Projet de Robotique Mobile

La société MobileRobots a mis en place ARIA qui est 'Advanced Robot Interface for Applications' qui est une bibliothèque C++ de développement logiciel (SDK) pour toutes les plates-formes MobileRobots / ActivMedia. ARIA peut contrôler dynamiquement la vitesse de votre robot, position, cap relatif (heading), et d'autres paramètres de mouvement soit par de simples commandes de bas niveau ou à travers son infrastructure haut niveau des actions. ARIA reçoit également des estimations de position, des lectures de sonar, et toutes les autres données de fonctionnement actuelles envoyés par la plate-forme du robot.

ARIA est livré avec un code source complet sous une licence GNU. La licence permet la redistribution du code aussi longtemps que ce dernier est distribué gratuitement. ARIA comprend un manuel de référence API complet et beaucoup d'exemples de code. ARIA doit être installé sur le PC où l'on compte compiler et tester les codes.

L'application Mobilesim permet de tester en simulation avant de "uploader" sur le robot réel pour les tests. Lors de ce TP vous procéderez au test et la validation par l'enseignant à chaque étape nécessaire.

1. Avant toute opération il faudra se déclarer et se connecter correctement au robot. Le code suivant permet d'arriver à ce résultat :

```
#include "Aria.h"


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

  Aria::init();
  ArRobot robot;
  ArArgumentParser parser(&argc, argv);
  parser.loadDefaultArguments();

  ArLog::log(ArLog::Terse, "WARNING: this program does no sensing or avoiding
  of obstacles, the robot WILL collide with any objects in the way! Make sure
  the robot has approximately 3 meters of free space on all sides.");

  // ArRobotConnector connects to the robot, get some initial data from it
  such as type and name,
  // and then loads parameter files for this robot.
  ArRobotConnector robotConnector(&parser, &robot);
  if(!robotConnector.connectRobot())
  {
    ArLog::log(ArLog::Terse, "simpleMotionCommands: Could not connect to the
    robot.");
    if(parser.checkHelpAndWarnUnparsed())
    {
        Aria::logOptions();
        Aria::exit(1);
    }
  }
  if (!Aria::parseArgs())
```

```
{
  Aria::logOptions();
  Aria::shutdown();
  return 1;
}

ArLog::log(ArLog::Normal, "simpleMotionCommands: Connected.");

// Start the robot processing cycle running in the background.
// True parameter means that if the connection is lost, then the
// run loop ends.
robot.runAsync(true);

// Print out some data from the SIP.

// We must "lock" the ArRobot object
// before calling its methods, and "unlock" when done, to prevent conflicts
// with the background thread started by the call to robot.runAsync() above.
// See the section on threading in the manual for more about this.
// Make sure you unlock before any sleep() call or any other code that will
// take some time; if the robot remains locked during that time, then
// ArRobot's background thread will be blocked and unable to communicate with
// the robot, call tasks, etc.

robot.lock();
ArLog::log(ArLog::Normal, "simpleMotionCommands: Pose=(%.2f,%.2f,%.2f), Trans.
Vel=%.2f, Rot. Vel=%.2f, Battery=%.2fV", robot.getX(), robot.getY(),
robot.getTh(),  robot.getVel(), robot.getRotVel(), robot.getBatteryVoltage());
robot.unlock();

// Sleep for 3 seconds.
ArLog::log(ArLog::Normal, "simpleMotionCommands: Will start driving in 3
seconds...");
ArUtil::sleep(3000);

// Set forward velocity to 50 mm/s
ArLog::log(ArLog::Normal, "simpleMotionCommands: Driving forward at 250 mm/s
for 5 sec...");
robot.lock();
robot.enableMotors();
robot.setVel(250);
robot.unlock();
ArUtil::sleep(5000);

ArLog::log(ArLog::Normal, "simpleMotionCommands: Stopping.");
robot.lock();
robot.stop();
robot.unlock();
ArUtil::sleep(1000);
```

```
ArLog::log(ArLog::Normal, "simpleMotionCommands: Rotating at 10 deg/s for
5 sec...");
robot.lock();
robot.setRotVel(10);
robot.unlock();
ArUtil::sleep(5000);

ArLog::log(ArLog::Normal, "simpleMotionCommands: Rotating at -10 deg/s
for 10 sec...");
robot.lock();
robot.setRotVel(-10);
robot.unlock();
ArUtil::sleep(10000);

ArLog::log(ArLog::Normal, "simpleMotionCommands: Driving forward at
150 mm/s for 5 sec...");
robot.lock();
robot.setRotVel(0);
robot.setVel(150);
robot.unlock();
ArUtil::sleep(5000);

ArLog::log(ArLog::Normal, "simpleMotionCommands: Stopping.");
robot.lock();
robot.stop();
robot.unlock();
ArUtil::sleep(1000);


// Other motion command functions include move(), setHeading(),
// setDeltaHeading().  You can also adjust acceleration and deceleration
// values used by the robot with setAccel(), setDecel(), setRotAccel(),
// setRotDecel().  See the ArRobot class documentation for more.


robot.lock();
ArLog::log(ArLog::Normal, "simpleMotionCommands: Pose=(%.2f,%.2f,%.2f),
Trans. Vel=%.2f, Rot. Vel=%.2f, Battery=%.2fV", robot.getX(), robot.getY(),
robot.getTh(), robot.getVel(), robot.getRotVel(), robot.getBatteryVoltage());
robot.unlock();


ArLog::log(ArLog::Normal, "simpleMotionCommands: Ending robot thread...");
robot.stopRunning();

// wait for the thread to stop
robot.waitForRunExit();
```

```
  // exit
  ArLog::log(ArLog::Normal, "simpleMotionCommands: Exiting.");
  return 0;
}
```

Notons que :

(a) "Aria.h" doit être inclut dans tous les programme afin de permettre l'utilisation de la librairie ARIA. Pour être utilisée elle doit s'initialiser avec Aria : :init()

(b) La classe ArRobot (instanciée en ligne 2 de la fonction main) est la base pour créer les objets robot auxquels vous allez vous connecter via les devices.

(c) La classe ArRobot exécute une boucle (soit dans le Thread courant en utilisant la méthode ArRobot : :run () ou dans un thread en arrière-plan en utilisant ArRobot : :RunAsync ()), qui est synchronisé avec les mises à jour de données envoyées par le robot au micro-contrôleur. Dans le programme ci-dessus le ArRobot : :RunAsync () est utilisé après que la connexion a été établie. Utiliser le robot en asynchrone garantit que si la connexion est perdue, le robot s'arrête.

(d) Notez que les commandes de mouvement émise explicitement à l'aide des commande setVel(), setVel2() et setRotVel(). Ces méthodes appartiennent à la classe ArRobot. La méthode setVel() définit le choix de la vitesse de translation du robot en millimètres par seconde, setVel2() définit la vitesse des roues de manière indépendante et setRotVel() définit la vitesse de rotation du robot en degrés par secondes.

(e) Une source documentaire complète existe sur votre ordinateur sur les méthodes utilisées dans : c :/program files/mobile robots/aria/docs/index.html

2. Testez le fonctionnement de votre installation en compilant puis en lançant le programme SimpleMotionCommand.cpp fourni en simulation. Notez les variables utilisées par le robot.
3. Ecrire un programme qui permet d'effectuer un carré en utilisant les commandes de vitesses précédentes. Le tester en simulation. Le faire valider. Afficher la trajectoire effectuée.
4. Charger le projet Controlrobot1 pour effectuer une ligne droite. Le stockage des données s'effectue via matlab. Vérifier la poursuite de trajectoire.
5. Proposer une méthode pour vérifier le respect des cadences temps réel. Modifier la variable cycle du code pour valider vos constatations.
6. Modifier cette trajectoire désirée pour effectuer un carré. Il est possible de travailler sur des tests sur la variable temps
7. Effectuer une trajectoire circulaire.
8. Refaire le fonctionnement wander de la démo en utilisant la génération de trajectoire linéaire couplée à des appels de fonctions mesures capteurs comme getRangeDevices() et/ou getSonarRange(). Faire la démo correspondante à valider par l'enseignant.