P&O EAGLE: Image processing module

Xu Jia, Tinne Tuytelaars

June 2016

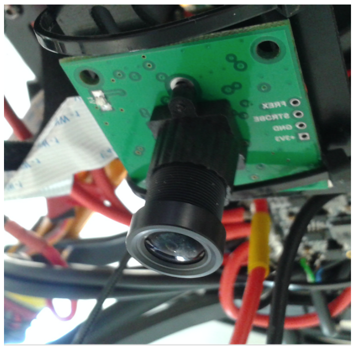

Figure 1: The downward looking camera mounted on the EAGLE drone serves as its main sensor for autonomous localization and navigation.

Contents

1 Introduction

Cameras are a popular sensor for drones: they are relatively cheap and lightweight, and convey a lot of information that can also easily be interpreted by humans. The camera mounted on the EAGLE drone is a downward looking one: it's mounted beneath the drone base looking down to the ground floor. When the drone is hoovering, the camera's optical axis is (roughly) aligned with the direction of the gravitational field and (roughly) perpendicular to the ground plane.

In the context of EAGLE, the camera will be the main sensor for performing the mission. It will be used both for localization of the drone (with respect to a grid of red lines on the ground floor), as well as for the detection and decoding of QR codes containing instructions for the drone.

2 Task

The work can be divided in two subtasks: localization and QR detection. Algorithms for performing these tasks will need to be designed, implemented and tested. Given the limited resources on the drone, it is also important that the implementation is efficient. This holds especially for the localization, as it needs to be integrated with the outer control loop for navigating the drone and therefore it is important that it runs in real-time.

The image processing module is critical for the EAGLE mission. Without it, autonomous flight will be impossible, as the drone will have no idea where it is, and no idea where it needs to go.

2.1 Localization with respect to red line grid

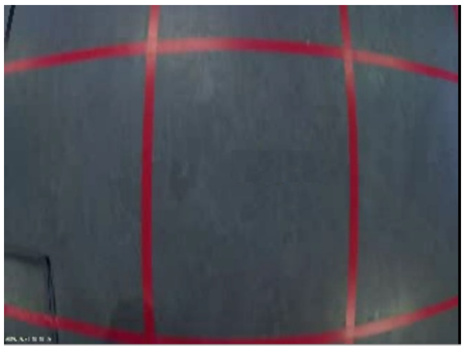

Figure 2: A rectangular grid of red lines is drawn on the ground floor, to be used by the drone to localize itself.

First, the drone needs to be aware of its location. To simplify this task, we put a rectangular grid of red lines on the ground floor (see figure 2). The localization task is significantly simplified if the team can agree to always keep the drone aligned with the main axes of the grid (i.e., such that the grid appears as horizontal and vertical lines in the image). Localization is only needed in 'autonomous' flying mode. When the drone is controlled by a pilot via the remote control, no image processing is needed. Switching to autonomous mode should only be done when the drone is hovering in a stable manner above one cell of the grid and aligned with the grid. This cell will then act as the origin (0,0) of a local coordinate axes frame. By keeping track of horizontal and vertical red line crossings, the drone should be able to always localize itself with respect to this local coordinate axes frame. Figure 2 shows an example of a frame captured by the camera during autonomous flight operation.

2.2 QR detection and decoding

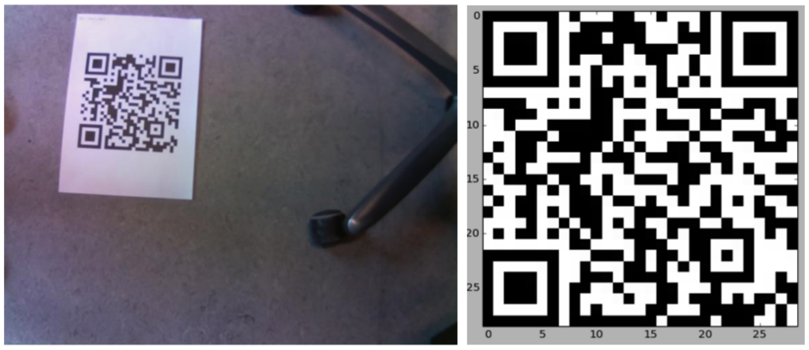

Figure 3: A QR code in the field of view of the camera (left) is first detected, and then the corresponding bitmap is extracted (right).

Most cells on the grid will be empty (like the one shown in Figure 2). However, some of them contain a QR code, as shown in the left part of Figure 6. The second task for the image processing module is to detect a QR code in a frame, and to extract it as a bitmap (as shown on the right of Figure 6). The decoded information contains navigation commands about where to go next - but it's still encrypted; in the cryptography module this will need to be deciphered further.

Apart from decoding the navigation commands, the QR codes could also be used to refine or correct the localization of the drone. It's up to the students to decide if and how they want to do this.

3 Material provided for the students

- Introduction to OpenCV library
- Introduction on image formation and camera calibration
- Description of QR code detection

- Webcam/Raspberry Pi
- QR code layout and example images
- Recorded images and videos with red lines in it
- Camera calibration toolbox and checkerboard image
- QR detection library (only for smaller teams)

4 Different sub-tasks

First semester

1. Material reading	2 students / 1 week
2. Familiarize with OpenCV library	2 students / 1 week
3. Localization	2 students / 5 weeks
4. Real-time localization on drone	2 students / 2 weeks

Second semester

1. QR detection	2 students / 6 weeks
2. Integration with other modules	2 students / 6 weeks

5 Milestones

- ”Understand and plan” phase:
  1. Understand the role of the image processing module in the project
  2. Familiarize with OpenCV library
  3. Algorithm design for localization → SOFT T1
- Module phase
  1. Red line detection demonstrated → HARD T2
  2. Localization demonstrated → HARD T3
  3. QR code detection demonstrated → HARD T4
- Integration phase
  1. Integration with navigation control module → HARD T4
  2. Integration with QR decryption module → SOFT T5
  3. Solve reliability issues → HARD T6

6 Sub-tasks: detail

- *Material reading:* The first task consists of getting familiar with the task and the role of the image processing module in the overall project. Read the documentation we provide. This consists of this document, as well as the documentation on the image formation process and camera models, needed to understand the relationship between image coordinates, camera coordinates and world coordinates.

- *Familiarize with the OpenCV library:* OpenCV is a library for image processing, that will be very useful for implementing the localization and QR detection tasks. It can work both with python and C++. In the *OpenCV* folder, we provide a tutorial with example programs containing several basic functions like I/O and visualization. Go over these examples step-by-step to get familiar with the library and basic operations.

- *Algorithm design for localization:* Design an algorithm that reads in frames of a streaming video and outputs the location of the drone, relative to a coordinate axes frame aligned with the grid of red lines. The origin of the coordinate axes frame is at the center of the cell visible in the middle of the first frame of the video. Keep efficiency in mind when designing the algorithm: this module will need to run in real-time, providing input for the navigation control.

- *Camera calibration:* You may have noticed that the lines in Figure 2 are not straight, but rather curved. This is due to radial distortion, as explained in the document on image formation and camera models. This effect can be compensated for by calibrating the camera (i.e. determine the internal camera parameters). Once these parameters are known, a frame can be 'dewarped' making the lines in the image straight again (see Figure 4 for an example). In the folder *Camera Calibration* we provide a toolbox that allows to calibrate the camera and compensate for the radial distortion. This step, however, is optional: teams can choose to use it, or make sure that the red line detection can cope with the curved lines in some other way (a decision taken as part of the algorithm design).

- *Red line detection:* A first step in the localization module consists of detecting the red lines in the input image. Implement a filter that selects all red pixels in the image, turning the original RGB image into a binary image with '1' values for (most of) the red line pixels and '0' everywhere else (see Figure 5, left). Next, write code that localizes the horizontal and vertical lines in the binary image and displays the result on screen (see Figure 5, right). To pass the milestone at T2, all the red lines in the images in the folder *Red Line Detection* should be detected correctly, and real-time detection of red lines should be demonstrated by overlaying the detected lines on the images on a PC screen.

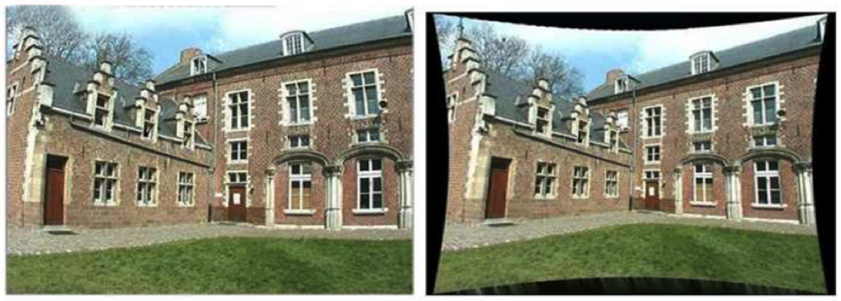

Figure 4: Left: An image exhibiting radial distortion. The vertical wall at the left of the building appears bent in the image and the gutter on the frontal wall on the right appears curved too. Right: The same image after removal of the radial distortion. Straight lines in the scene now appear as straight lines in the image as well.

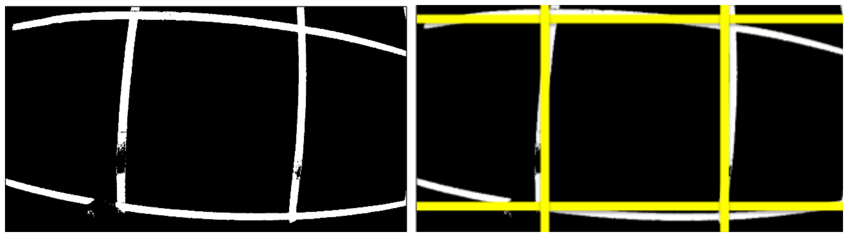

Figure 5: Different steps in the red line detection algorithm: segmentation (left) and final output (right).

- *Localization:* Implement, debug and test the full localization algorithm on PC, before porting it to the Zybo board. Timing the different steps may be a good idea. Optimize the code where needed. A couple of example videos are provided in the folder *Localization.* To pass the milestone at T3, the code should be able to process each of these videos correctly, i.e. generate as output the coordinates of the drone for each frame. Also, localization should be demonstrated by displaying the coordinates in real-time on a PC screen, while walking with the camera over the 2D line grid.

- *QR code detection:* Read the documentation on QR codes pattern layout and QR code detection, provided in the folder *QR detection.* Implement, debug and test the different steps in QR detection: 1. detection of the QR code identification markers; 2. determining the three distinct markers; 3. localizing the fourth corner; 4. orientation and perspective correction; and 5. binarization. Port the code to the ZyBo board and time it. To pass the milestone at T4, the code should be able to decode all the provided example images correctly. Also a demo, decoding QR codes held in front of a camera, should be presented. Smaller teams will be given a library for QR-detection, instead of having to implement everything themselves.

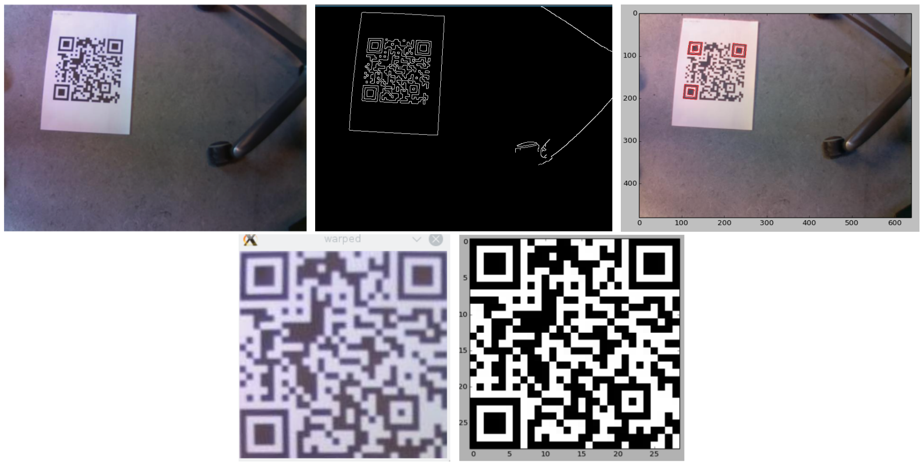

Figure 6: Different steps in the QR code detection algorithm.

- *Integration with other modules:* So far, the localization algorithm just wrote the coordinates to file or displayed them on screen. For integration with the navigation control, the interface and communication will need to be defined and implemented. Likewise, the binary image generated by the QR detection module will need to be integrated with the decoding and decryption done by the cryptography module.
- *Solve reliability issues:* Finally, additional image processing building blocks or refinements may be needed, depending on the overall design of your team's system and tests thereof. This may include tracking corners for more accurate localization, or determining the orientation of the lines to compensate for misalignment of the drone with the grid.

7 IP explanation

- *Introduction to OpenCV library:* This is an online tutorial, with example programs that gradually make you familiar with the OpenCV library. For instance, how image data is stored in OpenCV and several basic functions like I/O and visualization.
- *Introduction on image formation and camera calibration:* This is a document explaining basic camera models. Going over this document, you will learn the relation between image coordinates, camera

coordinates and world coordinates. You will also learn about radial distortion, i.e. why the lines sometimes appear curved in the image.

- *Description of QR detection:* This document describes the structure of QR codes. Additionally, to help you with the QR detection, we provide you with a high-level algorithm design. It describes the different steps that need to be implemented for QR detection.
- *Webcam/Raspberry Pi:* A camera is provided to record images. This camera is connected to a Raspberry Pi, which provides both compressed as well as uncompressed video output. For image processing, it's easiest to work on the uncompressed video stream. The compressed one is used by the wireless link module.
- *Recorded images and videos with red lines in it:* A set of images and videos are provided for students to test their red line detection and localization algorithms.
- *QR code example images:* A set of images with QR codes are provided to test the QR detection algorithm.
- *Camera calibration toolbox and checkerboard image:* We provide a toolbox for camera calibration, which allows to compensate for the effect of radial distortion. This process involves recording several images of a checkerboard. The checkerboard image is also provided.

8 Documentation

- Introduction to OpenCV: `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_tutorials.html`
- QR code layout: `http://qrcode.meetheed.com/question14.php` `http://dsynflo.blogspot.be/2014/10/opencv-qr-code-detection-and-extraction.html`
- Camera Calibration: `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration`