UAV exploration

Alessandro Melone*, Luigi D'Amico†

Automation Engineering and Robotics, University Federico II

6th July 2020

Contents

Abstract

In this repository is available the implementation of a technical project assigned during the "Robotics Lab" course of
Prof. Jonathan Cacace [1] at University of Naples Federico II.

1 Description of the problem

The goal of this project is to develop an autonomous control system for UAV to explore the world searching the 7 tag placed into the scene and later, be able to navigate the tag position with a given parametric input sequence.

In order to navigate the marked position, the UAV must land on each position, wait 5 seconds and take off to navigate the next marker location.

*https://github.com/AlessandroMelone
†https://github.com/luigidamico100
[1]https://github.com/jocacace

After that the specified sequence has been navigated, the robot must land on the initial position (the blue platform).

The world's dimension are 10x20 meters. It is delimited by colored pillars. These information can be used in the exploration phase.

To perform the exploration task it was assumed to know the dimensions of the environment and the initial position of the UAV, so the path planning considering x and y coordinates is based on a snake-like movement (common for domestic robots), while the z have a lower limit but its value is calculated on the basis of the generated octomap. The distance between two parallel segment of the path (considering the x and y coordinates) is the stride length. In order to have a stride length of 2.5 meters the z lower limit is computed considering the formula in Fig.1, in which the focal length correspond to the height of the UAV, and image width correspond to the stride length.

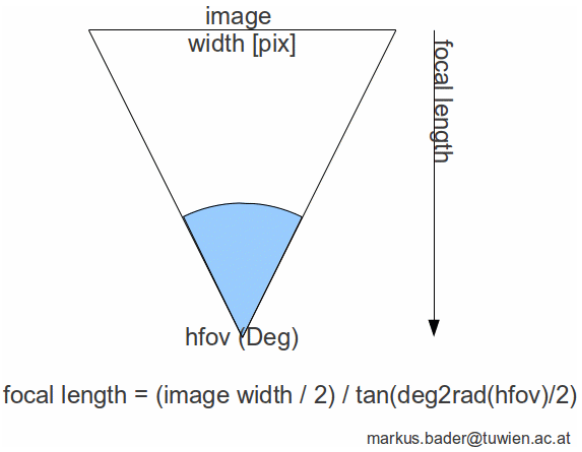

Figure 1: Formula used to compute minimum flight height

2 Requirements

Requirement to guarantee the correctly executing of the code:

- Operating system: `Ubuntu 18.04`
- ROS version: `Melodic`
- ROS package needed: `visp_auto_tracker`, `OMPL`, `Eigen`, `KDL`.

3 Structure of the project

The project is composed by four packages that communicates via `mavros` with `jocacace\Firmware` that allows to run a Px4 Software In the Loop (SITL) simulation in gazebo:

- `master_pkg`: this package contains two nodes, namely `master_node` and `tf_map2base_node`.
  - `master_node`: implement the interface between the user and the whole package, i.e. through a menu that gives the opportunity to start different task.

  -`tf_map2base_node`: read the position of the UAV from the `mavros` topic `/mavros/local_position/pose` and send the `tf` transform between the frame `map` and the frame `base_link`.
  Note that the transformation is not sent if the `mavros_state` is equal to `AUTO.LAND`, this choice is due to the strange value of `/mavros/local_position/pose` when the UAV is landed. If no remedial action are taken the octomap would be wrongly built when the UAV is on the ground.

- `qr_detector_pkg`: this package contains a node named `qr_detector_node`, the aim of this node is to use the information provided by VISP (camera frame) and transform them in order to behave like a server containing all the position (map frame) of the QR codes that have been read.

- `px4_planner`: contains two nodes, namely `fly_action_server_node` and `fly_action_client_node`.

  - `fly_action_client_node`: this node receive `planner_commander_service.srv` request from the master and use `ompl` to compute free obstacles path which are sent to `fly_action_server_node` via the "ROS Action Protocol".

  The aim of this node is to receive a "long path" (in the x and y coordinates) from the master node and split into more "short path" (in the x and y coordinates). The computed paths are sent to `fly_action_server_node` that directly commands the UAV. Once a path are sent, the node waits that is executed by the UAV and compute the next path using a RTT star planning algorithm.
  Given a path in x and y coordinates, the planning algorithm is used to obtain the z coordinates needed to avoid obstacles. The obstacles are detected using an octomap that is built during the UAV movements.

The solution to plan more "short path" is adopted because the entire octomap of the environment is unknown and has to be built.

- `fly_action_server_node`: the aim of this node is to implement an interface to allow the execution of exploration subtask (takeoff, landing, following path, landing on a QR code).

The communication is performed via the "ROS Action Protocol", i.e. this node implement a SimpleActionServer. To execute a subtask will be necessary to send a goal with the correct bitmask, which encoding is specified in the action definition file.

To avoid the stuck at the end of a "short path", caused by the time needed to communicate with the client and to construct the KDL trajectory object, the action server use a queue of KDL trajectory. Precisely when the action server receive the first FOLLOWPATH action goal waits the next goal before to start the path following. The purpose is to have in the queue, during the following of the current trajectory (corresponding to a "short path" element), the next ready to use KDL trajectory (corresponding to the next "short path" element), in order to let the PID controller fluently track the "long path" composed by "short path" elements.

- `exploration_world`: this package is created by the Prof. Jonathan Cacace and implement the gazebo world in which the UAV operates.

4 Risk analysis

Is necessary that the QR code surface is free from obstacles, otherwise there is a risk of collision of the UAV with the environment, the algorithm suppose that the QR code surface is free from obstacles.
When a QR code is tracked by VISP it will continue to be tracked until it go out of rgb sight, and since VISP can't track more than one QR code at the same time, if two QR code are near (distance less that the computed stride) there is a risk that one of the two QR codes are not detected during the exploration phase.

Test shows that rarely the planning algorithm, in order to avoid an obstacle, go under that (in general higher the obstacle, greater the possibility to go under it), more likely the generated path will pass over the obstacle. So if a QR code is placed under an obstacle, then rarely it will be detected.
Moreover if the QR code is placed on an obstacle another concern is that if the QR code is too close to the camera, then it will not be detected.

Lastly, even if the QR code is on the ground an obstacle could be so near to the RGB camera that doesn't allow to detect a QR code. This problem,

like others mentioned above, could be avoided if the scouting is repeated iteratively with slight different trajectories until all the QR codes are found (cause to slow simulation this solution is not implemented).

5 Rviz

After done the command `roslaunch master_pkg all.launch` will start rviz and will show:

- `tf` frame `base_link`: position of the UAV taken from the `mavros/local_position/pose`.
- `tf` frame `map`: correspond to the initial position of the UAV.
- desired pose pose of the UAV: red arrow.
- velocity command computed by the PID: rviz marker green arrow.
- last path sent by the action client to the action server: green line.

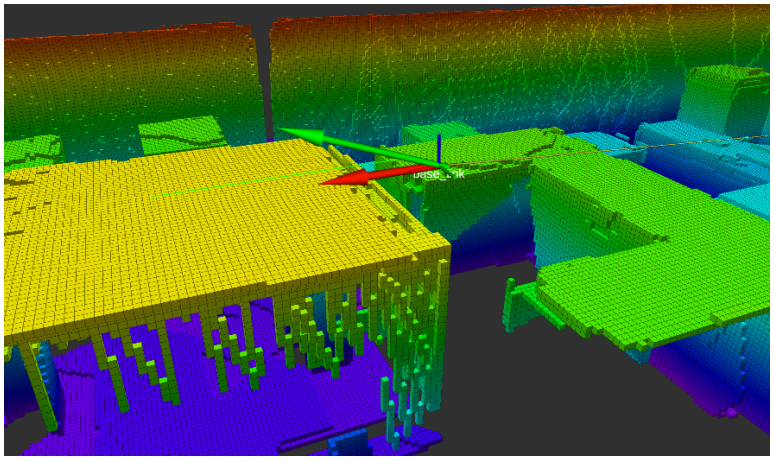

Figure 2: Rviz display during the exploration phase

6 Test

Multiple tests different QR codes positions can be viewed at `https://youtu.be/9q2p-nuvVdM`
To repeat the test, considering the menu that appears after launched `all.launch`, do:

- After the menu appears, type "1" to start the exploration phase.

- When the exploration phase is finished the menu will reappear, so type "3" to see the detected QR codes list
- To reach a QR code list type "7" and write the QR codes list, e.g. "5 8 4 6"