How to use *tud_firefly_landing_sim*

Tutorial

Ding Zhang

TECHNISCHE
UNIVERSITÄT
DARMSTADT

REGELUNGSMETHODEN
UND ROBOTIK rmr

How to use *tud_firefly_landing_sim*
Tutorial

Eingereicht von Ding Zhang
Tag der Einreichung: 15. August 2016

Gutachter: Prof. Dr.-Ing. Jürgen Adamy
Betreuer: Raúl Acuña Godoy

Technische Universität Darmstadt
Fachbereich ETiT
Institut für Automatisierungstechnik
Prof. Dr.-Ing. Jürgen Adamy

1 ROS and *catkin* workspace

1.1 ROS

ROS (**R**obot **O**perating **S**ystem) is a open source software environment which provides libraries and tools to help software developers create robot applications. Those applications are usually created modularly as "packages". By utilization of a package a corresponding process will be activated, this process is called a "node". The communitation between different nodes are realized via topics and services as revealed in figure 1.1. The concret tutorial for beginners to install ROS, to understand the basical elements in ROS such as a package, a node, a topic as well as to implement them using C++ and Python is available in [1].

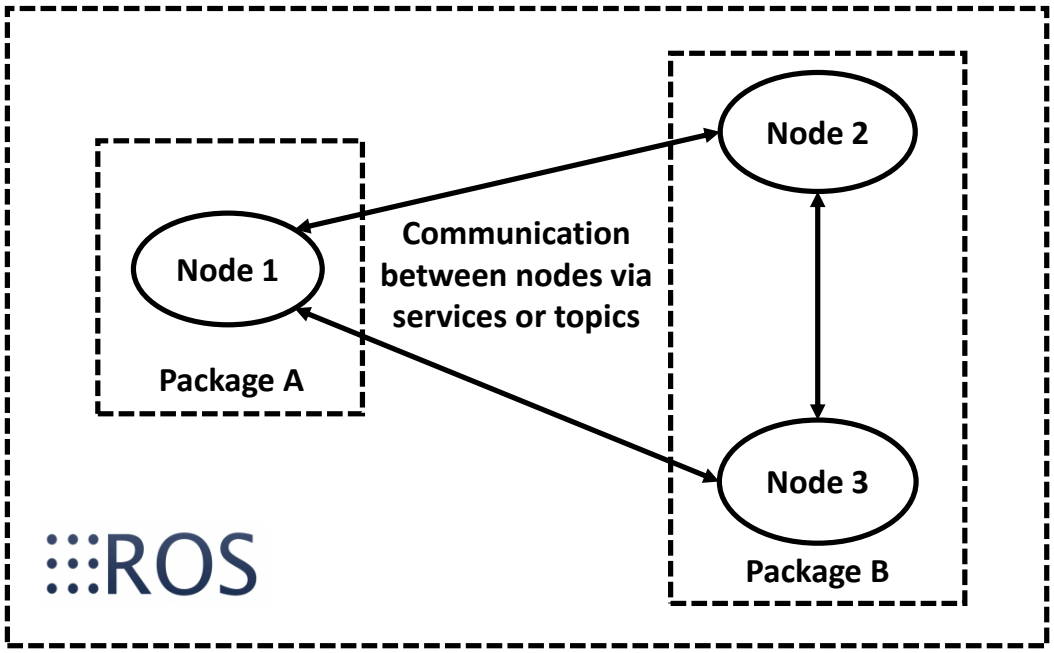


Figure 1.1: The basical structure of the applications in ROS environment.

1.2 *catkin* workspace

There are several ways of acquiring applications from online software repositories to ROS. E.g. you can install them by using the command "sudo apt-get install XXX(application name)", in this way applications are downloaded and installed into your ROS directly without available source codes. However if you wish to write or modify the source code of an application directly, a *catkin* workspace is required. *catkin* is a low-level compilation tool which is included by default when ROS is installed. As figure 1.4 shows, a *catkin* workspace is like a expansion of your ROS environment. You can create and modify your own ROS packages within the workspace and compile them into your ROS. The concret tutorial for creating and building a *catkin* workspace is available in [2] and [3]. A general overview of the user defined ROS packages in a *catkin* workspace is revealed in figure 1.3. All packages are to be located in the subfolder *src/*(source code) of the workspace folder. Each ROS package has its own *include/*

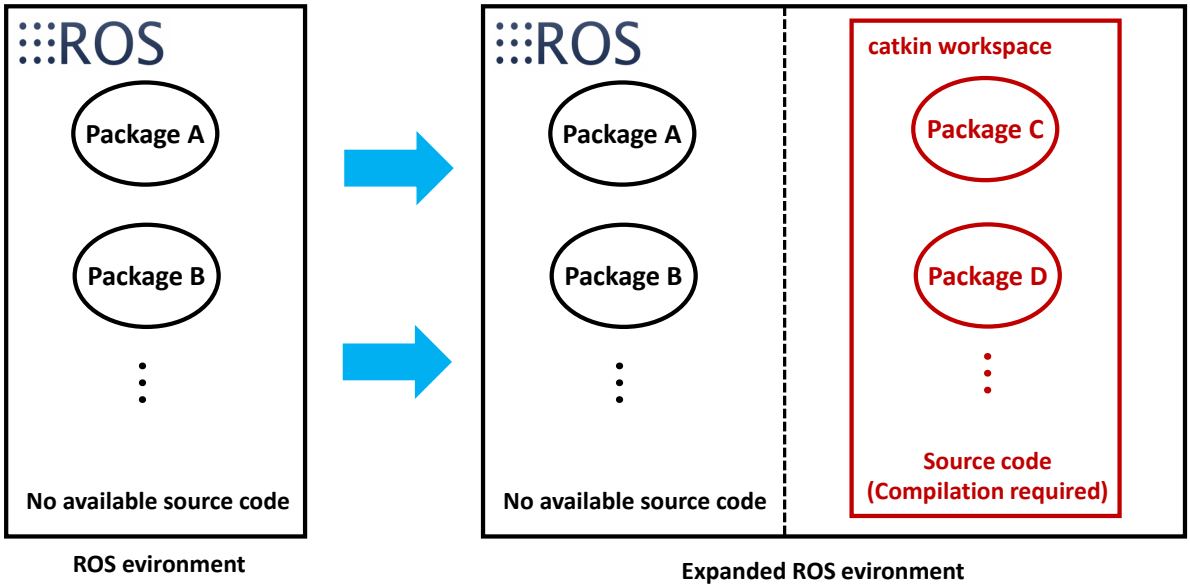


Figure 1.2: *catkin* workspace, the expansion of ROS.

folder and *src/* folder that contains the header files and the source codes defined by users. The build system of *catkin* is based on *CMAKE*, therefore a ROS package in a *catkin*-workspace has a corresponding *CMakeList.txt* file according to *CMAKE* tutorial [4]. *catkin* simplized the *CMAKE* and has its own grammar, which is available in [5]. According to the rules of *catkin*, all the other *catkin* ROS packages depended by this package has to be listed in a XML script called *package.xml*. After compilation, two new folders namely *devel/* and *build/* will be generated automatically. The former contains the compiled .so library files. And the latter contains the compiled executables. In ROS there are many ways to launch a ROS executable, the most prefered one is using .launch scripts, which are also writed in XML. For example, one could launch a the *launch_file_1.launch* in figure 1.3 with following command in a terminal:

$ *roslaunch package_1 launch_file_1.launch*

The tutorial about grammars in .launch scripts are available in [6].

1.3 Metapackage

When creating a large application with many diferent utilities, developers usually prefer to divde it into several small modular pieces, so that the design and maintain could be simpler. One way of realizing that in ROS is metapackage. Metapackage is a specific type of ROS package, which references one or more related packages that are loosely grouped together. The details about metapackage is available in [7]. The application *tud_firefly_landing_sim* that introduced in this tutorial is a ROS metapackage that contains 7 packages as figure revealed.

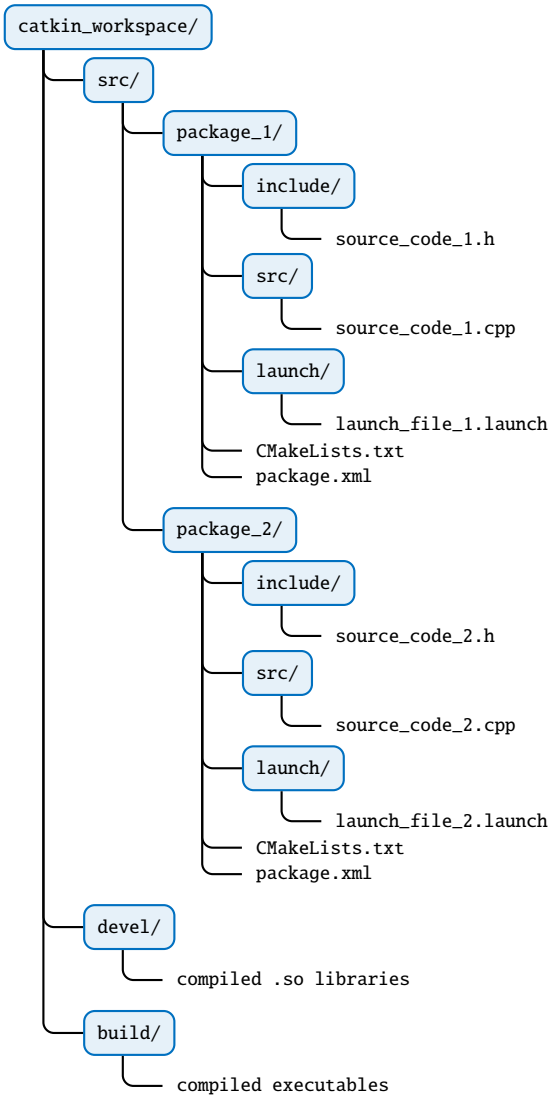


Figure 1.3: Structure of ROS packages in a *catkin* workspace

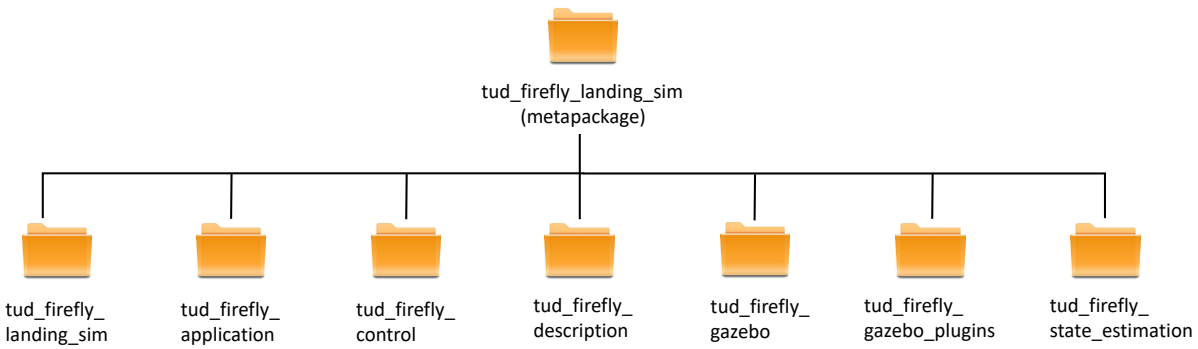


Figure 1.4: *tud_firefly_landing_sim*, the ROS metapackage that contains 7 individual ROS packages.

2 Necessary packages depended by *tud_firefly_landing_sim*

tud_firefly_landing_sim is designed based on several important ROS applications. So before introducing the functionalities of *tud_firefly_landing_sim*, those packages will firstly be interpreted.

2.1 *gazebo_ros*

Gazebo is a powerful simulation tool that provides high-perfomance physics engines to emulate the robot dynamics with advanced 3-D realistic rendering of environments, including high-quality lighting, shadows and textures as figure 2.1 shows. By installing ROS environment, a specific version of Gazebo called

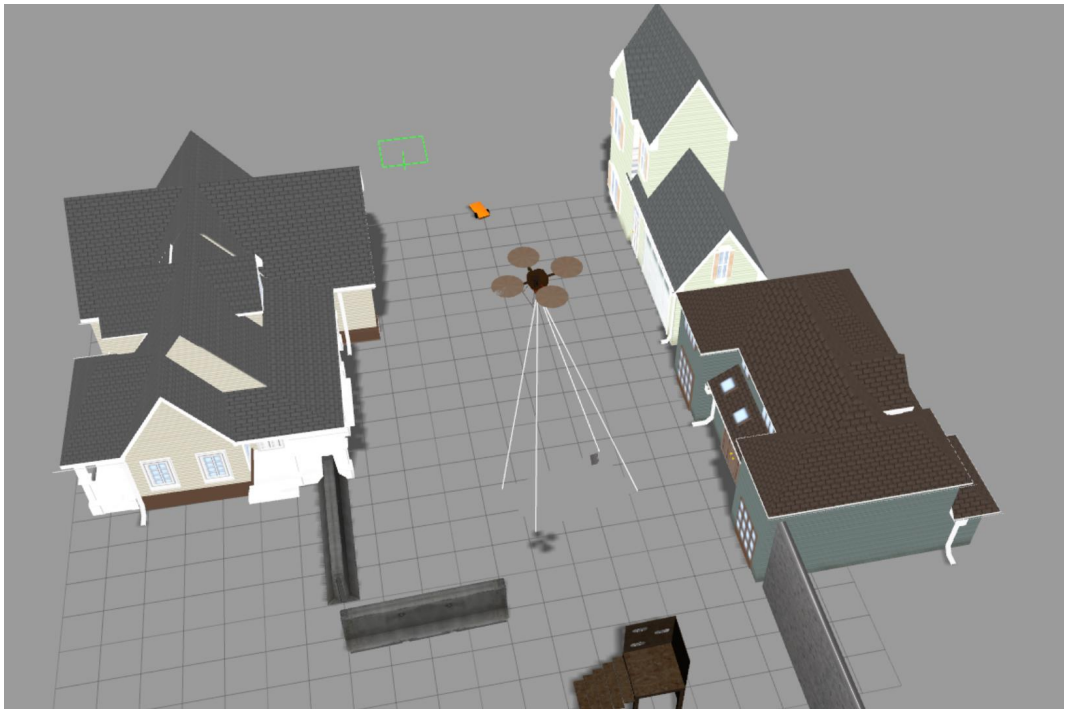


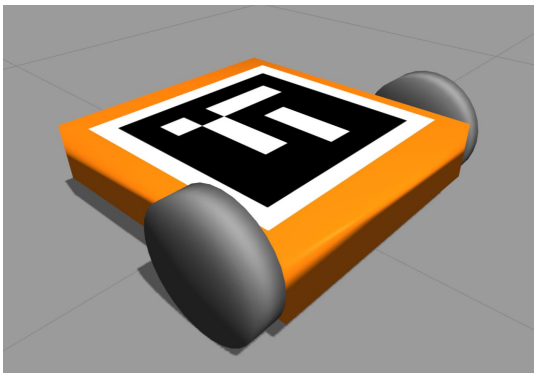
Figure 2.1: Gazebo simulation tool

gazebo_ros, which is compatible with ROS, is included by default. An overview of *gazebo_ros* can be found in [8]. In *gazebo_ros*, the robot modells can be described by a XML(E**x**tensible **M**arkup **L**anguage)-format language called URDF (**U**niversal **R**obot **D**escription **F**ormat). In *tud_firefly_landing_sim*. Those URDF scripts are usually defined as *.xacro* or *.gazebo* files. Using a ROS *.launch* script a virtual world including the robot described by the URDF script can be activated in the course of *gazebo_ros*. An example can be found in the following path:

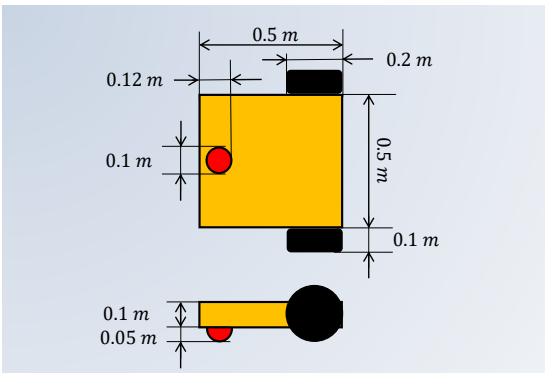
tud_firefly_landing_sim/tud_firefly_description/urdf/diff_wheeled_UGV.xacro

There, a differential-wheeled-UGV (**U**nmanned **G**raund **V**ehicle) defined in figure 2.2 is described. Now generate a .launch script, fill it with the following XML-codes:

```
<launch>
  <!-- these are the arguments you can pass this launch file, for example paused:=
      true -->
  <arg name="paused" default="false"/>
  <arg name="use_sim_time" default="true"/>
  <arg name="gui" default="true"/>
  <arg name="headless" default="false"/>
  <arg name="debug" default="false"/>

  <!-- We resume the logic in empty_world.launch -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="debug" value="$(arg debug)" />
    <arg name="gui" value="$(arg gui)" />
    <arg name="paused" value="$(arg paused)"/>
    <arg name="use_sim_time" value="$(arg use_sim_time)"/>
    <arg name="headless" value="$(arg headless)"/>
  </include>

  <!-- urdf xml robot description loaded on the Parameter Server-->
  <param name="robot_description" command="$(find xacro)/xacro.py '$(find
      tud_firefly_description)/urdf/diff_wheeled_UGV.xacro'" />

  <!-- Run a python script to the send a service call to gazebo_ros to spawn a URDF
       robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
      output="screen"
   args="-urdf -model diff_wheeled_robot -param robot_description"/>

  <node name="joint_state_publisher" pkg="joint_state_publisher" type="
      joint_state_publisher" >
    <remap from="odom" to="world"/>
  </node>

  <!-- start robot state publisher -->
  <node pkg="robot_state_publisher" type="robot_state_publisher" name="
      robot_state_publisher" output="screen" >
    <param name="publish_frequency" type="double" value="50.0" />
    <remap from="odom" to="world"/>
  </node>
</launch>
```



(a) The model used in Gazebo



(b) The bottom and right view of the model

Figure 2.2: The differential wheeled UGV model

Then execute that .launch file, then a visual world with a differential-wheeled-UGV in figure 2.2 will be generated. If you wish to perform control of the UGV, then you need to attach a gazebo plugin to the actuator parts of the robot model, the concret tutorial about attaching gazebo_plugins to URDF robot model is available in [9].

2.2 *rotors_simulator*

rotors_simulator is the most important application depended by *tud_firefly_landing_sim*, which is a MAV simulator based on *gazebo_ros*. It provides some multirotor models such as the AscTec Hummingbird, the AscTec Pelican, or the AscTec Firefly, but the simulator is not limited for the use with these multicopters. There are simulated sensors coming with the simulator such as an IMU, a generic odometry sensor, which can be mounted on the multirotor. [10] RotorS is also a ROS metapackage, the partial structure of RotorS is available in figure 2.3. As you can see, this package is also a metapackage. In figure 2.3 a few key subpackages are listed.

1. **rotors_description:** This package contains a lot of URDF scripts that define varies of UAV 3D models, the one that used by *tud_firefly_landing_sim* is *firefly.xacro* which defines the model of the hexacopter *AscTec* firefly. Because the UAV model is much complexer than simple robot models like a differential-wheeled-UGV, the UAV model has to be imported from a specific mesh file that generated via advanced modelling tools like *Blender*;

2. **rotors_gazebo_plugins:** As you can see in *firefly.xacro*, some gazebo plugins are attached to the actuators and some of the UAV parts to simulate the electronic actuator controller and the sensors. Those plugins are linked with .so libraries that compiled from the source codes that edited in this package. The concret information of those libraries can be found in the *CMakeList.txt* of this package;

3. **rotors_control:** This package contains the UAV control algorithms in the folder *library/*, it also contains the source codes for generating executables in the folder *nodes/*. The control algorithms subscribe on one hand the topics published by sensor plugins, and publish on the other hand the topics of actuator control signals that subscribed by the actuator plugins;

4. **rotors_gazebo:** This package contains firstly the UAV physical parameters that saved as .yaml scripts. It also contains the source codes of some application source codes like *hovering_example.cpp*. This application will publish a position setpoint message (1, 0, 1) which stands for the coordinates of the world coordinate system of the virtual world, this message will be subscribed by *lee_position_controller_node.cpp* as the setpoint for the position control algorithm. Input the following command via a terminal:

   $ *roslaunch rotors_gazebo mav_hovering_example.launch*

   Then a gazebo simulation of the UAV position control to a fest point (1, 0, 1) will be performed as figure 2.4 revealed;

A summary of listed *rotors_simulator* packages is described in the figure 2.5.

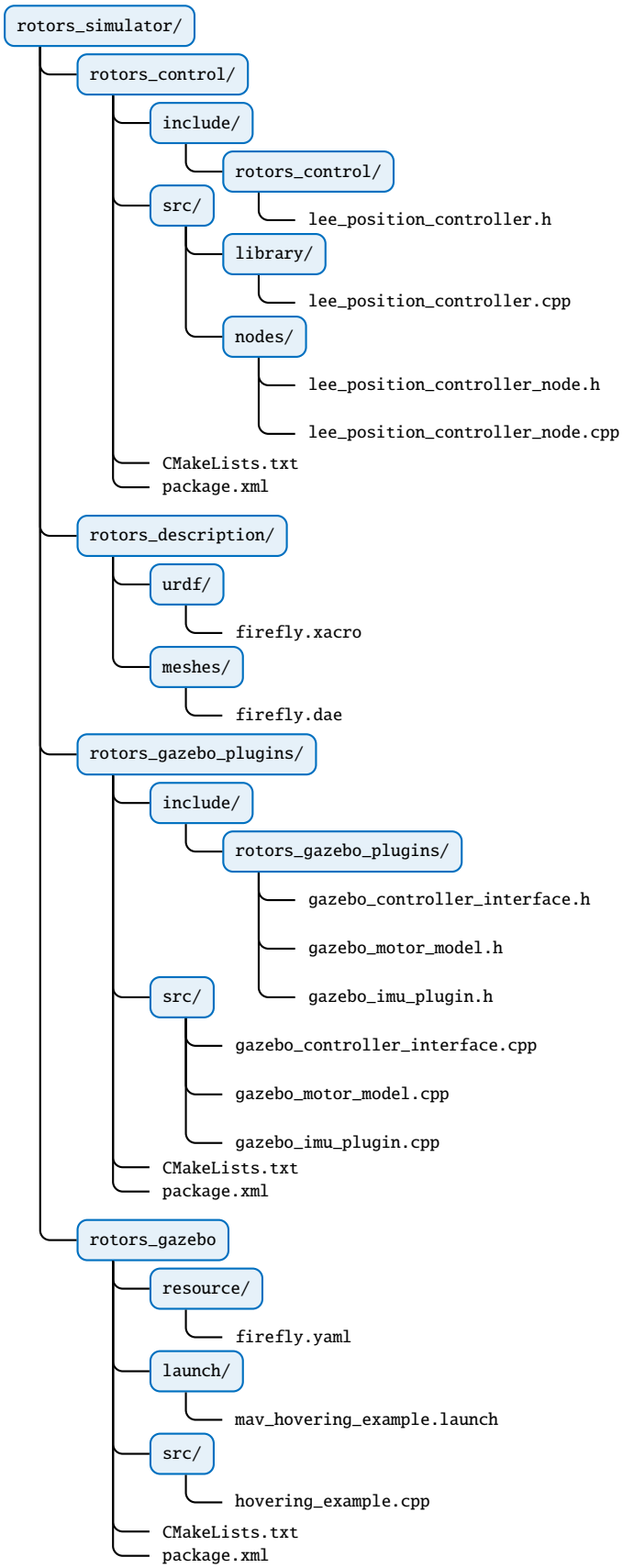


Figure 2.3: Structure of ROS packages in a *catkin* workspace

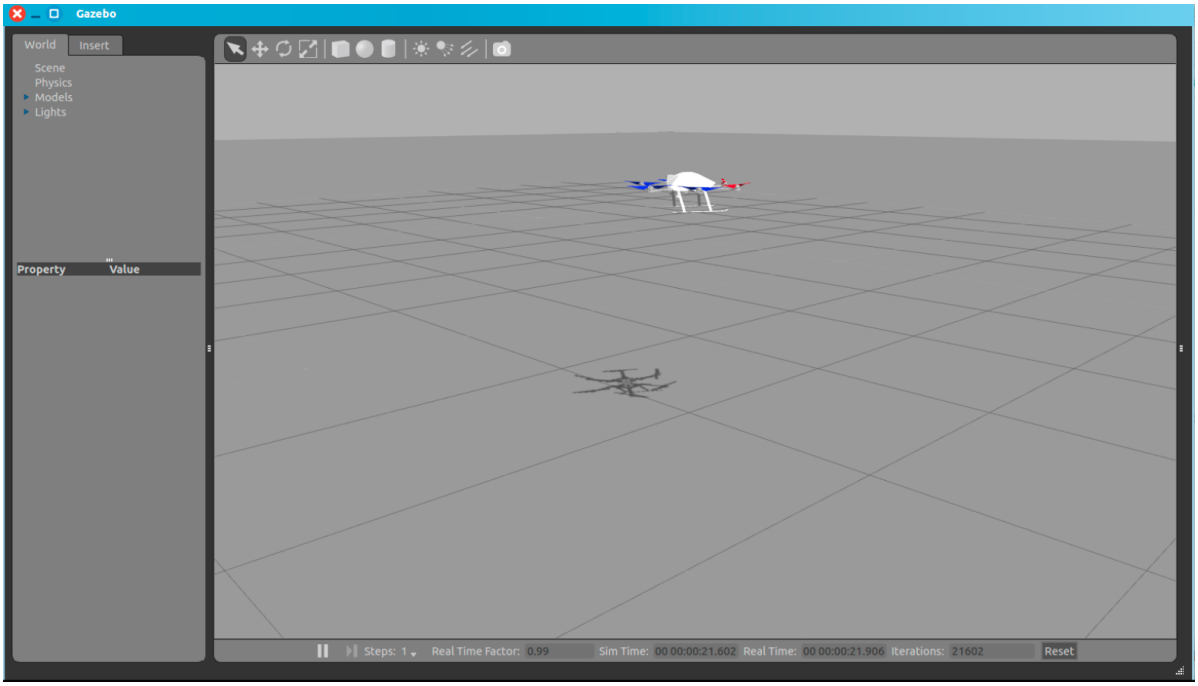


Figure 2.4: Hovering example of the UAV control

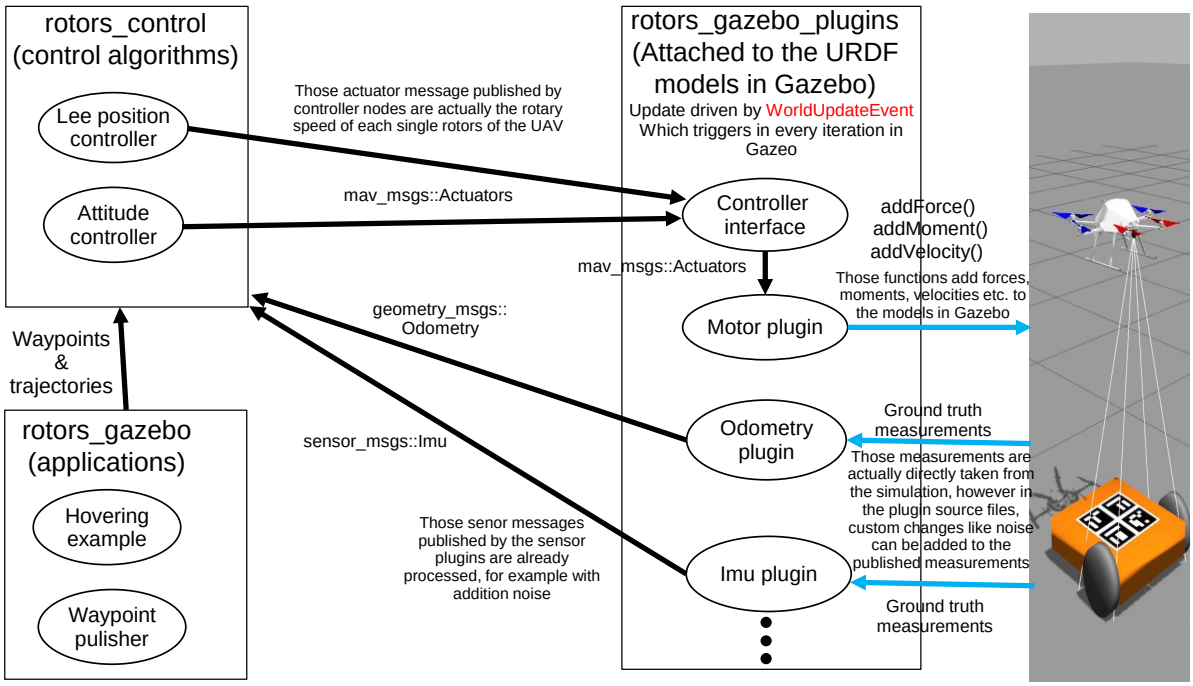


Figure 2.5: Structure of RotorS simulator

2.3 *ar_sys*

2.3.1 Vision based robot navigation via fiducial marker

ar_sys is a 3D pose estimation ROS package using fiduccial markers. As revealed in figure 2.7, the package provide useful executables to estimate the relative pose between camera frame coordinate and the marker based coordinate. The marker used in the simulation are called Aruco marker. The image of Aruco markers can be generated and downloaded from an online marker generator in [11]. The tutorial of *ar_sys* can be found in [12].

2.3.2 Create a fiducial marker .dae file via *Blender*

In *gazebo_ros*, not only the robot model but also the corresponding sensor system can be simulated including robot onboard cameras, which means the application *ar_sys* can also be integrated and implemented in the simulation. The Aruco marker images can be imported into the simulation as 3D model textures. The robot .dae format robot mesh file with custom textures can be designed via a software tool called *Blender*, as revealed in figure 2.6. The tutorials of *Blender* can be easily found on *youtube.com*

2.3.3 The vision based UGV detection in *gazebo_ros*

As revealed in figure 2.8, the test result shows that the *ar_sys* is compatible with *gazebo_ros*.

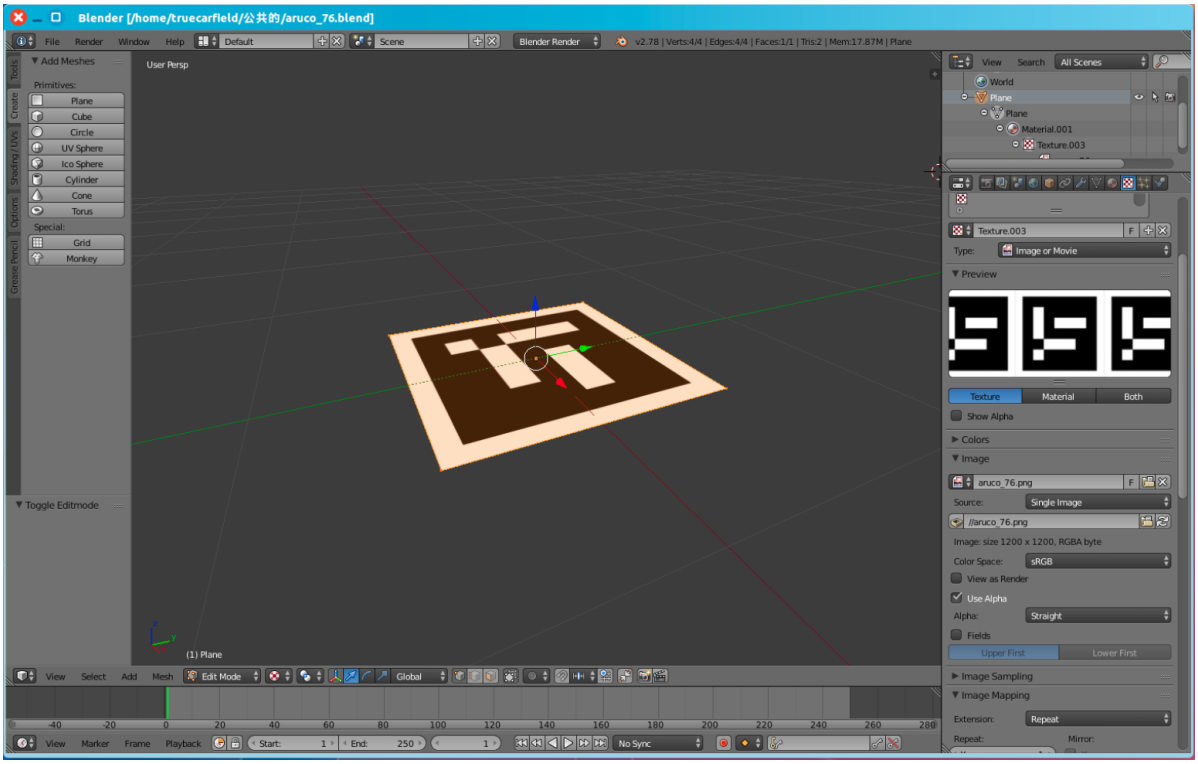
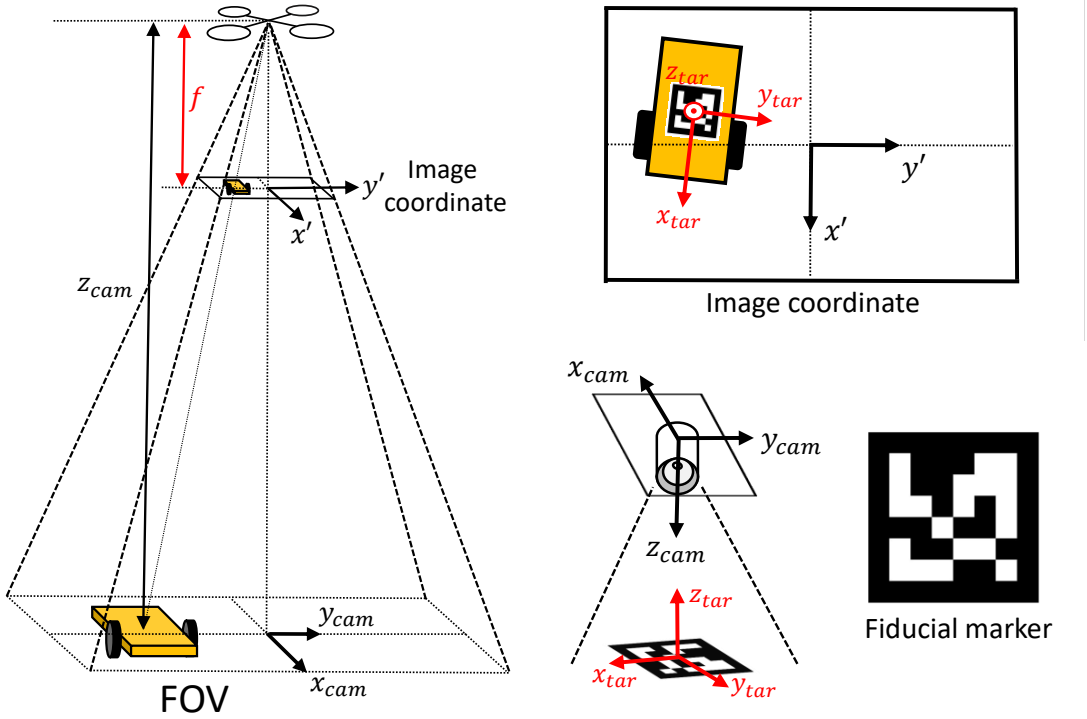
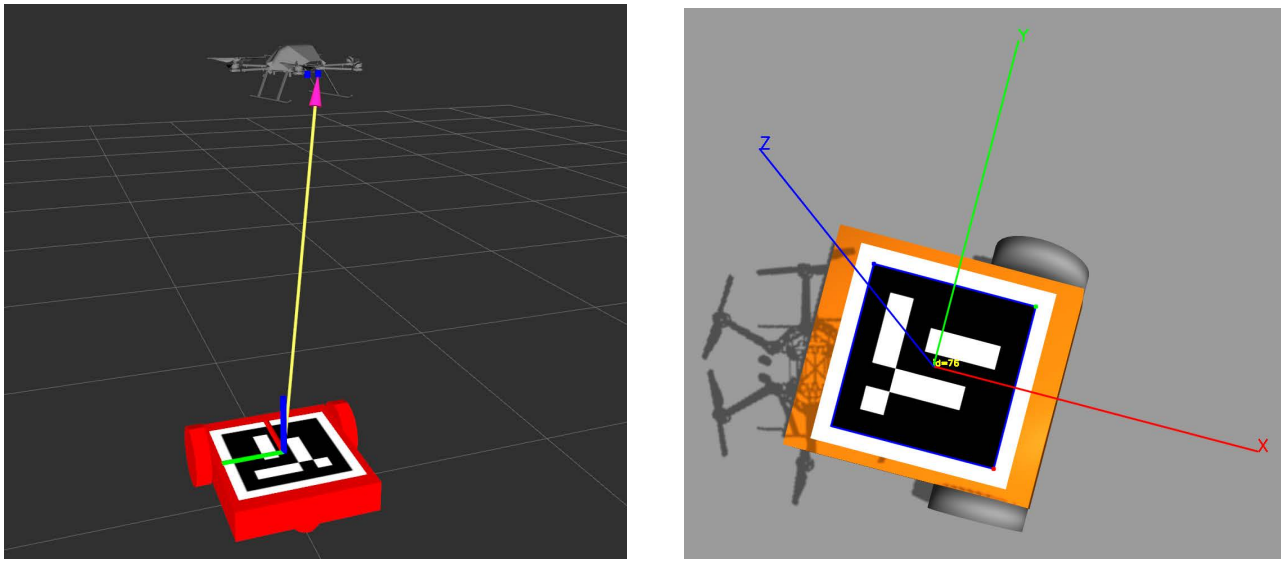


Figure 2.6: Create a fiducial marker .dae file via *Blender*



(a) Image projection in a camera frame (b) Image coordinate and target identification

Figure 2.7: The principle of vision-based UAV navigation



(a) Relative position vector (b) The marker pose

Figure 2.8: The relative pose between UAV camera- and UGV marker-coordinate identificated via *ar_sys*

2.4 *robot_localization*

robot_localization is a generic application for the estimation of robot states (position, attitude angles, angular rates etc.). The introduce of the package can be found in [13]. It is designed with C++. The main class of this application is *robot_localization::RosFilter* in the source file *ros_filter.cpp*, in the corresponding header file *ros_filter.h* you can find the concret explanation of the package. Here only the key data types and functions in class *robot_localization::RosFilter* will be introduced in order to explan the work flow the package *robot_localization*.

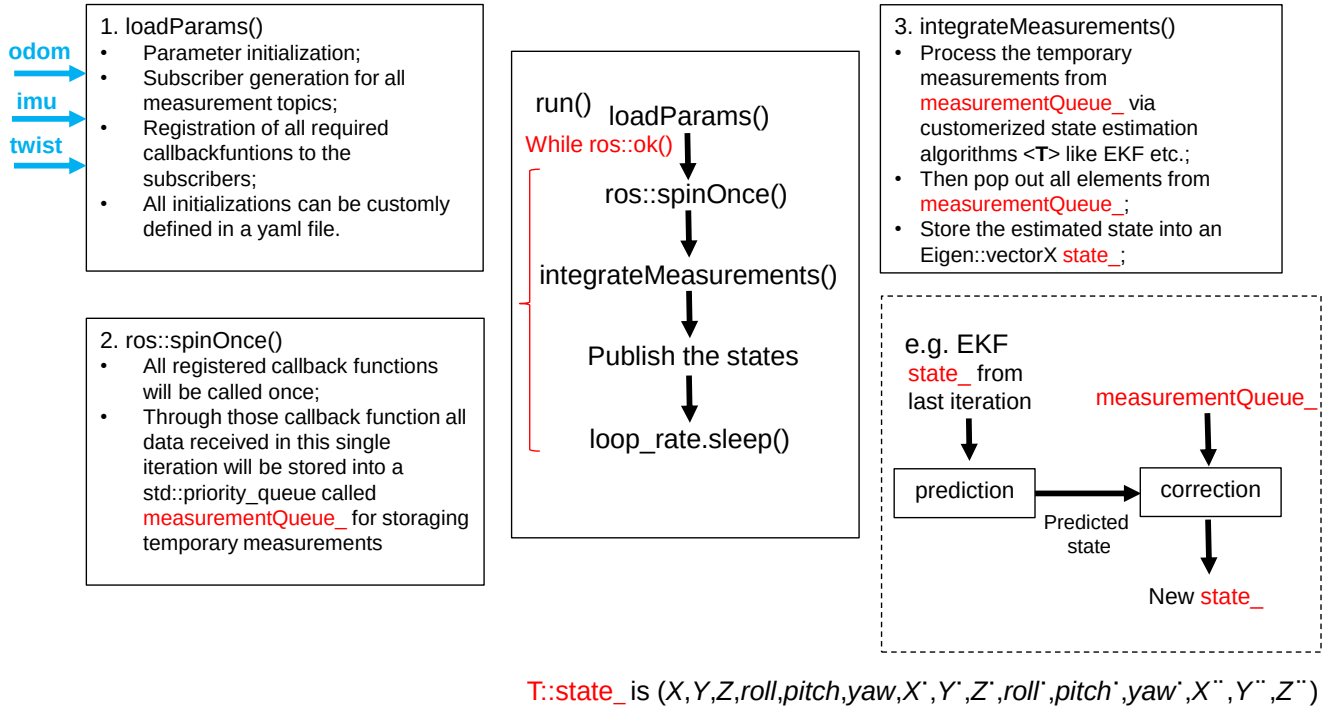


Figure 2.9: Key functions in *robot_localization*

Key data types:

1. ***state_***: This is a generic robot state vector with 15 dimensions, including the robot position in the world coordinate, attitude angles, angular rates and linear accelerations in the robot body coordinate;

2. ***measurmentQueue_:*** This is the C++ priorty queue that is responsable for the storage of temporary measurements in one iteration. This kind of queue will automatically group the current measurements by time when new data comes.

Key functions:

1. ***run():*** The main function that control the work flow, within it several important methods are executed as explaned below;

2. ***loadParams():*** Firstly after executing the member function *run()*, the method *loadParams()* is launched, it defines all the required parameters, generates the subscribers of the sensor measurement topics and the publishers of the estimated states;

3. ***integrateMeasurements():*** The template method where the key algorithm is used, there are two default methods, namely the EKF(**E**xtended **K**alman **F**ilter) and UKF(**U**scentted **K**alman **F**ilter) in

two source files *ekf.cpp* and *ukf.cpp*. Users can also define their own algorithm files and integrate them into the function *integrateMeasurements()*.

The working loop of each iteration is realized by ROS default *ros::spinOnce()* method, the sampling time can be defined with *ros::loop_rate* method. As a result, the estimated robot states are published in topic */odometry/filtered*.

3 tud_firefly_landing_sim

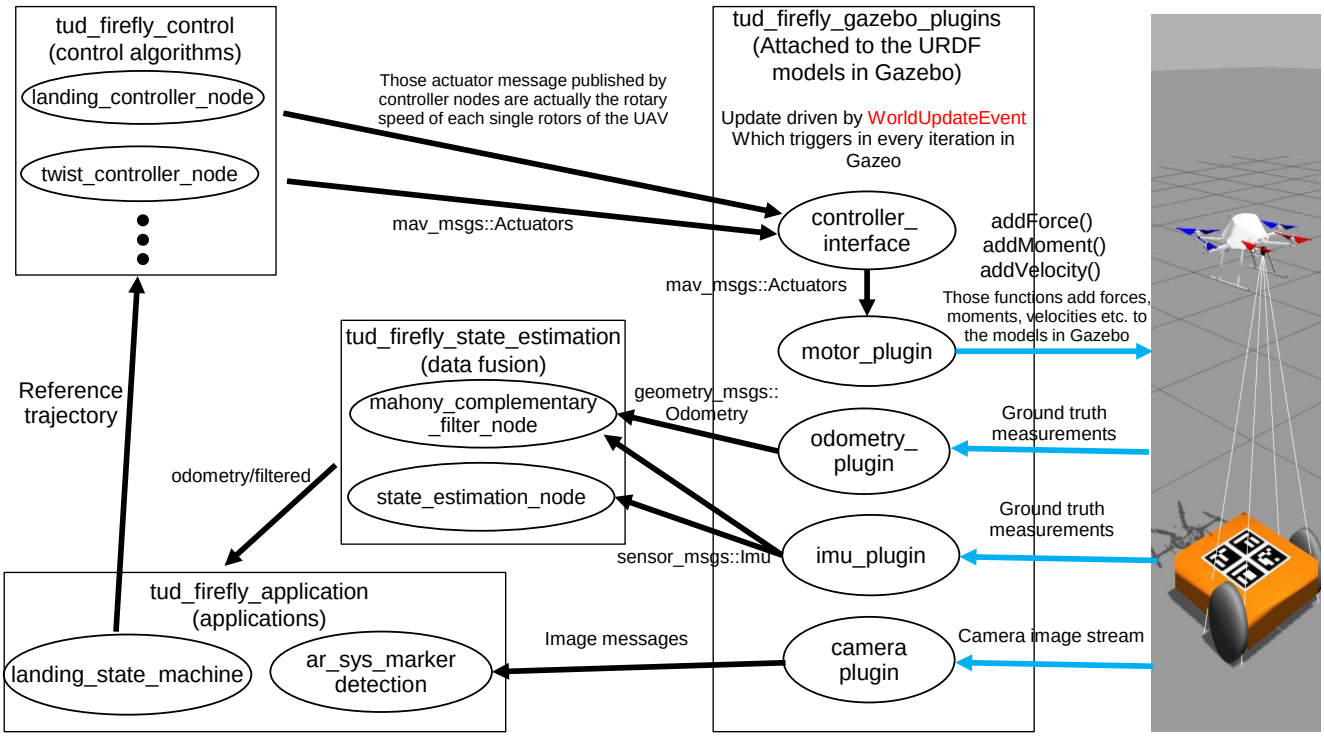


Figure 3.1: How packages within *tud_firefly_landing_sim* are working together.

The package *tud_firefly_landing_sim* is designed according to metapackage *rotors_simulator* to simulate the autonomous landing process of an UAV on top of a mobile UGV. This framework is modular, and can used to test more customised algorithms. It contains seven individual subpackages.

1. ***tud_firefly_landing_sim*:** This package plays the role of package list of a metapackage, in the *package.xml* file of this package all subpackages are listed in the run_depend list;

2. ***tud_firefly_description*:** This package contains the customerized marker mesh files and the URDF scripts of the AscTec firefly UAV as well as the UGV that to be landed;

3. ***tud_firefly_gazebo_plugins*:** This package contains source files of the gazebo plugins from the UAV model such as actuator controller, sensors like IMU and camera;

4. ***tud_firefly_state_estimation*:** This package contains the data fusion system of the UAV. Firstly an EKF algorithm developed based on the package *robot_localization*. Moreover, another algorithm called mahony-complementary-filter is developed in order to estimate the UAV attitude directly from IMU measurements, the final fused UAV states will be published to the topic */odometry/filtered*;

5. ***tud_firefly_control*:** This package contains the source code of UAV control algorithms such as attitude controller, twist controller, landing controller etc.;

6. ***tud_firefly_application*:** This package contains the source code of several subsystems that contribute to the UAV autonomous landing simulation, such as the marker detection system based on

ar_sys. And a landing state machine has been developed in order to define the landing process according to current measurements;

7. ***tud_firefly_gazebo*:** This package contains the launch files of the UAV autonomous landing simulation.

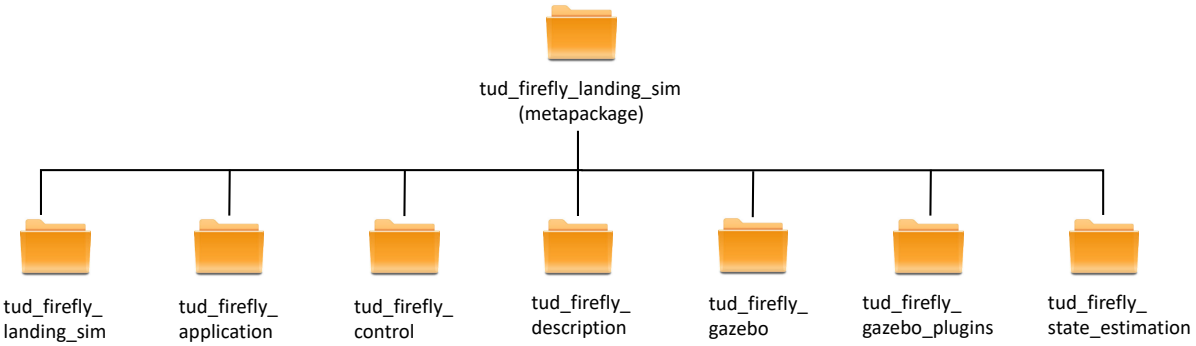


Figure 3.2: *tud_firefly_landing_sim*, the ROS metapackage that contains 7 individual ROS packages.

The working flow of *tud_firefly_landing_sim* is revealed in the figure 3.4. The .launch script of the UAV autonomous landing simulation is located in the following path:

tud_firefly_landing_sim/tud_firefly_gazebo/launch/firefly_landing_control.launch

with following XML codes:

```
1  <launch>
2
3    <!-- these are the arguments you can pass this launch file, for example paused:=
         true -->
4    <arg name="paused" default="false"/>
5    <arg name="use_sim_time" default="true"/>
6    <arg name="gui" default="true"/>
7    <arg name="headless" default="false"/>
8    <arg name="debug" default="false"/>
9
10   <!-- <env name="GAZEBO_MODEL_PATH" value="${GAZEBO_MODEL_PATH}:$(find
         tud_firefly_landing_sim)/models"/>
11   <env name="GAZEBO_RESOURCE_PATH" value="${GAZEBO_RESOURCE_PATH}:$(find
         tud_firefly_landing_sim)/models"/> -->
12
13   <!-- Generate the virtual world in gazebo_ros -->
14   <include file="$(find tud_firefly_gazebo)/launch/spawn_world.launch">
15     <arg name="debug" value="$(arg debug)" />
16     <arg name="gui" value="$(arg gui)" />
17     <arg name="paused" value="$(arg paused)"/>
18     <arg name="use_sim_time" value="$(arg use_sim_time)"/>
19     <arg name="headless" value="$(arg headless)"/>
20   </include>
21
22   <!-- Creating the differential wheeled UGV in Gazebo -->
23   <group ns="UGV">
24     <param name="robot_description" command="$(find xacro)/xacro.py '$(find
           tud_firefly_description)/urdf/diff_wheeled_UGV.xacro'" />
25     <node name="UGV_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
           output="screen"
26           args="-x  1.0
```

```
27                -y  -3.0
28                -z  0.0
29                -Y  1.57
30                -urdf
31                -model diff_wheeled_UGV
32                -param robot_description" />
33
34    <!-- Publish the UGV robot states to topic tf -->
35    <node name="UGV_robot_state_publisher" pkg="robot_state_publisher" type="
          robot_state_publisher" />
36    <node name="UGV_joint_state_publisher" pkg="joint_state_publisher" type="
          joint_state_publisher" />
37  </group>
38
39  <!-- Creating the Asctec firefly UAV in Gazebo -->
40  <group ns="firefly">
41    <!-- Send the robot XML to param server -->
42    <param name="robot_description" command="$(find xacro)/xacro.py '$(find
          tud_firefly_description)/urdf/firefly_for_landing.gazebo'
43                                            enable_logging:=true
44                                            enable_ground_truth:=true
45                                            enable_mavlink_interface:=false
46                                            log_file:=firefly
47                                            wait_to_record_bag:=false
48                                            mav_name:=firefly
49                                            namespace:=firefly" />
50
51    <!-- Push robot_description to factory and spawn robot in gazebo -->
52    <node name="spawn_firefly" pkg="gazebo_ros" type="spawn_model"
53          args="-param robot_description
54                -urdf
55                -x 0.0
56                -y 0.0
57                -z 0.0
58                -model firefly"
59       respawn="false" output="screen">
60    </node>
61
62    <!-- Publish the UAV robot states to topic tf -->
63    <node name="UAV_robot_state_publisher" pkg="robot_state_publisher" type="
          robot_state_publisher" />
64    <node name="UAV_joint_state_publisher" pkg="joint_state_publisher" type="
          joint_state_publisher" />
65  </group>
66
67  <!-- UAV position controller -->
68  <node name="firefly_landing_controller_node" pkg="tud_firefly_control" type="
        firefly_landing_controller_node" output="screen">
69    <rosparam command="load" file="$(find tud_firefly_control)/params/
          landing_controller.yaml" />
70    <rosparam command="load" file="$(find rotors_gazebo)/resource/firefly.yaml" />
71    <remap from="odometry" to="firefly/ground_truth/odometry" />
72    <remap from="command/trajectory" to="firefly/command/trajectory" />
73    <remap from="command/motor_speed" to="firefly/command/motor_speed" />
74    <remap from="R_des" to="firefly/R_des" />
75    <remap from="cmd_velocity" to="firefly/cmd_velocity" />
76    <remap from="angular_rate_des" to="firefly/angular_rate_des" />
77  </node> -->
78
79  <!-- Landing state machine -->
```

```
80  <node name="firefly_landing_state_machine" pkg="tud_firefly_application" type="
        landing_state_machine_node" output="screen">
81    <rosparam command="load" file="$(find tud_firefly_application)/params/
        landing_state_machine.yaml" />
82    <remap from="cmd_position" to="firefly/cmd_position" />
83    <remap from="cmd_velocity" to="firefly/cmd_velocity" />
84    <remap from="cmd_yaw" to="firefly/cmd_yaw" />
85    <remap from="predict_reporter" to="firefly/predict_reporter" />
86  </node>
87
88  <!-- <node name="hovering_example" pkg="rotors_gazebo" type="hovering_example"
        output="screen">
89    <remap from="command/trajectory" to="firefly/command/trajectory" />
90  </node> -->
91
92  <!-- Joystick interface node -->
93  <arg name="joy_dev" default="/dev/input/js4" />
94  <node name="joy" pkg="joy" type="joy_node" output="screen" >
95    <param name="dev" value="$(arg joy_dev)" />
96  </node>
97  <!-- Note that axis IDs are those from the joystick message plus one, to be able
        to invert axes by specifiying either positive or negative axis numbers.-->
98  <!-- Axis 2 from joy message thus has to be set as '3' or '-3'(inverted mode)
        below-->
99  <node name="joy_controller_node" pkg="tud_firefly_control" type="
        joy_controller_node" output="screen" >
100   <!-- <param name="x_axis" value="5"/>
101   <param name="y_axis" value="4"/>
102   <param name="z_axis" value="2"/>
103   <param name="yaw_axis" value="1"/> -->
104   <param name="x_velocity_max" value="1" />
105   <param name="y_velocity_max" value="1" />
106   <param name="z_velocity_max" value="1" />
107   <param name="yaw_rate_max" value="0.2" />
108   <param name="control_mode" value="twist"/>
109   <param name="x_axis" value="2"/>
110   <param name="y_axis" value="1"/>
111   <param name="z_axis" value="4"/>
112   <param name="yaw_axis" value="3"/>
113   <param name="motor_status" value="9"/>
114   <param name="landing_activation" value="10"/>
115   <param name="test_activation" value="2"/>
116   <remap from="firefly/cmd_vel" to="cmd_vel" /> -->
117 </node>
118
119 <!-- Image processor based on ar_sys -->
120 <!-- <node pkg="ar_sys" type="single_board" name="ar_sys_image_processor" output=
        "screen"> -->
121 <node name="firefly_image_processor" pkg="tud_firefly_application" type="
        ar_sys_single_board_mod_node" output="screen">
122   <remap from="/camera_info" to="firefly/camera_1_sensor/camera_info" />
123   <remap from="/image" to="firefly/camera_1_sensor/image_raw" />
124   <param name="image_is_rectified" type="bool" value="true" />
125   <param name="board_config" type="string" value="$(find tud_firefly_application)
        /params/marker_config/aruco_marker_76.yml"/>
126   <!--<param name="board_config" type="string" value="$(find
        tud_firefly_application)/params/marker_config/4_aruco_marker.yml"/> -->
127   <param name="board_frame" type="string" value="board76" />
128   <!--aruco marker 76 car: 0.3330 [m], in about 3.47 [m] available;
129       4 aruco marker car: 0.1710 [m], in about 1.78 [m] available;
```

```
130         single board:0,83 [m]; 4_aruco:0.43 [m] -->
131     <param name="draw_markers" type="bool" value="true" />
132     <param name="marker_size" type="double" value="0.3330" />
133     <param name="draw_markers_cube" type="bool" value="false" />
134     <param name="draw_markers_axis" type="bool" value="false" />
135     <param name="publish_tf" value="true" />
136   </node>
137
138   <!-- Image view programm that show the image result from ar_sys -->
139   <node name="image_view_camera_1" pkg="image_view" type="image_view" output="
        screen">
140     <remap from="image" to="firefly_image_processor/result" />
141     <!-- <remap from="image" to="whycon/image_out" /> -->
142     <param name="autosize" type="bool" value="true" />
143   </node>
144
145   <!-- Mahony complementary filter for UAV attitude estimation -->
146   <node name="firefly_mahony_complementary_filter" pkg="
        tud_firefly_state_estimation" type="mahony_complementary_filter_node"
        clear_params="true" output="screen">
147     <rosparam command="load" file="$(find tud_firefly_state_estimation)/params/
          mahony_complementary_filter.yaml" />
148   </node>
149
150   <!-- EKF state estimator of UAV -->
151   <node name="firefly_ekf_landing" pkg="tud_firefly_state_estimation" type="
        firefly_state_estimation_node" clear_params="true">
152     <rosparam command="load" file="$(find tud_firefly_state_estimation)/params/
          firefly_ekf_landing.yaml" />
153     <rosparam command="load" file="$(find rotors_gazebo)/resource/firefly.yaml" />
154     <remap from="odometry/filtered" to="firefly/odometry/filtered"/ -->
155     <remap from="accel/filtered" to="firefly/accel/filtered"/ -->
156   </node>
157
158   <!-- State monitor to send odometry/filtered result to tf so that it is available
         in rviz -->
159   <node name="firefly_state_monitor" pkg="tud_firefly_application" type="
        landing_monitor_node" output="screen">
160     <rosparam command="load" file="$(find tud_firefly_application)/params/
          landing_monitor.yaml" />
161     <remap from="odometry/filtered" to="firefly/odometry/filtered" />
162   </node> -->
163
164   <!-- Start rviz visualization with current config -->
165   <node pkg="rviz" type="rviz" name="rviz" args="-d $(find tud_firefly_gazebo)/
        rviz_cfg/tud_firefly_landing_sim.rviz"/>
166
167 </launch>
```

Normally, the UGV and the UAV can be controlled via joystick, the generic configuration of a xbox360 joystick is available in figure 3.3. However if you wish to launch an UAV autonomous landing simulation, input the following command in a terminal:

$ roslaunch tud_firefly_gazebo firefly_landing_control.launch

Press the button motor on/off, the UAV propellers will be activated/deactivated. Then by pressing the button landing activation/deactivation, the autonomous landing process will be activated/deactivated. You can use the joystick to control the UGV and drive it into the field of view from the UAV

onboard camera, if the target is identified by UAV and the landing process is activated, then the UAV will track the UGV autonomously until it lands on the UGV. The landing process can be divided into following phases:

1. **Land preparation mode:** The multicopter is initialized hovering above the origin of the earth plane, the height of the multicopter is set to 2.8 m to ensure that it has a horizontal FOV of at least 2.8 m; Once the target is available in the multicopter FOV, the tracking mode is activated.

2. **Trajectory tracking/prediction mode:** During this mode, the trajectory of the point, which is $1m$ right above the target UGV, will be set to the setpoint of the UAV. Two methods has been approached to track that specific point. Firstly a corresponding smooth trajectory could be defined by the cubic spline as

$$P(t) = P_0 + 3(P_d - P_0)\frac{t^2}{t_d^2} - 2(P_d - P_0)\frac{t^3}{t_d^3} \tag{3.1}$$

   for all three dimensions of UAV position in $\{N\}$-frame, where P_d is the desired position setpoint, p_0 is the corresponding UAV position where the trajectory get started, t_d is the desired time to reach the setpoint. The trajectory is smooth so that zero velocity along the trajectory for both beginning and end of the action. In the simulation, t_d is set equal to horizontal position error, so that the mean velocity is 1 m/s. The cubic spline trajectory will be updated every iteration when UGV is still in the multicopter FOV. After the trajectory tracking is over the time t_d, the system will be switched to approaching mode. Alternatively the trajectory prediction method based on curve fitting, which was interpreted in section **??**, is approached. The prediction algorithm is configured to require at least 100 sample points to derive the reliable UGV curve, the order of the curve is set to 4.

3. **Approaching mode:** In this mode, the UAV will keep tracking the setpoint but nolonger the trajectory. The target UGV will be continuously tracked during approaching mode, until the multicopter get to the point right upon the UGV, which means the error between the setpoint and the UAV position is smaller than a threshhold value, which is set to 5 cm in our simulation in order to increase the robusty of the system. The state machine will be switched into landing mode;

4. **Docking mode:** Once this mode is activated, the UAV is then descended with the cubic spline curve while tracking the UGV horizontal position still with hybrid position controller. The motors are turned off by the docking point.

Figure 3.3: The landing simulation workflow.

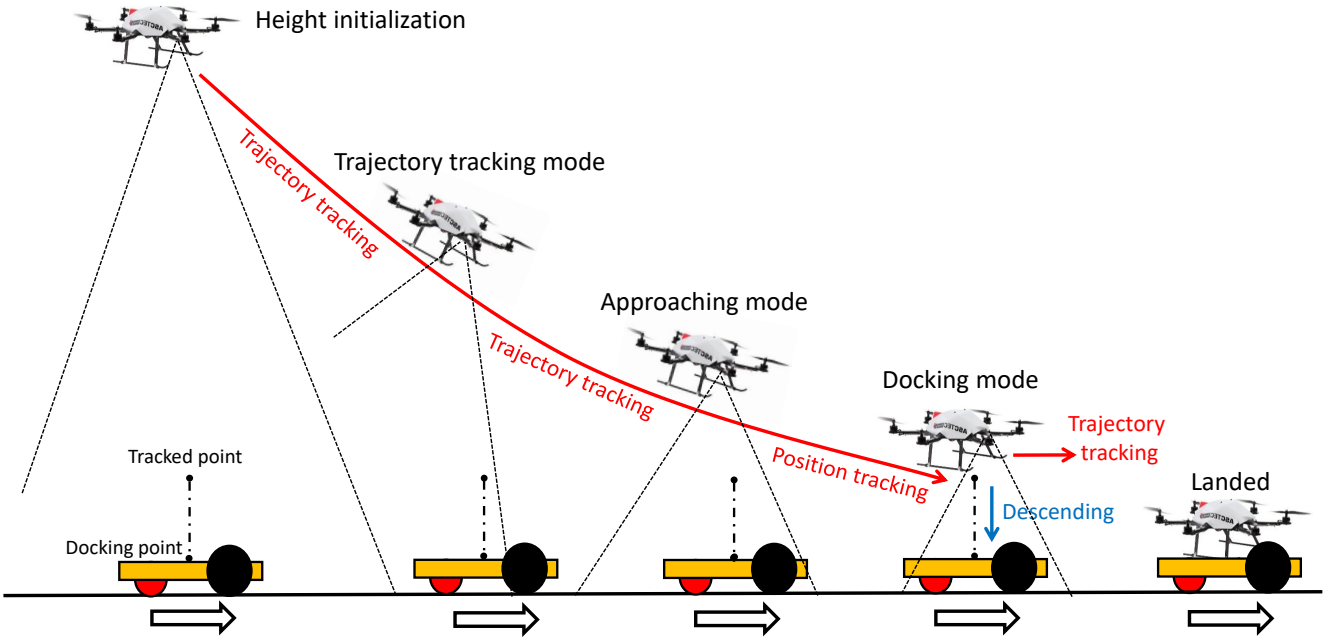


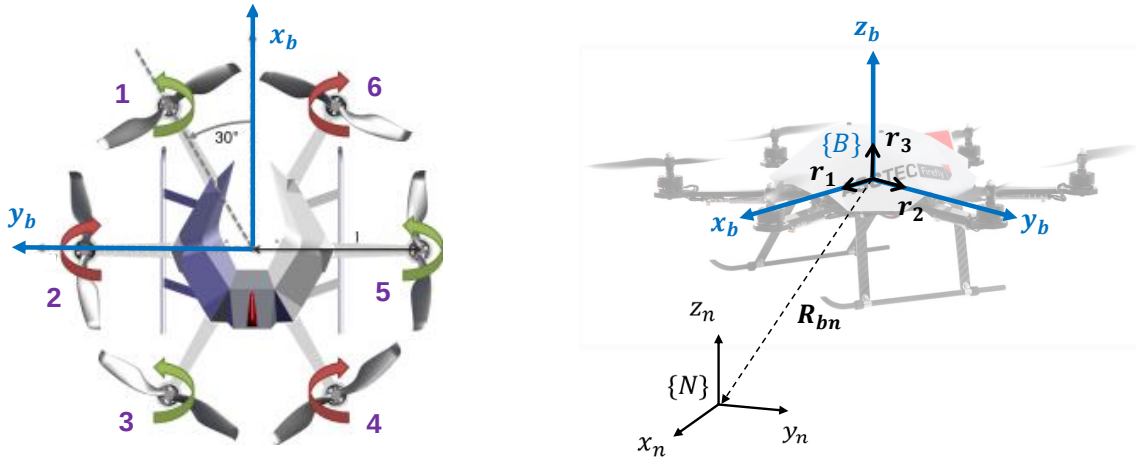
Figure 3.4: The landing simulation workflow.

4 The hexacopter UAV model: AscTec Firefly [14]

4.1 Assumptions

In order to simplify the modelling of the UAV and to make the control design easier, several reasonable assumptions are made:

- The UAV frame is rigid;
- The UAV structure is symmetrical, and thus its inertia matrix is diagonal;
- Blade flappinng and gyro effects of the rotors are neglected, as expected velocites are small;
- The UAV has a constant mass.



(a) UAV configuration (b) Reference frames

Figure 4.1: The principle of reference frames and rotational motion described by Euler angles [15]

4.2 Dynamic equations

The mathematical model of the hexacopter UAV is constructed according to the *AscTec Firefly* hexacopter, whose configuration is shown in figure 4.1 (a). Assuming the earth-fixed Cartisian coordinate system is $\{N\}$, and the UAV body-fixed Cartisian coordinate system is $\{B\}$ (whose unit orientation vectors are $\boldsymbol{r}_1$,$\boldsymbol{r}_2$,$\boldsymbol{r}_3$), as shown in figure 4.1. Those coordinate systems are ENU (East North Up) frames. The rotation

matrix from $\{N\}$-frame to $\{B\}$-frame is denoted by $\boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{n}}$, which can also be expressed in Euler angle form $\boldsymbol{\Theta} = (\phi,\theta,\psi)^T$ (roll, pitch, yaw), as given by

$$\boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{n}}(\psi,\theta,\phi) = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \cos\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix}. \quad (4.1)$$

The differential equations of the whole UAV system for control design can thus be described in vector form as

$$\dot{\boldsymbol{P}}_{\boldsymbol{n}} = \boldsymbol{V}_{\boldsymbol{n}}, \quad (4.2)$$

$$\dot{\boldsymbol{V}}_{\boldsymbol{n}} = \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{n}} \begin{bmatrix} 0 \\ 0 \\ \frac{T}{m} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{n}} \boldsymbol{F}_{\boldsymbol{d}}, \quad (4.3)$$

$$\dot{\boldsymbol{R}}_{\boldsymbol{b}}^{\boldsymbol{n}} = \boldsymbol{R}_{\boldsymbol{b}}^{\boldsymbol{n}} \boldsymbol{S}(\boldsymbol{\Omega}), \quad (4.4)$$

$$\boldsymbol{J}\dot{\boldsymbol{\Omega}} = \boldsymbol{\Omega} \times \boldsymbol{J}\boldsymbol{\Omega} + \boldsymbol{\tau}_{\boldsymbol{a}} + \boldsymbol{M}_{\boldsymbol{r}}, \quad (4.5)$$

with the system state vectors:

- Absolute UAV position in $\{N\}$-frame: $\dot{\boldsymbol{P}}_{\boldsymbol{n}} = (x_n,y_n,z_n)^T$;
- Linear velocity in $\{N\}$-frame: $\boldsymbol{V}_{\boldsymbol{n}} = (v_x,v_y,v_z)^T$;
- UAV orientation in Euler angle form: $\boldsymbol{\Theta} = (\phi,\theta,\psi)^T$;
- Angular rates in $\{B\}$-frame: $\boldsymbol{\Omega} = (p,q,r)^T$, whose skew-symmetric matrix is $\boldsymbol{S}(\boldsymbol{\Omega})$.

And with the inputs signals:

- Thrust force of actuation in $\{B\}$-frame: $\boldsymbol{T} = (0,0,T)^T$;
- Rotary moments of actuation in $\{B\}$-frame: $\boldsymbol{\tau}_{\boldsymbol{a}} = (\tau_1,\tau_2,\tau_3)^T$.

And disturbances, which will be discussed in the next section:

- Air drag force due to relative wind in $\{B\}$-frame: $\boldsymbol{F}_{\boldsymbol{d}}$;
- Rolling moment due to relative wind in $\{B\}$-frame: $\boldsymbol{M}_{\boldsymbol{r}}$.

As well as gravity and UAV parameters: [10]

- Gravity factor: $g = 9.81[m/s^2]$;
- UAV total mass in at center of gravity: $m = 1.56779[kg]$;
- UAV inertial matrix $\boldsymbol{J} = diag(I_u,I_v,I_w) = diag(0.0347563,\quad 0.0458929,\quad 0.0977)[kg \cdot m^2]$.

Equation (4.4) can also be expressed as

$$\dot{\boldsymbol{\Theta}} = \boldsymbol{W}\boldsymbol{\Omega}, \quad (4.6)$$

where the 3×3-matrix $\boldsymbol{W}$ is given by

$$\boldsymbol{W} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix}. \quad (4.7)$$

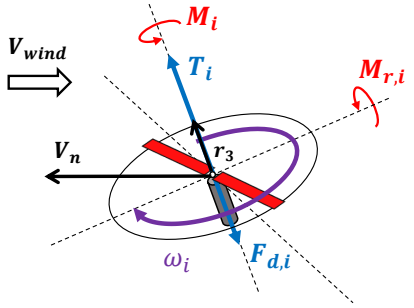


Figure 4.2: The forces and torques acting on the single rotor in $\{B\}$-frame

4.3 Rotor propulsion model

As shown in figure 4.2, the actuation dynamic of a single rotor i $(i = 1,2,\cdots,6)$ can be expressed as

$$T_i = k_T \omega_i^2 \tag{4.8}$$

$$M_i = (-1)^{i-1} k_M T_i, \tag{4.9}$$

with propeller states:

- Propeller rotary speed: $\omega_i[rad/s]$,
- Generated thrust force: $T_i[N]$,
- Generated rotary moment: $M_i[Nm]$.

And the propeller parameters: [10]

- Propeller thrust constant: $k_T = 8.54858 \times 10^{-6}$,
- Propeller rotary moment constant: $k_M = 1.6 \times 10^{-2}$.

Thus the map from the rotary speed vector of the six rotors $(\omega_1,\omega_2,\cdots,\omega_6)^T$ to the input vector of the UAV model $(T,\tau_1,\tau_2,\tau_3)^T$ can be described as a so called allocation matrix $\mathfrak{A}$, as expressed below:

$$\begin{bmatrix} T \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \underbrace{\begin{bmatrix} k_T & k_T & k_T & k_T & k_T & k_T \\ -lk_T\sin(\frac{\pi}{6}) & -lk_T & -lk_T\sin(\frac{\pi}{6}) & lk_T\sin(\frac{\pi}{6}) & lk_T & lk_T\sin(\frac{\pi}{6}) \\ -lk_T\cos(\frac{\pi}{3}) & 0 & -lk_T\cos(\frac{\pi}{3}) & lk_T\cos(\frac{\pi}{3}) & 0 & lk_T\cos(\frac{\pi}{3}) \\ k_M k_T & -k_M k_T & k_M k_T & -k_M k_T & k_M k_T & -k_M k_T \end{bmatrix}}_{\mathfrak{A}} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \\ \omega_5^2 \\ \omega_6^2 \end{bmatrix}, \tag{4.10}$$

where $l = 0.215[m]$ denotes the UAV arm length. [10]

Furthermore, the rotor can also generate disturbance forces and moments due to the relative wind. Assuming the quadrotor is flying with linear velocity vector $\boldsymbol{V_n}$ and the wind speed in $\{N\}$-frame is $\boldsymbol{V_{wind}}$. Thus the relative wind speed of the UAV is $\boldsymbol{V_a} = \boldsymbol{V_{wind}} - \boldsymbol{V_n}$, whose projection onto the rotor propeller disc plane is [16]

$$\boldsymbol{V_a^{\perp}} = \boldsymbol{V_a} - (\boldsymbol{V_a} \cdot \boldsymbol{r_3}) \cdot \boldsymbol{r_3}. \tag{4.11}$$

The corresponding air drag force $\boldsymbol{F_{d,i}}$ at the rotor i and rolling moment $\boldsymbol{M_{r,i}}$ at the CoG of the UAV can thus be described as

$$\boldsymbol{F_{d,i}} = \omega_i k_d \boldsymbol{V_a^{\perp}}, \tag{4.12}$$

$$\boldsymbol{M_{r,i}} = \omega_i k_r \boldsymbol{V_a^{\perp}}, \tag{4.13}$$

with the rotor drag constant $k_d = 8.06428 \times 10^{-5}$ and the rolling moment constant $k_r = 1.0 \times 10^{-6}$. [10] The total air drag force $\boldsymbol{F_d}$ and rolling moment $\boldsymbol{M_r}$ is then

$$\boldsymbol{F_d} = \sum_{i=1}^{6} \boldsymbol{F_{d,i}}, \tag{4.14}$$

$$\boldsymbol{M_r} = \sum_{i=1}^{6} \boldsymbol{M_{r,i}}. \tag{4.15}$$

4.4 Linearization of the UAV model around hover state

For the control design and tuning a corresponding linear model is required, this can be achieved via linearization. It is based on the theorem that a non-linear system can be approximately considered as a linear system in a small domain around a equilibrium point. A generic non-linear system model is given by

$$\dot{\boldsymbol{x}} = \boldsymbol{f(x,u)}, \tag{4.16}$$

where $\boldsymbol{x}$ is the system state variable vector, and $\boldsymbol{u}$ is the system input vector. Defining that $\boldsymbol{f(x,u)}$ is continuously differentiable in a small range around a equilibrium point $(\boldsymbol{x_e,u_e})^T$. Thus the linearized model around this point is given by

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu} \tag{4.17}$$

with

$$\boldsymbol{A} = \left.\frac{\partial \boldsymbol{f(x,u)}}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x=x_e,u=u_e}}, \qquad \boldsymbol{B} = \left.\frac{\partial \boldsymbol{f(x,u)}}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x=x_e,u=u_e}}. \tag{4.18}$$

The non-linear quadrotor model equations (4.2)-(4.3) can be linearized around the hover state $\phi = 0$, $\theta = 0$, where $\sin\phi \approx \phi$, $\sin\theta \approx \theta$ and $\cos\phi \approx 0$, $\cos\theta \approx 0$. The linearized model is then derived as

$$\dot{\boldsymbol{x}} = \boldsymbol{Ax} + \boldsymbol{Bu}, \qquad \boldsymbol{y} = \boldsymbol{Cx}, \tag{4.19}$$

with

$$\boldsymbol{x} = \begin{bmatrix} x_n \\ y_n \\ z_n \\ u \\ v \\ w \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} T - g \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} \quad \boldsymbol{y} = \begin{bmatrix} x_n \\ y_n \\ z_n \\ \phi \\ \theta \\ \psi \end{bmatrix} \quad \boldsymbol{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{A}=\begin{bmatrix}
0 & 0 & 0 & \cos\psi & -\sin\psi & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & \sin\psi & \cos\psi & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\qquad
\boldsymbol{B}=\begin{bmatrix}
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
\frac{1}{m} & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
0 & \frac{1}{I_u} & 0 & 0\\
0 & 0 & \frac{1}{I_v} & 0\\
0 & 0 & 0 & \frac{1}{I_w}
\end{bmatrix},$$

where the vector $\boldsymbol{V_b}=(u,v,w)^T=\boldsymbol{R_{nb}V_n}$ is the UAV linear velocity vector in $\{B\}$-frame.

Bibliography

[1] Ros tutorial. http://wiki.ros.org/ROS/Tutorials, 2015.

[2] Building a ros package. http://wiki.ros.org/ROS/Tutorials/BuildingPackages, 2015.

[3] *catkin* tutorial. http://wiki.ros.org/catkin, 2015.

[4] *CMAKE* tutorial. https://cmake.org/.

[5] *CMakeList.txt* in a *catkin* ros package tutorial. http://wiki.ros.org/catkin/CMakeLists.txt.

[6] *roslaunch* tutorial. http://wiki.ros.org/roslaunch/XML.

[7] Metapackage tutorial. http://wiki.ros.org/Metapackages, 2015.

[8] An overview of gazebo_ros. http://gazebosim.org/tutorials?tut=ros_overview#CMakeLists.txt.

[9] Using gazebo plugins with ros tutorial. http://gazebosim.org/tutorials?tut=ros_gzplugins.

[10] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[11] Aruco marker online generator. http://terpconnect.umd.edu/~jwelsh12/enes100/markergen.html.

[12] Aruco marker online generator. http://wiki.ros.org/ar_sys.

[13] *robot_localization* tutorial. http://docs.ros.org/indigo/api/robot_localization/html/index.html.

[14] Mina Kamel, Kostas Alexis, Markus Achtelik, and Roland Siegwart. Fast nonlinear model predictive control for multicopter attitude tracking on so (3). In *Control Applications (CCA), 2015 IEEE Conference on*, pages 1160–1166. IEEE, 2015.

[15] AscTec Firefly wiki, 2016.

[16] Philippe Martin and Erwan Salaün. The true role of accelerometer feedback in quadrotor control. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1623–1629. IEEE, 2010.