Feature tracking using vision on an autonomous airplane

van der Molen, K.

Published: 01/01/2004

Document Version
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the author's version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

Citation for published version (APA):

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 13. Jan. 2019
Feature Tracking using Vision on an Autonomous Airplane

K. v.d. Molen

Report No. DCT 2004.48

Traineeship report

Coaches: S. Bouabdallah and S. Michaud
Advisor: prof.dr.ir. R. Siegwart
Supervisor: prof.dr.ir. M. Steinbuch

Eindhoven University of Technology (TU/e)
Department of Mechanical Engineering
Control Systems Technology Group

Eindhoven, 16th April 2004
Contents

1 Introduction 5

2 The setup 7
  2.1 Equipment 7
    2.1.1 Autopilot 8
    2.1.2 AGL 9
    2.1.3 Servo board 9
    2.1.4 Batteries 9
    2.1.5 Wireless serial link 10
    2.1.6 RC transmitter and receiver 10
    2.1.7 Video Equipment 10
  2.2 Synchronization 12

3 Feature tracking 15
  3.1 Methods 15
    3.1.1 Recognition 15
    3.1.2 Line extraction 16
    3.1.3 Point Tracking 16
  3.2 Pyramidal Lucas-Kanade point tracking 16
    3.2.1 Implementation 17

4 Target definition 19
  4.1 Camera calibration 19
    4.1.1 The calibration 20
    4.1.2 Implementation 22
    4.1.3 Resolution 23
  4.2 The Relative Position 23
    4.2.1 Offline test 25
    4.2.2 Online test 26

5 The Autopilot 29
  5.1 Introduction to the Autopilot 29
    5.1.1 Feedback controllers 29
    5.1.2 High-level commands 29
Chapter 1

Introduction

There is great interest in air-borne exploration in different fields. For example traffic control, planetary exploration and search and rescue. Also there exist areas on Earth or for example on the planet Mars that are difficult accessible over ground, but can easily be reached by flying robots.

The Autonomous Systems Lab (ASL) in Lausanne aims to built an autonomous, solar-powered, model airplane that can fly continuously, first on Earth, then on Mars. This large-scale project, the StarTiger III project, is subdivided into smaller projects. One subproject is to realize a 24-hour flight on earth.

The project that is treated in this report is a first study to autonomous flight. It aims to control a model airplane with autonomous capabilities using onboard vision. A target can be defined after which the airplane will automatically find it's way towards the target.

Objectives

The main objectives of the project are listed below.

- Excluding the use of feedback from the current used GPS and replace the functionality by a camera.
- Extracting the relative position of the target to design a controller that navigates the airplane to, and flies around the target.
- Creating an application to track the target and send control commands back to the autopilot.

In chapter two first the equipment used for this project is presented. Chapter five will describe the autopilot in more detail. Chapter three shortly discusses the available image-processing techniques. Next, chapter four describes the extraction of the relative position. This relative position is used in chapter six to control the airplane, followed by the test results in chapter seven. Finally the conclusions are drawn in chapter eight.
Chapter 2

The setup

As an introduction, this chapter lists and shortly discusses all the equipment used for the project. References of the products as type-numbers are given in appendix C. Table 2.1 summarizes the dimensions and power usage for the most important products, and figure 2.1 shows the hardware setup onboard the plane. Here the arrows display the data traffic to and from the various elements. Printed Circuit Boards (PCB’s) are presented by rectangles, communication devices by rounded rectangles and the three batteries are named inside the ellipses.

The latter section of this chapter discusses the subject of synchronizing the two sources of data.

2.1 Equipment

A ready-to-fly autonomous model airplane was ordered at the Canadian company MicroPilot. This purchase included a module which is capable of controlling a medium-sized model airplane. The module came fitted in the fuselage of the plane, together with a wireless modem. The control module

<table>
<thead>
<tr>
<th>Name</th>
<th>Width</th>
<th>Length</th>
<th>Height</th>
<th>Weight</th>
<th>Voltage</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autopilot</td>
<td>40</td>
<td>100</td>
<td>15</td>
<td>28</td>
<td>6</td>
<td>140</td>
</tr>
<tr>
<td>Ultrasonic sensor</td>
<td>40</td>
<td>70</td>
<td>23</td>
<td>28</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>Servo board</td>
<td>31</td>
<td>48</td>
<td>13</td>
<td>14</td>
<td>5</td>
<td>-</td>
</tr>
<tr>
<td>GPS antenna</td>
<td>25</td>
<td>34</td>
<td>10</td>
<td>32</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Camera</td>
<td>32</td>
<td>32</td>
<td>28</td>
<td>30</td>
<td>12</td>
<td>90</td>
</tr>
<tr>
<td>Video overlay card</td>
<td>27</td>
<td>89</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>70</td>
</tr>
<tr>
<td>Video transmitter</td>
<td>24</td>
<td>50</td>
<td>8</td>
<td>38</td>
<td>5</td>
<td>450</td>
</tr>
<tr>
<td>RC Receiver</td>
<td>24</td>
<td>46</td>
<td>18</td>
<td>18</td>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>
CHAPTER 2. THE SETUP

includes a GPS\(^1\) receiver and the antenna for it was mounted in the tail of the plane. Also the antenna for the wireless modem was mounted in the tail. Further the plane was equipped with an ultrasonic height sensor called the 'AGL'.

The following sections describe the elements needed for the autonomous flight. The products that are needed for the video stream to be available on the ground PC include a camera, a transmitter for the camera, a video overlay card and a frame grabber. These can be found in the section 'Video Equipment'.

2.1.1 Autopilot

The autopilot is a very small, lightweight PCB, optimized for usage in model airplanes. The main functions and features are listed below.

- Stabilizing an aircraft
- 24 controllable servos with user-defined PID feedback loops. At this moment 7 servos are controlled using the factory-PID settings
- 3 flying modes
  - UAV (Unmanned Aerial Vehicle). Executes commands in the flight-plan
  - RPV (Remote Piloted Vehicle). Stabilizes the plane and maintains current heading and altitude
  - PIC (Pilot in Command). Manual mode
- Onboard sensors
  - GPS

\(^1\)Global Positioning System.
2.1. **EQUIPMENT**

- 3-axis gyroscope
- 3-axis acceleration meter
- 2 pressure sensors for airspeed and altitude
  - 16 MHz, 2k RAM Motorola Processor
  - Recording of up to one hour of data during flight

**Discussion** The MP2028 is the follow-up of the larger MP2000, which has no integrated GPS receiver and is considerably larger. The weight and size of the MP2028 are very low for the functions and sensors that are included.

2.1.2 **AGL**

The AGL is an ultrasonic height sensor. The sensor itself is located underneath the wing and has a range of about 8 meters. The data from the sensor is preprocessed on the onboard PCB before it is read by the autopilot. The frequency of the sensor-update is about twice per second. The autopilot uses the AGL only for autonomous takeoff and landing. Since only manual takeoffs and landings are planned, the PCB was removed to save space and weight.

**Discussion** The PCB that goes with the AGL inside the fuselage is quite large with respect to its function in comparison with the autopilot. As the autopilot has been optimized for size and functionality, the AGL is rather old. No tests have been done with the AGL.

2.1.3 **Servo board**

The servo board is a PCB made by MicroPilot wherewith the servos are connected. In total eight servos can be connected per board. The board receives position-signals from the autopilot and translates them to PWM-signals, which are used to control the servos. The PCB has to be powered since the autopilot does not provide the current needed to run the servos.

**Discussion** Also the servo board is rather large and not optimized for size and weight compared with the autopilot.

2.1.4 **Batteries**

Three onboard battery packs provide power for the elements. The autopilot uses a separate pack of 6V, and also powers the serial modem and the AGL. The servos are separately powered by a pack of 4 batteries, 4.8V.
### Table 2.2: The onboard batteries and their dimension

<table>
<thead>
<tr>
<th>Battery</th>
<th>Diameter</th>
<th>Height</th>
<th>Weight</th>
<th>Voltage</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autopilot</td>
<td>5*15 mm</td>
<td>50 mm</td>
<td>125 g</td>
<td>6 V</td>
<td>1200 mAh</td>
</tr>
<tr>
<td>Servo</td>
<td>4*15 mm</td>
<td>50 mm</td>
<td>100 g</td>
<td>4.8 V</td>
<td>700 mAh</td>
</tr>
<tr>
<td>GPS</td>
<td>25 mm</td>
<td>8 mm</td>
<td>8 g</td>
<td>3 V</td>
<td>1000 mAh</td>
</tr>
</tbody>
</table>

The third battery is the GPS backup battery. It is needed for the GPS to remember the last settings of the satellites, stored in the so-called almanac file. When this battery is disconnected, the GPS takes over 20 minutes to lock. Table 2.2 lists the batteries and specifications onboard the plane.

#### 2.1.5 Wireless serial link

A small wireless modem is connected to the autopilot. It provides a 900MHz serial link with the ground control station. The modem, manufactured by MaxStream, is connected to the external antenna located in the tail of the plane.

The transceiver on the ground is connected to the serial port of the PC and can be powered by an adapter or a 9V battery.

#### 2.1.6 RC transmitter and receiver

The RC transmitter that was included in the package from MicroPilot is standard model. 4 channels are controlled by two pokes. A fifth channel switches between modes of the autopilot, automatic or manual control. This channel is controlled by a switch located on top of the transmitter.

#### 2.1.7 Video Equipment

The autopilot can control a video overlay card. The video signal from camera is first lead in the overlay card and from there transmitted using the video transmitter. Next the three products are discussed that establish the video and data retrieval.

**Overlay card**

The BOB III is the video overlay card that can plot data over a regular video signal. This card is compatible with the autopilot and can plot values of internal autopilot fields on the video image. The necessity of the BOB III is be discussed in section 2.2. Figure 2.2 displays an example of a part of the image obtained from the camera with overlayed the values of certain fields of the autopilot. The numbers are approximately 9 by 7 pixels.
2.1. EQUIPMENT

Figure 2.2: Top part of the camera image with BOB III overlaid numbers

Discussion The card is very small and highly configurable. However in this case the autopilot controls the card, and so it is not possible to change settings. As will be seen later, the size of the digits plotted on the streaming images are too small for a recognition algorithm.

Camera

The camera is a CCD board camera. Demands for the camera are clearly lightweight, but also a high resolution is preferable and also a wide-angle lens can oversee more landscape. The view-angle of this camera is roughly 90 degrees over the width of the image. Over the height of the image the view-angle is about 75 degrees.

The resolution obtained from the camera is somewhat confusing. The specifications gave an image resolution of 300k pixels; capturing an image using a windows driver results in 442k pixels, and when streaming, the image size is only 320x240 (77k pixels). These contradictions may exist because each time a different driver is used. Also the frame grabber and the camera may not be compatible on the various resolutions.

Discussion The camera has a clear image, however the image had a huge distortion. Section 4.1 treats the subject of calibrating the camera. Also the camera becomes quite hot when powered on, which means energy is lost. The camera turned out to be rather heavy, and as the obtained resolution is not higher than a much smaller CMOS camera, replacing it would save weight without loss of image quality.

Video transmission

The video signal is transmitted with a 2.4GHz transmitter. The output power is 600mW, which should be sufficient for several kilometers. Roughly a choice had to be made between 900MHz and 2.4GHz transmitters, varying from 100mW to 5W output power. Of course size and weight are important, however also the quality of the image is important.

The 2.4GHz transmitter is chosen to avoid interference with the 900MHz serial modem, and because the quality of the transmitted video signal is higher than 900MHz transmitter. 600mW output power is quite a lot, but this transmitter is reasonably small compared to other lower power transmitters.
Discussed First tests proved a clear image. Very little noise is added by the transmission compared to an image directly from the camera. It does occasionally seem to have problems with interference, while then the screen is very noisy for a moment. The transmitter is very small for its capabilities.

Frame grabber

The Dazzle frame grabber is an external device connected to the PC with the USB. Windows recognizes the frame grabber as an "imaging device", and can so easily be called by any imaging program. The inputs are audio, video and s-video, where here only the video input is used. The frame grabber influences the size of the image read from the input, however no attempts were made to improve the image size.

2.2 Synchronization

The ground station now has two different sources of information: the video stream and the wireless modem which is communicating with the autopilot. In order to extract the relative position of the target using a point on the image from the camera stream, the exact pitch and roll of the plane at the time the picture was taken needs to be known.

The pitch and roll angles are send by the modem and the image by the video transmitter. Thus the two sources of information need to be synchronized. The ideal situation would be a time stamp overlaid on the video with the time in at least milliseconds. This was presumed possible because the MP2028 can control an overlay card. However the GPS is only updated once a second. Millisecond accuracy is available from the GPS [Michaud], but only once a second. The overlayed pitch and roll information should be accompanied with a time in milliseconds, but this is not the case. It is therefore not possible to accurately synchronize the video and the information from the autopilot with a video overlay card. Another difficulty is the size of the numbers as the time on the image should be recognized by software. Figure 2.3 displays the enlarged number 0, extracted from figure 2.2. The number is clearly influenced by the noise of the transmitter and even more
by the background. It is therefore extremely hard to accurately recognize the numbers using for example the [OCR] recognition software.
The proposed solution, given the circumstances, is to measure the time it takes for the video signal to be present in the program. If also the delay of the values of roll and pitch is known, it can be compensated.
Chapter 3

Feature tracking

To restate the goal of the project, the plane is supposed to be guided by the camera, where the user defines the target. As the user defined the target, the plane must autonomously search its way.

Of course there has been a lot of research in computer vision, including at the ASL. This project does not aim to improve these techniques, rather to find the most suitable method given the circumstances and the desired capabilities of the plane.

This chapter will shortly discuss several techniques that can be used for this project. A choice is made and the implementation of the method is explained in the latter paragraph.

3.1 Methods

In robotics, computer vision is a vivid topic. New methods are rapidly developing and others are improved. As for tracking and recognition several techniques are available. Also there exist several good open-source programs with image processing functions implemented in different programming languages. Examples are the Image Processing Toolbox for Matlab and the Intel openCV toolbox. The latter is an open-source C++ toolbox that includes many image processing functions.

As it was desirable to implement all processing and communication in a C++ application, the openCV libraries [openCV] fit in this approach. Also openCV is freely downloadable from the internet. The following sections shortly describe three main techniques that are suited to define a target on an image. All three methods are available in the openCV libraries.

3.1.1 Recognition

Recognition is a wide topic that tries to recognize objects or figures, mostly by learning. In general the algorithm can also localize the object in an image.
The criteria that define an object can exist of several features. Probably the easiest is color: any area in the picture with a certain color can be seen as the object. Also shape, size, or these in combination with color are features used for recognition [Sonka].

3.1.2 Line extraction

It is possible to extract determined lines from an image. An example is first using an edge detection technique. That extracts a binary image with white points where a line was detected. In edge detection certain high gradients in color are defined as an edge. For example the Sobel method is a well known technique that simply thresholds the gradient of an image. After the edge detection the Hough transform can be applied to look for straight lines in the image. The lines are now known by angle and position from the origin.

3.1.3 Point Tracking

The aim of point tracking in computer vision is to find the similar point on a second image. This image is slightly shifted compared to the first image. The point tracking is more straightforward than the two former techniques; the result is either right or wrong. A disadvantage of this technique is that the tracked points may not leave the image, then the points will not be found again.

3.2 Pyramidal Lucas-Kanade point tracking

In this first feasibility project to attempt to control the plane and guide it using a camera, the point tracking algorithm is well suited and relatively easy to implement. Also the points that are tracked give a deterministic target, in contradiction to the less robust recognition method and the line extraction.

As mentioned, the openCV libraries are used to implement the point tracking. The algorithm that is implemented in the openCV is called 'Pyramidal Lucas-Kanade point tracking'. This algorithm first down-samples both first and second image to different layers, where every layer is half the resolution of the former layer. From the third and highest layer from both images, and thus the layers with the lowest resolution, a vector is extracted that describes the shift between the first and second image. Since the resolution is low the calculations are fast but the resulting vector is not very accurate. However, this vector is used as an initial guess in the layer underneath which has a higher resolution. Continuing this to the original image, the new vector quite precisely describes the shift of the second image. This method is referred to as the 'Pyramidal' method.
3.2. PYRAMIDAL LUCAS-KANADE POINT TRACKING

The Lucas-Kanade method uses the gradients in colors of a certain point on the image to define the point. Using the vector that describes the shift, the search area for a similar point on the second image is highly reduced, and so fast and robust. This technique is in detail described in [Bouguet]. Tests with the function show that points are very robustly tracked when the points are located on an area of the image where big differences in color exist. Even if only two images are used where the area of points moves from one side on the first image to the other side on the second image, the function is still capable of tracking the points. However, if points are located on a 'line', a long edge, they will shift along that line if no other object is crossing the line. Also point located on an area consisting of mainly one color, for example a wall, the points will randomly travel over this area.

3.2.1 Implementation

As the user clicks on the streaming camera image on the screen, a smaller image around the mouse point is extracted which is set to 20 by 20 pixels. In this smaller image 'strong points', points that are well suited to track, are located. The method of finding 'good features to track' was first described in [Shi] and is implemented in the openCV libraries. Usually the method finds around 5 strong points on the 20 by 20 pixel image. These points are tracked and the average position of the points is marked as the target.

To make the system more robust it is extended with an extra feature. As mentioned the points might move over a line or a plane. Also the points might widen as the plane approaches the target. therefore, if a point moves too far away, out of the 20 by 20 pixel bounding box around the target, the points are redefined in the current bounding box. This method is of course not precisely accurate, but it prevents the multiple points to wander over the whole image.
Chapter 4

Target definition

As a point is tracked on the camera image, the image coordinates of that point do clearly not correspond to the actual location of the target. Several steps have to be taken to calculate the position of the target, relative to the camera, using the image coordinates.

The first important step is calibrating the camera. All cameras, especially those with a wide-angle lens, display a distorted image. The method and application of the compensation of this distortion are discussed in the section 'Camera calibration'. Next, assume a flat ground. Suppose the rotations of the camera are known, as well as the height of the camera. Now, using the image coordinates of the undistorted image, the relative location of the target can be calculated. Section 4.2 describes this technique and shows the results.

4.1 Camera calibration

As mentioned, all cameras have a certain distortion. The images don’t display the true perspective projection. The true projection is called the pinhole projection and is graphically displayed in figure 4.3. Since further calculations assume a pinhole projection, the distortion must be compensated.

Calibrating a camera is a well-known problem. A good toolbox is The Camera Calibration Toolbox for Matlab [Calib]. It is a free tool, downloadable from the Internet and easy to use. This calibration toolbox returns the intrinsic values of the camera, in other words the specific parameters of a basic model of cameras in general.

The function mapping a point of a pinhole projection to a point on the distorted image is as follows. Let

\[ P_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix} \]  

(4.1)
be coordinates on the pinhole projection and

\[ P_d = \begin{bmatrix} x_d \\ y_d \end{bmatrix} \]  

(4.2)

be the coordinates on the distorted image. Also define

\[ r^2 = x_n^2 + y_n^2 \]  

(4.3)

Then

\[ P_d = \left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) x_n + \begin{bmatrix} 2k_3 x_n y_n + k_4 \left(r^2 + 2x_n^2\right) \\ 2k_4 x_n y_n + k_3 \left(r^2 + 2y_n^2\right) \end{bmatrix} \]  

(4.4)

is the coordinate of the point on the distorted image [Calib]. Here \( k \) is a 5x1 vector containing the 5 parameters describing the model.

Besides \( k \), the calibration toolbox also returns the focal length in number of pixels. The focal length is the distance of the image plane to the central point; in figure 4.3 the central point is denoted by the origin \( O^2 \).

Furthermore the principal point is the actual center of the image, taking into account the bending of the lens. The value returned is also in pixels.

4.1.1 The calibration

The toolbox works as follows. Several images of a checkerboard from different angles are loaded into the toolbox. For each image the far corners of the checkerboard have to be defined. A Matlab routine can now detect all corners of the checkerboard with a corner-extraction tool. The distortion of the mesh of the board is now known for all images. These distortions are inputs for an iterative function, comparing the results to the presented camera model. The outputs are as described in the former section.

For the calibration 16 images of a checkerboard were taken with the board camera. The size of the images are 576 by 768 pixels. Figure 4.1 gives an example. The results as returned by the toolbox are shown in table 4.1.

The results from the calibration are not optimal. The focal length has an uncertainty of about 1.5 pixel and the principal point about 2.5 pixel. Also especially note the uncertainty of the 3rd and 4th value of the distortion parameters. Since the calibration is based on a 6th order distortion model, it is obvious that not all irregularities of the camera can be extracted. Also due to the fact that the camera is relatively cheap and the lens is very wide-angle, the model does not suffice. As the time for the project is limited and the results have acceptable errors, the model and its parameters are kept and used for the compensation of the distortion.
4.1. CAMERA CALIBRATION

![Image of a calibration object](image)

Figure 4.1: An image from the camera used for calibration

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Uncertainty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Focal Length:</td>
<td>591.40115</td>
<td>1.51311</td>
</tr>
<tr>
<td></td>
<td>575.01596</td>
<td>1.53674</td>
</tr>
<tr>
<td>Principal point:</td>
<td>356.04448</td>
<td>2.44502</td>
</tr>
<tr>
<td></td>
<td>293.29578</td>
<td>2.23398</td>
</tr>
<tr>
<td></td>
<td>-0.38128</td>
<td>0.00294</td>
</tr>
<tr>
<td>Distortion vector:</td>
<td>0.11378</td>
<td>0.00278</td>
</tr>
<tr>
<td></td>
<td>0.00017</td>
<td>0.00050</td>
</tr>
<tr>
<td></td>
<td>-0.00089</td>
<td>0.00050</td>
</tr>
<tr>
<td></td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
</tbody>
</table>

Size The size of the images used for the calibration is not equal to the size of the streaming image. The images used for calibration are larger than the 240x320 streaming image size. These bigger images are used to get better performance from the toolbox. Also, corners are more easily detected with a bigger image.

Using the values returned by the toolbox for a smaller image involves a scaling of the principal point and the focal length only. The distortion factors remain the same for each image size.

The following scaling factors are used to generate the matrices.

\[
f_{\text{stream}} = f_{\text{calibration}} \left( \frac{S_{\text{stream}}}{S_{\text{calibration}}} \right)
\]

\[
c_{\text{stream}} = c_{\text{calibration}} \left( \frac{S_{\text{stream}}}{S_{\text{calibration}}} \right)
\]

with \( f \) the focal length, \( c \) the principal point and \( S \) the size of the image. The sub-index 'stream' denotes the streaming image; 'calibration' denotes the image that is used for calibration. Note that \( f, c \) and \( S \) are vectors.
Inverse Using the presented camera model it is possible to map a point from a pinhole projection to the distorted image. We, however, would like to do the opposite: the distorted image is known from the camera and we need to know the pinhole projection of a certain point. The model is not analytically invertible. Therefore the toolbox inhibits an iterative function that calculates the inverse: normalize.m.

4.1.2 Implementation

As the inverse mapping of pixels is not dependent on time, it is possible to precalculate the pixel coordinate in the pinhole projection using the presented function from the toolbox.

It is implemented in the Matlab file make_calibration_matrices.m. The results are stored in two matrices with on the \( ij \)th position the coordinates of the pinhole projection of the pixel on \((i,j)\), or

\[
P_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} M_x(x_d, y_d) \\ M_y(x_d, y_d) \end{bmatrix}
\]

(4.7)

with \( M_x \) and \( M_y \) the two matrices storing the inverse mapping.

In order to easily transfer the calibration information to an application, the matrices are stored as arrays in a text file. One file contains the x-coordinates of the pinhole projection and the other the y-coordinates. The index of the array returns the x or y coordinate of the pinhole projection, defined as

\[
P_n = \begin{bmatrix} x_n \\ y_n \end{bmatrix} = \begin{bmatrix} m_x(x_d + y_d * Iwidth) \\ m_y(x_d + y_d * Iwidth) \end{bmatrix}
\]

(4.8)

where \( m_x \) and \( m_y \) are the two arrays containing the x- and y-coordinates respectively.

The method is very fast and hence suited for online processing. However, if another camera or even another image-size is used, the matrices must be re-calculated and re-read by the application.
4.2. THE RELATIVE POSITION

From the previous section, the target defined on the camera image can be transformed to pinhole projection coordinates. Using the autopilot’s gyro to read the current pitch and roll, an estimation of the relative position of the target can be made.

Figure 4.3 shows schematically how points in space are projected on an image plane. This projection is the definition of a pinhole projection: all points in space draw a line to a certain center, and the rays are captured on the image plane, located at a certain distance from the center.

Reversing the order, projecting an image pixel on the ground, gives the relative position. From the figure it is clear that, given the height and the rotations of the camera, this reversed approach is deterministic, with the only limitation that the ground must be flat.

If two coordinate systems are considered, one attached to the camera ($e_2$)
and one to the ground ($e_1$), the projection can be defined in $e_2$ as

$$P_n = P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \tag{4.10}$$

and the projected point in $e_1$ as

$$P_1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \tag{4.11}$$

The 2D image coordinates of $P'$, which is the pinhole projection of the 3D point $P_1$ in space are given by the linear equations

$$x' = x_2 \frac{f_x}{z_2} \tag{4.12}$$

$$y' = y_2 \frac{f_y}{z_2} \tag{4.13}$$

where $f_{x,y}$ is the focal length of the lens, and $x_2$ and $y_2$ the representations of $x$ and $y$ in $e_2$. A difference is made between $f_x$ and $f_y$ because cameras can have a different focal length for the horizontal lines and the vertical lines [Calib]. $P'$ defines the target and so $x'$ and $y'$ are known. $P_2$ represented in the camera-fixed, rotated coordinate system $e_2$ can now be defined as

$$P_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = \begin{bmatrix} x' z_2 / f_x \\ y' z_2 / f_y \\ z_2 \end{bmatrix} \tag{4.14}$$

Note that all three coordinates depend on the only unknown variable $z_2$. Rotating the coordinate system $e_2$ to the ground-fixed system $e_1$ results in

$$P_1 = A \cdot P_2 + \begin{bmatrix} 0 \\ 0 \\ h \end{bmatrix} = \begin{bmatrix} x_2 A_{11} z_2 / f_x + y_2 A_{12} z_2 / f_y + A_{13} z_2 \\ x_2 A_{21} z_2 / f_x + y_2 A_{22} z_2 / f_y + A_{23} z_2 \\ x_2 A_{31} z_2 / f_x + y_2 A_{32} z_2 / f_y + A_{33} z_2 + h \end{bmatrix} \tag{4.15}$$

with $A_{ij}$ the $ij^{th}$ entry of the rotation matrix that maps $e_2$ to $e_1$ according to Euler's Rotation Theorem, and $h$ the height of the camera. $P_1$ now is a function of known variables and one unknown variable $z_2$. As the height is added to $P_1$ the $z$-coordinate of $P_1$, $P_1(3)$ must be zero. Solving for $z_2$ this results in

$$z_2 = -h \left( A_{31} x_n / f_x + A_{32} y_n / f_y + A_{33} \right)^{-1} \tag{4.16}$$
4.2. THE RELATIVE POSITION

Now substituting the expression for $z_2$ in (4.15), the relative position of the target $P$ can be expressed as

$$P_1 = \begin{bmatrix} -h(x_1 x_n f_y + A_{12} y_n f_x + A_{13} f_x f_y) \\ -h(x_1 x_n f_y + A_{32} y_n f_x + A_{33} f_x f_y) \\ -h(x_1 x_n f_y + A_{22} y_n f_x + A_{23} f_x f_y) \\ h(x_1 x_n f_y + A_{32} y_n f_x + A_{33} f_x f_y) \\ 0 \end{bmatrix} \quad (4.17)$$

4.2.1 Offline test

The algorithm is implemented in the point tracking application. The image coordinates as indicated by the defined target are used in the above algorithm and the relative position of the target is shown on the screen. Figure 5.2 displays a screen capture of the interface.

The system is tested by mounting the camera on a 53cm high table. The rotations of the camera, the pitch and the roll, were not precisely zero and also hard to measure. The yaw is easily calibrated by the edge of the board of the camera and so assumed to be known and zero. The point on the ground directly under the lens was marked and defined as the target. With this setup the relative position algorithm should output zero.

First the pitch and roll were calibrated; because the relative position should be zero, the pitch and roll were updated until the output indeed indicated zero. This resulted in a pitch of -0.021, and a roll of -0.01 radians.

Now, defining another target on the ground, the position of that point could be read from the screen, as well as directly measured from the marked point directly underneath the camera. Figure 4.4 displays the set of testing points, together with the algorithms output. Assuming the errors are similar for every quadrant of the camera only the lower-left square is calibrated.
Figure 4.5: A sequence of images showing the correction of pitch and roll. In the upper-left image the pitch and roll are zero, in the above-right image the pitch is 0.2 rad and the roll is zero, bottom-left the pitch is zero and the roll is 0.2. In the bottom-right image the pitch and roll are both 0.2. In all images the x and y position of the target show (0,0), indicating the target is right underneath the camera.

The figure shows that the error more towards the edge of the camera image is larger. Note that the resolution of pixels per meter is less towards the edges because of the high bending of the image, which also influences the error.

For another test the camera was mounted at a pitch-angle of approximately 45 degrees. This showed errors with a very similar pattern. We can conclude that the relative position algorithm is working well, but that the camera calibration is not optimal.

4.2.2 Online test

The pitch and roll correction is demonstrated by a sequence of images in figure 4.5. The images are taken with the board camera above a checkerboard print with a block size of 25 mm. The camera was mounted on the plane so the autopilot inside the plane could measure the pitch and roll.

The figure shows four images. In each image the plane with the autopilot and the camera mounted on it is tilted approximately to the in the figure indicated values. The actual relative position of the target is for every image zero as the target was right underneath the camera. The output of the algorithm is overlayed on the image and also displays a relative position of zero, indicating that the system works.
Remarks  The camera was not directly fixed with the autopilot because the autopilot was mounted inside the plane and the camera underneath. So the assumed zero mounting angles are not accurate. Also the Gyro of the autopilot is not exactly calibrated.

From the previous offline test it is clear that the error resulting from the calibration is more of influence towards the edges of the image. In this test the target remains near to the center of the image, so the influence of the calibration error is low.

Still, the images presented here only show that the system works online. The test-setup was not accurate enough to draw conclusions about the accuracy of the system.
Chapter 5

The Autopilot

The autopilot is an automatic pilot. It reads the onboard pressure sensors to determine height and speed, a GPS gives position and heading, and the onboard gyro provides information about the pitch and roll of the plane. These values are the feedback inputs for the onboard PID controllers. The controller outputs send their signals to the various servos. The user commands that are processed by the autopilot are the reference values for the control schemes.

The first section of this chapter will discuss the most important functionalities of the autopilot.

5.1 Introduction to the Autopilot

5.1.1 Feedback controllers

In the autopilot 24 PID controllers can be configured, with user defined inputs and outputs. Also the proportional, differential and integral parameters can be set for certain ranges of speed. The PID settings include an anti-aliasing and saturation setting. The autopilot can switch between control loops for various flying-modes. For example a descent needs a different control scheme than a level-flight [MP2028].

The autopilot came with a set of controllers and servo settings that are suited for this specific plane. These settings stabilize the plane and guide it through the flight plan.

5.1.2 High-level commands

The autopilot can be programmed with high-level commands. The onboard compiler translates the commands and saves the values in specific fields. For example flyTo (100, 100) sets a new relative waypoint, 100 meters in x and y direction from the position where the plane took off. A waypoint is simply a point on earth where the plane should fly to. The autopilot
CHAPTER 5. THE AUTOPILOT

determines whether it has reached the waypoint by defining a square around
the position with a width and height as specified in the autopilot (field 122
[MP2028, p. 92]). Upon reaching the waypoint the autopilot calculates the
heading for the next waypoint and sets this as the reference heading.
FlyTo (46:36.122N, 6:53.233E) can also be used and is clearly an abso­
lute waypoint in GPS coordinates. Another example of a high-level com­
mand is climb 20. This command sends the plane to an altitude of 20
meters.
All the commands together form the flight program of the autopilot. The
next section discusses the flight program and how it is stored in the autopilot.

5.1.3 Flight program

The autopilot can be programmed with an extensive flight program of up
to 1000 commands. This flight program entered in a text-file, and can only
be uploaded to the autopilot with the Horizon software (see 5.2.1). The
onboard compiler reads the commands and stores the values in reserved
fields. These fields, where the flight program is stored, start at number
10000. Ten fields are reserved for each command, and so the next command
is stored in the fields starting at 10010. Ten fields per command are used
because the command itself, the units of the parameters and the values itself
must be stored.
In the SDK manual [MPSDK, p. 25] a list of the fields for each parameter
is included. Also the commands and their numeric equivalent are listed.
Appendix A displays the final flight program that is used for the test flights.
It shows that patterns are defined that deal with a signal failure. See section
7 and appendix B.2 for details.

5.1.4 Autopilot settings

On the autopilot all values for all settings, including PID settings, user­
commands and servo settings are stored in fields. All fields are in long­
format. With software it is possible to read from or write to a specific
field. The Horizon program includes a dialog screen where all settings can
be entered and saved to a text file. This text file has the extension .vrs and
can in once be read or written to the autopilot. This is useful to backup all
current fields of the autopilot.
Also using the SDK (section 5.2.3) fields can be read or set. This is useful to
monitor the status of the autopilot or remotely set for example the desired
heading.

5.1.5 Flying modes

While in-flight, the autopilot has three modes, namely UAV, PIC and RPV.
5.1. INTRODUCTION TO THE AUTOPILOT

- The UAV mode (also CIC, Computer In Command) is the mode where the plane is stabilized and the flight program is executed.

- In PRV mode the plane is stabilized and controlled to maintain the current heading, airspeed and altitude.

- The PIC mode stands for 'Pilot In Command' and gives the pilot full control over the plane with the RC transmitter.

In PIC mode the autopilot is still running and a datalog is recorded. The switch on the RC transmitter controlling channel 5 is the method to switch the autopilot to PIC or CIC mode. In this way it is possible to quickly manually recover a control failure.

Before takeoff, the pilot can test the functionality of the autopilot by switching to CIC mode. The servos should now move in a stabilizing manner responding on rotations of the plane. For example the rudder should move to the right when the plane is yawed to the right, see the manual [MP2028, p. 43] for details.

The PIC mode is used to launch the plane. When airborne, the pilot can switch to CIC mode where he gives control to the autopilot. For switching to RPV mode a radio link must be active; only then Horizon or an application using the fnSetModeDLL-function from the SDK can switch to RPV mode.

5.1.6 Servos

The autopilot is capable of controlling 24 servos. For each servo the range and zero position can be entered in the servo-setting fields. In the current setup there are seven servos connected to the autopilot.

5.1.7 Global Positioning System

The autopilot uses a Global Positioning System (GPS) as a feedback for the position and heading. Each second the GPS is updated and the new heading is calculated. To be able to calculate a heading the plane must have a certain speed [MP2028, p. 123]. If the speed is high enough and heading fixes are generated, field 1127 [MP2028, p. 102] is set to 1.

When the GPS backup battery is disconnected the GPS looses its memory. Upon power up the GPS must now download a new almanac file that contains the information about the current position of the satellites. This procedure can take up to 20 minutes [Garmin].

Table 5.1 shows the latitude and longitude coordinates of a point in the center of Lausanne according to [LongLat].
Table 5.1: GPS coordinates of Lausanne

<table>
<thead>
<tr>
<th>Latitude</th>
<th>46 32’ North</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longitude</td>
<td>6 39’ East</td>
</tr>
</tbody>
</table>

5.2 Monitoring and communicating

The autopilot uses the serial RS232 for communication. The wireless transceiver onboard the plane communicates with the ground transceiver on a 900MHz band. According to the specifications the range should be a few kilometers, however no tests have been done to verify this range. MicroPilot designed a re-entrant protocol, which means that more than one thread can be communicating with the autopilot at the same time. This protocol is described in the SDK-manual [MPSDK] and implemented in the Windows library Simdll.dll.

MicroPilot offered several possibilities to communicate with the autopilot. The first is the Ground Control Station (GCS) Horizon. This software package is specially written for the autopilot and capable of monitoring its status and sending commands. Second there is the basic Windows serial link interface called HyperTerminal. This is a text-based program that can basically do the same as Horizon, and has some additional features. However, with HyperTerminal it is not possible to change variables on the autopilot when in-flight. Last, MicroPilot offered a Software Development Kit. With this it is possible to create your own interface using C++ or Microsoft Visual Basic. The functions include reading and writing variables on the autopilot, also in-flight. The next three paragraphs will discuss the three mentioned communication possibilities and evaluate their capabilities and performances.

5.2.1 Horizon

Horizon is the GCS that is delivered with the autopilot. The main view displays a map and the location of the plane on the map. During flight the position of the plane can be monitored. Also the pitch and roll are graphically displayed. Horizon is the only program that can send the flight-program files to the autopilot. It has an interface where all the settings can be entered. These settings can be transmitted or received using the program. Further the status bar displays the status of the GPS, the serial link and the mode of the autopilot. The number of found satellites can be monitored and the program indicates when the GPS is locked and the sensors have initialized. Also the autopilot settings and the flight-program can be tested using a simulator. Figure 5.1 displays a screenshot of the program with on the background a low-resolution picture of Lausanne. The yellow circles represent waypoints.
5.2. MONITORING AND COMMUNICATING

Performance

The main functionality of the program works well, however the program is very unstable. A wrong setting or for example zooming out too far on the map generates a runtime error and the program ends. This makes it hard to debug the session. It is not possible to disconnect with the autopilot when it is still initializing or locking on to satellites. It can be useful to monitor the pilot during startup using HyperTerminal, but then Horizon must be closed.

5.2.2 HyperTerminal

The serial program included on every windows system is HyperTerminal (HT). When HT is active while the autopilot is initializing, it lists the progress of the sensor initialization and the GPS status. After initialization HT displays the current value for every sensor. HT can be used to list a sensor report, display the onboard flight-program, and enter the setup-mode where every field of the autopilot can be seen and modified. As a first attempt to be able to automatically control the path of the plane from the ground, HT was relied on to establish this control. However, as mentioned, the setup-mode cannot be entered while the plane is in-flight, which makes it impossible to send variables to the pilot using HT. See section 6 for more details on the control.
Performance

In general HT is the most reliable program. It however also has some disadvantages. First of all, when another program like Horizon has been connected with the pilot, HT cannot be used anymore; it doesn’t respond to the key-commands and lists no report. A restart of the autopilot is necessary. Further is it not possible to exit the setup-mode, for that also a restart is necessary. With HT a list of the flight-program can be displayed by pressing the p-key four times. However this displayed flight program is not the full program (See appendix B.2). This is quite confusingly and leaves no other possibility to check the current flight-program than going through the 10000+ fields in setup-mode (see section 5.1.3 for details). Also after using HT for a while it starts to generate unnecessary output.

5.2.3 Software Development Kit

In contrast to the expectations that HyperTerminal could be used to have automatic control over the path of the plane, an additional package was needed. This package consists of a Windows API and the manual [MP-SDK]. The Software Development Kit (SDK) is used in the windows C++ application created for the point tracking and now also functions as a controller and a monitor of the autopilot. Figure 5.2 shows the interface of the application. The goal creating the application was not to make a stand-alone program, rather to implement all functionality in classes and create a test environment because ongoing tests with the autopilot will require different functionalities. Also fixed variables that are used are not updatable in a dialog window, but rather in the code itself.

The main functions used are fnInitLinkDLL to open a serial connection, and fnReadVarDLL and fnWriteVarDLL to read and write values to specified fields in the autopilot. Multiple threads are used to read and write variables using the given functions because each command takes about 0.2 seconds to accomplish. When the functions would be used in the main code the application would respond very slowly.

A distinction is made between fields that are useful for monitoring the status of the autopilot and field that are used to control the path of the plane. Both groups of fields, stored in a structure, are continuously read or written using two background threads in the main code.

A function describing the desired heading as a function of the position updates the variable that is also written by the thread to the autopilot. Also the main program reads the same variables updated by the monitor-thread in order to plot them on the screen. A list of the fields that are now used to monitor and control the autopilot can be found in the documentation of the application.
5.3 FLIGHT SIMULATOR

![Graphical display of a flight simulator](image)

**Figure 5.2:** The program with the implemented SDK functions

**Performance**

Having the SDK gives a lot of freedom creating a ground control station application. It in fact would be possible to for example 'drag' the plane with movements of the mouse. The updates remain slow now the control group consists of five variables to read or write. Each field is thus updated only about once a second. The fields used to monitor are updated even slower, about once every two seconds since the number of variables is double. If a response code from any function used is not zero, i.e. the function could not be executed [MP-SDK], a variable is set to the response number and the main program can respond accordingly. In a newer version it would be useful to monitor the connection and try to re-connect when it is lost. A similar functionality can be found in the Horizon software.

5.3 Flight simulator

The library provided by MicroPilot includes a simulator. The flight program or the real-time control can be tested as the simulator outputs predicted values for all fields on the autopilot. The same functions to read and write variables to the autopilot are used, however with the first parameter set to one. In Horizon the simulator is implemented and a flight program can be simulated. But with Horizon it is not possible to test the control application made with the SDK. For that the application could be extended with the simulator functionality.
Chapter 6

Controlling the flight path

As mentioned, the autopilot has preset settings for the PID feedback loops. For example a PID feedback loop is configured for controlling the current heading to the desired heading.

When the autopilot is in RPV mode, the only task is to maintain the current altitude, speed and heading. New directions of waypoints are not calculated, and the desired heading is not changed by the autopilot itself. This allows the user to set a desired heading using the SDK. Setting this desired heading, the controller will control the plane so that this heading is reached and maintained. Now the path of the plane can be more or less controlled. As there is no dynamic model of the plane it is hard to exactly predict the behavior of the plane. Also disturbances like the wind influence the path. The current PID settings are trusted and relied on to control the plane to the desired heading.

With the knowledge of the ability to set the desired heading, the question is what the optimal path is that the plane should follow being at a certain position, in order to fly around and keep sight of the target. The next section discusses the proposed solution.

6.1 Flying around the target

Of course, the optimal path to fly around the target is circular. The preferred radius of the circle is discussed in the latter of this paragraph; first a new technique that guides the plane to, and lets it keep track of the circle is discussed.

For now let us assume the plane knows its position and the position of the target using the GPS or camera. Thus at any point it knows the relative position of itself with respect to the target, where the target is defined as the origin $O(0,0)$.

The dynamic equations that have a circular stable limit cycle are given by
Figure 6.1: Vector field that describes the heading. The current circle describes a radius of ten meters, but for a larger circle the vector field is simply upscaled.

\[
\begin{align*}
\dot{x} &= x + y + x/r^2(x^2 + y^2) \\
\dot{y} &= -x + y + y/r^2(x^2 + y^2)
\end{align*}
\]  

(6.1)

where \( r \) denotes the radius of the limit cycle. This system can be used to calculate a desired heading: for every \((x, y)\) coordinate the vector \([\dot{x}, \dot{y}]\) points in the direction of the circle. The length of the vector is in this case not important, because only the desired heading is sent to the autopilot. Calculating the direction of the vector in radians is done with the inverse tangent function,

\[
\text{Heading} = \tan^{-1}(\dot{y}/\dot{x})
\]

(6.2)

Figure 6.1 displays the vectorfield resulting from this function around a certain origin. Note that all vectors now have the same length. The desired heading at any point can directly be set equal to the outcome of the function. Since the onboard controller is set quite weak, and the refresh rate of the heading is about once a second, it is most likely that an error will occur. This error means that the actual circle will be wider. The error will probably remain small because the further from the circle, the sharper the heading points towards the circle. As mentioned, the dynamics of the plane are not known and so an accurate prediction of the error cannot be made. However if the simulator as mentioned in section 5.3 is implemented in the application, the actual radius could be predicted.
6.1. **FLYING AROUND THE TARGET**

**Radius**

The radius of the circle that the plane should follow is dependent on two values. The first is the height; to have a proportional view of the target at different heights, the radius of the circle must be proportional to the height. Second, assume that the camera is mounted so that when the plane flies straight the ground directly under the plane is visible on the camera image. This might have the consequences that the camera should be equipped with an even wider-angle lens.

The current camera has a big distortion and therefore the angle of sight is not the same throughout the image. A test showed that this particular camera has approximately an view angle of 90 degrees horizontal over the middle of the image. The view angle in the corners is even larger. Analyzing the compensated camera image as depicted in figure 4.2 also shows the difference in angle of sight. Satisfying the demand that the target must stay in sight, the roll-angle $R$ at which the plane can fly the circle is the second variable that determines the preferred radius.

With a camera angle $\alpha$, roughly the maximum roll-angle that the plane can make in order to keep the target on the image is

$$R < \frac{\alpha}{2} - \arctan \left( \frac{r}{h} \right)$$  \hspace{1cm} (6.3)

where $r$ is the horizontal distance to the target (the radius) and $h$ the height of the plane above the assumed flat ground. See figure 6.2 for a graphical display of the mentioned variables. Supposing $R$ almost zero, the arctangent of the ratio $r/h$ may not exceed half of the view-angle of the camera in order to keep the target in sight. In this case the view-angle is 90. So when $R$ is zero, $r/h$ may not exceed 1. A $r/h$ ratio of 0.5 (the height is twice the distance to the target) results in a maximum roll-angle of 18 degrees.

At this moment the roll-angle $R$ is not known as a function of the radius $r$ because the roll-angle depends on the unknown dynamics of the plane. A conclusion of the preferred radius can therefore not be made at this point, however equation 6.3 and figure 6.2 present the boundaries.

### 6.1.1 Feedback from GPS

As supposed in the former section, it is indeed possible to use the GPS coordinates to locate the plane and the target. The SDK can set coordinates for the target, and the current position of the plane is read from the GPS. However the GPS coordinates are not meters east and west. The formula presented in [Michaud] transforms the coordinates to meters. Note this function is only valid around Lausanne and is different everywhere on earth.

$$dP_x = P_{\text{latitude}} \cdot 111158.6 + P_{\text{longitude}} \cdot 850$$
$$dP_y = P_{\text{latitude}} \cdot 1189.7 + P_{\text{longitude}} \cdot 76761.3$$  \hspace{1cm} (6.4)
Figure 6.2: The Roll-angle $R$ of the plane in relation to the position of the target $T$

Here $dP_x$ is the position in meters in south-north direction, and $dP_y$ the position in meters in west-east direction. Alternatively the autopilot uses an approximative algorithm to transform the GPS coordinates to meters east and west, relative to the origin (where the autopilot was initialized). This algorithm is less accurate but valid all over the world. The relative positions are stored in the autopilot and can also be used as a feedback.

**GPS error**

The absolute location measured by GPS has an error of up to 10 meters, but a relative position from the GPS with respect to it’s former position results in an accuracy of up to half a meter. This relative position is however time-dependent and therefore only valid for a short period of time. Still the relative position from the GPS is well suited for testing purposes; to define a target and apply the circular heading control.

### 6.1.2 Feedback from Camera

Of course, the main goal is to use the camera to provide information about the location of the target. As the algorithm provides relative position information about the target, these coordinates can almost directly be used in the equations for the circular heading. When the plane circles around the target the location of the target on the camera image theoretically remains the same. Therefore also the heading calculated with the algorithm remains the same. Contrary, the autopilot derives its heading from the GPS readings, so the heading is absolute. Using the camera as a feedback, the control
6.1. FLYING AROUND THE TARGET

Figure 6.3: A sequence of images supposing the plane flies around the target.

for heading must be adapted.

Suppose the plane flies east, 90 degrees. The target is straight ahead so the desired heading is also near 90 degrees. Approaching the target the desired heading according to figure 6.1 becomes e.g. 70 in order to follow the described circle, a difference of 20 degrees. The desired heading according to the camera now indicates -20 degrees, which is not the right absolute heading to fly to.

Flying with a feedback from the camera involves reading the actual absolute heading calculated by the GPS and adding the heading that the camera feedback indicates. As now the GPS is only used to give information about the current heading, this device could be replaced by a compass.

**Alternative approach**  The current heading can also be entered as yaw in function (4.17). The relative position will now be defined so that the x-axis is parallel to north south, and the desired heading following from 6.2 now returns the absolute heading. As the influence of yaw on the algorithm is not yet calibrated, the method above is not used. Instead the former described method is implemented.

6.1.3 Online test

To demonstrate the determination of the desired heading, a series of images from a web-camera is taken. Note that the demonstration of this algorithm is independent of the camera; here another camera is used than the board
Figure 6.4: Two images displaying the desired heading if the plane is (1) too far from the target and (2) too close to the target.

camera. The images are taken above the checkerboard print with block size of 25 mm. The radius of the circle is set to 20 mm.

Figure 6.3 displays four images. The white cross is the average of the small white dots, the tracked points. The line with the dot indicates the heading that is desired according to the position of the target. As described in section 6.1.2 the desired heading is a correction to the current heading and therefore the relative direction the plane must fly.

As the 'plane' approaches the target the heading points a bit left, indicating that the plane should take a left turn (upper-left image). This is conform the vectorfield as seen in figure 6.1. After taking this turn the heading points again more forward in the upper-right image. In the 3rd image (lower-left) the plane flies aside the target and the desired heading is straight. Would the plane now go straight, the heading will point right to keep it on the circle. When turned right as indicated in the last figure, the heading still points straight.

Figure 6.4 depicts what would happen if the plane is too far (left) or too close (right) to the target. The line again points in the direction that the plane should fly in order to follow the prescribed circle.
Chapter 7

Tests and results

The test flights are in the first place to test the autopilot itself. An experienced model airplane flyer was willing to help flying the plane.

7.1 First test flight

First a test flight was planned to test the plane without the autopilot. The autopilot board was removed from the fuselage, and the RC receiver and the batteries for the servos were kept onboard. The batteries for the autopilot were glued to the fuselage, and therefore also not removed. But as the payload changes the behavior of the plane, it was actually good to test the plane including these payloads.

Results

The plane responded well and the engine seemed powerful enough to handle a bigger payload.

7.2 Second test flight

The aim for the second test flight was to test the autopilot. Since the RC transmitter had a 5th channel to control the mode (UAV or PIC) of the autopilot, no laptop was needed. The onboard datalog would automatically start recording when a GPS heading was measured, and the datalog is extended enough to do a good analysis; see [MP2028, p. 111] for the fields recorded in the datalog.

Without laptop, it is possible to monitor if the autopilot is finished initializing and fixed the GPS-position by watching the servos in PIC mode. If they respond to the handles, the autopilot is ready.
CHAPTER 7. TESTS AND RESULTS

Results

After initialization, the ailerons, elevator and rudder responded in the right manner. The plane was manually launched and flown for a few minutes to test the behavior of the plane with the new, heavier payload. Everything seemed fine until there was a problem with the engine, probably some dirt in the fuel. The plane was landed and the engine restarted. Again, still on the ground the CIC mode was tested. This time the servos did not respond to the movements of the plane but rather went periodically from one side to the other. This error was not seen before. Still another takeoff was performed hoping the error would vanish as the GPS would be able to fix a heading again. This was not the case as the plane, switching to CIC mode while in-flight, tumbled down. Switching back to PIC mode, the pilot was still capable to recover a stable flight after this drop. The datalog would have had to record data as it is self-triggering, but none was found.

7.3 Third test flight

This test flight was planned to test the plane with a serial link to a laptop. A power converter, converting the car-battery to 220 volts, powered the laptop and the modem. Now the initialization could be monitored, the status during flight could be monitored and the flight-path controller could be tested. However, the first aim was to have a successful autonomous flight, and second to test the new controller.

Results

Initializations of the autopilot were good, however, after the GPS had locked, it took another few minutes for the sensors to initialize. This is not normal (appendix B.2) and can probably be avoided by completely fixing the plane. The sensors must stabilize before the initialization completes. The windy conditions prevented the plane to be really steady. The modem-link was only working when the plane was on the bench next to the laptop. The range of the link should be a few kilometers, however when the plane was taken for takeoff, the link was lost. The power converter might have been the problem that the communication was bad. Using 9-volt batteries for the transceiver instead might solve the problem. Also while in UAV mode, the servos sometimes moved randomly for a second. Although the RC transmitter was on, this problem persisted. Probably because of interference of other signals. A newer version of the software on the autopilot is available soon and should fix this problem (B.2).
7.4. FOURTH, GENERAL TEST

Again, no datalog was available from the autopilot. According to the answers in appendix B.2 there are two possible explanations. Or the GPS lock was lost before flying, or the datalog was deleted before being able to download it.
The datalog is important for evaluation; with the new information from B.2 the datalog should be working and the procedure for downloading should be clear.

7.4 Fourth, general test

This test was conducted to verify the new information provided by MicroPilot. There were three main goals. One was the persisting loss of the GPS signal, the second the bad communication link and the last was the onboard datalog.
This time a 9v battery was used for the wireless modem, as the power converter might not provide 'clean' power.
The flight plan was updated with a command that would handle the case if the GPS signal was lost. The action taken should be flying to an altitude of 5 meters.

Results

The serial communication was quickly lost again. The only solution to this problem seems to remount the antenna on the plane. The autopilot did record a datalog this time.
This datalog revealed that the servos again showed the periodical behavior, indicating a fatal error. It is unknown why the servos indicated the fatal error as there was a pattern defined that should have dealt with the problem of a loss of the GPS signal.
Chapter 8

Conclusion

A first objective was to determine whether it is possible to replace the functionality of the GPS by a camera. The autopilot needs the GPS for a position feedback, but the presented system only uses the GPS for the heading feedback. This heading feedback is also achievable by for example a compass, and therefore function of the GPS is discarded. The camera replaced the feedback for position.

For realization of the feature tracking a streaming video is imported into an application where at any time a picture can be captured for analysis. These images are used for a point tracking feature, realized using the openCV libraries. The tracking seems very robust as it consists of several tracked points; however the tracking has not yet been tested on a video stream recorded from an onboard camera. The redefinition of lost points prevents the target to drift over the image. The only disadvantage is that the target must stay in sight of the camera; also clouds could disrupt the target tracking.

The camera is calibrated with the Calibration Toolbox for Matlab. The image can be compensated for the distortion easily using the precalculated matrices. The now assumed pinhole projection is not completely accurate and has an error of maximal 2.5 pixels in the far corners of the image. This error is for this application however not at all disturbing.

The target on the image is projected to the actual position of the target, relative to the plane’s position. This algorithm uses the height, pitch and roll of the plane, read from the autopilot. The error of the relative position is maximal 5% and only this large in the far corners of the image. This has a strong relation to the errors of the calibration of the camera.

With the relative position of the target a function is presented that will guide the plane to fly towards, and then around the target.

In the application created for the feature tracking, the ability to read and write fields on the autopilot is implemented. This extends the application to a monitor for the autopilot, and a control station that can send commands
to the airborne autopilot. With this functionality integrated, the application forms a good tool for testing the system and the autopilot.

The autopilot showed it has capabilities of autonomous flight stabilization. Communicating with the autopilot on a longer range remains a problem. With the desired heading uploaded to the autopilot, the solution is found to control the actual path of the plane.
Bibliography

http://www.vision.caltech.edu/bouguetj/calib_doc/index.html

Image Processing, Analysis, and Machine Vision.

MP2028g Installation and Operation.
October 2003.

MP-SDK.

Le Shrimp-AT

Pyramidal Implementation of the Lucas Kanade Feature Tracker.
Intel Corporation.

Good features to track.
IEEE Conference on Computer Vision and Patter Recognition. Seattle,
June 1994.

[opencV] The openCV libraries.
http://www.intel.com/research/mrl/research/opencv/

[Garmin] Garmin.com, about GPS.
http://www.garmin.com/aboutGPS/

http://www.simpleocr.com/

Appendix A

The flight program

The final fly-file used for the test flights

```
metric
takeoff
climb 20
waitClimb 10
flyTo (25,25)
flyTo (25,-25)
flyTo (-25,-25)
flyTo (-25,25)
repeat -4
definePattern rcFailed
flyTo [home]
repeat -1
definePattern gpsFailed
climb 5
repeat -1
```
Appendix B

MicroPilot feedback

B.1 Q&A after receiving the equipment, 20-01-2004

Questions regarding HyperTerminal

1. How can we tell the AP that we perform a manual takeoff? Or, how does the AP determine it’s "flying" or "on ground" after manual takeoff?

You do not need to tell the autopilot that you are doing a manual takeoff. It will assume that you have when it sees that ground (GPS) speed and airspeed are at a sustained, elevated level. The autopilot will switch from ground mode to flight mode when these speeds rise above approximately 10 fps. Note that even if you are performing a manual takeoff, you must include the takeoff command in your .fly file. The autopilot will automatically skip to the next command when it enters flight mode.

2. We would like to fly once (manual takeoff) without using a serial link, since the AP records the data. How can we set the AP to GIG mode when the plane is still on the ground? This to test the movements of the servos. Is the only way to initiate a takeoff with the "tttt" command?

You can switch into GIG mode using the Ch.5 switch on the transmitter when the plane is on the ground. Not all of the same feedback loops that are in use in the air will be active on the ground, however, you should still be able to see the rudder respond to yaw, the elevator respond to pitch and the ailerons respond to roll.

3. How can we make the AP set the throttle to idle, to test the motor at this setting?

The autopilot will use the idle setting after it has initialized, when it is on the ground.
Questions regarding Horizon

1. During a simulation, using Horizon, the throttle becomes 0.0% sometimes. I hope this will not happen in-flight?

   While in flight, the throttle will not drop below the idle setting.

2. The number of fields of the AP and the HORIZON simulator don’t agree: Version AP: 20031020, Version SIM: 20030728. I’ve set the right "override" in “Edit GCS” menu set, but still get error: "x range error" when simulating this .vrs file.

   The x range error is likely due to the fact that the older code version used by the simulator had the xacc sense reversed as compared to the newer code version. You may wish to create a different .vrs file for simulations which has the "Negative X" box checked off in the .vrs Sensors menu.

3. The wireless radio-modem works fine, using HyperTerminal. But when I want to connect HORIZON to the AP with this modem, no connection can be established. (Direct connection of AP and HORIZON works fine)

   Are all of the communication parameters the same between HyperTerminal and the Horizon software? Does the modem communicate at a different baud rate than what you use with the cable? You may also need to increase the min_wait_ms and max_retries parameters in the simulate.ini file.

4. I’m not able to send a new .fly file to the AP-1, "MP2000 is not set for in-flight reprogramming". How can I? (Note: there is no “fixed” command currently in the default waypoints in the AP).

   If you want to send .fly files while in "Connect" mode, the "fixed" command must be present in both the .fly file that you want to send and the .fly file currently in the autopilot. If you have not entered "Connect" mode, you can send .fly files without the presence of the "fixed" command. Unless you are actually in-flight, you should send the .fly files before connecting. Otherwise, you can only send those with the "fixed" command, and only the portion of the file above the "fixed" command will be transmitted (patterns/additional threads cannot be reprogrammed in-flight, only the main flight plan can be replaced).

5. Adding a "fly to (absolute or relative waypoint)” by clicking the right mouse button in the field (and then simulating) generates the error "invalid .fly file".

   Did you select headers and footers for this .wpt file? If not, there will be commands missing that are essential. Clicking on the map only
generates waypoints; you also need to include the "takeoff" command, a "climb" command, as well as some command such as "repeat -1" or "circuit" to terminate the flight. If you wish to generate .wpt files in this way, you will have to create your own header and footer or use the default ones.

6. Also, after setting the right GPS long-lat positions, the absolute (long-lat) values to add (see 5), given after pressing the "append" button, do not correspond to the values stored in the .fly file! (They somehow mix up with the initial starting long-lat-position setting in HORIZON).

Can you give me a detailed example of this problem, including the initial starting position set in the GCS Options, the lat/lon's selected when you created the waypoint, and the values stored in the .wpt file that you generated? I would like to attempt to duplicate this problem as precisely as possible.

7. When we switch between .fly files in-flight, when does it execute the new commands? Direct or does it wait until the previous command is finished?

When you send a new .fly file to the plane in-flight, it overwrites the main portion of the .fly file. Thus, unless the autopilot is executing a pattern that you have set up (remember that patterns are not replaced by in-flight reprogramming) it will immediately begin executing the new commands.

The following questions are related to our desire to have full control of the path of the plane, and not necessarily send the plane to a certain waypoint.

1. Executing a "turn" command in a .fly file, how and when does the MP decide to execute the next command? In simulation the plane fly's to an angle of approx 75 degrees, and then executes the next statement. Please see the included capture (on the next page) of the simulation, note that the path of the plane after a "turn 90" command resembles an exponential function instead of a desired quarter-circle path. We plan to use the turn-command to have the desired control over the plane's path.

The "turn" command is supposed to turn the autopilot to the desired heading, and then immediately move on to the next command. However, the autopilot will begin to reduce the angle of bank as the current heading approaches the target heading (otherwise, the aircraft would severely overshoot the target heading). Hence, the commands that you have specified (turn 90, turn 180, turn 270, turn 360) will not generate a smooth circle, as the turn will become more gradual as current heading converges with the target heading.
Figure B.1: The results from a simulation using Horizon. The flight-plan is listed on the left.

2. If we a C++ program to generate e.g. new waypoints, and want to transmit them automatically, do we have to generate a .fly file and send it using e.g. an I2C-protocol? Or how could it work otherwise? Or is it possible to send certain values to the MP from the ground-station? We which to be able to use for example "turn [value1]" with "value1" a changeable value.

Yes you will have to create a new .fly file. You can then send this file using the Horizon. (Note: if you are in flight your original .fly will have to have a fixed command) You can also send this file by writing a custom C/C++ or VB application utilizing the functions found in MicroPilot's SDK. You cannot send certain values to the AP from the ground-station example "turn [value1]" individually; they must be part of a .fly file.

3. Will the telemetry-report also work when the HORIZON-program is not used during flight?

If you are not connected to the Horizon software, the telemetry file will not be recorded. If you are considering making your own GCS software, you should be able to set it up to record the telemetry file. Note also that the autopilot stores an on-board datalog, which can record up to 1 hour of flight.

B.2 Q&A after the first test flight, 17-02-2004

1. Several times we've seen that the servos, in the UAV mode, periodically switched from left to right, about once a second. This error usu-
ally appeared a few minutes after initialization was complete, without touching any button or control. This was the main problem that persisted and made not being able to perform autonomous flights. Please find attached the text file with the sensor-output from HyperTerminal.

When the autopilot moves the rudder and elevator in this fashion it is because there is a fatal error in the system. When you initialize the system connected to the GCS, do you see any error message appear below the artificial horizon gauge? Note that one fatal error is a loss of GPS lock - if you were working over top of the plane, you may have caused the GPS to lose its lock which would in turn cause the autopilot to report a fatal error. I would also suggest that you check the low battery warning levels that you have set and make sure that they are appropriate for the batteries that you are using.

Initializing the autopilot returns no fatal error, and I’ve verified the low battery settings, so the problem must be the loss of the GSP-lock. For our application we actually don’t use the GPS coordinates. Can setting the gpsFailTimeout-field to for example 200 solve the problem? Or is it necessary to define a pattern with the gpsFailed instructions? Will this pattern also be executed when the plane is in RPV mode?

The autopilot uses the GPS information for relative and absolute waypoints, and also uses it to determine heading. If you are not using the GPS how will you navigate? Do you have a compass module for determining heading? The failure patterns should also work in RPV mode.

2. Also we’ve seen several times that, being in UAV mode, the servos move very randomly for a little second. After that they returns to normal.

Occasionally the autopilot receives a signal that interprets as being a valid signal from the R/C. This can happen when the transmitter is off - interference from external sources can cause this. Do you find that this problem disappears once you turn on the transmitter? There will be an updated version of autopilot available soon that eliminates this problem, but you can safely fly with the current code despite this issue.

If I remember correctly the RC transmitter was always on. However this is not a major issue, we’d be interested in the updated version.

3. The movement of the rudder, reacting on yawing the plane, was not consistent. Once it worked well after initialization. Then, after returning from PIC mode to start the engine, the rudder was not moving anymore. Please explain.

If you are connected to the GCS and watch the flight mode indicator (should say either gnd or fly) what state does the autopilot think it’s
in? If the airspeed sensor is not properly zeroed, it is possible for the autopilot to think it is in fly mode even when it is sitting on the ground. Note that when the autopilot is in fly mode, the rudder no longer corrects for yaw - instead it is used to coordinate turns in conjunction with the ailerons. I would also suggest that you carry out all of the post-installation checks listed in the manual and make sure that the yaw gyro values look reasonable.

I don't recall the indication in Horizon, but the wind might have triggered the fly-mode. I'll carry out the post-installation checks again.

4. The onboard datalog has not yet once worked. What should exactly trigger the record? What exactly deletes the previous records?

The datalog does not start recording until the autopilot enters flight mode, or until you fake the GPS lock. The log is cleared each time the autopilot initializes. What happens when you try to capture the datalog?

The manual states that the logging starts when the GPS-speed is greater than zero. Is this also the trigger for the AutoPilot to determine that it's flying? Also, the manual says that to download the log, you'll have to turn on the AutoPilot and wait for 5 seconds until it starts to initialize. Can I conclude that this should always be done inside, since the log is cleared when the GPS locks? Can I also download the log directly after the flight, without cycling power of the AutoPilot?

The autopilot does look at ground speed as well as airspeed when determining whether it has entered flight mode. Note that the autopilot typically takes a minimum of 30 seconds to initialize, and that once the datalog loading program starts, the initialization process is halted. Thus, you can get the log outdoors, but you have a fairly small window in which to do so. Generally I wait about 10 seconds to be sure that the autopilot is ready to transmit the log and then I start the load.exe - this seems pretty foolproof.

5. The serial link with the AutoPilot is very bad. During initialization the link is mostly well, except that Horizon almost periodically indicates a red background in the link-status. During flight we have not yet had any link, and so also no telemetry report or ability to send or read information.

I would suggest that you thoroughly check all connectors on your radio modems, especially those between the antenna and modem modules. You may also want to do a continuity check on the antenna extension cables. Your communication issues could also be caused by inappropriate communication settings in the GCS. Can you send me a copy
of your simulate.ini file so that I can see what COM settings you are using? Do you have COM problems when connected to HyperTerminal with the serial cable or does this only happen when using the GCS? If you have problems with HyperTerminal as well, you might need to check the serial cable itself to make sure there are no faulty contacts.

We have had problems with both HyperTerminal and the GCS. However while performing some tests on the roof, with the computer in the office, I encountered no problems. Transportation might have caused a faulty contact.

If you are experiencing intermittent problems, it sounds very much like this could be a wiring issue. It’s probably worth spending the time to find where the problem is - this may end up saving you time and frustration in the long run.

6. Uploading the flight plan (.fly file) using Horizon Always gives an error, mostly when “writing to flash”, after “uploading” and “verifying” are successful. The error is ”No communication link with the MP2028”. Now after restart I have no fatal error. Printing the pppp-report lists the flight plan, however not complete. Also when I checked the flight-plan-fields, #10100 seemed to be 'invalid', without giving an error at startup. Please explain and tell me where I can find the very current flight plan onboard the MP2028.

The PDA report ”PPPP” will give you the first few lines of the current .fly file in the autopilot. You may need to adjust the aforementioned communication parameters in the simulate.ini file for the .fly file transfer to work properly.

The error persists after changing the settings. Could you please send me the right parameters? Also I would suggest a possibility to check the whole flight-plan in a newer version.

Can you tell me what version of the GCS software you are using? You may want to check our support website for the latest GCS software version. There were some communication problems discovered between the 2028 and older GCS versions. The maximum values I would suggest trying for min_wait.ms and max_retries are 500 and 20 - if you still cannot send at these settings it is likely a version conflict. Contact Jennifer Bell at MicroPilot (jbell@micropilot.com) if you do not have a username/password to access the support site.

7. It seemed necessary to have Horizon running and connected during startup? This was the only time that after initialization the rudder responded to yaw etc.

The autopilot should initialize properly whether you are connected to the GCS, HyperTerminal, or not connected to anything (assuming that
you are not required to fake the GPS lock to complete initialization).

8. Also during initialization there was some wind and the speed sensor was not covered. Could this be of influence to the autopilots behavior? **It is possible that if there was enough wind, the autopilot could make the determination that it is in flight mode. I would suggest placing a wind block in front of the pilot when initializing in windy conditions.**

9. When the GPS was locked it took mostly another 5 minutes to indicate ready in Horizon (there was some wind, but the plane was lying steady).

   This is definitely unusual. Typically the gyros will stabilize very shortly after the GPS locks. Again, I would suggest that you carry out the post-installation checks in the manual to verify that the sensors are all working properly and generating reasonable values.

10. The autopilot has once entered the setup-mode by itself. HyperTerminal displayed "MP2000:" How could this happen?

    Were you using a direct serial connection at the time this happened or were you communicating through a modem? It is possible that the autopilot received some garbage characters that it erroneously interpreted as a command to enter setup mode. However if this happened with a direct serial connection, I would doubt this is the case. Let me know what the circumstances were and certainly inform me if it happens again.

    We were communicating with a modem. I’ve seen it only once now, and will notify you if it might happen again.

    What kind of radio modems are you using? Do they perform any kind of error checking? You may want to enable CRC checksum mode in the autopilot and GCS - this might help to avoid a repeat of this problem. Set field 581 to a value of 1, and set crc.type=1 in the simulate.ini file.
Appendix C

Product references

Autopilot

Website:
http://www.micropilot.com/products/MP2028g.html
Airspeed sensor: Motorola, MPXV10G, K0325, 30.
Altitude sensor: Motorola, MPXA4115A, K0321, 01.
Processor: Motorola, 16 MHz, 2k RAM, MC68LK332ACPV16, 1J66A, QQAF0313.

Wireless serial link

Manufacturer: MaxStream
Frequency: 900 MHz
Onboard module
   Name: 9XStream
   Website: http://www.maxstream.net/products/xstream/module/9xstream.php
Ground-station module
   Name: 9XStream-PKG-R
   Website: http://www.maxstream.net/products/xstream/pkg/9xstream.php

RC transmitter and receiver

Manufacturer: JR
Frequency: 72 MHz
Transmitter
   Type: XF421
   Number of channels: 5
Receiver
   Type: R700
Number of channels: 7

**Video overlay card**

Name: BOB III  
Website: http://www.decadenet.com/bob3/bob3.html

**Camera**

Name: Board Color Camera  
Manufacturer: RF concepts  
Website:  
http://www.rfconcepts.co.uk/board-colour.htm

**Video transmitter and Receiver**

Name: 600mW 2.4GHz Audio/Video Transmitter with Receiver  
Manufacturer: Microcameras  
Website:  
http://www.microcameras.com/video_transmitters/600mw_24ghz_transmitter.htm

**Frame Grabber**

Name: Digital Video Creator  
Manufacturer: Dazzle (Pinnacle)  
Website:  

**Propeller**

Manufacturer: APC  
Size: 8x4

**Engine**

Name: OS .15 LA  
Manufacturer: OS engines  
Website:  
http://www.osengines.com/engines/la.html
Table C.1: Batteries

<table>
<thead>
<tr>
<th>Battery</th>
<th>Manufacturer</th>
<th>Type</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autopilot</td>
<td>Cadnica</td>
<td>KR-1100AAE</td>
<td>1100mAh</td>
</tr>
<tr>
<td>Servo</td>
<td>Cadnica</td>
<td>N700AAC</td>
<td>700mAh</td>
</tr>
<tr>
<td>GPS</td>
<td>Panasonic</td>
<td>BR2477A</td>
<td>1000mAh</td>
</tr>
</tbody>
</table>

Batteries

Table C.1 summarizes the onboard batteries.
Website servo battery:
http://www.vaica.fr/vaica_pages/accus_nicd_ind.htm
Website GPS backup battery: