01 Weekly report 2

September 23, 2021

Information: *The weekly report supposed to be finished in week 4*

Written by: *Zihao Xu*

Last update date: *September.23.2021*

1 Overview

First of all, I need to apologize for not having much progress after almost four weeks. I was busy working on the registration affairs in the first two weeks, which was mentioned in the meeting with Jianwen. After that, I spend roughly one-week time reviewing the ROS operations and finding some tutorials of gazebo. Sorry for the low productivity in the past few weeks and I would try to compensate for the missing time in the upcoming weeks.

The first part of this report is the notes I made when I was reviewing the ROS operations. It's a necessary quick reference for me given that it has been a long time since the last time I worked with ROS. I separate it from this report.

The second part is the notes I took to modify the **hunter_gazebo** package. I was trying to make it clear the structure of the projects, which is essential for me to add or delete some functions in the simulation.

The third part is some notes for future integration. Some files have been edited but not yet pushed to git repo, because I'm not sure if that's necessary given that I'm only making some small changes and whether to keep the old codes.

The last part is an summary for past work and a plan of future work.

2 Package Analysis: hunter_gazebo

2.1 package.xml

In the **package.xml** file, I noticed the author claimed the dependencies in a quite complex way.

[ ]:
```
<build_depend>geometry_msgs</build_depend>
<build_depend>rospy</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>std_msgs</build_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>sensor_msgs</build_export_depend>
```

```
<build_export_depend>std_msgs</build_export_depend>
<exec_depend>geometry_msgs</exec_depend>
<exec_depend>rospy</exec_depend>
<exec_depend>sensor_msgs</exec_depend>
<exec_depend>std_msgs</exec_depend>
```

According to the ROS Wiki, if a dependency is a build, export, and execution dependency, it can be claimed by *<depend>package</depend>*. Therefore, I simplified the claims.

[ ]:
```
<depend>geometry_msgs</depend>
<depend>rospy</depend>
<depend>sensor_msgs</depend>
<depend>std_msgs</depend>
```

The *catkin_make* command works well.

2.2 run_simulation.launch

2.2.1 Arguments for this launch file

- First declare several arguments with default values. These values can be overridden by command-line argument or via *include* passing.
- *<arg>* tag allows to create more re-usable and configurable launch files by specifying values that are passed via the command-line, passing in via an *<include>*, or declared for higher-level files.

[ ]:
```
<arg name="world_name" default="basic"/>
<arg name="enable_logging" default="false"/>
<arg name="enable_ground_truth" default="true"/>
<arg name="enable_gui" default="true"/>
```

- This is an argument with a constant value which cannot be overridden.

[ ]:
```
<arg name="paused" value="false"/>
```

2.2.2 Environment variables for nodes

- Then set the environment variables on nodes that are to be launched in this file.
- Here declares the path of the **gazebo model** and the **gazebo resource**. Their absolute path is *src/drone_boat_simulation_rotors_simulator/rotors_gazebo/models*. They are models embedded in the **RotorS** package.

[ ]:
```
<env name="GAZEBO_MODEL_PATH" value="${GAZEBO_MODEL_PATH}:$(find rotors_gazebo)/
↪models"/>
<env name="GAZEBO_RESOURCE_PATH" value="${GAZEBO_RESOURCE_PATH}:$(find␣
↪rotors_gazebo)/models"/>
```

2.2.3 Gazebo environment: empty world

- The next step is to import the launch file built in gazebo into the current file to create an empty world.
- The absolute path of this launch file is */opt/ros/noetic/share/gazebo_ros/launch/empty_world.launch*. The arguments that can be passed to it can be viewed in the launch file.
- Here only three arguments are passed to it:
  - *world_name*
  - *paused*
  - *gui*

[ ]:
```
<include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find rotors_gazebo)/worlds/$(arg␣
↪world_name).world"/>
    <arg name="paused" value="$(arg paused)"/>
    <arg name="gui" value="$(arg enable_gui)"/>
</include>
```

2.2.4 Quadcopter Nodes

- In the original launch file, the following are two groups of launch commands which starts two UAVs in the created empty world. Given that we're going to use only a **Quadcopter** for the project, the launch commands for the Hexacopter is deleted in the new launch file.
- Here are the commands in the *group* tag whose namespace is *hunter*. It sets a group of nodes which simulates the Quadcopter.

Quadcopter Model Node

- The first part is to import the launch file embedded in the **RotorS** package which creates the UAV in gazebo. The absolute path of this launch file is *src/drone_boat_simulation_rotors_simulator/rotors_gazebo/launch/spawn_mav.launch*. It also provides a lot of editable arguments.
- The model for this Quadcopter is an **AscTec Pelican**.
- The created model is a running node with several parameters (positions, rotations, etc.) continuously updated.

[ ]:
```
    <include file="$(find rotors_gazebo)/launch/spawn_mav.launch">
      <arg name="mav_name" value="pelican" />
      <arg name="namespace" value="hunter" />
      <arg name="model" value="$(find rotors_description)/urdf/
↪mav_with_vi_sensor.gazebo" />
      <arg name="enable_logging" value="$(arg enable_logging)" />
      <arg name="enable_ground_truth" value="$(arg enable_ground_truth)" />
      <arg name="log_file" value="pelican"/>
      <arg name="x" value="4.0"/>
      <arg name="y" value="0"/>
      <arg name="z" value="1"/>
    </include>
```

Position Controller Node

- The second part is a position controller node embedded in the **RotorS** package.

```
[ ]:   <node name="lee_position_controller_node" pkg="rotors_control"␣
↪type="lee_position_controller_node" output="screen">
     <rosparam command="load" file="$(find rotors_gazebo)/resource/
↪lee_controller_pelican.yaml" />
     <rosparam command="load" file="$(find rotors_gazebo)/resource/pelican.
↪yaml" />
     <remap from="odometry" to="ground_truth/odometry" />
   </node>
```

State Publisher Nodes

- The third part consists of two state publisher nodes which publish robot states and joint states.

```
[ ]:   <node name="robot_state_publisher" pkg="robot_state_publisher"␣
↪type="robot_state_publisher" />
   <node name="joint_state_publisher" pkg="joint_state_publisher"␣
↪type="joint_state_publisher" />
```

Path Planning Node

- The last part is a custom node responsible for path planning. It communicates with the position controller node and state publisher node.
- The script for path planning has been edited to draw a heart-like shape, mentioned in last report.

```
[ ]:   <node name="waypoint_publisher" pkg="hunter_gazebo" type="follow_waypoints.
↪py" output="screen" respawn="true">
     <param name="~dt" value="0.1"/>
     <param name="~velocity" value="5"/>
     <param name="~inertial_frame" value="world"/>
     <param name="~base_frame" value="/hunter/base_frame"/>
     <remap from="odometry" to="ground_truth/odometry"/>
   </node>
```

2.2.5 nodelet

- Currently I cannot understand what's the function of this node while it's supposed to provide a way to run multiple algorithms in the same process with zero copy transport between algorithms.

```
[ ]: <node pkg="nodelet" type="nodelet" name="stereo_proc_manager" args="manager"␣
↪output="screen"/>
```

3 Notes for future integration

3.1 Git Repository

Some latest commits are using the submodule tool in Github. That is great but I suggest that some changes should be made to the instructions in README.md about cloning the project. When a project with a submodule in it, by default the directories that contain submodules would be got but none of the files within them yet.

One way is to run the following two commands after cloning the project as usual:

- *git submodule init*
- *git submodule update*

Another way is to pass *--recurse-submodules* to the *git clone* command. That is:

- *git clone --recurse-submodules <git repo address>*

However, I cannot succesfully build the work space after downloading all the submodules while I do not know what's going wrong in the building process. I only downloaded the *mav_comm* submodule which is necessary for me to simulate the motions of the UAV.

3.2 Editted Files

3.2.1 package.xml

Directory: *catkin_ws/src/drone_boat_simulation/hunter_gazebo/package.xml*

Description: Simplify the dependency claims.

3.2.2 run_simulation.launch

Directory: *catkin_ws/src/drone_boat_simulation/hunter_gazebo/launch/run_simulation.launch*

Description: Comment out the launch commands for *drone_tracker*, which is currently unused. Comment out the nodes group for the hexacopter *firefly*. Add detailed comments for each part.

3.2.3 follow_waypoints.py

Directory: *catkin_ws/src/drone_boat_simulation/hunter_gazebo/scripts/follow_waypoints.py*

Description: Add detailed comments. Comment out unused functions and imports. Add one new heart-shape trajectory for the drone.

4 Summary and future work

4.1 Summary

In the past few weeks, I

- reviewed the ROS operations and made some notes about ROS for quick reference
- analyzed the file system of this project carefully and optimized the dependency claims
- made it clear which nodes are launched and functioning
- optimized the launch file according to test needs

- got a rough understanding of how to use gazebo

4.2 Future work

In the next week, I plan to

- check the position controller node and find ways to get control of primitive motions
- add obstacles in Gazebo environment for future tests
- explore existing obstacle avoidance algorithms