ETH
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

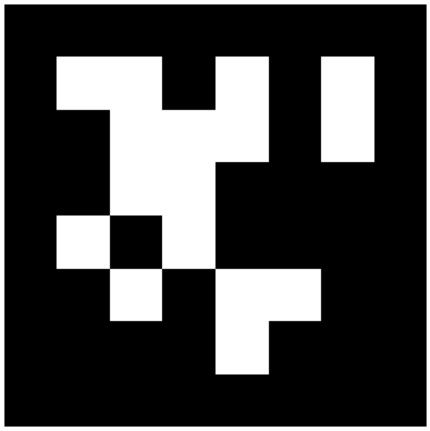

Figure 2.1: Ein Bild

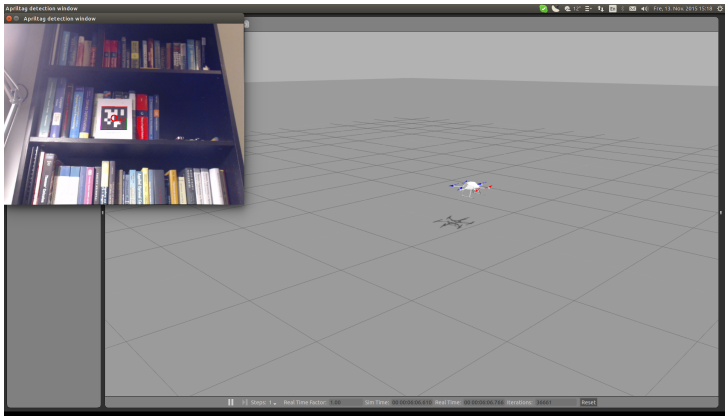

Figure 2.2: Ein Bild

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.

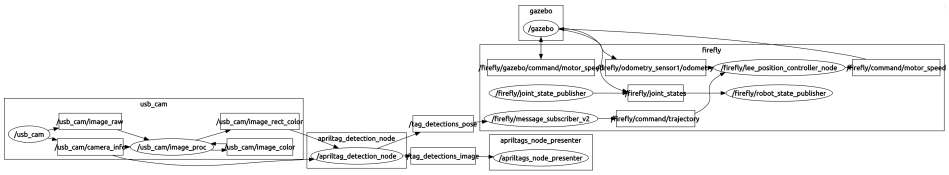

Figure 2.3: Ein Bild

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

Chapter 3

Coordinate Frames

A very crucial part of this project, was the analysis of the MAV's coordinate systems. This is very important because the MAV takes its desired position from what it detects from its camera and has to transform it to the IMU frame with respect to the world frame. Furthermore, the apriltag's position in the simulated world is defined by it's pose as detected from the usb camera. For the transformations the ASL's minkindr library was used. It should be noted, that the color coding in the axes frames follows the RVIZ color convention, meaning blue defines the z axis, green the y axis and red the x axis respectively. Furthermore the ASL's frame naming convention [**?**] is followed throughout this project.

3.1 Coordinate Frames in the Simulation

The first thing that should be resolved in the simulation world, was the correct positioning of the apriltag with respect to the real camera detection. Although the two frames share the same origin point, the have different orientations as seen by figure 3.1, the distance between the two frames is set to emphasize the rotation. As it can easily be seen the transformation form camera frame to the world frame consists of one rotation of 90° around the world's y axis ($rot_{y_{world}}(90°)$) and a successive rotation of $-90°$ around the current z axis ($rot_z(-90°)$) as presented in the equation 3.1. With the aforementioned transformation, the coordinates as seen from the camera match the actual world coordinates. Furthermore, it is easily understood that the apriltag's yaw in the real world corresponds to the negative pitch of the detected image.

$$\begin{bmatrix} x_{wc} \\ y_{wc} \\ z_{wc} \end{bmatrix} = Rot_{y_{world}}(90°)Rot_z(-90°) \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \quad (3.1)$$

After the transformation of the camera frame, now come the coordinate transformations regarding the MAV. The Firefly's reference body frame coincides with its IMU frame. The real system sends the world's coordinates with respect to the aforementioned frame to the position controller so as to define its position. The camera frame as can be seen from the figure 3.2 is rotated and titled downwards relative to the IMU frame. The transformation between the inertial system and the MAV's body frame is provided by subscribing to the Firefly's odometry sensor, named as odometry_sensor_1, the obtained transformation is represented by T_{WB}. Then, the transform from the body frame to the camera link frame is obtained, represented as T_{BC}. So the final transformation from the camera to the world frame, noted as

Figure 3.1: The camera frame and the world frame.

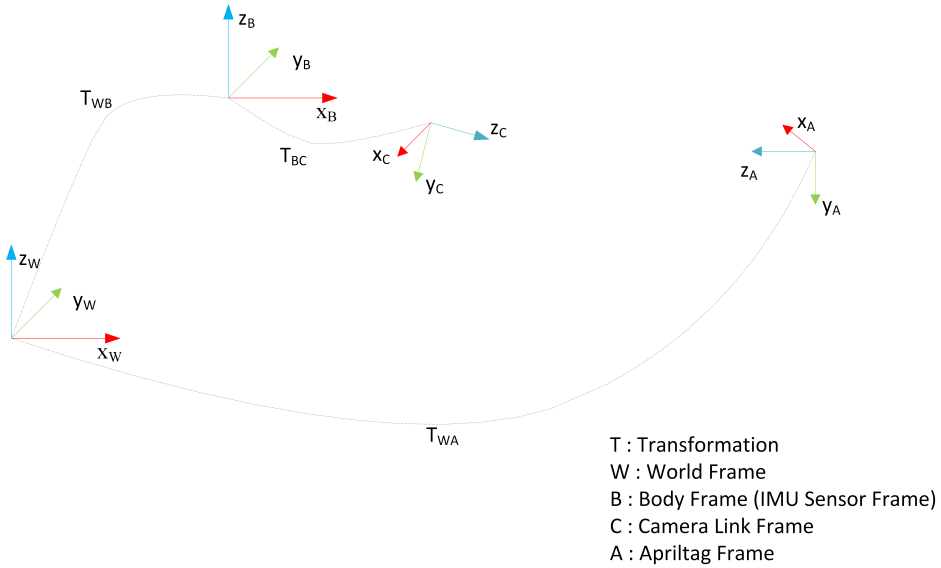

Figure 3.2: The camera frame and the world frame.

T_{WC} is given by equation 3.2. In order to simplify the hole process in simulation and since the error that is inserted is small, the coordinates of the camera with respect to the world are sent to the position controller.

$$T_{WC} = T_{WB} * T_{BC} \tag{3.2}$$

3.2 Coordinate Frames in the Real System

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.