**IUT Sénart/Fontainebleau **

**Avenue Pierre Point**

**77567 Lieusaint cedex**

**M. Sabourin Christophe**

**Contribution à la mise en œuvre de TP**

**en informatique industrielle et robotique**

**Girard Benjamin Université Paris Est Créteil**

**Formation initiale IUT Sénart/ Dpt GEII**

**Année universitaire 2009/2010 Tuteur : Mme Garric Dominique**

**Stage du 12 /04/10 au 25/06/10**

**Remerciements :**

Je tiens à remercier Christophe Sabourin, mon maître de stage, pour m’avoir accordé sa confiance dans cette mission.

Je remercie aussi, Dominik ramik Doctorant au LISSI, pour toute l’aide qu’il m’a apporté sur les problèmes rencontrés.

Mais aussi le reste de l’équipe du LISSI, pour la bonne ambiance, ainsi que mon camarade stagiaire, Fabien Gautero, pour ses conseils pratiques.

Sommaire

[Introduction 4](#_Toc264979705)

[I – Présentation de l’entreprise 5](#_Toc264979706)

[1 – L’IUT de Sénart/Fontainebleau 5](#_Toc264979707)

[2 – Le département GEII 6](#_Toc264979708)

[II – Présentation du projet 7](#_Toc264979709)

[1 – Quelques mots sur le projet 7](#_Toc264979710)

[2 – Le cahier des charges 8](#_Toc264979711)

[III – Le Projet 9](#_Toc264979712)

[1 – Travaux réalisés sur le robot e-puck 9](#_Toc264979713)

[A – Présentation de webots et du robot e-puck 9](#_Toc264979714)

[B – Sujet de TP N°1 : l’odométrie 11](#_Toc264979715)

[C – Sujet de TP N°2 : Suivi de la balle 14](#_Toc264979716)

[D – Rédaction des TP 16](#_Toc264979717)

[E – bilan 17](#_Toc264979718)

[3 – Le bras-robot Lynx6 17](#_Toc264979719)

[A – Présentation de Lynx6 17](#_Toc264979720)

[B – Travaux réalisé sur Lynx6 19](#_Toc264979721)

[IV – Conclusion 22](#_Toc264979722)

[V – Bibliographie 23](#_Toc264979723)

[ANNEXES 24](#_Toc264979731)

# 

# Introduction

Ce rapport est le fruit de l’investissement du travail réalisé pendant mon stage de fin d’étude dans le cadre d’un diplôme DUT GEII (génie électrique et informatique industrielle). Stage qui s’est déroulé du 12 avril au 25 juin 2010 au sein même de département GEII de l’IUT de Sénart/Fontainebleau.

La mission que l’on m’a confiée était de contribuer à la mise en œuvre de TP en informatique industrielle et robotique. J’ai choisi ce stage, non seulement pour son coté pédagogique (élaboration, écriture et mise en place des TP puis encadrement) qui m’attire par curiosité mais aussi le coté informatique industrielle (Systèmes embarqués, programmation) qui est la principale motivation de mon projet professionnel.

Durant ce stage, j’ai réalisé deux sujets de TP sur le robot e-puck dont un en simulation sur le logiciel Webots et l’autre qui intègre la simulation sur Webots et l’application sur le modèle réel. J’ai construis et mis en service deux bras mécaniques Lynx 6, et réalisé la préparation d’un projet tuteuré pour l’année prochaine.

# I – Présentation de l’entreprise

## 1 – L’IUT de Sénart/Fontainebleau

Le premier département (Techniques de commercialisation) de l’IUT de Sénart/Fontainebleau à ouvert en 1988 sur le site de Sénart. C’est en 1990 que le département Génie industriel et maintenance à ouvert suivit du département Génie des entreprises et des administrations en 1992 et du département Génie électrique et informatique industrielle en 1993. En 2007 ouvre le département Carrières Sociales.

Le site de Fontainebleau compte 2 départements, celui du Génie des entreprises (1993) et Informatique (1995).

Voici l’organigramme de l’organisation actuelle de l’IUT Sénart

Figure 1: Organigramme de l'IUT de Sénart

Le site de Sénart/Fontainebleau offre 7 formations en DUT mais aussi 18 licences professionnelles regroupés en 4 pôles :

* -Gestion, Commerce, Management
* -Economie sociale et solidaire
* -Maintenance, Génie électrique
* -Informatique et Réseaux

L’IUT offre un large panel de formation avec la possibilité de suivre en initiale, continue ou en alternance. Des partenariats sont créés avec les entreprises, des liens internationaux s’ouvrent, l’IUT apporte une large vision sur les poursuites d’études. Le département GEII contribue à cette pleine expansion et au développement depuis plus de dix ans.

## 2 – Le département GEII

Le Département Génie Electrique et Informatique Industrielle forme des Techniciens Supérieurs en Génie Electrique (Electronique analogique et numérique, Electrotechnique, Electronique de Puissance) et en informatique industrielle (Automatismes, régulation, Microcontrôleurs, Langages informatiques et systèmes d'exploitation, réseaux locaux, systèmes embarqués).  
  
Le Diplôme Universitaire de Technologie (DUT) de Génie Electrique et Informatique Industrielle (GEII) est un diplôme national sanctionnant une formation qui est à la fois ciblée (sur les domaines de l'électricité) et suffisamment large de façon à :

- permettre l'adaptation des diplômés à une grande variété d'emplois    
- faciliter leur mobilité et toute évolution ultérieure de carrière   
- leur donner la possibilité d'une poursuite d'études.

Des projets tuteurés sont proposé en deuxième année offrant aux étudiants la possibilité de travailler sur un projet du domaine de leur choix comme par exemple, le kart, le robot NAO.



Figure 2 photo de NAO

Au 4éme semestre, des parcours orientés vers différents domaines tel-que l’informatique industrielle, l’informatique de gestion, l’énergie sont proposés. Ces parcours permettent un approfondissement du domaine choisi pour la future orientation. On y trouve :

* Le parcours informatique industriel : programmation d’un robot, programmation d’un PDA et connexion entre les deux.
* Le parcours informatique gestion : base de données, JAVA.
* Le parcours qualité d’énergie
* Le parcours ouverture scientifique : physique des composants, électrostatique, probabilités.

# II – Présentation du projet

## 1 – Quelques mots sur le projet

Aujourd’hui, les applications de la robotique sont nombreuses et continuent de se diversifier. De la chaine de montage complètement automatisée utilisée dans l’industrie, à la robotique médicale permettant d’assister le chirurgien, en passant par les robots spatiaux, les robots intègrent petit à petit notre quotidien. Initialement sous forme de jouets destinés à des applications ludiques, les robots personnels vont progressivement nous assister dans nos tâches quotidiennes. La plus connue des applications est très certainement le robot aspirateur (figure 3) que de nombreuses marques proposent aujourd’hui.



Figure 3 Robot aspirateur

Dans les prochaines années, la robotique va très certainement prendre une place de plus en plus importante dans notre société. Par conséquent, outre le support pédagogique que peut représenter un robot en GEII, la conception et la mise en œuvre des robots nécessitera, dans un avenir proche, des connaissances spécifiques.

Depuis plusieurs années, le département GEII de l’IUT de Sénart-Fontainebleu a décidé de mettre en avant de sa formation certains aspects de la robotique. Ceci s’est notamment concrétisé par la mise en place de modules en première et deuxième année ayant comme support des petits robots mobiles ainsi que des projets tuteurés (Coupe robotique GEII, robot NAO). Afin de continuer à promouvoir la robotique au sein du GEII, il est aujourd’hui nécessaire de développer de nouvelles applications.

## 2 – Le cahier des charges

Objectifs :

* Réalisé deux TP sur le robot-epuck en utilisant la plateforme de simulatuion Webots
* Mettre en service les deux bras robotisé lynx6
* Tester la communiquer par port série du bras-robot avec un PIC
* Mettre en place un environnement pour un projet tuteuré l’année prochaine

Contraintes :

* L’étudiant doit être capable de faire le TP en autonomie et comprendre clairement ce qui lui est demandé.

# III – Le Projet

## 1 – Travaux réalisés sur le robot e-puck

### A – Présentation de webots et du robot e-puck

Webots est un logiciel développé par Cyberbotics (www.cyberbotics.com) qui permet de simuler dans un environnement 3D l’évolution d’un robot en fonction du programme que vous avez codé. Webots se présente sous cette forme.

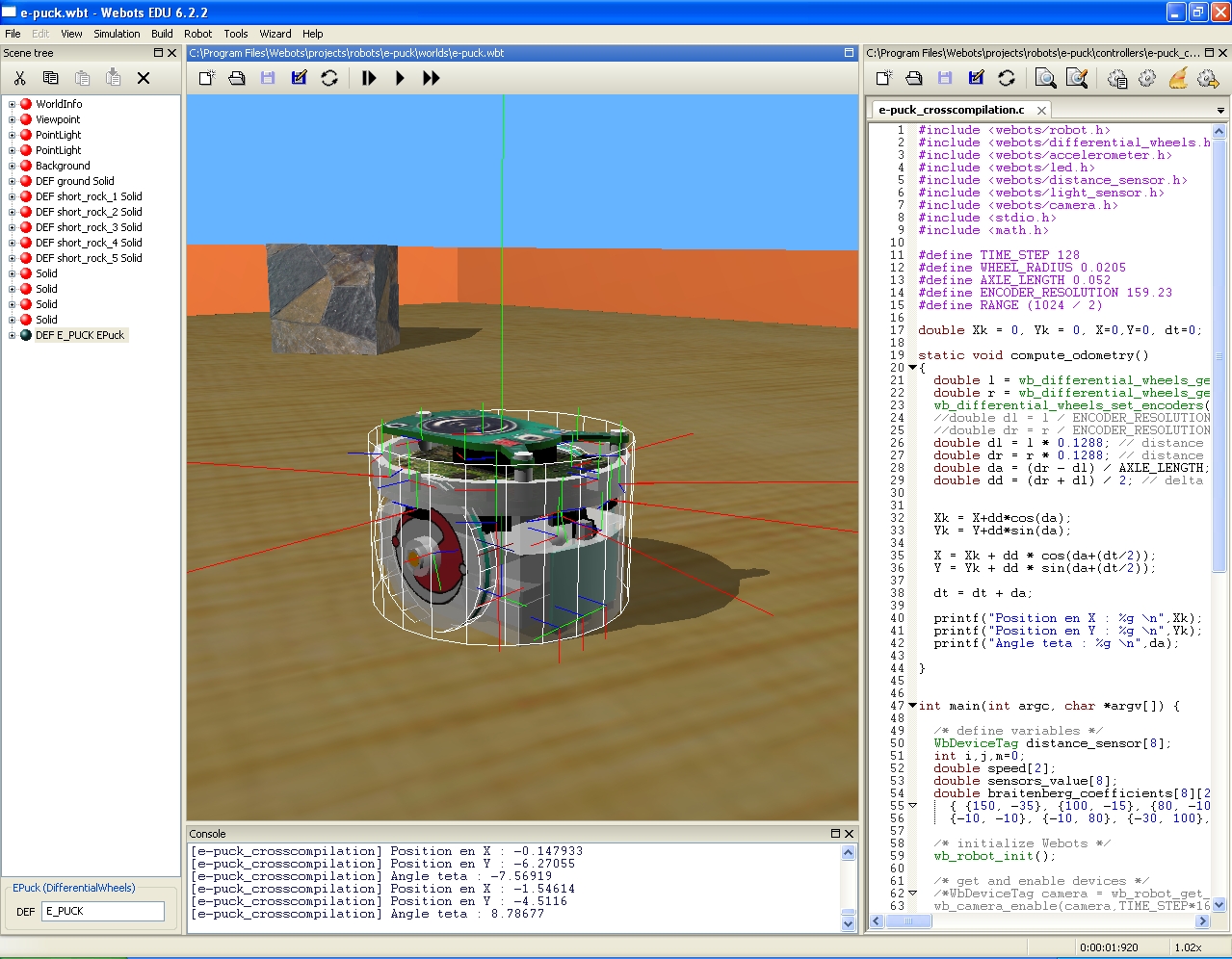


Figure 4 Fenêtre principale de Webots

Webots (figure 4) se décompose en 3 parties. A gauche on trouve l’arbre de scène qui permet la construction et la configuration complète du monde en 3D. Au centre, la partie visualisation et console de débogage. Enfin à droite, la partie où l’on code le programme.

Webots propose aussi un système de contrôle distant (exclusivement pour le robot e-puck), c'est-à-dire que l’on peut aussi bien simuler le programme avec le robot virtuel et passer sur le modèle réel grâce à la connexion Bluetooth.

Pour utiliser le contrôle distant, on va implanter un firmware (logiciel embarqué) dans le robot e-puck qui va se charger d’interpréter le code compilé sous Webots (figure 5).

**Transcodage**

**Robot réel**

**Firmware**

**Webots**

Figure 5 Schéma explicatif sur la commande à distance d'un robot sous Webots

Le robot e-puck (figure 6) a été initialement conçu pour l’enseignement et la recherche. Conçu à l’origine en 2004 par Francesco Mondada et Michael Bonani à l’[EPFL](http://fr.wikipedia.org/wiki/EPFL) (Ecole polytechniques fédéral de Lausane). Le robot e-puck est open-hardware et open-source, plusieurs compagnies le produisent et le vendent.



Figure 6 e-puck

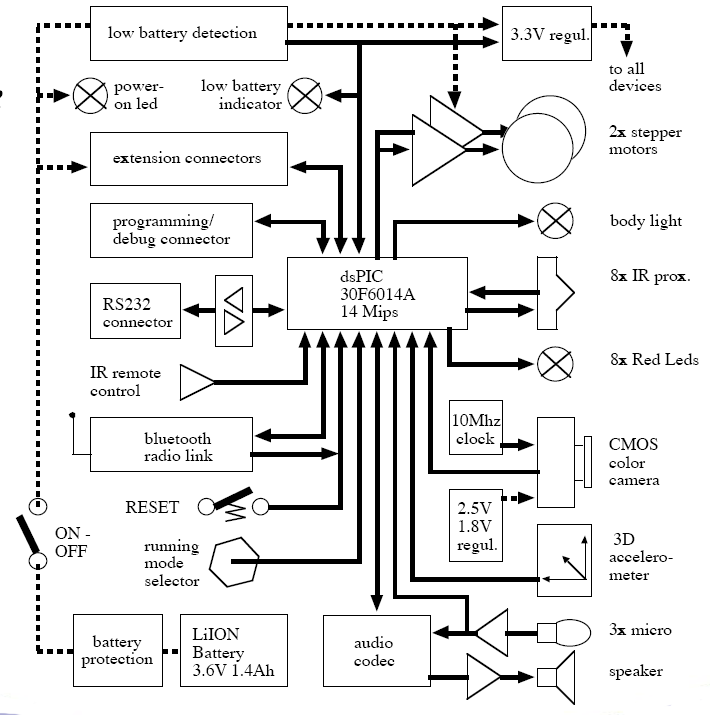


Figure 7 Diagramme du robot e-puck

Le robot e-puck offre un ensemble de capteurs (infrarouge, microphone, caméra, accéléromètre..) qui lui permettent une meilleure évolutivité. De plus, il est possible de rajouter des cartes d’extensions. Enfin le robot e-puck peut communiquer par liaison Bluetooth ou par liaison filaire série RS232. Cf : figure 7.

### B – Sujet de TP N°1 : l’odométrie

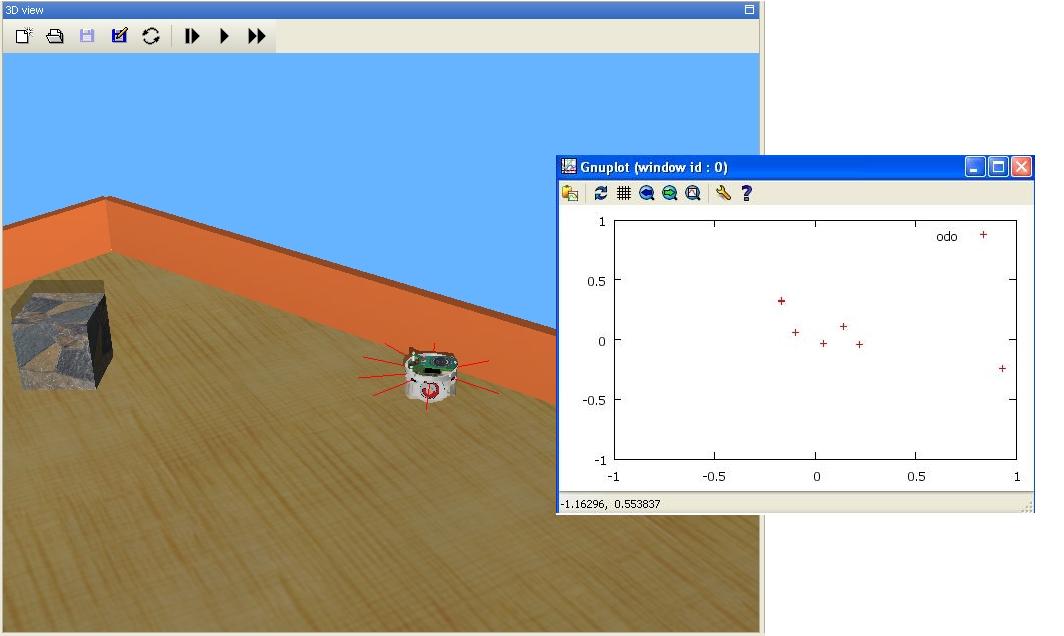
Une des problématique majeure en robotique mobile concerne la détermination de la position du robot et la construction d’une cartographie de l’environnement (SLAM : Simultaneous Localization and Mapping). L’objectif de ce TP est de déterminer les coordonnées du robot e-puck et construire une carte des obstacles rencontrés (figure 8).

Figure 8 Représentation des obstacles rencontrés en simulation

(Chaque point représente un obstacle)

Tout d’abord une petite introduction sur le principe de l’odométrie. L’odométrie est une méthode de positionnement à l’estime qui permet de connaitre les coordonnées d’un robot mobile. Pour ce faire, on mesure le déplacement individuel de chaque roue pour déterminer le mouvement général du robot. En partant d’une position initiale connue et en intégrant les déplacements mesurés, il est alors possible de connaitre à chaque instant la position du robot.



Figure 9 Paramètres du robot à roues différentielles

Pour trouver la distance parcouru par une roue, prenons la gauche ici, il faut multiplier le diamètre de la roue par PI que l’on divise par n, n étant le nombre d’impulsion pour un tour. Et on multiplie le tout par N qui est le nombre d’impulsions mesurées, soit :



Pour trouver la distance totale parcourue par le robot, on fait la moyenne des deux distances parcourue par les deux roues soit :



Ensuite on calcul l’angle qui mesure l’écart entre la position initiale et la position finale :





Figure 10 Evolution d'un robot à roues différentielles

Pour finir, on utilise les notions de trigonométrie :



Après une étude approfondie du principe de l’odométrie, j’ai dû interpréter les calculs en langage de programmation C. Pour cela, j’ai repris le programme d’odométrie réalisé au cours du module MC TNS.

J’ai ensuite effectué un ensemble de tests pour valider le fonctionnement du programme d’odométrie. Malheureusement les résultats obtenus n’ont pas été probant.

Afin de bien comprendre le problème et pour l’isoler plus facilement, je me suis documenté en cherchant des articles scientifiques sur cette méthode, j’ai examiné des programmes fait par d’autres personnes sur leur robot afin de comprendre comment ils ont structurés leur code.

Les sources du problème étaient :

* L’unité des résultats. On doit obtenir des mètres hors ce n’est pas le cas.
* La gestion du changement de cap. L’angle θ n’est pas cohérent.
* L’implication de ces deux problèmes créant des résultats finaux incohérents. Des valeurs complètements absurdes avec de mauvaises unités.

En cherchant dans la documentation de Webots, j’ai trouvé un fichier source contenant des fonctions d’odométrie que j’ai intégrées à mon code. En analysant le code, j’ai tout de suite compris pourquoi mon programme d’avant ne me donnait pas les résultats escomptés ; je faisais des erreurs d’interprétations dans mes calculs et il manquait des données pour calculer avec précisions la distance parcourue par chaque roue.

Les résultats obtenus étaient plus que satisfaisant :

* L’unité des résultats étaient bien des mètres et des radians
* La gestion du cap était prise en compte correctement
* La cohérence des résultats était bonne

L’objectif principal de ce TP soit la récupération des coordonnées du robot à tout instant était pleinement remplie.

### C – Sujet de TP N°2 : Suivi de la balle

La robotique mobile autonome nécessite des informations permettant de percevoir l’environnement. Une des techniques les plus utilisée en robotique est la vision.

Le but de ce TP (figure 11), est de faire suivre une balle (de couleur verte ici en l’occurrence) en utilisant des informations issues d’une caméra.

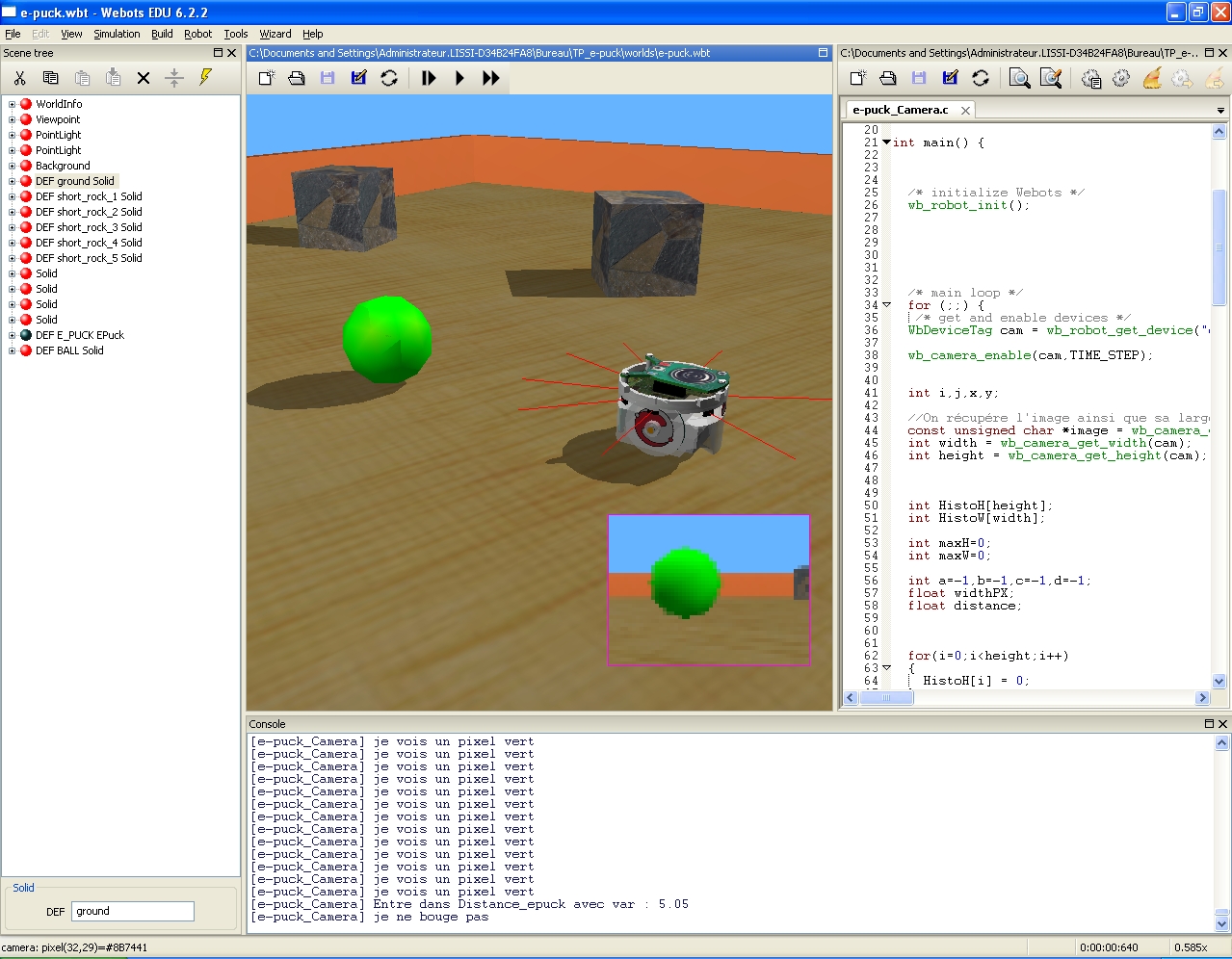


Figure 11 E-puck et la balle verte sous webots

Dans ce TP, l’objectif principal est la détection de la balle. Pour ce faire, il faut traiter l’image c'est-à-dire que l’on récupère l’image puis on examine chaque pixel que l’on décompose en 3 composantes, le rouge, le vert, le bleu (figure 12), qui retournent une valeur comprise entre 0 et 255

Pour ne voir que du vert, il faut d’abord mettre en place, grâce à ces 3 composantes, des filtrages en utilisant des valeurs seuils.

**Pixel**

**Pixel**

**Pixel**

**Image**

**Pixel**

Figure 12 Décomposition en RVB

Une fois que l’on a détecté un pixel qui nous intéresse, il faut mémoriser sa position. Pour cela on utilise des histogrammes. Un histogramme représente, dans notre cas, la concentration en pixel en fonction de la hauteur d’une part, et de la largeur de l’image d’autre part. Si l’on trace le graphique de l’histogramme, on obtient la forme de balle.

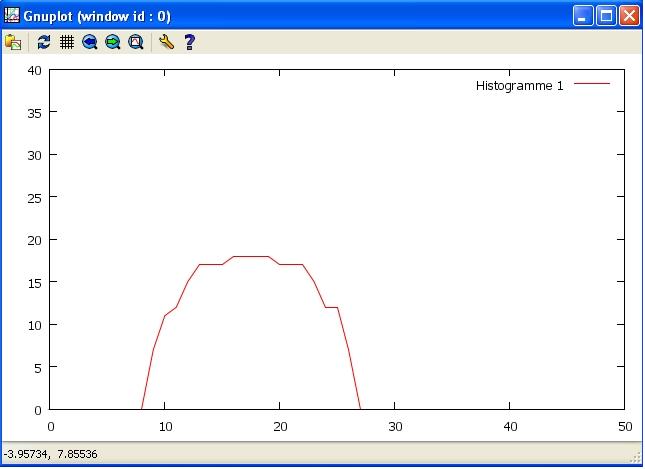


Figure 13 Représentation d'un histogramme

On voit concrètement que la courbe représente la balle (figure 13).

Afin de bien comprendre cette méthode, M. ramik m’a expliqué le principe sur lequel j’ai pu bâtir une base solide.

En cherchant dans les programmes fournit avec Webots, j’ai trouvé un algorithme qui reprend exactement cette idée sauf qu’il n’utilisait pas d’histogramme. Il m’a donc fallut adapter l’algorithme avec les histogrammes.

Maintenant que l’algorithme de détection est en place, il faut calculer la distance qui sépare la balle du robot.

Distance D

Caméra

Taille de l’objet

Alpha

Figure 14 Calcul de la distance par triangulation

La figure 14 représente ce sur quoi l’on s’appuie pour calculer la distance. On multiplie la taille de l’objet par la tangente d’alpha, où alpha correspond à la taille angulaire de l’objet :



Tobjet/pixel: Taille de l’objet en pixel

φ : Champ de vision de la caméra

width : largeur en pixel de l’image

Maintenant que l’on connait la distance, il est possible de déplacer le robot en fonction de celle-ci.

### D – Rédaction des TP

La rédaction, fut une partie très intéressante. L’enjeu pour moi, fut de me mettre, en quelques sortes, à la place du professeur et d’élaborer des sujets qui respectent le cahier des charges.

Le sujet de TP sur le suivi de la balle a été celui où j’ai passé le plus de temps à mettre en place. Beaucoup de notion comme la détection de la balle, la mémorisation de la présence de la balle, le calcul de la distance qui sépare le robot de la balle. Il fallait implémenter tout ceci dans le sujet sans que le sujet soit trop compliqué, pas bien expliqué.

Ensuite il fallait trouver un moyen de faire un sujet avec des niveaux de difficultés progressifs. Pour finir, la construction du squelette du programme en adéquation avec le sujet.

Les sujets de TP sont disponibles en annexe.

### E – bilan

Le bilan pour cette partie est très positif, les TP étaient opérationnels pour le début du module de robotique, ils ont répondu au cahier des charges, les étudiants n’ont, dans l’ensemble, pas eu de difficulté à comprendre le sujet et on vite saisit l’objectif du TP. Pour ma part j’ai grandement apprécié encadrer ces TP durant les heures réservées à cet effet, ce fut une expérience agréable et enrichissante. De plus, ce travail sur la réalisation m’a apporté de nouvelles connaissances en programmation mais aussi en matière de réflexion.

## 3 – Le bras-robot Lynx6

### A – Présentation de Lynx6

Lynx6 (figure 15) est la modélisation d’un bras mécanique semblable à ceux que l’on trouve dans les usines automatisé. Pourvu de 7 servomoteurs commandé par une carte SSC-32, il est possible de la piloter par ordinateur via le programme livré avec le robot ou bien on peut programmer un PIC et le commander via le port série RS232.



Figure 15 Lynx6

Voici un aperçu du programme RIOS (figure 16 Robotic Arm interactive operating system)

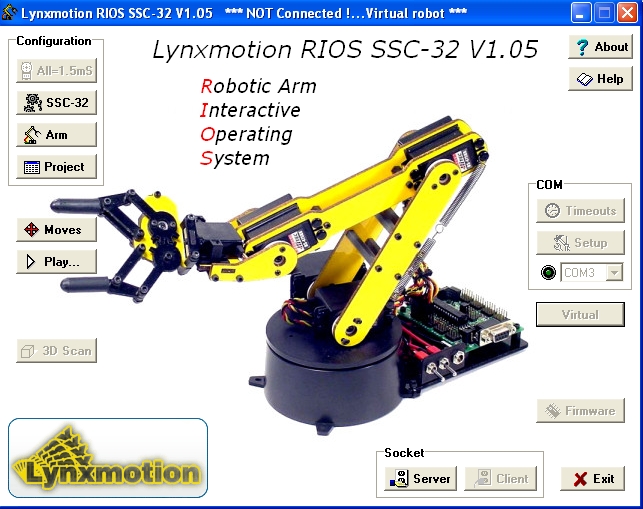


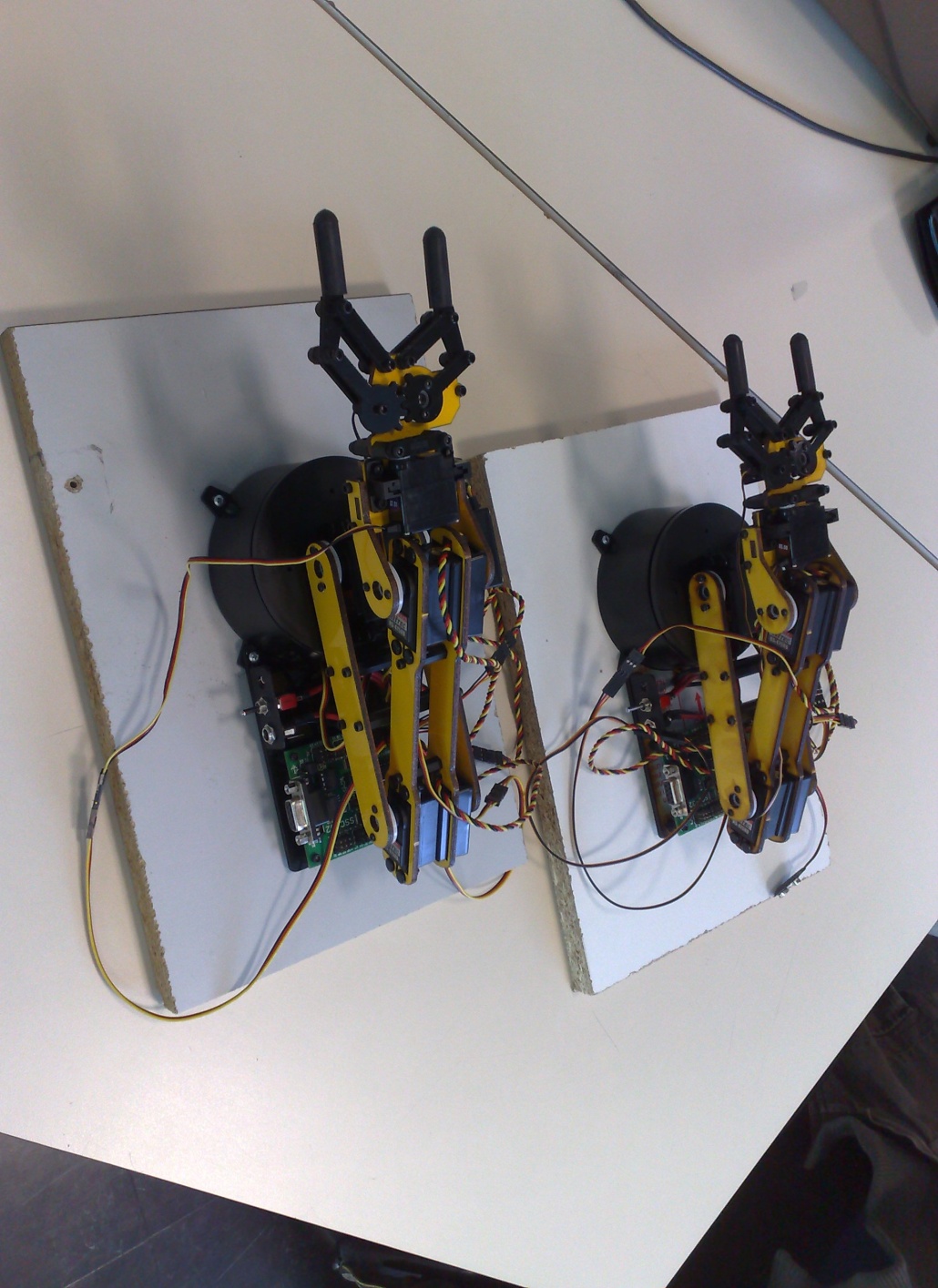
Figure 16 : Interface du programme RIOS

C’est grâce à ce logiciel que l’on va pouvoir configurer la carte SSC-32, régler le bras, créer ses séquences de mouvements et les tester.

Ce programme est simple d’utilisation, intuitif et efficace mais n’offre pas beaucoup d’intérêt si l’on veut faire une application en rapport avec l’informatique industrielle.

### B – Travaux réalisé sur Lynx6

Le Lynx6 est livré en kit à monter, j’ai donc monté les deux bras sans grandes difficultés, cela s’apparente à des mécanos. Afin d’augmenter leur stabilité j’ai fixé les bases sur deux planches en bois (figure 17).

Figure 17 Lynx6 montés sur socles

Afin de facilité la compréhension du programme fournis avec le bras, j’ai créé un petit tutoriel qui explique dans les grandes lignes chaque aspect du programme.

J’ai réalisé une étude sur les capacités du robot afin d’évaluer les possibilités sur d’éventuels sujet de projet tuteurés.

Comme la carte qui contrôle les servomoteurs possède une liaison série, tout programme qui peut se connecter via une liaison série RS232 est capable d’envoyer des commandes à la carte.

C’est pourquoi, en utilisant le logiciel putty, qui s’apparente à un HyperTerminal, j’ai testé l’envoie de commande par port série.

L’essaie fut concluant, la commande envoyé à fait correctement répondre le bras.

La commande se décompose ainsi : #**(Canal)** P**(longueur de l’impulsion)** <cr>

Exemple : #0 P1500 <cr>

Concrètement, on donne d’abord le numéro du canal (Tableau 1) qui correspond servomoteur que l’on souhaite bouger.

|  |  |
| --- | --- |
| Numéro du canal | Servomoteur |
| 0 | Base |
| 1 | Epaules |
| 2 | Coude |
| 3 | Poignet |
| 4 | Grip |
| 5 | Poignet (rotation) |

Tableau 1Correspondance entre canal et servomoteurs

La longueur de l’impulsion correspond à la position où l’on veut que le servomoteur se déplace, elle est exprimée en micro seconde (figure 18).

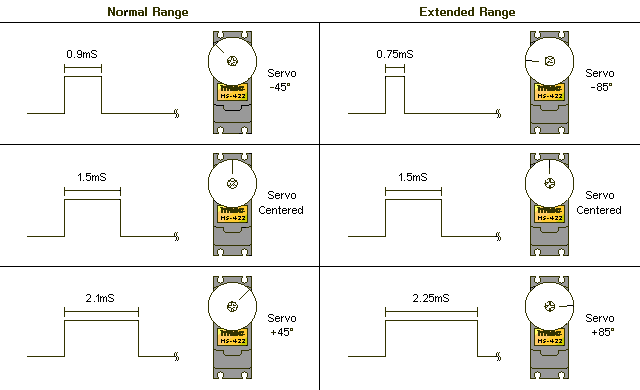
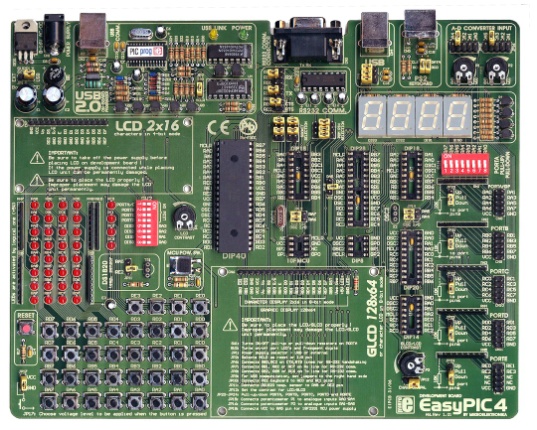
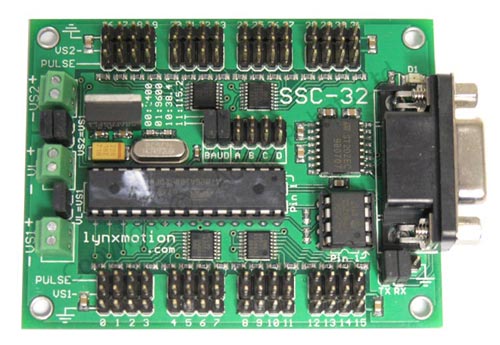


Figure 18 Positionnement des servomoteurs en fonction de la commande

Le dernier élément correspond au retour chariot qui est nécessaire pour que l’action s’exécute.

Maintenant que l’on sait que l’envoi d’une commande par port série est possible via un HyperTerminal, on peut imaginer envoyer ces commandes par un PIC



**Figure 19 Illustration de l’idée du projet tuteuré**

Servomoteur

Plateforme de programmation PIC

Envoie de la commande au servomoteur

Carte de contrôle SSC-32

Envoie de la commande par liaison RS-232 depuis le PIC vers la carte SSC-32

# IV – Conclusion

Pendant le déroulement de ce stage, j’ai pu travailler dans un domaine qui m’intéresse du point de vue de mon projet professionnel, les systèmes embarqués, en l’occurrence ici le support est un robot. Mais aussi j’ai pu goûter au coté pédagogique que ma mission offrait, et j’en suis pleinement satisfait.

La réalisation des deux sujets de TP m’ont permis d’étudier des applications comme l’odométrie et le traitement d’image ce qui m’a beaucoup intéressé. Ainsi que le coté rédaction du sujet et encadrement qui ont été une expérience enrichissante d’un point de vue personnel mais pourquoi pas professionnel en tant que projet de carrière.

J’aurais aimé pouvoir aller plus loin dans la mise en place du projet tuteuré, car il exploite l’électronique, élément qui me passionne le plus.

Ce stage fut une très bonne expérience à la fois enrichissante et passionnante et m’aura donné l’envie de d’aller plus loin dans la programmation et de continuer dans la logique de mon projet professionnel soit le domaine des systèmes embarqués.

# V – Bibliographie

# Sites internet :

# 

# [www.e-puck.org](http://www.e-puck.org) (site officiel d’e-puck)

# [www.lynxmotion.com](http://www.lynxmotion.com) (site officiel de Lynxmotion)

* [www.gctronic.com](http://www.gctronic.com) (site officiel du revendeur du robot e-puck)

# Articles :

# « Capteurs et méthodes pour la localisation des robots mobiles » par Marie-José ALDON (chargée de recherche au CNRS)

# « On differential drive robot odometry with application to path planning » de Evangelos Papadopoulos et Michael Misailidis

# « Software development for the kinematic analysis of a Lynx6 robot arm » de Baki Koyuncu et Mehmet Güzel

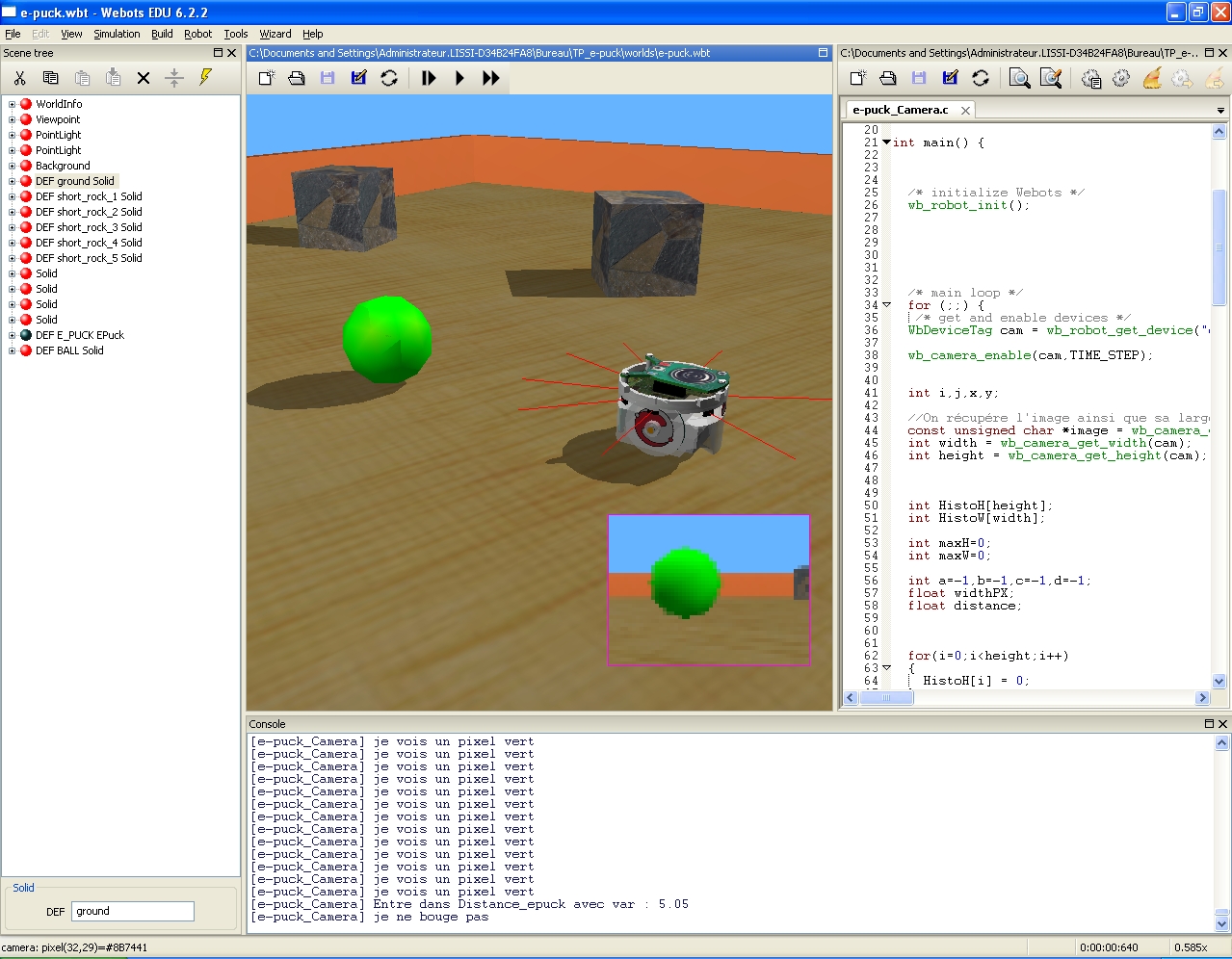
# ANNEXES

**Programmation du robot Epuck**

**TP3 : Suivi balle**

I – Présentation du TP

Le but de ce TP est de détecter une balle avec le robot e-puck, de définir sa taille et d’en déduire la distance qui les sépare.



Tout d’abord, il faut récupérer le dossier contenant le TP, aller sûr le serveur GEII puis récupérer le dossier TP\_e-puck Balle. Copier le sûr votre espace perso. Lancer le fichier e-puck.wbt dans le dossier worlds et lancer la simulation.

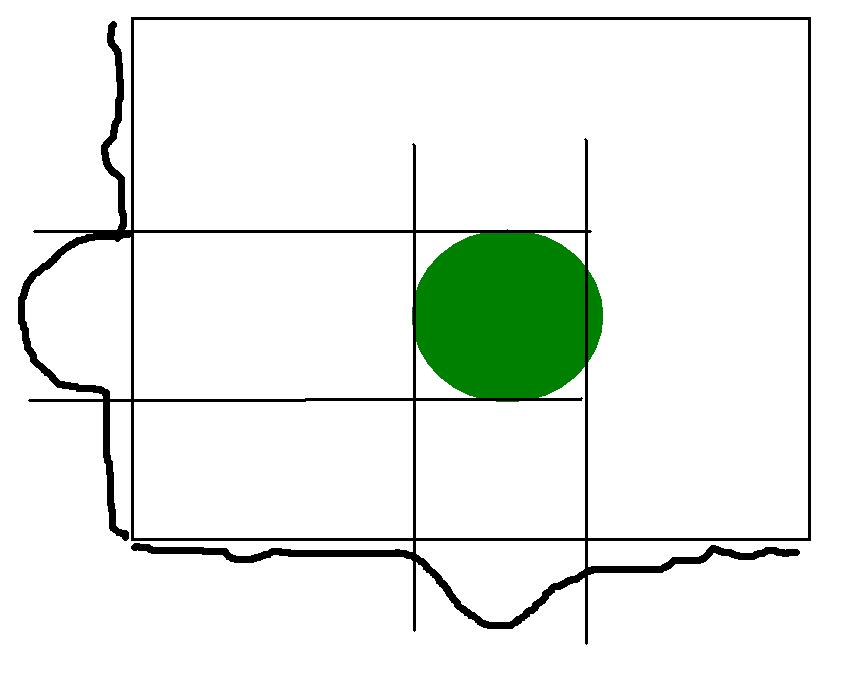
II – Détection de la balle

1. Détermination des Histogrammes.

Pour détecter la balle, il est nécessaire premièrement de récupérer l’image et deuxièmement de la traiter. Chaque image peut être représentée sous la forme de trois tableaux de pixels  RVB (rouge vert bleu). Pour détecter un objet particulier (géométrie, taille, couleur, etc..), une solution consiste alors à déterminer un ou des histogrammes de l’image.

Un histogramme est un graphique statistique permettant de représenter la distribution des intensités des pixels d'une image, c'est-à-dire le nombre de pixels pour chaque intensité lumineuse.

Par exemple, si on désire détecter une balle verte, il est nécessaire calculer l’histogramme de l’image à partir de seuil de référence Concrètement, on balaye l’image et à chaque pixel vert rencontré, on le mémorise dans un tableau. Si l’on représente graphiquement le tableau, on obtient un graphique comme ci-dessus avec la concentration en pixel vert symbolisée par les « bosses ».



Voici l’algorithme permettant de balayer l’image :

Pour i de 0 à Width

Pour j de 0 à height

r = wb\_camera\_image\_get\_red(image,width,i,j)

g = wb\_camera\_image\_get\_green(image,width,i,j)

b = wb\_camera\_image\_get\_blue(image,width,i,j)

Si le pixel entre dans la gamme de vert alors

Histogramme\_width[i] ++

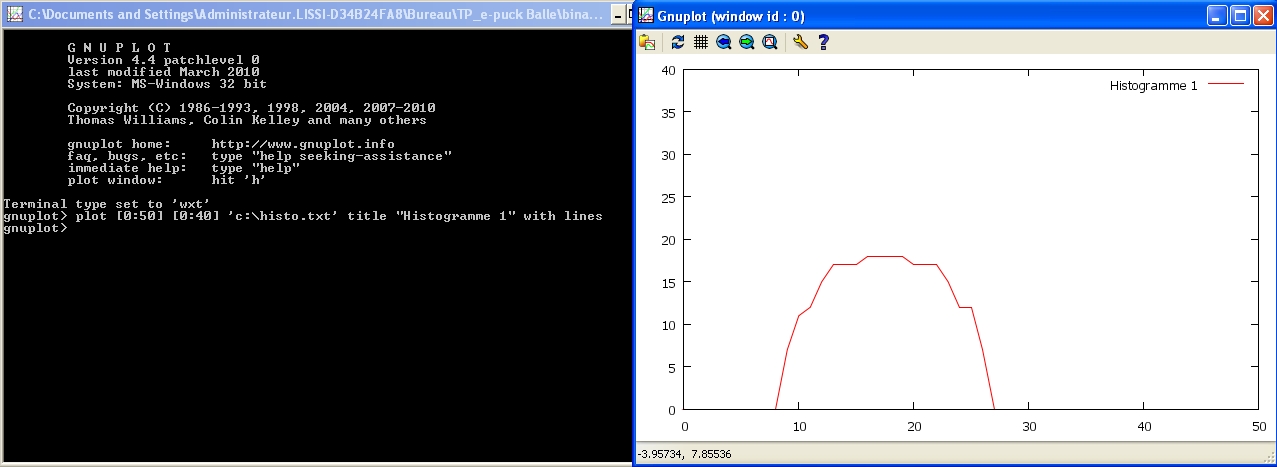
Histogramme\_height[j] ++

Fin pour

Fin pour

A vous de chercher les seuils maintenant. Une fois que vous avez fixé les seuils, récupérer l’intégral d’un histogramme dans un fichier.

Une fois fait, vous allez tracer le graphique de l’histogramme à l’aide du petit soft Gnuplot.



Voici à quoi ressemble Gnuplot, c’est un logiciel en ligne de commande qui permet de tracer des graphiques, et dans notre cas à nous de pouvoir traiter un fichier de données et d’en sortir un graphique.

Pour tracer un graphique à partir d’un fichier, lancer Gnuplot.exe qui se trouve dans le dossier binary de votre TP, puis une fois la console ouverte, taper cette commande.

Plot [x :x] [y :y] ‘***emplacement de votre fichier*** ‘ title « ***titre de votre graphique*** » with lines.

[x :x] : coordonnées en x

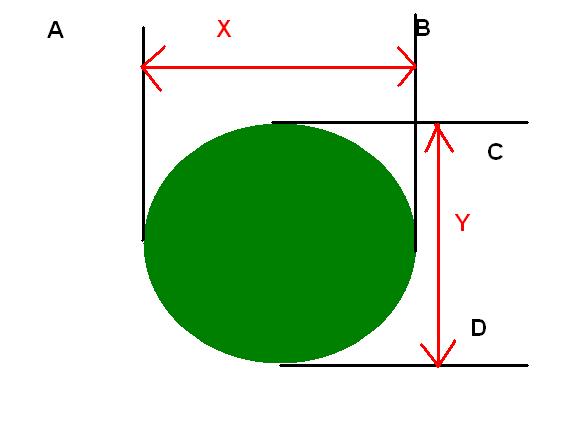
[y :y] : coordonnées en y

1. Traitement des histogrammes

Dans cette partie, vous allez devoir traiter les histogrammes afin de supprimer les valeurs inutiles, parasites. Pour ce faire, il faut chercher le maximum pour les deux histogrammes et à partir du maximum, fixer un seuil où toutes les valeurs qui lui sont inférieures soient remplacées par un zéro.

1. Détermination de la taille et de la distance

Maintenant que vos histogrammes sont traités, il faut déterminer la taille de la balle. Pour cela, il faut chercher les valeurs qui cadrent l’intervalle dans lequel on voit du vert.



Voici l’algorithme pour récupérer ces valeurs :

Pour i de 0 à width

Si Histo\_width != 0 et a = -1 alors

a = i

Si Histo\_width == 0 et a >= 0 alors

b = i – 1

Fin Pour

Si b = -1 alors

b = width – 1

Faite en de meme pour c et d avec la hauteur

x = fabs(a-b);

y = fabs(c-d);

Pour en arriver à la taille en pixel de la zone verte :

widthPX = (float)(x+y)/2;

Et enfin pouvoir calculer la distance qui sépare e-puck de l’objet, voici le calcul :

Distance = Wobjet  \* tan (widthPX\*(Phi/Wimage))

Wobjet : Taille de l’objet utilise en cm (ici la balle fait 7cm)

widthPX : Taille en pixel de la zone verte

Phi : Champ de vision de la caméra (ici 1)

Wimage : largeur en pixel de l’image (ici 52)

III- Gestion de la distance

Maintenant que l’on peut connaitre la distance qui sépare le robot de la balle, on peut se fixé un intervalle qui va nous permettre de garder le robot à distance de la balle.

#include <webots/robot.h>  
#include <webots/differential\_wheels.h>  
#include <webots/accelerometer.h>  
#include <webots/led.h>  
#include <webots/distance\_sensor.h>  
#include <webots/light\_sensor.h>  
#include <webots/camera.h>  
#include <stdlib.h>  
#include <stdio.h>  
#include <time.h>  
#include <string.h>  
#include <math.h>  
  
#define TIME\_STEP 64  
#define RANGE (1024 / 2)  
  
int main()

{  
   */\* initialisation de Webots \*/*  
  wb\_robot\_init();  
    
   FILE\* fichier = NULL;  
   */\* Boucle principale \*/*  
   while(1)   
   {  
   */\* Récupération du TAG et activation \*/*  
  WbDeviceTag cam = wb\_robot\_get\_device("camera");  
 wb\_camera\_enable(cam,TIME\_STEP);  
   
   //On récupére l'image ainsi que sa largeur et sa hauteur  
  const unsigned char \*image = wb\_camera\_get\_image(cam);  
   int width = wb\_camera\_get\_width(cam);  
   int height = wb\_camera\_get\_height(cam);  
  int HistoH[height], HistoW[width];  
   int i,j,x,y;  
   int maxH=0, maxW=0;  
   int a=-1,b=-1,c=-1,d=-1;  
   float widthPX ,distance;  
  
  for(i=0;i<height;i++)  
   {  
    HistoH[i] = 0;  
   }  
   for(i=0;i<width;i++)  
   {  
    HistoW[i] = 0;  
   }  
    
  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //Question 1: détermination de l'histogramme  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  for (i=0;i<width;i++)   
  {  
     for (j=0;j<height;j++){  
      int r = wb\_camera\_image\_get\_red(image,width,i,j);  
      int g = wb\_camera\_image\_get\_green(image,width,i,j);  
      int b = wb\_camera\_image\_get\_blue(image,width,i,j);  
        
      if (r<60 && g>50 && b<75)  
      {   
        HistoH[j]++;  
      HistoW[i]++;  
      }  
    }

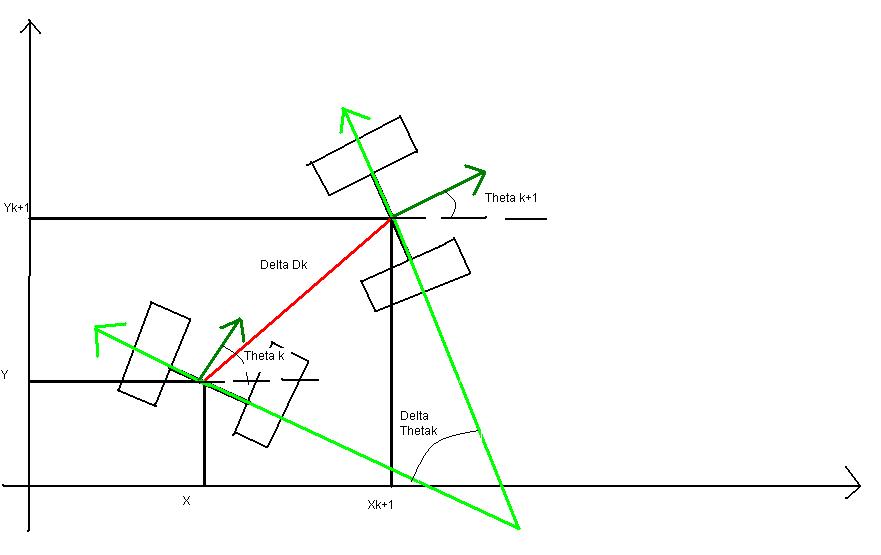
  }  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //Ecriture dans un fichier  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  fichier = fopen("histoH.txt","a");  
    
  for(i=0;i<height;i++)  
  {  
    if(fichier != NULL)  
    {  
      fprintf(fichier,"%d %d**\n**",i,HistoH[i]);   
    }  
  }  
  fclose(fichier);  
    
 fichier = fopen("histoW.txt","a");  
  for(i=0;i<width;i++)  
  {  
    if(fichier != NULL)  
   {  
      fprintf(fichier,"%d %d**\n**",i,HistoW[i]);  
   }  
  }  
  fclose(fichier);  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*   
    
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //Question 2: Traitement des histogrammes  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
      
    for(i=0;i<height;i++)  
    {  
       if(HistoH[i] > maxH)  
      {  
        maxH = HistoH[i];  
      }  
    }  
    for(i=0;i<width;i++)  
    {  
      if(HistoW[i] > maxW)  
       {  
        maxW = HistoW[i];  
     }  
    }  
  
  for(i=0 ; i<height;i++)  
  {  
     if(HistoH[i] <= ((float)0.3\*maxH))  
     {  
      HistoH[i] = 0;  
     }  
  }  
  for(i=0;i<width;i++)  
  {  
    if(HistoW[i] <= ((float)0.3\*maxW))  
     {  
      HistoW[i] = 0;  
     }  
  }  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //Questions 3 : Détermination de la taille et de la distance  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  for(i=0;i<height;i++)  
  {  
    if(HistoH[i] !=0 && a==-1)  
   {  
      a = i;  
     }  
     if(HistoH[i] == 0 && a >= 0)  
     {  
      b = i - 1;  
     }    
  }   
    if(b == -1)  
    {  
      b = height - 1;  
    }  
      
  for(i = 0;i<width;i++)  
  {  
    if(HistoW[i] !=0 && c == -1)  
     {  
      c = i;  
     }  
    if(HistoW[i] == 0 && c >= 0)  
     {  
      d = i - 1;  
     }  
  }     
    if(d == -1)  
    {  
      d = width - 1;  
    }  
    y = fabs(a-b);  
    x = fabs(c-d);  
      
    widthPX = (float)(x+y)/2;  
      
    distance = 7 \* tan(widthPX\*((float)1/52));  
      
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
    
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //Question 4: Etablissement de la gestion de la distance  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  printf("Entre dans Distance\_epuck avec var : %.2f**\n**",dist);  
    
  if(dist > 7.1)  
  {  
    wb\_differential\_wheels\_set\_speed(-250,-250);  
    printf("je recule**\n**");  
  }  
  else if(dist > 5 && dist < 7)  
  {  
    wb\_differential\_wheels\_set\_speed(0,0);  
      printf("je ne bouge pas**\n**");  
  }  
  else if(dist < 5)  
  {  
    wb\_differential\_wheels\_set\_speed(250,250);  
     printf("j'avance**\n**");  
  }  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
    wb\_robot\_step(TIME\_STEP);  
  }   
  return 0;}

**Programmation du robot Epuck**

**TP4 : SLAM**

Le but de ce TP, est de récupérer la position du robot quasiment en temps réel afin de déterminer par la suite la position d’obstacle. Pour cela, nous allons utiliser le principe de l’odométrie. L’odométrie permet de déterminer la position en [x,y] et le cap Theta d’un véhicule naviguant sur un sol plan, par rapport au repère de référence qui était celui du robot dans sa configuration initiale.



Tout d’abord on mesure la distance parcourue par chaque roue de cette manière.



Avec

r : rayon de la roue

n : nombre d’impulsion mesurées

k : facteur de correction

N : nombre d’impulsion pour 1 tour

Q : Coefficient de conversion (configuration.wheel\_conversion\_xxx)

Où R est le rayon de la roue, n le nombre d’impulsions délivrées par le codeur de résolution alpha.

Ensuite, il faut calculer le déplacement élémentaire et la rotation élémentaire

Déplacement élémentaire : 

Rotation élémentaire :  Où E est la distance séparant les deux roues.

Une fois ceci fait, on va pouvoir déterminer les positions à l’instant k+1.



Où Theta2 vaut : 

Maintenant on peut déterminer les positions et l’angle final.



1. **Détermination de X, Y et Theta**

A l’aide des explications précédentes, déterminer les coordonnées en x,y et le cap theta.

***Voir l’annexe pour l’utilisation de l’objet ot.***

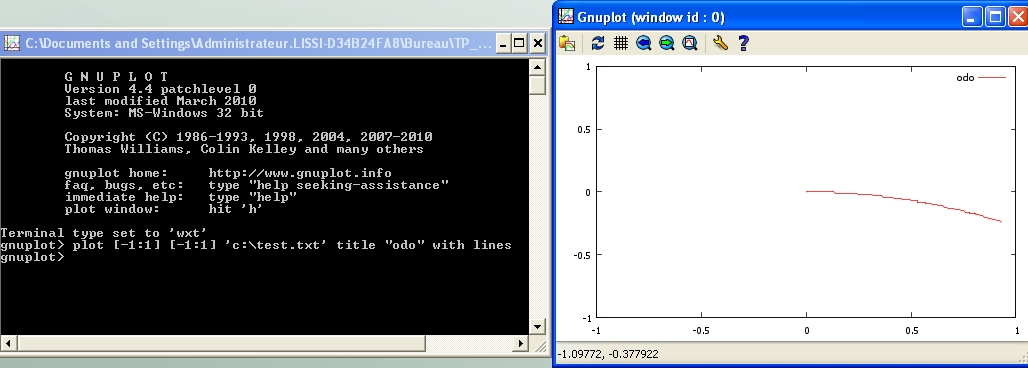
1. **Récupération des coordonnées dans un fichier**

Maintenant que vos coordonnées sont vérifiées, récupérer celles-ci dans un fichier pour pouvoir le traiter par la suite.

Pour une meilleure vérification, faites aller le robot tout droit pendant 10 secondes, puis à l’aide de Gnuplot tracer le résultat obtenu grâce à l’odométrie.

Pour rappel, Gnuplot se trouve dans le dossier Binary de votre TP, lancer le fichier Gnuplot.exe et taper cette ligne de commande :

Plot [x :x] [y :y] ‘***emplacement de votre fichier*** ‘ title « ***titre de votre graphique*** » with lines.



Voilà ce que l’on doit obtenir, l’odométrie n’est pas une science exacte alors cela reste imprécis

1. **Cartographie d’obstacles**

***Avant toute chose, enlever le commentaire qui encadre l’appel de la fonction run() et supprimer le code que vous avez fait pour la question 2.***

Dans cette partie, le robot a un comportement Braitenberg qui lui permet d’éviter tous les obstacles qu’il rencontre sur sa route. Le but est de cartographier chaque objet que le robot croise afin de connaitre la position.

Voici l’algorithme qui nous permet de récupérer la position :

Pour i de 0 à 8

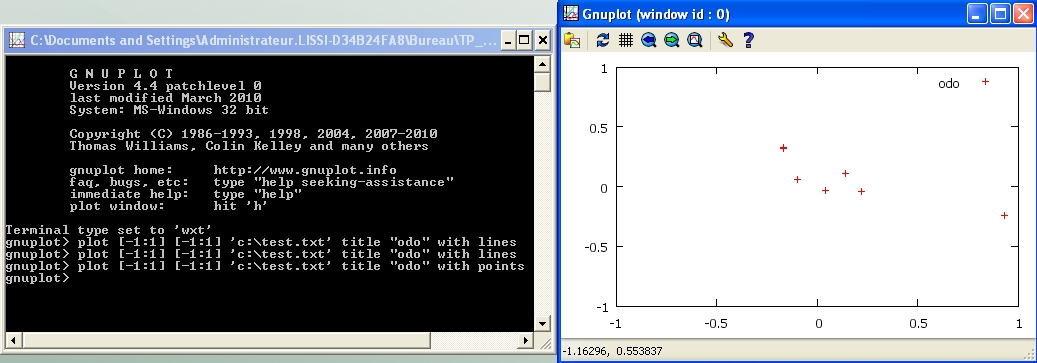
Si capteur[i] > valeur seuil alors

On écrit dans un fichier les coordonnées dans un fichier

On ferme le fichier ensuite

Fin pour

Une fois que votre repérage est fait et que votre fichier est remplit, il nous vous reste plus qu’à tracer le graphique avec Gnuplot.



Ici, chaque point représente un obstacle.

**ANNEXE**

Afin de vous aider à mieux utiliser les variables, nous allons vous expliquer comment accéder à un objet.

struct sOdometryTrack ot;

Cela signifie que l’on déclare une structure qui s’appelle sOdometryTrack à qui l’on associe l’objet ot.

Dans le fichier odometry.h, on définie cette structure de la manière suivante

struct sOdometryTrack {

struct {

float wheel\_distance;

float wheel\_conversion\_left;

float wheel\_conversion\_right;

} configuration;

struct {

int pos\_left\_prev;

int pos\_right\_prev;

} state;

struct {

float x;

float y;

float theta;

} result;

};

Dans cette structure, on a aussi définie trois structures qui contiennent chacune des variables. Pour accéder à ces variables, il faut procéder de cette manière :

Je suis dans mon programme principale, je veux accéder à la variable x de l’objet result, il suffit d’écrire la syntaxe suivante ot->result.x.

Concrètement, j’ai un tiroir qui s’appel ot avec dedans une boite result qui contient un élément x.

#include <webots/robot.h>  
#include <webots/differential\_wheels.h>  
#include <webots/accelerometer.h>  
#include <webots/led.h>  
#include <webots/distance\_sensor.h>  
#include <webots/light\_sensor.h>  
#include <webots/camera.h>  
  
#include <stdlib.h>  
#include <stdio.h>  
#include <time.h>  
#include <string.h>  
#include <math.h>  
#include "odometry.h"  
  
#define RANGE (1024 / 2)  
#define LEFT               0  
#define RIGHT              1  
#define TIME\_STEP          64  
#define PI                 3.141592653589793  
#define NB\_DIST\_SENS       8  
#define THRESHOLD          1500  
  
WbDeviceTag ps[NB\_DIST\_SENS];  
  
float angle\_offset[NB\_DIST\_SENS] = {0.2793, 0.7854, 1.5708, 2.618, -2.618, -1.5708, -0.7854, -0.2793};  
  
int weightleft[8] = {-10,-10,-5,0,0,5,10,10};  
int weightright[8] = {10,10,5,0,0,-5,-10,-10};  
  
float increments\_per\_tour = 1000.0; // from e-puck.org  
float axis\_wheel\_ratio = 1.4134;     // from e-puck.org  
float wheel\_diameter\_left = 0.0416;  // from e-puck.org  
float wheel\_diameter\_right = 0.0404; // from e-puck.org  
float scaling\_factor = 0.976;         // default is 1  
  
struct sOdometryTrack ot;  
  
static void run();  
  
  
static void run(void)  
{  
  int i;  
  float speed[2] = {300,300};  
  int sensor;  
  FILE\* fichier = NULL;  
   
  for(i=0;i<NB\_DIST\_SENS; i++)  
  {  
    if((wb\_distance\_sensor\_get\_value(ps[i]) == 0 || 1 || 6 || 7)&&(wb\_distance\_sensor\_get\_value(ps[i])> THRESHOLD))  
    {  
      fichier = fopen("test.txt","a");  
        
      if(fichier != NULL)  
      {  
        fprintf(fichier,"%.2f  %.2f **\n**",ot.result.x,ot.result.y);  
        fclose(fichier);  
      }  
    }  
  }  
  
  for(i=0;i<8;i++)  
  {  
    sensor = wb\_distance\_sensor\_get\_value(ps[i]);  
    speed[LEFT] += weightleft[i] \* (sensor>>4);  
    speed[RIGHT] += weightright[i] \* (sensor>>4);  
  }  
    
  speed[LEFT] = speed[LEFT] > 500 ? 500 : speed[LEFT];  
  speed[LEFT] = speed[LEFT] < -500 ? -500 : speed[LEFT];  
  speed[RIGHT] = speed[RIGHT] > 500 ? 500 : speed[RIGHT];  
  speed[RIGHT] = speed[RIGHT] < -500 ? -500 : speed[RIGHT];  
    
  wb\_differential\_wheels\_set\_speed(speed[0], speed[1]);  
}  
  
int main(int argc, char \*\*argv) {  
  
  wb\_robot\_init();  
  wb\_differential\_wheels\_enable\_encoders(TIME\_STEP);  
    
  int i;  
    
  char textPS[] = "ps0";  
  for (i=0;i<NB\_DIST\_SENS;i++) {  
    ps[i] = wb\_robot\_get\_device(textPS);  
    wb\_distance\_sensor\_enable(ps[i],TIME\_STEP);  
    textPS[2]++;  
  }  
  
  odometry\_track\_start(&ot);  
  
  for (;;) {  
       
    odometry\_track\_step(&ot);  
     
    run();  
      
    wb\_robot\_step(TIME\_STEP);  
  }   
  return 0;  
}  
  
void odometry\_track\_init() {  
}  
  
int odometry\_track\_start(struct sOdometryTrack \* ot) {  
    return odometry\_track\_start\_pos(ot, wb\_differential\_wheels\_get\_left\_encoder(), wb\_differential\_wheels\_get\_right\_encoder());  
}  
  
int odometry\_track\_start\_pos(struct sOdometryTrack \* ot, int pos\_left, int pos\_right) {  
    ot->result.x = 0;  
    ot->result.y = 0;  
    ot->result.theta = 0;  
  
    ot->state.pos\_left\_prev = pos\_left;  
    ot->state.pos\_right\_prev = pos\_right;  
  
    // Odometry values  
    ot->configuration.wheel\_distance = axis\_wheel\_ratio \* scaling\_factor \* (wheel\_diameter\_left + wheel\_diameter\_right) / 2;  
    ot->configuration.wheel\_conversion\_left = wheel\_diameter\_left \* scaling\_factor \* PI / increments\_per\_tour;  
    ot->configuration.wheel\_conversion\_right = wheel\_diameter\_right \* scaling\_factor \* PI / increments\_per\_tour;  
    return 1;  
}  
  
void odometry\_track\_step(struct sOdometryTrack \* ot) {  
    odometry\_track\_step\_pos(ot, wb\_differential\_wheels\_get\_left\_encoder(), wb\_differential\_wheels\_get\_right\_encoder());  
}  
  
void odometry\_track\_step\_pos(struct sOdometryTrack \* ot, int pos\_left, int pos\_right) {  
    long delta\_pos\_left, delta\_pos\_right;  
    float delta\_left, delta\_right, delta\_theta, theta2;  
    float delta\_x, delta\_y;  
  
    delta\_pos\_left = pos\_left - ot->state.pos\_left\_prev;  
    delta\_pos\_right = pos\_right - ot->state.pos\_right\_prev;  
    delta\_left = delta\_pos\_left \* ot->configuration.wheel\_conversion\_left;  
    delta\_right = delta\_pos\_right \* ot->configuration.wheel\_conversion\_right;  
    delta\_theta = (delta\_right - delta\_left) / ot->configuration.wheel\_distance;  
    theta2 = ot->result.theta + delta\_theta \* 0.5;  
    delta\_x = (delta\_left + delta\_right) \* 0.5 \* cosf(theta2);  
    delta\_y = (delta\_left + delta\_right) \* 0.5 \* sinf(theta2);  
  
    ot->result.x += delta\_x;  
    ot->result.y += delta\_y;  
    ot->result.theta += delta\_theta;  
  if (ot->result.theta > PI) ot->result.theta -= 2\*PI;  
  if (ot->result.theta < -PI) ot->result.theta += 2\*PI;  
    ot->state.pos\_left\_prev = pos\_left;  
    ot->state.pos\_right\_prev = pos\_right;  
}

**Résumé :**

J’ai effectué mon stage au sein du département GEII de l’IUT de Sénart/Fontainebleau. Durant ce stage, j’ai créé deux Travaux pratiques basé sur le robot e-puck ainsi qu’un environnement pour un projet tuteuré l’année prochaine sur le bras mécanique Lynx 6.

Au cours de ce stage j’aurais mis à contribution mes compétences en informatique pour la réalisation des programmes des différents travaux pratiques. Mais j’aurai aussi découvert le coté pédagogique de ce stage ce qui m’a fortement intéressé.

Ce stage confirme mon projet professionnel et m’aura offert de nouvelle perspective de projet d’avenir.

**Abstract :**

I attended my course in the department GEII of IUT Sénart/Fontainebleau. During this course, I created two labs work based on the e-puck robot and I made an environement for a supervised project about the robotic arm Lynx6.

In this course, I exerciced my computer skills on implementation of programs of different labs work. I also discovered an attractive educational side of the course

This placement confirms my professional project and it offers me a new perspective plan for the future.