Trinôme : BERRETTI Cédric
MANRIQUE Nicolas
TREMEGES Jérôme

TER : Projet n°23

Développement d'une interface graphique D'une maquette de TP d'automatique

(Encadrant : Hassan Noura)

Master SIS M1

Sommaire

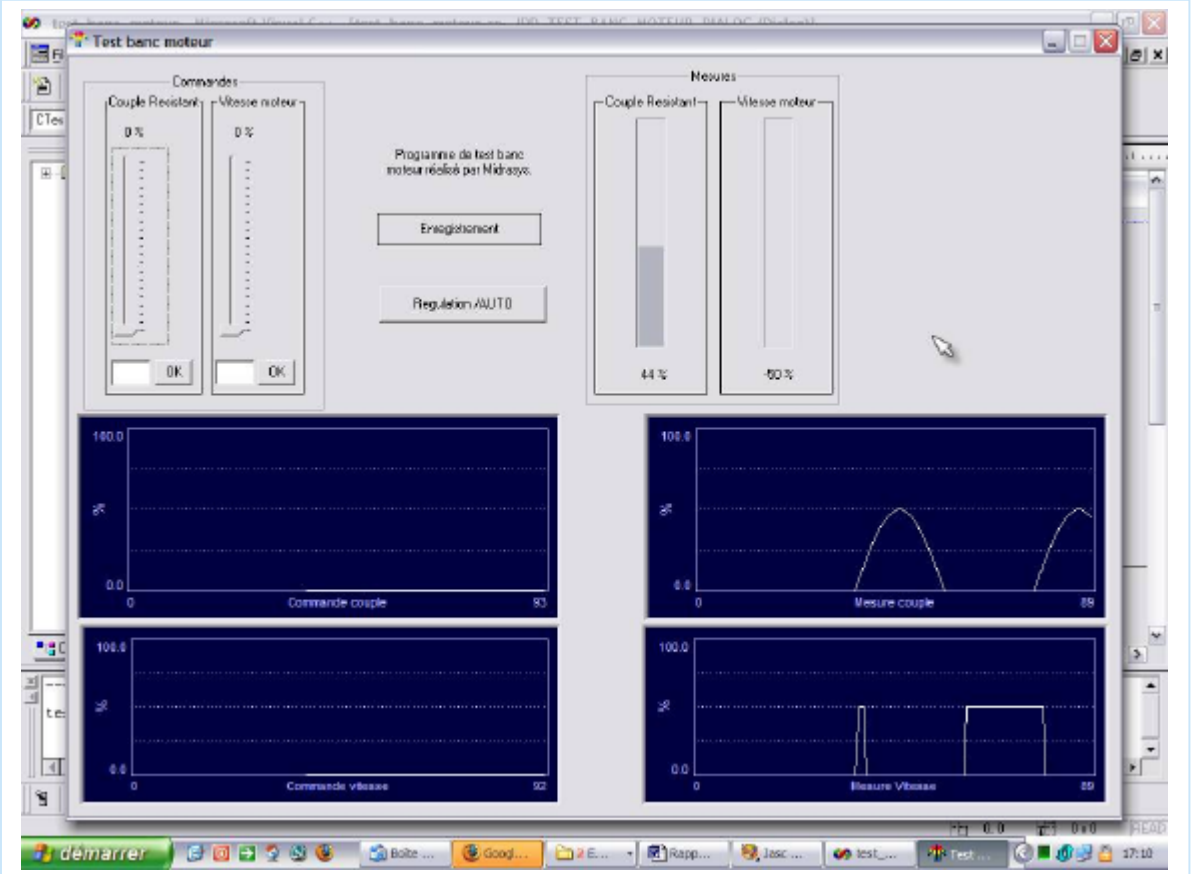
1) Présentation du projet

Notre projet de TER consiste à développer l'interface graphique d'une maquette de TP d'automatique qui est disponible au département GII de l'école Polytech' Marseille.

Cette maquette est constituée d'un banc moteur et d'un réducteur de couple sur lesquels ont été placé des capteurs de pression, de vitesse et de vibration. Elle peut être pilotée soit manuellement à partir de potentiomètres présents sur l'armoire de commande ou bien à distance via un PC associé à une carte d'acquisition de données (de la marque Advantech Automation).

La société Midrasys qui a construit ce banc moteur avait développé un premier programme de pilotage, il nous a été demandé d'y ajouter de nouvelles fonctionnalités. Ce programme initial est conçu en Visual C++ avec des fonctions de gestion des entrées/sorties de la carte fournit par le constructeur.

2) Aperçu du programme existant



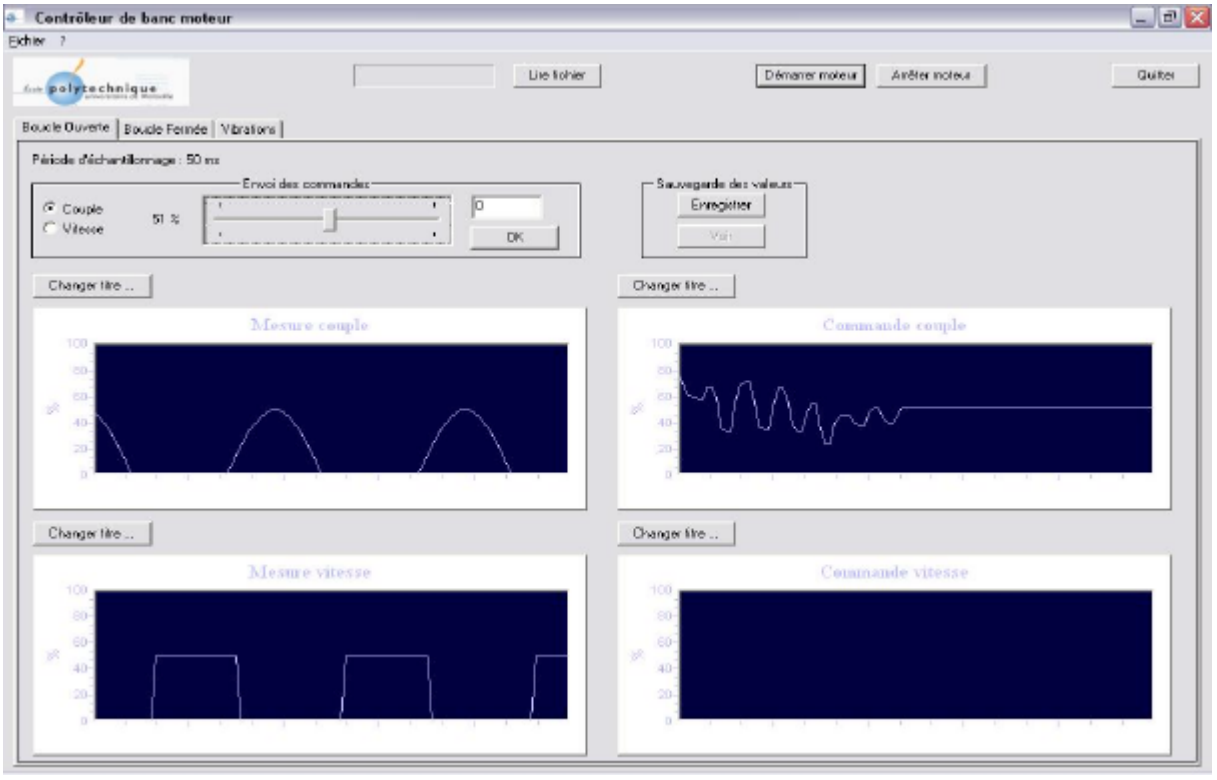
Ce programme affiche les valeurs des mesures des capteurs de couple et de vitesse (en haut à droite sur l'image).
A partir des deux « sliders » (en haut à gauche sur l'image), on règle en pourcentage, le couple du réducteur ainsi que la vitesse de rotation du moteur, jouant le rôle des potentiomètres du tableau de commande. On peut donc visualiser sur quatre graphiques l'affichage des différentes valeurs d'entrées/sorties.
Au centre de la fenêtre « Test banc moteur », on distingue deux boutons.
Le premier bouton, permet d'enregistrer dans un fichier texte les valeurs acquises.
Le second, « Régulation/Auto », place le système en boucle fermée, c'est à dire que toutes les valeurs entrées par les sliders seront des consignes que le système devra atteindre.
Pour y arriver le programme utilise un algorithme de régulation (qui nous a été donné et qui fait l'objet d'un TP d'automatisme).

3) Modifications apportées au projet

Afin de pouvoir avancer dans notre projet, nous avons dû installer une carte d'acquisition virtuelle sur notre ordinateur (voir en page annexe pour plus de détails).
Les modifications que nous avons apportées sur le programme existant sont :

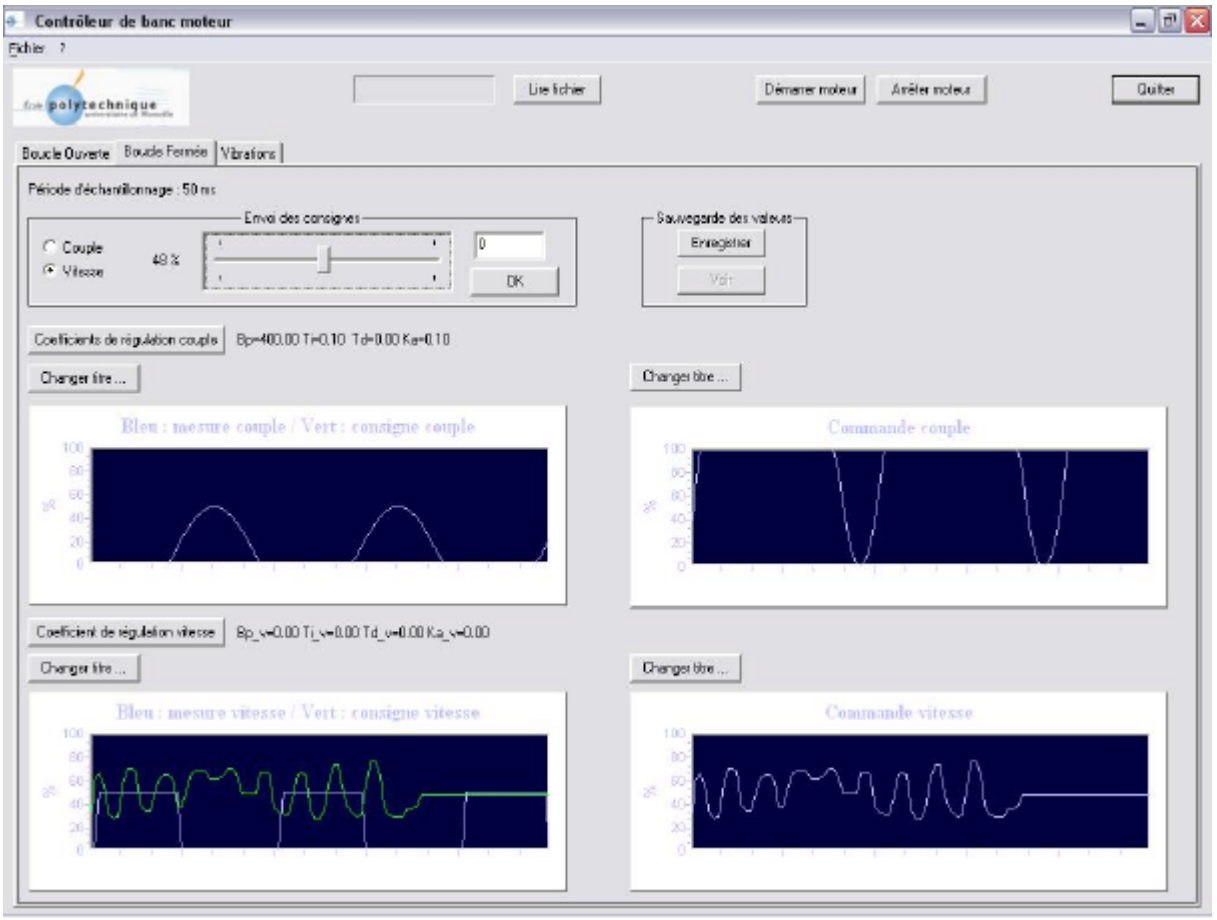
- Intégration d'un onglet boucle ouverte

  L'ouverture de cet onglet passe le logiciel en mode boucle ouverte. Dans ce mode, le programme se contente d'envoyer des commandes de vitesse et de couple au banc moteur et d'afficher les valeurs reçues. Contrairement au mode boucle fermée, il n'y a pas d'interaction entre les valeurs lues sur la carte d'acquisition et celles qu'on envoie au moteur.



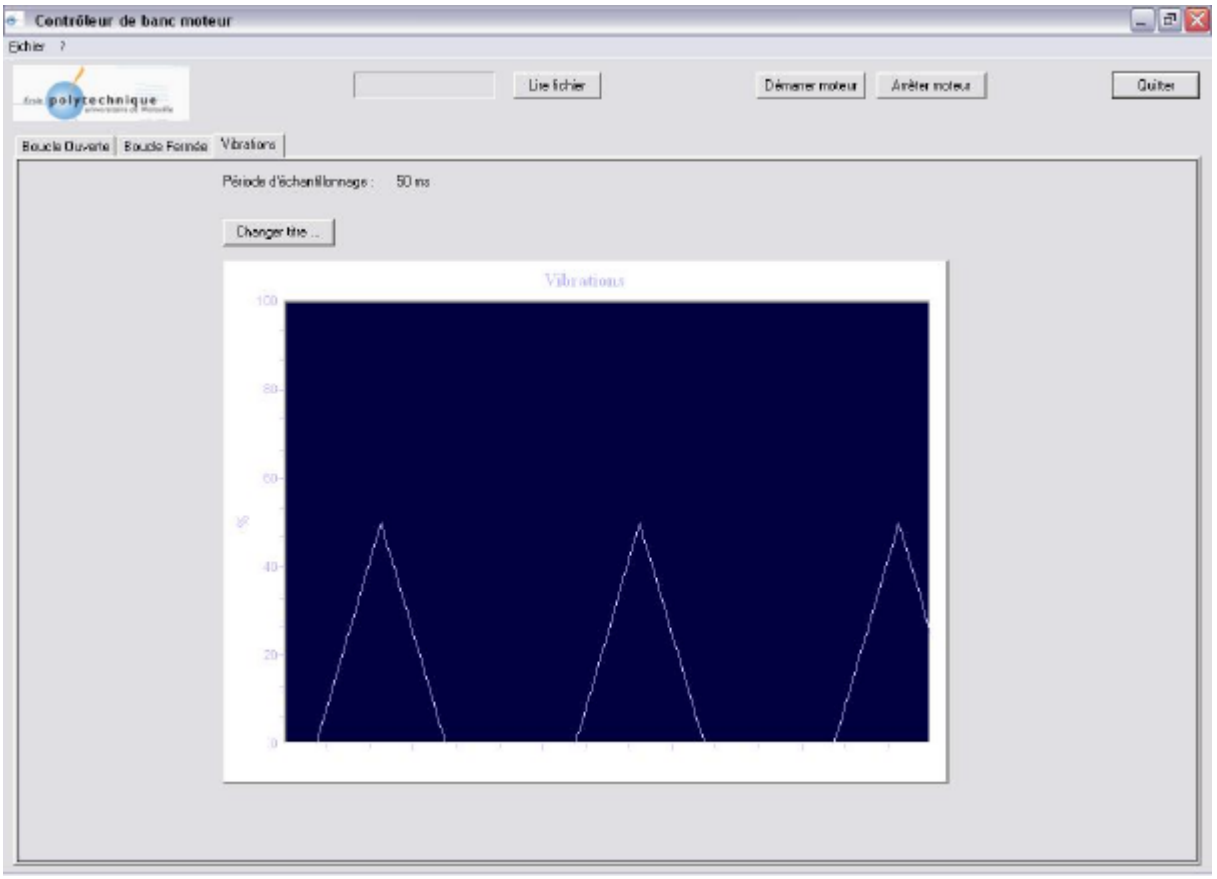
- Intégration d'un onglet boucle fermée

  En mode boucle fermée, l'utilisateur entrera non pas des commandes mais des consignes de vitesse et de freinage au programme et ce dernier se chargera de les respecter. C'est à dire que cette fois-ci les valeurs de vitesse et de couple envoyées dépendront de celles qui ont été reçues. Grâce à un algorithme de régulation, les commandes de couple et de vitesse envoyées permettront de converger vers les valeurs des consignes. Ainsi on parle de boucle fermée car les valeurs de sortie sont utilisées pour calculer les nouvelles valeurs d'entrée.



- Intégration d'un onglet vibrations

  Cet onglet contient un graphique permettant de visualiser les valeurs d'un capteur de vibration placé sur le banc moteur.



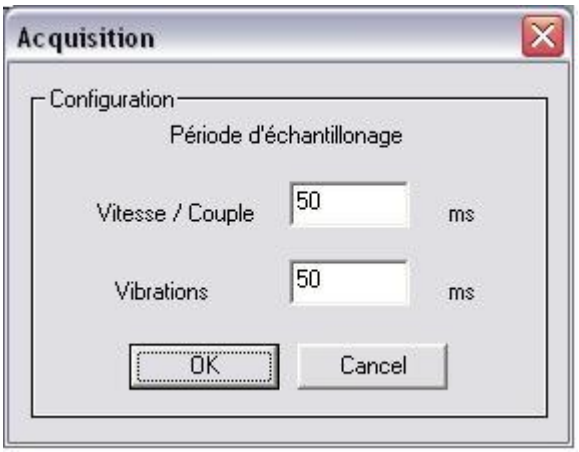
- La possibilité de lire un fichier de données.

  Le bouton « Lire fichier » permet de lire dans un fichier texte les valeurs de vitesse et de couple à envoyer. La syntaxe du fichier doit être la suivante : une ligne doit contenir 2 valeurs entre 0 et 100 séparées par un espace. La première correspond à la valeur de couple et la seconde à la valeur de vitesse. A chaque période d'échantillonnage un nouveau couple de valeurs est envoyé au moteur.

- La possibilité d'enregistrer et de visualiser les données acquises par le système.

  Sur les fenêtres des onglets boucle ouverte et boucle fermée, il existe un bouton « Enregistrer » qui permet d'écrire dans un fichier les valeurs lues et envoyées à chaque période d'échantillonnage.
  En boucle ouverte les valeurs écrites sont les suivantes : Mesure du couple, Mesure de la vitesse, Commande couple et Commande vitesse.
  En boucle fermée les valeurs enregistrées sont : Mesure du couple, Mesure de la vitesse, Commande couple, Commande vitesse, Consigne couple et Consigne vitesse.
  A tout moment, il est possible de voir le contenu de ces fichiers en appuyant sur « visualiser ».

- Possibilité de choisir les périodes d'échantillonnage de la vitesse, du couple et de la vibration.



- En boucle fermée, possibilité de modifier les valeurs des coefficients de régulation du couple: Bp, Ti, Ka, Td ainsi que celle des coefficients de régulation de la vitesse Bp_v, Ti_v, Ka_v, Td_v.

- Intégration d'un nouveau type de graphes.

  Les graphes du programme initial ne permettaient pas de tracer plusieurs courbes à la fois, or en boucle fermée nous voulions avoir sur le même graphe les consignes souhaitées et les valeurs mesurées. Nous avons donc intégrer un nouveau type de graphe permettant de faire cela : le 2D graph activex control de Nikolai Teofilov.
  Cf La page web de ce composant :
  http://www.codeproject.com/miscctrl/ntgraph_activex.asp
  Se référer à l'annexe pour des détails sur l'intégration de ce composant au projet.

- La possibilité d'éditer les titres des graphes.

- La possibilité de donner des commandes ou des consignes par le biais de boites d'édition.

  Nous avons ajouté des champs de texte pour taper directement des valeurs de commande et de consigne en plus des sliders initiaux.

- La création d'un installeur installshield qui se charge de copier sur le disque dur de l'utilisateur l'exécutable du programme ainsi que les fichiers dont il dépend c'est à dire les .dll de la carte d'acquisition et le fichier .ocx pour les graphes.

- Création d'un DVD contenant tout le nécessaire pour travailler sur la maquette :
  - Les pilotes de la carte d'acquisition.
  - L'installeur du programme que nous avons réalisé.
  - Les fichiers sources du programme.
  - L'environnement de programmation Visual C++
  - Ce rapport.

4) Ce que nous avons appris

Ce projet nous a permis de nous familiariser avec l'environnement de programmation Microsoft Visual C++ qui est souvent utilisé dans l'industrie. Bien qu'étant un outil de développement performant, nous avons été néanmoins obligé de programmer en C++ et donc de revoir les concepts fondamentaux de la programmation objet abordés en Licence d'informatique. La puissance de ces différents concepts nous a bien fait apparaître le langage C++ comme une amélioration notable du C.
Lors de la création de l'installeur du DVD, nous avons aussi dû apprendre quelques notions du langage de programmation VBScript qui est un langage dérivé du visual basic et qu'on peut aussi rencontrer dans l'industrie.
Bien qu'étant spécifié dans notre sujet qu'aucune compétence en automatisme n'était requise, nous avons tous de même pu découvrir cette discipline notamment la notion de régulation.
Enfin ce projet nous a appris à programmer en équipe, ce qui n'est pas facile notamment lorsqu'il s'agit de fusionner des parties de codes faites par différentes personnes.

5) Sitographie

- https://secure.codeproject.com/ (exemple et code en visual c++)
- http://c.developpez.com/faq/vc/ (FAQ Visual c++)
- http://msdn.microsoft.com/visualc/ (site officiel d'aide MSDN pour visual c++)
- http://www-timc.imag.fr/Antoine.Leroy/tutoriaux/VisualCPP/VisualCPP.html

6) Conclusion

Pour conclure nous pourrions nous demander que changerions nous si nous devions refaire ce projet. Et par exemple aurions nous aussi opté pour Visual C++ si nous avions dû faire le programme du début à la place de la société Midrasys. Nous aurions eu le choix entre les langages Delphi et Visual basic car les constructeurs de la carte d'acquisition proposent des fonctions aussi dans ces langages. Delphi repose sur le langage pascal et Visual Basic sur le basic de Microsoft. Or nous pensons qu'ils offrent moins de possibilités que le Visual C++ qui permets d'utiliser tous les concepts de la programmation objet (héritage, encapsulation, polymorphisme).
Enfin voici quelques modifications que l'on pourrait apporter au programme :

- Possibilité d'afficher le programme correctement dans des résolutions inférieures à 1024x768.
- Possibilité de changer les périodes d'échantillonnage au cours de l'exécution du programme.

Page Annexe

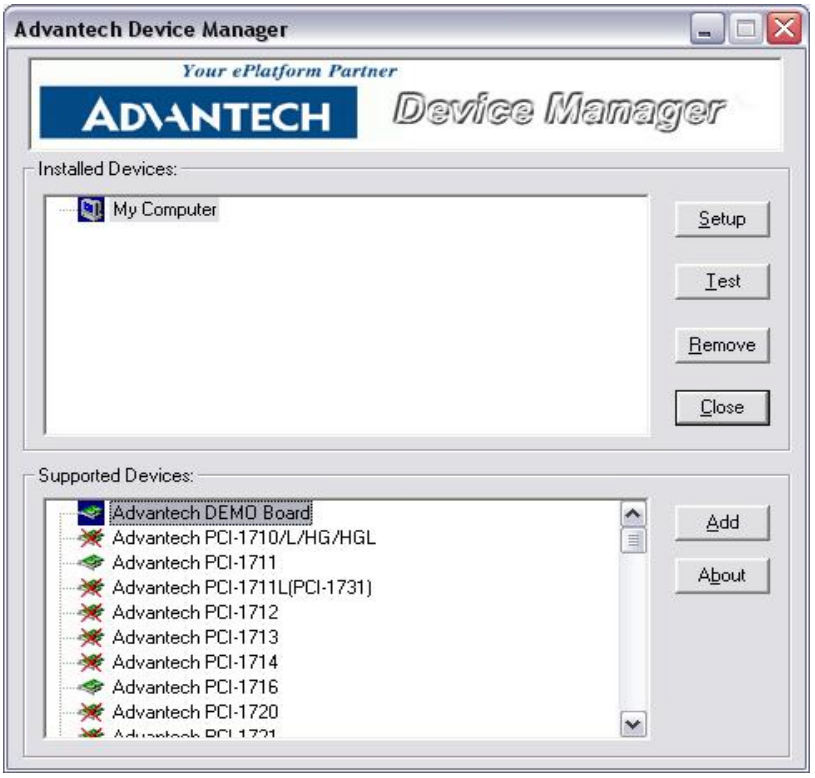
Annexe 1 : Comment installer une carte d'acquisition virtuelle ?

Afin de pouvoir travailler chez nous sans la maquette, nous avons installé sur nos machines personnelles la carte de démonstration du CD-ROM Advantech. Il s'agit en fait d'une carte d'acquisition virtuelle qui envoie différents signaux sur ses sorties (signaux sinusoïdaux, créneaux cf. les copies d'écran plus haut ...).
Voici la procédure à suivre pour installer cette carte. Il faut d'abord ouvrir le CD-ROM Advantech Automation et la page suivante s'affichera :

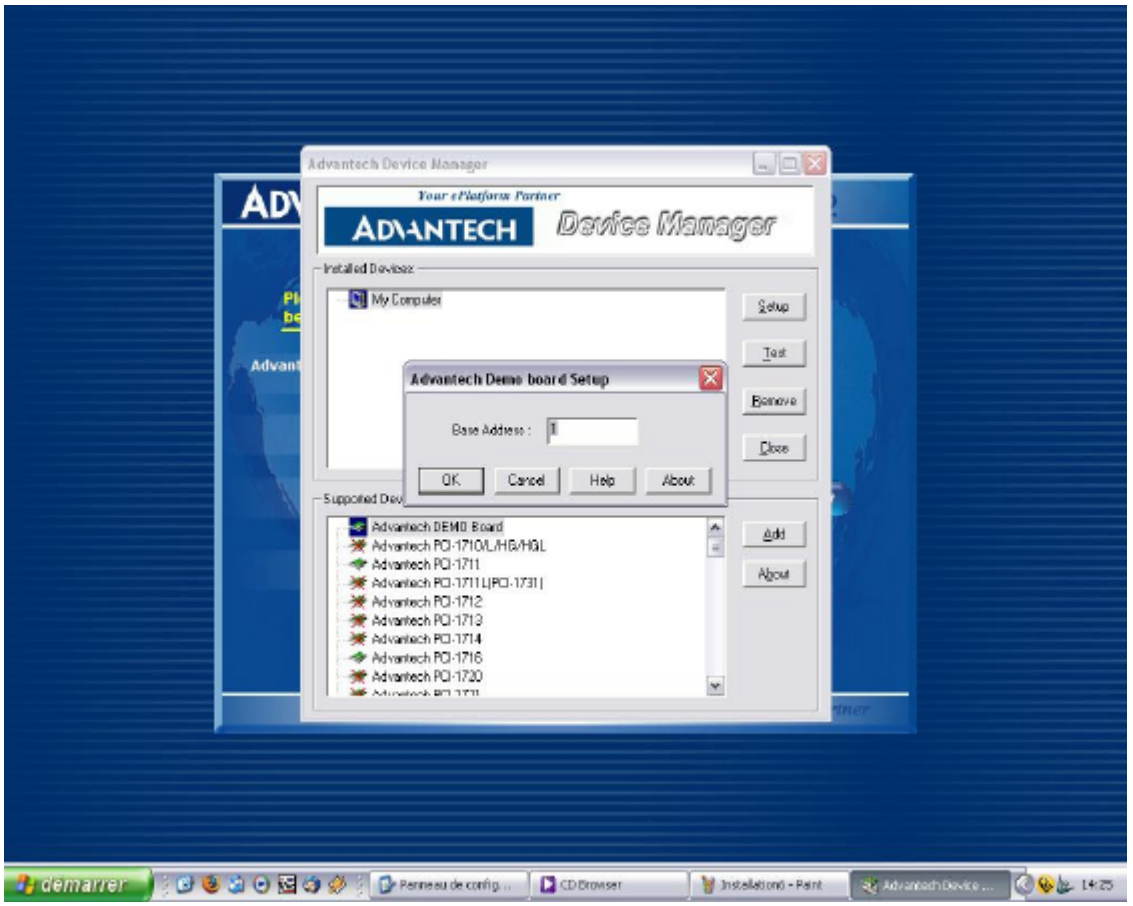
Cliquez sur « continue » pour arriver au menu ci-dessous :

Choisissez le bouton Installation .

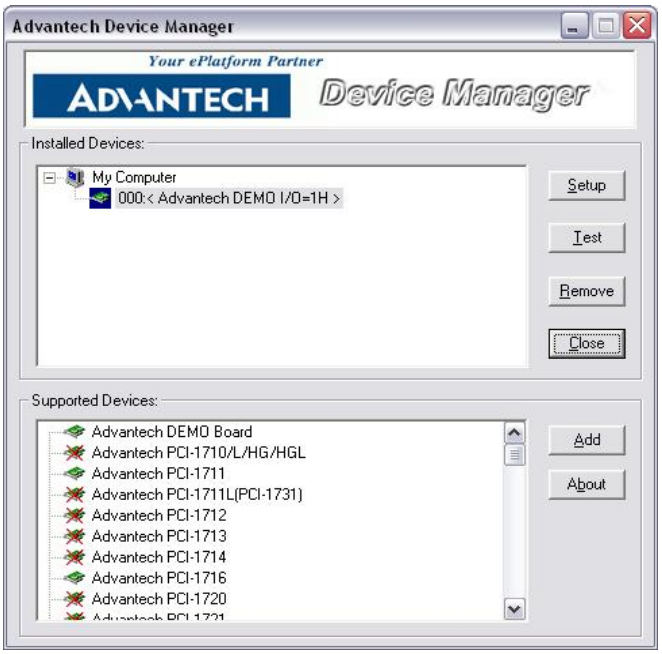
Installer le « device manager », c'est ce logiciel qui permet l'installation de la carte virtuelle. Laissez-vous guider par l'installeur du programme puis lancer le à partir du raccourci qui a normalement été créé dans votre menu démarrer.



Sélectionnez « Advantech DEMO Board » puis cliquez sur Add.



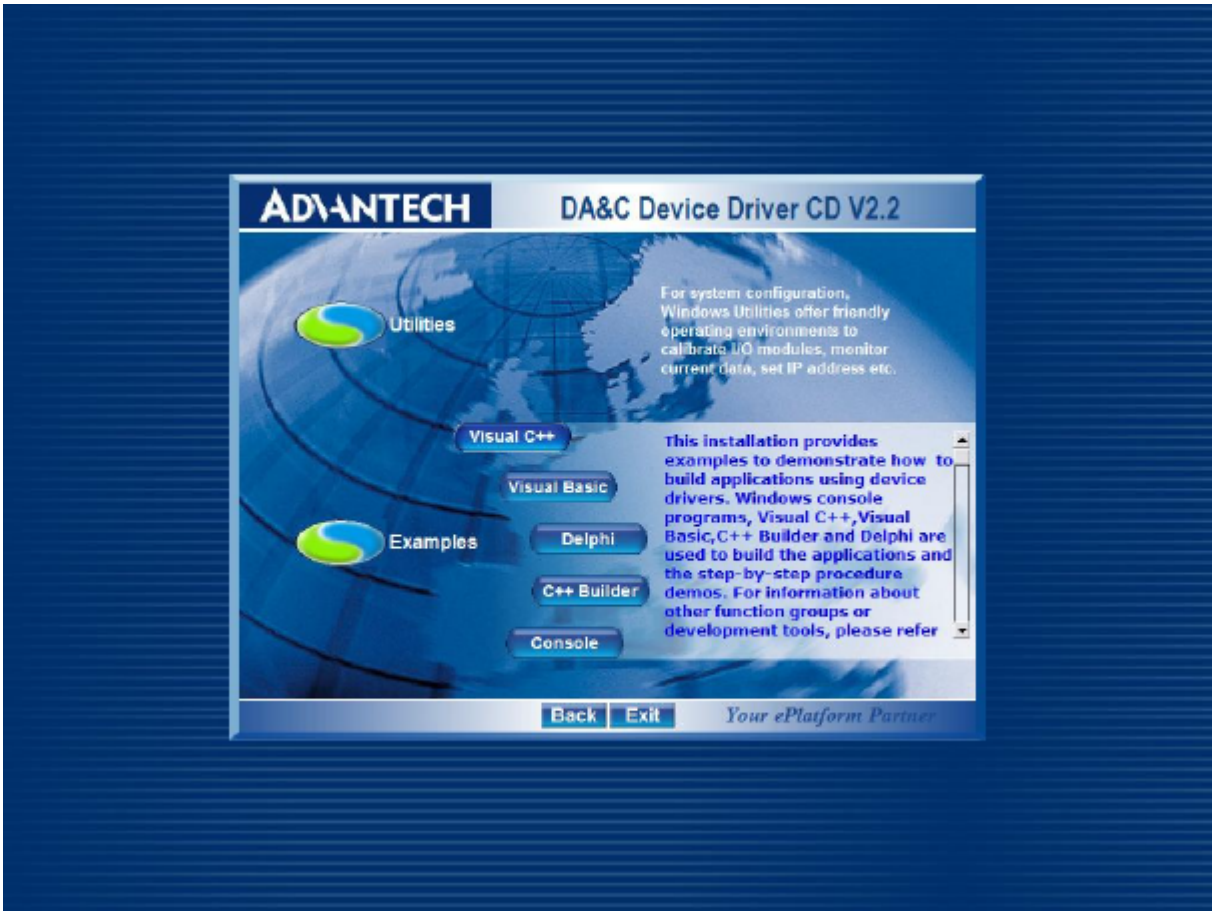
Laissez « 1 » comme « Base Address » et cliquez sur OK.



Voilà, la carte virtuelle est installée, vous pouvez fermer l'application et utiliser maintenant l'application banc moteur en simulant la maquette.

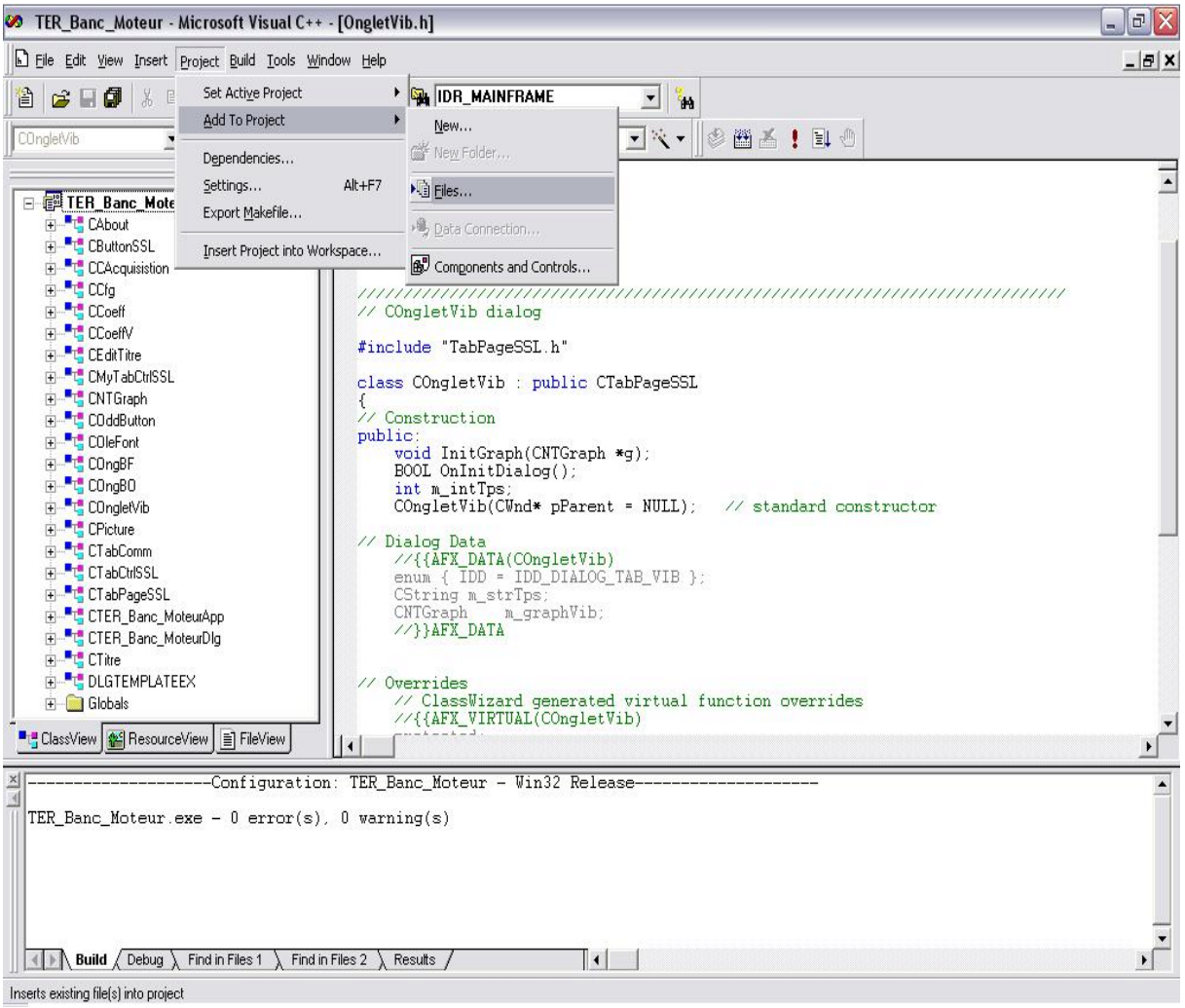
Annexe 2 : Comment utiliser les fonctions de la carte d'acquisition dans Visual C++ ?

Il faut tout d'abord installer les fichiers « examples » et « utilities » du CD-ROM Advantech. Pour cela il faut lancer l'installeur de ce CD-ROM puis choisir installation puis Example & Utility, on obtient alors la fenêtre suivante :



Pour notre application nous devons choisir d'installer les utilitaires pour Visual C++. Puis, pour se servir des fonctions qui manipulent la carte d'acquisition, il faut ensuite importer dans Visual C++ le fichier entête de ces fonctions. La seconde étape consiste à lier la bibliothèque à l'exécutable du programme.

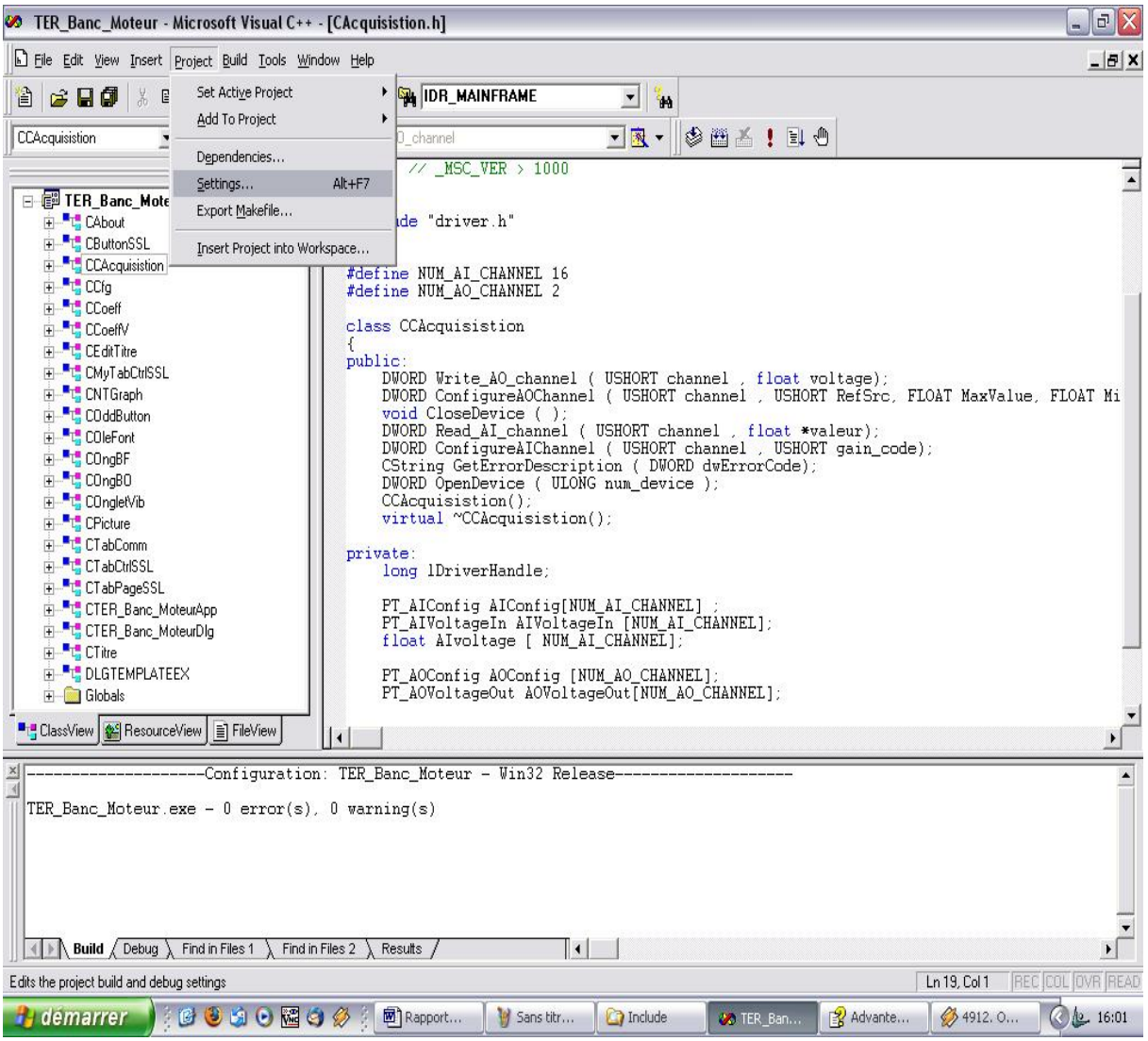
Tout d'abord, pour importer le fichier entête dans notre projet, on doit faire ceci dans Visual C++ :



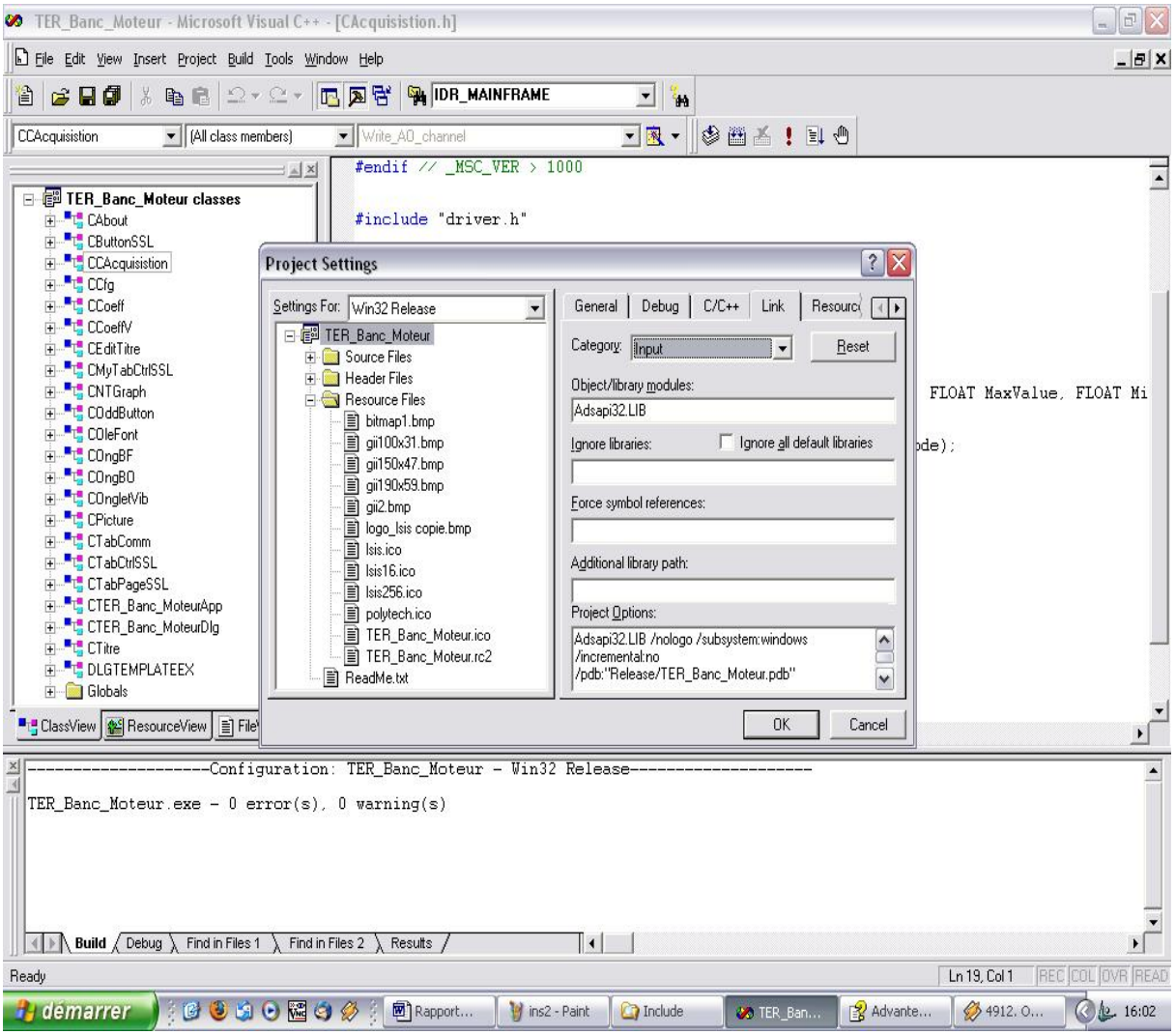
Quand la boite de dialogue pour ajouter des fichiers apparaît, sélectionner le fichier Driver.h qui doit normalement se trouver dans le répertoire C:\Program Files\ADVANTECH\ADSAPI\Include sauf si vous avez spécifié un chemin différent lors de l'installation. Pour utiliser les fonctions de manipulation de la carte, il faudra ensuite spécifier dans le fichier source la directive `#include « driver.h »`

Maintenant pour ajouter le .dll lors de la compilation du programme, nous devons faire ceci dans Visual C++ :

Choisir Project / Settings :



Cette manipulation ouvre la fenêtre suivante :

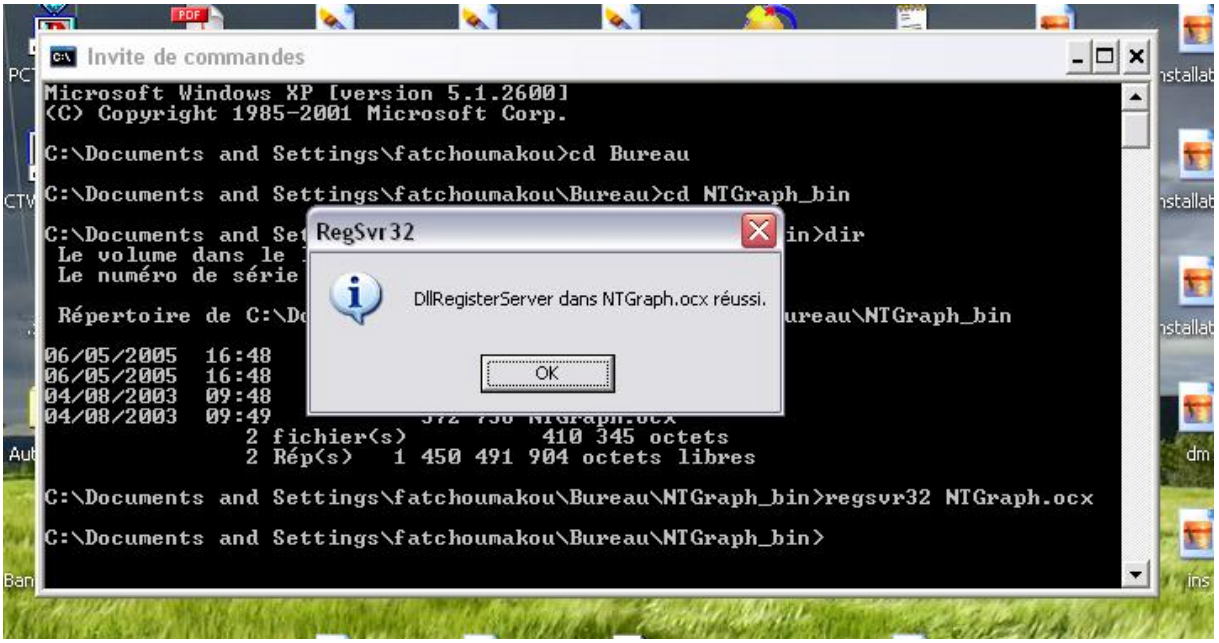


Dans l'onglet Link choisir Input comme Category et dans le champ « Object/library modules » entrer le chemin complet du fichier Adsapi32.LIB. Le mieux est de faire comme dans l'exemple c'est-à-dire de copier ce fichier dans le même répertoire que le projet et de taper seulement Adsapi32.LIB dans le champ.
Par défaut, ce fichier est initialement dans le répertoire :
C:\Program Files\ADVANTECH\ADSAPI\Lib .

Annexe 3 : Installation du fichier ocx pour les graphes

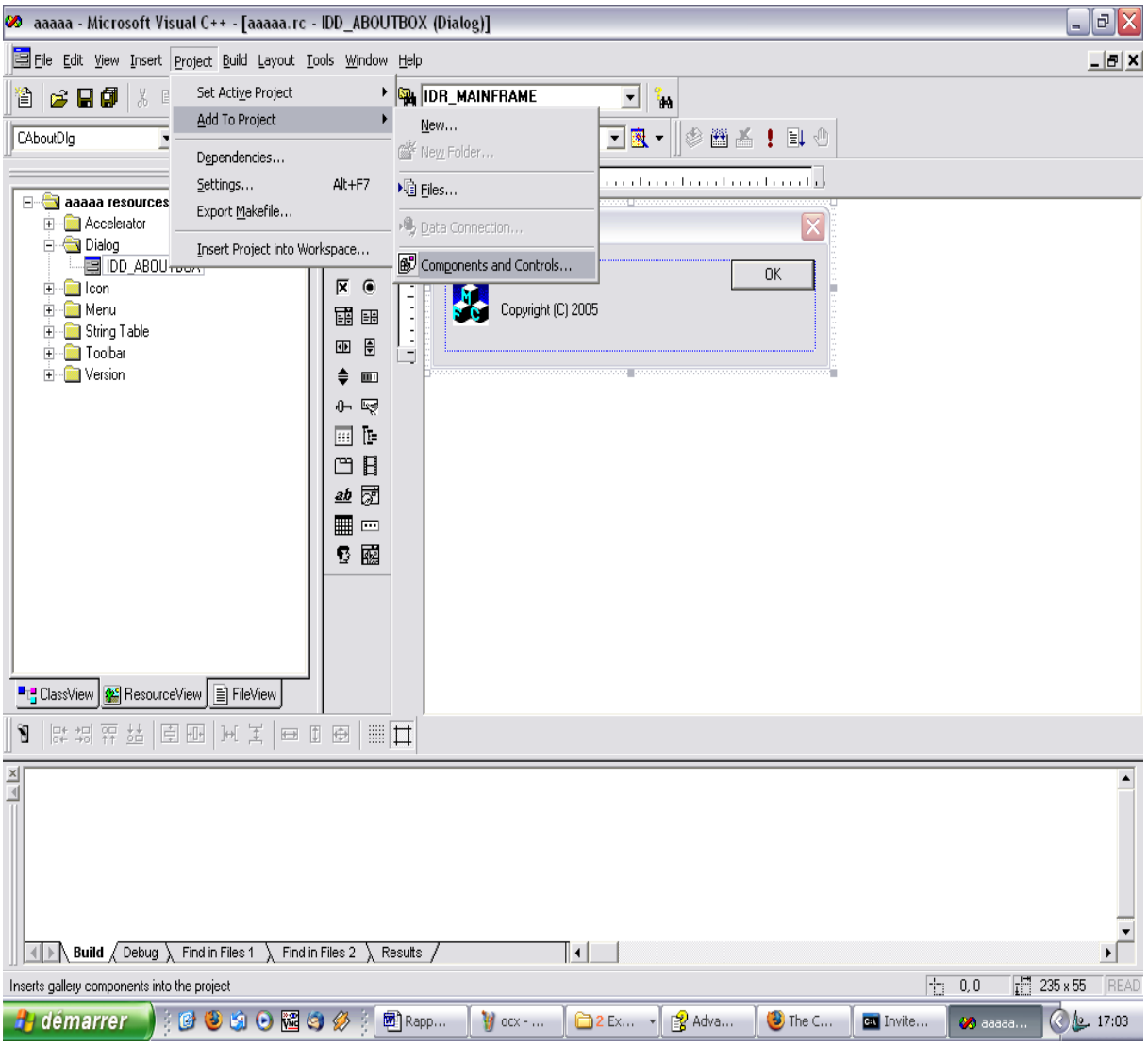
L'installation de ce fichier est faite automatiquement par l'installeur BancMoteur.exe. Cependant si vous souhaitez seulement travailler sur le code source du programme sans avoir au préalable installé l'exécutable, il faudra vous charger vous-même de l'installation de ce fichier.

Commencer par télécharger le fichier NTGraph.ocx sur la page http://www.codeproject.com/miscctrl/ntgraph_activex.asp.
Puis avec une fenêtre de commande MS-DOS il faut se mettre dans le répertoire qui contient le Fichier NTGraph.ocx et tapez la commande : regsvr32 NTGraph.ocx

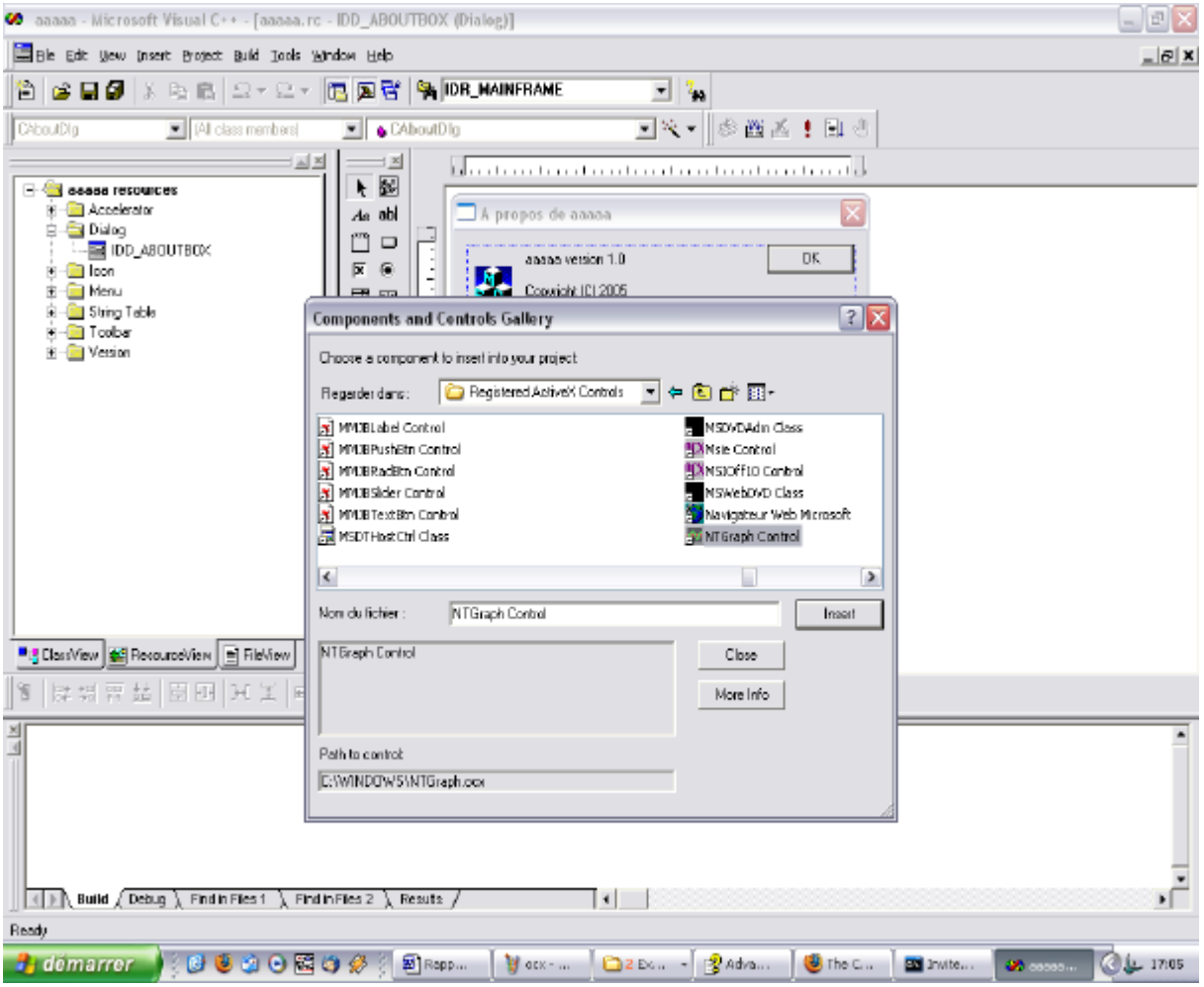


L'opération inverse se fait par la commande suivante : regsvr32 /u NTGraph.ocx

Par la suite, pour intégrer ce composant au projet nous avons cliqué sur :
Project / Add To Project / Components and Controls …



Puis nous avons choisit le NTGraph Control dans la liste des composants, ce qui ajoute une classe CNTGraph au projet et cela ajoute ce graphe à la liste des composants déjà disponible dans l'éditeur de ressources.



Annexe 4 : Description interne du programme

Le programme est réalisé en C++. Il s'agit d'un langage orienté objet, c'est-à-dire qu'il est composé d'éléments appelés « classes » qui ont des tâches spécifiques et qui interagissent. Nous allons décrire le rôle des principales classes du programme.

- CCAcquisistion : Elle permet le « dialogue » avec la carte d'acquisition.
Ses principales fonctions sont l'ouverture du périphérique de la carte, sa configuration avec notamment la spécification des valeurs maximales et minimales pouvant être reçues. Cette classe possède aussi évidemment les fonctions pour lire et écrire des valeurs sur la carte. Ce sont les fonctions les plus importante car elles permettent d'acquérir les mesures de couple et de vitesse ainsi que d'envoyer les commandes.

- CTER_Banc_MoteurApp : Cette classe contient la première fonction exécutée au lancement du programme. Il s'agit de la fonction « InitInstance ». Son rôle est d'ouvrir la fenêtre de dialogue qui demande d'entrer les périodes d'échantillonnage et ensuite d'ouvrir la fenêtre principale du programme.

- CTER_Banc_MoteurDlg : Cette classe correspond à la fenêtre principale du logiciel. Son initialisation conduit à la création des trois onglets : Boucle ouverte, Boucle fermée et Vibrations. Cette classe contient aussi une fonction primordiale : la fonction « OnTimer », elle est appelée à chaque période d'échantillonnage et c'est elle qui se charge de lire les mesures sur la carte d'acquisition et d'envoyer les commandes. Cette fonction est décrite en détail plus bas.

- COngBO : Celle-ci correspond à l'onglet de boucle ouverte. Son rôle est la création des graphes pour la boucle ouverte et la lecture des commandes de l'utilisateur soit par le biais du slider, soit par celui du champ de texte.

- COngBF : C'est la classe de l'onglet de boucle fermée. Son rôle est le même que celui de la boucle ouverte : création de graphes et gestion des commandes de l'utilisateur. Elle permet aussi à l'utilisateur d'entrer de nouveaux coefficients de régulation.

- COngletVib : C'est la classe du troisième et dernier onglet de la fenêtre principale. Elle affiche le graphe des vibrations du moteur.

Les fichiers picture.cpp, picture.h, StdAfx.cpp, StdAfx.h, font.cpp, font.h, MyTabCtrlSSL.cpp, MyTabCtrlSSL.h, ntgraph.cpp et ntgraph.h sont des librairies pour l'interface graphique, utilisé pour les onglets, les images, les graphes 2D, la police des caractères, etc.

Ci-dessous la fonction « OnTimer(UINT nIDEvent) » de la classe « CTER_Banc_MoteurDlg ».

```
…
void CTER_Banc_MoteurDlg::OnTimer(UINT nIDEvent)
{
	m_bMoteurStopped = !(m_OngletBO.m_bRedemarrer || m_OngletBF.m_bRedemarrer);

	if (nIDEvent == 2)
	{
		// Acquisition Vibrations

		DWORD error_code;
		float mesure_vib;

		if ( (error_code= m_CarteAcqui.Read_AI_channel( 2 , &mesure_vib)) != SUCCESS)
		{
			CString desc_error = m_CarteAcqui.GetErrorDescription ( error_code);
			MessageBox ( desc_error , "Erreur d'acquisition", MB_OK | MB_ICONERROR);
			KillTimer(2);
		}

		static int xvib = 0;
		static int xvibmax = 100;
		if (xvib > xvibmax)
		{
			xvibmax = xvib;
			MAJGraphVib(xvibmax);
		}

		m_OngletVib.m_graphVib.PlotXY(xvib,mesure_vib*10.,0);

		xvib++;

		CDialog::OnTimer(nIDEvent);

		return;
	}

	// Acquisition Couple /Vitesse

	DWORD error_code;
	float mesure_couple, mesure_vitesse;

	if ( (error_code= m_CarteAcqui.Read_AI_channel( 0 , &mesure_couple)) != SUCCESS)
	{
		CString desc_error = m_CarteAcqui.GetErrorDescription ( error_code);
		MessageBox ( desc_error , "Erreur d'acquisition", MB_OK | MB_ICONERROR);
		KillTimer(1);
	}

	if ( (error_code= m_CarteAcqui.Read_AI_channel( 1 , &mesure_vitesse)) != SUCCESS)
	{
		CString desc_error = m_CarteAcqui.GetErrorDescription ( error_code);
		MessageBox ( desc_error , "Erreur d'acquisition", MB_OK | MB_ICONERROR);
		KillTimer(1);
	}

	// Calcul des commandes et des consignes

	float commande_couple, commande_vitesse;
	int consigne_couple, consigne_vitesse;

	if (m_bMoteurStopped)
	{
		commande_couple = 0;
		commande_vitesse = 0;
		consigne_couple = 0;
		consigne_vitesse = 0;
	}
```

Indique l'état du moteur. Si le bouton démarré n'a été enclenché, la variable m_bMoteurStopped est à 0.

Appel de la fonction OnTimer

Si un problème est apparut lors de la réception des données vers le PC, un message apparaît indiquant le type d'erreur.

```
else if (m_bLectureFic)            // bouton lecture de fichier cliqué
{
        CTabComm & tab_com = MyList.GetNext(PosListe);

        commande_couple = tab_com.Tab[0]/10;
        commande_vitesse = tab_com.Tab[1]/10;

        m_Progress.SetPos(m_Progress.GetPos()+1);

        if (PosListe==MyList.GetTailPosition())
        {
                m_Progress.SetPos(0);
                m_bLectureFic = FALSE;
        }
}
else if (m_TabCtrl.m_bBoucleOuverte)
{
        // Boucle ouverte
        consigne_couple = m_OngletBO.BOCouple;
        consigne_vitesse = m_OngletBO.BOVitesse;
        commande_couple = (float)m_OngletBO.BOCouple/10;
        commande_vitesse = (float)m_OngletBO.BOVitesse/10;
}
else
{
        // Boucle fermée
        //=====================================================
        //      Algo de régulation pour le couple.
        //=====================================================
        //coef PID
        float Bp = m_OngletBF.Bp;
        float Ti = m_OngletBF.Ti;
        float Ka = m_OngletBF.Ka;
        float Td = m_OngletBF.Td;
                                        ……

        //=====================================================

        //=====================================================
        //      Algo de régulation pour la vitesse.
        //=====================================================
        float bps_v = m_OngletBF.bps_v;
        float ti_v = m_OngletBF.ti_v;
        float ka_v = m_OngletBF.ka_v;
        float td_v = m_OngletBF.td_v;

        commande_vitesse = (float)m_OngletBF.BFVitesse/10;
                                        ……

        //=====================================================

        consigne_couple = m_OngletBF.BFCouple;
        consigne_vitesse = m_OngletBF.BFVitesse;
        commande_couple = comm_couple;
}

// Envoi des commandes
if(commande_couple>=10)     commande_couple=10;
if(commande_couple<=0)      commande_couple=0;
if(commande_vitesse>=10)    commande_vitesse=10;
if(commande_vitesse<=0)     commande_vitesse=0;

if ( (error_code= m_CarteAcqui.Write_AO_channel( 0, commande_couple)) != SUCCESS)
{
        CString desc_error = m_CarteAcqui.GetErrorDescription ( error_code);
        MessageBox ( desc_error , "Erreur d'acquisition", MB_OK | MB_ICONERROR);
}
if ( (error_code= m_CarteAcqui.Write_AO_channel( 1, commande_vitesse)) != SUCCESS)
{
        CString desc_error = m_CarteAcqui.GetErrorDescription ( error_code);
        MessageBox ( desc_error , "Erreur d'acquisition", MB_OK | MB_ICONERROR);
}
```

Récupération des données enregistrées dans un fichier texte.

Récupération des coefficients du couple entrer par l'utilisateur sur la boite de dialogue nommée algorithme de régulation.

Récupération des coefficients de vitesse entrer par l'utilisateur sur la boite de dialogue nommée algorithme de régulation.

Si un problème est apparut lors de l'envoie des données vers la carte d'acquisition, un message apparaît indiquant le type d'erreur.

```
// Dessin des courbes
static int x = 0;
static int xmax = 100;

if (x > xmax)
{
	xmax = x;
	MAJGraphEnt(xmax);
}

m_OngletBO.m_scopeBOEntCouple.PlotXY(x,mesure_couple*10.,0);
m_OngletBO.m_scopeBOEntVitesse.PlotXY(x,mesure_vitesse*10.,0);

m_OngletBF.m_scopeBFEntCouple.PlotXY(x,mesure_couple*10.,0);
m_OngletBF.m_scopeBFEntVitesse.PlotXY(x,mesure_vitesse*10.,0);

m_OngletBF.m_scopeBFEntCouple.PlotXY(x,m_OngletBF.BFCouple,1);
m_OngletBF.m_scopeBFEntVitesse.PlotXY(x,m_OngletBF.BFVitesse,1);

x++;

if (m_TabCtrl.m_bBoucleOuverte)		// Si l'onglet boucle ouverte est actif
{
	static int x_bo = 0;
	static int xmax_bo = 100;
	if (x_bo > xmax_bo)
	{
		xmax_bo = x_bo;
		MAJGraphBO(xmax_bo);
	}

	m_OngletBO.m_scopeBOSortCouple.PlotXY(x_bo,commande_couple*10.,0);
	m_OngletBO.m_scopeBOSortVitesse.PlotXY(x_bo,commande_vitesse*10.,0);

	x_bo++;
}

else		// Si l'onglet boucle fermé est actif
{
	static int x_bf = 0;
	static int x_bfmax = 100;
	if (x_bf > x_bfmax)
	{
		x_bfmax = x_bf;
		MAJGraphBF(x_bfmax);
	}

	m_OngletBF.m_scopeBFSortCouple.PlotXY(x_bf,commande_couple*10.,0);
	m_OngletBF.m_scopeBFSortVitesse.PlotXY(x_bf,commande_vitesse*10.,0);

	x_bf++;
}

// Enregistrement des valeurs dans un fichier.
if (m_TabCtrl.m_bBoucleOuverte && m_OngletBO.m_bEnr)
{
	fprintf(m_OngletBO.m_FicEnr,"%.2f\t%.2f\t%2.f\t%2.f\n",
				mesure_couple*10., mesure_vitesse*10.,
				commande_couple*10.,commande_vitesse*10.);
	fflush(m_OngletBO.m_FicEnr);
}

if ((!m_TabCtrl.m_bBoucleOuverte) && m_OngletBF.m_bEnr)
{
	fprintf(m_OngletBF.m_FicEnr,"%.2f\t%.2f\t%.2f\t%.2f\t%d\t%d\n",
		mesure_couple*10.,mesure_vitesse*10.,
		commande_couple*10.,commande_vitesse*10.,
		consigne_couple,consigne_vitesse);
	fflush(m_OngletBF.m_FicEnr);
}

CDialog::OnTimer(nIDEvent);
```

Appel de la fonction OnTimer

}…

En rouge, les valeurs associées à des boutons, onglet, champs de saisie…

En mode boucle ouverte :

m_OngletBO.BOCouple : Pour récupérer la valeur du couple entrée par l'utilisateur.
m_OngletBO.BOVitesse : Pour récupérer la valeur de la vitesse entrée par l'utilisateur.

En mode boucle fermé :

m_OngletBF.BFCouple : Pour récupérer la valeur du couple entré par l'utilisateur.
m_OngletBF.BFVitesse : Pour récupérer la valeur de la vitesse entrée par l'utilisateur.

Pour le couple:

m_OngletBF.Bp : coefficient Bp.
m_OngletBF.Ti : coefficient Ti.
m_OngletBF.Ka : coefficient Ka.
m_OngletBF.Td : coefficient Td.

Pour la vitesse:

m_OngletBF.Bp_v : coefficient Bp.
m_OngletBF.Ti_v : coefficient Ti.
m_OngletBF.Ka_v : coefficient Ka.
m_OngletBF.Td_v : coefficient Td.