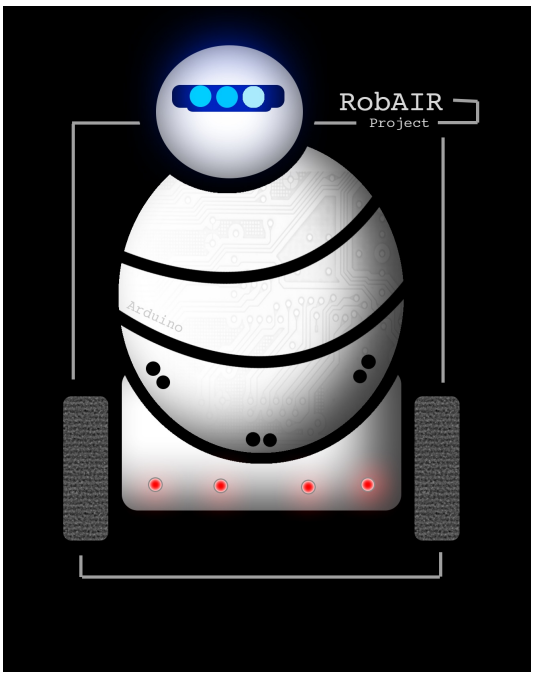


Projet Reseau - Multimedia : RobAIR

MARIAGE Paul, LEVAYER David

Le 7 Mars 2014

Table des matières

1 Introduction

Le projet que nous avons réalisé au cours de notre 4ème année à Polytech'Grenoble est un projet orienté robotique appelé RobAIR. Ce projet est une reprise des projets des années précédentes, réalisées par les promotions RICM4, ainsi que des étudiants de l'école ENSIMAG de Saint Martin d'Hères. Le projet RobAIR consiste à développer un robot de téléprésence, utilisant une plateforme de type Open-Source. Cette année, il est divisé en deux parties : une première partie correspondant au montage complet et à la configuration du robot en ré-utilisant les travaux des années précédentes, et une partie concernant le contrôle à distance utilisant une application à base de WebRTC. Nous parlerons ici de la partie montage et configuration.

1.1 Pourquoi ce robot ?

L'objectif de ce projet est de réaliser un robot de téléprésence. Prenons par exemple une personne handicapée, qui aurait l'envie de visiter un musée inadapté aux personnes à mobilité réduite. Grâce à RobAIR, elle pourra grâce à l'application de l'autre groupe controlé le robot que nous avons monté afin d'effectuer cette visite. De même, un étudiant étant dans l'impossibilité d'assister à ses cours pour diverses raisons pourra d'utiliser un robot disponible dans son établissement afin de ne pas prendre de retard. C'est dans cette optique d'accessibilité que le projet a été pensé, et que nous l'avons réalisé.

1.2 Les pré-requis

Le montage du robot nécessite différents composants :

- Une structure de robot. La notre a été designée par Didier Donsez, professeur et responsable de parcours RICM à Polytech'Grenoble et fabriquée à l'aide d'une découpeuse laser par la FabLab présente à la Maison Jean Kuntzmann de Saint Martin d'Hères
- Un système de propulsion de robot 24V de type Devantech RD02
- Un contrôleur de moteur (livré avec le système de propulsion)
- Deux batteries 12V de la marque Yuasa
- Des capteurs ultrasons et infrarouges adaptés à Arduino
- Une Kinect (pas utilisée dans notre projet, mais qui sera utilisée plus tard pour la création de map)
- Une carte Arduino Mega 2560

1.3 Robot Operating System

Robot Operating System, ou Robot OS, ou encore ROS, est un système d'exploitation sous licence open source permettant de contrôler un robot. Il est développé par la société américaine Willow Garage, à l'origine pour son robot PR2.

Il peut s'interfacer avec la plate-forme de développement pour la robotique Urbi2.

Plusieurs robots sont compatibles avec ROS : Qbo3, Robotino, Nao4 ...

ROS utilise la bibliothèque de traitement d'image OpenCV pour la vision du robot.

Plus d'informations sur les bags à la page suivante :

http ://wiki.ros.org

2 Travail réalisé

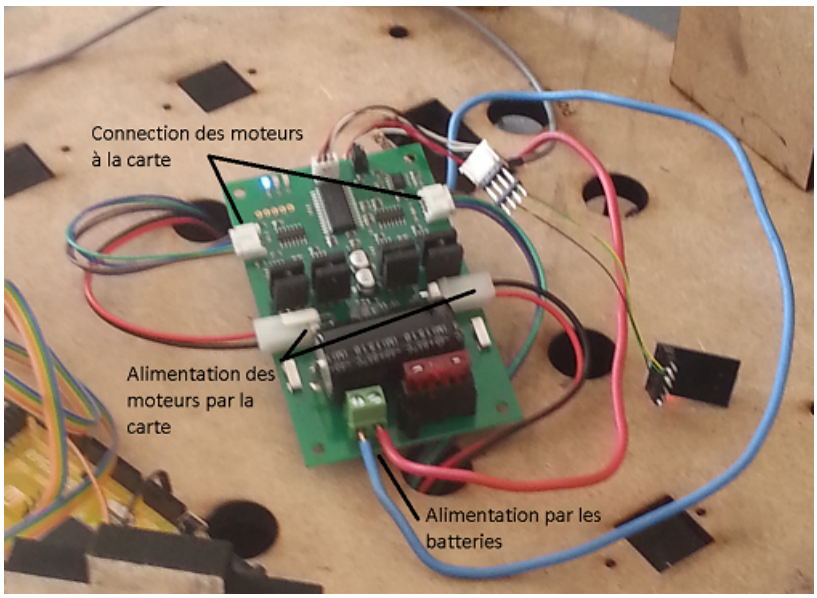
2.1 Montage du robot

2.1.1 Montage des roues

Notre robot va utiliser un système de propulsion de robot 24V de type Devantech RD02. Nous avons donc monter ces deux roues sur le dessous de la base de notre robot grâce aux vis qui nous ont été fournies. Pour plus de stabilité, nous avons ajouté des roues multidirectionnelles à l'avant et à l'arrière, afin que le robot puisse bouger correctement.

2.1.2 Montage des batteries

Les batteries que nous avons choisi d'utiliser sont des batteries 12V de la marque Yuasa. Nous avons positionnées les supports de batteries sur la base du robot puis les avons reliées en séries via les connecteurs puis directement sur le contrôleur des moteurs.



Les roues sont donc alimentées par les batteries via la carte. De plus, les moteurs sont connectés directement à la carte, qui va envoyer l'information à la tablette via le connecteur que nous pouvons voir sur la droite.

2.1.3 Fixation des capteurs

Pour la suite du montage de notre robot, nous avons fixé l'ensemble des capteurs qui seront utilisé sur notre structure. Nous avons deux types de capteurs différents : les capteurs infrarouges fixés sur la base, et les capteurs ultrasons fixés sur les cotés de la structure.

- *Capteurs infrarouge*
  Les capteurs infrarouge que nous avons utilisés sont de type Arduino commutateur du capteur infrarouge réglable détecter 3-80cm. Le commutateur du capteur infrarouge réglable est un ensemble de transmetteur et récepteur à l'un des capteurs commutateur photoélectrique. Nous avons donc ajusté la distance de détection. Le capteur a une portée de détection de 3cm-80cm, ce qui correspond totalement à notre attente. Nous les avons donc fixés à la structure grâce aux bagues fournies avec les capteurs
- *Capteurs ultrasons*
  Notre robot va utiliser des capteurs ultrasons de type HC-SR04. Ce module dispose simplement de 4 pins de sortie : VCC , TRIG, ECHO, GND . Il est très facile à utiliser avec le microcontrôleur. Le processus complet est le suivant : Mettre la pin "TRIG" une impulsion de niveau haut (5V) durant au moins 10µs et le module démarre sa lecture. A la fin de la mesure, s'il détecte un objet devant lui, la pin "ECHO" passe au niveau haut (5V) "PulseIn()". La distance où se situe l'obstacle est proportionnelle à la durée de cette impulsion. Il est donc très facile de calculer cette distance. Nous les avons collés à la structure sur les côtés du robot.

2.1.4 Montage de la Kinect

Dans notre réalisation, nous n'avons pas utilisé la caméra Kinect, mais comme elle sera utilisée dans les itérations suivantes pour créer la map, nous avons décidé de la positionner. Etant donné que la caméra Kinect va permettre la communication à distance, nous devons la placer sur le haut du robot, de façon à ce qu'il est une vue globale de la pièce, juste au dessus du support de la tablette. Elle sera relié directement à la tablette via le hub USB.

2.1.5 Positionnement du Lidar

Au cours du déplacement du robot, nous allons construire dynamiquement une carte représentant la pièce. Pour ce faire, nous avons besoin de placer le Lidar sur le dessus du support, juste au dessus de la Kinect. Le Lidar que nous avons utilisé est de type Capteur de Distance à Balayage Laser URG-04LX-UG01 Hokuyo. Il utilise un télémètre à balayage laser URG-04LX-UG01 Hokuyo. Nous pouvons grâce au Lidar indiquer la distance de 20 mm à 5600 mm (résolution 1 mm) dans un arc de 240º (résolution angulaire 0,36º). Il sera relié à la tablette directement via le hub USB.

2.1.6 Vue finale du robot

2.2 Configuration du robot

Nous allons voir les fichiers les plus importants dans la manipulation et le contrôle du robot. Nous allons donc voir comment nous avons configuré le sketch Arduino, puis à quoi servent les différents dossiers et fichiers, puis comment les configurer de façon à ce que le robot marche correctement.

2.2.1 Paramétrage du sketch Arduino

Dans un premier temps, nous allons régler les variables globales de façon à ce que notre nombre de capteurs dans le sketch corresponde bien au nombre de capteurs dans le robot. Pour cela, nous avons utilisé les variables **INFRA_NB** et **SONAR_NUM**. Ensuite, nous allons définir un tableau d'entiers afin de répertorier les pins sur lesquels nous avons branché les capteurs infrarouges :

```
const int INFRA_PINS[INFRA_NB] = {A0,A1,A2,A3,A4,A,A6,A7};
```

Pour les capteurs ultrasons, nous allons définir un tableau de NewPing, correspondant à des objets de type Ultrasonic Sensor :

```
NewPing sonar[SONAR_NUM] = {
  NewPing(27, 26, MAX_DISTANCE),//north left
  NewPing(2, 3, MAX_DISTANCE), //north center
  NewPing(4, 5, MAX_DISTANCE),//north right
  NewPing(6, 7, MAX_DISTANCE),//south right
  NewPing(8, 9, MAX_DISTANCE),//south center
  NewPing(10, 11, MAX_DISTANCE)//south left
};
```

Les valeurs dans les NewPing correspondent aux numéros des pins sur lesquels nous avons branché les TRIG et ECHO de nos capteurs.

2.2.2 Descriptif et configuration des fichiers

- *Dossier bags*
  Dans ce dossier-ci se trouveront tous les fichiers d'extension .bag . Les « bags »sont des formats pour stocker et rejouer les messages échangés. Ce système permet de collecter par exemple les données mesurées par les capteurs et les rejouer ensuite autant de fois qu'on le souhaite afin de faire de la simulation sur des données réelles. Ce système est également très utile pour déboguer un système à posteriori. Donc chaque fois que nous allons effectuer un enregistrement de carte par le Lidar, nous allons sauvegarder les messages

dans ce dossier au format bag. Plus d'informations sur les bags à la page suivante :

http ://wiki.ros.org/rosbag

- *Dossier launch*
  Pour le lancement des différents scénarios de notre robot, nous avons besoin d'utiliser des fichiers .launch. Un fichier « launch »permet de lancer plusieurs noeuds / applications ainsi que le noyau ROS en une ligne de commande. Un exemple de fichier simple pour la navigation par clavier :

```
<launch>
  <!--Permet de deplacer le robot simplement au clavier-->
  <node pkg="robair_demo" type="kb_control.py" name="kb_control"/>
  <node pkg="robair_demo" type="motion_control_node.py" name="motion_control_node"/>
  <node pkg="robair_demo" type="arduino_sensors.py" name="arduino_sensors"/>
</launch>
```

  On voit donc que dans ce launch, nous allons utiliser les noeuds kb_control, motion_control_node et arduino_sensors. Par la suite, nous pourrons complexifier ces launch en ajoutant des paramètres à nos noeuds pour réaliser les actions que nous désirons. Plus d'informations sur les launch à la page suivante :

  http ://wiki.ros.org/roslaunch/XML

- *Dossier maps*
  Le dossier maps est l'endroit où toutes les cartes générées seront enregistrées. La création de maps peut se faire soit avec le noeud hokuyo_node soit avec le noeud hector_slam qui existent pour les différentes versions de ros. Il existe des tutoriels assez développés aux adresses suivantes :

  http ://wiki.ros.org/hokuyo_node
  http ://wiki.ros.org/hector_slam

  Ainsi, lors de l'exécution du noeud de création de map, nous pouvons créer et enregistrer un fichier de type .yaml représentant notre carte. L'idée de fond de YAML est que toute donnée peut être représentée par une combinaison de listes, tableaux (de hachage) et données scalaires. YAML décrit ces formes de données (les représentations YAML), ainsi qu'une syntaxe pour présenter ces données sous la forme d'un flux de caractères (le flux YAML).

- *Dossier msg*
  C'est dans ce dossier que nous allons positionner les fichiers qui définiront les noms de toutes les parties du robots :
  - *Fichier Command.msg*
    Dans ce fichier, il a été défini tous les mouvements possibles de notre robot :
    int8 move
    uint8 speed1
    uint8 turn

  - *Fichier encoderData.msg*
    Dans ce fichier, il a été défini les noms de nos deux roues :
    int32 wheelRight
    int32 wheelLeft

  - *Fichier InfraredPotholes.msg*
    Dans ce fichier, il a été défini le nom des capteurs infrarouges. Les capteurs infrarouges ont été définis comme des booléens. Ce booléen est à Vrai si il y a un trou, à Faux sinon. Ils ont été appelés de la façon suivante :

    bool rear_left
    bool rear_center_left
    bool rear_center_right
    bool rear_right

    bool front_left
    bool front_center_left

bool front_center_right
bool front_right

- *Fichier UltrasoundObstacles.msg*
  Comme le fichier InfraredPotholes.msg, il a été défini dans ce fichier les capteurs ultrasons. La valeur de ces capteurs correspond à la distance qu'ils ont avec un obstacle. Ils ont été appelés de la façon suivante :

  uint32 north_left
  uint32 north_right
  uint32 north_east

  uint32 south_left
  uint32 south_right
  uint32 south_east
  Ainsi, si nous voulons construire un robot disposant de plus de capteurs, que ce soit infrarouges ou ultrasons, ou ajouter des mouvements, ce sont ces fichiers que nous allons modifier.

- *Dossier rviz_cfg*
  Dans ce dossier, nous avons toutes les configurations du programme rviz, qui va servir à l'affichage de la map que nous créera le lidar. Cependant, l'extension est de type vcg, et l'extension actuelle des fichiers de configuration du programme rviz est en .rviz. Il va donc falloir redéfinir un fichier de configuration.

- *Dossier script*
  Dans ce dossier, nous avons tous les fichiers de scripts qui tourneront sur notre tablette, et qui serviront au fonctionnement de notre robot.
  - *kb_control.py*
    Ici sont définies les touches qui servirons à faire avancer, reculer ou tourner le robot. Il faudra modifier ce fichier si nous voulons utiliser des touches différentes.
  - *motion_control_node.py*
    Ce fichier correspond au noeud de contrôle du robot. Il faudra bien vérifier que le port a été configuré correctement et qu'il soit branché au contrôleur.
  - *arduino_sensor.py*
    Ce fichier correspond au noeud de gestion des sensors. Il faudra aussi vérifier que le port des sensors est correct.

- *Dossier src*
  - *Arduino.py*
    Dans ce fichier, nous avons toutes les fonctions qui vont gérer les capteurs. Les deux fonctions principales sont :
    - process_infrared_line
      Cette fonction va boucler sur tous les capteurs infrarouges du robot, et mettre à jour la valeur de ces capteurs.
    - process_ultrasound_line
      Cette fonction est similaire à la précédente, mais elle travaille sur les capteurs ultrasons.

  - *keylogger.py* Ce fichier sert à l'enregistrement des informations saisies au clavier. Il n'est pas nécessaire de modifier ce fichier.

  - *motorcmd.py*
    - send_order : Fonction d'envoi d'un ordre de déplacement au robot
    - isFrontSensorsOK : Fonction qui vérifie si aucun obstacle/trou ne gène le mouvement du robot vers l'avant
    - isRearSensorsOK : Fonction qui vérifie si aucun obstacle/trou ne gène le mouvement du robot vers l'arrière
    - move : Envoi l'ordre au robot de se déplacer

- *Dossier util*
  Dossier de configuration de l'IP local et des ports séries. Nous n'avons pas modifié ces différents fichiers, étant donné que le travail avait été fait les années précédentes et fonctionne correctement

– *Dossier voix*
  Dans ce dossier, nous avons positionner tous les fichiers audio qui serviront au robot, lorsqu'il se déplace et rencontre une personne ou autre. Les fichiers audio doivent être de type wav.

2.3 Etablissement de la documentation

Au cours de notre projet, nous avons été étonné par le manque de documentation lié au montage et à la configuration du robot. Nous avons donc passé beaucoup de temps à effectuer le montage du robot, étant deux novices en robotique, et la prise en main du programme ROS était aussi ardue, bien que le code était particulièrement bien commenté.

C'est pourquoi nous avons décidé de réaliser une documentation regroupant la plupart des informations qui nous auraient été nécessaires, que ce soit pour le montage du robot ou sa configuration. Nous avons donc mis en ligne trois tutoriaux qui serviront aux années suivantes lors de la réalisation de leur robot :
– Un tutoriel pour le montage disponible à l'adresse suivante :
  http ://air.imag.fr/index.php/Proj-20132014RobAIR2/How_to_create_robAIR
– Un tutoriel pour l'installation de ROS
  http ://air.imag.fr/index.php/Proj20132014RobAIR2/getStarted
– Un tutoriel pour la configuration des fichiers
  http ://air.imag.fr/index.php/Proj20132014RobAIR2/How_to_configure_robAIR

3 Conclusion

En conclusion, nous pouvons dire que les objectifs de notre projet sont globalement remplis : nous avons réussi à reconstruire le robot suivant l'architecture qui a été définie auparavant, nous avons pu reprendre le code pour l'adapter à notre propre configuration, nous avons pu afficher des nuages de point grâce au Lidar, et finalement réaliser une documentation que nous jugeons solide pour les prochains étudiants qui reprendront notre projet.

4 Sources

– Pour les packages ROS : http ://wiki.ros.org

– Pour les différentes définitions : https ://fr.wikipedia.org

– Pour les informations sur robAIR et les réalisations des années précédentes : http ://air.imag.fr/index.php/RobAIR