THE PENNSYLVANIA STATE UNIVERSITY
Networked Robotic Systems Laboratory

Introduction of AR.Drone

Shicheng Liu

October 31, 2019

An introduction
to summarize the notification
and the methods of how to use several
equipment in the lab

Abstract

This introduction presents details of how to use AR.Drone, Vicon and the Simulator of this laboratory. I divide this introduction into three parts: AR.Drone, Vicon, Simulator; each part includes the instruction of how to use it, weird problems that may occur and possible solutions to the problems. Since you may meet some more annoying problems I have never found, this introduction is only a reference.

Contents

1 AR.Drone

This section introduces details of how to control the AR.Drone in ths lab. There are four AR.Drones in our lab but only one of them can be operated. The other three all have motors problems.

1.1 Basic Control of AR.Drone

There are two simple ways to control AR.Drone: node.js and ROS, and ROS is highly recommended because of its quantities of functions you can use. To get a quick start, you can go to this website and have fun.

The first step is always to plug in the battery: the lights under four propellers will turn red first and turn green after four times of sounds. If you find the lights still red for a long time after you plug in battery, you need to change another battery and charge this one. Then, you need to connect to the wifi of your AR.Drone(the only drone you can use is: ardrone2_128594).

Please protect the drone with foam guard.

1.2 Common Problems

Problem1: What if the drone crash while flying?

Answer1: After crash, you cannot control the drone anymore and you will find the lights of the drone red. To solve this problem, you need to unplug the battery and plug in again.

Problem2: The lights of the drone are green but the drone cannot takeoff.

Answer2: The lights will only turn red in two situations: crash and almost-zero battery. However, the drone will not takeoff if the battery is under 14 percent, to the drone in the lab, maybe more. Accordingly, you need to change a new battery and charge this one.

The lights indicate the current status of the drone and almost every problems related to the lights can be solved by unplugging and plugging in again or change batteries

Problem3: The drone will move to a certain direction after takeoff but I did not send any velocity command to it.

Answer3: This is because the drone is telling you it needs a rest. The AR.Drone and its battery are really old and sometimes may perform weirdly after long-time-operation, what you need to do is to give it a relax. Besides, since they are all old, you need to take care of them: stop when the battery is heating and do not use the drone too long.

2 Vicon

There are two ways to use vicon in this lab: matlab and ROS. Since all the students here use matlab, I will only introduce how to use ROS to interact with Vicon.

2.1 Connect to Vicon

To use AR.Drone, we can only use ROS to interect with Vicon because we need ROS to control your drone and your Ubuntu computer where ROS is installed on can only connect to one wifi(AR.Drone's wifi), so you have no ways to receive vicon data from the Windows computer where Vicon Tracker is installed on. Of course, there is another way: you can find matlab code that controls AR.Drone on the website so that you can use matlab to both receive the data of vicon and control the drone, but I never succeeded using that matlab code, maybe you can have a try.

Now, we have only one way to interact with both AR.Drone and Vicon: using ROS to directly receive vicon data and to control the drone at the same time.

To realize this, we need a ROS package called vicon_bridge. Even though you can only find indigo version of this package, you can still use the indigo package on kinetic. To know the details of how to receive data and control the drone, you can go to this website.

As I have already configured a router in the lab, you can directly use it: there is a router with orange tags and a blue wire, just plug the other side of the blue wire in your computer and you are good. If you want to configure a router yourself, you can follow the steps on the website just mentioned.

If you install vicon_bridge directly from its original author not from my repository ardrone_vicon, please go to the launch file vicon.launch and change the content as the following figure:

```
<launch>
        <node pkg="vicon_bridge" type="vicon_bridge" name="vicon" output="screen">
                <param name="stream_mode" value="ClientPull" type="str" />
                <!--param name="datastream_hostport" value="192.168.0.13:801" type="str" /-->
                <param name="datastream_hostport" value="192.168.10.1:801" type="str" />
                <param name="tf_ref_frame_id" value="/world" type="str" />
                </node>
</launch>
```

Figure 1: vicon.launch

After you have configured vicon_bridge, open Vicon Tracker on your Windows computer, select an object and follow all the steps on my repository *ardrone_vicon*.

2.2 Receive data from Vicon

After you have connected to Vicon successfully, you can now write your own ROS code to get the pose of your chosen objects. The data type transmitted by Vicon is $TransfromStamped$. It consists of 7 elements: x, y, z coordinate values and quaternion. What you need to do is to subscribe certain topics and then you can get the data!

If you are not familiar with ROS, you can go to my repositary *ardrone_vicon* (ardrone_vicon/src/ardrone_vicon.cpp) and get an idea of how to receive Vicon data.

2.3 Common Problems about Vicon

Before going to this section, I want you to have a concept that when you are using Vicon to do experiments, it is not neccessarily the problem of your code or your robots when your robots are running crazy. It might because of the Vicon! I will explain this in detail in **Problem1**.

Problem1: I have checked my code but the drone just does not follow the commands. Is it because the Vicon is not accurate?

Answer1: Vicon is always accurate, the problem occured because the Vicon sends too little data. Usually, Vicon will send at least 20 data a second, where robots can operate normally. However, Vicon may be tired sometimes and will only send one or two or even none a second. In this situation, the robots are still using old data but already in new position, as a result, the robots will not operate accurately since they do not have a good understanding of their real position.

To solve this problem, you need to restart Vicon and everything will be fine. Vicon works best when it is just turned on.

Problem2: Vicon cameras blink while conducting experiment and those blinking cameras turn red in Vicon Tracker.

Answer2: Vicon cameras will blink when they just start and they will become stable a few seconds later. If you watch Vicon Tracker, you will find those cameras are red at first and become green later. This is normal starting process.

However, if Vicon cameras blink after start(you will see those blinking cameras turning red in Vicon Tracker), that means Vicon is dead now. What you need to do is to close Vicon Tracker and restart it(you need to restart even if only one camera is dead).

If Vicon is still abnormal after restart, you need to unplug their power(there are 8 wires connecting to each camera and you need to unplug all of them) and plug in again. Please be sure that the Vicon Tracker is turned off when you unplug the power.

Problem3: When I restart Vicon Tracker, there is an error showing about you cannot open a second Vicon Tracker cause you have already opened one.

Answer3: You might feel weird cause you have already turn off Vicon Tracker. What you need to do is to restart the Windows computer and unplug Vicon power and plug in again.

Basically, any Vicon problems can be solved by restarting the Vicon Tracker software or unplug the power and plug in again. If these still do not work, you need to unplug the power and let Vicon rest for 20 minutes and then plug in again.

3 Simulator

The simulator can control three kinds of robots: AR.Drone, Hectorquadrotor, and iRobot-create(which represents khepera in the lab). You can go to the first two repositaries to install the simulator and have fun. The third simulation is not public yet and are not allowed to be spread. A simulator consists of two main parts: environment and robots. This section will tell you how to create an environment and how to spawn robots in your environment. We use

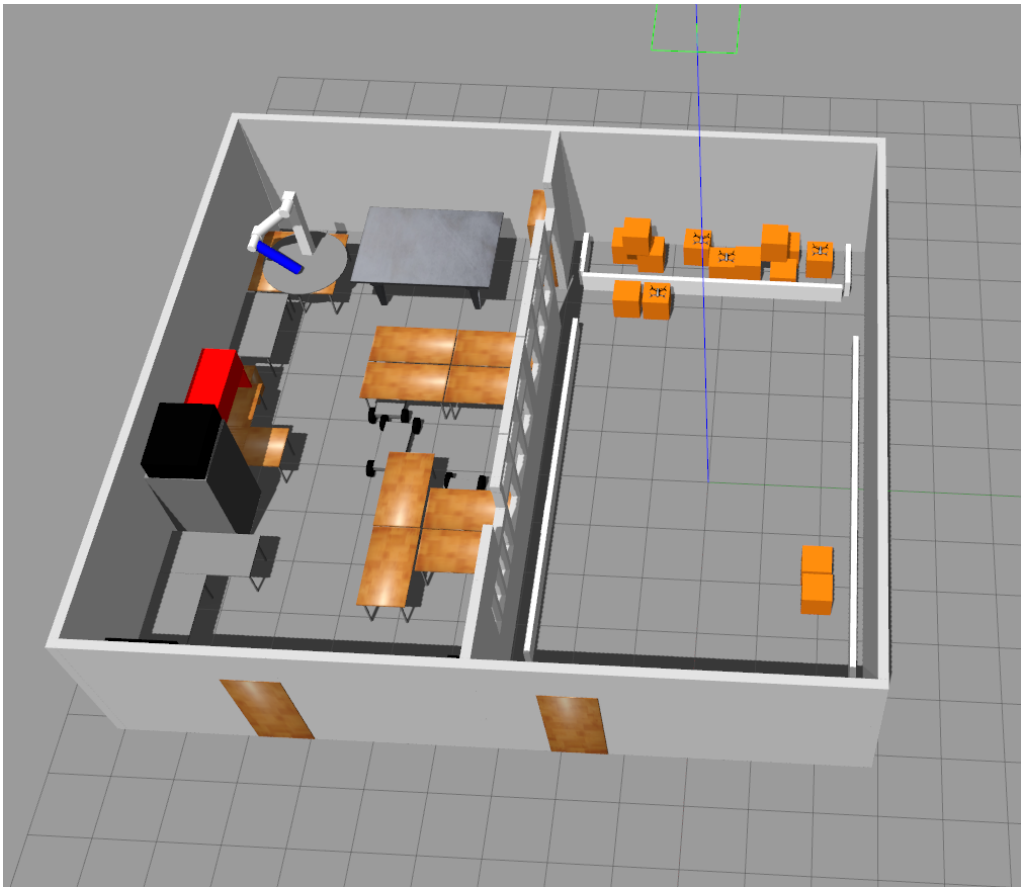

Figure 2: Laboratory Environment

gazebo to create the simulation and ROS to control the robots.

3.1 Revise the simulator environment

Here, we already have a coarse environment of this lab for you to revise. If you want to create your own environment, the steps are similiar to revise an existing environment.

All the gazebo environments are stored as files with *.world* extension. For example, Figure 2 is *nrsl.world*. To open *.world* file, you need to run the following command in your terminal:

```
$ gazebo nrsl.world
```

Listing 1: Open gazebo environment

After opening the environment, you can add objects, like chairs, into it. **Please do not add the robots you want to use at this step since you can only add static objects at this step. I will tell you how to spawn robots in this environment in next subsection.**

After finishing your environment, you can go to *file* and then *save world as* to save this environment, for example, as *newnrsl.world*.

Next time, if you want to open your new environment, just run:

```
$ gazebo newnrsl.world
```

Listing 2: Open new gazebo environment

3.2 Spawn robots in the environment

This is a very important part of the simulator since we can use the same environment but we may need different robots.

We need *launch* file to connect robots and environment, which means spawning robots into the environment. Here is an example of how to write certain launch file:

```
1  <?xml version="1.0"?>
2  <launch>
3    <!-- We resume the logic in empty_world.launch, changing only the name of
       the world to be launched -->
4    <include file="$(find gazebo_ros)/launch/empty_world.launch">
5      <arg name="world_name" value="$(find cvg_sim_gazebo)/worlds/nrsl.world"
      />
6    </include>
7
8    <!-- Spawn simulated quadrotor uav -->
9    <include file="$(find cvg_sim_gazebo)/launch/spawn_quadrotor.launch" >
10     <arg name="model" value="$(find cvg_sim_gazebo)/urdf/quadrotor_sensors.
      urdf.xacro"/>
11   </include>
12 </launch>
```

Listing 3: Launch file to spawn a robot into the environment

It is obvious that this launch has two parts: *launch the environment* and *launch the robot*. In Line 5, you can see that the environment file *nrsl.world* is included. In Line 10, the file *quadrotor_sensors.urdf.xacro* is the robot model file. See, it is quite simple.

We usually download robot model file from website since it is really difficult to create a robot with multiple sensors yourself. After downloading the model file, you can spawn the robot into the environment using a launch file.

However, what if we want to spawn same multiple robots into one environment? Can I just copy Line 8 - Line 11? No, you cannot!

To spawn same robots, we need to use different namespace to distinguish them. Moreover, please be careful that the initial position of two different robots cannot be the same.

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>


  <!-- start the world -->
  <node name="gazebo" pkg="gazebo_ros" type="gazebo"
    args="$(find iRobot_create)/worlds/nrsl.world" respawn="false" output="
   screen" />

  <group ns="robot1">
    <include file="$(find iRobot_create)/launch/spawn_robot.launch">
        <arg name="robot_name" value="iRobot_create1" />
        <arg name="x" value="1" />
        <arg name="y" value="1" />
        <arg name="z" value="1.0" />
        <arg name="roll" value="0"/>
        <arg name="pitch" value="0"/>
        <arg name="yaw" value="0.0" />
        <arg name="sdf_robot_file" value="$(find iRobot_create)/models/
   create/model-1_4.sdf" />
    </include>
  </group>

  <group ns="robot2">
    <include file="$(find iRobot_create)/launch/spawn_robot.launch">
        <arg name="robot_name" value="iRobot_create2" />
        <arg name="x" value="1.0" />
        <arg name="y" value="-1" />
        <arg name="z" value="1.0" />
        <arg name="roll" value="0"/>
        <arg name="pitch" value="0"/>
        <arg name="yaw" value="0.0" />
        <arg name="sdf_robot_file" value="$(find iRobot_create)/models/
   create/model-1_4.sdf" />
    </include>
  </group>

  <group ns="robot3">
    <include file="$(find iRobot_create)/launch/spawn_robot.launch">
        <arg name="robot_name" value="iRobot_create3" />
        <arg name="x" value="-1.0" />
        <arg name="y" value="-1" />
        <arg name="z" value="1.0" />
        <arg name="roll" value="0"/>
        <arg name="pitch" value="0"/>
        <arg name="yaw" value="0.0" />
        <arg name="sdf_robot_file" value="$(find iRobot_create)/models/
   create/model-1_4.sdf" />
    </include>
  </group>

  <group ns="robot4">
    <include file="$(find iRobot_create)/launch/spawn_robot.launch">
        <arg name="robot_name" value="iRobot_create4" />
        <arg name="x" value="-1.0" />
        <arg name="y" value="1" />
```

```
        <arg name="z" value="1.0" />
        <arg name="roll" value="0"/>
        <arg name="pitch" value="0"/>
        <arg name="yaw" value="0.0" />
        <arg name="sdf_robot_file" value="$(find iRobot_create)/models/
    create/model-1_4.sdf" />
     </include>
  </group>
</launch>
```

Listing 4: Launch file to spawn robots into the environment

Here, you can see four different robots will be launched at four different positions.

3.3 Get the pose of objects in the simulator

To control a robot, the most fundamental requirements are knowing the pose of the robot and controlling the motion of the robot. Gazebo will publish a service called *get_model_state* to publish the state of all the objects in the environment and the data type of this service is $GetModelState$. This data type is quite similiar to Vicon data type $TransformStamped$, it consists of 7 elements: x, y, z coordinate and quaternion. What we need to do is to subscribe this service and we get the data.

If you are not familiar with *ROS service*, you can refer to the following code:

```
  subPose = rospy.ServiceProxy('/gazebo/get_model_state', GetModelState)
  robot1_Pose=subPose("iRobot_create1", "link")
```

Listing 5: Subscribe get_model_state service

Here, we get the pose of iRobot_create1 and $robot1\_Pose$ includes 7 values of the pose of the robot.

To get the pose of an object in gazebo, we need the name of the object and the name of a link of that object. In this situation, the object is called $iRobot\_create1$ and its link is called *link*.

So, how can we know the exact names of the object and its link?

After opening the gazebo environment, you need to open a terminal and type in:

```
$ rosservice call /gazebo/get_world_properties "{}"
```

Listing 6: Get the name of the object

You can get the name of all the objects in your environment and you can find the name of your robot. To get the name of a link of your robot, you can

type in this in your terminal, just replace *your_robot_name* with the real name of your robot.

```
$ rosservice call /gazebo/get_model_properties your_robot_name
```

Listing 7: Get the name of a link

You will see a lot information about your robot. Focus on the *body_names* and choose one to be your link name.

Great! Now, we can get the pose of the robots and next step is to use the topic *cmd_vel* to control your robot. The data type of *cmd_vel* is $Twist$, it has 6 elements: x, y, z velocity and x, y, z Euler angular velocity.