Master Thesis

Computer vision based flight control for the pole racing competition

Author(s):
Browarnik, Diego

Publication Date:
2010

Permanent Link:
https://doi.org/10.3929/ethz-a-006422950

Rights / License:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information please consult the Terms of use.
Master Thesis:

**PixHawk**

Computer vision based flight control for the pole racing competition

Diego Browarnik
Legi-Nr. 01-924-018
diegob@student.ethz.ch
D-INFK

Friedrich Fraundorfer, PhD
Institute of Visual Computing
Assistant

Prof. Marc Pollefeys
Institute of Visual Computing
Tutor
Date: 08/2009 Version: 0.5
# Contents

1 Introduction                                4
   1.1 Introduction and Motivation            4
   1.2 Statement of the Problem               5
       1.2.1 Previous Work                    5
   1.3 Objectives                             6
   1.4 System Overview                        6
   1.5 Hardware Details                       7

2 Image Acquisition & Preprocessing          10
   2.1 Image Acquisition & Preprocessing      11
       2.1.1 Camera Calibration                11
   2.2 Edge-Detection                         11
   2.3 Color Thresholding                     11

3 The Computer Vision                        13
   3.1 The Computer Vision Dataflow           13
   3.2 The Prediction Projection              14
       3.2.1 Pole Model                      14
       3.2.2 World Model                    15
       3.2.3 Projection                     15
   3.3 Pole-Tracking                          18
       3.3.1 Edge-Tracking                  19
       3.3.2 Pole Recognition               20
3.4 Pose Update ................................................................. 20
   3.4.1 Mathematical Framework of the Pose Update ........... 21

4 Path Planning ............................................................... 23
   4.1 Analyzing the Path ...................................................... 23
   4.2 The Desired Path ...................................................... 23

5 Results ................................................................. 24
   5.1 Setup of the Experiment ............................................. 24
   5.2 One Single Image ...................................................... 25
      5.2.1 Working with Synthetic Data ................................. 25
      5.2.2 Working with Real Frames from Camera .................. 27
   5.3 Edge Segmentation ..................................................... 27
      5.3.1 Segmentation with normal edge detection .................. 27
      5.3.2 Texture edge detection ......................................... 30
   5.4 Fine Tunning .......................................................... 30
      5.4.1 Algorithm and Least Square ................................. 30

6 Conclusions and Future Work ........................................ 31
   6.1 Conclusions ............................................................ 31
   6.2 Future Work ........................................................... 31

References ............................................................... 32
Chapter 1

Introduction

1.1 Introduction and Motivation

In the context of the next robotic competition *EMAV09* (European Micro Aerial Vehicle) to be held in Delft, Netherlands, the PixHawk group formed with people of the Computer Vision and Geometry Lab will participate representing the ETH. This involves the development of the own *micro aerial vehicle*, for short *MAV*, with certain constraints in weight and size. The competition consists in mastering different tasks, in which the *MAV* should be challenged in velocity, autonomy, stability, agility, etc, depending on task.

In the last years, the focus on autonomy became a highly scored feature. The current technology allows to have a mounted camera on a *MAV* and a *CPU* running computer vision algorithms, substantially changing the results in autonomy. But autonomy of the *MAV* in our case implies an important problem that all the robotic community already knows well: the estimation of the self location and orientation, a.k.a. the pose. Many approaches rely on marks or special preparations on the environment to be recognized, putting more weight in the camera perception, and its computer vision algorithm yet for the indoor navigation of a robot it is of central interest where there is no such additional information such as the GPS data that could be obtained outdoor.

Thus for a real-time environment, the processor speed should be fast, the algorithms also, but without losing the view in the trade-off with the accuracy. The main contribution of this thesis is in tailoring an algorithm to the local needs, to recognize the poles, working reliably, fast and robustly for the geometry of the poles. Optimized for this environment, the only knowledge of the poles are their positions.

The focus of these conferences, competitions and prototypes, is not only for the obvious interest of the military field. Civilian applications such as disaster
management, involving search and rescue, urban traffic monitoring are also of interest.

1.2 Statement of the Problem

The task to be mastered here is an indoor dynamics mission of the *EMAV09* with the name “pole racing” and is a test for speed and agility. The micro aerial vehicle has to fly in figure eights around two poles, which will be situated 10 meters away one from each other. Figure 1.1 shows such configuration. The start position is located in the middle, between the two poles. As it is an indoor competition it will be held in the indoor sport hall of the TU Delft.

The rules are:

- Preparation time: 5 minutes.
- Flight time: 3 minutes.
- Scoring: Each lap will be awarded 1 mission point. Points will not be awarded for a lap when the MAV flies above the pole height (4 m).

![Figure 1.1: Pole Racing: Task of EMAV09 Competition](image)

1.2.1 Previous Work

The publications of the last MAV competitions are not so well developed in the field of computer vision, so some information, such as a fast color segmentation
are from Robocup publications (e.g. *Robocup*, a competition of soccer robots team). Fast color segmentation is an important area there, e.g. [7]. For the pose estimation the main works used as reference was the Phd-thesis of Kemp [2] which also uses the edge-tracking algorithm proposed in RAPID [1], the first real-time application of such algorithm.

All these applications were implemented for geometric objects with a more complicated side-view, i.e. not like our pole, which will maintain the same side-view long rotation around itself.

A very complete survey of tracking methods and a state-of-the-art in this field is given [3].

### 1.3 Objectives

The objective of this work is to develop a system that allows the *MAV* to autonomously fly between two objects on a desired path. The two objects, in this case two poles, should be recognized from 2-D images taken from a camera mounted in the *MAV*.

The following x steps resume the three main tasks in which the work proceeded:

- Color Segmentation. Trying to extract the desired pixel colors in different illumination constraints
- Camera Calibration.
- Model of the Poles and Projection.
- Generating Synthetic Data.
- Get the position correction step
- Working with Real Data. Images captured from camera, for a single image in known position
- Video sequence

### 1.4 System Overview

Figure 1.2 shows an abstraction of the main systems of the *MAV* which are of interest in this context. The *MAV Computer Vision System* is referred to
in detail in Chapter 3. After getting an image from the camera and processing it, it calculates and provides the estimated pose of the MAV to the **MAV Path Planning System** (Chapter 4). The latter is involved together with other information from the sensors (attitude and rotation) in calculating and informing the **MAV Control System** in which direction the MAV should proceed, to follow the desired path.

The MAV Control System, which refers to the automatic control of the motors as well as to the hardware communications, it is outside of the scope of this thesis.

---

To understand the context in which the computer vision system works and this thesis will mainly handle, see Figure 1.2. The control of the micro-helicopter that will allow to autonomously fly in this task, will be reduced to the following two steps:

- Pose estimation of the MAV in the environment.  
  (This complex step will be described in detail in chapter 3).
- Calculate where the MAV should move.  
  (See Chapter 4)

### 1.5 Hardware Details

Figure 1.3 shows the latest prototype revision of PIXHAWK, that could participate in the competition. This coaxial helicopter prototype was developed and designed by electrical, computer science, and mechanical engineering students of ETH.
Below are listed some features and hardware details. See the Webpage of the Pixhawk group [6] for more details or current state of the development.

- Camera: 640 × 480 pixel
- CPU: 2x OMAP3530 CPU and TMS320C64x + DSP (Gumstix Overo Fire COM)
- Communication: Wifi, Bluetooth
- Sensors: 3D Accelerometer, 3D Gyroscope, 3D Compass, Barometric pressure.
- Weight: 40 gramms (only mechanical frame)
- Electronic stabilization
Figure 1.3: PixHawk a coaxial Helicopter prototype for the MAV competitions
Chapter 2

Image Acquisition & Preprocessing

Overview:
This chapter summarizes in a very short form the most important features of the image acquisition sub-system. In this sub-system we combine the independent states before an image arrives to the computer vision system. The acquired images here are then pre-processed to later allow an easier work for the MAV vision system. The path of this section is independent of the rest of the system. Its pipeline produces an image every fixed amount of time to be processed by the rest of the algorithm.

![Diagram of Acquisition and Preprocessing Pipeline](image.png)

Figure 2.1: Data Flow of the Image Acquisition Sub-System
2.1 Image Acquisition & Preprocessing

2.1.1 Camera Calibration

- Details of used camera
- The internal parameters of the camera
- Will be calculated once off-line and the lens configuration fixed
- The images will be on-line rectified because of the radial distortion caused for the lens
- Once calibrated the camera it is possible to obtain the pole width from the well known start position

2.2 Edge-Detection

- Canny
- Low Pass Filter horizontal and Vertical

2.3 Color Thresholding

At the begin of this thesis it was not decided yet in the competition rules, which colors the poles should be and although the image process of the earlier image is done in greyscale, the use of a color camera is still recommendable. The idea here was to use the color information of a pixel, to validate the found pole, or to first segment the pixel where the pole could be

The Methods:

- Gaussian Mixture Models.
  Achieve a robust color segmentation at different illumination conditions.
  But not needed in our indoor approach where we need just one color.

- Thresholding
  A very simple thresholding was used. A Transform to L*ab color space of the image.
  For the first time, in pole region calculate the average for a and b values inside, and use the average with a tolerance as a threshold, every pixel between in + or - tolerance
• Growing regions
  maybe better for changes in color but computationally more expensive
Chapter 3

The Computer Vision

Overview:

In this chapter the MAV vision system is introduced in more detail.

3.1 The Computer Vision Dataflow

![Diagram of 3-D model-based edge-tracking system]

Figure 3.1: 3–D model-based edge-tracking system

The implemented 3–D model-based edge-tracking system in Figure 3.1 shows
the scheme that which will be repeated after every arrival of a new frame from the image acquisition sub-system. The 3-D model of the poles are projected (projection box in figure) on screen coordinates system using an estimate of the camera pose. Our estimate will be the camera pose calculated in the previous frame. We assuming that after moving, the position and orientation of the camera should have a small variation following a small displacement of the poles on the screen. In this way the poles are easy to find in the neighborhood of the projected ones and consecutively segmented (Poles segmentation box in figure). The distances between both projections is measured on screen (Error measure) and used to estimate the transformation matrix, which relates the pose calculated in the previous frame with the camera pose in the new frame, Thus allowing the micro aerial vehicle to update the knowledge of its own pose.

Flying with a velocity of 1m/s will mean that the camera position from frame to frame could be 10cm away. As every frame comes approximately at most every 100ms\textsuperscript{1}, displacements limited to this range are not a problem for the position tracker, but fast orientation changes, when the edge to be tracked is located far away from the camera, due to the fast changes on screen that this implies, will result in a difficult edge to be tracked or the loss of the pole.

We assume here a known start pose, which is the case of the EMAV competition task. It is located between the two poles, exactly in the middle of the path. At the beginning, the start pose is assumed as the actual one and then is used to calculate the first projection on screen, and so the normal cycle of the system continues frame by frame.

### 3.2 The Prediction Projection

#### 3–D Model & Projection

#### 3.2.1 Pole Model

The model adopted here for the poles representation in 3–D world coordinates is simple. Figure 3.2 shows the points stored in model for each of them. Only a sequence of points located equidistantly at the circumference of the top and the bottom edges of the pole sides are stored. These are in world coordinates system. The parameters, the radius of the pole and the height, could be changed, in the case its values should be learned in a learn-phase. The number of points for each pole could be selected. Here 50 points were used from the beginning.

\textsuperscript{1}The idea is to have a fast implementation on the device, allowing a better rates of image processing.
of the development of this project with comfortable results but with the idea of
gaining by empiric proceeding a good trade-off between speed and accuracy (all
this optimization steps are addressed in 5.4).

Figure 3.2: Pole Model: The only stored point of the poles are the ones at the
top and the bottom edge sides.

3.2.2 World Model

The world coordinates are the same as the pole coordinates. In Figure 3.3 the
relationship between this coordinates systems and the camera coordinates system
is shown. A rotation of $-\frac{\pi}{2}$ around the $X$–Axis is needed to align both systems.
The view direction of the camera is its own $Y$–Axis (in $Z$–Axis of world system).
The path, which the $MAV$ will follow, calculated in Chapter 4 is also referenced
in this coordinates system.

3.2.3 Projection

When we refer to camera pose, which means position and orientation of the
camera. It is represented with a $4 \times 4$ matrix of the form:

$$E_{\text{world2camera}} = \begin{bmatrix} R^T & -R^T t \\ 0_3^T & 1 \end{bmatrix}$$

where:
Figure 3.3: Relation between World and Camera Coordinates Systems. The view of the camera is aligned with its Z-Axis.

$R$ is the rotation matrix of the camera with respect to the world coordinates, and $t$ is the translation of the camera (also in the world coordinates).

The projection of a point in world coordinates to the screen coordinates is resumed in the following equation (for more details on the projection matrix, please refer to [4]):

$$
\begin{bmatrix}
    u \\
    v \\
    1
\end{bmatrix} =
\begin{bmatrix}
    f_x & s & c_x \\
    0 & f_y & c_y \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    I_3 & 0_9 & R^T \\
    0_9 & 1
\end{bmatrix}
\begin{bmatrix}
    X \\
    Y \\
    Z
\end{bmatrix}
$$

(3.1)

$p_{\text{screen}} = K * E_{\text{world2camera}} * p_{\text{world}}$

Where the matrix $K$, known as the camera matrix (for more detail on $K$ and its 5 parameters, see 2.1.1), containing the calculated intrinsic camera parameters, will be constant once the camera is already internal calibrated.

The matrix $E_{\text{world2camera}}$ will change both the rotation matrix $R$ and the translation vector $t$ describing the camera position and orientation, i.e. the pose.

As seen before at the beginning of this section, only the points of the top and bottom edges of the pole are stored, but after using the projection equation 3.1 the model points will be in screen coordinates. The other points of the left and right pole edges are only calculated when we known which are the visible points on screen. A line with equidistant points is traced from the left-most point of the
top side to the left-most point of the bottom side, for the left edge of the pole, the same is made for the right opposed part. Finally, to obtain the interesting points of the model, the visibility of the points must be checked, i.e. if they lie inside the screen limits (in our case our camera resolution gives a max. of 640 × 480 pixels) as well if there is occlusion of one pole by the other. In this manner points that are occluded or outside the screen are discarded for the next steps.

Figure 3.4: From the points in 3-D model to the screen points. (From left to right: The stored points. Projected on screen and its convex hull. Added the sides of the pole. Occluded points discarded)

The projection on screen from model is resumed in algorithm 1 below and is possible to see in Figure 3.4 the pipeline in which the points go through in such process.

**Algorithm 1** Screen Projection of the Poles (used as prediction for the edge-tracking)

**Require:** $K$, Rotation, position

$PolesNotVisible \leftarrow false$

$pose \leftarrow f(Position_{actual}, Rotation_{actual})$

$points_{screen} \leftarrow K \ast E_{pose} \ast points_{model}$

$points_{contour} \leftarrow convexHull(points_{screen})$

*if* $points_{contour}$ inside screenLimits(640 × 480) *then*

$points \leftarrow addSides(points_{contour})$

$points \leftarrow discardOccludedPoints(points)$

*else*

$PolesNotVisible \leftarrow true$

*end if*

Figure 3.5 shows, how at the end of this stage, the projected prediction on screen
The predicted pole superimposed in image is superimposed to the actual image captured from the camera. In the next section will explain how to extract the poles from the image.

### 3.3 Pole-Tracking

**Overview: Pole Segmentation & Error Measure.**

Once the prediction is projected on screen, the next step is to find the respective pixels from the image that correspond to the pole in the actual position. For that purpose the images used will be those preprocessed from the *image acquisition* sub-system. The filtered image with edge information will be used for the search, and the additional color information from the color thresholded one, is used for the verification of the poles, i.e. checking that the edges that are being selected are enclosing an area of the same color of the searched pole. The color information is also used for a more exhaustive search in the case the pole is lost by the tracker and must be searched for on a wider range. Depending on where the pole is found located on screen, there will be a mismatch between the prediction and the pole edges that recently arrived. The measure
of this mismatch is used as an error function, measuring how far are the pixel edges from one pole to the other (from prediction pole edges to the image pole edges). An estimate of the camera movement from prediction to the actual pose is obtained after minimizing this measured distances. Then it is possible to update the actual position and orientation of the MAV.

Resuming the Edge-based approach as explained in the edge-detection algorithm in [5], this is performed rapidly in the following way:

- Prediction of the position of the object
- Projection of the object using that position
- Measure the distance from real image to the prediction. (Search normal to the edges poles in 1–D direction for the edges in image).
- Update the current position. (best-fit motion that minimizes in the sense of least square the error between the two projections).

### 3.3.1 Edge-Tracking

As explained in the edge-detection algorithm in [5], the search of the image edges points correspondences is done in the direction of the edge normal for every point in the model. Because of the aperture problem this is not a big imprecision, when the objects to track have a more complicated geometry, but in our case, the poles do not give much information from its edges, just think that revolving around a pole with the camera always keeping the same distance from the object, gives for the whole path the same edge configuration on screen. To give a bit more information about the movement, when possible, in the case when a corner of the pole is found on screen, the right match of correspondences is done. The following are the steps of our edge-tracker algorithm:

- Follow left side of the model points taking the nearest edges points,
- Run a RANSAC to find vertical lines and select which image points are of interest,
- Find the offset in vertical and in horizontal direction of the first side,
- Use the offset and follow the search of the other sides and correct,
- continue with the remained pole sides.
Figure ?? shows how this local search of the edge is done. The blue points are the tested points looking for an edge in both directions of the normal edge and the opposite one (entering the edge and outgoing it). A group of point edges are selected (red points) and finally, after the RANSAC, the selected points (cyan points), are assumed as part of the side of the pole.

Once pixels corresponding to the edge of the image are selected, which are near the edges of the projected model, they should be analyzed. The idea that the closest edge is the correct side of the pole in frame will not always work, taking also into account that from frame to frame an object could be more than 10 pixels away and in a new edge could appear between the background lines like windows, columns, etc. This situation is shown in Figure ??.

**Distance Measure**

The measure of the distance is simply the number of pixels between edge point correspondences. (To obtain more information of the movement, but if it is not possible to check the correspondences it is a good measure, the number of pixels between the edges measuring in normal direction of the edge.

### 3.3.2 Pole Recognition

- It must be checked if the enclosed area by the found edges has the geometrical properties of the pole side view
- Also the color is checked, but this is done while finding the edges.

### 3.4 Pose Update

The Least square estimate the 3D translation and rotation made by the object between consecutive frames.

Thus, the result of the iteration is a $3 \times 4$ transformation matrix $V = [R - t]$ (composed by a $3 \times 3$ rotation matrix and a $3 \times 1$ translation vector), called the motion matrix, that transforms the pose calculated in the previous frame into the pose of the object in the new frame.

In the end, the system knows a relation between the 3D coordinates of the control points, their 2D projections and the 2D translation of the points made between consecutive frames.

If the pose update does not work: path planning will update the position, and here will be the connection between
the Path planning sub-system.

### 3.4.1 Mathematical Framework of the Pose Update

For a better understanding of what every matrix and position references and what needs to be calculated see the Figure 3.6 with its related equations below. The most important thing to note is that there are 2 different camera positions (with their related rotations): The *predicted position* or camera 1 and the *real position* or camera 2. Note that in the Figure 3.6 is also depicted a *start position*. This is in case we need to model the camera movement because of a difficult path, or very fast changes from frame to frame, allowing a nearer start position for the algorithm.

Knowing position 1, and the 3-D model of the pole we obtain the projection in camera coordinates using equation 3.2.

![Figure 3.6: Tracking the Pole and Updating the Pose](image-url)
\[ P_{C_1} = E_{C_1W}p_w \] (3.2)

Analogously for camera 2 is eq 3.3, but we don’t know here the matrix \( E_{C_2W} \), the transformation from camera 2 to world, and we want to know the correction matrix from camera 1 to camera 2. Combining eq. 3.3 with eq. 3.4 we obtain 3.5

\[ P_{C_2} = E_{C_2W}p_w \] (3.3)
\[ E_{C_2W} = E_{C_2C_1}E_{C_1W} \] (3.4)
\[ P_{C_2} = E_{C_2C_1}E_{C_1W}p_w \] (3.5)

Using the exponential map of eq. 3.6 to model 3-D rotation and translation with \( \mu \) as 6–dimension vector, and eq. 3.2 to obtain eq. 3.7 we obtain an finally an easily to linearize system.

\[ E = \exp(\mu) \equiv e^{\sum_{j=1}^{6} \mu_j G_j} \] (3.6)
\[ P_{C_2} = \exp(\mu)E_{C_2W}P_{C_1} \] (3.7)

The derivative of 3.7:

\[ \frac{\partial P_{C_2}}{\partial \mu_j} \]

then calculating the Jacobian with:

\[ J_{ij} = \frac{\partial d_i}{\partial \mu_j} \] (3.8)
\[ \frac{\partial p_{C_2}}{\partial \mu_j} \bigg|_{\mu=0} = G_j E_{C_1W}p_w \] (3.9)

Where \( d_i \) is the measured distance for every point found on screen

To finally solve it with least square to obtain the \( \mu \):

\[ \mu = \arg \min |J\mu - d|^2 \] (3.10)

obtaining the desired solution that represent the correction.
Chapter 4

Path Planning

4.1 Analyzing the Path

As one of the task of the MAV is to flight fast the path should be:

- short
- easy to flight for the mav
- and get most information of the landscape at every position (the camera direction at every point) points in which both poles are seen are preferred

4.2 The Desired Path

A spline approximation of the points will be used
The orientation of the camera also is part of the path i.e. every position coordinates on the path own a specific camera rotation. That will be used to correct the camera rotation when the pole is loss. An approximation will be used, calculated from the current camera rotation and the corresponding one in which the camera is supposed to be.
Chapter 5

Results

This chapter reports the results obtained in the MATLAB implementation. The idea was to switch to C once the results would have been sufficiently satisfactory, and then to analyze also the performance on the real A device, but the MAV will be ready to run the software only in a couple of days.

Basically there are three approaches used during the development of the algorithm to understand the different behavior at every stage. The difference between the three methods depends on the image data to be processed. The methods are the following:

- **Synthetic data with correspondences.** Calculate distances from formula directly and no need to search the edges, because position on screen is known for every corresponding point. The acquired image is synthetic, generated from our 3-D model, in addition that there is no noise in the edges, we also know the correct correspondences between points and there is no need to search for it.

- **Synthetic Data without correspondences.** No advice is given about the generated synthetic frame. So it must be searched for the poles and measured the distance in pixel as in the real image. But there is no noise and the edge detection is assumed to be perfect.

- **Real Data**

5.1 Setup of the Experiment

As well as for the synthetic as for the real data a configuration similar to the competition is recreated and the two poles are located, separately at a given
distance. At a fix point a picture will be captured from a good measured position and orientation.

5.2 One Single Image

5.2.1 Working with Synthetic Data

Figure 5.1 shows superimposed in the same image, the prediction (green line), the segmented edges of the image (the red line, in this case is not segmented, just generated) and the updated (the blue line). The last one is only a reprojection of model points with the new updated transformation matrix, so as expected, are the blue points near the red ones. In Figure 5.2 is shown what this update means in term of world coordinates.

Figure 5.1: working with synthetic data.
Figure 5.2: The Position update for synthetic as image. The red point is the real position, the green one is the start point of prediction and the magenta is the update after the least square.
5.2.2 Working with Real Frames from Camera

Figure 5.3 shows the first stage of the algorithm, when the prediction is already known (here in green) and projected on the edge segmented image. And after the search for the edges, in Figure 5.4 the segmented sides of the pole are shown. Finally the Figure 5.5 (color cyan for doing visibly the segmented edges) shows the reprojected points with the new position up to date.(blue).

Figure 5.3: The search of the correspondences is done in the edge extracted image

5.3 Edge Segmentation

5.3.1 Segmentation with normal edge detection

Canny etc, etc
Figure 5.4: The found sides of the poles in image
Figure 5.5: After the update, the model poles are reprojected (the blue line). They should be near the segmented sides of the pole.
5.3.2 Texture edge detection

5.4 Fine Tunning

5.4.1 Algorithm and Least Square

By proceedings of this work the MAV was mounted with a couple of sensors, one that provides a fast estimation of the rotation angle on his own axis and another one that supplies the attitude of the MAV. Therefore the least square problem addressed in section 3.4.1 has been reduced in the number of parameters. Remembering that the Jacobian Matrix of the problem will be of size: number of measurements * number of parameters, is also possible to reduce more this matrix. Ignoring also in the least square calculations, the parameters for the tilt of MAV in the other 2 axes, still reduces the problem by 2 more parameters, allowing the minimization step to more accurately calculates the minimum with the 2 degrees of freedom in the XY-Plane of the world coordinates system.

Due to the possible tilts on flight instabilities,
Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this Master’s thesis a complete pathway for image analysis for the pole racing competition task was presented. The pathway consist of a segmentation of the pole edges (which are found fast because of the help of a prediction projection) and a measure of the displacement error. This is used to compute an update of the camera position. This approach was tested using simulations and was shown to work reasonably well. When considering only a single pole, there is not enough information to correctly maintain the camera position over longer cicles of the algorithm (note that seeing 2 poles on the screen help the algorithm to find again the position). This is due to the accumulation of small errors which can not be avoided, in the position updates.

6.2 Future Work

Improving the correspondences between some special points as corners would give a more accurate position update. When the prediction is located far away of the real image pole, another method for minimizing could be used one well suited method would be the Levenberg-Marquardt, then our method for far away start points could not converge at all.

For the pole racing, only an implementation on the MAV itself, as well as an implementation of the path planner are still needed.
Bibliography


