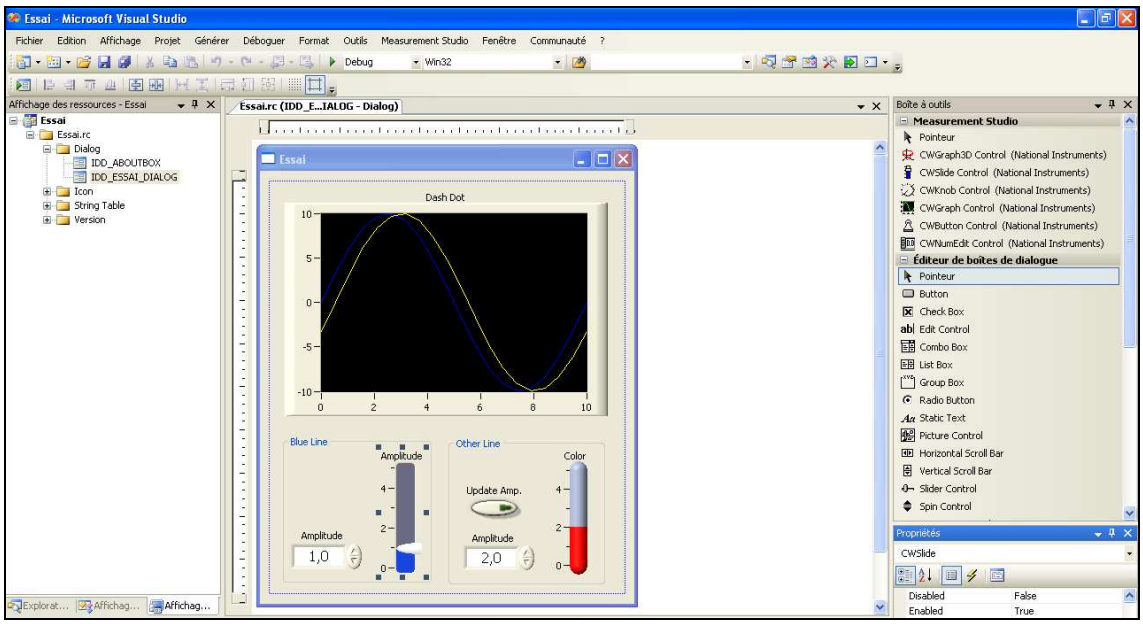

Fig. 4.12 : Edition des Ressources de Measurement Studio

Dans la boîte à outils de la figure 4.12, on peut voir les ressources apportés par *Measurement Studio* en haut, ajoutées à celles apportées par Visual Studio en bas. Dans le dialogue on trouve un graphique avec deux signaux sinusoïdaux qui varient en amplitude et en couleur à l'aide de contrôles graphiques et numériques.

En plus des ressources de Visual Studio, il faut signaler un composant pour connaître ou changer son identificateur. Donc, après on appuie sur le foudre on verra les événements auxquels on peut ajouter un gestionnaire. Mais il y a une nouvelle fonctionnalité, si l'on appuie sur le bouton de la boîte de propriétés le plus à droite, on verra une boîte de dialogue qui permettra changer profondément l'apparence et caractéristiques des composants. Voici quelques exemples :

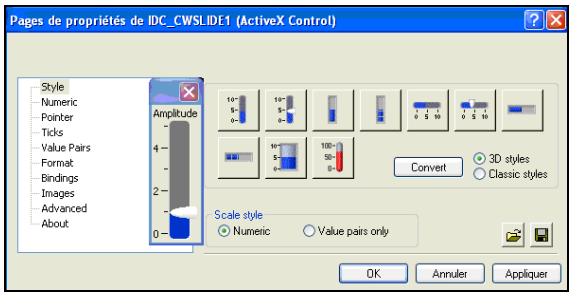

Fig. 4.13 : Propriétés d'un Slide Control

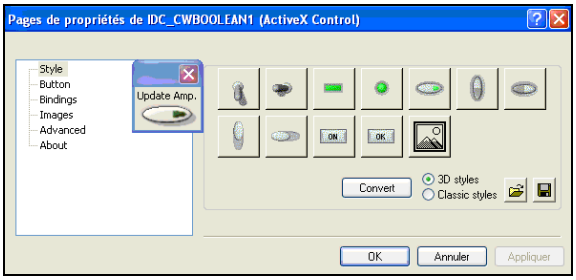

Fig. 4.14 : Propriétés d'un Button Control

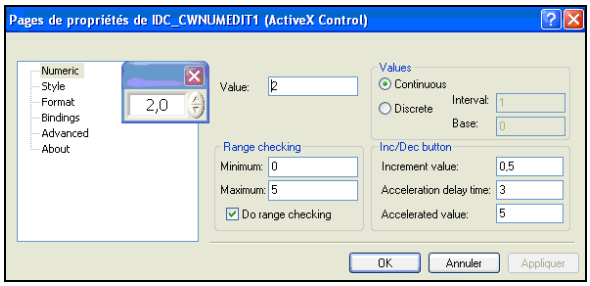

Fig. 4.15 : Propriétés d'un NumEdit Control

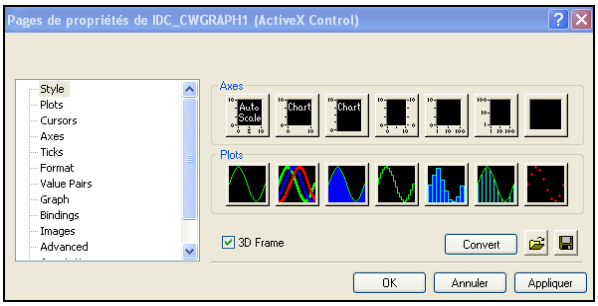

Fig. 4.16 : Propriétés d'un Graph Control

Toutes ces fonctionnalités sont gérées à travers de la classe CNiControl, dérivée de CWnd. Cette classe contient, de même, les dérivées CNiButton, CNiGraph, CNiGraph3D, CNiKnob, CNiNumEdit et CNiSlide. En modifiant les paramètres sur ces boîtes et directement sur l'édition des ressources on aura une grande flexibilité pour la conception de l'interface.

4.4. Applications sur le robot.

Auparavant, on a déjà montré une application qui marche sur le calculateur du robot dont s'appelle *Enregistrer v2.4*. Ce logiciel fut uniquement conçu pour prendre des mesures des capteurs de distance et réaliser certaines études afin de rendre opératif le système acquisition de données à travers de ces capteurs. On se limite ici à rappeler que son code est expliqué dans l'annexe A7.

Le but de ce paragraphe est plutôt de présenter une autre application qui a été développée pour obtenir les mesures des distances des capteurs et les comparer avec une estimation calculée grâce au modèle du paragraphe 3. Tout cela tandis que le robot bouge et à travers du système d'acquisition récemment mis en œuvre.

Il faut dire que ce logiciel, appelé *Robot_HAMMI v2.2*, a été conçu à partir de la plus performante des versions du logiciel de l'année dernière, qui faisait bouger le robot dans une trajectoire droite et puis décrire un arc de cercle (toujours sans le fauteuil accroché), soit *Robot_HAMMI v1.0*.

Voici l'interface d'usager de ce logiciel avec ses parties expliquées :

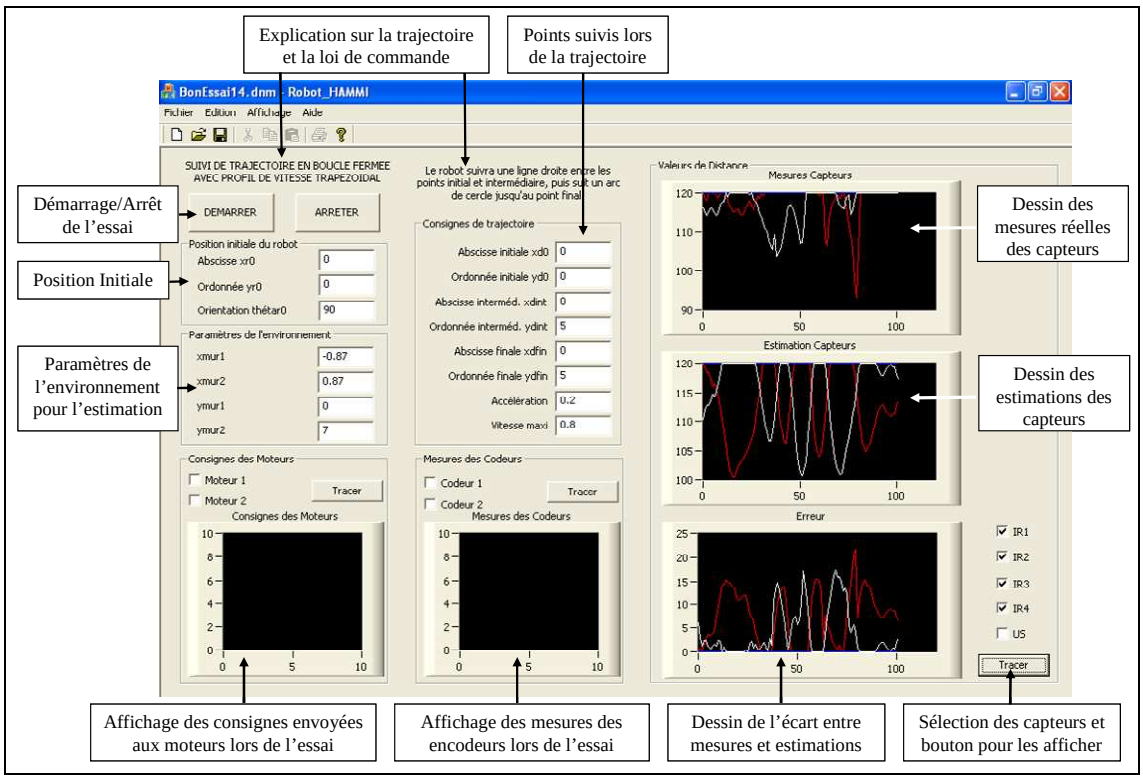

Fig. 4.17 : Interface d'usager de Robot_HAMMI v2.2

La loi de commande, la communication avec les moteurs et les encodeurs, ainsi que les fichiers de base (document, vue, déclarations, etc.), étaient déjà inclus dans le projet de *Visual Studio 2005* de l'année dernière.

Cette année on a refait l'interface usager, qui est présentée dans la figure 4.17 et il a fallu inclure des nouveaux fichiers et fonctions dans le projet. C'est ainsi qu'on a appliqué le traitement de données et l'estimation auparavant développés.

Les fichiers modifiés et ajoutés, ainsi que la cause de la modification ou la fonction qui réalisent, sont montrés dans le tableau de la figure 4.18 :

Fichier	Cause de la modification / Fonction du fichier ajouté
Fichiers Modifiés	
Prototype_fonctions.h	Ajouter les déclarations des nouvelles fonctions
Variables_globales.h	Ajouter certaines nouvelles variables
Robot_HAMMIDoc.cpp	Initialiser des nouvelles variables / Fonction pour enregistrer
Robot_HAMMIView.h	Ajouter des graphiques et des variables de l'environnement
Robot_HAMMIView.cpp	Initialiser des variables / Changer certains gestionnaires
Robot_HAMMI.rc	Refaire l'interface d'usager en ajoutant des éléments
Fichiers Ajoutés	
Traitement_donnees_capteurs.cpp	Appliquer le filtre numérique et étalonner les mesures
Calcul_estimation_capteurs.cpp	Estimer les mesures des capteurs à partir de la position du robot
Calcul_erreur_capteurs.cpp	Calculer l'écart absolu entre mesures réelles et estimations

Fig. 4.18 : Tableau des modifications pour obtenir Robot_HAMMI v2.2

Les morceaux des modifications les plus significatives des fichiers existants et le code le plus important des fichiers ajoutés seront présentés un peu plus bas, dans l'annexe A8.

Pour vérifier ce logiciel on a réalisé un essai avec une trajectoire en ligne droite dans un couloir. Ainsi, on a testé si le logiciel comporte des nouvelles modifications sans nuire les fonctions existantes, en obtenant des mesures des capteurs, des estimations et leurs écarts et graphiques.

La largeur du couloir est de 1,74 mètres et le robot s'est situé en plein milieu juste à 7 mètres du bout du couloir. De telle façon, les distances du centre de gravité du robot aux murs du couloir sont fixées selon la figure 4.17. Ensuite, le robot avancera 5 mètres tout au long du couloir à une vitesse de 0,8 mètres par seconde. Si bien ces paramètres peuvent être modifiés dans l'interface d'usager, ils sont automatiquement initialisés pour cet essai selon la figure 4.17.

D'autre part, les données de l'essai peuvent être enregistrées depuis le logiciel *Robot_HAMMI v2.2*, d'abord dans une matrice et après dans un fichier avec une extension *.dnm*. Ensuite, grâce à un autre logiciel les données sont traduites dans un fichier *.txt* qui comporte le bon format pour passer la matrice directement au logiciel mathématique *Matlab* en copiant-collant. Cette matrice contient les pas de calcul, plusieurs paramètres du robot (parmi lesquels les paramètres de l'environnement), les mesures et estimations des capteurs et les points que suit le centre de gravité du robot lors de sa trajectoire. Voici l'interface d'usager du logiciel appelé *Traducteur v1.0* :

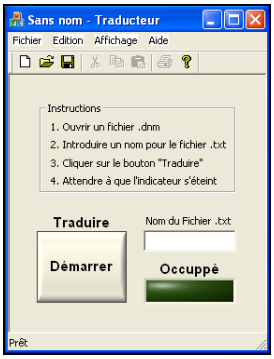

Fig. 4.19 : Interface d'usager de Traducteur v1.0

C'est ainsi que, à l'aide du fichier *comparer.m* de *Matlab* (présenté dans l'annexe A9), on analyse les résultats de l'essai. Mais d'abord, il faut dire que les premiers dessins de l'essai étudié sont déjà représentés dans l'interface d'usager de la figure 4.17.

La première chose à vérifier est si l'estimation des mesures des capteurs à été bien traduite du langage *Matlab* au *Visual C++*. De telle façon, dans la figure 4.20 on peut voir les distances estimées avec le code du logiciel *Robot_HAMMI v2.2*, les distances estimées grâce au code modele.m (basé sur le code modele_capteurs.m) et aux points que suit le centre de gravité du robot et finalement, l'écart entre les deux méthodes :

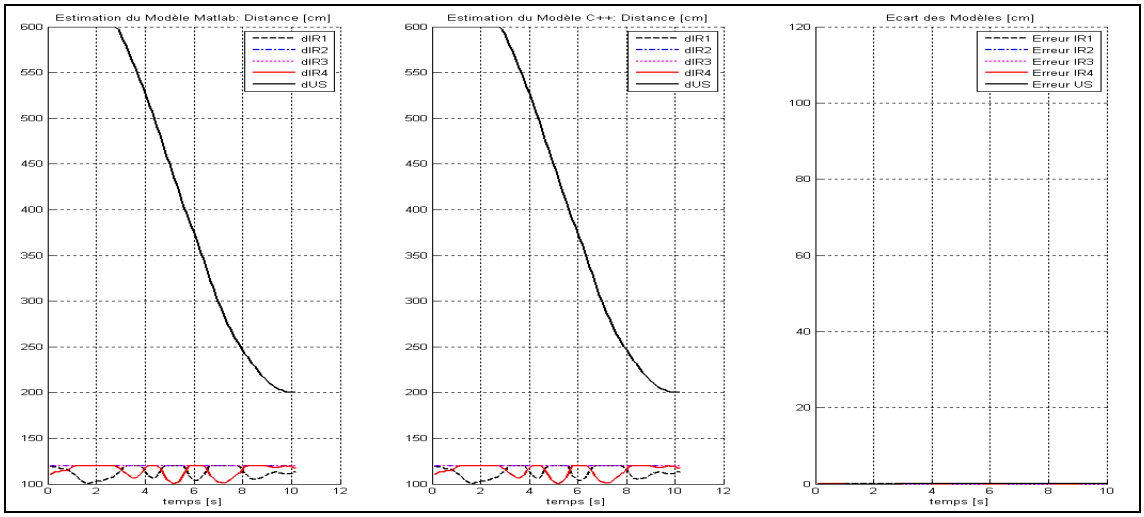

Fig. 4.20 : Figure 1 du code comparer.m (Comparaison des Modèles)

La conclusion n'est pas difficile : la traduction a été bien faite puisque le résultat de l'exécution du code de l'estimation sur *Matlab* et celui obtenu dans le calculateur du robot sont identiques. On peut voir que l'écart des modèles sur *Matlab* et sur *C++* est nul.

Il faut aussi tester si les résultats sont ceux attendus. Si le robot s'approche au mur *Xmur1* la distance estimée pour le capteur IR1 diminue tandis que celle du capteur IR4 augmente et s'il s'approche au mur *Xmur2*, il arrive à l'envers. Les capteurs IR2 et IR3 ne mesurent pas puisque le rayon impacte au-delà des limites de perception. La distance estimée pour le capteur à ultrasons diminue toujours à mesure que le robot s'approche au mur *Ymur2*. Voyons la trajectoire que le robot réalise dans la figure 4.21 en vérifiant que les résultats sont logiques :

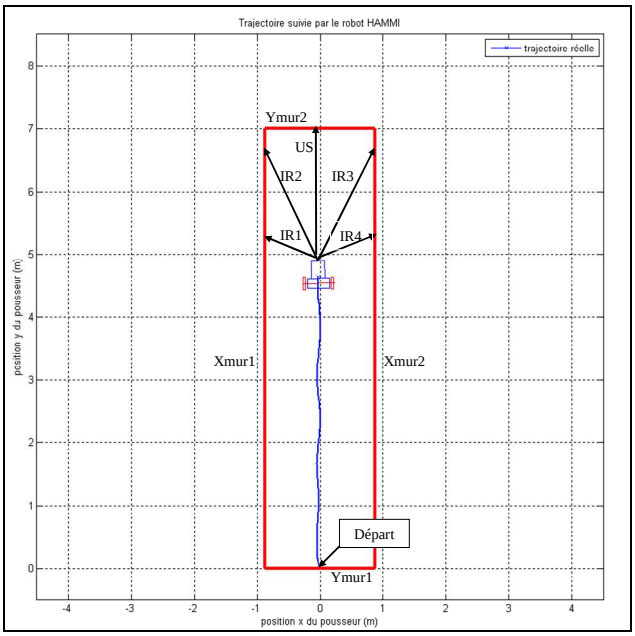

Fig. 4.21 : Figure 4 du code comparer.m (Trajectoire Suivie)

Ensuite, on va comparer les distances de l'estimation sur le calculateur avec les mesures prises par les capteurs et traitées numériquement dans le logiciel. Pour une meilleure compréhension, on présente d'abord les résultats concernant les capteurs infrarouges :

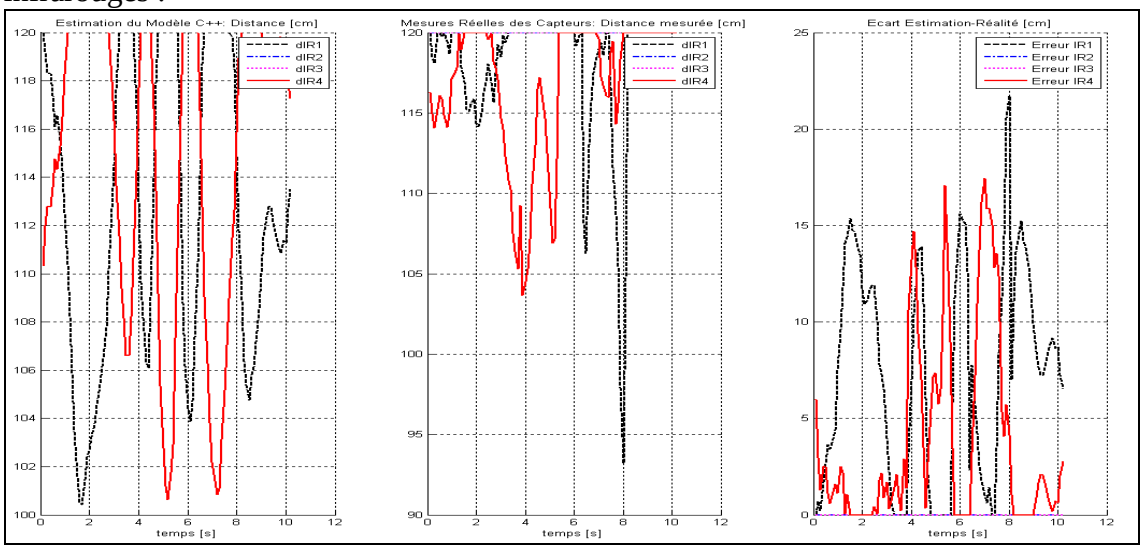

Fig. 4.22 : Figure 2 du code comparer.m (Mesures et Estimations des Capteurs IR)

Commentons certains points. D'abord, à cause du bruit structurel (celui dû à la flexibilité de la structure en mouvement) et les réflexions des rayons, les mesures réelles ne sont pas aussi propres que les estimations et parfois on a des pics trop exagérés. Ensuite, notons un certain retard des mesures réelles qui est provoqué par la chaîne du système d'acquisition et plus précisément, par les filtres mis en place. Enfin, il faut dire que même l'estimation peut ne pas être tout à fait précise puisque parfois les points de la trajectoire du centre de gravité du robot ne sont pas exacts non plus.

A part de ces effets, on devine que les mesures essaient de suivre les estimations, ce qui est très satisfaisant. En gros, l'erreur maximale commise entre les deux données n'atteint qu'à 15 centimètres, en étant normalement au-dessous. Notons aussi que, lorsque le robot suit la trajectoire en ligne droite dans le couloir, les capteurs centraux ne mesurent pas à cause de leur limite de perception. On devrait réviser donc l'emplacement puisque on a vérifié que, lorsque le fauteuil est accroché au robot, tous les capteurs deviennent inutilisables.

On montre alors les mêmes dessins, vis-à-vis du capteur à ultrasons, dans la figure 4.23 :

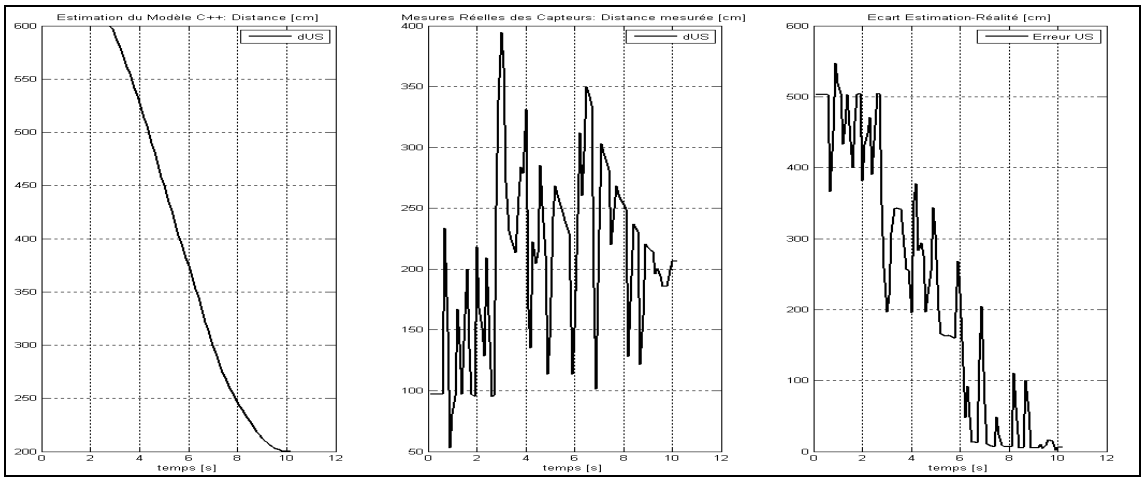

Fig. 4.23 : Figure 3 du code comparer.m (Mesures et Estimations du Capteur US)

Hélas, on n'a pas eu la même chance avec le capteur à ultrasons. On voit ici que le bruit dans ce cas est énorme. Les causes proviennent du bruit structurel et des réflexions des rayons (effet très observé dans ce type de capteur). Mais, il faut surtout dire que le cône de perception est tellement large que, dans le couloir, le capteur mesure avant les murs latéraux que le bout du couloir. Effectivement, lors que le robot s'approche au bout du couloir vers la 6^{e} seconde, il commence à mesurer le mur appelé *Ymur2* et, sauf le bruit toujours présent, l'écart tend vers zéro.

Ayant vérifié la fonctionnalité du système d'acquisition de données et des estimations, il ne reste à décrire que le système d'arrêt automatique (SAA). Celui-ci est capable d'arrêter doucement le robot au cas où un capteur mesure une distance de moins de 50 centimètres. En même temps, l'interface d'usager signale un message d'avertissement comme celui qui est montré dans la figure 4.24 :

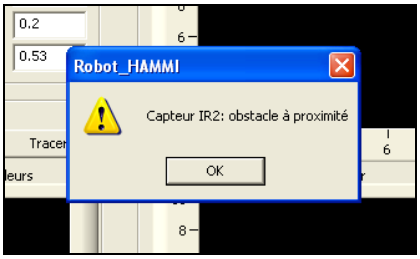

Fig. 4.24 : Message d'Avertissement du SAA

Finalement, on peut dire que les changements qui ont été réalisés sur le logiciel existant sont complètement satisfaisants : les estimations sont cohérentes par rapport à la trajectoire qu'apporte la loi de commande, le logiciel réalise un traitement des mesures des capteurs obtenant des données exploitables et un système d'arrêt automatique a été mis en place. En bref, ces résultats son encourageants et mettent un point final à tous les travaux auparavant réalisés et à ceux réalisés lors de ce rapport.

5. CONCLUSION ET TRAVAUX FUTURS.

Le but de ce projet a été globalement de développer et de mettre en place les systèmes nécessaires sur le robot HAMMI pour que l'implantation d'un filtre de Kalman, qui aide à concevoir les trajectoires, devienne évidente et immédiate.

Tout au long des derniers mois, j'ai travaillé sur tout le système d'acquisition de données des capteurs en étudiant les façons d'améliorer les mesures et de les inclure dans le dernier logiciel de commande du robot. C'est ainsi qu'on aboutit à la réalisation des ces tâches : conception et construction d'une nouvelle carte avec des filtres qui enlèvent le bruit des capteurs, conception d'un logiciel complet pour l'essai des capteurs et des essais nécessaires, conception d'un modèle géométrique d'estimation des mesures des capteurs et mise en place de tout ce développement sur un logiciel existant pour l'exploitation des données.

A la fin de ce projet, on a obtenu des mesures étalonnées et exploitables et des estimations de ces mesures à partir du modèle géométrique pour l'ensemble robot plus environnement. Toujours dans un logiciel de commande sur le calculateur du robot, on a soustrait ces mesures de sorte qu'on a obtenu l'erreur estimée, qui est la base de l'implantation d'un filtre de Kalman. Il ne reste que l'application de cette erreur à la commande du robot et donc, on peut affirmer que le projet a été une grande réussite.

Pourtant, il y a pas mal de travaux futurs qui sont prévus. Bien sûr, le premier d'entre eux est celui de mettre en place le filtre de Kalman à partir des données qu'on a déjà acquises. Mais, il faut remarquer aussi qu'analyser la possibilité de construire un autre type de modèle d'estimation pour les capteurs, inclure le nouveau code dans une autre loi de commande ou encore réviser l'emplacement et le type des capteurs dans le robot, sont tâches susceptibles de réalisation. Il faudra aussi prendre en compte les nouvelles restrictions qu'accrocher le fauteuil implique, puisqu'on a déjà vérifié que les capteurs ne sont pas utilisables dans cette situation.

D'un point de vue personnel, ce projet a été très satisfaisant puisque il m'a permis d'acquérir beaucoup de connaissances par rapport à l'automatique, aux systèmes d'acquisition de données et à la programmation de systèmes robotiques mobiles. En bref, très utile de côte de la didactique, ce projet m'a permis aussi de m'intégrer au sein du laboratoire LMSP et de travailler en équipe avec d'autres élèves, de provenances différentes, en échangeant des problèmes, des solutions et des connaissances acquises.

6. BIBLIOGRAPHIE.

[VEN07] Robotique mobile pour l'assistance aux personnes handicapées : conception mécanique et implantation électronique ; rapport final de PJE ; Adrien VENGEANT ; ENSAM Paris 2007.

[SIX07] Partie électronique du robot HAMMI ; rapport final de PJE ; Aurélien SIXTE ; ENSAM Paris 2007.

[NUN07] Robot HAMMI : programmation du robot pour l'assistance aux personnes handicapées ; rapport final de PJE ; David NUNEZ ; ENSAM Paris 2007.

[RUB07] Robot mobile pour l'assistance aux personnes handicapées : Partie Commande ; Sébastien RUBRECHT ; ENSAM Paris 2007.

[EVE08] Optimisation électronique d'un robot mobile d'assistance aux personnes handicapées : le robot HAMMI ; EVEN Pierre ; ENSAM Paris 2008.

[GEN08] Optimisation mécanique et programmation d'un robot mobile d'assistance aux personnes handicapées : le robot HAMMI ; Florian GENEST ; ENSAM Paris 2008.

[MIC00] Formation à Visual C++ 6.0 ; Chuck Sphar ; Microsoft Press 2000.

[BEL93] M. Bellanger, " Traitement numérique du signal", Edition Dunod, 1993.

[HDP07] Rapport de l'IReSP (Institut de Recherche en Santé Publique) « LE HANDICAP, NOUVEL ENJEU DE SANTE PUBLIQUE » :
http://www.cnrs.fr/infoslabos/appels-offres/docs/Appel_a_projets_handicap_IReSP_2007.pdf

[MNI99] Description et User Manual de la carte PCI-6602 dans le site de National Instruments:
http://sine.ni.com/nips/cds/view/p/lang/en/nid/1123.
http://www.ni.com/pdf/manuals/322137b.pdf.

[MNI08] Description et User Manual de la carte PCI-6221 dans le site de National Instruments :
http://sine.ni.com/nips/cds/view/p/lang/en/nid/14132.
http://www.ni.com/pdf/manuals/371022k.pdf.

[TNI09] Tutorial "Creating a Measurement Studio for Visual C++ 6.0 Project" dans le site de National Instruments :
http://zone.ni.com/devzone/cda/tut/p/id/3177.

[STG09] Site du logiciel *STATGRAPHICS Centurion XV* où l'on peut télécharger :
www.**statgraphics**.com/

ANNEXES

SOMMAIRE DES ANNEXES

SOMMAIRE DES FIGURES

ANNEXE A1 : LIAISONS ENTRE LES CARTES NI ET LES AUTRES COMPOSANTS.

Cette annexe a pour but éclairer les détails du lien des éléments du robot, surtout par rapport à ses cartes NI. Auparavant on a déjà passé sur ce point-ci, mais c'est maintenant qu'on fera d'une façon plus précise.

Tout d'abord, on présente dans la figure A1.1 un diagramme simplifié des éléments qui composent actuellement le robot pousseur :

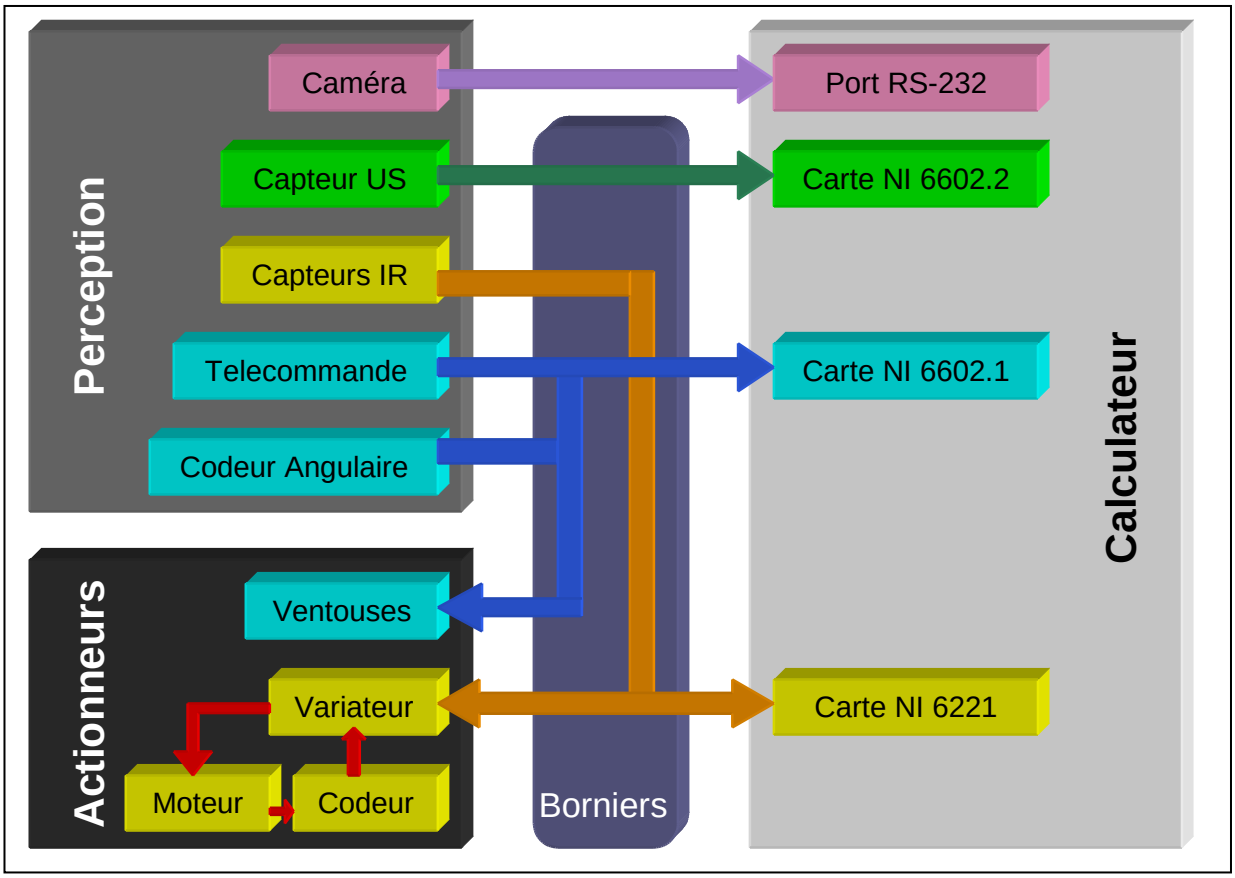

Fig. A1.1 : Diagramme Simplifié des Liaisons

On remarque ici les différents composants du robot, que l'on a présenté au paragraphe 1.1. On a groupé les éléments en trois classes pour une meilleure compréhension : tout ce qui est dans le calculateur, tout ce qui concerne la perception et ce qui concerne les actionneurs. On peut remarquer aussi que les liaisons, montrées d'une autre façon dans le tableau 1.7, coïncident.

En plus des flèches, les éléments connectés sont coloriés de la même couleur. Il faut noter aussi que les câbles et les borniers qui permettent les liaisons de la figure A1.1 ont leurs propres appellations : ANA pour la carte NI 6221, NUM1 pour celle NI 6602.1 et NUM2 pour celle NI 6602.2. Ces Câbles et borniers sont ainsi étiquetés sur le robot.

Maintenant, on montre le détail des liaisons aux ports des cartes NI ainsi qu'aux bornes des borniers NI. On a groupé les liaisons selon les cartes :

NI 6221	Voie	Bornier ANA
Consigne Vitesse Moteur 1	AO 1 / AO GND	21 /54
Consigne Vitesse Moteur 2	AO 0 /AO GND	22 /55
Mesure Vitesse Codeur 1	AI 1 (+) / AI 9 (-)	33 (+) / 66 (-)
Mesure Vitesse Codeur 2	AI 8 (+) / AI 0 (-)	34 (+) / 68 (-)
Capteur à Infrarouges 1	AI 3	30
Capteur à Infrarouges 2	AI 4	28
Capteur à Infrarouges 3	AI 11	63
Capteur à Infrarouges 4	AI 12	61

NI 6602.1	Voie	Bornier NUM1
Channel 3 télécommande	PWM Gate 0	3
Channel 2 télécommande	PWM Gate 1	8
Channel 6 télécommande	PWM Gate 2	67
Codeur Angulaire	PF_0 – PF_12	53(b12)-51/48-47/ 17-12/45-44/10(b0)
Ventouses	PFI_13	54

NI 6602.2	Voie	Bornier NUM2
Capteur à Ultrasons	PWM Gate 0	3

Fig. A1.2 : Tableaux des Liaisons des Cartes NI

Avec ces nouveaux renseignements, on est tout à fait en disposition de comprendre la programmation future des cartes NI, c'est-à-dire, les applications qui seront faites et exécutées dans le calculateur du robot pousseur. Toutes ces informations ont été obtenues à partir des rapports référencés en bibliographie [VEN07], [SIX07], [NUN07], [EVE08] et [GEN08], ainsi que du site de National Instruments [MNI99] et [MNI08].

ANNEXE A2 :
DONNEES DE L'ETUDE DU BRUIT DES CAPTEURS.

La finalité de cette annexe est de présenter toutes les données en rapport avec l'étude du bruit proposée dans le paragraphe 2.2.1, encadré sur le traitement électrique des signaux des capteurs. D'abord, on montre les trois cas essayés pendant la première analyse des capteurs, avant d'implanter les filtres analogique et numérique.

Échantillon	Moyenne [mV]	Variance [x10^{-3} V^2]
1	713,272	0,043395
2	713,158	0,053569
3	714,364	0,058075
4	713,369	0,058641
5	712,561	0,033776
6	714,672	0,037925
7	712,992	0,035924
8	714,552	0,112933
9	713,928	0,034215
10	713,222	0,051537
11	713,923	0,058586
12	713,508	0,05706
13	715,167	0,051062
14	714,879	0,111352
15	712,515	0,054938
16	713,803	0,063717
17	712,69	0,028231
18	715,328	0,079828
19	713,005	0,052481
20	714,165	0,047291
21	714,383	0,04778
22	713,609	0,044519
23	713,068	0,044279
24	714,058	0,063205
25	714,699	0,039124

Fig. A2.1 : Tableau du 1er Cas

Échantillon	Moyenne [mV]	Variance [x10^{-3} V^2]
1	712,258	0,058547
2	713,821	0,065809
3	714,391	0,077026
4	712,159	0,058765
5	712,066	0,049001
6	713,751	0,077791
7	711,712	0,048639
8	711,679	0,089031
9	713,529	0,064161
10	714,492	0,082975
11	713,573	0,060482
12	711,196	0,050091
13	711,679	0,052776
14	714,028	0,120878
15	713,102	0,055328
16	713,418	0,079349
17	711,634	0,05874
18	711,736	0,053353
19	712,256	0,055351
20	712,12	0,065727
21	711,409	0,057254
22	712,149	0,05728
23	711,725	0,052995
24	711,046	0,077111
25	713,144	0,114566

Fig. A2.2 : Tableau du 2^{e} Cas

Échantillon	Moyenne [mV]	Variance [x10^{-3} V^2]
1	714,806	0,225606
2	713,751	0,176076
3	714,257	0,138862
4	714,821	0,158006
5	712,739	0,191618
6	714,077	0,168371
7	714,054	0,153742
8	715,756	0,206761
9	714,019	0,139774
10	714,071	0,141736
11	715,164	0,140803
12	713,964	0,139399
13	713,64	0,171579
14	713,771	0,12758
15	715,007	0,180689
16	713,755	0,151081
17	714,051	0,196296
18	716,877	0,200907
19	713,312	0,129202
20	715,14	0,112548
21	714,194	0,186319
22	714,634	0,166655
23	713,097	0,118742
24	715,376	0,166714
25	712,956	0,147922

Fig. A2.3 : Tableau du 3^e Cas

Pour finir cette annexe, il faut présenter le tableau de données qui correspond à l'essai avec tous les capteurs connectés après avoir mis en œuvre la solution choisie. C'est l'analyse de ces données qui a permis d'affirmer que la solution est satisfaisante. Voici le tableau :

Échantillon	Moyenne [mV]	Variance [x10^{-3} V^2]
1	711,619	0,049
2	712,505	0,049
3	712,491	0,049
4	711,854	0,036
5	712,097	0,049
6	713,051	0,036
7	711,029	0,025
8	712,352	0,036
9	711,801	0,036
10	712,410	0,049
11	712,783	0,049
12	712,609	0,036
13	711,700	0,036
14	713,074	0,049
15	712,462	0,036
16	713,282	0,049
17	712,706	0,049
18	712,275	0,036
19	712,643	0,036
20	712,808	0,064
21	712,412	0,036
22	712,415	0,049
23	712,659	0,049
24	711,391	0,036
25	711,762	0,036

Fig. A2.4 : Tableau du 3^e Cas (Mise en Place des Filtres)

ANNEXE A3 :
DONNEES DE L'ETUDE D'ETALONNAGE DES CAPTEURS.

Dans cette annexe on va exposer toutes les données recueillies lors de l'essai pour étalonner les capteurs de distances du robot ; ce qui est décrit dans le paragraphe 2.2.2. Rappelons que, pour présenter la méthode de cet essai, les données par rapport au capteur IR4 sont déjà affichées dans ce paragraphe.

Commençons par le capteur à ultrasons, c'est-à-dire, le capteur US :

Essai US	Moyenne [ms]	Distance [cm]
1	0.928532	20
2	2.148	40
3	3.334	60
4	4.52	80
5	5.558	100
6	6.744	120
7	7.93	140
8	8.821	160

Fig. A3.1 : Données pour l'étalonnage du Capteur US

Continuons par les capteurs infrarouges, exactement par le capteur IR1 :

Essai IR1	Moyenne [V]	Distance [cm]
1	0.4819544	120
2	0.5657984	110
3	0.5777564	100
4	0.618007	95
5	0.6571066	90
6	0.7250232	85
7	0.763473	80
8	0.8079788	75
9	0.8764126	70
10	0.9512368	65
11	1.032	60
12	1.1248	55
13	1.2556	50
14	1.312	45
15	1.57	40
16	1.7568	35
17	2.032	30
18	2.315	25
19	2.5732	20

Fig. A3.2 : Données pour l'étalonnage du Capteur IR1

Voici les résultats pour le capteur IR2 :

Essai IR2	Moyenne [V]	Distance [cm]
1	0.509743	120
2	0.5658072	110
3	0.5890556	100
4	0.646677	95
5	0.6993642	90
6	0.738361	85
7	0.8009894	80
8	0.8580592	75
9	0.9261948	70
10	0.9988758	65
11	1.0776	60
12	1.1946	55
13	1.3098	50
14	1.4756	45
15	1.64	40
16	1.863	35
17	2.1386	30
18	2.4074	25
19	2.6624	20

Fig. A3.3 : Données pour l'étalonnage du Capteur IR2

Enfin, le tableau de données du capteur IR3 :

Essai IR3	Moyenne [V]	Distance [cm]
1	0.4672332	120
2	0.5252224	110
3	0.6065078	100
4	0.6218164	95
5	0.6452556	90
6	0.689134	85
7	0.7606944	80
8	0.8184098	75
9	0.8724766	70
10	0.8915198	65
11	0.9896116	60
12	1.098	55
13	1.2324	50
14	1.3626	45
15	1.511	40
16	1.73	35
17	2.0034	30
18	2.2778	25
19	2.5568	20

Fig. A3.4 : Données pour l'étalonnage du Capteur IR3

ANNEXE A4 :
CODE DU FICHIER INTERPOLER.M.

Le but de cette annexe est de présenter le code du fichier de *Matlab* appelé *interpoler.m*. Ce fichier est utilisé lors de l'étalonnage du paragraphe 2.2.2 pour obtenir les coefficients des polynômes qui réalisent la conversion des mesures primaires (volts ou millisecondes) en des mesures secondaires (centimètres).

Les données introduites dans ce fichier sont exposées dans l'annexe précédente et son fonctionnement est de même expliqué dans le paragraphe 2.2.2. Ainsi, il ne reste que montrer le code commenté :

```
//*************************************************************************************//
//*************************************************************************************//
//                                  Javier SÁEZ CARDADOR                               //
//                                      Février  2009                                  //
//                                    E.N.S.A.M. Paris                                 //
//                                                                                     //
//                                      interpoler.m                                   //
//                    Fichier pour l'analyse de l'étalonnage des capteurs de distance: //
//     Ce code prend les vecteurs x pour les mesures prises [IR=V/US=ms] et y pour les distances //
//     essayées [cm] et la variable US indiquant si l'interpolation est réalisée pour un capteur ultrason //
//    (US=1)ou// //pas. Ensuite, il rend 4 vecteurs avec les coefficients des interpolations réalisées en //
//     ordre descendant  (0 si non réalisée), dessine les mesures réels ainsi que l'écart type (à partir de //
//                    polyval) et calcule la moyenne de l'écart type et l'erreur relatif.  //
//*************************************************************************************//
//*************************************************************************************//

function [p5,p4,p3,p1] = interpoler(x,y,US)
%Obtention du vecteur x des mesures des capteurs [IR=V / US=ms]
%Obtention du vecteur y des distances essayées [cm]
%Indicateur du type de capteur [US=0->Infrarouges / US=1->Ultrasons]

%Interpolation pour les capteurs Infrarouges
if(US==0)

%Interpolation des mesures obtenant un polynôme de 5e degré
[p5,s5]=polyfit(x,y,5);
%Evaluation du polynôme obtenu pour le représenter
[int5,d5]=polyval(p5,x,s5);
%Pareil pour les polynômes de 4e et 3e degré
[p4,s4]=polyfit(x,y,4);
[int4,d4]=polyval(p4,x,s4);
[p3,s3]=polyfit(x,y,3);
[int3,d3]=polyval(p3,x,s3);

%Représentation des polynômes et des mesures réelles
subplot(1,2,1),plot(x,int5,'m-*',x,int4,'r-x',x,int3,'b-+',x,y,'k','LineWidth',2,'MarkerSize',11);
legend('5e degré','4e degré','3e degré','Mesures');
title('Distance Mesurée [cm] et Poly-interpolation');
xlabel('Mesure du Capteur [V]');
grid;

%Représentation de l'estimation de l'écart de l'interpolation pour chaque distance essayée
subplot(1,2,2),plot(y,d5,'m-*',y,d4,'r-x',y,d3,'b-+','LineWidth',2,'MarkerSize',11);
legend('5e degré','4e degré','3e degré');
title('Écart Type Estimé [cm]');
xlabel('Distance [cm]');
grid;
```

```
%Initialisation des variables pour les calculs de l'estimation de l'écart type moyen et l'erreur relative
ecart_type_moyen5=0;
ecart_type_moyen4=0;
ecart_type_moyen3=0;
erreur_relative5=0;
erreur_relative4=0;
erreur_relative3=0;
iterations=length(d5);

%Boucle de calcul
for i = 1 : 1 : iterations
   %Somme des estimations des écarts déjà dessinés pour le 5e degré
   ecart_type_moyen5=ecart_type_moyen5+d5(i);
   %Calcul et somme des erreurs relatives pour le 5e degré
   erreur_relative5=erreur_relative5+((abs(y(i)-int5(i)))/y(i));
   %Pareil pour les polynômes de 4e et 3e degré
   ecart_type_moyen4=ecart_type_moyen4+d4(i);
   erreur_relative4=erreur_relative4+((abs(y(i)-int4(i)))/y(i));
   ecart_type_moyen3=ecart_type_moyen3+d3(i);
   erreur_relative3=erreur_relative3+((abs(y(i)-int3(i)))/y(i));
end

%Division de la somme des estimations d'écarts pour obtenir la moyenne (5e degré)
ecart_type_moyen5=ecart_type_moyen5/iterations
%Pareil pour les polynômes de 4e et 3e degré
ecart_type_moyen4=ecart_type_moyen4/iterations
ecart_type_moyen3=ecart_type_moyen3/iterations
%Division de la somme d'erreurs relatives et expression en pourcentage
erreur_relative5=(erreur_relative5/iterations)*100
%Pareil pour les polynômes de 4e et 3e degré
erreur_relative4=(erreur_relative4/iterations)*100
erreur_relative3=(erreur_relative3/iterations)*100

%Il n'y a pas eu d'interpolation linéaire
p1=0;

%Interpolation pour les capteurs Ultrasons
else

%Interpolation linéaire pour le capteur US
[p1,s1]=polyfit(x,y,1);
%Evaluation de la fonction obtenue pour la représenter
[int1,d1]=polyval(p1,x,s1);

%Représentation de la ligne et des mesures réelles
subplot(1,2,1),plot(x,int1,'b-x',x,y,'k','LineWidth',2,'MarkerSize',11);
legend('Interpolation linéaire','Mesures');
title('Distance Mesurée et Interpolation Linéaire [cm]');
xlabel('Mesure du Capteur US [ms]');
grid;

%Représentation de l'estimation de l'écart de l'interpolation pour chaque distance essayée
subplot(1,2,2),plot(y,d1,'b-x','LineWidth',2,'MarkerSize',11);
legend('Interp. linéaire');
title('Écart Type Estimé Interpolation [cm]');
xlabel('Distance [cm]');
grid;

%Initialisation des variables pour les calculs de l'estimation de l'écart type moyen et l'erreur relative
```

```
ecart_type_moyen1=0;
erreur_relative1=0;
iterations=length(d1);

%Boucle de calcul
for i = 1 : 1 : iterations
   %Somme des estimations des écarts déjà dessinés
   ecart_type_moyen1=ecart_type_moyen1+d1(i);
   %Calcul et somme des erreurs relatives
   erreur_relative1=erreur_relative1+((abs(y(i)-int1(i)))/y(i));
end

%Division de la somme des estimations d'écarts pour obtenir la moyenne
ecart_type_moyen1=ecart_type_moyen1/iterations
%Division de la somme d'erreurs relatives et expression en pourcentage
erreur_relative1=(erreur_relative1/iterations)*100

%Il n'y a eu que d'interpolation linéaire
p5=0;
p4=0;
p3=0;

end
```

Fig. A4.1 : Code du Fichier Interpoler.m

ANNEXE A5 :
CODE DU FICHIER MODELE_CAPTEURS.M.

Par rapport au modèle cinématique pour les capteurs, qui a été expliqué dans tout le paragraphe 3 ; dans cette annexe on présente le code du fichier de *Matlab* où ce modèle a été implanté : c'est le fichier *modele_capteurs.m*.

Afin d'améliorer la compréhension du code, on montre le diagramme de flux du code :

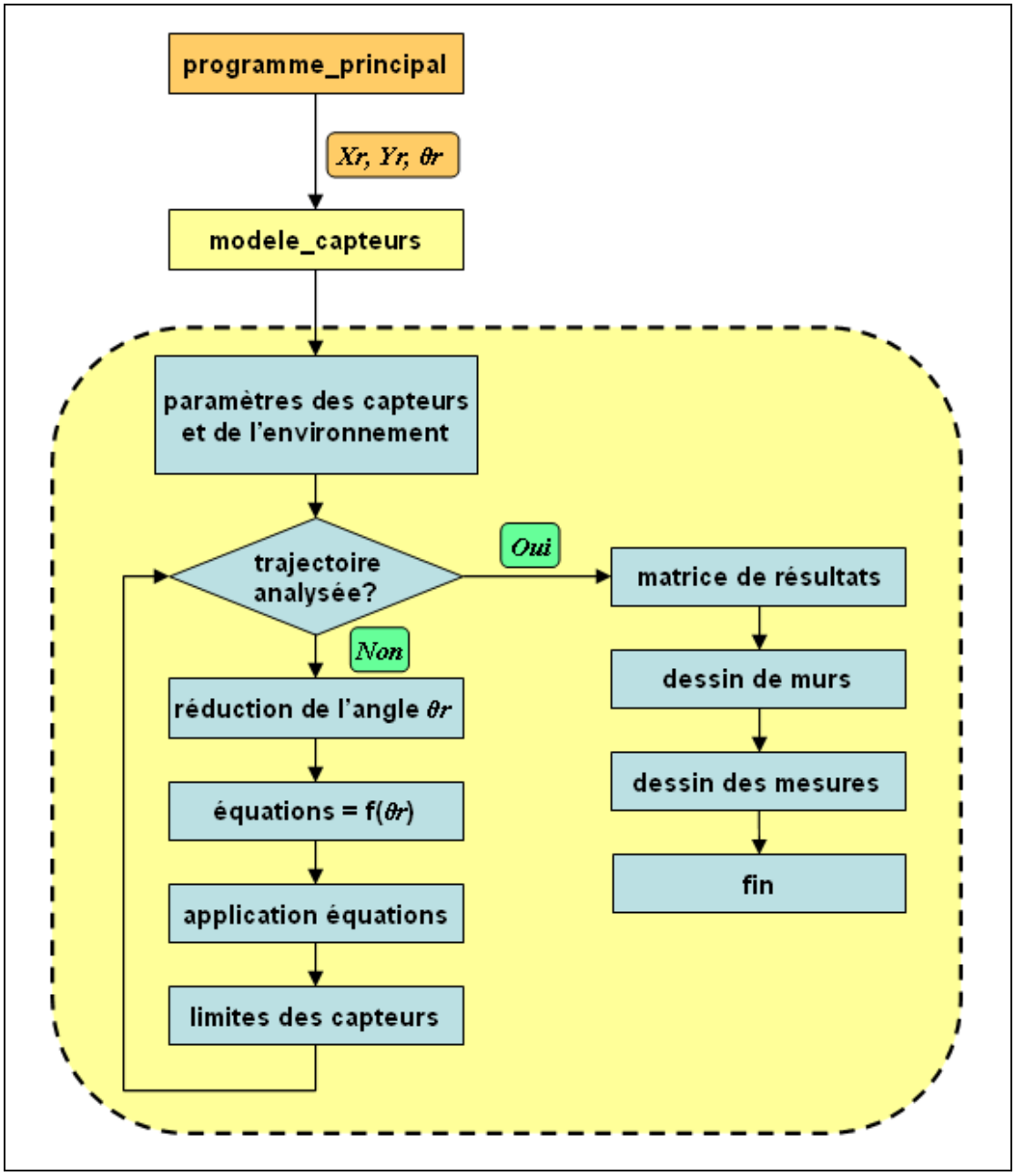

Fig. A5.1 : Diagramme de flux du fichier « modele_capteurs.m »

En finissant ce sujet, il ne reste qu'exposer le code du fichier « modele_capteurs.m » en format *MATLAB*. Ce fichier est exécuté lorsqu'il est appelé dès n'importe quel fichier « programme_principal.m » selon ce qui a été expliqué au paragraphe 3 et à la figure 3.1. Il prend du programme principal le vecteur ***t*** avec les temps de chaque pas de calcul et la matrice ***x*** ; donc ses trois premières colonnes apportent la trajectoire suivie par le centre de gravité du robot (point ***Ar***) à travers des coordonnées ***Xr***, ***Yr*** et ***θr***.

Voici le code commenté:

```
%//*****************************************************************************//
%//*****************************************************************************//
%//                                Javier SÁEZ CARDADOR                         //
%//                                      Avril  2009                            //
%//                                   E.N.S.A.M. Paris                          //
%//                                                                             //
%//                                  modele_capteurs.m                          //
%//                        Fichier du modèle cinématique pour les capteurs de distance:  //
%//                  Ce code prend la matrice x=[Xr,Yr,Thetar...] et le vecteur t, rend un tableau  //
%//       x=[Xr,Yr,Thetar,temps,distances capteurs] et dessine les distances calculées pour chaque pas  //
%//                                      de calcul.                             //
%//*****************************************************************************//
%//*****************************************************************************//

function modele_capteurs(t,x)
%Obtention du temps de chaque pas de calcul en t
%Obtention de la trajectoire (points Ar-->x=[Xr,Yr,Thetar...])

%Paramètres liés au modèle des capteurs (Tableau figure 1.3)
%Angles en rad
gammaIR1 = (50*pi)/180;
gammaIR2 = (72*pi)/180;
gammaIR3 = (110*pi)/180;
gammaIR4 = (135*pi)/180;
gammaUS = (90*pi)/180;
%Distances en mètres
lyIR14 = 0.329;
lyIR23 = 0.355;
lyUS = 0.363;
lxIR1 = 0.091;
lxIR2 = 0.044;
lxIR3 = -0.042;
lxIR4 = -0.090;

%Paramètres de l'Environnement en mètres
xmur1 = -0.5;
xmur2 = 0.5;
ymur1 = 0;
ymur2 = 4;


%Boucle pour l'application de la cinématique des capteurs
taille=size(x);

for i=1 : 1 : taille(1)

   %Réduction de l'angle thetar au rang (0,+/-360)
   while (x(i,3) > 2*pi) || (x(i,3) < -2*pi)
      x(i,3) = x(i,3) - 2*pi;
   end
```

```
equations(i,5)=x(i,3)*180/pi;


%Identification des équations à utiliser en fonction de l'angle thetar
%[x(i:3)]

%IR1
%xmur1,ymur2
if ((x(i,3) > gammaIR1) && (x(i,3) <= (gammaIR1+pi/2))) || ((x(i,3) > (gammaIR1-2*pi)) && (x(i,3)
<= (gammaIR1-3/2*pi)))
    equations(i,1) = 1;
%xmur1,ymur1
elseif ((x(i,3) > (gammaIR1+pi/2)) && (x(i,3) <= (gammaIR1+pi))) || ((x(i,3) > (gammaIR1-3/2*pi))
&& (x(i,3) <= (gammaIR1-pi)))
    equations(i,1) = 2;
%xmur2,ymur1
elseif ((x(i,3) > (gammaIR1+pi)) && (x(i,3) <= (gammaIR1+3/2*pi))) || ((x(i,3) > (gammaIR1-pi)) &&
(x(i,3) <= (gammaIR1-pi/2)))
    equations(i,1) = 3;
%xmur2,ymur2
elseif ((x(i,3) > (gammaIR1+3/2*pi)) || (x(i,3) <= gammaIR1)) && ((x(i,3) > (gammaIR1-pi/2)) ||
(x(i,3) <= (gammaIR1-2*pi)))
    equations(i,1) = 4;
end

%IR2
%xmur1,ymur2
if ((x(i,3) > gammaIR2) && (x(i,3) <= (gammaIR2+pi/2))) || ((x(i,3) > (gammaIR2-2*pi)) && (x(i,3)
<= (gammaIR2-3/2*pi)))
    equations(i,2) = 1;
%xmur1,ymur1
elseif ((x(i,3) > (gammaIR2+pi/2)) && (x(i,3) <= (gammaIR2+pi))) || ((x(i,3) > (gammaIR2-3/2*pi))
&& (x(i,3) <= (gammaIR2-pi)))
    equations(i,2) = 2;
%xmur2,ymur1
elseif ((x(i,3) > (gammaIR2+pi)) && (x(i,3) <= (gammaIR2+3/2*pi))) || ((x(i,3) > (gammaIR2-pi)) &&
(x(i,3) <= (gammaIR2-pi/2)))
    equations(i,2) = 3;
%xmur2,ymur2
elseif ((x(i,3) > (gammaIR2+3/2*pi)) || (x(i,3) <= gammaIR2)) && ((x(i,3) > (gammaIR2-pi/2)) ||
(x(i,3) <= (gammaIR2-2*pi)))
    equations(i,2) = 4;
end

%IR3
%xmur1,ymur2
if ((x(i,3) > gammaIR3) && (x(i,3) <= (gammaIR3+pi/2))) || ((x(i,3) > (gammaIR3-2*pi)) && (x(i,3)
<= (gammaIR3-3/2*pi)))
    equations(i,3) = 1;
%xmur1,ymur1
elseif ((x(i,3) > (gammaIR3+pi/2)) && (x(i,3) <= (gammaIR3+pi))) || ((x(i,3) > (gammaIR3-3/2*pi))
&& (x(i,3) <= (gammaIR3-pi)))
    equations(i,3) = 2;
%xmur2,ymur1
elseif ((x(i,3) > (gammaIR3+pi)) || (x(i,3) <= (gammaIR3-pi/2))) && ((x(i,3) > (gammaIR3-pi)) ||
(x(i,3) <= (gammaIR3-5/2*pi)))
    equations(i,3) = 3;
%xmur2,ymur2
elseif ((x(i,3) > (gammaIR3-pi/2)) && (x(i,3) <= gammaIR3)) || ((x(i,3) > (gammaIR3-5/2*pi)) &&
```

```
(x(i,3) <= (gammaIR3-2*pi)))
      equations(i,3) = 4;
   end

   %IR4
   %xmur1,ymur2
   if ((x(i,3) > gammaIR4) && (x(i,3) <= (gammaIR4+pi/2))) || ((x(i,3) > (gammaIR4-2*pi)) && (x(i,3)
<= (gammaIR4-3/2*pi)))
      equations(i,4) = 1;
   %xmur1,ymur1
   elseif ((x(i,3) > (gammaIR4+pi/2)) && (x(i,3) <= (gammaIR4+pi))) || ((x(i,3) > (gammaIR4-3/2*pi))
&& (x(i,3) <= (gammaIR4-pi)))
      equations(i,4) = 2;
   %xmur2,ymur1
   elseif ((x(i,3) > (gammaIR4+pi)) || (x(i,3) <= (gammaIR4-pi/2))) && ((x(i,3) > (gammaIR4-pi)) ||
(x(i,3) <= (gammaIR4-5/2*pi)))
      equations(i,4) = 3;
   %xmur2,ymur2
   elseif ((x(i,3) > (gammaIR4-pi/2)) && (x(i,3) <= gammaIR4)) || ((x(i,3) > (gammaIR4-5/2*pi)) &&
(x(i,3) <= (gammaIR4-2*pi)))
      equations(i,4) = 4;
   end

   %US
   %xmur1,ymur2
   if ((x(i,3) > gammaUS) && (x(i,3) <= (gammaUS+pi/2))) || ((x(i,3) > (gammaUS-2*pi)) && (x(i,3) <=
(gammaUS-3/2*pi)))
      equations(i,5) = 1;
   %xmur1,ymur1
   elseif ((x(i,3) > (gammaUS+pi/2)) && (x(i,3) <= (gammaUS+pi))) || ((x(i,3) > (gammaUS-3/2*pi))
&& (x(i,3) <= (gammaUS-pi)))
      equations(i,5) = 2;
   %xmur2,ymur1
   elseif ((x(i,3) > (gammaUS+pi)) && (x(i,3) <= (gammaUS+3/2*pi))) || ((x(i,3) > (gammaUS-pi)) &&
((x(i,3) <= (gammaUS-pi/2)) || (x(i,3) <= (gammaUS-5/2*pi))))
      equations(i,5) = 3;
   %xmur2,ymur2
   elseif ((x(i,3) > (gammaUS+3/2*pi)) || ((x(i,3) <= gammaUS)) && ((x(i,3) > (gammaUS-pi/2)) || (x(i,3)
<= (gammaUS-5/2*pi)))) || (x(i,3) <= (gammaUS-2*pi))
      equations(i,5) = 4;
   end


   %Application des équations selon les résultats antérieurs
   %IR1
   if equations(i,1) == 1 || equations(i,1) == 2   %Le rayon est dirigé vers le mur xmur1
      dxIR1 = abs((x(i,1) - xmur1 + lyIR14*sin(pi/2-x(i,3)) - lxIR1*cos(pi/2-x(i,3)))/(cos(gammaIR1 +
(pi/2-x(i,3)))));
   else                                %Le rayon est dirigé vers le mur xmur2
      dxIR1 = abs((xmur2 - x(i,1) + lyIR14*sin(x(i,3)-pi/2) + lxIR1*cos(x(i,3)-pi/2))/(cos((pi/2+x(i,3)) -
gammaIR1)));
   end
   if equations(i,1) == 1 || equations(i,1) == 4   %Le rayon est dirigé vers le mur ymur2
      dyIR1 = abs((ymur2 - x(i,2) - lyIR14*cos(x(i,3)-pi/2) + lxIR1*sin(x(i,3)-pi/2))/(cos(gammaIR1 -
x(i,3))));
   else                                %Le rayon est dirigé vers le mur ymur1
      dyIR1 = abs((x(i,2) - ymur1 + lyIR14*cos(x(i,3)-pi/2) - lxIR1*sin(x(i,3)-pi/2))/(cos(gammaIR1 -
(x(i,3)-pi))));
   end
   dIR1(i)=min(dxIR1,dyIR1);
```

```
   %IR2
   if equations(i,2) == 1 || equations(i,2) == 2   %Le rayon est dirigé vers le mur xmur1
      dxIR2 = abs((x(i,1) - xmur1 + lyIR23*sin(pi/2-x(i,3)) - lxIR2*cos(pi/2-x(i,3)))/(cos(gammaIR2 +
(pi/2-x(i,3)))));
   else                                 %Le rayon est dirigé vers le mur xmur2
      dxIR2 = abs((xmur2 - x(i,1) + lyIR23*sin(x(i,3)-pi/2) + lxIR2*cos(x(i,3)-pi/2))/(cos((pi/2+x(i,3)) -
gammaIR2)));
   end
   if equations(i,2) == 1 || equations(i,2) == 4   %Le rayon est dirigé vers le mur ymur2
      dyIR2 = abs((ymur2 - x(i,2) - lyIR23*cos(x(i,3)-pi/2) + lxIR2*sin(x(i,3)-pi/2))/(cos(gammaIR2 -
x(i,3))));
   else                                 %Le rayon est dirigé vers le mur ymur1
      dyIR2 = abs((x(i,2) - ymur1 + lyIR23*cos(x(i,3)-pi/2) - lxIR2*sin(x(i,3)-pi/2))/(cos(gammaIR2 -
(x(i,3)-pi))));
   end
   dIR2(i)=min(dxIR2,dyIR2);

   %IR3
   if equations(i,3) == 1 || equations(i,3) == 2   %Le rayon est dirigé vers le mur xmur1
      dxIR3 = abs((x(i,1) - xmur1 - lyIR23*sin(x(i,3)-pi/2) - lxIR3*cos(x(i,3)-pi/2))/(cos(gammaIR3 +
(pi/2-x(i,3)))));
   else                                 %Le rayon est dirigé vers le mur xmur2
      dxIR3 = abs((xmur2 - x(i,1) - lyIR23*sin(pi/2-x(i,3)) + lxIR3*cos(pi/2-x(i,3)))/(cos((pi/2+x(i,3)) -
gammaIR3)));
   end
   if equations(i,3) == 1 || equations(i,3) == 4   %Le rayon est dirigé vers le mur ymur2
      dyIR3 = abs((ymur2 - x(i,2) - lyIR23*cos(x(i,3)-pi/2) + lxIR3*sin(x(i,3)-pi/2))/(cos(gammaIR3 -
x(i,3))));
   else                                 %Le rayon est dirigé vers le mur ymur1
      dyIR3 = abs((x(i,2) - ymur1 + lyIR23*cos(x(i,3)-pi/2) - lxIR3*sin(x(i,3)-pi/2))/(cos(gammaIR3 -
(x(i,3)-pi))));
   end
   dIR3(i)=min(dxIR3,dyIR3);

   %IR4
   if equations(i,4) == 1 || equations(i,4) == 2   %Le rayon est dirigé vers le mur xmur1
      dxIR4 = abs((x(i,1) - xmur1 + lyIR14*sin(pi/2-x(i,3)) - lxIR4*cos(pi/2-x(i,3)))/(cos(gammaIR4 +
(pi/2-x(i,3)))));
   else                                 %Le rayon est dirigé vers le mur xmur2
      dxIR4 = abs((xmur2 - x(i,1) - lyIR14*sin(pi/2-x(i,3)) + lxIR4*cos(pi/2-x(i,3)))/(cos((pi/2+x(i,3)) -
gammaIR4)));
   end
   if equations(i,4) == 1 || equations(i,4) == 4   %Le rayon est dirigé vers le mur ymur2
      dyIR4 = abs((ymur2 - x(i,2) - lyIR14*cos(x(i,3)-pi/2) + lxIR4*sin(x(i,3)-pi/2))/(cos(gammaIR4 -
x(i,3))));
   else                                 %Le rayon est dirigé vers le mur ymur1
      dyIR4 = abs((x(i,2) - ymur1 + lyIR14*cos(x(i,3)-pi/2) - lxIR4*sin(x(i,3)-pi/2))/(cos(gammaIR4 -
(x(i,3)-pi))));
   end
   dIR4(i)=min(dxIR4,dyIR4);

   %US
   if equations(i,5) == 1 || equations(i,5) == 2   %Le rayon est dirigé vers le mur xmur1
      dxUS = abs((x(i,1) - xmur1 + lyUS*sin(pi/2-x(i,3)))/(cos(gammaUS + (pi/2-x(i,3)))));
   else                                 %Le rayon est dirigé vers le mur xmur2
      dxUS = abs((xmur2 - x(i,1) + lyUS*sin(x(i,3)-pi/2))/(cos((pi/2+x(i,3)) - gammaUS)));
   end
   if equations(i,5) == 1 || equations(i,5) == 4   %Le rayon est dirigé vers le mur ymur2
      dyUS = abs((ymur2 - x(i,2) - lyUS*cos(x(i,3)-pi/2))/(cos(gammaUS - x(i,3))));
```

```
    else                                %Le rayon est dirigé vers le mur ymur1
        dyUS = abs((x(i,2) - ymur1 + lyUS*cos(x(i,3)-pi/2))/(cos(gammaUS - (x(i,3)-pi))));
    end
    dUS(i)=min(dxUS,dyUS);


    %Modélisation des limites des capteurs
    if dIR1(i) < 0.2
        dIR1(i) = 0;
    elseif dIR1(i) > 1.2
        dIR1(i) = 1.2;
    end
    if dIR2(i) < 0.2
        dIR21(i) = 0;
    elseif dIR2(i) > 1.2
        dIR2(i) = 1.2;
    end
    if dIR3(i) < 0.2
        dIR3(i) = 0;
    elseif dIR3(i) > 1.2
        dIR3(i) = 1.2;
    end
    if dIR4(i) < 0.2
        dIR4(i) = 0;
    elseif dIR4(i) > 1.2
        dIR4(i) = 1.2;
    end
    if dUS(i) < 0.1
        dUS(i) = 0;
    elseif dUS(i) > 6
        dUS(i) = 6;
    end
end


%Remplissage de matrice des résultats

x(:, 4 : 9)=[t,dIR1',dIR2',dIR3',dIR4',dUS'];
x(:,:)

%Dessin des murs dans la figure 1
figure(1)
hold on
i=[-1,1];
j=[-1,5];
xmur1(1 : 2) = xmur1;
xmur2(1 : 2) = xmur2;
ymur1(1 : 2) = ymur1;
ymur2(1 : 2) = ymur2;
plot(xmur1,j,'-r',xmur2,j,'-r',i,ymur1,'-r',i,ymur2,'-r','LineWidth',3)
hold off

%Dessin des mesures des capteurs
figure(3)
hold on
plot(t,dIR1,'-+k','LineWidth',2,'MarkerSize',11)
plot(t,dIR2,'-ob','LineWidth',2,'MarkerSize',11)
plot(t,dIR3,'-*m','LineWidth',2,'MarkerSize',11)
plot(t,dIR4,'-xr','LineWidth',2,'MarkerSize',11)
plot(t,dUS,'-sk','LineWidth',2,'MarkerSize',11)
```

```
hold off
legend('dIR1','dIR2','dIR3','dIR4','dUS');
xlabel('temps [s]')
title('Distance mesurée [m]')
grid minor

end
```

Fig. A5.2 : Code du Fichier « modele_capteurs.m »

ANNEXE A6 :
EXEMPLE DE PROGRAMMATION AVEC DES COMPOSANTS MEASUREMENT STUDIO.

Dans cette annexe on va expliquer comment faire la programmation de l'application qui est montrée comme exemple dans le paragraphe 4.3 et plus précisément, dans la figure 4.12. On rappelle cette figure :

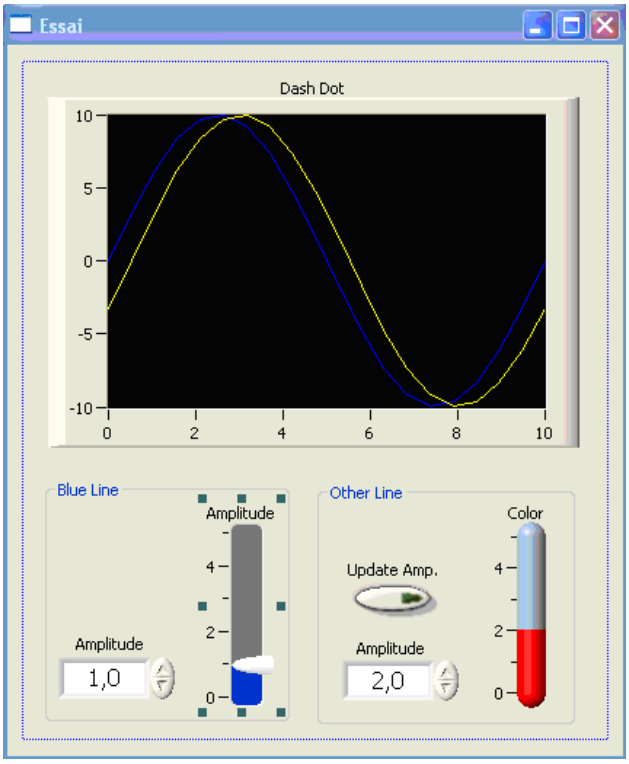

Fig. A6.1 : Application de l'Exemple

Il s'agit d'une application simple qui montre deux signaux sinusoïdaux. Ces signaux varient en leur amplitude et l'un d'eux en couleur, à travers des contrôles ajoutés.

Il y a deux rectangles qui entourent les contrôles des signaux. Ils sont nommes *Blue Line* et *Other Line* selon les lignes qui mènent. Ainsi, dans le rectangle *Blue Line*, il y a une barre qui permet de varier l'amplitude de la sinusoïde bleue et aussi une boîte numérique qui peut être changé avec les flèches ou en écrivant dedans et qui a la même fonction que la barre. Les changements seront effectifs en même temps dans la ligne, la barre et la boîte numérique ; n'importe lequel peut être modifié.

Un autre mécanisme est programmé pour l'autre signal ; il pourra tout simplement être changé à travers de la boîte numérique en écrivant directement le chiffre ou en appuyant les flèches. Pour compléter le changement, on verra qu'un indicateur a été allumé sur le bouton nommé *Update Amp* ; alors, il faudra l'appuyer. En ce moment-là, l'amplitude du signal change, ainsi que sa couleur et son trait en dépendant du rang du nombre introduit. La barre appelée Color est également actualisée et montre la nouvelle amplitude en rouge et change aussi sa couleur au fond.

On a déjà expliqué les fonctions de cette application didactique et on passera maintenant à montrer le code et le processus pour arriver à construire une telle

application. Tout d'abord, il faut dire que, cette fois, l'architecture choisie a exceptionnellement été celle de *Dialog Based*. On fait ce choix pour simplifier l'exemple qui a vraiment pour but l'usage des composants de *Measurement Studio*. Pour cette procédure, on va remonter au deuxième pas du paragraphe 4.2.1 (Usage Basique) et on va montrer, dans la figure suivante, les choix qu'il faut retenir en ce cas :

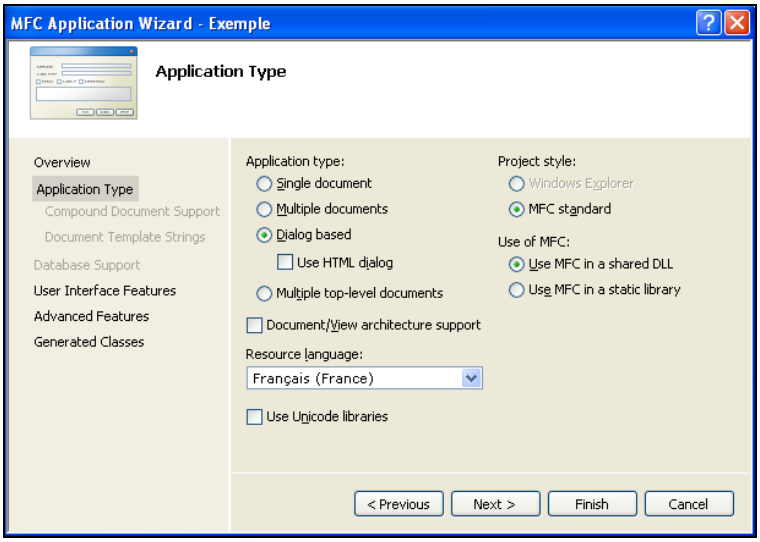

Fig. A6.2 : Choix pour l'Architecture Dialog-Based

Après avoir choisi les choix indiqués, on peut clique *Finish* pour commencer l'édition et programmation.

D'abord, il faudra ouvrir le fichier ressources et la vue du dialogue *IDD_NOM_DIALOG*, tel qu'on peut voir dans la figure 4.12. Selon l'expliqué dans le paragraphe 4.3. (National Instruments) on n'a qu'ajouter la fenêtre et éditer les éléments qu'offre *Measurement Studio* jusqu'à arriver à la figure A6.2. Pour le graphique on a utilisé le *CWGraph*, pour les barres le *CWSlide*, pour le bouton le *CWButton* et pour les boîtes numériques le *CWNumEdit*.

Quant à l'édition du graphique, on peut remarquer qu'il faut choisir les deux signaux sinusoïdaux, les axes et échelles convenants, les couleurs, etc. Pour éditer le graphique, il faut sélectionner le graphique et appuyer, dans la fenêtre de propriétés, sur le bouton de *Pages de Propriétés* à droite. On a déjà vu un peu de cette édition dans la figure 4.16. Maintenant, on apprécie l'addition des lignes en appuyant sur Plots à gauche :

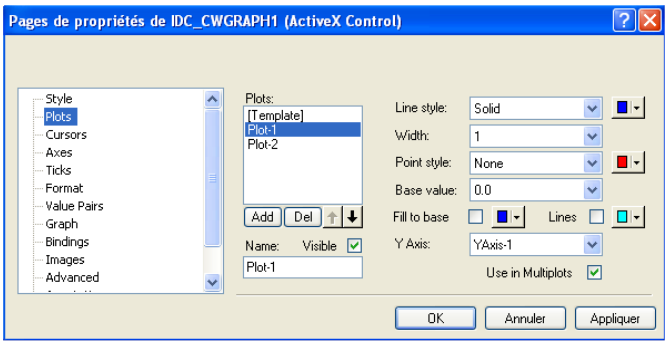

Fig. A6.3 : Propriétés du CWGraph (Plots)

On ajoute ainsi la deuxième ligne ou signal et on établie les couleurs de toutes les deux lignes, lesquelles peuvent être variées au cours de l'exécution de l'application.

Par rapport au *CWSlide*, de même façon que le *CWGraph*, son édition a été déjà montrée dans la figure 4.13 du paragraphe 4.3. (National Instruments). On va commenter simplement ici que l'apparence du *CWSlide* peut varier dès une barre avec pointeur (celui de la ligne bleue) jusqu'à un thermomètre (celui de l'autre ligne). Aussi, il peut être vertical ou horizontal, on peut fixer la couleur ou la valeur initiale, l'échelle, etc. La procédure pour éditer cet élément est pareille à celle des autres.

Egalement, on peut éditer d'autres composants de *Measurement Studio* qui apparaissent dans cette application en obtenant la figure montrée. Il n'y a que traîner les éléments de la boîte d'outils à la fenêtre du dialogue et les modifier.

Une fois on a atteint l'apparence requise, c'est le moment de commencer à ajouter les gestionnaires d'événements qui apporteront de l'utilité aux contrôles et de la fonctionnalité aux graphiques. Dans le paragraphe 4.2.1. (Usage Basique) on a déjà vu comment ajouter ces gestionnaires. Maintenant on va préciser les gestionnaires de cette application ainsi que leurs fonctions.

De cette façon, pour les boîtes numériques on va utiliser le gestionnaire *ValueChanged*, pour les barres le *PointerValueChanged* et pour le bouton le *Click*. Ainsi, on arrive à la configuration suivante :

- Barre de la ligne bleue : PointerValueChangedCwslide1.
- Barre de l'autre ligne : PointerValueChangedCwslide2.
- Boîte N. de la ligne bleue : ValueChangedCwnumedit2.
- Boîte N. de l'autre ligne: ValueChangedCwnumedit1.
- Bouton : ClickCwboolean1.

Il reste encore de remplir tout le code ajouté automatiquement dans le fichier NOMDlg.cpp et éventuellement, d'ajouter quelques variables dans le fichier NOMDlg.h. Mais d'abord, on présente l'initialisation du graphique ainsi que de la barre à gauche dans le commence du fichier NOMDlg.cpp :

```
// TODO : ajoutez ici une initialisation supplémentaire
    CNiMath::SineWave(m_wave,100,1);          //Génère un signal sinusoïdale.Ech:100/Amp:1
    m_graph.Plots.Item(1).PlotY(m_wave);      //Peint le signal sur le graphique
    CNiReal64Vector temp(m_wave);             //Crée un signal pareil
    temp.Scale(2);                            //Multiplie par 2 l'amplitude du 2e signal
    m_graph.Plots.Item(2).PlotY(temp);        //Peint le 2e signal
    m_slide1.Value=1;                         //Initialise CWSlide1 à 1
```

Fig. A6.4 : Initialisation du Dialogue

On remarque, dans la figure A6.4, le code qui établit les valeurs initiales des signaux du graphique et de la valeur de la barre à gauche (*CWSlide1*). Il est facile de suivre les instructions à travers les commentaires. On voit ici quelques éléments apportés par NI comme la classe *CNiReal64Vector* pour créer un signal ou l'objet m_slide, déclarée dans le fichier *NOMDlg.h*, de la classe *CNiSlide*.

A la suite, on expliquer enfin le code ajouté aux gestionnaires générés par Visual Studio. Tout d'abord, on montre quelques-uns des gestionnaires assez simples dans la figure suivante :

```
void CEssaiDlg::PointerValueChangedCwslide1(long Pointer, VARIANT* Value)
{
    CNiReal64Vector temp(m_wave);                  //Génère un signal pour la modifier
    temp.Scale(CNiVariant(Value));                 //Modifie l'amplitude selon la valeur du pointeur
    m_graph.Plots.Item(1).PlotY(temp);              //Peint le signal
    m_amp.Value=m_slide1.Value;                    //Actualise la valeur de la boîte numérique2
}

void CEssaiDlg::ValueChangedCwnumedit2(VARIANT* Value, VARIANT* PreviousValue, BOOL
OutOfRange)
{
            CNiReal64Vector temp(m_wave);          //Génère un signal pour la modifier
            temp.Scale(CNiVariant(Value));         //Modifie l'amplitude selon la valeur de la boîte2
            m_graph.Plots.Item(1).PlotY(temp);     //Peint le signal
            m_slide1.Value=m_amp.Value;            //Actualise la valeur barre1
}

void CEssaiDlg::ClickCwboolean1()
{
            CNiReal64Vector temp(m_wave);          //Génère un signal pour la modifier
            temp.Scale(CNiVariant(m_color.Value)); //Modifie l'amplitude selon la boîte1
            m_graph.Plots.Item(2).PlotY(temp);     //Peint le signal
            m_slide2.Value=m_color.Value;          //Actualise la valeur de la barre2
}

void CEssaiDlg::ValueChangedCwnumedit1(VARIANT* Value, VARIANT* PreviousValue, BOOL
OutOfRange)
{
          m_on.Value=TRUE;                         //Met à 1 la valeur du bouton à traver du code

}
```

Fig. A6.5 : Gestionnaires Partie 1

Le premier gestionnaire qu'on peut voir est celui de la barre (*CWSlide1*) et on peut remarquer comment il peint le signal selon la nouvelle valeur et après actualise la boîte numérique associée. Une procédure similaire peut être observée dans le gestionnaire de la boîte numérique à gauche(*CWNumEdit2*).

Après cela, le gestionnaire du bouton (*ClickCwboolean1*) modifie l'amplitude du signal qui change sa couleur et aussi actualise la valeur de la barre à droite ou thermomètre. Tout cela est fait selon la valeur qui a la boîte numérique à droite (*m_color.Value*) au moment où le bouton est appuyé. La valeur du bouton est mise à TRUE et il est montré comment appuyer lorsque l'on modifie pour la première fois la boîte numérique à droite.

Pourtant, le changement du trait du signal qui varie sa couleur ainsi que celui de la couleur de fond de la barre on fait dans le gestionnaire de la barre à droite. Cette barre ou thermomètre n'a pas de pointeur et donc elle ne peut pas être modifiée dans le dialogue lors de l'exécution de l'application. Sa valeur est ainsi changée à travers du code et plus précisément, à travers de le gestionnaire du bouton.

Le code du gestionnaire *PointerValueChangedCwslide2* est montré dans la figure A6.6 :

```
void CEssaiDlg::PointerValueChangedCwslide2(long Pointer, VARIANT* Value)
{
	int color=CNiVariant(Value);			//Crée une variable avec la valeur de la barre
	CNiPlot plot=m_graph.Plots.Item(2);		//Génère un signal pour la modifier
	CNiReal64Vector temp(m_wave);
	temp.Scale(CNiVariant(m_color.Value));	//Conserve l'amplitude du signal existant
	switch(color)
	{
		case 0:
			plot.LineColor=CNiColor(0,255,255);		//Coleur du signal cyan
			m_slide2.InteriorColor=CNiColor(0,255,255);//Coleur du fond de barre cyan
			plot.LineStyle=CNiPlot::LineSolid;		//Style du signal solide
			break;
		case 1:
			plot.LineColor=CNiColor(0,255,0);		//Coleur du signal vert
			m_slide2.InteriorColor=CNiColor(0,255,0);	//Coleur du fond de barre vert
			plot.LineStyle=CNiPlot::LineSolid;		//Style du signal solide
			break;
		case 2:
			plot.LineColor=CNiColor(255,255,0);		//Coleur du signal jaune
			m_slide2.InteriorColor=CNiColor(255,255,0); //Coleur du fond de barre jaune
			plot.LineStyle=CNiPlot::LineSolid;		//Style du signal solide
			break;
		case 3:
			plot.LineColor=CNiColor(255,0,0);		//Coleur du signal rouge
			m_slide2.InteriorColor=CNiColor(255,0,255); //Coleur du fond de barre rouge
			plot.LineStyle=CNiPlot::LineSolid;		//Style du signal solide
			break;
		case 4:
			plot.LineColor=CNiColor(255,0,0);		//Coleur du signal rouge
			m_slide2.InteriorColor=CNiColor(255,0,255); //Coleur du fond de barre mag.
			plot.LineStyle=CNiPlot::LineDash;		//Style du signal tiret
			break;
		default:
			plot.LineColor=CNiColor(255,0,255);		//Coleur du fond de barre mag.
			m_slide2.InteriorColor=CNiColor(255,0,255); //Coleur du fond de barre mag.
			plot.LineStyle=CNiPlot::LineDash;		//Style du signal tiret
	}
	plot.PlotY(temp);				//Peint le signal
}
```

Fig. A6.6 : Gestionnaires Partie 2

On peut remarquer que, d'abord, on crée une variable qui sert à gérer le *switch* et après, on génère un signal avec la même amplitude fixé selon la boîte numérique à droite mais permettant de changer son style et sa couleur. Puis, dans le *switch*, on assigne une couleur à la ligne du signal et à la barre à droite (qui parfois peuvent ne pas coïncider) et aussi un style au signal (soit solide, soit à tirets).

Pour finir, on peut commenter que toutes les déclarations des classes ici utilisées se trouvent dans le fichier NOMDlg.h et qu'elles sont, presque toutes, automatiquement ajoutées lors de la création des gestionnaires. Ces déclarations et celles des gestionnaires peuvent être appréciées dans la figure A6.7 :

```
public:
        NI::CNiGraph m_graph;   //Graphique. Sert à peintre
public:
        NI::CNiSlide m_slide1;      //Barre à gauche. Contient la valeur parmis d'autres
        CNiReal64Vector m_wave; //Vecteur ajouté à main. Garde les signaux
public:
        DECLARE_EVENTSINK_MAP()
public:                             //Gestionnaire de la barre à gauche
        void PointerValueChangedCwslide1(long Pointer, VARIANT* Value);
public:
        NI::CNiNumEdit m_amp;//Boîte numérique à gauche. Contient la valeur...
public:                             //Gestionnaire de la boîte numérique à gauche
        void ValueChangedCwnumedit2(VARIANT* Value, VARIANT* PreviousValue, BOOL
OutOfRange);
public:
        NI::CNiNumEdit m_color;//Boîte numérique à droite. Contient la valeur...
public:                             //Gestionnaire de la boîte numérique à droite
        void ValueChangedCwnumedit1(VARIANT* Value, VARIANT* PreviousValue, BOOL
OutOfRange);
public:
        NI::CNiButton m_on;     //Etat du bouton (1=appuyé)
public:
        void ClickCwboolean1();  //Gestionnaire du bouton
public:
        NI::CNiSlide m_slide2;    //Barre à droite. Contient la valeur parmis d'autres
public:                             //Gestionnaire de la barre à droite
        void PointerValueChangedCwslide2(long Pointer, VARIANT* Value);
```

Fig. A6.7 : Déclarations des Classes et des Gestionnaires

Pour se familiariser avec la programmation *Visual C++*, on a développé une autre application d'exemple qui utilise aussi quelques ressources de *Measurement Studio* et d'autres de *Visual Studio*. Dans cette application, on peut dessiner un signal de différents types et couleurs selon le choix qu'on fasse dans les menus personnalisés. On a créé aussi des butons et on ajouté des gestionnaires habituels et d'actualisation pour donner la fonctionnalité souhaitée. Dans ce cas, l'architecture choisie est Document-Vue.

On montre une vue de cette application dans la figure A6.8 :

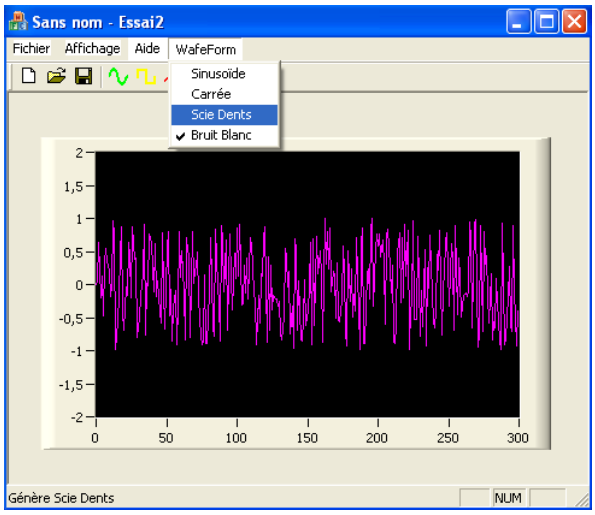

Fig. A6.8 : Application de Dessin des Signaux

Ces exemples sont basés en ceux qui sont proposés comme matériel didactique dans le rapport référencé en bibliographie [NUN07] ainsi que dans le site de National Instruments [TNI09].

ANNEXE A7 :
CODE DU LOGICIEL ENREGISTRER V2.4.

Cette annexe a par but celui de montrer et d'expliquer les principaux éléments du code pour l'essai des capteurs de distance du robot appelé *Enregistrer v2.4*. Ce logiciel a été largement utilisé pour rendre exploitables les mesures de ces capteurs et il est donc exposé dans les chapitre 2.

C'est ainsi que la figure 2.1 de ce chapitre montre l'interface usager du logiciel avec ses fonctionnalités et un peu plus bas, elles sont toutes décrites. Du coup, cette explication est ici omise et l'affichage et le commentaire des parties les plus importantes du code sont réalisés.

En rappelant les choix possibles pour la programmation d'un logiciel, il faut dire que celui-ci a été développé sur l'architecture Document-Vue. Tous les renseignements par rapport à cette architecture peuvent être trouvés dans le paragraphe 4.1 et 4.2.2.

Tout d'abord, on présente un tableau avec les gestionnaires associés aux éléments graphiques de l'interface d'usager qu'on peut voir à la figure 2.1. Ces gestionnaires font toujours partie du fichier de la *Vue* du code. De même, dans ce tableau sont écrites les variables de ces éléments, au cas où elles existeraient. Voici le tableau :

Elément de l'interface	Gestionnaire Associé	Variable Associée
Bouton Démarrer	ClickCwboolean1	m_indicateur
Nombre d'échantillons	---	m_nom_ech
Boutons pour le capteur affiché	Click1014-1018	boutonIR1-IR4, boutonUS
Paramètre Tau du filtre	ValueChangedCwnumedit2	m_tau
Affichage (sup/inf)	ValueChangedCwnumedit3/4	m_aff_debut/m_aff_fin
Prélèvement (Statistique)	OnCbnSelchangeCombo1	SelPrelevement
Résultats Statistiques	---	m_moyenne, m_sigma, m_sigmac
Echantillon d'analyse	ValueChangedCwnumedit5	m_obt_mes
Résultats pour l'échantillon	---	m_valeur_se, m_valeur_e
Coefficients du polynôme (V^4-TI)	ValueChanged1019-1023	puissance4-0
Enregistrer Matrice de Données	ClickCwboolean7	---
Graph. des signaux sans étalonnage	---	m_graph1
Graph. des signaux étalonnés	---	m_graph2

Fig. A7.1 : Tableau des Gestionnaires et Variables de *Enregistrer v2.4*

Les autres variables ou fonctions, non associées aux éléments de l'interface usager, sont montrées à la figure A7.2, faisant partie de la définition de tous ces composants commentés (fichier d'en tête *EnregistrerView.h*) :

```
public:                          //Fonctions
        void dessiner_essai(void);                  //Fonction de dessin des graphiques
        void statistique(void);                     //Fonction des calculs statistiques
public:                          //Variables Auxiliaires
        CNiReal64Vector temp1, temp2, temp3, temp4, temp5;      //Vecteurs de mesures
        CNiReal64Vector plot1, plot2, plot3, plot4, calculs;    //Vecteurs de dessin et calculs
        int i, capteur, ech_affichage;              //Compteur, nombre d'éch. et capteur à afficher
        double alfa, moyenne, sigmac;               //Constante du filtre et paramètres statistiques
        unsigned int a,b;                           //Dimensions de la matrice de données
                                                    //a et b sont aussi des variables auxiliaires
```

Fig. A7.2 : Variables et Fonctions de *Enregistrer v2.4* non Associés à l'Interface

Il y a encore une dernière variable à présenter, c'est celle qui enregistre toutes les données de l'essai, appelée *m_donnees*, du type CNiReal64Matrix et dans le fichier *VariablesGlobales.h*.

Voyons le premier gestionnaire commenté :

```
void CEnregistrerView::ClickCwboolean1()          //Gestionnaire du bouton démarrer
{
        if(m_indicateur.Value==1)                                   //Déclenchement d'une mesure
        {
                if(((int)m_nom_ech.Value)==0)     //Si on n'a pas choisi le nombre d'échantillons
                {                                 //on accroche un message d'information
                        AfxMessageBox("Introduisez les Paramètres de l'Essai");
                        m_indicateur.Value=0;
                }
                else                              //Sinon
                {
                        SetTimer(1,100,NULL);     //On démarre le Timer avec une période de 100ms
                        m_donnees.SetSize(8,(int) m_nom_ech.Value);
                                                  //On fixe les dimensions de la matrice de données
                        i=0;                      //On initialise le compteur i
                }
        }
        else
                m_indicateur.Value=1;             //Empêche le pas à cero par l'utilisateur
}
```

Fig. A7.3 : Gestionnaire du Bouton Démarrer

Notons qu'il existe un mécanisme pour déclencher un essai et éviter qu'il soit arrêté pour l'usager. De même, si le nombre d'échantillons n'a pas été fixé on accroche un message qui rappelle qu'il doit être introduit. Enfin, si tout va bien, on démarre un temporisateur qui lance la fonction *OnTimer* chaque 100 millisecondes, on fixe la dimension de la matrice d'enregistrement à 9 lignes (temps d'échantillon, 5 capteurs plus 3 signaux traités) avec autant de colonnes que le nombre d'échantillons et on initialise le compteur d'échantillons.

On montre maintenant la fonction *OnTimer*, l'une des plus importantes de ce code, à la figure A7.4 :

```
void CEnregistrerView::OnTimer(UINT_PTR nIDEvent)      //Gestionnaire du Timer
{
        //Remplit la matrice de données
        if(((int)m_nom_ech.Value) > i)                 //Tandis que le compteur n'atteint le nombre d'éch.
        {
                m_donnees(0,i)=100*i;                  //On enregistre le temps d'échantillon
                CIR1_Code IR1;                         //On déclare l'objet IR1 de la classe CIR1_Code
                IR1.GetData(temp1);                    //On obtient la mesure et on enregistre sur temp1
                m_donnees(1,i)=temp1[0];               //On enregistre la mesure sur la matrice de donnees

                CIR2_Code IR2;                         //On répète pour tous les capteurs
                IR2.GetData(temp2);
                m_donnees(2,i)=temp2[0];

                CIR3_Code IR3;
                IR3.GetData(temp3);
                m_donnees(3,i)=temp3[0];

                CIR4_Code IR4;
                IR4.GetData(temp4);
                m_donnees(4,i)=temp4[0];
```

```
                CUS_Code US;
                temp5[0]=US.GetData();
                m_donnees(5,i)=temp5[0]*1000;

                if(i==0)                                //On filtre le signal et on enregistre
                        m_donnees(6,i)=m_donnees(capteur,i);
                else
                        m_donnees(6,i)=m_donnees(capteur,i)*(1-alfa)+(m_donnees(6,(i-1)))*alfa;

                                                        //On étalonne le signal et on enregistre
                m_donnees(7,i)=(puissance4.Value)*(pow(m_donnees(capteur,i),4)) +
(puissance3.Value)*(pow(m_donnees(capteur,i),3))
                        + (puissance2.Value)*(pow(m_donnees(capteur,i),2)) +
(puissance1.Value)*(m_donnees(capteur,i)) + (puissance0.Value);

                                                        //On étalonne le signal filtré et on enregistre
                m_donnees(8,i)=(puissance4.Value)*(pow(m_donnees(6,i),4)) +
(puissance3.Value)*(pow(m_donnees(6,i),3))
                        + (puissance2.Value)*(pow(m_donnees(6,i),2)) +
(puissance1.Value)*(m_donnees(6,i)) + (puissance0.Value);

                i++;                                    //On accroît le compteur
        }
        else    //Une fois qu'on a atteint le nombre d'échantillons choisi
        {
                m_indicateur.Value=0;   //On éteint le bouton
                KillTimer(1);           //On arrête le Timer
                m_aff_debut.Value=0;    //On choisit le début d'affichage par l'éch.0
                if(((int)m_aff_fin.Value)==((int)m_nom_ech.Value-1))
                        dessiner_essai();  //Répète l'essai dans les mêmes conditions
                m_aff_fin.Value=((int)m_nom_ech.Value-1); //On finit par le nombre d'ech.choisi-1
        }

        CFormView::OnTimer(nIDEvent);
}
```

Fig. A7.4 : Fonction OnTimer de Enregistrer v2.4

Cette fonction vérifie toujours à travers du compteur *i* si on a atteint le nombre d'échantillons fixé. Si ce n'est pas le cas, on remplit la matrice des données échantillon par échantillon : temps d'échantillon, IR1, IR2, IR3, IR4, US, signal du capteur sélectionné filtré, étalonné et filtré et étalonné. Si on a atteint le nombre d'échantillons, on arrête le temporisateur, on lâche le bouton de démarrage, on dessine l'essai (à travers de la fonction *dessiner_essai*) et on fixe la valeur de la boîte de la fin d'affichage au dernier échantillon pris.

Il n'est pas nécessaire de montrer le gestionnaire du paramètre τ du filtre (*ValueChangedCwnumedit2*) puisqu'il est très simple. Il ne fait qu'appeler la fonction *dessiner_essai* en redessinant les graphiques chaque fois que le filtre change. Ainsi, on peut noter les effets du filtre instantanément.

Les gestionnaires des boîtes qui gèrent l'affichage à travers de la sélection de l'échantillon de début et de fin, appelés *ValueChangedCwnumedit3* et *ValueChangedCwnumedit4*, sont aussi assez simples. Ils vérifient que les échantillons soient cohérents et affichent un message si ce n'est pas le cas. Par contre, si tout va bien, ils dessinent les nouveaux graphiques selon les limites souhaitées.

Comme pour les cas antérieurs, il est facile de comprendre le gestionnaire *ValueChangedCwnumedit5*, c'est-à-dire, celui qui permet de choisir un échantillon pour l'analyser. D'abord, il vérifie que l'échantillon est dans les limites de l'affichage. Ensuite, il fait une simple assignation des valeurs (filtrées sans étalonner et filtrées étalonnées) de l'échantillon souhaité de la matrice de données aux variables, qui leur affichent dans l'interface.

D'ailleurs, il est intéressant d'étudier le code du gestionnaire qui choisit le prélèvement statistique et appelé *OnCbnSelchangeCombo1* :

```
void CEnregistrerView::OnCbnSelchangeCombo1()          // Gestionnaire du selecteur du prélèvement
{
        //Obtention de l'indicateur de la chaîne choisie à travers du sélecteur
        i = SelPrelevement.GetCurSel();

        //Si on a sélectionné un choix dans le sélecteur i != 0
        if(i != -1)
        {
                //Si on n'a pas choisi le nombre d'échantillons
                //on suppose qu'on n'a fait aucun essai
                if(((int) m_nom_ech.Value)==0)
                        AfxMessageBox("Faites un essai d'abord");
                else
                {
                        SelPrelevement.GetLBText(i, Prelevement);

                        //Obtention de la ligne à analyser dans la matrice de données
                        if((Prelevement.CompareNoCase("Signal Pur"))==0)
                                calculs=plot1;     //Signal non traité
                        if((Prelevement.CompareNoCase("Signal Filtré"))==0)
                                calculs=plot2;     //Signal filtré
                        if((Prelevement.CompareNoCase("Signal Etalonné"))==0)
                                calculs=plot3;     //Signal étalonné
                        if((Prelevement.CompareNoCase("S. Filtré et Etalonné"))==0)
                                calculs=plot4;     //Signal filtré et étalonné

                        statistique();       //Appelle la fonctions des calculs statistiques
                }
        }
}
```

Fig. A7.5 : Gestionnaire du Sélecteur de Prélèvement

Les premières instructions du code de la figure A7.5 obtiennent l'indicateur de la chaîne pour l'obtenir un peu plus bas. Ensuite, on vérifie que cet indicateur a pris une valeur valable et qu'on a fait un essai auparavant. Puis, on obtient la chaîne sélectionné dans la variable *Prelevement* et elle est comparée pour assigner le vecteur nécessaire au vecteur pour le calcul statistique. Enfin, on appelle la fonction du calcul statistique pour réaliser les opérations nécessaires sur le vecteur *calcul*.

Les gestionnaires des boutons pour la sélection du capteur à afficher sont complètement pareils, donc on ne montre que celui du bouton IR1, c'est-à-dire, celui qui est appelé *Click1014* :

```
void CEnregistrerView::Click1014()                    // Gestionnaire du bouton de sélection du capteur IR1
{
        if(boutonIR1.Value==1)   //Si le bouton est cliqué par la première fois
        {
                //Les boutons des autres capteurs ne sont pas allumés
                boutonIR2.Value=0;
                boutonIR3.Value=0;
                boutonIR4.Value=0;
                boutonUS.Value=0;

                //Sélection de la ligne de la matrice de
                //données avec laquelle on va travailler
                capteur=1;

                //Si on n'a pas choisi le nombre d'échantillons
                //on suppose qu'on n'a fait aucun essai
                if(((int) m_nom_ech.Value)!=0)
                        dessiner_essai();
        }
        else
                boutonIR1.Value=1;
}
```

Fig. A7.6 : Gestionnaire du Bouton IR1

Ces gestionnaires vérifient que le bouton a été appuyé et pas lâché. Ensuite, ils lâchent tous les autres boutons sauf celui qui a été appuyé. Puis, on assigne à la variable *capteur* le numéro correspondant au capteur sélectionné (IR1=1, IR2=2, IR3=3, IR4=4 et US=5). Après avoir vérifié qu'un essai a déjà été réalisé, on redessine les graphiques avec les signaux du nouveau capteur.

Les gestionnaires des coefficients du polynôme d'étalonnage sont encore très simples. Ils ne font que vérifier qu'un essai a été réalisé et puis dessinent les signaux à nouveau en appelant la fonction *dessiner_essai* comme d'habitude.

Voyons le dernier gestionnaire à commenter, celui du bouton qui actualise la matrice de données, c'est-à-dire, le ClickCwboolean7 :

```
void CEnregistrerView::ClickCwboolean7()     // Gestionnaire du bouton Enregistrer Matrice de Données
{
        for(i=0;((int)m_nom_ech.Value) > i;i++)        //Tandis que le compteur n'atteint le nombre d'éch.
        {
                if(i==0)                                        //On filtre le signal et on enregistre
                        m_donnees(6,i)=m_donnees(capteur,i);
                else
                        m_donnees(6,i)=m_donnees(capteur,i)*(1-alfa)+(m_donnees(6,(i-1)))*alfa;
                                                        //On étalonne le signal et on enregistre
                m_donnees(7,i)=(puissance4.Value)*(pow(m_donnees(capteur,i),4)) +
(puissance3.Value)*(pow(m_donnees(capteur,i),3)) + (puissance2.Value)*(pow(m_donnees(capteur,i),2))
+ (puissance1.Value)*(m_donnees(capteur,i)) + (puissance0.Value);
                                                        //On étalonne le signal filtré et on enregistre
                m_donnees(8,i)=(puissance4.Value)*(pow(m_donnees(6,i),4)) +
(puissance3.Value)*(pow(m_donnees(6,i),3)) + (puissance2.Value)*(pow(m_donnees(6,i),2)) +
(puissance1.Value)*(m_donnees(6,i)) + (puissance0.Value);
        }
}
```

Fig. A7.7 : Gestionnaire du Bouton d'Actualisation de la Matrice de Données

Ce code actualise les lignes 6, 7 et 8 de la matrice de données avec les derniers choix réalisés (tau, capteur affiché, etc.), ce qui correspond aux signaux traités par ce logiciel. Par contre, il laisse invariables les mesures non traitées des capteurs. Ensuite, cette matrice dynamique peut être enregistrée dans un fichier tel qu'elle soit.

La fonction *dessiner_essai* réalise les opérations d'affichage graphique du logiciel. Voici son code :

```
void CEnregistrerView::dessiner_essai(void)                    // Fonction de dessin des signaux et de calculs
{
        //Initialise quelques paramètres
        if(((int) m_nom_ech.Value)==0)    //Obtient le nombre d'échatillons au cas où on ne
        {                                 //ne l'aurait pas (lorsqu'on ouvre un fichier)
                m_donnees.GetSize(a,b);
                m_nom_ech.Value=(int) b;
        }

        //Le nombre d'éch.qu'on va afficher est égal à la différence entre les valeurs
        //introduites plus un. Ex: Afficher de l'éch.5 à l'éch.6-->6-5+1=2 échantillons
        ech_affichage=(int) (m_aff_fin.Value-m_aff_debut.Value+1);

        //Fixe la dimension des vecteurs de dessin et calcul
        plot1.SetSize(ech_affichage);
        plot2.SetSize(ech_affichage);
        plot3.SetSize(ech_affichage);
        plot4.SetSize(ech_affichage);
        calculs.SetSize(ech_affichage);

        //Calcule le paramètre alfa pour appliquer le filtre de premier ordre
        alfa=((double) exp((double) (-0.1/m_tau.Value)));

        //Boucle de calcul des vecteurs de dessin
        for(i=0;ech_affichage > i;i++)                  //Jusqu'à dépasser les éch. qu'on va afficher
        {                       //Remplit les vecteurs selon l'affichage choisi
                                //et à partir de la matrice de donnees
                plot1[i]=m_donnees(capteur,i+((int)m_aff_debut.Value));

                if(i==0)                                                        //Signal filtré
                        plot2[i]=m_donnees(capteur,i+((int)m_aff_debut.Value));
                else
                        plot2[i]=m_donnees(capteur,i+((int)m_aff_debut.Value))*(1-alfa)+(plot2[i-
1])*alfa;
                                                                                //Signal étalonné
                plot3[i]=(puissance4.Value)*(pow(plot1[i],4)) + (puissance3.Value)*(pow(plot1[i],3))
                        + (puissance2.Value)*(pow(plot1[i],2)) + (puissance1.Value)*(plot1[i]) +
(puissance0.Value);
                                                                                //Signal filtré et étalonné
                plot4[i]=(puissance4.Value)*(pow(plot2[i],4)) + (puissance3.Value)*(pow(plot2[i],3))
                        + (puissance2.Value)*(pow(plot2[i],2)) + (puissance1.Value)*(plot2[i]) +
(puissance0.Value);
        }

        //Actualisation des valeurs statistiques
        OnCbnSelchangeCombo1();

        //Contrôle des boîtes qui montrent les valeurs d'un éch. lors d'un nouvel affichage
        m_obt_mes.Value=m_aff_debut.Value;
        m_valeur_se.Value=plot2[((int) m_obt_mes.Value)-((int)m_aff_debut.Value)];
        m_valeur_e.Value=plot4[((int) m_obt_mes.Value)-((int)m_aff_debut.Value)];
```

```
        //Début d'axe X des graphiques selon l'éch. de début choisi
        m_graph1.Plots.Item(1).DefaultXFirst=((double) m_aff_debut.Value);
        m_graph1.Plots.Item(2).DefaultXFirst=((double) m_aff_debut.Value);
        m_graph2.Plots.Item(1).DefaultXFirst=((double) m_aff_debut.Value);
        m_graph2.Plots.Item(2).DefaultXFirst=((double) m_aff_debut.Value);

        //Dessin des vecteurs construits
        m_graph1.Plots.Item(1).PlotY(plot1);
        m_graph1.Plots.Item(2).PlotY(plot2);
        m_graph2.Plots.Item(1).PlotY(plot3);
        m_graph2.Plots.Item(2).PlotY(plot4);
}
```

Fig. A7.8 : Fonction d'Affichage Graphique

Les premiers lignes du code sont destinées à obtenir le nombre d'échantillons au cas où on ne le saurait pas, c'est le cas de l'ouverture récente d'un fichier de données. Ensuite, on calcule le nombre d'échantillons qui seront affichés pour fixer la taille des vecteurs de dessin et de calcul statistique et pour contrôler une boucle un peu plus bas. Puis, on calcule la constante alfa pour le filtre numérique de premier ordre.

A ce moment-là, une boucle de type *for* où l'on rempli l'un pour un tous les vecteurs de dessin selon le capteur choisi : *plot1* (signal pur), *plot2* (signal filtré), *plot3* (signal étalonné) et *plot4* (signal filtré et étalonné). Ensuite, on appelle le gestionnaire *OnCbnSelchangeCombo1()* qui actualise toutes les valeurs statistiques et on actualise les boîtes qui concernent à l'échantillon d'analyse.

Enfin, on peut trouver plusieurs instructions uniquement graphiques ; d'abord pour adapter l'axe *X* des graphiques au nouvel affichage et ensuite pour représenter les vecteurs remplis auparavant (de *plot1* à *plot4*).

La dernière fonction à commenter est la fonction de calcul statistique appelée *statistique*. Ce n'est pas nécessaire d'expliquer en profondeur cette fonction puisque son code ne fait que de calculs simples de statistique. Il suffit de dire que le but de cette fonction est, bien évidemment, de calculer la moyenne, l'écart typique et la variance et de les assigner aux boîtes correspondantes.

ANNEXE A8 :
CODE DU LOGICIEL ROBOT_HAMMI V2.2.

Dans cette annexe on présentera les principales modifications et les fonctions les plus importantes ajoutées au code du logiciel *Robot_HAMMI v1.0* pour obtenir la version actuelle, décrite dans le paragraphe 4.4. De même, dans la figure 4.18, on a montré un tableau qui résume ces fichiers modifiés et ajoutés qui seront expliqués un peu plus bas.

Commençons avec les modifications accomplies dans le fichier *Robot_HAMMIView.cpp*. Les changements du code les plus importants de ce fichier sont dans le gestionnaire du temporisateur appelé *OnTimer* :

```
//************************************************//
//************************************************//
//                FICHIER MODIFIE PAR             //
//************************************************//
//                Javier SÁEZ CARDADOR            //
//                      Juin 2009                 //
//                  E.N.S.A.M. Paris              //
//************************************************//
//************************************************//

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////  Le fichier a été modifié pour obtenir et traiter numériquement les signaux des capteurs  /////
/////  de distance. De même, grâce à une estimation géométrique des distances, il calcule les  /////
/////  erreurs des mesures comises. Enfin, il est capable de représenter et d'enregistrer les  /////
/////  mesures des capteurs, leurs estimations et leurs erreurs.                               /////
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

// Robot_HAMMIView.cpp : implémentation de la classe CRobot_HAMMIView

[…]

//Fonction déclenchée par le signal Timer à chaque période d'echantillonnage
//Modifié par Javier Saez Cardador en Juin 2009
void CRobot_HAMMIView::OnTimer(UINT_PTR nIDEvent)
{

[…]

        //////// Lecture des capteurs
        CCapteurs_IR_Code mes_IR;   // Création d'une instance de la classe CCapteurs_IR_Code
        mes_IR.GetData(IR);                 // Transfert des valeurs en volts délivré par les IR
        CCapteur_US_Code mes_US;    // Création d'une instance de la classe CCapteur_US_Code
        US = mes_US.GetData()*1000;     // Transfert du signal en millisecondes délivré par l'US

        //////// Début de la boucle de calcul des consignes moteurs
        if(ech<=nb_echantillons) // Test de continuité des calculs en fonction du nombre d'échantillons
voulus
        {
                if (ech==1) //la valeur initiale de ech est 1; on ne sait pas ce qu'il reste dans les
variables pour le premier calcul
                {
                        //initialisation de toutes les variables
                        Omega_1=0;      //pour eviter de lire n'importe quelle donnée qui restait sur la
carte
```

```
                    Omega_2=0;      //pour eviter de lire n'importe quelle donnée qui restait sur la carte
                    m_donnees.SetSize(nb_variables+1,nb_echantillons+1);        // Initialisation de la taille des matrices de stockage des données
                    TR_Capteurs.SetSize(5);  //Initialisation de la taille des vecteurs de calcul des nouvelles données
                    ES_Capteurs.SetSize(5);
                    ER_Capteurs.SetSize(5);
                }

[…]

                //Fonction de traitement des données des capteurs
                traitement_donnees_capteurs();

                //Vérification d'obstacles avec les données récemment obtenues (SAA)
                if (TR_Capteurs[0] < distance_arret_urgence)
                {
                    OnBnClickedArreter();
                    AfxMessageBox("Capteur IR1: obstacle à proximité");
                }
                if (TR_Capteurs[1] < distance_arret_urgence)
                {
                    OnBnClickedArreter();
                    AfxMessageBox("Capteur IR2: obstacle à proximité");
                }
                if (TR_Capteurs[2] < distance_arret_urgence)
                {
                    OnBnClickedArreter();
                    AfxMessageBox("Capteur IR3: obstacle à proximité");
                }
                if (TR_Capteurs[3] < distance_arret_urgence)
                {
                    OnBnClickedArreter();
                    AfxMessageBox("Capteur IR4: obstacle à proximité");
                }
                if (TR_Capteurs[4] < distance_arret_urgence)
                {
                    OnBnClickedArreter();
                    AfxMessageBox("Capteur US: obstacle à proximité");
                }

                //Fonction d'estimation des mesures des capteurs
                calcul_estimation_capteurs(xr, yr, thetar, xmur1, xmur2, ymur1, ymur2);

                //Fonction de calcul de l'erreur entre mesure réelle et estimation
                calcul_erreur_capteurs();

                // On remplit la matrice m_donnees avec les donnees que l'on veut analyser
                m_donnees(0,ech)=ech*Tech;      //Temps des pas de calcul
                m_donnees(1,ech)=Consigne_Mot_1;
                m_donnees(2,ech)=Consigne_Mot_2;
                m_donnees(3,ech)=Omega_1;
                m_donnees(4,ech)=Omega_2;
                m_donnees(5,ech)=xr;
                m_donnees(6,ech)=yr;
                m_donnees(7,ech)=thetar;
                m_donnees(8,ech)=theta11;
                m_donnees(9,ech)=theta12;
                m_donnees(10,ech)=xd;
```

```
                m_donnees(11,ech)=yd;
                m_donnees(12,ech)=ex;
                m_donnees(13,ech)=ey;
                m_donnees(14,ech)=IR[0];          //Mesures brutes des capteurs de distance
                m_donnees(15,ech)=IR[1];
                m_donnees(16,ech)=IR[2];
                m_donnees(17,ech)=IR[3];
                m_donnees(18,ech)=US;
                m_donnees(19,ech)=TR_Capteurs[0];       //Mesures traitées des capteurs de distance
                m_donnees(20,ech)=TR_Capteurs[1];
                m_donnees(21,ech)=TR_Capteurs[2];
                m_donnees(22,ech)=TR_Capteurs[3];
                m_donnees(23,ech)=TR_Capteurs[4];
                m_donnees(24,ech)=ES_Capteurs[0];       //Estimation de la distance mesurée
                m_donnees(25,ech)=ES_Capteurs[1];
                m_donnees(26,ech)=ES_Capteurs[2];
                m_donnees(27,ech)=ES_Capteurs[3];
                m_donnees(28,ech)=ES_Capteurs[4];
                m_donnees(29,ech)=ER_Capteurs[0];       //Erreur entre la mesure et l'estimation
                m_donnees(30,ech)=ER_Capteurs[1];
                m_donnees(31,ech)=ER_Capteurs[2];
                m_donnees(32,ech)=ER_Capteurs[3];
                m_donnees(33,ech)=ER_Capteurs[4];
                m_donnees(34,ech)=xmur1;          //Paramètres de l'environnement
                m_donnees(35,ech)=xmur2;
                m_donnees(36,ech)=ymur1;
                m_donnees(37,ech)=ymur2;
                ech++; // Incrémentation du nombre d'échantillons traités
        }       // fin de la boucle

        // Arrêt de l'horloge et réinitialisation une fois le nombre d'échantillons atteint
        else
        {
                // Appel de la fonction de réinitialisation
                OnBnClickedArreter();
        }       // Fin du else

        CFormView::OnTimer(nIDEvent); // Référence à la fonction modèle
}
```

Fig. A8.1 : Gestionnaire OnTimer

D'abord, dans la figure A8.1, notons qu'un cadre de présentation a été mis en tête de ce fichier pour identifier les fichiers modifiés. Ensuite, il faut faire attention aux marqueurs *[…]*, qui signifient qu'un morceau de code non modifié a été enlevé. Remarquons aussi la nouvelle matrice de données (*m_donnees*) avec plus de paramètres stockés, le système d'arrêt automatique (SAA) ou les appelles aux nouvelles fonctions.

Un autre gestionnaire, très modifié pour avoir un arrêt du robot progressif, est celui appelé *OnBnClickedArreter*, qui répond aussi à l'appuie de la touche appelée *ARRETER*. Voici la figure A8.2 :

```
// Fonction de réinitialisation déclenchée par l'appui sur Arrêter l'essai
//Modifié par Javier Saez Cardador en Juin 2009
void CRobot_HAMMIView::OnBnClickedArreter()
{
        int i;
        double a, b;
        a=v_moteur[0]/6;//Obtention des marches pour la réduction
        b=v_moteur[1]/6;        //des vitesses lors du freinage
        KillTimer(1);           // Arrêt du timer
```

```
        for (i=1;i<=6;i++)          // Mise à zéro des commandes
        {
                v_moteur[0]=v_moteur[0]-a;        //Réduction progressive des vitesses
                v_moteur[1]=v_moteur[1]-b;
                CCommandes_moteurs_Code envoi_consignes;       // Crée un objet de la classe
CCd_Mot_1_Code
                envoi_consignes.WriteData(v_moteur);      // Envoi des consignes des vitesses aux
moteurs
                Sleep(200);                  //Attend que la nouvelle vitesse soit acquise physiquement
        }
        ech = 1; // Réinitialisation du compteur d'échantillons
}
```

Fig. A8.2 : Gestionnaire OnBnClickedArreter

Pour finir avec le fichier *Robot_HAMMIView.cpp*, dans la figure A8.3 on présente le gestionnaire qui dessine les courbes des mesures, des estimations et des écarts des capteurs :

```
//Fonction déclenchée par appui sur le bouton Tracer mesure IR et US
//Modifié par Javier Saez Cardador en Juin 2009
void CRobot_HAMMIView::OnBnClickedTracerIr()
{
        int i,nb_courbes=1;          //Déclaration et initialisation des variables auxiliaires
        CNiReal64Vector v_graphe_mesures; //Déclaration des vecteurs remplis pour la représentation
        CNiReal64Vector v_graphe_estimations;
        CNiReal64Vector v_graphe_erreurs;
        v_graphe_mesures.SetSize(nb_echantillons);  //Initialisation des vecteurs
        v_graphe_estimations.SetSize(nb_echantillons);
        v_graphe_erreurs.SetSize(nb_echantillons);
        UpdateData(true);
        graphe_IR.ClearData();    //Efface les graphiques
        graphe_estimation.ClearData();
        graphe_erreur.ClearData();
        UpdateData(false);

        if(IR1_Check==TRUE)    //Vérifie s'il faut designer les données du capteur IR1
        {
                for(i=1;i<=nb_echantillons;i++)  //Boucle pour remplir les vecteurs de dessin
                {
                        v_graphe_mesures[i-1]=m_donnees(19,i);
                        v_graphe_estimations[i-1]=m_donnees(24,i);
                        v_graphe_erreurs[i-1]=m_donnees(29,i);
                }
                graphe_IR.Plots.Item(nb_courbes).PlotY(v_graphe_mesures);
        //Assignation des vecteurs aux graphes
        graphe_IR.Plots.Item(nb_courbes).LineColor=CNiColor(255,0,0);      //Couleur de dessin
        graphe_estimation.Plots.Item(nb_courbes).PlotY(v_graphe_estimations);
                graphe_estimation.Plots.Item(nb_courbes).LineColor=CNiColor(255,0,0);
                graphe_erreur.Plots.Item(nb_courbes).PlotY(v_graphe_erreurs);
                graphe_erreur.Plots.Item(nb_courbes).LineColor=CNiColor(255,0,0);
                nb_courbes++;                //Incrément le nombre des courbes à dessiner
                graphe_IR.Plots.Add();     //Trace les courbes sur le graphiques des mesures réelles
                graphe_estimation.Plots.Add();  //Dessine le graphique des estimations
                graphe_erreur.Plots.Add(); //Dessine le graphique des erreurs
        }

[…]     //La méthode est pareille pour les autres capteurs

}
```

Fig. A8.3 : Gestionnaire OnBnClickedTracerIr

A part des modifications sur ce fichier, il y a trois fichiers ajoutés au code de l'année dernière. D'après la figure 4.18, ils sont : *Traitement_donnees_capteurs.cpp*, *Calcul_estimation_capteurs.cpp* et *Calcul_erreur_capteurs.cpp*. Le dernier de ces fichiers contient les opérations pour obtenir la valeur absolue de la soustraction de la mesure réelle et de celle estimée. Il dégage la fonction *OnTimer* mais il est tellement simple qu'il ne s'avère pas nécessaire de le commenter.

Voyons les deux autres fichiers :

```
//************************************************//
//************************************************//
//                  Javier SÁEZ CARDADOR          //
//                       Juin 2009                //
//                   E.N.S.A.M. Paris             //
//************************************************//
//************************************************//

//Traitement_donees_capteurs.cpp : implémentation de la fonction pour le traitement
//des données de distance.
//Appelée dans la vue

#include "stdafx.h"
#include "variables_globales.h"
#include "math.h"

void traitement_donnees_capteurs()
{
	//Filtrage numérique RII 1er ordre
	//Calcul du paramètre alfa à partir de tau
	double alfa;
	alfa=((double) exp((double) (-0.1/tau)));
	//Initialisation du filtre dans le premier échantillon
	if(ech==1)
	{
		TR_Capteurs[0]=IR[0];
		[…]   //Pareil pour les autres capteurs
	}
	//Calcul pour le filtre le reste des échantillons
	else
	{
		TR_Capteurs[0]=IR[0]*(1-alfa)+(m_donnees(14,ech-1))*alfa;
		[…]   //Pareil pour les autres capteurs
	}

   //Etalonnage selon les polynômes calculés [Rapport Paragraphe 2.2.2.2, page 21]
   TR_Capteurs[0]= 13.6409*(pow(TR_Capteurs[0],4)) - 104.6611*(pow(TR_Capteurs[0],3))
	+ 300.0149*(pow(TR_Capteurs[0],2)) - 402.2714*(TR_Capteurs[0]) + 254.2031 + corIR1;
   […]   //Pareil pour les autres capteurs (Chacun avec son polynôme d'étalonnage)

   //Délimitation de la perception des capteurs
  if (TR_Capteurs[0] < 20)
     TR_Capteurs[0] = 0;
  else
	{
		if (TR_Capteurs[0] > 120)
			TR_Capteurs[0] = 120;
	}
   […]   //Pareil pour les autres capteurs (Le capteur à ultrasons jusqu'à 600 cm)
}
```

Fig. A8.4 : Fonction Traitement_donnees_capteurs

```
//*************************************************//
//*************************************************//
//                  Javier SÁEZ CARDADOR           //
//                       Juin 2009                 //
//                  E.N.S.A.M. Paris               //
//*************************************************//
//*************************************************//

//Calcul_estimation_capteurs.cpp : implémentation de la fonction pour l'estimation
//des distances mesurées.
//Appelée dans la vue

#include "stdafx.h"
#include "variables_globales.h"
#include "math.h"

void calcul_estimation_capteurs(double xr,      double yr,      double thetar,
                                double xmur1, double xmur2, double ymur1, double ymur2)
{
        //Paramètres liés au modèle des capteurs (Tableau figure 1.3)
        //Angles en rad
        double gammaIR1 = (50*pi)/180;
        double gammaIR2 = (72*pi)/180;
        double gammaIR3 = (110*pi)/180;
        double gammaIR4 = (135*pi)/180;
        double gammaUS = (90*pi)/180;
        //Distances en mètres
        double lyIR14 = 0.329;
        double lyIR23 = 0.355;
        double lyUS = 0.363;
        double lxIR1 = 0.091;
        double lxIR2 = 0.044;
        double lxIR3 = -0.042;
        double lxIR4 = -0.090;

        //Variables et constantes auxiliaires
        double thetar_corrigee;
        double vector_equations[5];
        double dxyIR1[2], dxyIR2[2], dxyIR3[2], dxyIR4[2], dxyUS[2];

  //Réduction de l'angle thetar au rang (0,+/-360)
        thetar_corrigee=thetar;

  while ((thetar_corrigee > 2*pi) || (thetar_corrigee < -2*pi))
                thetar_corrigee = thetar_corrigee - 2*pi;

  //Identification des equations a utiliser en fonction de l'angel thetar_corrigee
  //IR1
  //xmur1,ymur2
  if (((thetar_corrigee > gammaIR1) && (thetar_corrigee <= (gammaIR1+pi/2))) || ((thetar_corrigee >
     (gammaIR1-2*pi)) && (thetar_corrigee <= (gammaIR1-3/2*pi))))
        vector_equations[0] = 1;
  else
  {
         //xmur1,ymur1
        if (((thetar_corrigee > (gammaIR1+pi/2)) && (thetar_corrigee <= (gammaIR1+pi))) ||
           ((thetar_corrigee > (gammaIR1-3/2*pi)) && (thetar_corrigee <= (gammaIR1-pi))))
              vector_equations[0] = 2;
        else
        {
```

```
                    //xmur2,ymur1
                    if (((thetar_corrigee > (gammaIR1+pi)) && (thetar_corrigee <= (gammaIR1+3/2*pi)))
                        || ((thetar_corrigee > (gammaIR1-pi)) && (thetar_corrigee <= (gammaIR1-pi/2))))
                            vector_equations[0] = 3;
                    else
                    {
                        //xmur2,ymur2
                        if (((thetar_corrigee > (gammaIR1+3/2*pi)) || (thetar_corrigee <= gammaIR1)) &&
                           ((thetar_corrigee > (gammaIR1-pi/2)) || (thetar_corrigee <= (gammaIR1-2*pi))))
                                vector_equations[0] = 4;
                    }
            }
    }

    […]   //Pareil pour les autres capteurs (Chacun avec ses intervalles)

    //Application des equations selon les résultats anterieurs
    //IR1
    if (vector_equations[0] == 1 || vector_equations[0] == 2)    //Le rayon est dirigé vers le mur xmur1
        dxyIR1[0] = abs((xr - xmur1 + lyIR14*sin(pi/2-thetar_corrigee) - lxIR1*cos(pi/2-thetar_corrigee)) /
                        (cos(gammaIR1 + (pi/2-thetar_corrigee))));
    else    //Le rayon est dirigé vers le mur xmur2
        dxyIR1[0] = abs((xmur2 - xr + lyIR14*sin(thetar_corrigee-pi/2) + lxIR1*cos(thetar_corrigee-pi/2))
                         /(cos((pi/2+thetar_corrigee) - gammaIR1)));
    if (vector_equations[0] == 1 || vector_equations[0] == 4)    //Le rayon est dirigé vers le mur ymur2
        dxyIR1[1] = abs((ymur2 - yr - lyIR14*cos(thetar_corrigee-pi/2) + lxIR1*sin(thetar_corrigee-pi/2)) /
                        (cos(gammaIR1 - thetar_corrigee)));
    else    //Le rayon est dirigé vers le mur ymur1
        dxyIR1[1] = abs((yr - ymur1 + lyIR14*cos(thetar_corrigee-pi/2) - lxIR1*sin(thetar_corrigee-pi/2)) /
                        (cos(gammaIR1 - (thetar_corrigee-pi))));
    ES_Capteurs[0]=min(dxyIR1[0],dxyIR1[1]);

    […]   //Pareil pour les autres capteurs (Chacun avec ses formules)

    //Modélisation des limites des capteurs
    if (ES_Capteurs[0] < 0.2)
        ES_Capteurs[0] = 0;
    else
    {
            if (ES_Capteurs[0] > 1.2)
                    ES_Capteurs[0] = 1.2;
    }

    […]   //Pareil pour les autres capteurs (Le capteur à ultrasons jusqu'à 6 m)

    //Conversion à centimètres pour le calcul de l'erreur
    ES_Capteurs[0]=ES_Capteurs[0]*100;
    […]   //Pareil pour les autres capteurs
}
```

Fig. A8.5 : Fonction Calcul_estimation_capteurs

Notons que cette dernière fonction n'est qu'une traduction du modèle qu'on a conçu sur *Matlab* au langage *Visual C++*. C'est justement ce code ce qui est testé dans la figure 4.20 à l'aide du code présenté dans l'annexe A9.

Finalement, il faut dire que l'interface d'usager a été très modifiée pour accueillir des nouveaux graphiques et des boîtes pour introduire des paramètres. Un dessin explicatif a été montré dans la figure 4.17. Donc, concernant le fichier *Robot_HAMMI.rc*, la seule chose qu'il reste à montrer est la boîte de dialogue appelée *AboutBox* (A propos de) et l'icône de l'application ici inclut :

Fig. A8.6 : AboutBox de Robot_HAMMI v2.2

ANNEXE A9 :
CODE DU FICHIER COMPARER.M.

Le but de cette annexe est de présenter le code du fichier de *Matlab* appelé *comparer.m*. Ce fichier sert à vérifier la validité du logiciel *Robot_HAMMI v2.2* en représentant des graphiques plus conviviaux que celles de l'interface du logiciel.

Les données introduites dans ce fichier sortent d'un fichier *.txt* qui est créé par le logiciel Traducteur v1.0. En même temps, ce logiciel les reçoit de la matrice de données remplie lors de l'exécution d'une trajectoire, grâce à l'application *Robot_HAMMI v2.2*. Voici le code :

```
%//*************************************************************//
%//*************************************************************//
%//                    Javier SÁEZ CARDADOR                     //
%//                         Juin  2009                          //
%//                      E.N.S.A.M. Paris                       //
%//                                                             //
%//                        comparer.m                           //
%// Fichier pour vérifier l'estimation sur C++ et analyser les résultats:  //
%// Ce code prend une matrice générée par le logiciel Traducteur.exe avec  //
%//  les données enregistrées par le robot HAMMI lors de sa trajectoire.    //
%// Après avoir calculé l'écart entre le modèle implémenté sur matlab et   //
%//   celui implémenté sur C++ et entre les estimations et les mesures      //
%// réelles, il représente toutes les données touchées par ces écarts et    //
%//  dessine la trajectoire suivie par le robot dans son environnement.     //
%//*************************************************************//
%//*************************************************************//

function []=comparer(m_donnees)
%Obtention des données de l'essai du robot dans la matrice m_donees
%Structure des donnees dans m_donnees:
%Ligne 1: Instant de la prise des échantillons (pas de calcul) [s]
%Lignes 6, 7 et 8: Points Xr, Yr et Théta r respectivement [m] [rad]
%Lignes 20-24: Mesures réelles et traitées des capteurs IR1-IR4 et US [cm]
%Lignes 25-29: Estimations des distances des capteurs sur C++ [cm]
%Lignes 35-38: Données de l'environnement xmur1, xmur2, ymur1 et ymur2 [m]

%Nettoyage des fenêtres
close all

%Obtention de la matrice mm (matrice du modèle matlab) qui contient les
%estimations des capteurs IR1-IR4 et US à partir des points Ar du robot
mm=modele(m_donnees);

%Calcul des erreurs entre le modèle sur C++ et celui sur matlab
errm(1,:)=abs(m_donnees(25,:)-mm(1,:));
errm(2,:)=abs(m_donnees(26,:)-mm(2,:));
errm(3,:)=abs(m_donnees(27,:)-mm(3,:));
errm(4,:)=abs(m_donnees(28,:)-mm(4,:));
errm(5,:)=abs(m_donnees(29,:)-mm(5,:));

%Calcul des erreurs ente les estimations et les mesures réelles
erre(1,:)=abs(m_donnees(25,:)-m_donnees(20,:));
erre(2,:)=abs(m_donnees(26,:)-m_donnees(21,:));
erre(3,:)=abs(m_donnees(27,:)-m_donnees(22,:));
erre(4,:)=abs(m_donnees(28,:)-m_donnees(23,:));
erre(5,:)=abs(m_donnees(29,:)-m_donnees(24,:));
```

```
%Dessin pour la vérification du modèle sur C++
figure(1)
%Graphique de l'estimation à partir du modèle matlab
subplot(1,3,1)
hold on
plot(m_donnees(1,:),mm(1,:),'--k','LineWidth',2)
plot(m_donnees(1,:),mm(2,:),'-.b','LineWidth',2)
plot(m_donnees(1,:),mm(3,:),':m','LineWidth',2)
plot(m_donnees(1,:),mm(4,:),'r','LineWidth',2)
plot(m_donnees(1,:),mm(5,:),'k','LineWidth',2)
hold off
grid on
legend('dIR1','dIR2','dIR3','dIR4','dUS');
xlabel('temps [s]')
title('Estimation du Modèle Matlab: Distance [cm]')

%Graphique de l'estimation à partir du modèle sur C++
subplot(1,3,2)
hold on
plot(m_donnees(1,:),m_donnees(25,:),'--k','LineWidth',2)
plot(m_donnees(1,:),m_donnees(26,:),'-.b','LineWidth',2)
plot(m_donnees(1,:),m_donnees(27,:),':m','LineWidth',2)
plot(m_donnees(1,:),m_donnees(28,:),'r','LineWidth',2)
plot(m_donnees(1,:),m_donnees(29,:),'k','LineWidth',2)
hold off
grid on
legend('dIR1','dIR2','dIR3','dIR4','dUS');
xlabel('temps [s]')
title('Estimation du Modèle C++: Distance [cm]')

%Graphique de l'écart entre les deux estimations
subplot(1,3,3)
hold on
plot(m_donnees(1,:),errm(1,:),'--k','LineWidth',2)
plot(m_donnees(1,:),errm(2,:),'-.b','LineWidth',2)
plot(m_donnees(1,:),errm(3,:),':m','LineWidth',2)
plot(m_donnees(1,:),errm(4,:),'r','LineWidth',2)
plot(m_donnees(1,:),errm(5,:),'k','LineWidth',2)
axis([0 10 0 120]);
hold off
grid on
legend('Erreur IR1','Erreur IR2','Erreur IR3','Erreur IR4','Erreur US');
xlabel('temps [s]')
title('Ecart des Modèles [cm]')

%Dessins pour la comparaison entre les estimations et les mesures réelles
%(IR)
figure(2)
%Graphique de l'estimation à partir du modèle sur C++
subplot(1,3,1)
hold on
plot(m_donnees(1,:),m_donnees(25,:),'--k','LineWidth',2)
plot(m_donnees(1,:),m_donnees(26,:),'-.b','LineWidth',2)
plot(m_donnees(1,:),m_donnees(27,:),':m','LineWidth',2)
plot(m_donnees(1,:),m_donnees(28,:),'r','LineWidth',2)
hold off
grid on
legend('dIR1','dIR2','dIR3','dIR4');
xlabel('temps [s]')
title('Estimation du Modèle C++: Distance [cm]')
```

```
%Graphique des mesures réelles des capteurs
subplot(1,3,2)
hold on
plot(m_donnees(1,:),m_donnees(20,:),'--k','LineWidth',2)
plot(m_donnees(1,:),m_donnees(21,:),'-.b','LineWidth',2)
plot(m_donnees(1,:),m_donnees(22,:),':m','LineWidth',2)
plot(m_donnees(1,:),m_donnees(23,:),'r','LineWidth',2)
hold off
grid on
legend('dIR1','dIR2','dIR3','dIR4');
xlabel('temps [s]')
title('Mesures Réelles des Capteurs: Distance mesurée [cm]')

%Graphique de l'écart entre l'estimation et la réalité
subplot(1,3,3)
hold on
plot(m_donnees(1,:),erre(1,:),'--k','LineWidth',2)
plot(m_donnees(1,:),erre(2,:),'-.b','LineWidth',2)
plot(m_donnees(1,:),erre(3,:),':m','LineWidth',2)
plot(m_donnees(1,:),erre(4,:),'r','LineWidth',2)
hold off
grid on
legend('Erreur IR1','Erreur IR2','Erreur IR3','Erreur IR4');
xlabel('temps [s]')
title('Ecart Estimation-Réalité [cm]')

%Dessins pour la comparaison entre les estimations et les mesures réelles
%(US)
figure(3)
subplot(1,3,1)
plot(m_donnees(1,:),m_donnees(29,:),'k','LineWidth',2)
grid on
legend('dUS');
xlabel('temps [s]')
title('Estimation du Modèle C++: Distance [cm]')
subplot(1,3,2)
plot(m_donnees(1,:),m_donnees(24,:),'k','LineWidth',2)
grid on
legend('dUS');
xlabel('temps [s]')
title('Mesures Réelles des Capteurs: Distance mesurée [cm]')
subplot(1,3,3)
plot(m_donnees(1,:),erre(5,:),'k','LineWidth',2)
grid on
legend('Erreur US');
xlabel('temps [s]')
title('Ecart Estimation-Réalité [cm]')

%Représentation de la trajectoire du robot et de l'environnement
figure(4)
%Obtention des données sur l'environnement
j=[m_donnees(35,2),m_donnees(36,2)];
k=[m_donnees(37,2),m_donnees(38,2)];
xmur1(1 : 2) = m_donnees(35,2);
xmur2(1 : 2) = m_donnees(36,2);
ymur1(1 : 2) = m_donnees(37,2);
ymur2(1 : 2) = m_donnees(38,2);

%Représentation à chaque pas de calcul de la trajectoire suivie, du robot
```

```
%et de l'environnement
t = max(size(m_donnees));
for i=1 : 1 : t
   representation(m_donnees(:,i))
   hold on
   xr=m_donnees(6,1:i);
   yr=m_donnees(7,1:i);
   plot(xr,yr,'-b','LineWidth',2);
   plot(xmur1,k,'-r',xmur2,k,'-r',j,ymur1,'-r',j,ymur2,'-r','LineWidth',3)
   hold on
   pause(0.1);
   hold off
end
%Grille et légendes
grid on
legend('trajectoire réelle');
xlabel('position x du pousseur (m)')
ylabel('position y du pousseur (m)')
title('Trajectoire suivie par le robot HAMMI')

end
```

Fig. A9.1 : Code du Fichier « comparer.m »

Observons que ce code appelle deux fonctions ou fichiers de *Matlab*. Ils comportent des modifications du code montré dans l'annexe A5 (*modele_capteurs.m*) et d'une autre fonction de Sébastien Rubrecht pour la représentation du robot en mouvement (*representation_graphique.m*).

Les variations principales sont la façon d'obtenir les données d'entrée puisque, dans ce cas, toutes ces données viennent de la matrice de données obtenue dans le logiciel *Robot_HAMMI v2.2*, chargée sur *Matlab*. C'est pour cela qu'on va montrer seulement l'obtention de ces données, au début de ces fonctions :

```
%//*********************************************************//
%//*********************************************************//
%//                    Javier SÁEZ CARDADOR                 //
%//                         Juin  2009                      //
%//                      E.N.S.A.M. Paris                   //
%//                                                         //
%//                         modele.m                        //
%//    Fichier du modèle cinématique pour les capteurs de distance:  //
%//    Ce code est une adaptation du fichier modele_capteurs.m pour  //
%//     prendre les données d'une matrice issue de la trajectoire    //
%//     suivie par le robot HAMMI lors des essais. La structure de   //
%//       la matrice est expliqué dans le fichier comparer.m        //
%//*********************************************************//
%//*********************************************************//

 function [mm]=modele(m_donnees)
%Obtention des données à partir de la matrice m_donnees
%Structure de la matrice en fichier comparer.m

%Paramètres liés au modèle des capteurs (Tableau figure 1.3)
%Angles en rad
gammaIR1 = (50*pi)/180;
gammaIR2 = (72*pi)/180;
gammaIR3 = (110*pi)/180;
gammaIR4 = (135*pi)/180;
gammaUS = (90*pi)/180;
```

```
%Distances en mètres
lyIR14 = 0.329;
lyIR23 = 0.355;
lyUS = 0.363;
lxIR1 = 0.091;
lxIR2 = 0.044;
lxIR3 = -0.042;
lxIR4 = -0.090;

%Paramètres de l'Environnement en mètres
xmur1 = m_donnees(35,2);
xmur2 = m_donnees(36,2);
ymur1 = m_donnees(37,2);
ymur2 = m_donnees(38,2);

%Obtention des coordonnées Xr, Yr et Thétar
x=[m_donnees(6,:)',m_donnees(7,:)',m_donnees(8,:)'];

[…]

end
```

Fig. A9.2 : Début du code du Fichier « modele.m »

```
%//***********************************************************//
%//***********************************************************//
%//                    Javier SÁEZ CARDADOR                   //
%//                         Juin  2009                        //
%//                      E.N.S.A.M. Paris                     //
%//                                                           //
%//                      representation.m                     //
%//    Fichier pour le dessin du robot lors d'une trajectoire suivie:  //
%//   Ce code est une modification du fichier representation_graphique.m  //
%//    de Sébastien Rubrecht (06/2007) pour représenter le robot durant  //
%//     sa trajectoire. La roue folle a été ôté pour plus de simplicité  //
%//   et les paramètres du robot sont inclus pour n'avoir besoin que des  //
%//   coordonnées du centre de gravité Ar lors de la trajectoire suivie  //
%//***********************************************************//
%//***********************************************************//

function [  ] = representation(m_donnees)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%% Récupération des données %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Paramètres fonctionnels
ler =   0.450;  %Longueur de l'Essieu du Robot
ar =    0.100;  %position du point Ar (= point de commande du Robot)
rr =    0.100;  %Rayon de la roue arrière du Robot

%Paramètres liés à la cinématique des roues folles
ler3r = 0.330;  %Longueur entre l'Essieu et la Roue3 (roue folle) du Robot
dr =    0.050;  %Décalage entre la liaison roue folle et le centre roue folle Robot

%Paramètres liés à la cinématique roue folle non récupéré
r13 =   0.050;  %Rayon roue 13

%Paramètres lié la géométrie extérieure
lar =   0.520;  %Largeur entre les deux points extrêmes du Robot
lear =  0.080;  %Longueur entre l'Essieu et l'Arrière du Robot
lo1r =  0.135;  %Longueur 1 Robot
```

```
la1r =  0.350;  %Largeur 1 Robot
lo2r =  0.280;  %Longueur 2 Robot
la2r =  0.210;  %Largeur 2 Robot
la3r =  0.075;  %Largeur 3 Robot
llr =   0.210;  %Longueur entre la Liaison et le Robot

%Paramètres annexes
larfr = 0.025;  %Largeur Roue Folle Robot
larar = 0.035;  %Largeur Roue Arrière Robot

%Récupération du vecteur d'état
xr = m_donnees(6,:)';
yr = m_donnees(7,:)';
thetar = m_donnees(8,:)';

[…]

end
```

Fig. A9.3 : Début du code du Fichier « representation.m »

Pour finir, il faut dire que dans cette dernière fonction, en raison de simplicité, on a aussi enlevé la représentation de la roue folle du robot. Cela n'empêche pas que ce fichier donne une bonne représentation et l'obtention de données devient plus simple.