Local sampling for indoor flight.

G.C.H.E. de Croon C. de Wagter B. Remes R. Ruijsink

Aerospace Software and Technologies Institute, Technical University of Delft, Delft, the Netherlands

Abstract

A challenging problem in artificial intelligence is to achieve vision-based autonomous indoor flight with Micro Air Vehicles. Approaches to this problem currently do not make use of *image appearance features*, because these features generally are computationally expensive. In this article, we deliver a proof-of-concept that appearance features can be extracted computationally efficient enough to be used for autonomous flight. In particular, we present a novel height control algorithm that uses *local sampling*; it estimates the height at which an image is taken by processing small image patches. We vary the specific number of image patches to directly influence the trade-off between processing time and the accuracy of the height estimation. The algorithm is first tested on image sets and then on videos taken from a real platform. Finally, we test the algorithm on a 15-gram ornithopter in an office room. The experiments show that very few image patches (0.56% of all possible patches) are already sufficient for the task of height control.

1 Introduction

Micro Aerial Vehicles (MAVs) form a promise to observe places that are either too small or too dangerous for humans to enter. However, autonomous indoor flight of MAVs remains a challenging and largely unresolved problem. A reason for this is that MAVs have little weight, sometimes in the order of grams. As a consequence, MAVs can often only carry a camera onboard. An example of such an MAV is the *ornithopter*[1] shown in Figure 1. Because of the scarce sensors, successes in autonomy obtained on other, larger, robotic systems (e.g., [4]) cannot be copied in a straightforward manner. At the moment, there are two main approaches for achieving indoor flight on the basis of onboard camera images.

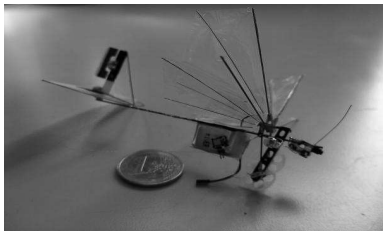


Figure 1: Our research goal is to achieve autonomous indoor flight with an MAV such as the *DelFly Micro*, a 3.07-gram ornithopter with a wing span of 10 cm. It carries a small camera and transmitter.

The first approach uses the camera to accurately estimate the state of the MAV (3D position, pitch, yaw, and roll). Such a state estimate can be obtained by using a 3D model of the environment, which can either be prefabricated [9] or learned (e.g., [5, 1]). There is an increasing number of computer vision algorithms that succeed in building a model of the environment. However, they are currently still computationally expensive. A negative consequence of this is the necessity of heavier, more energy consuming processors for eventual onboard processing. In addition, this results in delays on the state estimates, which may lead to

[1] An ornithopter is an entity that flies by flapping its wings.

control problems. For example, when entering a new room, the state estimation of the MAV does not work immediately[2].

The second approach is a bio-inspired, computationally more efficient, approach to indoor flight. Studies of this approach typically abandon a state estimate altogether and focus on training the right responses to incoming visual inputs. For example, [2] use optic flow algorithms and evolutionary algorithms to train MAVs to avoid walls. In fact, they show in simulation that height control can be treated equally to wall avoidance; the floor and ceilings being just differently posed walls. Studies of the second approach generally exploit optic flow and the measurements of accelerometers (cf. [8, 7, 12]), since similar sensor readings have been shown to play an important role in insect flight [6, 3].

Currently, both approaches work best in textured evironments. The reason for this is that they both rely on feature point tracking, be it for constructing a world model [1] or for calculating optic flow [2]. The algorithms for feature point tracking rely on the clear presence of texture, which is not always available in indoor spaces, e.g., when facing a white office wall.

While feature point tracking is a common computer vision tool in efforts for reaching autonomous flight, *image appearance* has been largely neglected. Image appearance features could be useful for autonomous indoor flight, since they are an obvious complement to optic flow. For example, the absence of texture (a fail-case for feature point tracking) can be successfully detected by extracting appearance features. The neglect of using appearance features so far is due to two main reasons. First, the extraction of image appearance features is generally computationally expensive. Second, it is not evident how image appearance features will generalize to different environments.

In this article, we address the first reason for not using appearance features: we show that local sampling allows an appearance extraction algorithm to be computationally efficient enough for indoor flight. Specifically, we present an algorithm that learns to estimate the height of the MAV on the basis of small local image samples (called *textons*, cf. [14]). The number of these samples determines the trade-off between the computational effort of the algorithm and the accuracy of the estimate. A low number of extracted samples leads to a quick and inaccurate estimate, while a high number of samples leads to a slower but more accurate estimate. We show that somewhere on this trade-off is a point suitable for achieving autonomous height control.

We remark that we will address the second reason for not using appearance features in future work: we believe that the solution to the generalization problem lies in the combination of appearance features with optic flow and accelerometers.

The remainder of the article is organised as follows. In Section 2, we describe the height estimation algorithm, the selection of its parameters, and the offline tests we performed. Then, we focus on the implementation of the algorithm for controlling a 15-gram ornithopter in Section 3. We explain the experimental setup, the controller, and evaluate the results of the algorithm in flight. Subsequently, we discuss the limitations and advantages of the proposed algorithm in Section 4. Finally, we draw our conclusion in Section 5.

2 Height estimation algorithm

The height estimation algorithm processes the images from a forward pointing camera. The algorithm uses appearance differences between images taken at different heights, assuming little variance in the pitch of the MAV. Simply put: an image taken close to the ceiling is different from an image taken close to the floor. The algorithm is based on the work of [14]. They showed that the computationally efficient *texton* method performed better than computationally intensive filtering methods (such as Gabor filters) on a texture classification task. We use the texton method, since it has a rather good performance on image classification tasks and it is amenable to our local sampling approach. Below, we describe our implementation of the texton method (Subsection 2.1). It is computationally even more efficient than the implementation used in [14], since we severely limit the number of local samples used for classification. In our description, we do not mention the exact parameter settings, since they were determined by experimenting on a hand-made image set (Subsection 2.2). Finally, we show the results of the height estimation on a video taken from a flying coaxial helicopter (Subsection 2.3).

[2]Remark that not all studies of the first approach use an accurate 3D-model of the world to obtain an accurate estimate. Instead, some studies make specific assumptions on the space in which the MAV is flying. For example, state estimation can be performed with the help of 'vanishing points' (cf. [11]). The vanishing points can be determined on the basis of parallel lines that are in sight. The method assumes that such lines are known and are in the field of view.

2.1 Description

The texton method starts with the gathering of an image set that represents the type of space in which the MAV has to fly. The images in the set need to be taken at H different heights[3]. After constituting the image set, we commence the learning of a *dictionary* of n textons (also referred to as *visual words*). To this end, we extract small image samples of size $w \times h$ pixels from each image in the set. The extraction locations are drawn from a uniform distribution over the entire image. The extracted samples are clustered by means of a Kohonen network (cf. [10]). There are other - more advanced - clustering techniques (see [13] for a MATLAB©toolbox with most recent methods), but the Kohonen network has the advantage of learning the clusters in an iterative fashion. It finds clusters quickly, and can be stopped as soon as the clusters seem to cover the different samples sufficiently.

We can use the dictionary to represent an image as a histogram g of visual words. From the image, we extract s image samples. For each sample, we determine which visual word i in the dictionary is closest (Euclidian distance), and increment the corresponding bin in the histogram $g(i)$. Normalisation of the histogram results in a maximum likelihood estimate of the probability p of each word in the image: $p(i) = g(i)/s$.

The so-formed probability distributions are used as feature vectors in the learning and classification of different heights. We divide the image set in subsets of images taken at the same height $h \in 1, 2, \ldots, H$. For each subset we determine the probability estimates for all images. We then estimate the average probability for each word $\overline{p_h(i)}$ and the corresponding standard deviation $\sigma(p_h(i))$.

The classification of an image starts with the extraction of s image samples. This leads to the estimate of p, which can be compared with the learned p_h. We made use of two very basic classification methods to classify p: Naive Bayes (NB) and a variation of Nearest Neighbour (NN). We vary on the NN method, since it is normally slow at execution time. Instead of comparing a new point with all points in the training set, we only compare it with the H 'centroid' points $\overline{p_h}$ (the average of all points p observed for a height h during training). Algorithm 1 contains pseudocode for the classification.

Algorithm 1 Algorithm to classify an image as being of height class $h \in \{1, 2, \ldots, H\}$

```
empty the histogram g
for i = 1 to s do
    pick a random location (x, y) in the image
    extract an image sample centered on the location
    for j = 1 to n do
        determine the Euclidian distance of the sample to word j
    end for
    select the closest word, k
    increment bin g(k)
end for
for i = 1 to n do
    p(i) ← g(i)/s
end for
if NN then
    for i = 1 to H do
        determine the Euclidian distance of p to p_i
    end for
    h ← i, for the p_i closest to p
else if NB then
    for i = 1 to H do
        determine the probability of p being generated by p_i, according to independent Gaussian distributions for each visual word j, i.e.,
        ~N( p_i(j)-bar, σ(p_i(j)) )
    end for
    h ← i of the p_i leading to the largest probability
end if
```

Although our choices have been tuned towards fast learning of textons and of different heights, the most important aspect of the algorithm is a fast execution. Given a classification method, the computational effort of algorithm 1 depends on the number of words n and the number of extracted samples s. The computational effort c is approximately:

$$c \approx sn(nW) + n + H(nC) = sn^2W + n + HnC \tag{1}$$

, where W is the cost of calculating the Euclidian distance between two single bins, and C is the cost of comparing two single bins with the chosen classification method. During execution, n is fixed, but s can

[3] In principle, height estimation could be approached as a regression problem. The choice for classification is due to practical advantages concerning the formation of training data.

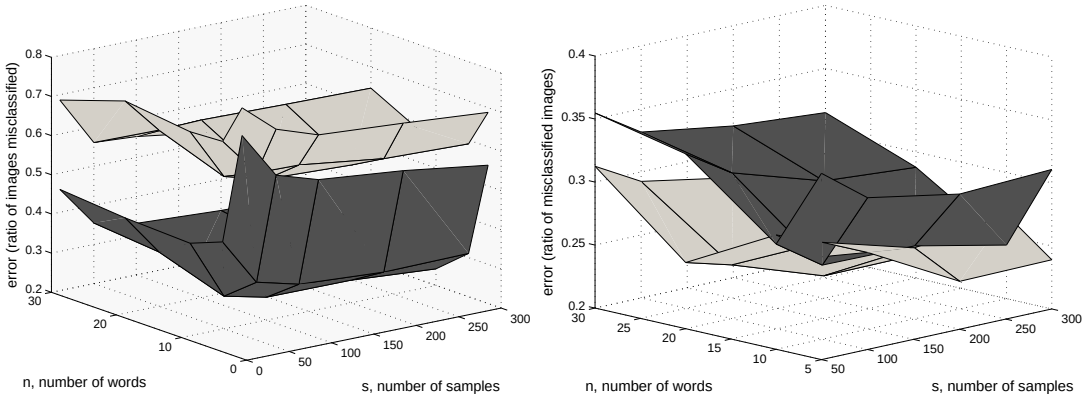


Figure 2: **Left:** Mean error (expressed as the ratio of misclassified images) when using a separation in the image (dark grey), and no separation (light grey), for $s \in \{10, 50, 100, 200, 300\}$ and $n \in \{2, 5, 10, 20, 30\}$. **Right:** Mean error when using colour images (dark grey) and when using black-and-white images (light grey) for $s \in \{50, 100, 200, 300\}$ and $n \in \{5, 10, 20, 30\}$.

be varied freely. Applications of the texton method in the computer vision literature typically set s equal to the number of all possible local samples. However, here we try out smaller numbers to explore the trade-off between the computational effort and the accuracy of the classification method (see Subsection 3.2).

2.2 Parameter Selection

The texton method involves a number of choices, including the setting of parameter values. We made these choices on the basis of experiments on a hand-made image set. We created the set by walking up and down our office corridor holding the camera at three different heights ($H = 3$): close to the ground (15 cm), at waist level (100 cm), and close to the ceiling (250 cm). In order to get reliable results, we used a separate training set and test set (different walk with varying heights). Because of the random nature of the selection of the samples, we performed ten different training and test runs per parameter setting / choice. Since n and s are the most important parameters, we always tried out different values for these parameters.

One important choice is to divide the image into a bottom part and a top part. If we represent an image as a single probability distribution of words, we throw away all spatial information related to the words. We can preserve coarse spatial information by dividing the image in a bottom and a top part. The histogram and probability distribution then have double the size[4]. The left part of Figure 2 shows the difference in average error on the test set for the plain method (mean error shown in light grey) and the method with a bottom and top part (mean error shown in dark grey), for $n \in \{2, 5, 10, 20, 30\}$ and $s \in \{10, 50, 100, 200, 300\}$. Dividing the image leads to a better average performance, at the cost of $n + HnC$ extra computational effort. Another interesting comparison is that between the use of colour images (dark grey mean) and black-and-white images (light grey mean), shown in the right of Figure 2. Surprisingly, black-and-white images seem to lead to a slightly lower error than colour images for most configurations. This is most likely due to overfitting of the colour method.

Similar experiments led us to set the sample size to 5×5, the image size to 160×120, and to choose the nearest neighbour algorithm. Concerning the last choice, the naive Bayes classifier can lead to a better performance, but has a larger standard deviation in the results. The choice for the nearest neighbour algorithm is a choice for reliability.

Importantly, all of the experiments showed roughly the same relation between the number of samples and the error: increasing the number of samples only has a modest influence on the error above 100 samples. In addition, they all demonstrated that it is unnecessary to have more than 10 words. The final result of the experiments is an error of around $0.22 = 22\%$. In the next subsection, we investigate whether this error is low enough to estimate the height of a flying MAV.

[4]The computational cost becomes: $c \approx sn^2W + 2n + 2HnC$

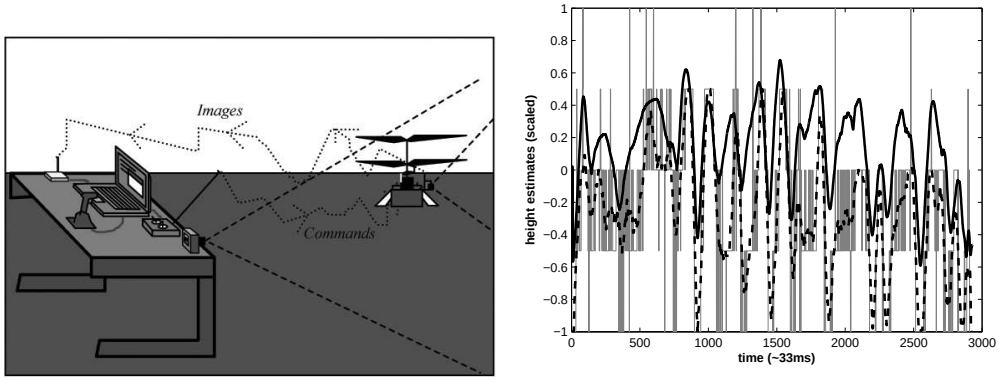


Figure 3: **Left:** setup of the experiment (see the text for details). **Right:** comparison of the helicopter's y-coordinate in the external camera (solid line, scaled to $[-1, 1]$ with 1 being the top of the image), and the height estimated by the algorithm (grey line, scaled to $[-1, 1]$ with $h = 3$ mapped to 0). The black dashed line is a smoothed version of the grey line.

2.3 Test on MAV images

Further experiments were performed to determine how well the classification of the algorithm correlated with the height of a flying MAV. The differences with a hand-made video include more realistic image shake, WLAN-noise, and realistic light differences. In this experiment, we used a coaxial toy helicopter (a modified version of the *Lama V4*) equipped with a small onboard colour camera and transmitter.

We illustrate the experimental setup in Figure 3 (left). The pilot used the joystick to control the helicopter. The computer sent the joystick commands to an RC transmitter, which ported them through to the helicopter. The pilot flew the helicopter so that it attended different heights. During the flight, the helicopter sent its images to a laptop computer running our ground station software - *SmartUAV*. The software ran the texton method, with the settings as determined in the last subsection. The only difference was in the number of height levels, $H = 5$. An external camera filmed the helicopter. In this manner, we were able to get an impression of the height of the helicopter and compare it with the outcome of the texton method.

After the experiment, we aligned the videoframes of the external camera and that of the laptop computer. Then, we used straightforward motion detection to detect the helicopter in the external camera images and registered its median y-location in the image over time. The right part of Figure 3 shows the y-coordinate over time (solid line), the height classification (grey line), and a smoothed version of the height classification (black dashed line). The figure shows that the uncertain and discrete classifications can be smoothed to give a 'continuous' signal, which correlates well with the y-coordinate of an external camera.

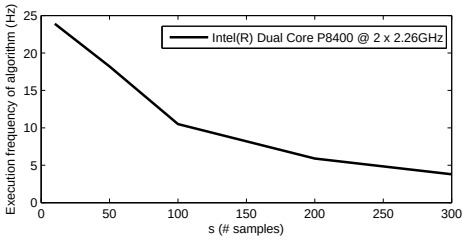
3 Flight experiment

After the success of the algorithm on the images of a flying platform, we tested the algorithm inside the control loop of a 15-gram ornithopter. In what follows, we explain the experimental setup (Subsection 3.1), the controller (Subsection 3.2), and we show the results of our flight test (Subsection 3.3)

3.1 Experimental setup

The experimental setup is the same as in Figure 3, but now we use the ornithopter as the flying platform. We perform the experiments in an office room, which underwent no other preparation than switching on the lights and pushing the furniture aside. For the online (in-the-loop) test, we film the ornithopter from two view points (see the map in Figure 4, right). In this way, we can reconstruct the 3D trajectory of the ornithopter during the test. The experiment starts by the pilot controlling the ornithopter via a joystick. He takes off, gives a trim to the throttle of the ornithopter, and then pushes the firebutton. When the firebutton is pressed, the computer is in full control of the throttle. During the height control experiment, the pilot

Figure 4: **Left:** the 15-gram ornithopter used for the height control experiments. **Right:** map of the room used in the flight experiment. The pilot only controls the rudder of the ornithopter; the height is controlled autonomously. The ornithopter is filmed from two viewpoints: by the webcam of the laptop and the camera used in the previous experiment.



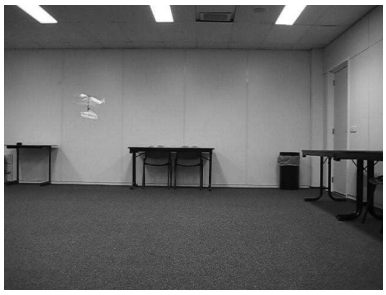


Figure 5: **Left:** frequency (in Hz) of the thread running the height estimation algorithm. **Right:** picture of the height control experiment.

continues to control the rudder of the ornithopter to avoid the walls. The elevator is constant throughout the experiment.

The left part of Figure 4 shows the ornithopter we have used for our experiments. It is a 15-gram ornithopter with two black-and-white cameras and transmitter on-board: only the forward pointing camera is used. Its X-tail allows it to perform vertical take-offs and landings. Powered by a LiPo battery, it can in principle perform a hovering flight for 15 minutes. We have to remark that the particular ornithopter used in the experiments of this article is part of a series produced in 2007. It was originally designed for 10 flying hours, but has already seen more than 200 flying hours; it is still able to hover, but only achieves a few minutes of hovering flight.

3.2 Controller

The controller used for the experiments is rather straightforward. The ground station receives an image, and uses Algorithm 1 to classify it as one of the heights $h \in \{1, 2, 3, 4, 5\}$. The height is scaled to the interval $[-1, 1]$, with height 3 mapped to 0. This height value is used as an error value in a regulator for the throttle. The throttle has a value in the interval $[-1, 1]$, with -1 being full throttle. Concerning the regulator: for the experiment with the ornithopter a P-controller was already sufficient to achieve height control, $P = 0.45$.

By running the controller and moving the ornithopter up and down (while being turned off), we could explore the trade-off between the number of samples s and the performance / computational effort. We noticed that the algorithm already performs sufficiently with only $s = 100$ samples, corresponding to the height algorithm running at $\sim$10.5 Hz. To compare, with $s = 300$, the algorithm runs at $\sim$3.8 Hz. Figure 5 shows the frequency of the algorithm on an Intel(R) Dual Core with 2.27 GHz for $n = 10$ and $s = 10, 50, 100, 200,$ and 300. Note that this frequency was determined while also running the controller and video receiving / viewing / recording software.

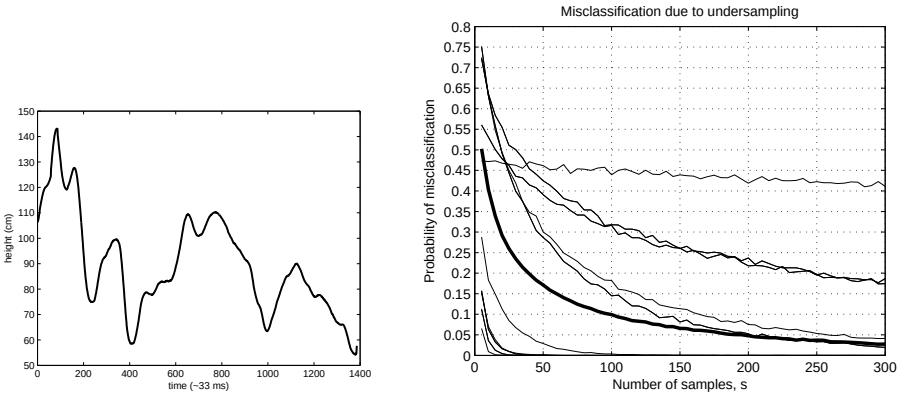


Figure 6: **Left:** Height of the ornithopter over time. **Right:** Misclassification due to subsampling, $H = 5$. Thick line: average error when sampling from the learned distributions. Thin lines: average errors for ten images from the height control experiment. See Section 4 for an explanation.

3.3 Results

The ornithopter successfully maintained a sane height during the experiment; it neither touched the floor nor got close to the ceiling. To get more insight into the height during flight, we reconstructed the trajectory of the ornithopter as follows. First, we determined the median x- and y-coordinates of the motion detected in both (external) camera images[5]. Then, we used the knowledge of the experimental setup in determining the X,Y-coordinate at which the rays backprojected from the cameras intersect in the office room. Finally, we calculated the corresponding height Z.

The left part of Figure 6 shows the reconstructed height (Z) over time. The ornithopter stayed within an acceptable bandwidth around the height classified as $h = 3$ ($\sim$90 cm). Please note that the motion detection from the different view points leads to rather noisy estimates. As a consequence, the actual Z-coordinate may deviate from the shown (smoothed) trajectory in the order of $\sim$20 cm. The height approximately varies between 0.55m and 1.45m. The reader may have a look at the original experimental videos online at http://www.delfly.nl/.

4 Discussion

The experiments showed that 100 samples sufficed for achieving height control. This represents only $\sim 0.56\%$ of the total number of possible image samples. In this section, we perform a preliminary analysis of how many of the errors are actually caused by taking fewer samples. For this analysis, we need to differentiate between two types of classification errors: (1) if the actual distribution is misclassified, it is referred to as a generalization error, and (2) if the actual distribution would be classified correctly, subsampling can still lead to an estimated distribution that is classified differently - a subsampling error. In the experiments, both types of errors were intermingled.

Here, we investigate the second type of error. The right part of Figure 6 contains the quantification of the subsampling errors. For ten randomly selected images from the experiments, we isolated the subsampling errors from the generalization errors as follows. First, each image was fully sampled and the resulting (actual) distribution classified as height h - which may differ from the actual height level at which the image was taken. Then, the image was classified multiple times with different numbers of samples. A classification was counted as a subsampling error if it was not equal to h. The ten thin lines show the relation between the subsampling error and the number of samples for the ten images. The thick line shows the same relation, but then for the case in which the texton distribution exactly corresponds to one of the learned distributions. Please note that random classification would result in an error of 0.8 (due to the 5 height levels). In addition, note that all misclassifications are confounded, while for height control a misclassification of $h = 1$ as $h = 2$ is less of a problem than a misclassification of $h = 1$ as $h = 5$. The main observation from Figure 6 is that a relatively low number of samples already considerably reduces the subsampling error. After that, there are

[5] We omitted video frames in which the ornithopter was only visible to one of the cameras.

diminishing returns: increasing the number of samples from 50 to 100 has a larger performance impact than increasing from 100 to 150.

5 Conclusion and future work

We conclude that appearance features can be extracted and processed fast enough for use in vision-based autonomous flight. The light-weight MAV *DelFly II* successfully used our height estimation algorithm to keep a sane height in an office room. To obtain successful height control, we did not prepare the room other than switching on the lights and putting the tables and chairs aside. The algorithm can be applied to almost any room, requiring only very limited training time ($< \sim$30 minutes).

In our future work, we want to investigate the generalization of appearance features to rooms not in the training set. To overcome the probable limitations of appearance features alone, we intend to combine our appearance-based algorithms with other algorithms based on, e.g., optic flow and accelerometers.

Acknowledgements

We thank Matthijs Amelink for helping us with the modification of the *Lama V4* toy helicopter.

References

[1] Spencer Ahrens. Vision-based guidance and control of a hovering vehicle in unknown environments. Master's thesis, MIT Aerospace Controls Lab, 2008.

[2] A. Beyeler, J-C. Zufferey, and D. Floreano. 3d vision-based navigation for indoor microflyers. In *2007 IEEE International Conference on Robotics and Automation, Roma, Italy*, pages 1336–1341, 2007.

[3] T.S. Collett. Insect vision: Controlling actions through optic flow. *Current Biology*, 12:615–617, 2002.

[4] M. Darms, P. Rybski, and C. Urmson. Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments. In *Proceedings of the 2008 IEEE Intelligent Vehicles Symposium*, pages 1197–1202, 2008.

[5] A.J. Davison and D.W. Murray. Simultaneous localisation and map-building using active vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002.

[6] N. Franceschini, J.M. Pichon, C. Blanes, and J.M.Brady. From insect vision to robot vision. *Philosophical Transactions: Biological Sciences*, 337(1281):283–294, 1992.

[7] J.S. Humbert and M.A. Frye. Extracting behaviorally relevant retinal image motion cues via wide-field integration. In *American Control Conference*, number 14–16, page 6 pp, 2006.

[8] F. Iida. Goal-directed navigation of an autonomous flying robot using biologically inspired cheap vision. In *Proceedings of the 32nd ISR (International Symposium on Robotics)*, 2001.

[9] C. Kemp. *Visual Control of a Quad-Rotor Helicopter*. PhD thesis, Churchill College, University of Cambridge, 2006.

[10] T. Kohonen. *Self-Organizing Maps, third edition, Springer Series in Information Sciences, Vol. 30*. Springer, 2001.

[11] J.H. Lim and Y. Liu. A queuing network model for eye movement. In *ACM Transactions on Computer-Human Interaction (TOCHI)*, volume 13, pages 37–70, 2006.

[12] F. Ruffier and N.H. Franceschini. Aerial robot piloted in steep relief by optic flow sensors. In *Intelligent Robotics and Systems (IROS 2008), Nice, France*, pages 1266–1273, 2008.

[13] L.J.P. van der Maaten. An introduction to dimensionality reduction using matlab, micc 07-07. Technical report, Maastricht University, the Netherlands, 2007.

[14] M. Varma and A. Zisserman. Texture classification: are filter banks necessary? In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), Madison, WI*, volume 2, pages 691–698. IEEE Computer Society, 2003.