


XHZhang01 / **autonomous-navigation** Public

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#)

 master ▾


...

autonomous-navigation / [Group_Report](#) / **Report.md**



XHZhang01 Update Report.md

 **History**

 1 contributor

 209 lines (140 sloc) | 12.2 KB

...

Project: Autonomous Navigation of Quadrotor

- i. [Author](#)
- ii. [Architecture](#)
- iii. [Important commits](#)
 - 3.1. [Construction of the overall architecture](#)
 - 3.2. [Trajectory publisher](#)
 - 3.3. [State machine](#)
 - 3.4. [Goal sender](#)
 - 3.5. [Use ros-time](#)
 - 3.6. [Add keyboard control packages](#)
 - 3.7. [Change the frame "/>Quadrotor/Sensors/DepthCamera"](#)
 - 3.8. [Add frame "body_foot"](#)
 - 3.9. [Convert the depth image](#)
 - 3.10. [Move_base and the optimization of parameters](#)
 - 3.11. [Setup Octomap](#)
 - 3.12. [Add a skycamera in the Unity environment](#)

- iv. [External code](#)

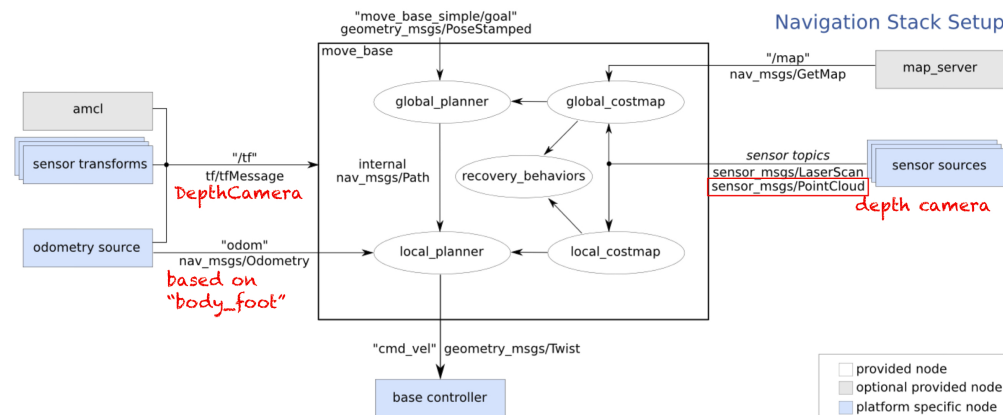
1. Author

- Xuhui Zhang (xuhui.zhang@tum.de)

2. Architecture

We tried firstly to use octomap to build the voxel-grid representation of the environment and used the projected map as the basis for the 2D-navigation. However, the size of the projected map can not be manually adjusted so that the navigation did not work directly, if we set the goal location outside of the map. As a result, we decided to only use the package "move_base", which is capable of building maps and planning for 2D-navigation in real time.

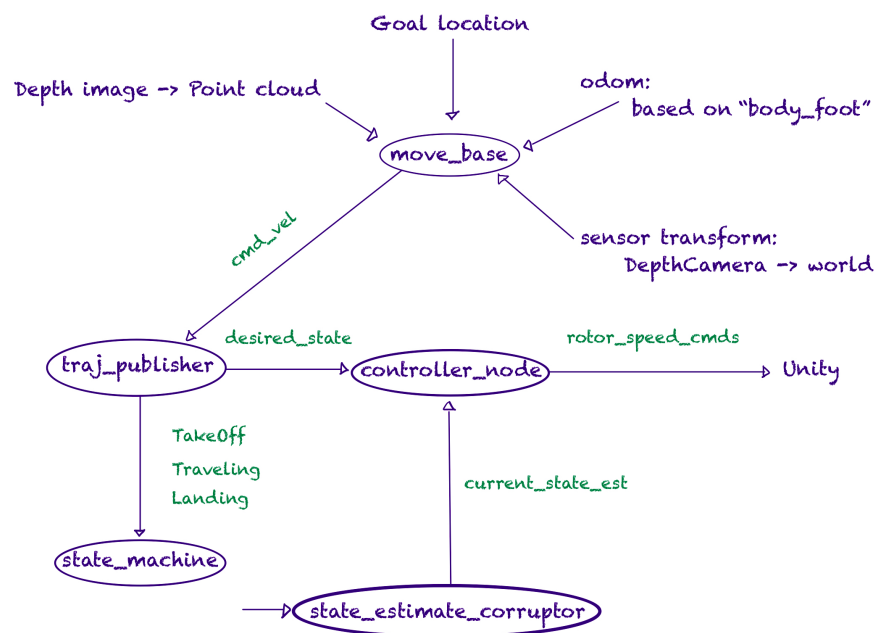
An illustration of move_base from ROS wiki is shown below.



Essential configuration options for move_base in our project:

- **sensor source:** we converted the depth image from the depth camera into the point cloud.
- **sensor transforms:** frame relation between the depth camera frame and the world frame.
- **odometry source:** based on the ground-projected frame "body_foot" from the frame "body" of the quadrotor.

The global planner generates the global path based on the goal location and the global costmap. The local planner publishes the velocity commands, which be subscribed by the node `traj_publisher`, as can be seen in the following illustration.



The node `traj_publisher` will integrate the velocity commands into the desired state. Furthermore, the node `traj_publisher` will publish signal flags to the node `state_machine` so that the states "take off", "traveling" and "landing" can be switched and displayed. In addition, the node `controller_node` will use the desired state and the corrupted current state to generate the rotor speed commands, which will be sent to unity to update the simulation environment.

3. Important commits

3.1. Construction of the overall architecture

simulation/launch/simulation.launch

```

simulation/launch/move_base_kinect.launch simulation/launch
/octomap_mapping.launch(optional) simulation/launch
/point_cloud_convert.launch navigation/rviz/nav_kinect.rviz
tf2/static_transform_publisher * 6 simulation/src
/state_estimate_corruptor_node controller_pkg/src/controller_node.cpp
simulation/launch/unity_ros.launch simulation/Linux_build.x86_64
simulation/w_to_unity

```

controller_pkg/src/state_machine.cpp

simulation/launch/mission_starter.launch

controller_pkg/src/traj_publisher.cpp

navigation/src/send_goal.cpp

In order to realize the software architecture we designed, we modified or added some .launch files. Their structure is shown in the figure above. Bold text represents modified or new files.

3.2. Trajectory publisher

controller_pkg/src/traj_publisher.cpp

We modified this file so that the trajectory_publisher node can switch different states according to different signal flags, including take off, travelling, and landing. And publish the current states to state_machine node. Besides, it can subscribe and integral the speed command "/cmd_vel" from move_base during the travelling state. Then publish the generated trajectory point to the controller node.

3.3. State machine

controller_pkg/src/state_machine.cpp

This ROS node is responsible for showing the current state of the quadrotor. For this purpose, subscribers are generated for receiving the state-changing information from the corresponding publishers in the ROS node "traj_pub.cpp" in terms of "take off", "traveling" and "landing".

3.4. Goal sender

navigation/src/send_goal.cpp

This file is modified from a programm on github. [\[Link\] \(https://github.com/guyuehome/ros_basic_tutorials/blob/dfb8afac81929040ea659a40c03d1d41f2897b5f/mbot_navigation/mbot_navigation/src/move_test.cpp\)](https://github.com/guyuehome/ros_basic_tutorials/blob/dfb8afac81929040ea659a40c03d1d41f2897b5f/mbot_navigation/mbot_navigation/src/move_test.cpp)

By default, the node will send the goal (100,0,0) to move_base. You can also specify a goal (x, y, 0) by running

```
roslaunch navigation send_goal x y 0
```

3.5. Use ros-time

simulation/src/unity_ros.cpp

By default all messages from unity will be stamped with unity-time, which starts from 0 s in the beginning. This will cause synchronization failures in move_base as the other messages use ros-time. So we made the following change:

```
// original code, use unity_time:
    //header.timestamp = static_cast<double>(timestamp_raw) * 1e-7;
// use ros_time:
    header.timestamp = ros_time;
```

3.6. Add keyboard control packages

teleop teleop_twist_keyboard_cpp

In order to control the quadrotor with keyboard and for debug reason we added two keyboard control nodes to the /src folder.

The "teleop" package is cloned from https://github.com/guyuehome/ros_basic_tutorials.git, /mbot_navigation/mbot_teleop. This package can only send commands to /cmd_vel in x- and omega_z-direction but acts smoother than the other.

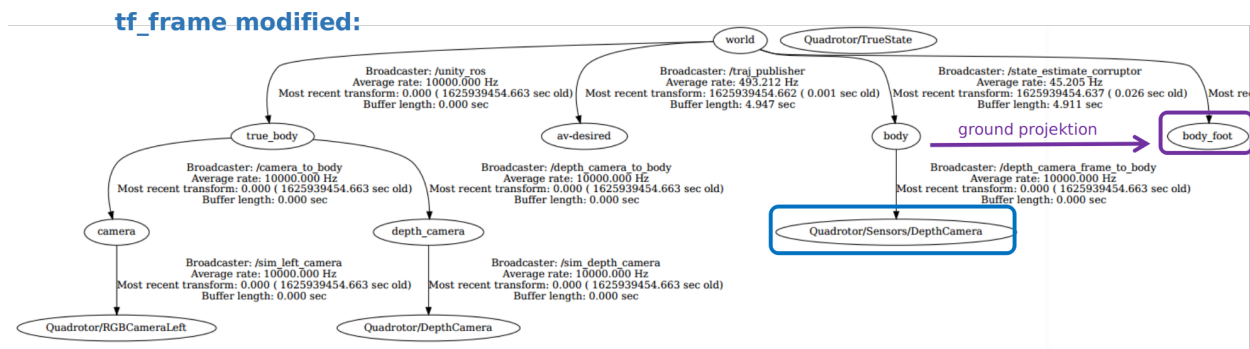
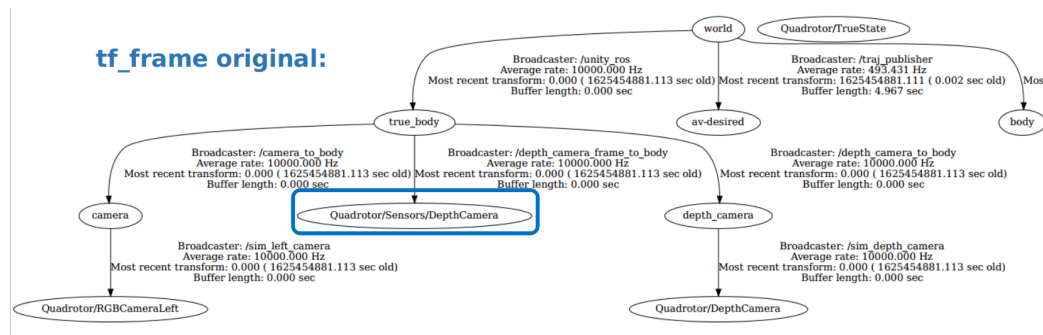
The "teleop_twist_keyboard_cpp" package is cloned from https://github.com/methylDragon/teleop_twist_keyboard_cpp. This package can control the quadrotor omnidirectionally.

3.7. Change the frame "/Quadrotor/Sensors/DepthCamera"

simulation/launch/simulation.launch

The parent frame of "/Quadrotor/Sensors/DepthCamera" was set to "/true_body", which is impossible in the real sense. So we changed its parent frame to corrupted frame "/body" and updated the relative pose between them.

```
<!-- change the parent frame from '/true_body' to 'body' -->
<node pkg="tf2_ros" type="static_transform_publisher"
    name="depth_camera_frame_to_body"
    args="0 0 0 -1.571 0 -1.571 /body /Quadrotor/Sensors/DepthCamera" ,
```



3.8. Add frame "body_foot"

simulation/src/state_estimate_corruptor_node

Both **move_base** and **Octomap** need the ground projection of the quadrotor's frame "body" as a input. So we implemented it in state_estimate_corruptor by adding

```

ros::Publisher corrupted_state_pub_foot;
corrupted_state_pub_foot = nh.advertise<nav_msgs::Odometry>("/current_state_foot");
void PublishCorruptedState_Foot(geometry_msgs::TwistStamped const& corrupt_state) {
    void PublishBodyTransform_Foot(geometry_msgs::PoseStamped const& pose) {

```

to the file.

3.9. Convert the depth image

simulation/launch/point_cloud_convert.launch

We use the recommended package **depth_image_proc** from http://wiki.ros.org/depth_image_proc to generate point cloud from depth image.

3.10. Move_base and the optimization of parameters

simulation/launch/move_base_kinect.launch navigation/config/*.yaml

This is the most difficult and time-consuming part of this project. We have been optimizing the various parameters in the five .yaml files to ensure that the quadrotor can quickly recognize the environment, plan the path, and then fly to the goal smoothly.

Some key parameter settings are listed below:

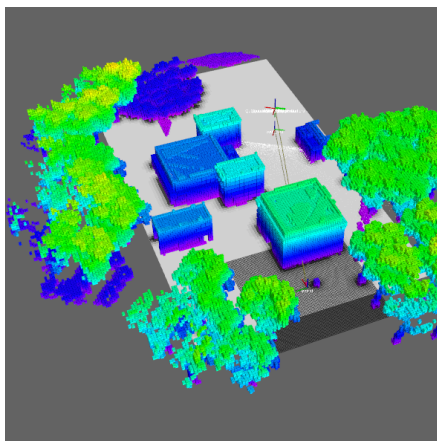
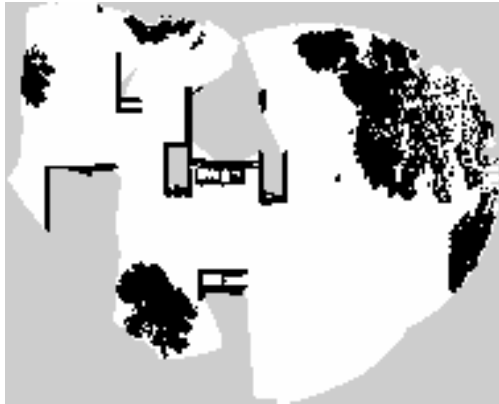
- global_costmap_params.yaml: **robot_base_frame:** body_foot The input frame must be the ground projection of the quadrotor's frame.
- dwa_local_planner_params.yaml: **max_vel_y:** 0, **min_vel_y:** 0, **acc_lim_y:** 5 We noticed before that the quadrotor did not face forwards when turning and was very unstable. The reason is that, we set the acc_y to 0, which is not suitable for a quadrotor. Imagine the following, a quadrotor is moving along a circle path, its x-axis points forward and the y-axis points to the center of the circle. In the quadrotor coordinate system, $v_x = \text{cons.}$, $v_y = 0$ and $a_x = 0$. The most important is that a_y is a value greater than 0. After updating the parameter, the quadrotor can always face forwards and moves more stable.
- dwa_local_planner_params.yaml: **max_vel_x:** 6, **max_vel_trans:** 6, **acc_lim_x:** 20 ... These parameters decide the dynamic performance of the quadrotor. Limited by the control frequency and map update frequency, they cannot be set too large.
- move_base_params.yaml: **controller_frequency:** 3
dwa_local_planner_params.yaml: **controller_frequency:** 3 Theoretically these two parameters are both used to set the control frequency, and the first parameter has a higher priority so there is no need to set the second one. But in fact, they must be the same value, otherwise the desired control frequency will be missed.
- dwa_local_planner_params.yaml: **xy_goal_tolerance:** 1, **yaw_goal_tolerance:** 1 If these two values are set too small, the quadrotor will oscillate above the goal point.

3.11. Setup Octomap

```
simulation/launch/octomap_mapping.launch
```

We use **Octomap** to generate voxel-grid 3D map from point cloud. To filter out the ground, the parameter "base_frame_id" should be set to "body_foot", which is the ground projection of the quadrotor's frame "body". Other parameters are also important for the filtering effect.

We can also get a 2D projected map from the 3D octomap. In theory, the projected map can be used for navigation. But the original size of the projected map is limited to the detection distance at the beginning (about 30*30). And it is not easy to send a goal out of the map to move_base directly, e.g. (100,0,0), without increasing travel time. So we only use this package for mapping in keyboard controll mode to get a voxel-grid representation of the environment.



3.12. Add a skycamera in the Unity environment

In order to better observe the relative position of the quadrotor and the building, I changed the unity environment and added a skycamera which accompanies the quadrotor. Both the executable files and the modified unity environment can be dowanloaded from the following link: <https://syncandshare.lrz.de/getlink/fi4C8XwwgNP3gtPScV2b48pW/>

[git_gif](#)

4. External code

-

navigation/src/send_goal.cpp: https://github.com/guyuehome/ros_basic_tutorials/blob/dfb8afac81929040ea659a40c03d1d41f2897b5f/mbot_navigation/mbot_navigation/src/move_test.cpp

- teleop: https://github.com/guyuehome/ros_basic_tutorials.git
- teleop_twist_keyboard_cpp: https://github.com/methylDragon/teleop_twist_keyboard_cpp