MASTER ISTR

UE : Conception orientée objet des systèmes de commande

COMMANDE D'UN ROBOT MOBILE

Année scolaire 2023-2024

Enseignant :	P.BERTHOU
E-mail enseignant :	berthou@laas.fr
Réalisé par :	-SEGHIER Aissa et -RAHMOUN Lokmane

Table des matières

Table des figures

1. Introduction

La Programmation Orientée Objet (POO) permet de représenter des concepts, des idées ou des entités du monde physique à travers des entités appelées objets. Ces objets possèdent des propriétés intrinsèques et exposent des opérations publiques pour les manipuler.

Les objets peuvent être considérés comme des briques fournissant des services aux autres objets, les rendant ainsi réutilisables. L'interaction entre les objets via leurs relations permet de concevoir et de réaliser les fonctions attendues.

La conception revêt donc une importance cruciale pour modéliser les éléments du monde réel et les transcrire en code. Cependant, la conception reste une tâche complexe dans le développement logiciel, en raison de plusieurs facteurs :

- Les principes fondamentaux de la POO tels que l'encapsulation, l'héritage et le polymorphisme ne sont pas toujours suffisants pour guider efficacement dans la conception.
- Les design patterns, qui représentent des abstractions de solutions à des problèmes récurrents, ne parviennent pas toujours à former un ensemble cohérent pour la construction de designs complets.

Dans ce travail, nous avons cherché à appliquer les connaissances acquises lors du cours de conception orientée objet spécifiquement le c++, tout en tirant parti de notre informations aqcuis en automatique, notamment en ce qui concerne les lois de commande.

1.1. But de la manipulation

L'objectif de cette manipulation est la réalisation de schémas de pilotage élémentaires du robot Pekee. Trois types de commandes seront étudiées :

- Une commande en boucle ouverte.
- Une commande en boucle fermée.
- Une commande événementielle.

Elles seront effectuées successivement en simulation,

1.2. Matériels utilisés

- Un PC avec Windows XP et Visual C++.
- Un simulateur Maeva.
- Un outil de traçage de courbes Gnuplot.
- Matlab.

1.3. Cahier des charges

Cette manipulation se propose de synthétiser et d'analyser des mouvements en ligne droite selon la direction principale du robot.

1.4. Description du Robot

Le tableau suivant décrit les différents capteurs et actionneurs équipant notre robot pekee :

Actionneurs	Désignation
Moteur à courant continue 1	Commande roue motrice 1
Moteur à courant continue 2	Commande Roue motrice 2
Moteur à courant continue 3	Commande roue folle
Capteurs	**Désignation**
Capteurs proprioceptifs	Renseignement sur l'état interne
Odomètres	Mesure de position
Géomètres	Orientation Robot
Capteur de proximité à infra-rouge	Mesure de distance robot/obstacles
Caméra vidéo couleur	Image sur l'environnement
Capteurs de choc/lumière /température	Plus d'information sur l'environnement

Figure 1.1 – Déscription des actionneurs et des capteurs du robot.

Dans le prochain chapitre, nous entamerons la définition du diagramme des cas d'utilisation ainsi que le diagramme de classe. Ces étapes sont cruciales pour la conception de notre programme de commande du robot.

2. Conception orientée objet

La conception est une étape importante pour modéliser les éléments du monde réel et les transcrire en code. Elle implique une réflexion approfondie sur l'architecture du système, les interactions entre ses composants et les différents aspects de sa mise en œuvre.

2.1. Diagramme des cas d'utilisation (use case)

Les diagrammes de cas d'utilisation sont des diagrammes UML. C'est une vue du système qui souligne le comportement tel qu'il est perçu par les utilisateurs extérieurs. Un modèle UC partitionne les fonctionnalités du système en transaction (cas d'utilisation) qui sont significatives pour les utilisateurs (les acteurs). Le diagramme de cas d'utilisation de notre système Robot est représenté dans la figure suivante : Il permet d'identifier les possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire toutes les fonctionnalités que doit fournir le système. Il permet aussi de délimiter le système. le diagramme de USE CASE qui represente l'interaction entre l'utilisateur ,le système et le Robot est presenté dans la figure suivante :

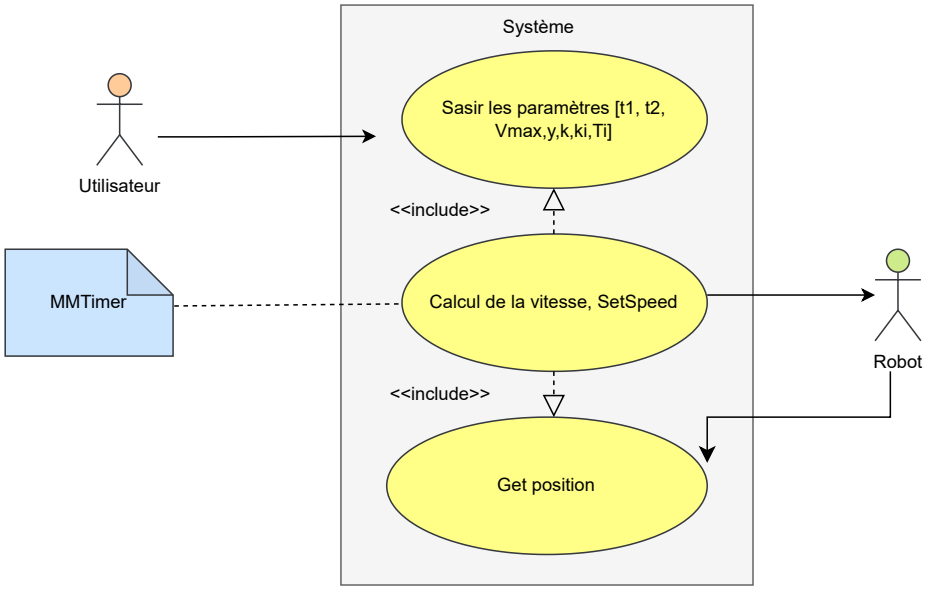

Figure 2.1 – Diagramme des cas d'utilisation.

2.2. Le diagramme de classe

Le diagramme de classes est un schéma utilisé en génie logiciel pour représenter les classes, les interfaces et leurs relations. Il fait partie de la partie statique d'UML, offrant une vue d'ensemble de la structure d'un système logiciel.
Une classe définit les responsabilités, le comportement et le type d'un ensemble d'objets, représentant ainsi l'ensemble de fonctions et d'attributs liés par un champ sémantique. Ces classes, utilisées dans la programmation orientée objet, permettent de découper une tâche complexe en travaux simples. Les relations entre les classes, telles que l'héritage et d'autres associations, sont représentées dans le diagramme de classes, facilitant la compréhension des interactions entre les différents éléments du système.
L'objectif principal de ce diagramme est d'identifier et de définir les types qui constituent un système logiciel.
Dans la figure suivante, nous présentons la représentation des éléments du système Robot et de leurs interactions à travers le diagramme de classe :

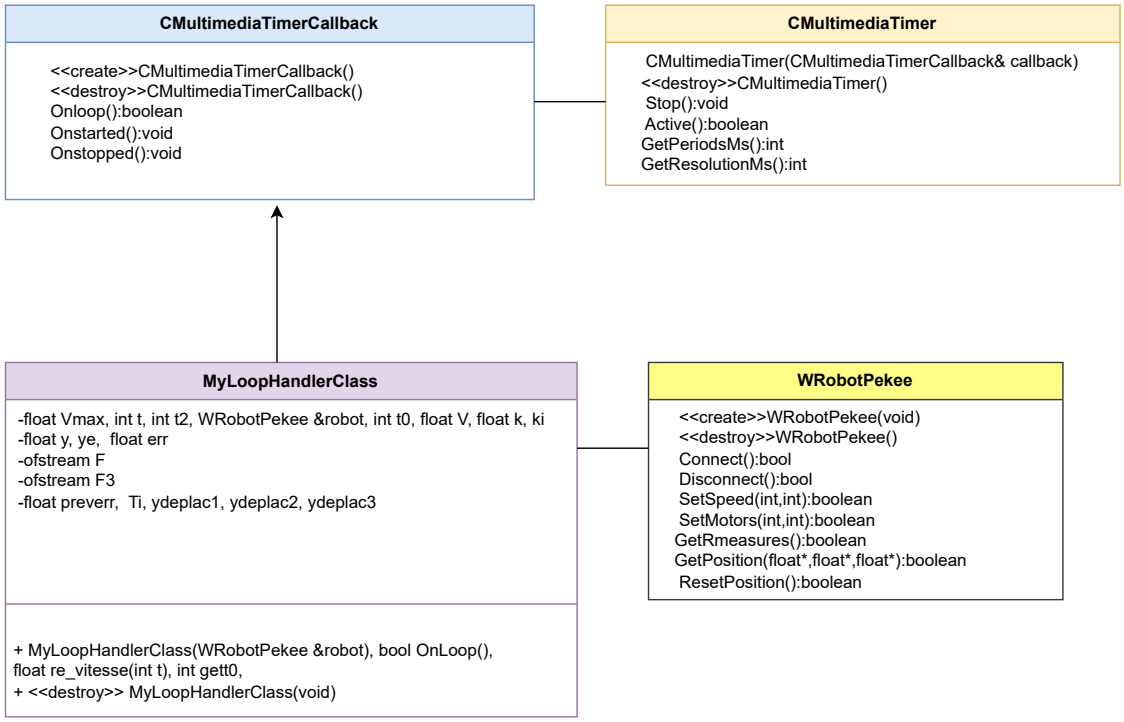

Figure 2.2 – Diagramme de classes.

2.3. Mise en oeuvre

2.3.1. Boucle Ouverte

Le but est de faire évoluer le robot selon une ligne droite de façon qu'il atteigne une position finale dite référence ou consigne.
On va écrire un programme qui en fonction des valeurs entrées de $t1$, $t2$ et $vmax$ il va applique le profil de vitesse donné ci-dessous au robot Pekee :

- le robot subit dans un premier temps un mouvement uniformément accéléré lui permettant d'atteindre une vitesse V au bout d'un délai $t1$
- Durant l'intervalle temporel $[t1, t2]$, le déplacement s'effectue à la vitesse constante V
- enfin, le robot décélère uniformément jusqu'à l'instant tf de fin du mouvement.

Les développements pour cette partie comprennent les fichiers MyLoopHandlerClass.h et MyLoopHandlerClass.cpp, ainsi que le fichier maeva.sql qui sont présentés ci-dessous :

MyLoopHandlerClass.h

```
#pragma once

// Inclusion des fichiers d'en tete nécessaires
#include "mmtimer.h"
#include "libRobotPekee.h" //fichier d'en tete pour la classe WRobotPekee

// Définition de la classe MyLoopHandlerClass
class MyLoopHandlerClass :
    public CMultimediaTimerCallback // Herite de la classe multimedia
{
private:
    float Vmax;  // Vitesse maximale
    int t1;      // Temps necessaire pour atteindre la vitesse maximale
    int t2;      // Duree pendant laquelle la vitesse maximale est maintenue
    WRobotPekee &robot; // Reference a un objet de la classe WRobotPekee
```

```
    int t0;             // Temps initial
    float V;            // Vitesse actuelle du robot

public:
    MyLoopHandlerClass(WRobotPekee &robot); // Constructeur
    bool OnLoop(); // Methode appelee lors de chaque boucle
    float re_vitesse(int t);  // Methode pour calculer la vitesse
    int gett0();  // Methode pour obtenir le temps initial

public:
    ~MyLoopHandlerClass(void);   // Destructeur
};
```

MyLoopHandlerClass.c

```
#include "stdafx.h"
#include "MyLoopHandlerClass.h"
#include <iostream>

using namespace std;

// Constructeur de la classe MyLoopHandlerClass
MyLoopHandlerClass::MyLoopHandlerClass(WRobotPekee &rob) : robot(rob)
{
    // Initialisation des paramètres du mouvement
    cout << "Veuillez entrer la valeur de t1 en (ms)" << endl;
    cin >> t1;

    cout << "Veuillez enter le valeur de t2 en (ms)" << endl;
    cin >> t2;

    cout << "Veuillez enter le valeur de Vmax (cm/s)" <<endl;
    cin >> Vmax;

    t0 = GetTickCount();  // Instant initial
}

// Destructeur de la classe MyLoopHandlerClass
MyLoopHandlerClass::~MyLoopHandlerClass(void)
{
}

// Méthode appelée lors de chaque boucle de la minuterie
bool MyLoopHandlerClass::OnLoop()
{
    int t = GetTickCount()-t0; // Calcul du temps écoulé
    cout << "le temps est:" << t<<endl;

    cout << "Vous êtes entrer dans la boucle" << endl;
    if (t<t1) // Accélération
    {
        cout << "Le robot est en accélération" << endl;
        V = Vmax*t/t1; // Calcul de la vitesse
        robot.SetSpeed(V,0); // Réglage de la vitesse du robot
    }
    else if (t<t2) // Vitesse constante
    {
        cout << "Le robot est en vitesse constante" <<endl;
        V = Vmax;
        robot.SetSpeed(V,0);
```

```
    }
    else if(t<(t1+t2)) // Décélération
    {
        cout << "Le robot est en décélération" <<endl;
        V = Vmax - Vmax*(t-t2)/t1; // Calcul de la vitesse décroissante
        robot.SetSpeed(V,0);
    }
    else // Arrêt
    {
        cout << "Le robot est à l'arrêt" <<endl;
        V = 0;
        robot.SetSpeed(V,0);
        exit(0);
    }

    // cout << "------------------------" <<endl;

    return true;
}

// Méthode pour calculer la vitesse en fonction du temps
float MyLoopHandlerClass::re_vitesse(int t)
{
    if (t<t1)
    {
        cout << "Le robot est en accélération" << endl;
        return Vmax*t/t1;
    }
    else if (t<t2)
    {
        cout << "Le robot est en vitesse constante" <<endl;
        return Vmax;
    }
    else if(t<(t1+t2))
    {
        cout << "Le robot est en décélération" <<endl;
        return (Vmax - Vmax*(t-t2)/t1);
    }
    else
    {
        cout << "Le robot est à l'arrêt" <<endl;
        return 0;
    }
}

// Méthode pour obtenir le temps initial
int MyLoopHandlerClass::gett0()
{
    return t0;
}
```

maevasql

```
    // maeva_sql.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>

#include "libRobotPekee.h"
```

```
#include "MyLoopHandlerClass.h"

#include <fstream>

using namespace std;

//Declarations globales au programme
//

WRobotPekee Pekee;

// -- Commenter en fonction pour selectionner le mode de commande
// Configuration mode Simulation
BUFFER experience="simulation sur le RSL.";
BUFFER szRSLHostName="localhost";

// Configuration mode Reel
//BUFFER experience="commande du robot reel.";
//BUFFER szRSLHostName="130.120.12.101";


int main(int argc, char* argv[])
{
        MyLoopHandlerClass handler(Pekee);
        CMultimediaTimer timer(handler);
        timer.Start(20,1);

        printf("Coucou\n");

        // Connexion au robot
        //
        std::cout << "Essai de connection a " << szRSLHostName << std::endl;
        if(Pekee.Connect(szRSLHostName))
    {
                //std::cout << "La connexion est etablie" << std::endl;
                float t;
                ofstream fichier("trajectoire.txt");
                cout << "Ecriture dans un fichier" << endl;

                if (fichier.is_open() == false)
                {
                        cout << "Erreur lors de l'ouverture du fichier" <<endl;
                        exit(0);
                }

                while (1)
                {
                        //Pekee.SetSpeed(100,0) ;
                        //
                        Sleep(1);
                        t = GetTickCount();
                        t = t - handler.gett0();
                        fichier <<t<<"    "<< handler.re_vitesse(t) <<endl;
                }

          // --------------------------------- Initialisation du Timer

          // --------------------------------- Tracage de la Position
```

```
            // ---------------------------------- Arret du Timer

            // Arret de la connexion au Robot
        fichier.close();
      Pekee.Disconnect();
    }
        else
    {
                //std::cout << "La connexion n'a pu etre etablie."<< std::endl;
    }
        return 0;
}
```

Note :

Dans le cadre de notre travail, nous avons opté pour l'utilisation de l'outil Matlab afin de tracer les courbes de la boucle ouverte et des étapes suivantes. Pour ce faire, nous avons importé le fichier texte généré par le code C++. Cette méthode a été choisie pour améliorer la visibilité et simplifier l'analyse des données. Les graphiques obtenus sont de haute qualité, ce qui facilite grandement l'interprétation des résultats.

Le code utilisé sous matlab pour le tracé est le suivant :

```
%étape 1 : Importer les données
%data = load('trajectoirBO.txt');
%data = load('trajectoireBF.txt');
%data = load('trajectoirebfkp.txt');
data = load('trajectoiresuivikp.txt');

%  tape  2 : Traiter les données
x = data(:, 1); % Supposons que la première colonne est l'axe x
y = data(:, 2); % Supposons que la deuxième colonne est l'axe y

%  tape  3 : Tracer les données
plot(x, y); % Tracé des données
xlabel('Temps'); % Ajouter une étiquette à l'axe des x
ylabel('Distance'); % Ajouter une étiquette à l'axe des y
%title('Tajectoire du robot en boucle ouverte '); % Ajouter un titre au tracé
%title('Tajectoire du robot en boucle fermé '); % Ajouter un titre au tracé
%title('Tajectoire du robot en boucle fermé avec un gain kp  ');
title('Suivi de trajectoire avec un gain kp');
grid on; % Activer la grille
```

On prend t1=2000ms, t2=8000ms et Vmax=200cm/s nous obtenons le tracé suivant :

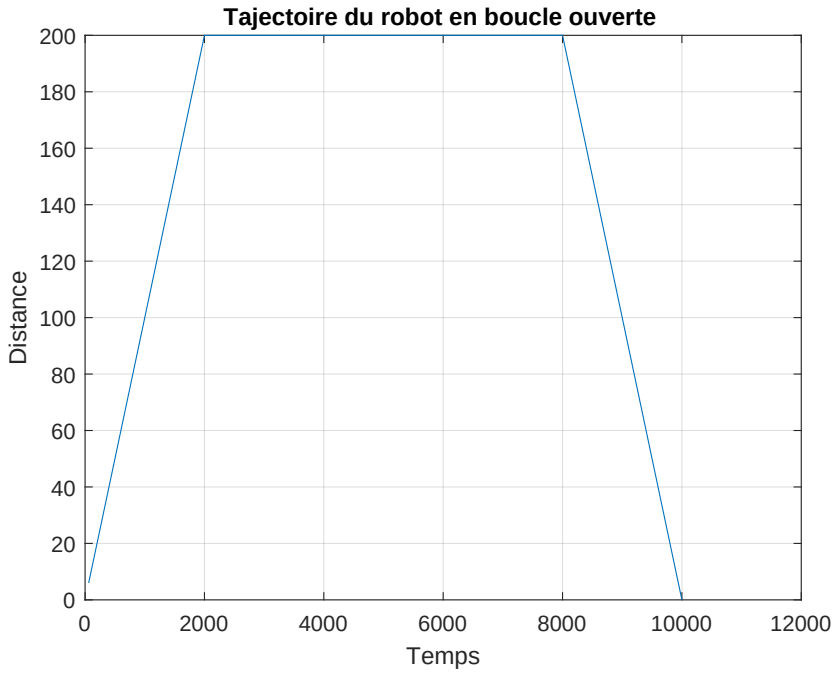

Figure 2.3 – Trajectoire du robot en boucle ouverte.

À partir du tracé obtenu, on peut bien identifier que la trajectoire du robot est divisée en trois zones : la première pour l'accélération, la deuxième pour la vitesse constante et la troisième pour la décélération.

2.3.2. Boucle Fermée

Afin de pallier aux éventuelles perturbations que pourrait subir le robot, on va mettre en œuvre une commande en « boucle fermée » basée sur les mesures retournées par les odomètres liés aux roue.

2.3.3. Retour unitaire

Le type de cette commande devra être : u(t) = k(y*y(t)) avec u(t) la vitesse commandée.
Nous avons ensuite testé cet asservissement de position avec y*=1 mètre et k=1, Les développements pour cette partie comprennent les fichiers MyLoopHandlerClass.h et MyLoopHandlerClass.cpp, ainsi que le fichier maeva.sql qui sont présentés ci-dessous :

MyLoopHandlerClass.h

```
#pragma once
#include "mmtimer.h"
#include "libRobotPekee.h"
#include <iostream>
#include <fstream>

using namespace std;
class MyLoopHandlerClass :
        public CMultimediaTimerCallback
{

private:
        float Vmax;
        int  t1;
        int  t2;
        WRobotPekee &robot;
        int  t0;
        float V ;
```

```
        float x;
        float theta;
        float k;
        float y, ye;
        ofstream F;

public:
        MyLoopHandlerClass(WRobotPekee &robot);
        bool OnLoop();
        float re_vitesse(int t);
        int gett0();

public:
        ~MyLoopHandlerClass(void);
};
```

MyLoopHandlerClass.c

```
#include "stdafx.h"
#include "MyLoopHandlerClass.h"
#include <iostream>

using namespace std;


MyLoopHandlerClass::MyLoopHandlerClass(WRobotPekee &rob): robot(rob)
{
        //Boucle fermee

        cout << "Entrer la valeur de Y " << endl;
        cin >> y;
        cout << "Entrer la valeur de k " << endl;
        cin >> k;



        // Initialisation
        t0 = GetTickCount();
        V = 0;
        ye = 0;
        F.open("trajectoire2.txt");
}

MyLoopHandlerClass::~MyLoopHandlerClass(void)
{
        F.close();
}

bool MyLoopHandlerClass::OnLoop()
{

         int t = GetTickCount() - t0;


                robot.GetPosition(ye, x, theta);
                cout << "la position est :" << ye << " ; " << x << endl;
                V = k * (y - ye);
                cout << "La vitesse du robot = " << V << endl;
                robot.SetSpeed(V,0);
```

```
                F << t << " " << ye << endl;

                if ((y-ye)<0.005)
                {
                        exit(0);
                }

        return true;

}




float MyLoopHandlerClass::re_vitesse(int t)
{

        if (t<t1) {

                cout << " Le robot est en accélération " << endl;
                return ( Vmax / t1 )* t;



        }
        else if ( t < t2) {
                cout << " Le robot est en vitesse constatne " << endl;
                return Vmax;



        }
        else if ( t < (t1 + t2)) {
                cout << " Le robot est en décélération " << endl;
                return Vmax - (Vmax * (t-t2)/t1);


        }

        else {
                cout << " Le robot est en arrét " << endl;
            return 0;
        }
}


int  MyLoopHandlerClass::gett0()
{
        return t0;
}
```

maevasql

```
// maeva_sql.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
```

```
#include "MyLoopHandlerClass.h"
#include "libRobotPekee.h"
#include "mmtimer.h"

#include <fstream>

using namespace std;

//Declarations globales au programme
//
float x,y,theta;

WRobotPekee Pekee;

// -- Commenter en fonction pour selectionner le mode de commande
// Configuration mode Simulation
BUFFER experience="simulation sur le RSL.";
BUFFER szRSLHostName="localhost";

// Configuration mode Reel
//BUFFER experience="commande du robot reel.";
//BUFFER szRSLHostName="130.120.12.101";


int main(int argc, char* argv[])
{

        MyLoopHandlerClass handler(Pekee);
        CMultimediaTimer timer(handler);
        timer.Start(20,1);

        printf("Coucou\n");

        // Connexion au robot
        //
        std::cout << "Essai de connection a " << szRSLHostName << std::endl;
        if(Pekee.Connect(szRSLHostName))
    {
                //std::cout << "La connexion est etablie" << std::endl;



                while (1)
                {Sleep(1); //Pekee.SetSpeed(100,0) ;


                }

          // ---------------------------------- Initialisation du Timer

          // ---------------------------------- Tracage de la Position

          // ---------------------------------- Arret du Timer

          // Arret de la connexion au Robot

       Pekee.Disconnect();
    }
        else
```

```
    {
                //std::cout << "La connexion n'a pu etre etablie." << std::endl;
    }
        return 0;
}
```

Après avoir réalisé l'asservissement, nous traçons la courbe de la trajectoire de notre système en boucle fermée et nous obtenons le résultat suivant :

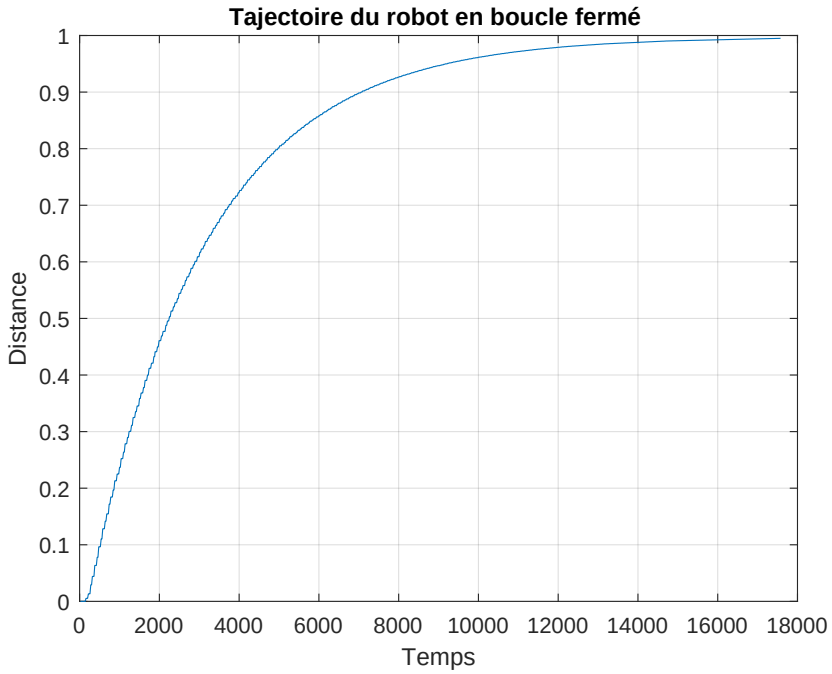

Figure 2.4 – Tracé de la boucle fermée.

2.3.4. Boucle Fermée avec un gain kp

Afin d'améliorer la réponse obtenu est dimunié l'erreur on va untiliser un gain propotionnel kp=200 :

MyLoopHandlerClass.h

```
#pragma once
#include "mmtimer.h"
#include "libRobotPekee.h"
#include <iostream>
#include <fstream>

using namespace std;
class MyLoopHandlerClass :
        public CMultimediaTimerCallback
{

private:
        float Vmax;
        int  t1;
        int  t2;
        WRobotPekee &robot;
        int  t0;
        float V ;
        float x;
        float theta;
        float k, ki;
        float y, ye;
        float err;
```

```
        ofstream F;
        ofstream F3;
        float preverr;
        float Ti;

public:
        MyLoopHandlerClass(WRobotPekee &robot);
        bool OnLoop();
        float re_vitesse(int t);
        int gett0();

public:
        ~MyLoopHandlerClass(void);
};
```

MyLoopHandlerClass.c

```
#include "stdafx.h"
#include "MyLoopHandlerClass.h"
#include <iostream>

using namespace std;


MyLoopHandlerClass::MyLoopHandlerClass(WRobotPekee &rob): robot(rob)
{
        //Boucle fermee

        cout << "Entrer la valeur de Y " << endl;
        cin >> y;
        cout << "Entrer la valeur de k " << endl;
        cin >> k;
        cout << "Entrer la valeur de ki " << endl;
        cin >> ki;
        cout << "Entrer le temps d'échantillonage " << endl;
        cin >> Ti;



        // Initialisation
        t0 = GetTickCount();
        V = 0;
        ye = 0;
        err = 0;
        preverr = 0;
        F.open("trajectoire2.txt");

}

MyLoopHandlerClass::~MyLoopHandlerClass(void)
{
        F.close();

}

bool MyLoopHandlerClass::OnLoop()
{

         int t = GetTickCount() - t0;
```

```
                robot.GetPosition(ye, x, theta);
                cout << "la position est :" << ye << " ; " << x << endl;
                err = y - ye;
// pour tracer la trajectoire de la boucle fermé avec seulemnt un gain Kp
//on prend Kp=200; ki=0; et la période Ti=10
                V = V+ k * (err -preverr)+((Ti*ki)/2)*(err + preverr);
                cout << "La vitesse du robot = " << V << endl;
                robot.SetSpeed(V,0);
                preverr = err;

                F << t << " " << ye << endl;


                if ((y-ye)<0.01)
                {
                        exit(0);
                }

        return true;

}







float MyLoopHandlerClass::re_vitesse(int t)
{

        if (t<t1) {

                cout << " Le robot est en accélération " << endl;
                return ( Vmax / t1 )* t;



        }
        else if ( t < t2) {
                cout << " Le robot est en vitesse constatne " << endl;
                return Vmax;



        }
        else if ( t < (t1 + t2)) {
                cout << " Le robot est en décélération " << endl;
                return Vmax - (Vmax * (t-t2)/t1);


        }

        else {
                cout << " Le robot est en arrét " << endl;
            return 0;
        }
}
```

```
int  MyLoopHandlerClass::gett0()
{
        return t0;
}
```

maevasql

```
// maeva_sql.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include "MyLoopHandlerClass.h"
#include "libRobotPekee.h"
#include "mmtimer.h"

#include <fstream>

using namespace std;

//Declarations globales au programme
//
float x,y,theta;

WRobotPekee Pekee;

// -- Commenter en fonction pour selectionner le mode de commande
// Configuration mode Simulation
BUFFER experience="simulation sur le RSL.";
BUFFER szRSLHostName="localhost";

// Configuration mode Reel
//BUFFER experience="commande du robot reel.";
//BUFFER szRSLHostName="130.120.12.101";


int main(int argc, char* argv[])
{

        MyLoopHandlerClass handler(Pekee);
        CMultimediaTimer timer(handler);
        timer.Start(20,1);

        printf("Coucou\n");

        // Connexion au robot
        //
        std::cout << "Essai de connection a " << szRSLHostName << std::endl;
        if(Pekee.Connect(szRSLHostName))
    {
                //std::cout << "La connexion est etablie" << std::endl;



                while (1)
                {Sleep(1); //Pekee.SetSpeed(100,0) ;


                }
```

```
            // ---------------------------------- Initialisation du Timer

            // ---------------------------------- Tracage de la Position

            // ---------------------------------- Arret du Timer

            // Arret de la connexion au Robot

        Pekee.Disconnect();
    }
         else
    {
//std::cout << "La connexion n'a pu etre etablie." << std::endl;
    }
         return 0;
}
```

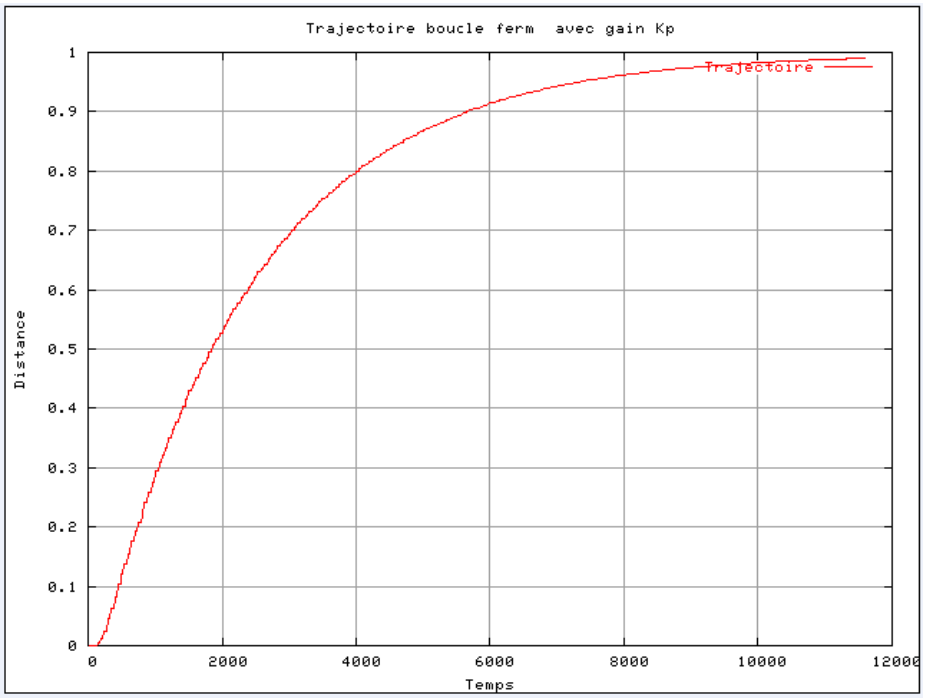

Figure 2.5 – Tracé de la boucle fermée avec gain kp.

Si l'on compare le résultat obtenu avec le retour unitaire et le résultat précédent, il est clair que notre système est devenu plus rapide. Auparavant, il atteignait la consigne en 18000 ms, alors qu'actuellement il le fait en 12000 ms. Cette amélioration est due à l'efficacité de notre régulateur kp

2.4. Suivit de trajectoire

On souhaite maintenant utiliser les résultats précédemment obtenus afin d'améliorer la commande en vitesse. Le problème s'apparente donc à un suivi de trajectoire dont la trajectoire (le profil de vitesse) est prédéterminée. Pour cela, on combinera les deux commandes précédentes, avec y* qui n'est plus une constante, mais fonction de la commande en boucle ouverte.
Pour un premier temps, on va essayer de faire le suivi en utilisant seulement un gain proportionnel. Les codes développés pour cette partie sont :
MyLoopHandlerClass.h

```
#pragma once
#include "mmtimer.h"
#include "libRobotPekee.h"
#include <iostream>
```

```
#include <fstream>

using namespace std;
class MyLoopHandlerClass :
        public CMultimediaTimerCallback
{

private:
        float Vmax;
        int  t1;
        int  t2;
        WRobotPekee &robot;
        int  t0;
        float V ;
        float x;
        float theta;
        float k, ki;
        float y, ye;
        float err;
        ofstream F3;
        float preverr;
        float ydeplac1, ydeplac2, ydeplac3;

public:
        MyLoopHandlerClass(WRobotPekee &robot);
        bool OnLoop();
        float re_vitesse(int t);
        int gett0();

public:
        ~MyLoopHandlerClass(void);
};
```

MyLoopHandlerClass.c

```
#include "stdafx.h"
#include "MyLoopHandlerClass.h"
#include <iostream>

using namespace std;
float ydeplac1, ydeplac2, ydeplac3;
int tf;


MyLoopHandlerClass::MyLoopHandlerClass(WRobotPekee &rob): robot(rob)
{


    cout << "Veuillez entrer la valeur de t1 en (ms)" << endl;
        cin >> t1;

        cout << "Veuillez enter le valeur de t2 en (ms)" << endl;
        cin >> t2;

        cout << "Veuillez enter le valeur de Vmax (cm/s)" <<endl;
        cin >> Vmax;
        cout << "Entrer la valeur de Y " << endl;
        cin >> y;
        cout << "Entrer la valeur de k " << endl;
        cin >> k;
        cout << "Entrer la valeur de ki " << endl;
```

```
        cin >> ki;
        cout << "Entrer le temps d'échantillonage " << endl;
        cin >> Ti;



        // Initialisation
        t0 = GetTickCount();
        V = 0;
        ye = 0;
        err = 0;
        preverr = 0;
        F3.open("trajectoire3.txt");
}
// fonction pour calculer les déplacement dans les trois modes
void ydeplac(int t1, int t2, int Vmax) {
                tf = 2*t1 + t2;
                ydeplac1 = (Vmax*t1/2)*0.001;
                ydeplac2 = (Vmax*(t2 - t1))*0.001;
                ydeplac3 = ydeplac1;
cout << "le déplacement1:" << ydeplac1 <<"le déplacement2 \n:"
<< ydeplac2 << "le déplacement3: \n" << ydeplac3;
        }

MyLoopHandlerClass::~MyLoopHandlerClass(void)
{

        F3.close();
}

bool MyLoopHandlerClass::OnLoop()
{

         int t = GetTickCount() - t0;
         robot.GetPosition(ye, x, theta);
         cout << "la position est :" << ye << " ; " << x << endl;
         V = k *(y - ye);
         cout << "La vitesse du robot = " << V << endl;
                F3 << t << " " << ye << endl;


         cout << "Vous étes entrer dans la boucle" << endl;
         ydeplac( t1,  t2,  Vmax);

        if (t<t1)
        {
                cout << "Le robot est en acceleration" << endl;
                V = k *(ye - ydeplac1);
                 cout << "La vitesse 1 du robot = " << V << endl;
                robot.SetSpeed(V,0);
        }
        else if (t<t2)
        {
                cout << "Le robot est en vitesse constante" <<endl;
                V = k *(ye - ydeplac2);
                 cout << "La vitesse 2 du robot = " << V << endl;
                robot.SetSpeed(V,0);
        }
        else if(t<(t1+t2))
        {
```

```
                cout << "Le robot est en deceleration" <<endl;
                V = k *(ye - ydeplac3);
                cout << "La vitesse 3 du robot = " << V << endl;
                robot.SetSpeed(V,0);
        }
        else
        {
                cout << "Le robot est en arret" <<endl;
                V = 0;
                robot.SetSpeed(V,0);
        }
        F3 << t << " " << ye << endl;
        cout << "l'erreur " << y-ye << endl;
        // t1=1000 t2=1000 k=1 V=100
        if ((y-ye)<0.01)
                {
                        //exit(0);
                }

        return true;

}



float MyLoopHandlerClass::re_vitesse(int t)
{

        if (t<t1) {

                cout << " Le robot est en accélération " << endl;
                return ( Vmax / t1 )* t;



        }
        else if ( t < t2) {
                cout << " Le robot est en vitesse constatne " << endl;
                return Vmax;



        }
        else if ( t < (t1 + t2)) {
                cout << " Le robot est en décélération " << endl;
                return Vmax - (Vmax * (t-t2)/t1);


        }

        else {
                cout << " Le robot est en arrét " << endl;
            return 0;
        }
}


int  MyLoopHandlerClass::gett0()
{
        return t0;
```

```
}
```

maevasql.

```
    // maeva_sql.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>
#include "MyLoopHandlerClass.h"
#include "libRobotPekee.h"
#include "mmtimer.h"

#include <fstream>

using namespace std;

//Declarations globales au programme
//
float x,y,theta;

WRobotPekee Pekee;

// -- Commenter en fonction pour selectionner le mode de commande
// Configuration mode Simulation
BUFFER experience="simulation sur le RSL.";
BUFFER szRSLHostName="localhost";

// Configuration mode Reel
//BUFFER experience="commande du robot reel.";
//BUFFER szRSLHostName="130.120.12.101";


int main(int argc, char* argv[])
{

        MyLoopHandlerClass handler(Pekee);
        CMultimediaTimer timer(handler);
        timer.Start(20,1);

        printf("Coucou\n");

        // Connexion au robot
        //
        std::cout << "Essai de connection a " << szRSLHostName << std::endl;
        if(Pekee.Connect(szRSLHostName))
    {
                //std::cout << "La connexion est etablie" << std::endl;



                while (!kbhit())
                {Sleep(1); //Pekee.SetSpeed(100,0) ;


                }

          // --------------------------------- Initialisation du Timer

          // --------------------------------- Tracage de la Position
```

```
            // ---------------------------------- Arret du Timer

            // Arret de la connexion au Robot

        Pekee.Disconnect();
    }
         else
    {
                //std::cout << "La connexion n'a pu etre etablie." << std::endl;
    }
         return 0;
}
```

Pour t1=1000ms, t2=1000ms , k=1 et v=100cm/s nous avons obtenu le résultat suivant :

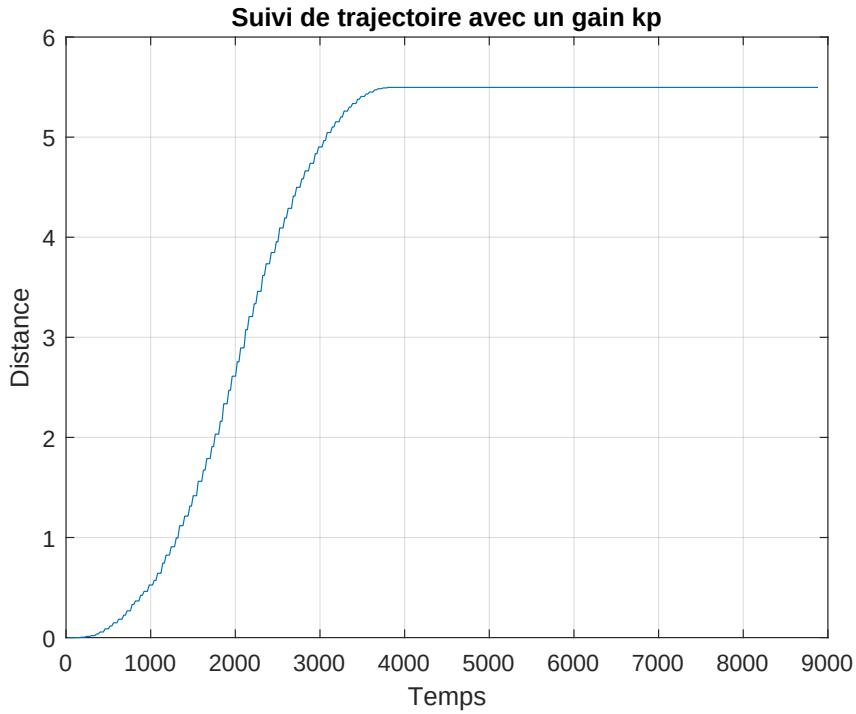

Figure 2.6 – Suivit de trajectoire avec gain kp.

2.4.1. Suivit de trajectoire avec un gain kp ki

Afin d'améliorer les performances de notre contrôleur, on va ajouter l'action intégrale pour réduire l'erreur. Dans ce cas, on va effectuer les modifications suivantes :

```
#include "stdafx.h"
#include "MyLoopHandlerClass.h"
#include <iostream>

using namespace std;
float ydeplac1, ydeplac2, ydeplac3;
int tf;


MyLoopHandlerClass::MyLoopHandlerClass(WRobotPekee &rob): robot(rob)
{
```

```
    cout << "Veuillez entrer la valeur de t1 en (ms)" << endl;
        cin >> t1;

        cout << "Veuillez enter le valeur de t2 en (ms)" << endl;
        cin >> t2;

        cout << "Veuillez enter le valeur de Vmax (cm/s)" <<endl;
        cin >> Vmax;
        cout << "Entrer la valeur de Y " << endl;
        cin >> y;
        cout << "Entrer la valeur de k " << endl;
        cin >> k;
        cout << "Entrer la valeur de ki " << endl;
        cin >> ki;
        cout << "Entrer le temps d'échantillonage " << endl;
        cin >> Ti;



        // Initialisation
        t0 = GetTickCount();
        V = 0;
        ye = 0;
        err = 0;
        preverr = 0;
        F3.open("trajectoire3.txt"); // Pour le suivi
}

void ydeplac(int t1, int t2, int Vmax) {
                tf = 2*t1 + t2;
                ydeplac1 = (Vmax*t1/2)*0.001;
                ydeplac2 = (Vmax*(t2 - t1))*0.001;
                ydeplac3 = ydeplac1;
  cout << "le déplacement1:" << ydeplac1<<"le déplacement2 \n:"
  << ydeplac2 << "le déplacement3: \n" << ydeplac3;
        }

MyLoopHandlerClass::~MyLoopHandlerClass(void)
{

        F3.close();
}

bool MyLoopHandlerClass::OnLoop()
{

         int t = GetTickCount() - t0;
         robot.GetPosition(ye, x, theta);
         cout << "la position est :" << ye << " ; " << x << endl;
         V = k *(y - ye)+ ki*(ye-y);
         cout << "La vitesse du robot = " << V << endl;
                F3 << t << " " << ye << endl;


         cout << "Vous étes entrer dans la boucle" << endl;
         ydeplac( t1,  t2,  Vmax);

        if (t<t1)
        {
                cout << "Le robot est en acceleration" << endl;
```

```
                V = k *(ye - ydeplac1)+ki*((Vmax*t1/2)*t2/2);
                 cout << "La vitesse 1 du robot = " << V << endl;
                robot.SetSpeed(V,0);
        }
        else if (t<t2)
        {
                cout << "Le robot est en vitesse constante" <<endl;
                V = k *(ye - ydeplac2)+ki*(Vmax*(t2-t1)*t);
                 cout << "La vitesse 2 du robot = " << V << endl;
                robot.SetSpeed(V,0);
        }
        else if(t<(t1+t2))
        {
                cout << "Le robot est en deceleration" <<endl;
                V = k *(ye - ydeplac3)+ki*((Vmax*t1/2)*t2/2);
                cout << "La vitesse 3 du robot = " << V << endl;
                robot.SetSpeed(V,0);
        }
        else
        {
                cout << "Le robot est en arret" <<endl;
                V = 0;
                robot.SetSpeed(V,0);
        }
        F3 << t << " " << ye << endl;
        cout << "l'erreur " << y-ye << endl;
        // t1=1000 t2=1000 k=1 V=100
        if ((y-ye)<0.01)
                {
                        //exit(0);
                }


        return true;

}

float MyLoopHandlerClass::re_vitesse(int t)
{

        if (t<t1) {

                cout << " Le robot est en accélération " << endl;
                return ( Vmax / t1 )* t;



        }
        else if ( t < t2) {
                cout << " Le robot est en vitesse constatne " << endl;
                return Vmax;



        }
        else if ( t < (t1 + t2)) {
                cout << " Le robot est en décélération " << endl;
                return Vmax - (Vmax * (t-t2)/t1);
```

```
        }

        else {
                cout << " Le robot est en arrét " << endl;
            return 0;
        }
}


int  MyLoopHandlerClass::gett0()
{
        return t0;
}
```

2.5. Conclusion

Dans ce travail pratique, nous avons eu l'occasion de découvrir les concepts puissants de la programmation orientée objet, offerts par C++, et d'en exploiter les avantages en termes de modularité et de portabilité. De plus, en utilisant un simulateur, nous avons concrètement mis en relation ces concepts avec le domaine de la robotique. En parallèle, nous avons appliqué nos connaissances en automatique, en mettant en œuvre des techniques de contrôle telles que la commande en boucle fermée et l'utilisation des régulateurs proportionnel et proportionnel-intégral. En adoptant l'approche orientée objet avec C++, nous avons également bénéficié de la flexibilité de l'héritage, ce qui nous a permis de structurer notre code de manière plus modulaire, réutilisable et facilement maintenable.