
# SY31 TP2

## Odométrie

**Pellerin Romain, Luo Yin, Baaziz Mohamed**

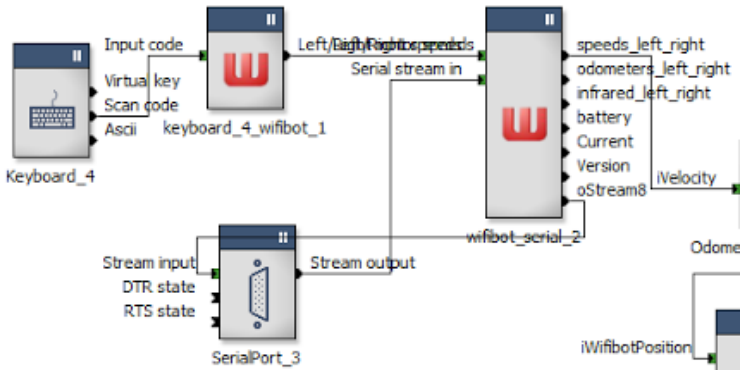
**22/10/2014**

# Introduction :

L'objectif de ce TP est de reconstruire la trajectoire du Wifibot et de l'afficher à l'écran. C'est-à-dire, afficher l'évolution de l'odométrie (position x et y et orientation θ) au cours du temps. Afin de déterminer ces coordonnées, nous avons besoin des informations de vitesse fournies par les 2 codeurs de roues situés sur les moteurs avant du WifiBot. Ainsi, pour ce réaliser cet objectif, nous allons développer un module RTMaps prenant comme entrées les vitesses des roues et affichant en sortie l'évolution de l'odométrie au cours du temps.
Nous avons à notre disposition un robot WifiBot muni des logiciels RTMaps et Visual Studio. C'est sur le robot que nous réalisons la programmation.

## I- Travail préparatoire sur RTMaps

Avant de commencer à étudier l'odométrie du robot à proprement parlé, nous devions nous assurer de bien connaître les commandes qui contrôlent le déplacement du robot. Pour cela, nous avons réalisé un programme RTmaps :




Le bloc Keyboard_4 lit les commandes entrées au clavier, les transmet au clavier virtuel du WifiBot. Celui-ci établit une communication en série avec le bloc wifibot_serial_2 qui contrôle la vitesse et l'odométrie des roues gauche et droite. Ce programme RTmaps nous permet donc de contrôler le mouvement des roues du WifiBot.
Après exécution, un simple essai sur les touches du clavier nous permet de connaître les commandes de contrôle des roues :

- flèche du haut : donne une vitesse positive aux roues (mouvement avant)
- flèche du bas : donne une vitesse négative aux roues (mouvement arrière)
- flèche droite : tourne les roues vers la droite
- flèche gauche : tourne les roues vers la gauche
- bar espace : arrête le robot

## II- Programmation du composant

Afin de générer le composant RTMaps, nous avons dû, à partir d'un squelette fourni, écrire les fonctions C++ nécessaires au calcul de l'odométrie du robot.

Trois classes nous étaient fournies :

- OdometryComponent
- PositionViewer
- Wifibot

Il nous a fallu remplir le corps des fonctions de **odometryComponent.cpp** et de **wifibot.cpp**.

### 1) odometryComponent.cpp

Cette classe représente le composant RTMaps. Elle gère toute la durée de vie du composant : naissance, fonctionnement et mort (arrêt du composant) ainsi que sa réinitialisation. Voici le code que nous avons complété, commenté :

```
//...
const double NB_TICKS_PER_WHEEL_TURN = 2448; // = 2Pi
const double WIFIBOT_ENTRAX_IN_METER = 0.3;
const double WIFIBOT_RADIUS_IN_METER = 0.0625;
const double TWOPI = 2 * M_PI;
//...

////////////////////////////////////////////////////////////////
// RTMaps - Birth
////////////////////////////////////////////////////////////////
void
OdometryComponent::Birth()
{
  // Appel du constructeur avec les paramètres requis
  mWifibot=Wifibot(WIFIBOT_ENTRAX_IN_METER,WIFIBOT_RADIUS_IN_METER);
  // On stocke le timestamp (temps) actuel en microsecondes
  mPreviousTimestamp = MAPS::CurrentTime();
}

////////////////////////////////////////////////////////////////
// RTMaps - Core
////////////////////////////////////////////////////////////////
void
OdometryComponent::Core()
{
  //...

  // On stocke le temps écoulé en secondes depuis le dernier appel de la fonction
  double elapsedTime = (MAPS::CurrentTime() - mPreviousTimestamp) / 1000000;
  // Vitesse actuelles des roues gauche et droite en tick/s
  double vLeft = static_cast<double>(input->Integer32(0));
```

```
  double vRight = static_cast<double>(input->Integer32(1));
  // Conversion de tick/s vers radian/s
  vLeft = (vLeft*TWOPI)/NB_TICKS_PER_WHEEL_TURN;
  vRight = (vRight*TWOPI)/NB_TICKS_PER_WHEEL_TURN;
  // Appel de la méthode updateOdometry de l'objet mWifibot pour mettre à jour son
      odométrie en fonction du temps écoulé, de la vitesse des roues gauche et droite
      en radian/s
  mWifibot.updateOdometry(elapsedTime,vLeft,vRight);

  //...

  // On stocke le timestamp actuel en nmicrosecondes
  mPreviousTimestamp = MAPS::CurrentTime();

}

void
OdometryComponent::reset()
{
  mWifibot.resetOdometry(); // On réinitialise l'odométrie du robot (toutes les
      valeurs à zéro)
}

//...
```

## 2) WifiBot.cpp

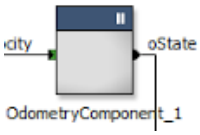
Cette classe représente le Wifibot. Elle permet de modifier les paramètres du robot (l'entrax, le rayon des roues ou l'odométrie). Elle permet également d'obtenir des informations sur le robot (la vitesse linéaire et la vitesse angulaire).
Pour la plupart des fonctions, nous avons réutilisé les formules fournies dans l'énoncé du TP. Voici le code que nous avons complété, expliqué en commentaires :

```
//...

// Constructeur qui prend comme paramètres l'entraxe et le rayon des roues
Wifibot::Wifibot(double entrax, double wheelRadius)
{
  mEntrax = entrax;
  mWheelRadius = wheelRadius;
}

// Mise à jour de l'odométrie du robot. On fournit en entrée le temps écoulé depuis
    la dernière mise à jour de l'odométrie (en secondes), ainsi que la vitesse des
    roues gauche et droite (en mètres/s)
void
Wifibot::updateOdometry(double dt, double left, double right)
{
  // La position (en x ou en y) est mise à jour de la manière suivante : on ajoute à
      l'ancienne position la vitesse linéaire actuelle, multipliée par le cosinus de l
      'orientation actuelle. La vitesse linéaire est obtenue grâce à une méthode de la
       même classe (voir plus bas).
```

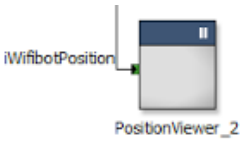
```
  mPosition.x += dt * getLinearSpeed(left, right)*cos(mPosition.th);
  mPosition.y += dt * getLinearSpeed(left, right)*sin(mPosition.th);

  // L'orientation est mise à jour de la façon suivante : on ajoute à l'ancienne
     orientation la valeur négative du rayon des roues, multipliée par la position x
     additionnée à y, divisées par l'entraxe.
  mPosition.th += -mWheelRadius*(mPosition.x+mPosition.y)/mEntrax;
}

// Remise à zéro de toutes les valeurs de l'odométrie
void
Wifibot::resetOdometry()
{
  mPosition.x = 0.0;
  mPosition.y = 0.0;
  mPosition.th = 0.0;
}

// Modifie l'entraxe
void
Wifibot::setEntrax(double entrax)
{
  mEntrax = entrax;
}
```

```
// Modifie l'entraxe
void
Wifibot::setEntrax(double entrax)
{
  mEntrax = entrax;
}

// Modifie le rayon des roues
void
Wifibot::setWheelRadius(double radius)
{
  mWheelRadius = radius;
}

// Retourne la vitesse linéaire en fonction de la vitesse des roues de gauche et de
   droite
double
Wifibot::getLinearSpeed(double left, double right) const
{
  // Vitesse linéaire = rayon multiplié par la vitesse des roues gauche + la vitesse
     des roues droite, le tout divisé par 2
  return mWheelRadius*( left + right ) / 2;
}

// Retourne la vitesse angulaire en fonction de la vitesse des roues de gauche et de
   droite
double
Wifibot::getAngularSpeed(double left, double right) const
{
  // Vitesse angulaire = rayon multiplié par la vitesse des roues gauche - la vitesse
     des roues droites, le tout divisé par l'entraxe
  return -mWheelRadius*(left - right) / mEntrax;
}
```

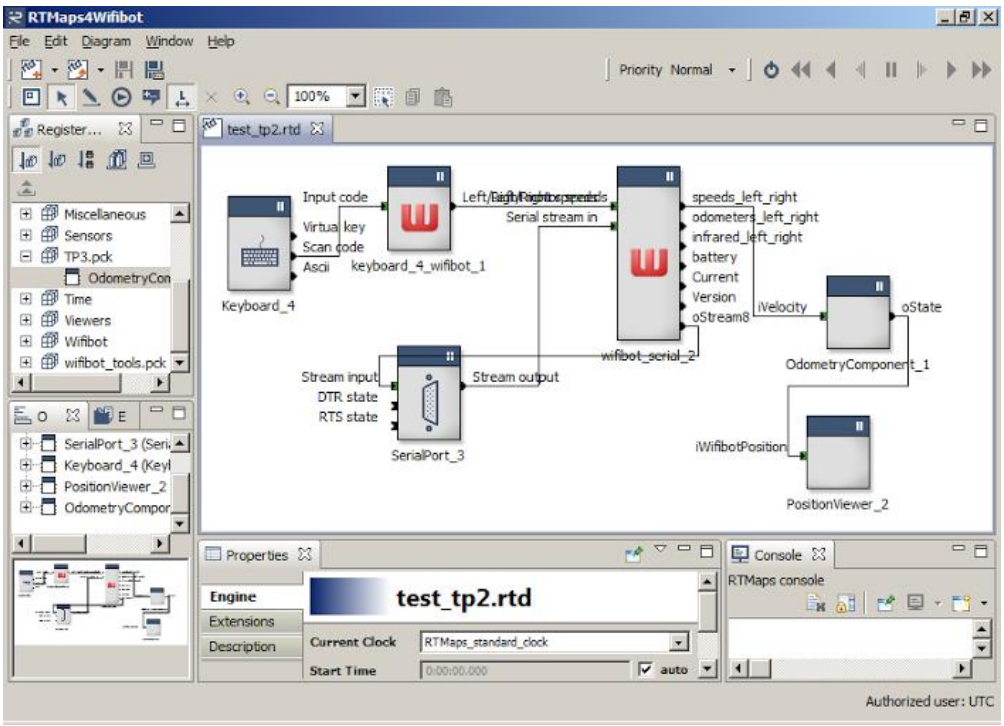
## III- Intégration RTMaps



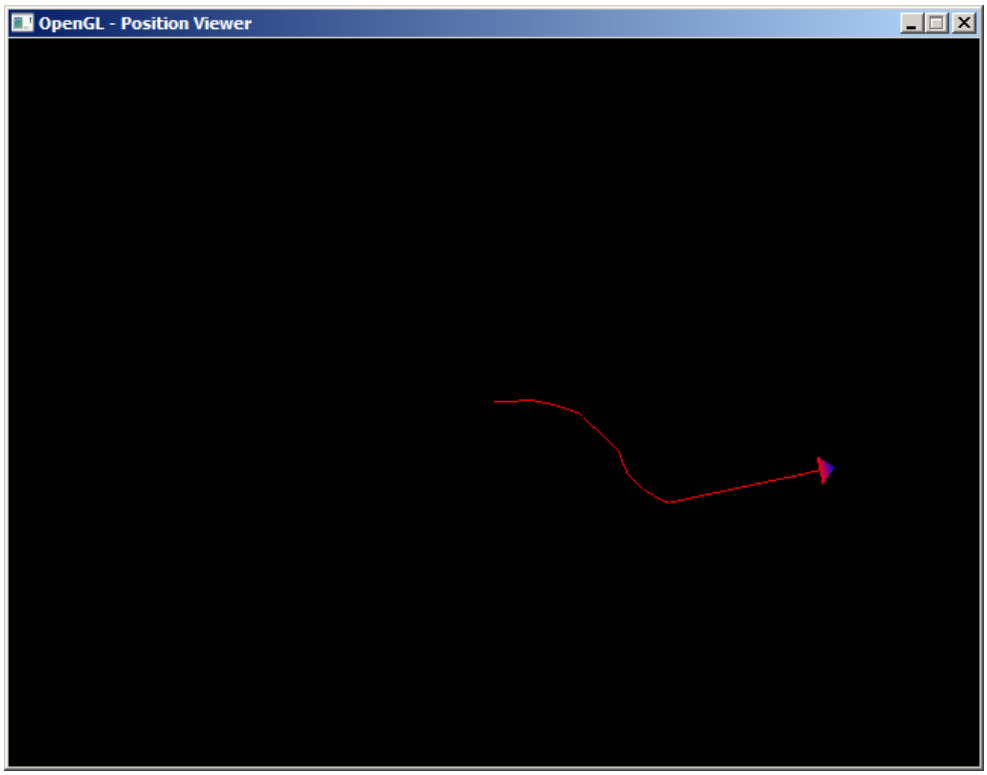

Après exécution du code C++, le compilateur produit un bloc RTmaps : OdometryComponent_1
Celui-ci prend en entrée la vitesse des roues et donne en sortie la position (odométrie) du robot.
Nous voulons obtenir un affichage du déplacement du robot, donc un affichage de l'évolution de l'odométrie du robot. Il nous suffit alors de connecter la sortie « speeds_left_right » du bloc wifibot_serial_2 à l'entrée « iVelocity » du bloc OdometryComponent_1 ; et de connecter la sortie « oState » à un afficheur de position : PositionViewer_2.




Ce bloc a pour entrée « iWifibotPosition » (reçoit l'odométrie du robot) et produit en sortie une interface graphique qui affiche la position du robot en continue.
Nous voici donc avec ce montage RTmaps final :



Après exécution, nous obtenons l'affichage en continu du déplacement du robot :




## Conclusion :

Ce TP nous a permis de nous familiariser avec le langage de programmation C++, ainsi qu'avec le logiciel RTMaps. De plus, nous avons appris à produire un bloc RTMaps à partir d'un code C++. Cependant nous avons rencontré quelques difficultés lors de l'ajout du bloc RTMaps PositionViewer_2 (problème de PATH). Le problème fut résolu en transposant nos fichiers sur un autre ordinateur.