Control Pioneer 3 Robots under ROS

Fei Liu

MARHES Lab, Department of Electrical and Computer Engineering
University of New Mexico
07/23/2014

Content

Introduction

This document shows how to control pioneer 3 robots (http://www.mobilerobots.com/ResearchRobots/P3AT.aspx) under ROS (http://wiki.ros.org). All the work described in this document is done in MARHES Lab, Department of Electrical and Computer Engineering, University of New Mexico from September 2013 to August 2014.

PART 1: Previous Preparation

This part includes installing **ROS** and **ROSARIA** on local computer and the on-board computer on the robot. The operating systems of both computers are Ubuntu 12.04 LTS.

Step 1: Install ROS

Follow the steps on http://wiki.ros.org/hydro/Installation/Ubuntu to install ROS (version: Groovy/Hydro) on your local computer and the on-board computer.

---

*Notes:

1) It's suggested to install the same version of ROS on both computers.

2) The major ROS versions released so far are

   http://en.wikipedia.org/wiki/Robot_Operating_System#Version_History

   1> 4 September 2013 – Hydro Medusa

   2> 31 December 2012 – Groovy Galapagos

   3> 23 April 2012 – Fuerte

   4> 30 Aug 2011 – Electric Emys

   5> 2 March 2011 – Diamondback

   6> 3 August 2010 – C Turtle

   7> 1 March 2010 – Box Turtle

   8> 22 January 2010 – ROS 1.0

   So, Hydro and Groovy are two suggestions for you.

3) To get the version infromation of ROS that is installed on your computer, run

   $ rosversion ros

Step 2: Install ROSARIA

1) Introduction of ROSARIA

ROSARIA (http://wiki.ros.org/ROSARIA) provides a ROS interface for most Adept MobileRobots, MobileRobots Inc., and ActivMedia mobile robot bases including Pioneer 2, **Pioneer 3** etc. that are supported by Adept MobileRobot's open source ARIA library(http://robots.mobilerobots.com/wiki/ARIA) (ARIA: MobileRobots' Advanced Robot Interface for Applications).

Information from the robot base, and velocity and acceleration control, is implemented via a *RosAria* node, which publishes topics providing data recieved from the robot's embedded controller by ARIA , and sets desired velocity, acceleration and other commands in ARIA when new commands are received from command topics.

2) Steps for installing ROSARIA

Follow the link below to install ROSARIA:

http://wiki.ros.org/ROSARIA/Tutorials/How%20to%20use%20ROSARIA

Collection of all the commands

1) Create a workspace. If a workspace exists already, skip to step 2).

$. /opt/ros/**hydro**/setup.bash

$ mkdir -p ~/**catkin_ws**/src

$ cd ~/catkin_ws/src

$ catkin_init_workspace

$ cd ~/catkin_ws

$ catkin_make

$ cd ~/catkin_ws/src

2) install rosaria package:

$ git clone https://github.com/amor-ros-pkg/rosaria.git

$ source ~/catkin_ws/devel/setup.bash

$ echo "source ~/catkin_ws/devel/setup.brash" >> ~/.bashrc

$ source ~/.bashrc

% *rosdep* may install any additional ROS packages not currently installed

$ rosdep update

$ rosdep install rosaria

$ cd ~/catkin_ws/

$ catkin_make

---

*Notes:

1)   After installation, the Aria library will be found in /usr/local/.

2)   After installation, three important files should be found in /catkin_ws/src/rosaria:

$ cd ~/catkin_ws/src/rosaria/

$ ls

  CMakeLists.txt

  RosAria.cpp

  package.xml

PART 2: Connect robot via wireless network

Step 1: Configuration

Set the IP addresses for all computers and make sure that they are connected to the same wireless network. Table 1 shows an example of the configuration.

Table 1. Example of configuration of local and on-board computer

	Local Computer	Robot(on-board computer)
IP	192.168.0.*** (e.g. 192.168.0.119)	192.168.0.*** (e.g.192.168.0.160)
OS	Ubuntu 12.04 LTS	Ubuntu 12.04 LTS
ROS	ROS Hydro/Groovy	ROS Hydro/Groovy
RosAria	installed	installed

Step 2: Connect the robot

Open a terminal and use "ssh" to access the on-board computer:

```
$   ssh -X -l root pioneer@192.168.0.160
%  pioneer@192.168.0.160's password: marhes
```

---

*Notes:

1) You can also type *ssh pioneer@192.168.0.160* to access the robot.

2) The parameters *–X –l root* will let you access the robot as the super user, where the most benefit is that after accessing the robot, you can use gedit to open and edit documents that are on the on-board computer.

PART 3: Control the robot

Step 1: Activate the robot

Run 'RosAria' by following the steps shown in Table 2. By running RosAria, the onboard computer will establish the communication between the lower-level microcontroller (controlling the motors and other sensors) and itself, and the robot is ready to receive the command in order to run.

Table 2. Running *RosAria*.

	Local Computer	Robot(Onboard Computer)
IP address	e.g. 192.168.0.119	e.g.192.168.0.160
Commands Steps ①à④	On local terminal: ① $ roscore	On remote terminal: ② $ export ROS_MASTER_URI=http://192.168.0.**119**:11311 ③ $ source ~/catkin_ws/devel/setup.bash ④ $ rosrun rosaria RosAria _port:=/dev/ttyS0
*Notes	\begin{tabular}1) *roscore* is the first step for running any code under ROS;\\ 2) If you run roscore on remote terminal, then you don't need to set the master (export\\ ROS_MASTER_URI=http://192.168.0.119:11311).\\ 3) If you get an error like "*Could not open serial port '/dev/ttyS0'*", run the following\\ command on on-board computer:\\ $ sudo chmod a+rw /dev/ttyS0\\ Then try to run $ rosrun rosaria RosAria _port:=/dev/ttyS0\end{tabular}		

Tip: Files transfer between local and remote computer

Sometimes you can edit the code/file on the local computer, then copy the file to the remote computer by using the command:

$ scp   file_name.cpp pioneer@192.168.0.191:~/catkin_ws/src/rosaria

where

1) file_name.cpp   : the files to be sent

2) pioneer@192.168.0.191: referring the remote computer

3) ~/catkin_ws/src/rosaria: the location where to save the files

> ***Note :**
> if it is denied to transfer the file, you may need to run the following command:
> $ sudo chown pioneer rosaria
> which is to make sure that the folder (e.g. rosaria) is fully accessible to the user "pioneer"

Step 2: Make the robot run

Here is to illustrate how to control the robot move via keyboard.

1. **Highlights of codes**

1)   **Highlights in RosAria.cpp**
https://github.com/amor-ros-pkg/rosaria/blob/master/RosAria.cpp

```cpp
class RosAriaNode
{
  public:
    …...
    void cmdvel_cb( const geometry_msgs::TwistConstPtr &);
    …...
  protected:
    ros::Subscriber cmdvel_sub;
}


// subscribe to services
cmdvel_sub = n.subscribe( "/RosAria/cmd_vel", 10, (boost::function
<void(const
geometry_msgs::TwistConstPtr&)>)boost::bind(&RosAriaNode::cmdvel_cb,
this, _1 ));


void RosAriaNode::cmdvel_cb( const geometry_msgs::TwistConstPtr &msg)
{
    …...
}
```

2) **Highlights in rob_key.cpp**
https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/rob_key.cpp

```cpp
ros::Publisher vel_pub;
…...
int main(int argc, char** argv)
{
    vel_pub = n.advertise<Twist>("/RosAria/cmd_vel", 1);
    …...
    vel_pub.publish(vel);
    ……
}
```

2. Run "RosAria" and "rob_key"
1) On remote terminal:
```
$ rosrun rosaria RosAria  _port:=/dev/ttyS0
```

2) On local terminal:

   $ rosrun rosaria rob_key

Now, you can use the keys '↑' '↓' '←' and '→' to control the robot move "forward","backward", turn "left" and "right"respectively, and use 'SPACE' to stop the robot.

---

*Note:

To compile *rob_key.cpp*, you need to add the following four commands in *CMakeLists.txt*:

add_executable(**rob_key rob_key.cpp**)

add_dependencies(**rob_key** rosaria_gencfg)

dd_dependencies(**rob_key** rosaria_gencpp)

target_link_libraries(**rob_key** ${catkin_LIBRARIES} ${Boost_LIBRARIES} Aria pthread dl rt)

PART 4: Simulation environment

A. Simulation in MobileSim

MobileSim (http://robots.mobilerobots.com/wiki/MobileSim) is software for simulating MobileRobots/ActivMedia platforms and their environments, for debugging and experimentation with ARIA.

1)   Download (e.g., mobilesim_0.7.3+ubuntu12+gcc4.6_i386.deb) and install MobileSim. (http://robots.mobilerobots.com/wiki/MobileSim)

2)   Run MobileSim by typing "MobileSim" in a terminal.

3)   Choose a map (here are two default 2-D maps located at "/usr/local/MobileSim/") and load it or just select "No Map".

4)   Open terminals and run

   1> $roscore

   2> $ source ~/catkin_ws/devel/setup.bash

   3> $ rosrun rosaria RosAria

5)   Now the virtual robot is ready to run as a real robot.

6)   Follow the steps in "Step 2, PART 3" to control the robot.

B. Simulation in Gazebo

Please refer to http://wiki.ros.org/sig/robots/Pioneer and https://github.com/dawonn/ros-pioneer3at for details of simulation in Gazebo.

PART 5: Experiments

Here is to illustrate 6 examples of operating pioneer 3 robots.

Based on the basic package *rosaria*, more experiments are implemented on pioneer 3 robots. The codes are available on

https://github.com/drfeiliu/Pioneer3_Control_ROS.

To get the codes:

$ *git clone https://github.com/drfeiliu/Pioneer3_Control_ROS.git*

---

*Note:

The file "RosAria.cpp" on https://github.com/drfeiliu/Pioneer3_Control_ROS is different from that on https://github.com/amor-ros-pkg/rosaria. The latter one is same with "**RosAria_origin.cpp**" which is divided into two files "RosAria.cpp" and "AriaRobot.h".

A. A simple demo

By running the demo, it can be chosen to control the robot via keyboard, let the robot wander, show the sonar/laser/bumper or position readings and control the gripper etc. you can switch to different modes by typing specific keys.

1. Highlights of codes

1) In *robotDemo.cpp*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/robotDemo.cpp.

```cpp
#include "AriaRobot.h"
int main( int argc, char** argv )
{
    ……
    node->robotDemo();
    ……
    node->spin();
    ……
}
```

2) In *AriaRobot.h*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/AriaRobot.h.

```cpp
void RosAriaNode::robotDemo()
{
  ……
// now add all the modes
    //ArModeLaser laser(&robot, "laser", 'l', 'L', &sick);
    ArModeTeleop teleop(robot, "teleop", 't', 'T');
    ArModeUnguardedTeleop unguardedTeleop(robot, "unguarded teleop", 'u',
'U');
    ArModeWander wander(robot, "wander", 'w', 'W');
    ArModeGripper gripper(robot, "gripper", 'g', 'G');
    ArModeCamera camera(robot, "camera", 'c', 'C');
    ArModeSonar sonar(robot, "sonar", 's', 'S');
    ArModeBumps bumps(robot, "bumps", 'b', 'B');
    ArModePosition position(robot, "position", 'p', 'P', &gyro);
    ArModeIO io(robot, "io", 'i', 'I');
    ArModeActs actsMode(robot, "acts", 'a', 'A');
    ArModeCommand command(robot, "command", 'd', 'D');
    ArModeTCM2 tcm2(robot, "tcm2", 'm', 'M');
    ……
}
```

2. Steps to run the demo

1) on local terminal:

$ roscore

2) on remote terminal:

$ export ROS_MASTER_URI=http://192.168.0.**119**:11311

$ source ~/catkin_ws/devel/setup.bash

% if it is simulating in MobileSim, run *rosrun Pioneer3_Control_ROS robotDemo*

$ rosrun Pioneer3_Control_ROS robotDemo   _port:=/dev/ttyS0

---

*Note:

In command "*rosrun Pioneer3_Control_ROS robotDemo   _port:=/dev/ttyS0*",

Pioneer3_Control_ROS is the package name, so here it is different from the commands in

PART3 AND PART4, where the command is *rosrun rosaria Rosaria*.

And it is noted that in the following parts, all the package names are *Pioneer3_Control_ROS*.

B. Wandering

This will make the robot wander in a clutered environment.

1. Highlights of codes

1) In *robotWander.cpp*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/robotWander.cpp.

```cpp
#include "AriaRobot.h"
int main( int argc, char** argv )
{
    ……
    node->robotWander();
    ……
    node->spin();
    ……
}
```

2)   In *AriaRobot.h*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/AriaRobot.h.

```cpp
void RosAriaNode::robotWander()
{
    ……
    //puts("This program will make the robot wander around. It uses some avoidance\n"
    //"actions if obstacles are detected, otherwise it just has a\n"
    //"constant forward velocity. \n\nPress Escape to exit.");

  ArActionStallRecover recover;
    //ArActionBumpers bumpers;
  ArActionAvoidFront avoidFrontNear("Avoid Front Near", 500, 0);
  ArActionAvoidFront avoidFrontFar;
  ArActionConstantVelocity constantVelocity("Constant Velocity",150);

  ArActionLimiterForwards limiter("speed limiter near", 300, 600, 250);
    // limiter for far away obstacles
    //ArActionLimiterForwards (const char *name="speed limiter", double stopDistance=250, double slowDistance=1000, double slowSpeed=200, double widthRatio=1)
  ArActionLimiterForwards limiterFar("speed limiter far", 300, 1100, 200);
```

```
   robot->addRangeDevice(&sonar);

   robot->addAction(&limiter, 95);
   robot->addAction(&limiterFar, 90);
   robot->addAction(&recover, 100);
   //robot->addAction(&bumpers, 75);
   robot->addAction(&avoidFrontNear, 50);
   robot->addAction(&avoidFrontFar, 49);
   robot->addAction(&constantVelocity, 25);
   ……
}
```

2. Steps to run *robotWander*

1) on local terminal:
  $ roscore

2) on remote terminal:
  $ export ROS_MASTER_URI=http://192.168.0.**119**:11311
  $ source ~/catkin_ws/devel/setup.bash
  % if it is simulating in MobileSim, run *rosrun Pioneer3_Control_ROS robotWander*
  $ rosrun rosaria Pioneer3_Control_ROS _port:=/dev/ttyS0

C. Reading sonar data

This part wil show how to read sonar data from pioneer 3 robot.

1. Highlights of codes

In *p3_sonar.cpp*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/p3_sonar.cpp

```cpp
#include <ros/ros.h>
#include <sensor_msgs/PointCloud.h>    //for sonar data
#include <math.h>
#include <iostream>

using namespace std;

//in this case, we just use 8 sonars that are in front of the robot
#define SONAR_NUM 8

int offset[SONAR_NUM] ={160, 220, 240, 240, 240, 240, 220, 160};

//sensor_msgs::PointCloud sonarData;
int seq = 0;

void get_sonarData_Callback(const sensor_msgs::PointCloud::ConstPtr
&sonarScanedData)
{
  float tmpX = 0.0, tmpY=0.0;
    seq = sonarScanedData->header.seq;

    printf("seq of sonar beam and  distance measured-->\n");
    printf("Frame[%d] :  \n", seq);
    for (int i=0; i<SONAR_NUM; i++)
    {
        printf("%d\t",i);
    }
    printf("\n");

    for (int i=0; i<SONAR_NUM; i++)
    {
        tmpX= sonarScanedData->points[i].x; //coordinate x
        tmpY= sonarScanedData->points[i].y; //coordinate y
        distToObstace[i] = int(sqrt(tmpX*tmpX+tmpY*tmpY)*1000-offset[i]);
```

```cpp
            printf("%d\t",distToObstace[i]);
      }
      printf("\n\n");
}


int main(int argc,  char** argv)
{
   ros::init(argc,  argv,  "p3_sonar");
   ros::NodeHandle n;

   ros::Subscriber get_sonar_data =
n.subscribe<sensor_msgs::PointCloud>("RosAria/sonar", 100,
get_sonarData_Callback);
   printf("\n********* Sonar Readings: *********\n");

   while (ros::ok())
   {
       ros::Duration(0.2).sleep();
       ros::spinOnce();
   }
   return 0;
}
```

2. Steps to run *p3_sonar*

1) on local terminal:
   $ roscore
2) on remote terminal:
   $ export ROS_MASTER_URI=http://192.168.0.**119**:11311
   $ source ~/catkin_ws/devel/setup.bash
   % if it is simulating in MobileSim, run *rosrun Pioneer3_Control_ROS RosAria*
   $ rosrun Pioneer3_Control_ROS RosAria _port:=/dev/ttyS0
3) on local terminal
   $ rosrun Pioneer3_Control_ROS p3_sonar

---

*Note: about the format of sonar data
1. in AriaRobot.h
 sonar_pub = n.advertise<**sensor_msgs::PointCloud**>("sonar", 50,
   boost**::**bind(**&**RosAriaNode**::**sonarConnectCb, **this**),
   boost**::**bind(**&**RosAriaNode**::**sonarConnectCb, **this**));
2.The details of the message read from the sonar, **sensor_msgs/PointCloud** is located at:
http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud.html

D. Following a man

This part shows how to make the robot follow a moving object (e.g., a man) by simply using the sonars (8 sonars in front of the robot).

1. Highlights of codes

In *p3_sonar_following_moving_object.cpp*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/p3_sonar.cpp

```cpp
//only the 8 front sonars are used
//                       8 front sonar beams
//                          3 4                   +/-10 degree
//                        2   ^   5               +/-30 degree
//                      1     |     6             +/-50 degree
//                    ___0____center_____7___     +/-90 degree
//                     |            |          |
//                     |            |          |
//                     |            |          |
//                     |       P3 ROBOT        |
//                     |            |          |
//                     |            |          |
//                     |            |          |
//                     |            |          |
//                     |_____|_____|
// distance offset(mm):
//rays no.: 0   1   2   3   4   5   6   7
//offset   :160 220 240 240 240 240 220 160

void FollowingObject();

int main(int argc, char** argv)
{
    ros::init(argc, argv, "p3_sonar");
    ros::NodeHandle n;

    ros::Publisher vel_pub;
    vel_pub = n.advertise<Twist>("/RosAria/cmd_vel", 1);
```

```
      ros::Subscriber get_sonar_data =
n.subscribe<sensor_msgs::PointCloud>("RosAria/sonar", 100,
Get_sonarData_Callback);
    while (ros::ok())
  {
      ......
      //execute following the object
      FollowingObject();
      vel_pub.publish(vel);
  }
    return 0;
}


void FollowingObject()
{
    ifObjectDetected = CheckIfObjectDetected();

    if (ifObjectDetected)
    {//if object detected
#ifdef DEBUG_PRINT
        printf("Object found!\n");
#endif

        CalculateDistaneAndHeading();
        Heading_Vel_Determination();
        Angular_Vel_Determination();

        /*Make sure the heading vel is between
         HEADING_VEL_MIN and HEADING_VEL_MAX*/
        if (heading_vel > HEADING_VEL_MAX)
        {
            heading_vel =  HEADING_VEL_MAX;
        }
        else if (heading_vel <  HEADING_VEL_MIN)
        {
            heading_vel =  HEADING_VEL_MIN;
        }

        /*Make sure the angular vel is between
         ANGULAR_VEL_MIN and ANGULAR_VEL_MAX*/
        if (angular_vel > ANGULAR_VEL_MAX)
        {
            angular_vel = ANGULAR_VEL_MAX;
        }
```

```cpp
        else if (angular_vel < ANGULAR_VEL_MIN)
        {
            angular_vel = ANGULAR_VEL_MIN;
        }

        /*for publishing the vel*/
        vel.linear.x = heading_vel;
        vel.angular.z  = angular_vel;
    }
    else
    {//if object disappears, stop
        vel.linear.x = 0;
        vel.angular.z  = 0;
#ifdef DEBUG_PRINT
        printf("The object disappears!\n");
#endif
    }
#ifdef DEBUG_PRINT
    printf("vel.linear.x = %f,  vel.angular.z = %f\n",vel.linear.x,
vel.angular.z);
#endif
}
```

2. Steps to run p3_sonar_following_moving_object.cpp

1) on local terminal:
 $ roscore
2) on remote terminal:
 $ export ROS_MASTER_URI=http://192.168.0.**119**:11311
 $ source ~/catkin_ws/devel/setup.bash
 % if it is simulating in MobileSim, run *rosrun Pioneer3_Control_ROS RosAria*
 $ rosrun Pioneer3_Control_ROS RosAria _port:=/dev/ttyS0
4) on local terminal
 $ rosrun Pioneer3_Control_ROS p3_sonar_following_moving_object

E. Reading laser data

A specific node *hokuyo_node* is provided to control the Hokuyo laser range finder under ROS. Please refer to http://wiki.ros.org/hokuyo_node to see the details. The source codes are located at https://github.com/ros-drivers/hokuyo_node.

---

*Note:

The details of the message read from the laser, **sensor_msgs/LaserScan Message** is located at:

http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html

F. Control robot via android device

1. Please refer to the link below to see the details.

http://wiki.ros.org/ROSARIA/Tutorials/iPhone%20Teleop%20with%20ROSARIA/Android%20teleop%20Pioneer%203att

2. Steps to run *android_teleop*

Code: https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/android_teleop.cpp

1) on local terminal:
```
$ roscore
```
2) on remote terminal:
```
$ export ROS_MASTER_URI=http://192.168.0.119:11311
$ source ~/catkin_ws/devel/setup.bash
% if it is simulating in MobileSim, run rosrun Pioneer3_Control_ROS RosAria
$ rosrun Pioneer3_Control_ROS RosAria _port:=/dev/ttyS0
```
3) **on local terminal**
```
$ rosrun Pioneer3_Control_ROS android_teleop
```
4) use your android device

   Make sure that all devices are assumed to be connected to the same Wireless network. Turn your phone right or left to move the robot right or left. Turn it up or down to move the robot forward or backward. Flip it up or down to stop the Robot.

3.   Follow this link to use you iphone to control the robot.

http://wiki.ros.org/ROSARIA/Tutorials/iPhone%20Teleop%20with%20ROSARIA/iPhone%20Teleop%20With%20ROSARIA

G. Control multiple robots

Here shows a simple method to control multiple robots simultaneously. It establishes a unique topic for each robot and can choose which robot to send message to, thus multiple robots can be controlled at the same time. Fig. 1 shows the diagram of multiple robots control.
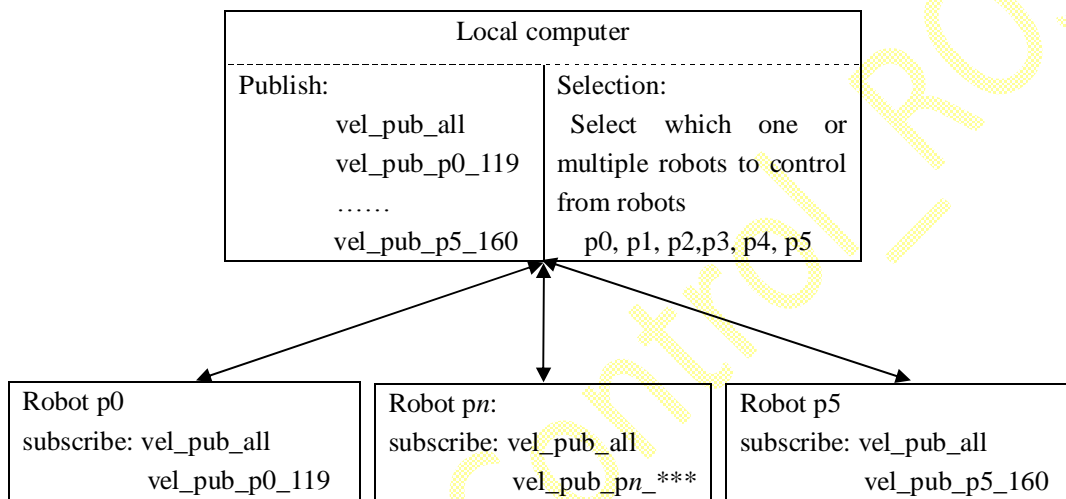
Local computer	
Publish:	Selection:
vel_pub_all	Select which one or
vel_pub_p0_119	multiple robots to control
……	from robots
vel_pub_p5_160	p0, p1, p2,p3, p4, p5

Robot p0	Robot p*n*:	Robot p5
subscribe: vel_pub_all	subscribe: vel_pub_all	subscribe: vel_pub_all
vel_pub_p0_119	vel_pub_p*n*_***	vel_pub_p5_160

Figure 1 diagram of multiple robots control

1. Highlights of codes

1) In *rob_key_multi_robot_control.cpp*

https://github.com/drfeiliu/Pioneer3_Control_ROS/blob/master/rob_key_con_multiple_robots.cpp

```
//Create a unique topic for each robot.

ros::Publisher vel_pub_all;    // for all the robots
ros::Publisher vel_pub_p0_119; // for robot with ip 192.168.0.119
............
ros::Publisher vel_pub_p5_160; // for robot with ip 192.168.0.160


vel_pub_all = n.advertise<Twist>("/RosAria/cmd_vel", 1);
vel_pub_p0_119 = n.advertise<Twist>("/RosAria/cmd_vel_p0_119", 1);
............
vel_pub_p5_160 = n.advertise<Twist>("/RosAria/cmd_vel_p5_160", 1);
```

2) In *AriaRobot.h* (e.g., on a robot with IP 192.168.0.119)

```
void cmdvel_cb( const geometry_msgs::TwistConstPtr &);
void cmdvel_cb_p0_192_168_0_119( const geometry_msgs::TwistConstPtr &);
ros::Subscriber cmdvel_sub;
ros::Subscriber cmdvel_sub_p0_119;


// subscribe to services
cmdvel_sub = n.subscribe( "/RosAria/cmd_vel", 10, (boost::function<void(const
geometry_msgs::TwistConstPtr&)>)
    boost::bind(&RosAriaNode::cmdvel_cb, this, _1 ));
//added by Fei, 05/16/2014
cmdvel_sub_p0_119 = n.subscribe( "/RosAria/cmd_vel_p0_119", 10,
(boost::function <void(const geometry_msgs::TwistConstPtr&)>)
    boost::bind(&RosAriaNode::cmdvel_cb_p0_192_168_0_119, this, _1 ));
```

2. Steps to run *rob_key_multi_robot_control*

1) on local terminal:
   $ roscore
2) on each remote terminal:
   $ export ROS_MASTER_URI=http://192.168.0.119:11311
   $ source ~/catkin_ws/devel/setup.bash
   % if it is simulating in MobileSim, run *rosrun Pioneer3_Control_ROS RosAria*
   $ rosrun Pioneer3_Control_ROS RosAria _port:=/dev/ttyS0
3) **on local terminal**
   $ rosrun Pioneer3_Control_ROS **rob_key_multi_robot_control**
4) Follow the tips on the terminal to choose one robot or multiple robots to control.