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[2].

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
ASL	Autonomous Systems Lab
IMU	Inertial Measurement Unit
UAV	Unmanned Aerial Vehicle
MAV	Micro Aerial Vehicle
ROS	Robot Operating System

Chapter 1

Introduction

During the past decades, robotics has evolved from the heavy industrial manipulators, that worked only to a very specific structured environment, to mobile robots that navigate themselves to various unstructured environments. A method that has gained a lot of attention lately, is localization and motion of the robot based only on information coming from cameras.
The reason for choosing cameras as the main source of information in modern robotics, and as in our specific example to an MAV, is that they provide an abundance of information while they weight less, their power consumption is low compared to other kinds of sensors and their prices keep reducing. Furthermore, nowadays exist advanced computer vision algorithms that allow us to robustly extract the necessary information and of course the proper hardware that allows the on board and real-time processing of the acquired data.
Furthermore, the scientific interest regarding autonomous mobile robots, is strongly attracted to low weight multi rotor Micro Aerial Vehicles (MAVs). Some of their merits over conventional designs such as fixed wing platforms are their ability for vertical take off and landing, they can hover over specific areas as long as their energy source allows and their ability to perform tight maneuvers in confined spaced. Their wide commercial availability combined with their prices which are considerably reduced compared with a few years ago make them a very attractive platform for experimentation. Their uses, which are rapidly increased, include parcel delivery, inspection of structures, search and rescue operations and several more.
In this project, the AscTec's Firefly MAV is used, to present the realization of the MAV's high level control only with the use of apriltag markers. Thus, the MAV uses only the data available from its onboard sensors to deduce what it must do based on the user's visual commands. Various motion scenaria were implemented. First the whole concept was build in simulation, using the RotorS simulator created by the ASL [3] and then it was implemented in the real system.

1.1 Summary

1.2 Related Work

Many of the teams that work with MAVs use the VICON system to calculate the pose of their MAV, thus to localize it, and based on that to perform various motions, such examples are [4], [5]. The VICON system gives the advantage of very precise position knowledge, but with the price that it can only be used structured environments that have multiple VICON cameras installed and also IR reflective markers on the MAVs, furthermore it is still expensive. Other approaches involve

pose estimation and navigation data taken from onboard cameras or dynamic vision sensors such as [6], [7]. Apriltags, were used by relatively few other teams in order to provide robust pose estimates to the MAVs either in the case of landing [16], [15] or in case of a pair of MAVs to provide visual pose estimate in case of a sensor failure [14]. In our case, we used different Apriltags detected from the MAV's camera to give it various commands regarding its pose.

Chapter 2

Description of the 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[8] and can be accessed at the ASL's GitHub page under the repository mav_demos[9].

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[2] 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[10]. 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

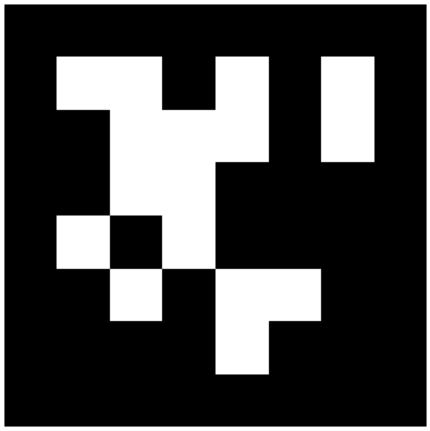

Figure 2.1: Ein Bild

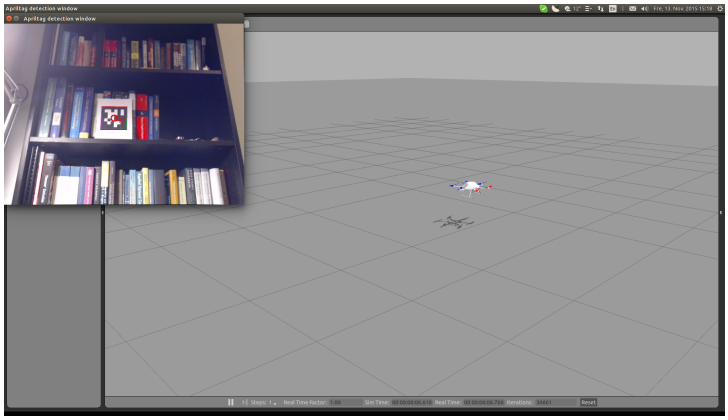

Figure 2.2: Ein Bild

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.

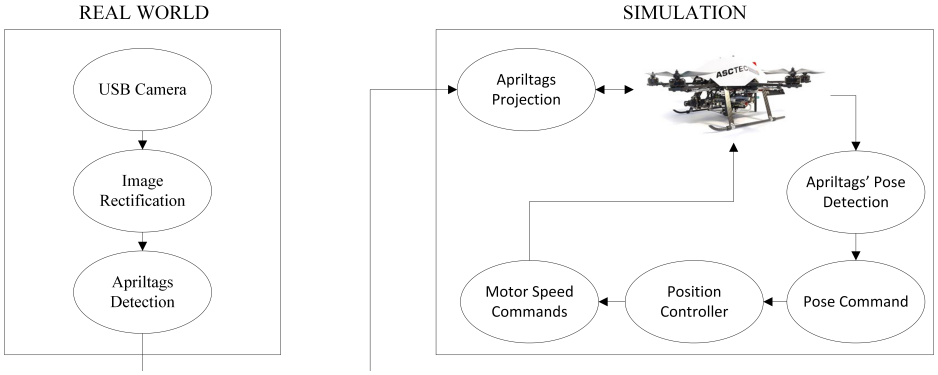

Figure 2.3: Ein Bild

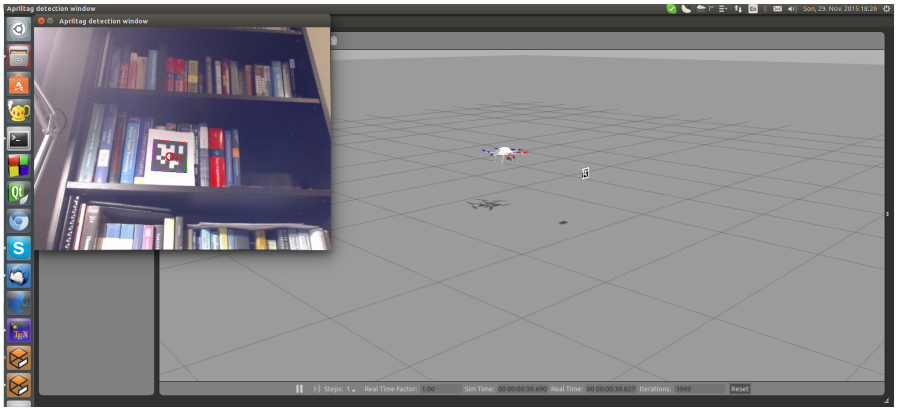

Figure 2.4: Ein Bild

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 as presented in image 2.4. 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 and z coordinates of The MAV before the landing command. It should be

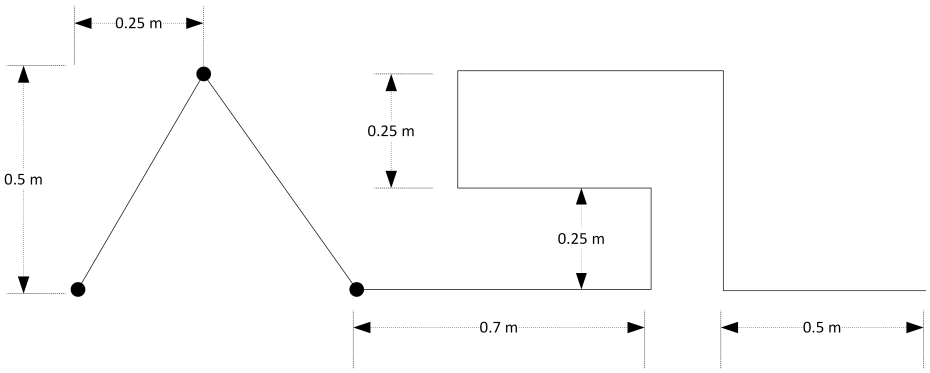

Figure 2.5: ASL logo

mentioned that in order to switch from one kind of motion to another the user must show to the MAV a specific marker that triggers a reset state. In this reset state, the MAV goes back to its starting position and height, just as before it started each motion. It should be mentioned, that throughout the simulation the MAV's inertial measurement frame is assumed to be initialized relative to the ground.
In addition to the aforementioned motion patterns, the MAV currently performs two more motions. First it is programmed to create a square in the air. Taking as initial position the current coordinates of the MAV, it computes the other vertices so as the length of each side of the square to be 1m long. The MAV will continue to implement the motion until the user presents the it the marker that triggers the reset state. The second motion, performs the ASL's logo in the air 2.5. Again starting from its initial position the Firefly computes the vertices so as to formulate the logo in the air, upon completion the MAV returns to its initial position.

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 the controller's reference 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 [11] 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 usb 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 y axis ($rot_y(90°)$) and a successive rotation of $-90°$ around the 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.

$$\begin{bmatrix} x_{wc} \\ y_{wc} \\ z_{wc} \end{bmatrix} = Rot_y(90°)Rot_z(-90°) \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} \tag{3.1}$$

After the transformation of the camera frame, now come the coordinate transformations regarding the MAV. The transformation between the MAV's body frame and the world frame is obtained by subscribing to the firefly's odometry sensor, in the simulation it is named as odometry_sensor1, the obtained transformation is represented by T_{W_B}. Then the static transform between the body frame and the onboard camera frame is provided as parameter by the configuration file and is represented as T_{B_C}. The aforementioned transform is passed as parameter in order to reduce the calculations needed to be perform and thus, speed up the procedure. The transformation between the MAV's camera and the Apriltag is obtained by subscribing to the MAV's tag_detection_pose topic. This provides the pose of the detected Apriltag with respect to the camera coordinate frame. Now the last part was to obtain the transformation between the Apriltag frame and the reference

Figure 3.1: The camera frame and the world frame.

frame. As it can be seen from the 3.2 the reference frame has the same orientation as the world system and is translated from it by the offsets specified by the user in the parameters' file. The transformation between the apriltag frame with respect to the reference frame can be seen from the image 3.2, while the rotation is composed by a rotation around the y axis by $-90°$ and a rotation around the z axis by $90°$.The offsets, and especially the offset in the x axis since this defines the straight distance between the MAV and the Tag, are added so as to ensure that the MAV will keep a safety distance between itself and the Apriltag. If the x offset is set to zero, then the reference frame coincides with the Tag's frame, and they will collide, since the reference position of the MAV is set to the exact equal coordinates of the Tag. The aforementioned are presented in equation 3.2. The final coordinates that are sent to the controller are the result of the transformation presented in equation 3.3 and they express the reference frame coordinates with respect to the world frame.

$$T_{R\_A} = Rot_y(-90°) * Rot_z(90°) * \begin{bmatrix} x_A \\ y_A \\ z_A \end{bmatrix} + \begin{bmatrix} offset\_x \\ offset\_y \\ offset\_z \end{bmatrix} \tag{3.2}$$

$$T_{W\_R} = T_{W\_B} * T_{B\_C} * T_{C\_A} * {T_{R\_A}}^{-1} \tag{3.3}$$

3.2 Coordinate Frames in the Real System

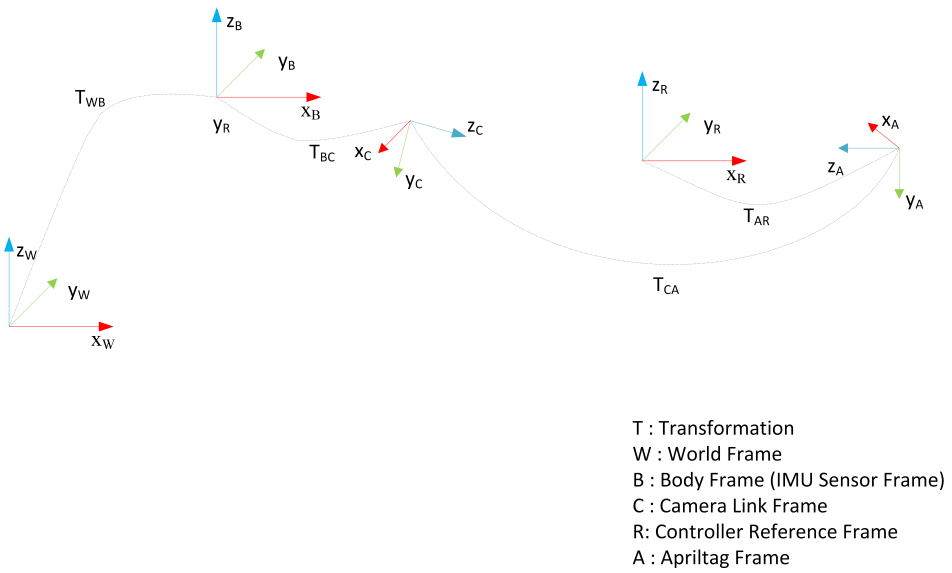

Figure 3.2: The camera frame and the world frame.

Bibliography

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

[2] 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.

[3] Open Source Robotics Foundation, RotorS Simulator http://gazebosim.org/.

[4] 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.

[5] 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.

[6] 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.

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

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

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

[11] Online repository, https://github.com/ethz-asl/minkindr/wiki/Common-Transformation-Conventions.

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

[13] 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.

[14] R. W. P. Hoogervorst, "Collaborative position control of an uav using vision and imu data," University of Twente, BSc report 017RaM2015, Jul. 2015.

[15] K. Ling, "Precision landing of a quadrotor uav on a moving target using low-cost sensors," University of Waterloo, Msc Thesis, Sep. 2014.

[16] S. M. Chaves, R. W. Wolcott, and R. M. Eustice, "NEEC research: Toward GPS-denied landing of unmanned aerial vehicles on ships at sea," *Naval Engineers Journal*, vol. 127, no. 1, pp. 23–35, 2015.