Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Autonomous Systems Lab
Prof. Roland Siegwart

Semester Thesis

High Level Control of an MAV through Marker-Based Visual Commands

Autumn Term 2015

Supervised by:
Helen Oleynikova
Michael Burri

Author:
Dimitris Gryparis

Declaration of Originality

I hereby declare that the written work I have submitted entitled

High Level Control of an MAV through Marker-Based Visual Commands

is original work which I alone have authored and which is written in my own words.[1]

Author(s)

Dimitris Gryparis

Student supervisor(s)

Helen Oleynikova
Michael Burri

Supervising lecturer

Roland Siegwart

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date

Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Contents

Abstract

This project presents the high level control of an Micro Aerial Vehicle (MAV) through Marker-Based visual commands. First experimental setups run in a computer simulated world inside Gazebo simulator[1] and then experiments are conducted with the real MAV. In the simulation the visual commands are given from a usb-camera connected to the computer while in the latter case from the MAV's camera sensor. In both cases the used visual markers are Apriltags[**?**].

Symbols

Symbols

ϕ, θ, ψ	roll, pitch and yaw angle

Indices

x	x axis
y	y axis
z	z axis

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
IMU	Inertial Measurement Unit
UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
ROS	Robot Operating System

Chapter 1

Introduction

Chapter 2

Description of the early packages used in simulation

At the beginning of the project, were created some ROS packages so as to obtain the necessary experience and get familiar with the system. Furthermore, possibly dangerous or unpredicted situations in the real system were avoided by carefully studying the MAV's simulated responses to the experiments. All the following packages were written on a computer running on Ubuntu 14.04 LTS alongside with ROS Indigo Igloo. The simulator used in this project is Gazebo[1] as provided by ROS. Now follows the description of the ROS packets, some packets with very similar role and structure are omitted from the description. All the aforementioned packages are written in the C++ programming language[2] and can be accessed at the ASL's GitHub page under the repository mav_demos[3].

2.1 Connecting the Camera with ROS

The very first task was to connect a camera to the ROS so as to obtain the images of the markers. The camera chosen for this project was the Logitech Tessar HD, for its calibration and image rectification the ROS camera_calibration package and the image _proc node were used respectively. After the camera setup, the detection of the Apriltags[4] came into focus. In order to detect the position and orientation of the aforementioned, the apriltag_ros package was used. This provides a ROS wrapper for the C++ library[13]. An example of such a tag is provided in figure 2.1. It should be mentioned that although this library detects distance and yaw accurately it faces some problems with the detection of the marker's roll and pitch angles.

2.2 Moving the Simulated MAV

After the camera setup, the main focus moved towards connecting the camera image with simulated MAV. This was conducted in the mav_demo_camera package. In there two nodes are created, except of course those referring to camera, one that detects the Apriltag and publishes the detection data to a ROS topic and another one that conducts the motion. In this specific example the user moves the tag in front of the camera and the simulated MAV moves accordingly. As can be seen from figure 2.2 when the package is ran, two windows are created, one depicting the detected apriltag, for the user to know whether or not the marker is detected

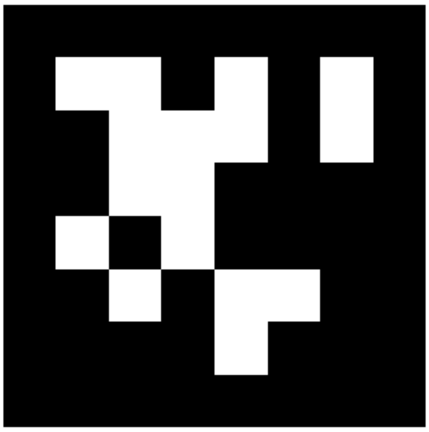
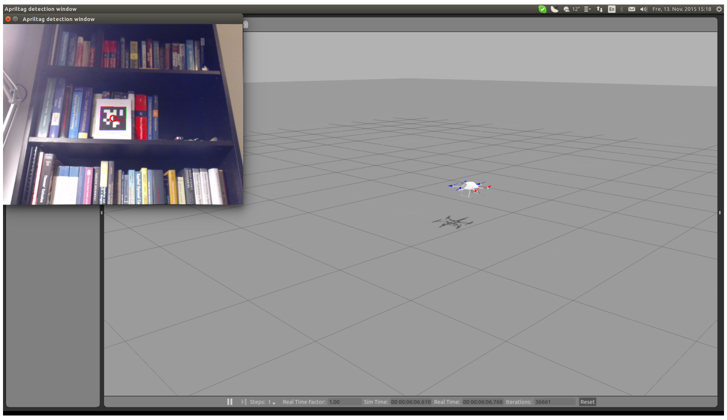


Figure 2.1: Ein Bild



Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:06:06.610 Real Time: 00 00:06:06.766 Iterations: 36661

Figure 2.2: Ein Bild

and the simulator's window with the moving MAV. In figure 2.3 are depicted all the ROS running nodes.

For safety reasons, the position of the MAV in the z axis is augmented by 50 cm. Thus, when the user holds the marker at the same height as the camera, the MAV does not collide with the floor.

2.3 Creating the Apriltag Models

In order to make the simulation realistic, the MAV should detect the Apriltags by its own camera. Thus, models represented the various tags were needed. The objects were simple cuboids which had as texture the Apriltag's image. It should be mentioned that the dimensions (width and height) of the cuboids should be identical as the parameters passed from the ROS parameter server to the detection node, so as to get the correct distance estimation.

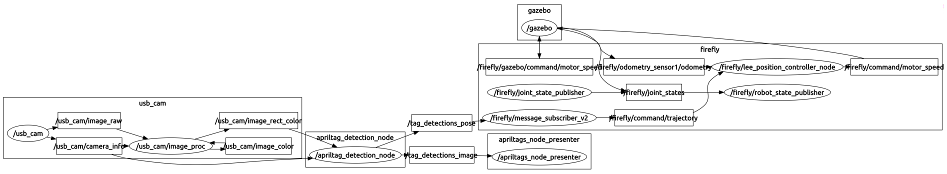

gazebo
/gazebo
firefly
/firefly/gazebo/command/motor_speed
/firefly/odometry_sensor1/odometry
/firefly/lee_position_controller_node
/firefly/command/motor_speed
/firefly/joint_state_publisher
/firefly/joint_states
/firefly/robot_state_publisher
/firefly/message_subscriber_v2
/firefly/command/trajectory
usb_cam
/usb_cam
/usb_cam/image_raw
/usb_cam/camera_info
/usb_cam/image_proc
/usb_cam/image_rect_color
/usb_cam/image_color
apriltag_detection_node
/apriltag_detection_node
/tag_detections_pose
/tag_detections_image
apriltags_node_presenter
/apriltags_node_presenter

Figure 2.3: Ein Bild

2.4 Realistic Simulation Scenarios

Before moving to the real system, were implemented some more realistic simulation scenarios so as to study the MAV's behavior under various circumstances. In this case the user inserts different markers inside the simulated world and the MAV performs a variety of motions with respect to the different markers it detects from its own camera. In order to perform different motions the MAV detects the id of each tag, and executes the predefined motions. It should be mentioned, that at each time only one Apriltag should be detected from the MAV's sensor so as to have a smooth operation.

The default motion, consists of the user moving a marker in front of the computer's camera which controls the pose of the Apriltag in the simulation. The MAV is programmed to follow the marker, by having adequate distance so as to guarantee the user's safety in the real world system. The predefined safety distance is set by the user at the launch of the simulation, in the case that the user enters a values less than a predefined threshold, the program automatically sets the input value to the threshold. When the user rotates the marker in the z axis (yaw) by ψ degrees the MAV performs a circle-like motion with respect to the marker's position.

Furthermore, the MAV is capable of performing take-offs and landings whenever the user wants. When the user wants to land the MAV, he just has to present a specific marker in front of the camera. The exact same procedure, but with a different marker, is followed for making the MAV hovering again. The program stores the last x,y coordinates of The MAV before the landing command and automatically sets the desired height at 1m.

//Depending on the implementation //Right now square

Bibliography

[1] Open Source Robotics Foundation, Gazebo Simulator http://gazebosim.org/.

[2] B. Stourstrup, *The C++ Programming Language*, 4th ed. Boston, MA: Addison-Wesley, 2013.

[3] Online code repository, https://github.com/ethz-asl/mav_demos.

[4] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).* IEEE, May 2011, pp. 3400–3407.

[5] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors," *Int. J. Rob. Res.*, vol. 31, no. 5, pp. 664–674, Apr. 2012.

[6] M. Faessler, F. Fontana, C. Forster, and D. Scaramuzza, "Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 1722–1729.

[7] E. Mueggler, B. Huber, and D. Scaramuzza, "Event-based, 6-dof pose tracking for high-speed maneuvers," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sept 2014, pp. 2761–2768.

[8] D. Scaramuzza, M. Achtelik, L. Doitsidis *et al.*, "Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments," *Robotics Automation Magazine, IEEE*, vol. 21, no. 3, pp. 26–40, Sept 2014.

[9] S. Weiss, M. Achtelik, M. Chli, and R. Siegwart, "Versatile distributed pose estimation and sensor self-calibration for an autonomous mav," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 31–38.

[10] G. Ducard and R. D'Andrea, "Autonomous quadrotor flight using a vision system and accommodating frames misalignment," in *Industrial embedded systems, 2009. SIES'09. IEEE international symposium on.* IEEE, 2009, pp. 261–264.

[11] A. Schollig, F. Augugliaro, S. Lupashin, and R. D'Andrea, "Synchronizing the motion of a quadrocopter to music," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* IEEE, 2010, pp. 3355–3360.

[12] Open Source Robotics Foundation, http://www.ros.org/.

[13] Open source robotics foundation, http://wiki.ros.org/apriltags_ros.

[14] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.