Multi-Agent Systems Using Kobuki Turtlebots: Leader-Follower Premise

Sinchiguano Cesar

***Abstract*— Multi-agent system is a topic that has generated a lot interest in the research community. This interest is because a multi-agent systems present a more robust and cheaper solution to a certain tasks that are better performed using several low-cost robots rather than a single one. Multi-agent system can be achieved through many approaches. The approach used in this paper is the one based on the leader-follower premise.**

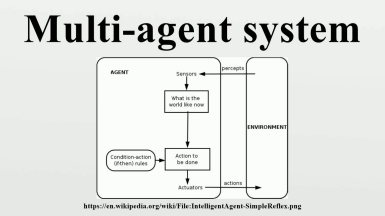

Fig. 1. An example of multi-agent system. [1]

I. Introduction

Multi-robot systems present a more robust and cheaper solution to certain tasks that are better performed using several low-cost robots rather than a single one. This gives rise to the formation control problem. The formation problem has been regarded as an important problem in multi-robot systems where the goal is to make a fleet of autonomous mobile robots move toward and maintain a desired geometric shape. According to the survey presented in (Guanghua et al., 2013), and the references therein, formation structure can be divided into three strategies: the leaderfollower strategy, the behavioural and the virtual structure approaches. Several approaches have been proposed in the literature to solve this problem. However, most of the existing literature tackle the theoretical side of the problem mainly the controller design. Nevertheless, some of them have carried on real experiments to prove the effectiveness of their proposed controller.

In this paper, we propose the use of ROS as a new framework so that real experiments for the formation control problem can be conducted effectively. Due to its simplicity and scalability, the leader-follower approach is considered in this paper also. However, the proposed framework can be extended so that other formation strategies or consensus algorithm can be implemented. The rest of this paper will be organized as follow: The introduction about ROS concepts is highlighted in Section II, with the software setup in Section III, followed by a detailed explanation of the software processes and the needed ROS packages are presented in Section IV, finally conclusions drawn from this experiment and some future work directions in Section VI

1) **Log.....**
   Gi [1].....
2) **AdaBoost classifier**
   AdaBoost.....:
3) **Random Forest classifier**
   Th.....
4) **Naive Bayesian (NB) Classifier**

II. Backgroung

1) **Robotic Operating System (ROS)**
   ROS is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behaviour across a wide variety of robotic platforms . It is based on the concepts of nodes, topics, messages and services. A node is an executable program that performs computation. Nodes need to communicate with each other to complete the whole task. The communicated data are called messages. ROS provides an easy way for passing messages and establishing communication links between nodes, which are running independently. They pass these messages to each other over a Topic, which is a simple string. However, topics are asynchronous, synchronous communication is provided by services. Services act in a call-response manner where one node requests that another node execute a one-time computation and provide a response. For more details about ROS, the reader can refer to [2].
2) **Architecture of the System**
   T.....

III. Experimental setup

The test space consist of two Kobuki TurtleBots. It is a low-cost, open source differential drive robot. It consists of a mobile base, an RGB-D sensor and an CPU making it a perfect entry-level mobile robot platform. The Kobuki was chosen because it is an open source UGV (Unmanned ground vehicle) platform, making it perfect for research and development. The Kobuki SDK (Software development kit) is based on ROS, which is the preferred development platform by the Intelligent and Mobile Robotics Group because of its intuitive publisher/subscriber message passing structure that allows robust and simple communication within multiple facets of a robotic system.

Fig. 2. Two turtlebots used in the experimental test bed.

The Fig 2 shows two Turtlebots which are equipped with a Kinect Sensor. The robots run ROS (Robot Operating System), and are supported by various ROS libraries.

1) Logistic Regression classifier
2) Naive Bayesian Classifier
   a) Grid Search
   b) Best parameters

IV. Software Processes

In addition to the basic nodes called turtlebot_bringup needed for running the robots, four additional nodes are indispensable for controlling the robots to achieve formation. The multi_agents node for the deploying the Turtlebot robots to work with, the tf_turtle responsible for keeping track of the robot's postures, multi_master node for data transmission and the turtle node that executes the control algorithm. The nodes are running simultaneously and thus they have to communicate to each other through ROS topics or ROS services as depicted in fig.3. Note that on each robot, all the nodes are executed under a specific namespace for the robot. The ROS parameter tf_prefix is exclusive for each robot as well.

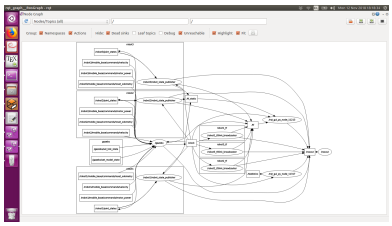

Fig. 3. A rosgraph showing what talks to what.

1) **Simulating Multiple Turtlebots in Gazebo**
   This is an example of simulating and controlling multiple Turtlebot robots. It is built as an extension of the Simulating Turtlebot tutorial on [4].

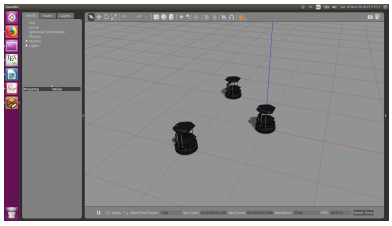

Fig. 4. A general view of the system.

a) **Quick Start**
   The main.launch file is a working example that instantiates two simulated Turtlebots in Gazebo.
   You will first need to install the following packages in your workstation:
   i) turtlebot[6].
   ii) kobuki[8].
   iii) turtlebot_simulator[7].
   iv) turtlebot_interactions[9].

   Then after you have these packages in your workspace, place the multi_agents[5] package in your src diretory.
   As long as all of the system dependencies of your packages are installed, we can now build your new packages.
   In a catkin workspace
   i) $ catkin_make.

   Before continuing source your new setup.*sh file:
   i) $ source devel/setup.bash.

   To make sure your workspace is properly overlayed by the setup script, make sure ROS_PACKAGE_PATH environment variable includes the directory you are in with the following command:
   $ echo $ROS_PACKAGE_PATH

b) **Description: multi_agents package**
   If everything was done with the commands above then it should just work. From there, to learn what it does, I woud recommend going over the multi_agent.launch, one_turle.launch and main.launch files those who can be located inside your catkin_ws with the following command:
   $ cd src/multi_agents/launch
   which will lead you to the launch files mentioned above. All of them are binding one another. The only one that it should be launched is the one named as main.launch file which does the heavy lifting of setting up each Turtlebot. Here is a high-level outline of what this package does.
   main.launch:
   i) Starts Gazebo (both the sim engine and the gui)
   ii) Start some visualization and debugging tools (rviz,rqt_console, etc.)
   iii) multi_agents.launch
      A) Accept namespace and initial pose arguments from one_turtle.launch.
      B) Set the tf_prefix based on the namespace
      C) Call a xacro robot_description with namespace and tf_prefix arguments
      D) Spawn the Turtlebot model in Gazebo using gazebo_ros
      E) Start a robot_state_publiser node

   Here are some screen captures and images to illustrate what you should see if things are working properly.
   Four Turtlebot robots in Gazebo

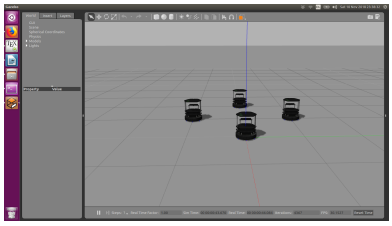

Fig. 5. A general view of the system with four multi-agents.

2) **Communication data**
When doing tasks with a robot it is crucial that the robot be aware of where it is itself as well as where the rest of the world is in relation to itself. For the leader-follower premise each robot takes another neighboring robot as a leader to determine its motion so that for the control laws proposed in this paper show that the leaders relative coordinates and velocities are needed in the control laws implemented on the followers. The package called tf_turtle will deal with it. It will start a static_transform_publisher node to establish the tf transfrom from the world to the specific odometry frame e.g robot2_tf/odom.

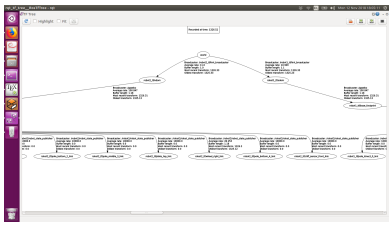

Fig. 6. View from rqt_tf_tree of the tf arrangement.

a) **Quick Start**
The robot_1.launch file is a working example that initialize the transform node for two Turtlebot robots. You will first need to install the following packages in your workstation:

i) geometry[10].

Then after you have these packages in your workspace, place the tf_turtle[11] package in your src diretory.
As long as all of the system dependencies of your packages are installed, we can now build your new package.
In a catkin workspace

i) $ catkin_make.

Before continuing source your new setup.*sh file:

i) $ source devel/setup.bash.

3) **Communication data**

V. Conclusion

j.....

References

[1]
[2]
[3] http://www.ros.org/about-ros/
[4] http://wiki.ros.org/turtlebot/Tutorials/indigo
[5] https://github.com/Sinchiguano/Multi-agent-system/tree/master/src/multi_agents
[6] http://wiki.ros.org/turtlebot
[7] http://wiki.ros.org/turtlebot_simulator
[8] http://wiki.ros.org/kobuki
[9] http://wiki.ros.org/turtlebot_interactions
[10] https://github.com/ros/geometry
[11]