Sub-sampling: Real-time Vision for Micro Air Vehicles

G.C.H.E. de Croon[1], C. De Wagter[2], B.D.W. Remes[2], and R. Ruijsink[2]*

Abstract

Small robotic systems such as Micro Air Vehicles (MAVs) need to react quickly to their dynamic environments, while having only a limited amount of energy and processing onboard. In this article, *sub-sampling* of local image samples is investigated as a straightforward and broadly applicable approach to improve the computational efficiency of vision algorithms. In sub-sampling only a small subset of the total number of samples is processed, leading to a significant reduction of the computational effort at the cost of a slightly lower accuracy. The possibility to change the number of extracted samples is of particular importance to autonomous robots, since it allows the designer to select not only the performance but also the execution frequency of the algorithm. The approach of sub-sampling is illustrated by introducing two novel, computationally efficient algorithms for two tasks relevant to MAVs: WiFi noise detection in camera images and onboard horizon detection for pitch and roll estimation. In the noise detection task, image lines and pixel pairs are sampled, while in the horizon detection task features from local image patches are sampled. For both tasks experiments are performed and the effects of sub-sampling are analyzed. It is demonstrated that even for small images of size 160×120 speed-ups of a factor 14 to 21 are reached, while retaining a sufficient performance for the tasks at hand.

*(1). Advanced Concepts Team, European Space Agency, (2). Micro Air Vehicle lab, Control and Simulation, Delft University of Technology. Contact: guido.de.croon@gmail.com

1 Introduction

Achieving autonomy of small robotic systems is one of the most challenging problems in the field of robotics. Small and light-weight robots can carry little energy, little processing, and few and inaccurate sensors. Nonetheless, in many cases the robots will have to react to their environment in real-time.

In this article, we focus on vision-based autonomous flight of Micro Air Vehicles (MAVs) as a case in point. An important requirement of the involved vision algorithms is that they should be computationally efficient. Since state-of-the-art vision algorithms are typically more directed towards performance than speed, it is desirable to find structural means for reducing their computational effort.

Sub-sampling is a broadly applicable method for reducing the computational effort of vision algorithms. If the algorithm involves the extraction of local image samples, sub-sampling implies that only a small subset of all the possible samples is used for the task. Typically, sub-sampling is associated to image resizing, in which a smaller image is constituted by sampling the pixels on an evenly spaced grid in the original image. Image resizing can lead to a considerable efficiency gain, but is limited by the amount of textural detail necessary for the vision task. Too small an image size may lead to the loss of important textural details.

In order to achieve a higher computational efficiency while retaining textural details, algorithms can extract local image samples larger than a single pixel from the original image. In particular, many studies focus on *active* sub-sampling, in which the information from the current sample is used to select the next [3, 31, 21, 43, 19, 30, 47, 11]. This can lead to large computational efficiencies, but also often creates a challenging Partially Observable Markov Decision Problem (POMDP). Such a POMDP is currently difficult to solve, and the mentioned studies either make strong assumptions on the task [21] or have to train a model for each different task [19].

Remarkably little vision research has been reported on plain *passive* sub-sampling. Two ways of passive sub-sampling are (1) random sampling, and (2) grid sampling. Random sampling has proven to be very effective to ameliorate the efficiency of machine learning techniques (cf. [32, 22, 2, 12, 34, 6]). In the context of vision, its application is more rare, although there are some examples in which the random sampling plays a central role (cf. [48, 42, 4, 16]). Sampling on a fixed grid is common in the sense that it is equal to image resizing if the granularity of textural features used by the vision algorithm remains at the pixel level (cf. [44]). However, the extraction of larger textural features located on a grid in the original image is more rare. Both passive sampling methods can be applied to a large group of algorithms that extract local features from images.

The ***main contribution*** *of this article is to advance computationally efficient vision algorithms for autonomous robots by making a case for the strategy of passive sub-sampling.* In sub-sampling, the number of samples permits a gradual exploration of the trade-off between the vision algorithm's accuracy and its computational efficiency. This property can be of particular importance for autonomous robots. Namely, it allows the execution of the algorithm at a required minimal frequency on almost any kind of processor. The price paid is a lower accuracy. However, as will be shown in this article, the method of sub-sampling has a graceful decay - allowing large gains in computational efficiency at the cost of only little accuracy. probability theory.

We illustrate the potential importance of sub-sampling for the autonomy of small robotic systems by performing two separate case studies. This leads to the ***two sub-contributions*** *of this article: (1) an efficient algorithm for offboard noise detection in images transmitted via an analog connection, and (2) an efficient algorithm for onboard horizon detection in images for estimating the pitch and roll of an outdoor flying MAV*[1]. Although the domains of the case studies are rather different, they both allow the investigation of sub-sampling. Most importantly, in both case studies it is demonstrated that a significantly higher computational efficiency comes at the cost of only a moderate loss in accuracy. Moreover, in both case studies the basic sampling strategies of random and grid sampling are compared. Finally, preliminary experiments are performed per case study to investigate the effects of *selective sampling*, in which not all of the image coordinates handed to the algorithm lead to the actual extraction of a sample. Selective sampling is shown to further reduce the number of extracted samples for a given accuracy.

The remainder of the article is organized as follows. In Section 2, we study the noise detection task. Subsequently, in Section 3, we investigate the horizon detection task. The usefulness of the sub-sampling approach for robotics is discussed in Section 4. Conclusions are drawn in Section 5.

[1]Both algorithms are publicly available at `http://www.bene-guido.eu/`.

2 Noise Detection

A well-known problem in the area of robotics is the noise in images transmitted by analog cameras. Besides thermal and white noise, images can also be corrupted by other types of noise. Often the cameras transmit on the 2.4 GHz channel, which is also used by WiFi. As a consequence, structured noise bands such as the one on the left in Fig. 1 can perturb the images. In addition to WiFi noise, sometimes the image receiver loses track of the image's start and end point, leading to a black bar that travels through the image with the top of the image being shown below the bar. It is needless to say that such noise-corrupted images are disastrous for many vision algorithms. While for many MAVs it is currently possible to avoid such noisy images by employing onboard vision processing (e.g., [9, 5, 37]), there are still platforms for which this is not possible due to weight restrictions. Examples of systems that currently still rely on analog transmission include flapping wing MAVs such as the recently introduced Nano Hummingbird of Aerovironment[2] and the DelFly II and DelFly Micro of Delft University of Technology [15]. As an illustration, the DelFly Micro weighs 3.07 grams, which implies that carrying a camera and transmitter is already a significant challenge. Successful noise detection would be a valuable asset for MAVs using analog transmission.

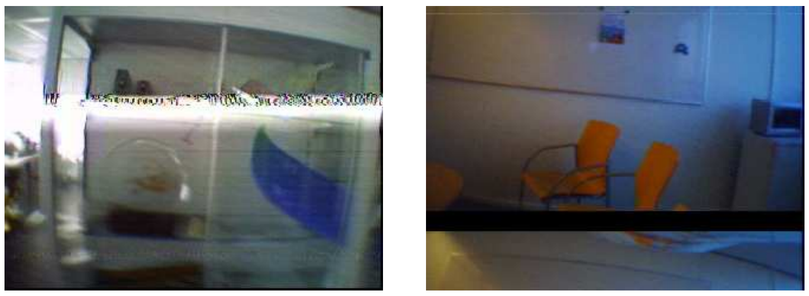

Figure 1: Example noisy / bad images. Left: WiFi transmits on the same frequency as the analog camera and introduces noise especially in a horizontal band. Right: the receiver loses track of the start and end point of the image.

Here, an algorithm is proposed to detect noisy lines as those in Figure 1. The algorithm first converts the images to grayscale. Then it exploits the fact

[2]`http://www.avinc.com/nano`

that subsequent pixels in noise-free image lines are normally well-correlated, while noisy lines have unrelated pixel values. The full-sampling version of the noise detection algorithm would scan each line in the image from left to right, while calculating the average correlation coefficient r between all pairs of subsequent pixels[3] v_x, v_{x+1}:

$$r(v_x, v_{x+1}) = \frac{\text{cov}(v_x, v_{x+1})}{\sqrt{\text{cov}(v_x, v_x)\text{cov}(v_{x+1}, v_{x+1})}}, \tag{1}$$

where:

$$\text{cov}(a, b) = E[(a - \overline{a})(b - \overline{b})], \tag{2}$$

with $\overline{a}$ and $\overline{b}$ the sample means of stochastic variables a and b. Finding multiple contiguous lines that have either a low correlation coefficient (lower than a threshold ϑ_r) or all black pixels is a strong indication of WiFi noise or black bars, respectively. An image is classified as noisy, if the number of noisy lines exceeds the threshold ϑ_l.

The computational complexity of the algorithm is approximately:

$$C \approx H(Wc), \tag{3}$$

where H is the number of image lines (height), W the number of pixels in an image line (width), and c is a constant representing the number of calculations performed per pixel in an image line.

The algorithm can be made computationally more efficient by employing sub-sampling. In particular, there are two places in the algorithm where sampling can save on computation. First, it can be applied to the selection of image lines. Instead of evaluating the correlation coefficient in every line, the algorithm can evaluate a limited number of lines $s < H$. If a line is considered noisy, the algorithm evaluates the contiguous lines to verify that it really concerns WiFi noise. As soon as the threshold number of noisy lines ϑ_l is reached, the algorithm stops evaluating the image (in the experiments, $\vartheta_l = 4$). Second, sub-sampling can be applied to the calculation of the correlation coefficient of a single line. The standard evaluation will make two complete passes over the image line, one for determining the average pixel value, and one for determining the covariances. Both these passes could be shortened to only a part of the line $n < W$ for further computational

[3]This implies that x ranges from the first pixel to the penultimate pixel in an image line.

efficiency. How much of the line is evaluated can be a fixed quantity or can be based on the observed pixels along the way.

In the following, first the experimental setup is discussed (Subsection 2.1). Afterwards, the sampling of image lines is investigated (Subsection 2.2), then the sampling of pixels in a line (Subsection 2.3), and finally the combination of the two (Subsection 2.4). The context of the investigation is formed by noise detection experiments on a data set of images made onboard MAVs employing analog 2.4GHz cameras. The experimental results will show the effect sub-sampling has on the computational efficiency and the classification performance of the noise detection algorithm. The performance of the algorithm is compared to the state-of-the-art (Subsection 2.5).

2.1 Experimental setup noise detection

The images used for the noise detection experiment come from two MAVs in various indoor environments with varying light conditions. Some images have been captured onboard a blimp, but most images come from the flapping wing MAV DelFly II. We used 100 ‘training’ images to select the thresholds used in the algorithm, $\vartheta_r = 0.70$ and $\vartheta_l = 4$. A separate collection of test images is used to evaluate the algorithm’s classification performance. The test set contains 84 noise-free images and 97 noisy images. Most of the images have been captured with a color camera. These images are converted to gray-scale for the experiment. The image size is 320×240 pixels. All MATLAB-scripts and a reduced image set can be downloaded from `http://www.bene-guido.eu/` for replication of the results mentioned in this section.

2.2 Sampling of image lines

Before we show the computational effort and classification performance of the noise classifier on the test set, a brief analysis is performed that provides an idea on what the effects will be of sampling the image lines.

2.2.1 Preliminary analysis

The starting points of the analysis are that (1) the number of lines affected by both WiFi noise and black bars is rather constant, and (2) the noisy lines are contiguous in the image. The first point allows us to assume a fixed probability for a line being noisy, while the second point allows us to assume that the detection of only one of the noisy lines is sufficient, since the rest of the noise will be adjacent to that line (This may take a few extra samples though).

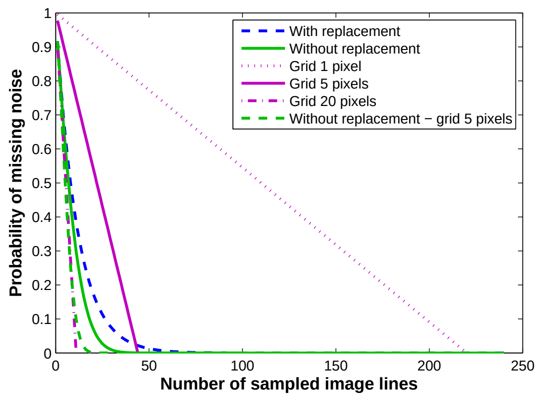

Figure 2: The probability of missing the WiFi noise when sampling image lines with the following strategies: random sampling with replacement (blue dashed line) and without replacement (green solid line), grid sampling (purple) with a 1-pixel grid (dotted), 5 pixel grid (solid), and a 20 pixel grid (dashed), and random sampling of 5 pixel grid lines (green dashed line).

Let us suppose that the number of lines affected by the noise is on average 20 for images of size 320×240 pixels. We investigate three different sampling strategies, which are explained below.

The first sampling strategy is *random sampling*. In random sampling, the probability of selecting a noisy line with uniform selection is $p_n = 20/240 = 1/12$, making the probability of missing it $p_m = 1 - p_n = 11/12$. When taking s independent samples with replacement, the probability of not finding any of the noisy lines is $p_m^{(s)}$. If a quarter of the lines is sampled, this probability is $p_M = p_m^{60} \approx 0.54\%$. If sampling is performed without replacement, the probability of missing all noise-affected lines is smaller: $p_M = \Pi_{i=1}^{s} \frac{(220-(i-1))}{240}$, which for a quarter of the lines gives $p_M \approx 7.46\ 10^{-5}\%$. Figure 2 shows the number of line samples s vs. the probability of not finding any of the noisy lines with replacement (dashed blue line) and without replacement (solid green line). In both curves, adding more samples has an increasingly smaller effect on the probability of missing the WiFi noise, suggesting that a relatively small number of samples may suffice. Of course, the difference between sampling with and sampling without replacement depends on the total number of samples, with equality when the total number of possible samples goes to infinite. Given that the noise-detection task has a relatively small total number of $S = 240$ samples, only sampling without replacement will be investigated.

The second sampling strategy is *grid sampling*, in which there is a fixed number of image lines between each sample. When sampling on a grid from the top of the image to the bottom, the number of samples before a detection depends on the location of the noise band. Assuming that the entire noise band is present in the image (20 lines) leads to the following probabilities. If the grid starts at image line 5 and has a step size of 5 pixels, the probability of a miss at the first sample is equal to the probability that the noise starts somewhere after the fifth line ($p_M = 215/220 \approx 0.977$). The probability of a miss at the second sample (image line 10) is equal to the probability that the noise starts after the second image line ($p_M = 210/220 \approx 0.955$). The probability then linearly decreases further to 0. For a 5 pixel grid, $p_M = 0$ at 44 line samples. The above reasoning is valid as long as the step size is smaller than or equal to the number of lines affected by the noise. For larger step sizes, there will be a remaining probability for a miss. Figure 2 shows the corresponding probabilities in purple for a grid with step size 1 (dotted), 5 (solid), and 20 (dashed-dotted).

From the analysis above one may conclude that at larger numbers of samples (smaller grid step sizes), a grid is likely to be less efficient than random sampling since the probability of missing the noise is higher. At smaller numbers of samples, grid sampling is more efficient than random sampling of image lines. This observation leads to the third sampling strategy: *random grid sampling.* In Figure 2 the green dashed line shows the probability of missing the noise if the image lines of a 5 pixel grid are randomly selected without replacement. Random sampling of grid lines results in a faster decrease of p_M than sequential sampling of grid lines. Please remark that it only reduces p_M at step sizes smaller than the number of noisy lines. At a step size equal to this number (assumed to be 20 in this analysis), both give equal results.

2.2.2 Results image line sampling

The three sampling strategies are applied to the noise detection task with a maximal height portion ranging from 0.05 to 1 with steps of 0.05. The actual height portions examined by the strategies are typically lower, since the sampling immediately stops if the number of noisy lines exceeds ϑ_l. The line sampling strategies with a random component are applied ten times to the test set. The results of the experiments are shown in Figure 3. The left part of Figure 3 shows the relation between the height portion examined (the sampled image lines divided by the image height) and the processing times of our MATLAB-scripts on an Intel Core i7 2.00 GHz processor for the case of random sampling (solid), grid sampling (dotted), and random sampling of grid lines (dashed). The right part of Figure 3 shows the relation between the height portion and the True Positive ratio (proportion of noisy images classified as noisy - blue) and False Positive ratio (proportion of noise-free images classified as noisy - red)[4]. Again random sampling has solid lines, grid sampling dotted lines, and random grid sampling dashed lines.

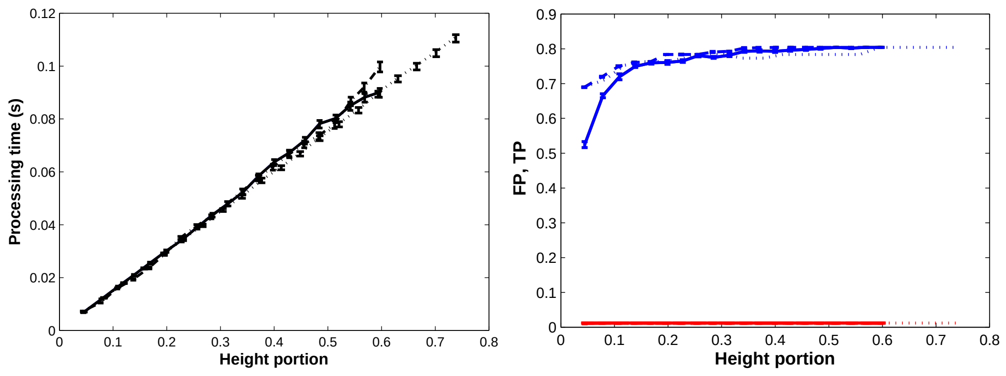

Figure 3: Results of sampling the image lines with random sampling (solid), grid sampling (dotted), and random grid sampling (dashed). Left: average computation time per image (and standard error bars) for different height portions. Images have size 320×240. Right: True Positive ratio (blue) and False Positive ratio (red) for different height portions.

The results shown in Figure 3 lead to three main observations. First, the computational effort increases roughly linearly with the height portion, as to be expected from Equation 3. The computational effort of the strategies that use random numbers is slightly higher than that of grid sampling due to the generation of these numbers. Grid sampling is the only one to reach a

[4]Please note that the performance of a classification method is also sometimes expressed in terms of its sensitivity and specificity. The first is equal to the true positive ratio TP, while the latter is equal to $1 - FP$, where FP is the false positive ratio.

height portion of 0.74, since it takes longer for the method to find the noise in a noisy image (as was expected from the analysis in Subsection 2.2.1).

Second, for all sampling strategies, increasing the height portion has the largest effect on the TP ratio for height portions lower than 0.20. For all methods, the FP ratio is constant over all height portions, while the TP ratio hardly rises after 0.20. In other words, *most of the performance is reached by a relatively small number of samples while leading to a quick execution of the algorithm.* For example, random grid sampling obtains a TP ratio of 0.78 at a height portion of 0.19, achieving a speed up of a factor ~ 5 at the cost of only 0.02 in TP ratio.

Third, randomly sampling the image lines in a grid gives the best results: it results in the highest TP ratio for all given height portions. While (sequential) grid sampling still performs almost as good, sampling at random locations significantly diminishes the TP ratio for small height portions.

2.3 Sampling of pixels in an image line

2.3.1 Fixed width portion

As explained, the full sampling algorithm makes two passes per image line: one for calculating the means of the pixel values $\overline{v_x}$ and $\overline{v_{x+1}}$, and a second for estimating the covariances necessary for determining $r(v_x, v_{x+1})$: $\mathrm{cov}(v_x, v_x)$, $\mathrm{cov}(v_{x+1}, v_{x+1})$, and $\mathrm{cov}(v_x, v_{x+1})$[5]. Sub-sampling can be applied to both passes, resulting in stochastic estimates of the mean and covariances.

In the experiments, four pixel sampling strategies are investigated. The first and the second sampling strategies extract a sequence of pixels from the image line. 'Random contiguous sampling' of a width portion of 0.05 signifies that $s_p = 16$ subsequent pixels are extracted from a random location in the line in order to calculate r. In 'fixed contiguous sampling', the first s_p pixels in the image line are sampled. The third and fourth sampling strategies extract separate pixel pairs from the image line. In 'random pair sampling' s_p pixel pairs are extracted from random locations in the image line without replacement. In 'grid pair sampling', the s_p pixel pairs are extracted at the locations of an evenly spaced grid.

The four pixel sampling strategies are applied to the noise detection task, while having the width portions vary from 0.05 to 1 with steps of 0.05. There

[5]In order to gain computational efficiency, one can assume $\overline{v_x} = \overline{v_{x+1}}$ and $\mathrm{cov}(v_x, v_x) = \mathrm{cov}(v_{x+1}, v_{x+1})$.

is no sub-sampling of image lines, implying a (maximum) height portion of 1. The pixel sampling strategies with a random component are applied ten times to the test set. The results are shown in Figure 4. The left part of Figure 4 plots the width portions vs. the processing time, while the right part of the figure plots the width portions vs. the FP and TP ratios. Random contiguous sampling is indicated with solid lines, fixed contiguous sampling with dotted lines, random pair sampling with dashed-dotted lines, and grid pair sampling with dashed lines.

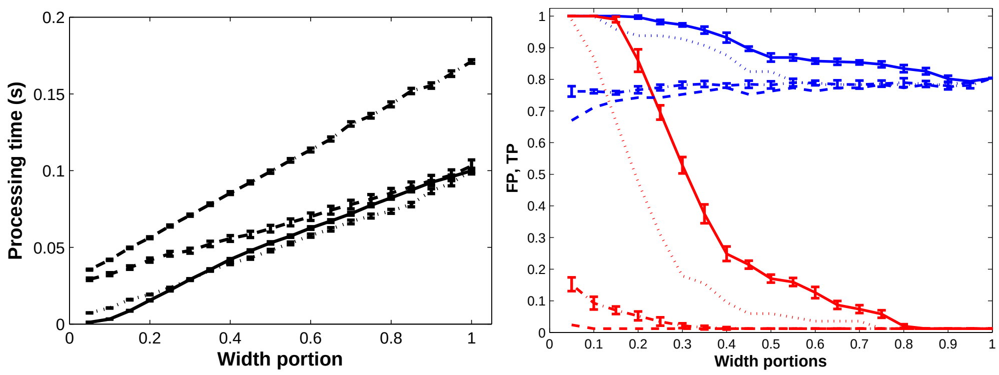

Figure 4: Results for random contiguous sampling (solid), fixed contiguous sampling (dotted), random pair sampling (dotted-dashed), and grid pair sampling (dashed). Left: average computation time per image for different width portions, for full sampling of the image lines. Right: True Positive ratio (blue) and False Positive ratio (red) for different width portions while fully sampling the image lines.

Figure 4 leads to three main observations. First, again as expected, for all methods the computational effort increases roughly linearly with the evaluated width portion. Second, for the contiguous sampling methods taking smaller width portions leads to higher TP and FP ratios. This means that they classify images more often as noisy. For the methods that sample pixel pairs, smaller width portions especially lead to a lower TP ratio. At first sight, the pixel pair methods look preferable, especially at lower width portions. However, it is interesting to note that the effect of contiguous pixel sampling is contrary to the effect of reducing the height portions (Figure 3), which may be of importance when applying random sampling both to the image and the width portions (see Subsection 2.4). Third, comparing the sampling schemes in terms of processing time leads to the conclusion that the pair sampling methods take more time than the contiguous sampling methods. This is mainly due to a lower number of false positives (the sampling of image lines stops if the noise threshold ϑ_l is reached). In addition, the methods involving random numbers take more time than their counterparts.

2.3.2 Selective sampling: variable width portion

Instead of taking a fixed number of samples from an image line, a *selective sampling* scheme can be employed in which the number of evaluated samples depends on the image line. Such a scheme should focus most of the sampling on the image lines for which it is necessary. One can achieve this by using probabilistic bounds. For example, Hoeffding's inequality [28] provides a probabilistic bound on the absolute difference between the sample average of independent identically distributed variables and the actual mean. Let $\{X_1, X_2, \ldots, X_N\}$ a set of i.i.d. variables with range R and mean μ and let their sample average $\overline{X_N} = \frac{1}{N}\sum_{i=1}^{N} X_i$. Hoeffding's inequality then states that with probability at least $1 - \delta$:

$$|\overline{X_N} - \mu| \leq R\sqrt{\frac{\log(2/\delta)}{2N}}, \tag{4}$$

The Hoeffding inequality is very general, but has as disadvantage that it scales linearly with R and cannot always provide sufficiently tight bounds. If there is a known bound on the variance, Bernstein's inequality can be used instead, resulting in significant improvements if this variance is small compared to the range R. Unfortunately, often there are no tight a priori bounds on the variance.

In [2] Bernstein's inequality is used to derive the *empirical Bernstein bound*, which makes use of the empirical standard deviation. The bound states that with probability at least $1 - \delta$:

$$|\overline{X_N} - \mu| \leq \overline{\sigma_N}\sqrt{\frac{2\log(3/\delta)}{N}} + \frac{3R\log(3/\delta)}{N}, \tag{5}$$

where $\overline{\sigma_N}$ is the empirical standard deviation of $\{X_1, X_2, \ldots, X_N\}$: $\overline{\sigma_N}^2 = \frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{X_N})^2$.

Assuming $\mathrm{cov}(v_x, v_x) \approx \mathrm{cov}(v_{x+1}, v_{x+1})$ and $\overline{v_x} \approx \overline{v_{x+1}}$ to be known or estimated, the correlation value r (Eq. 1) after sampling N pixel pairs in the line can be interpreted as $\overline{X_N}$ in Eq. 5. As a consequence, the empirical

Bernstein bound of r_N can be calculated and compared with the threshold value ϑ_r. If the lower bound is higher than ϑ_r or the upper bound is lower than ϑ_r, sampling can be stopped.

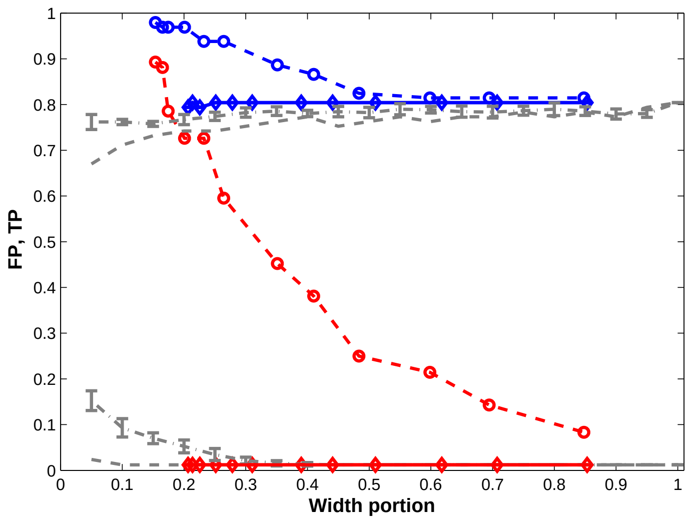

Figure 5: True Positive ratio (blue) and False Positive ratio (red) for different width portions for full sampling of the image height. The dashed lines with circle markers indicate the TP and FP ratio for empirical Bernstein sampling in which the bounds are applied symmetrically, the solid lines and diamond markers represent the results for an asymmetric application of the bounds. The grey lines are the TP and FP ratios for random pair sampling and grid pair sampling.

When employing the empirical Bernstein bound, one hopes to attain a higher performance with the same number of samples. Namely, the samples should be used for the image lines in which they matter most. However, applying the bound in a straightforward manner to the noise detection task does not give the expected results. This can be seen by looking at the dashed lines with circle markers in Figure 5, in which the results are plotted for $\delta = \{0.99, 0.95, 0.90, 0.80, 0.70, 0.60, 0.50, 0.40, 0.30, 0.20, 0.10, 0.05, 0.01\}$. In general, for a given number of sampled image lines, both the true positives and false positives are higher than those for fixed width portions. For reference, Figure 5 shows the results of fixed width portion sampling of individual pixel pairs with grey lines (see also Figure 4). The cause of the worse performance is the following: the δ-parameter determines the probability of misclassifying an image line, influencing both false positive and false negative classifications of image lines. Analysis shows that with the empirical Bernstein bound there are many false positive classifications of image lines and hardly any false negatives. In addition, for the noise detection task the impact of the false positive lines is larger than that of the false negatives, since only ϑ_l (false) positive line classifications suffice for classifying the image as noisy. This effect greatly influences the results especially if many image lines are evaluated. Choosing a lower δ reduces the number of false positive line classifications, but also increases the number of samples evaluated.

The skewed proportions of false positive and false negative classifications of image lines suggest the following remedy. The bound could be applied asymmetrically: the algorithm should only stop sampling if its lower bound is higher than ϑ_r, and not when its upper bound is lower than ϑ_r. Figure 5 shows the FP and TP ratios for the asymmetric application of the bound with solid lines and diamond markers. These results compare favorably with those when using fixed width portions (see the grey lines in Figure 5). *The asymmetrically applied empirical Bernstein bound results in TP and FP ratios comparable to those at a fixed width portion of* 1*, while only sampling on average a width portion of* 0.25 *(*$\delta = 0.80$*).*

Please note that the results for the asymmetric application of the empirical Bernstein bound form almost straight lines, implying that the resulting TP and FP ratio hardly depend on δ. Analysis of the results show that increasing δ from 0.01 to 0.99 only results in a few more false negatives (noisy lines classified as clean). On the level of images, these few more line misclassifications have no effect on the TP and FP ratio. The fact that there are not more false negatives can be explained by the bound not converging to 0 when δ converges to 1. Instead, Equation 5 shows that it converges to $\overline{\sigma_N}\sqrt{\frac{2\log(3)}{N}} + \frac{3R\log(3)}{N}$. Apparently this bound still ensures a safe enough margin for the noise detection task.

A last remark on the empirical Bernstein bound is that its calculation requires processing time as well. Therefore, in a final implementation it would be good (as in [34]) to make use of geometric calculation of the bounds (with increasing spaces in between calculations).

2.4 Combined sampling of lines and pixels

Finally, the sampling of image lines and pixels in an image line can be combined. The best results were obtained with random grid sampling of image lines and fixed contiguous sampling of pixels. Figure 6 shows multiple 3D-plots, with the height portion on the x-axis, the width portion on the y-axis, and respectively the TP ratio, FP ratio, and mean processing time on the z-axis.

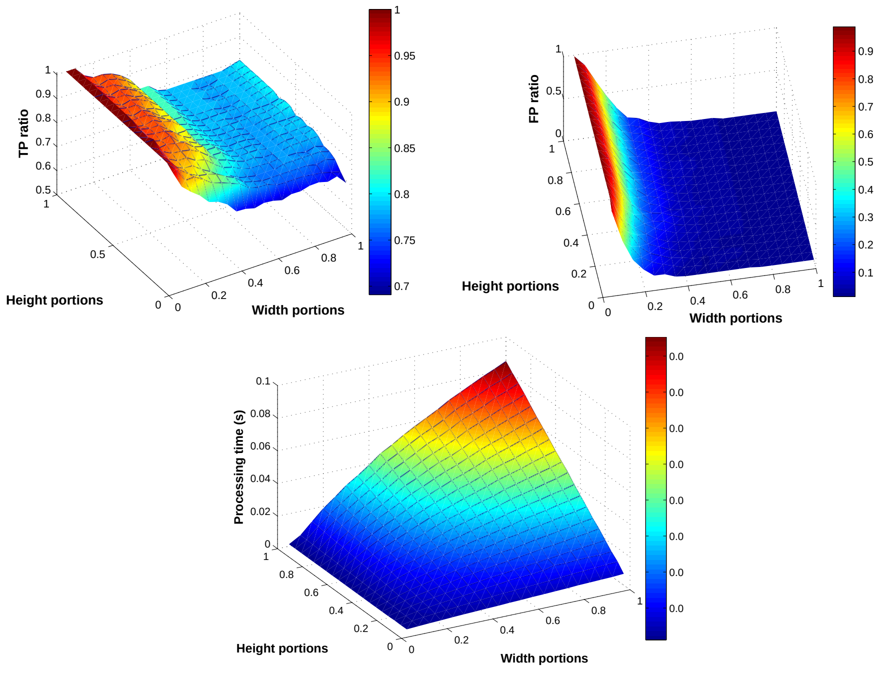

Figure 6: The relation between the investigated height portion, width portion, and: TP ratio (top left), FP ratio (top right), mean processing time (bottom).

The selection of the width portion and height portion used in the algorithm depends on the demanded performance and execution time / frequency. For example, in the autonomy experiments performed with the DelFly II, one should not want an FP ratio higher than 0.10, since too many images are then discarded. In addition, the execution should be so fast that it leaves room for other computer vision algorithms to be run in parallel. So let's presume a demanded execution frequency $\geq$ 100 Hz. One of the settings that satisfies these demands is a height portion of 0.10 and width portion of 0.40, with a TP ratio of 0.82, an FP ratio of 0.04 and an average processing time of 0.0060s (execution frequency of $\sim$ 167 Hz). A few results of this setting are shown in Figure 7. The green lines are sampled lines that were found to contain no noise, the red lines are classified as noisy. The right image shows a false positive: an image labeled as a noiseless image that was classified as noisy by the algorithm[6]. Zooming in on the image shows that the 'noisy' lines indeed contain some textural artifacts, but not strong enough to be labeled as noisy. Please note that a full sampling scheme would have implied a TP ratio of 0.80, an FP ratio of 0.01, and a processing time of 0.085 s (execution frequency of $\sim$ 12 Hz). *As a consequence of sub-sampling, a speed-up of* $\sim$ 14 *is reached with as consequence slightly more positive classifications.* The FP ratio increases with 0.03 while the TP ratio increases with 0.02 in comparison to full sampling.

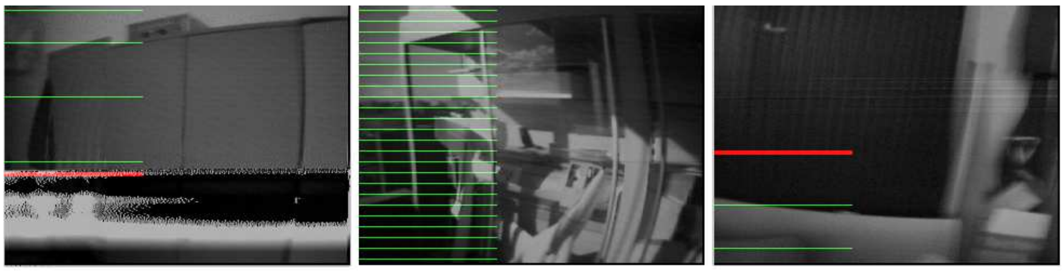

Figure 7: Noise detection results with a height portion of 0.10 and width portion of 0.40. The pixels sampled by the noise detection algorithm are colored. Green pixels in an image line imply that subsequent pixels correlated well, i.e., that the image lines were found to contain no noise. Red pixels did lead to a classification of the image line as noisy. Left: a true positive. Center: a true negative. Right: a false positive.

2.5 Comparison with state-of-the-art

Of course, the algorithm's efficiency is only interesting if its performance is reasonable in comparison with existing methods. Therefore, the performance

[6]In the text, 'labeling' is used for the process in which a human assigns ground truth values to instances. The computer then tries to match the labels by 'classifying' the instances in an automated manner.

of the resulting efficient noise detection algorithm is compared with the state-of-the-art in the literature. Byrne and Mehra [7] employ a supervised learning approach that consists of two stages. In the first stage the most noisy part of the image is selected. In the second stage various features are extracted from the selected image part (a horizontal band in the image). For the first stage, the entire image is filtered with steerable filters. The second stage involves features such as the principal components of a $\mathcal{C}_b\mathcal{C}r$-histogram and statistics on the calculated filters (mean, standard deviation, etc.). The algorithm of [7] is applied to the same MAV data set as the sub-sampling algorithm. Since it concerns a supervised algorithm, a 10-fold test is performed with the images. This leads to a performance of TP $= 0.76$ ($\sigma = 0.18$) and FP $= 0.23$ ($\sigma = 0.21$). These results are slightly worse than the results reported in [7], which was TP $= 0.81$ for FP $= 0.10$. Differences may be due to the different data set, or slight differences in implementation. The processing time of our implementation of the method in [7] is on average 0.11 s per image[7]. This corresponds to an execution frequency of ~ 9.1 Hz.

In summary, sub-sampling leads to a speed-up of a factor ~ 14, resulting in an algorithm that can execute at a frequency of 167 Hz, while performing at least as well as recent methods from the literature (executing at a frequency of ~ 9.1 Hz).

3 Horizon Detection

In order to obtain a broader validation of sub-sampling, we now turn our attention to a different task: horizon detection for pitch and roll estimation of outdoor MAVs [24, 46, 13, 33, 25, 8, 39, 49, 45, 35]. Typically it is assumed that the MAV is rather high in the sky, so that the skyline can be assumed equal to the horizon line. Each image is first segmented into sky and non-sky regions. The segmentation is then used for estimating which line separates the classes of sky and non-sky as well as possible. The slope of the line is related to the MAV's roll angle, while the vertical offset in the image is determined by the MAV's pitch angle.

Interestingly, most horizon detection algorithms use local features for the segmentation. Consequently, *it is straightforward to obtain significant*

[7]Some of the MATLAB code could be further optimized to gain a little bit of time, but the script already uses C-files for the convolution of image with the steerable filters (C-code is executed much faster than MATLAB code).

speed-ups by applying sub-sampling to the task of horizon detection. The remainder of this section is organized as follows. In Subsection 3.1, the experimental setup for the horizon detection task is explained. Subsequently, the effects of sub-sampling are investigated in Subsection 3.2. The resulting sub-sampling method is compared to the state-of-the-art in Subsection 3.3. Finally, the pitch and roll estimation algorithm is tested on a fixed wing MAV in Subsection 3.4.

3.1 Experimental setup horizon detection

In this subsection, first the method for classifying image coordinates as sky or non-sky is explained. Subsequently, the algorithm for learning a linear separator is introduced. Finally, the image set used in the experiments is discussed.

3.1.1 Sky Segmentation

The sky segmentation is performed with a decision tree, developed in [20]. The decision tree has been learned with the *C4.5* algorithm [38] on the basis of features extracted from the relatively large and publicly available *labelME* database [41][8] (7456 images containing an entity labeled as 'sky'). From the images in the training set 34 different features have been extracted. The detailed explanation of the features falls outside of the scope of this paper and can be found in [20]. Here, it is only relevant to realize that all features can be extracted locally, such as the $\mathcal{YC}_b\mathcal{C}_r$-value of a pixel, and that some of them involve texture, such as the mean absolute distance between a pixel's value and the values of its 8-pixel neighborhood. A decision tree has been selected on the basis of the camera and processing system available onboard the MAV used for the experiments: a Surveyor BlackFin camera. The decision tree implemented on the Surveyor BlackFin employs 5 of the 34 possible local features for segmentation. Extracting the 5 local features from one pixel coordinate (x, y) results in one local 'sample'. The performance of the decision tree on sky / non-sky classification compares favorably to most methods from the literature [20].

[8] `http://labelme.csail.mit.edu/`

3.1.2 Horizon Estimation

Horizon estimation involves the estimation of a linear separator of sky and non-sky pixels in the image[9]. Finding a linear separator on the basis of positive (sky) and negative (non-sky) samples is a standard problem with well-known solutions. One solution to finding the horizon line is to employ a linear algebraic formulation and determine a least-squares solution (cf. [35]). In this article, we will focus on using a linear perceptron to separate the sky and non-sky pixels. The main motivation for this is that the incremental perceptron learning lends itself well to achieving further speed-ups with selective sampling (see Subsection 3.2.3).

The weights w of the perceptron determine the classification as follows:

$$t_i = \mathrm{sgn}(w^\top A_i), \tag{6}$$

where A_i is a 3×1 column vector that represents the image coordinate as $(x_i, y_i, 1)^\top$, w is a 3×1 vector with the horizon line parameters, and 'sgn' a sign-fuction that is 1 if its argument is ≥ 0 and -1 otherwise. The weights w can be adapted on the basis of a single sample according to the delta-rule:

$$w_{i+1} \leftarrow w_i + \Delta w_i, \tag{7}$$

$$\Delta w_i = \gamma (t_i - c_i) A_i, \tag{8}$$

where t_i is the classification by the perceptron (Equation 6), c_i the classification by the sky segmentation, and γ is the learning rate. Equation 8 shows that the weights are only adapted if the sample is misclassified ($t_i \neq c_i$). The weights are intialized as $w = (0, -1, \frac{H}{2})$, implying prior pitch and roll angles of $0°$.

One of the problems of the straightforward application of the delta rule is that the order in which the samples are presented influences the decision boundary. For this reason the pixels should not be handed to the perceptron from the top left to the bottom right of the image. In addition, the separation line does not necessarily converge when evaluating more and more of the pixels. To reduce the aforementioned effects, in the implementation the final weights w' are not equal to the weights w_N. Instead, the final weights are taken to be the average of the weights at set intervals

[9]Throughout the text we assume that the camera is a perfect linear camera, or that images have first been undistorted before any further processing.

$w' = \overline{w_j}, j \in 0, u, 2u, \ldots, N$, where in the current experiments, $u = \frac{1}{10}N$. This weight averaging considerably improves the results (cf. [1]).

The final weights w' are then transformed to determine the estimated pitch and roll angle. If $w'(2) \neq 0$ (the horizon line is not vertical in the image), the slope of the horizon line is $a = w'(1)/w'(2)$. The height in pixels of the line at $x = 0$ is then $b = w'(3)/w'(2)$. The equation of the horizon line in image coordinates, $y = ax + b$, is used for estimating roll ($\hat{\phi}$) and pitch ($\hat{\theta}$). $\hat{\phi}$ is simply the angle of the line with the x-axis. $\hat{\theta}$ depends on the y-coordinate of the line at half of the screen. If the center coordinate of the image is (0,0) with positive coordinates up, $\hat{\theta} = y\text{FOV}_v/H$, where FOV_v is the field of view of the camera lens in the vertical direction.

3.1.3 Image set

For the horizon detection experiments, an image set with ground truth horizon lines was created. We have labeled the horizon lines in a collection of 116 images taken from a fixed wing MAV flying at altitudes between a few meters and 200m. The images have been captured with the camera that will be used onboard the MAV in the real-world experiments (see Subsection 3.4) and are of the size $W \times H = 160 \times 120$ pixels. Figure 8 shows some example images. Please note that the sky line as detected by the algorithm is not necessarily equal to the horizon line. At very low altitudes, the ground truth horizon lines are typically situated below the sky line, implying that a visual routine on the basis of sky classification will always have a slight error in its pitch estimate. Also please remark that the light conditions vary considerably over the image set.

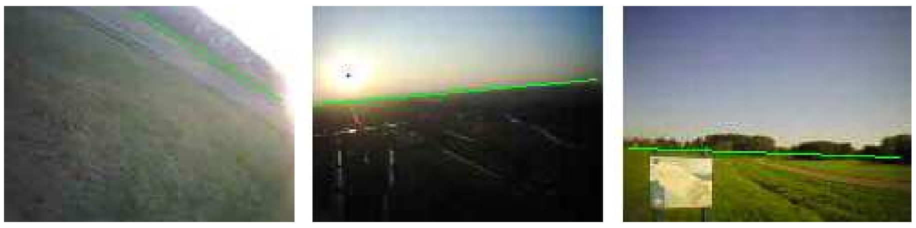

Figure 8: Example images used in the pitch and roll estimation experiments. The green lines are the ground-truth horizons. Please note that at low altitudes they are situated below the sky line, implying that a visual routine on the basis of sky classification will always have an error.

3.2 Effects of Sub-Sampling on Pitch and Roll Estimation

The standard way of determining the horizon line in an image would be to first segment the entire image and then use the resulting classifications for estimating the parameters of the sky / non-sky separation line. In that case, the computational complexity of pitch and roll estimation is approximately:

$$C \approx WH(F + S) + L + T, \tag{9}$$

where W and H are the width and height of the image, F is the computational effort spent in feature extraction, S the effort spent on segmentation (classification of a feature vector as sky / non-sky), L is the cost involved in determining the linear separator (typically also dependent on the number of pixels involved, WH), and T is a negligible cost of transforming the line parameters to estimates $\hat{\phi}$, $\hat{\theta}$.

Here we investigate the effects of only classifying a small subset s of all possible pixels in an image, $s \ll WH$, and using these classifications for determining the horizon line. The image size on which we focus is $W \times H = 160 \times 120$, since this is the image size used in the final implementation of the algorithm onboard the MAV (see Subsection 3.4). In the experiments, a border of 10 pixels is used, to avoid the worst effects of vignetting and lens distortion. This results in an effective total number of $140 \times 100 = 14000$ pixels.

On the basis of Equation 9, significant speed-ups can be expected if a few hundred pixel classifications suffice for estimating the horizon line. The speed-up can be proportionate to WH/s, assuming that either (a) L is negligible in comparison with $WH(F+S)$ (in which case it can be more or less ignored), or (b) L is dependent on the number of pixels s (in which case the computational costs will be reduced with a similar factor). In the case of perceptron learning, L increases linearly with s. Finally, please remark that larger image sizes would further improve the relative speed-up attained.

3.2.1 Accuracy

This subsection will focus on an empirical investigation of the effects of sub-sampling on the accuracy of the pitch and roll angle estimates ($\hat{\theta}$ and $\hat{\phi}$). In order to fly small MAVs, pitch and roll measurements are primordial. The accuracy required by a certain platform depends higly on its characteristics.

While high performance gliders are very sensitive for pitch, low aspect ratio planes, delta wings, flapping wing vehicles and some other low Reynolds number MAVs can still be flown accurately with pitch and roll measurement errors of up to 10 degrees. Especially slow and relatively constant errors like a pitch angle error due to the sky-line / horizon-line offset are easily compensated for by the outer loop or navigation loop PID controller that in essence automatically finds the trim angle.

Two sampling strategies are investigated: sampling at random locations and sampling on a grid. The grid is made so that the number of vertical and horizontal grid points (g_v, g_h) are proportional to the dimensions of the image area in which samples are taken. Since no sampling is performed in the border of 10 pixels, this proportion is $\frac{g_h}{g_v} = \frac{140}{100}$. The horizontal and vertical step sizes of the grid are then determined so that the grid spans the entire image area available for sampling. As mentioned, the success of perceptron learning depends on the sequence of the samples and sampling from the top left to the bottom right leads to bad results. Therefore, in the case of grid sampling the grid locations are sampled in a random order. For both sampling strategies each sample is extracted and subsequently evaluated by the segmentation decision tree. The line parameters are then updated with the perceptron learning rule of Equation 6. After all s samples have been processed, the weights w' are transformed to the estimated pitch and roll angles. The errors between the estimated and actual angles are stored ($e_\phi = \phi - \hat{\phi}$ and $e_\theta = \theta - \hat{\theta}$). As a final measure of interest, the mean absolute error is determined ($\overline{|e_\phi|}$ and $\overline{|e_\theta|}$). Due to their random elements, the mean errors of both sampling strategies can vary. Therefore each sample strategy is applied to the image set 10 times.

Figure 9 shows the relation between the number of samples and the mean absolute errors in the roll and pitch estimates. The numbers of samples s ranges from very few samples to full sampling: $s \in \{35, 140, 560, 1260, 2240, 5040, 8960, 14000\}$. These numbers have been chosen such that they lead to a grid with proportion $\frac{g_h}{g_v} = \frac{140}{100}$, where $g_v, g_h \in \mathbb{N}$. The solid lines represent the results for random sampling, while the dashed lines represent the results for grid sampling. The mean absolute pitch error is shown in red, while the roll error is shown in blue.

The main observation to be made from Figure 9 is that *extracting only 560 samples ($\sim 4\%$ of the total) is sufficient to obtain an average absolute error only slightly higher than the one with the full sampling of all 14000*

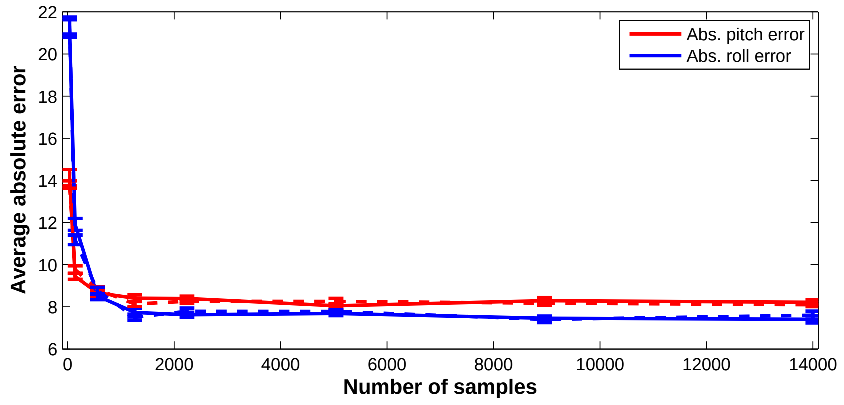

Figure 9: Average absolute errors (and corresponding standard errors) for the pitch (red) and roll (blue), for random sampling (solid) and grid sampling (dashed).

samples. Furthermore, it is of practically no use to extract and process more than 1260 samples.

One can intuitively understand the observation above by looking at the spatial distributions that result from sub-sampling. Figure 10 shows the application of the decision tree and the linear perceptron to an example image, with different numbers of samples, $s = \{35, 560, 1260\}$ (columns) and for the two different sampling strategies (rows). Red circles are samples classified by the decision tree as ground, blue crosses are samples classified as sky, the green line is the ground-truth horizon line, and the yellow line is the separation line of the linear perceptron. The figure illustrates that $s = 560$ already provides ample evidence on where the horizon line should be located. Please remark that for finding the linear separator it is important to have sufficient samples close to the horizon line. Both sampling methods seem to satisfy this criterium, especially for numbers of samples $s \geq 560$. Indeed, Figure 9 shows that random sampling and grid sampling have very comparable performances. At low numbers of samples such as $s = 35$, there can be few samples close to the horizon line. In combination with the perceptron learning, the resulting pitch and roll angles can then become quite erratic.

For the onboard implementation the horizon estimation algorithm is extended in one important way; 90% of the samples is used for determining the final weights, while 10% of the samples are used for estimating how well the line separates the two classes. The portion of misclassifications is a measure

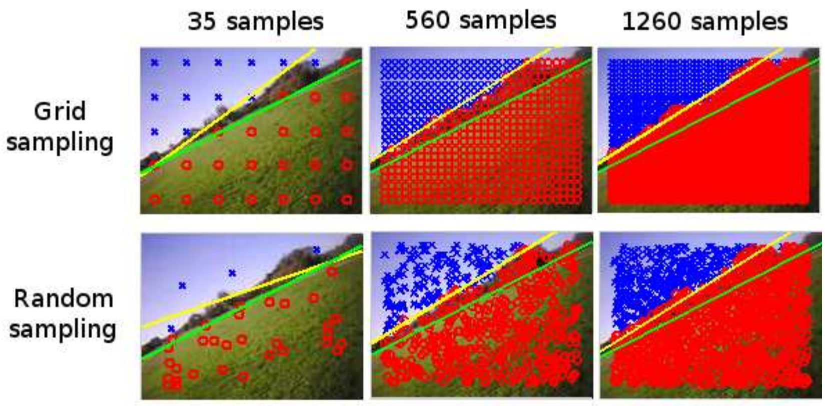

Figure 10: Application of the decision tree and the linear perceptron to an example image, with different numbers of samples. From left to right, $s = \{35, 560, 1260\}$. The top row shows the results for grid sampling, the bottom row for random sampling. Please remark that standard full sampling would involve the extraction and classification of 14000 samples.

of how reliable the pitch and roll estimates are. Unreliable estimates can be discarded, leading to a lower error. In the experiments, a threshold is used of 20% of the testing samples. So when extracting 1000 samples, 900 samples are used for determining the horizon line and 100 samples are used for determining the uncertainty. If the horizon line misclassifies more than 20 samples, the horizon estimate is discarded.

The left part of Figure 11 shows the errors in pitch (red) and roll (blue) for sample sizes $s \in \{140, 315, 560, 875, 1260\}$ with random sampling (solid) and grid sampling (dashed). The right part of the figure shows the corresponding portions of discarded images. With the error checking mechanism the sub-sampling methods obtain a lower error at these low numbers of samples, while discarding an acceptable portion of estimates. However, in the right part of Figure 11 one can observe that the portion of discarded images increases more and more towards fewer samples. At $s = 140$, close to 40% of the images are discarded, also reducing the execution frequency of the algorithm by a similar amount. Both sampling strategies give similar results. Still, for most sample sizes random sampling has a somewhat lower average absolute error than grid sampling, discarding slightly more estimates.

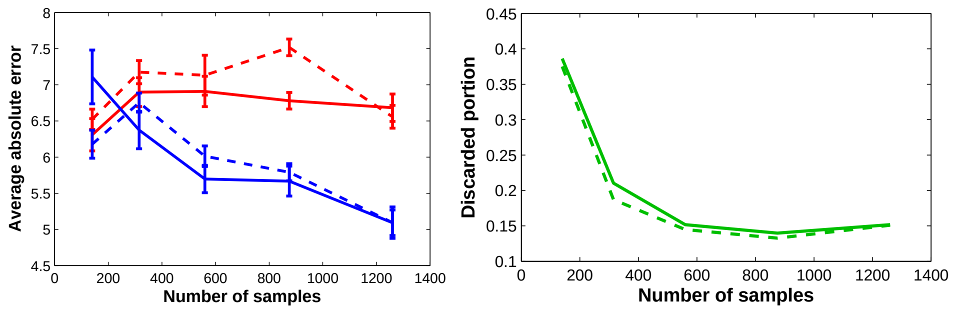

Figure 11: Left: Average absolute errors (and corresponding standard errors) for the pitch (red) and roll (blue), for random sampling (solid lines) and grid sampling (dashed lines). Right: portion of discarded horizon lines.

3.2.2 Computational Effort

The results in the last subsection suggest that a small number of samples already suffices to obtain the required accuracy. Here the implications for the computational effort are investigated. Figure 12 shows the relation between the number of samples and the average processing time per image, for the case of random sampling. The results for grid sampling are similar. The processing times are shown of the different parts of the horizon estimation algorithm: feature extraction (blue), feature vector classification (green), and the optimization of a linear separator (red). The aggregrated processing time is also shown (black).

Figure 12 shows that the computational effort increases approximately linearly with the number of samples, as expected from Equation 9. Furthermore, the classification is computationally the most expensive part and finding a linear separator is the least expensive part. While full sampling results in a processing time of 1.86 s, the use of 560 samples leads to a processing time of 0.09 s. This is a speed-up of a factor ∼21. Please note that the MATLAB implementation of the decision tree always extracts for each pixel all features present in the tree, and that an onboard implementation should extract only those features that are actually tested for in the sample's path in the decision tree.

Figure 13 shows the relation between the number of samples and the execution frequency of the entire algorithm in MATLAB (solid black line) and on the Surveyor BlackFin camera (dash-dotted blue line). The corresponding C-code for the segmentation and horizon estimation can be found at `http:`

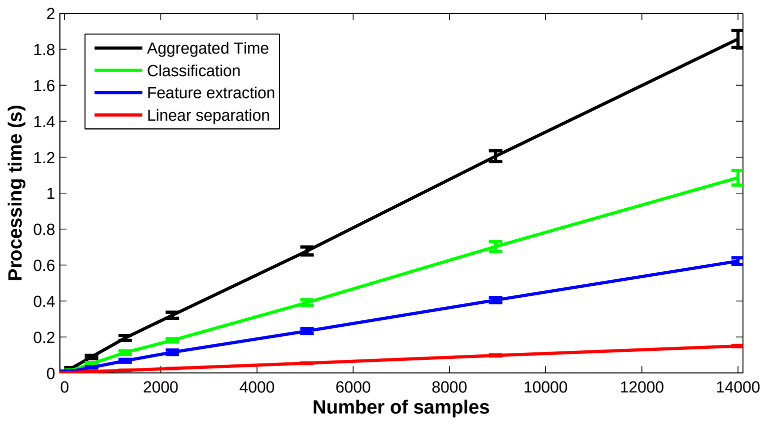

Figure 12: Average processing times (and corresponding standard errors) of the different parts of the pitch and roll estimation algorithm: feature vector classification (green), feature extraction (blue), and the optimization of a linear separator (red). The aggregrated processing time is also shown (black).

`//www.bene-guido.eu/`.

The main observation from Figure 13 is that the algorithm runs faster on the Surveyor BlackFin than on the 2.00 GHz laptop. The main reason for this is that the algorithm running on the laptop is implemented in MATLAB, while the algorithm running on the BlackFin is implemented in C. MATLAB-code is normally much slower than C-code.

3.2.3 Selective sampling: variable number of samples

As for the noise detection task, one can further limit the number of samples extracted from the image by accepting a variable number of samples. In contrast to the estimation of the correlation coefficient r in the noise detection task, the goal of sampling in the horizon detection task is not to determine an average value. In fact, different samples may carry different amounts of information on the horizon parameters w.

The problem setting of passive sampling in the horizon detection task lends itself well to application of a selective sampling algorithm from the field of active learning (cf. [10]). Such an algorithm learns a classifier on the basis of a sequence of unlabelled samples A_i. It can query a corresponding label c_i by paying a fixed cost u. The goal of the algorithm is to learn a

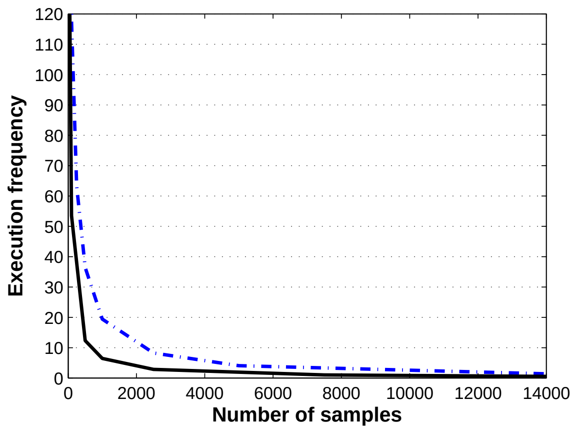

Figure 13: Execution frequency of the algorithm in MATLAB (solid black line) and on the Surveyor BlackFin (blue dashed-dotted line).

good classifier while using as few samples as possible. This problem setting corresponds well to the setting of the horizon detection task. Indeed, the main computational effort lies in the feature extraction and sky classification, and it would therefore pay off to be selective in the samples A_i that are evaluated.

There are several algorithms for selective sampling with a linear perceptron (cf. [10, 40, 14]). In our experiments, we employ the selective sampling algorithm of [10] since it is known to perform well on problems that are not (perfectly) linearly separable [36]. The central idea behind this method, referred to as *CB* in this article, is that samples close to the classification boundary are more informative than samples far away. Indeed, one can expect that samples extracted far away from the horizon line typically do not result in adjustments of the horizon line's parameters.

More formally, the method receives a sequence of s coordinates $A_i = (x_i, y_i, 1)$, $i \in \{1, 2, \ldots, s\}$. Per sample, it determines $p_i = w_i^\top A_i$ and draws a random number q from a uniform distribution in the interval $[0, 1]$. If $q \leq \frac{b}{b+|p_i|}$, then the algorithm queries the label $c_i \in \{-1, 1\}$ and compares it with $t_i = \mathrm{sgn}(p_i)$. It then executes the standard perceptron update (Equation 7): $w_{i+1} \leftarrow w_i + \gamma(t_i - c_i)A_i$. Else, if $q > \frac{b}{b+|p_i|}$, it justs moves on to the next sample. The parameter b determines the efficiency of the algorithm. For small values of b, the probability of selection is largely dependent on the magnitude of p_i that is proportionate to the distance between the coordinate and the classification boundary. If $\lim_{b \to \infty}$ the algorithm will select all samples, making it equal to standard perceptron learning.

The above-described selective sampling scheme bases its decision for extracting and classifying a sample on $|p_i|$, and is therefore insensitive to the predicted class t_i of a sample. As a consequence, the algorithm may perform badly on skewed data sets, which in the case of the horizon detection task implies images that have little sky or little ground. Coping with skewed data sets is a well-known problem in machine learning [27] and has even been investigated in the context of selective sampling [50, 23]. However, to our knowledge, there is no standard way in which the selective sampling method of [10] can cope with skewed data sets. In our experiments, an adaptation of the CB-method is tested that changes the selection criterium to:

$$q \leq \left(\frac{b}{b + |p_i|} \right) \left(2 \frac{\sum_{j=1}^{i-1} \delta\{t_j, -t_i\}}{i - 1} \right) \tag{10}$$

, where $\delta\{t_j, -t_i\}$ is the Kronecker delta, which is 1 if $t_j = -t_i$ and 0 otherwise. The added term is equal to twice the proportion of samples encountered of the other class. If the other class is sampled more often, the probability of selecting the sample is larger and viceversa. The factor 2 makes the second term equal to 1 if the classes are sampled equally, resulting in the standard algorithm [10]. The modified algorithm will be referred to as *CB'*.

Figure 14 shows the results for applying CB (dashed lines), and CB' (dotted lines). Random sampling is employed with various numbers of samples, $s = \{100, 500, 1000, 2500, 5000, 7500, 14000\}$. Since the evaluation of p_i hardly costs any computation time, the figure shows the corresponding average absolute errors at the number of samples that were actually extracted and classified. For reference, the figure also shows for similar sample numbers the results of normal sampling, in which each sample is extracted and classified (solid lines).

As expected, selective sampling leads to better performances at lower numbers of samples than plain random sampling. In addition, CB' outperforms CB for $s > 1800$. For sample numbers $s \leq 1800$, CB' gives better pitch angle estimates (in the order of $\sim 2^\circ$) while CB gives better roll angle estimates (in the order of $\sim 1^\circ$). The results therefore also show the advantage of taking into account skewed proportions of sky and ground pixels.

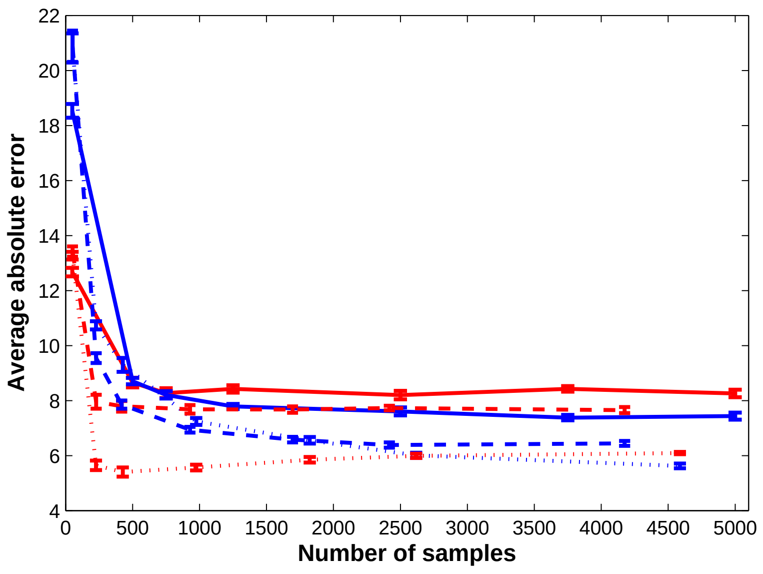

Figure 14: Average absolute errors in pitch (red) and roll (blue) for standard random sampling (solid), selective sampling method CB (dashed), and the skewed selective sampling method CB' (dotted).

3.3 Comparison with state-of-the-art

In this subsection, we compare the performance and processing time of the introduced sub-sampling algorithm with two methods from the literature. Each method consists of a combination of an image segmentation technique from the literature with the linear perceptron described in Section 3.1.2.

The first method is inspired by [39] and performs image segmentation on the basis of Hoiem's segmentation algorithm [29]. The segmentation algorithm starts with the superpixel (over-)segmentation [26]. Then it evaluates different combinations of superpixels by extracting many local features within each segment. Boosted decision tree learners select features from a large feature set, including color, texture, shape, and geometric features. The goal of these learners is to classify the superpixels in the images into a number of classes including the sky. In [39], it was found that varying the superpixel algorithm's K parameter [26] led to better sky classification. The K-parameter governs the resolution of the segmentation, with smaller values of K resulting in fewer and larger superpixels.

The second method is adopted from Thurrowgood et al. [45]. It learns

Table 1: Processing times and performances of various methods on the image set. The method with the best performance has been typeset in bold.

Method	Processing time (s)	$\overline{\lvert e_\phi \rvert}$	$\overline{\lvert e_\theta \rvert}$
Hoiem 2005, $K = 1$	1.19	12.0	14.2
Hoiem 2005, $K = 10$	1.13	10.8	14.3
Sub-sampling ($s = 500$)	**0.08**	**8.5**	**8.6**
Sub-sampling ($s = 250$)	0.04	10.1	9.2
Thurrowgood 2009	0.02	9.9	9.9

a Fisher's linear discriminant that linearly transforms the $\mathcal{RGB}$ image space so that the sky and non-sky class are well-separated. We have tried out the parameters mentioned in [45], but obtained better results by training the linear discriminant on the LabelME data set mentioned in Subsection 3.1.1. The latter results are reported in this article.

The methods are applied to the test set without discarding images. Moreover, the sub-sampling method uses random sampling of a fixed number of samples. Table 1 shows the absolute average errors for all methods. It leads to two observations. First, the best performance is obtained by the novel horizon detection algorithm. Second, the fastest algorithm is the one of Thurrowgood et al. [45], which uses one feature that is fast to extract. The sub-sampling method uses the decision tree explained in Subsection 3.1.1, obtaining its better accuracy by extracting more complex features. With $s = 500$, the sub-sampling method takes 0.06s more processing time than Thurrowgood's method. Without sub-sampling, the decision tree would have taken 1.84s more processing time.

3.4 Experiment on a Fixed Wing MAV

In this subsection an experiment on a fixed wing MAV is performed to show that the pitch and roll estimation algorithm indeed functions onboard a real MAV. In Subsection 3.4.1, the setup of this robotic experiment is explained. Subsequently, in Subsection 3.4.2, the results of the experiment are discussed.

3.4.1 Setup

The top part of Figure 15 shows the fixed wing MAV with which the experiment is performed. It is a modified Easystar, equipped with a Paparazzi autopilot[10]. On the nose, a Surveyor SRV-1 BlackFin camera is placed (bottom part of Figure 15).

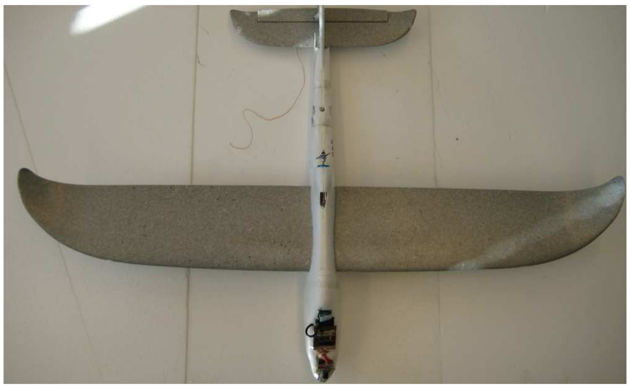
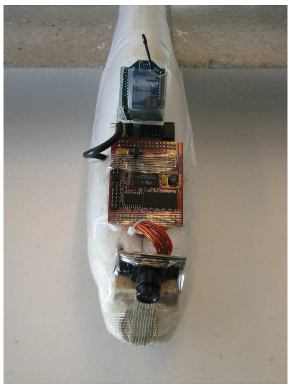

Figure 15: Left: Fixed wing MAV used for the real-world experiments. Right: modified Surveyor SRV-1 BlackFin camera mounted on the nose.

The BlackFin camera has been modified so that it can communicate with a ground station via a 2.4 GHz Xbee communication module. Furthermore, the BlackFin camera has been connected to the autopilot via ADC-channels. In principle one could send many values over the channels by encoding them over time. However, if communication speed is essential, the setup with two ADC-channels implies that the BlackFin camera can communicate only two values to the autopilot via PWM.

As mentioned in the previous section, the algorithms for segmenting the sky and estimating the pitch and roll angle with the perceptron algorithm have been implemented on the BlackFin DSP of a Surveyor camera. During flight, the camera continuously grabs an image and estimates the pitch and roll angle. It uses random sampling of 500 samples, while discarding estimates that have too high an error (see Subsection 3.2.1). The camera can send two values to the autopilot that have to be in the range from 0 to 3.3 V. In order to obtain a good resolution for the most relevant angles, both $\hat{\phi}$ and $\hat{\theta}$ are restricted to the interval $[-60°, 60°]$.

[10]`http://paparazzi.enac.fr/`

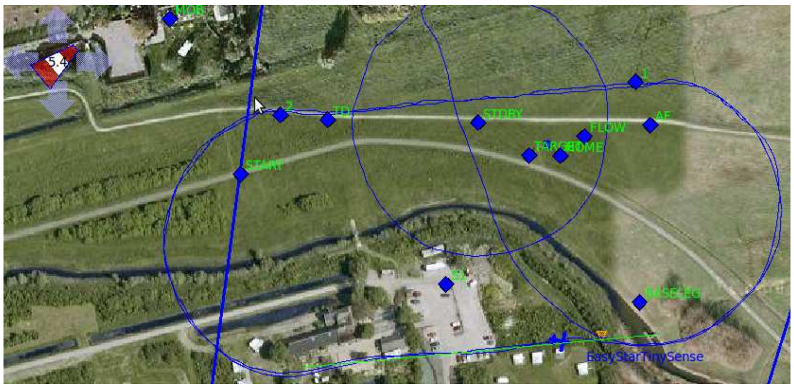

Figure 16: Part of the flight trajectory as shown on the Paparazzi ground control station. The MAV is flying along the green line in the direction of the orange 'carrot'.

A module has been added to Paparazzi that can receive the pitch and roll estimates from both the thermopiles and the BlackFin camera. During the flight only the estimates of the camera are used in the onboard state filter that is used for control, but all signals are logged for post-flight comparison. Although the thermopiles do not provide a real ground-truth value, it is well-known that the corresponding attitude estimates are reliable in clear-sky weather conditions (as was the case during the experiment). Therefore, we require the camera estimates to be similar to the thermopile estimates.

3.4.2 Results

The Paparazzi autopilot successfully used the pitch and roll estimates of the camera for controlling the MAV. Figure 16 shows the flight trajectory of the MAV. The MAV was commanded to first make rightward turning circles and then leftward turning circles, switching from one to the other after ~ 922 seconds. Subsequently, the MAV changed from a circular trajectory to an elliptical one. Figure 17 and 18 show the estimates logged during the part of the flight in which the MAV switches from the rightward turns to the leftward turns. The grey lines are for the thermopile-based estimates and the orange solid lines for the camera-based estimates. The smoother appearance of the thermo-based estimates is due to the slow characteristics of the thermopile infrared temperature sensors.

The roll estimates in Figure 18 correspond qualitatively to what we know

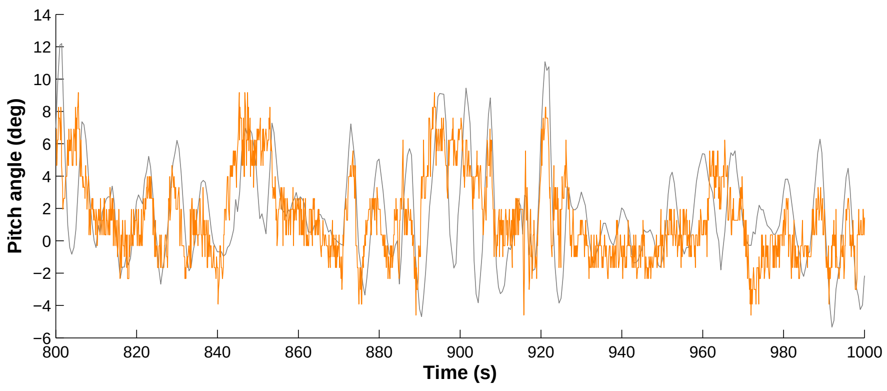

Figure 17: Pitch estimates of the thermopiles (grey) and the camera (orange).

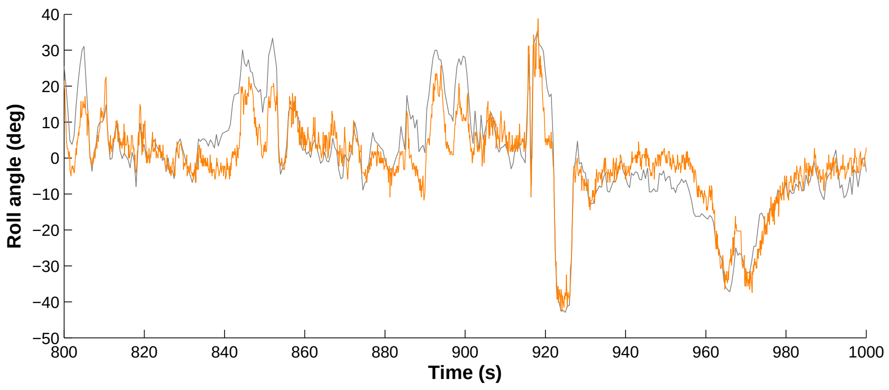

Figure 18: Roll estimates of the thermopiles (grey) and the camera (orange).

about the flight trajectory: before 922 seconds, the roll angles are predominantly positive, while after they are mostly negative. The strong wind and the turbulence during the flight are the causes that the roll angle is changing continuously. Despite these challenging conditions, the MAV succeeds in flying an approximate circular trajectory.

Quantitatively, the camera and thermopile estimates are similar. The largest deviations can be seen for the pitch angle, around 900 seconds. At the three points where the estimates deviate significantly, the uncertainty of the camera estimate is over the threshold ($>$ 10 out of 50 test points are misclassified by the horizon line). As a consequence, the estimates have been disregarded, meaning that a pitch angle of $0°$ was assumed. Over the entire flight, the average absolute deviation between the camera and thermopile estimates are 2.27 degrees pitch and 5.93 degrees roll. The 90^{th} percentile deviation is 4.9 degrees for the pitch angle and 12.3 degrees for the roll angle.

The choice between using the thermopiles or the camera for attitude estimation on MAVs will depend on the circumstances. The disadvantages of the thermopiles are that (1) they cannot cope with weather conditions in which there is too little temperature contrast between the earth and the sky, and (2) they have a relatively slow response time. It is certain that the camera will allow successful state estimation in many weather conditions that would lead to failure of the thermopiles. However, there are also some weather types in which the camera-based attitude estimation fails. For example, 'stormy' skies have a lot of high contrast and low illumination in the sky, which with the current sky-classifier leads to an increased number of misclassifications. Of course, highly foggy conditions also lead to state estimation problems.

Currently, a pre-flight check is always performed to verify that the image segmentation works well. Transient errors such as those occurring when the camera is suddenly overexposed to light do not impair the performance, also thanks to the error checking routine.

4 Discussion

With the results from Section 2 and 3 in mind, we revisit the argument for a more widespread use of sub-sampling techniques.

The reason that we make a case for sub-sampling is that there are few cases in which sub-sampling is placed at the core of (robotic) vision algorithms [48, 42, 4, 16]. Often, the possibility of extracting a small subset of samples is simply ignored. As an illustration, in pitch and roll estimation the entire image is typically processed, while almost all studies mention the problem of rendering the algorithm computationally efficient enough for onboard use [13, 33, 25, 8, 39, 49, 45, 35][11]. When fully processing the image, computational efficiency can only be achieved by employing simple local features, such as raw pixel values in $\mathcal{Y}\mathcal{C}_b\mathcal{C}_r$-space (e.g., [33]). In the case of more complex features / processing, such as in [46], the relatively large computational effort of the algorithm is simply accepted, leading to the requirement of a more powerful processor and a larger robotic system. The experimen-

[11]A noteworthy exception is the work in [24, 46], in which the horizon line is first estimated in a smaller version of the image and then refined with the help of the original image.

tal results in Section 3 show that sub-sampling paves the way for the use of more complex local features on small onboard processors: sub-sampling leads to the difference between an execution frequency of 1.4 Hz (full sampling) and 36 Hz (for 500 samples) on the Surveyor BlackFin, without noticeably changing the algorithm's performance. So the more widespread use of sub-sampling could facilitate the use of better performing algorithms on smaller processors / robotic platforms.

A higher computational efficiency is not only important for enhancing performance or allowing execution on small processors, but also for allowing robots to perform multiple visual tasks in parallel. For instance, the noise detection algorithm studied in Section 2 would never have been used if it were computationally less efficient. Namely, noise detection is not the primary goal of robotic vision. The noise detection algorithm explained in Section 2 has mainly been employed for improving the results of an optic flow module used for height control [17] and obstacle detection [18]. In [18], a 2.26 GHz dual core laptop runs the openCV[12] Lucas-Kanade optic flow algorithm, the noise detection algorithm discussed in this article, and a 'texton'-based[13] obstacle detector that also uses sub-sampling. All of these algorithms can only run at the same time because they can be made computationally efficient enough. The texton-based vision algorithm is rendered a factor ~ 100 faster with the help of sub-sampling, while still retaining a sufficient obstacle detection performance. In other words, sub-sampling can be used to extract information on a *need-to-know* basis, permitting robots to perform multiple vision tasks in parallel.

Finally, sub-sampling is broadly applicable. In particular, it can be applied to the broad class of vision algorithms that either use local features directly, or use local features to determine a global property of the image. An example of the former is given by [4] in which sub-sampling is used to search for a given pattern inside an image. Examples of the latter include the noise detection task (in which local samples are used to determine the correlation r and the number of noisy lines l) and the horizon detection task (in which the samples are used to determine line parameters).

[12]`http://opencv.willowgarage.com/wiki/`

[13]Textons are prototypical image patches.

5 Conclusions

We conclude that sub-sampling is a broadly applicable strategy for rendering vision algorithms computationally much more efficient at an acceptable cost in accuracy. The empirical results of the noise detection algorithm and horizon detection algorithm show that speed-ups of a factor ~ 14 and ~ 21 can be obtained at a moderate cost in performance. The resulting algorithms compare favorably with state-of-the-art algorithms from the literature.

Sub-sampling makes it possible for the vision algorithms to be executed on small onboard processors (as with the horizon detection) or to be executed in combination with other vision algorithms (as with the noise detection). In addition, the number of extracted samples forms a convenient parameter to explore the space of performance metrics and processing time, allowing robots to extract information on a *need-to-know* basis.

The article further explored basic sampling strategies, in particular sampling at random locations and sampling at grid locations. The latter is more successful on the noise detection task, since it ensures a better spread of the sampling locations. In the horizon detection task both strategies have a similar performance, since the spread of the sampling locations is less important than the number of samples close to the true horizon line. Remarkably, in both tasks the order in which locations on the grid are being sampled has to be randomized for better performance. In the noise detection task this is due to a faster reduction of the probability that the noise is missed (Subsection 2.2.1). In the horizon detection task, the cause of this lies with the perceptron learning algorithm that is sensitive to the order in which samples are presented.

Finally, it was shown that the computational efficiency can be further enhanced by means of selective sampling. In the noise detection task, the Bernstein bound is used to stop sampling if the classification is sufficiently certain. This results in a higher true positive ratio for the same number of samples. In the horizon detection task, a selective sampling algorithm for perceptrons is used in order to extract only those samples that are likely to lead to learning updates of the horizon line parameters. This results in lower error magnitudes at the same numbers of samples. A modification of the algorithm that takes into account a possibly skewed relation between sky and non-sky pixels improves the results even further.

We first discuss the future work on the introduced algorithms and then on sub-sampling in general. Future work regarding the introduced algorithms mostly involves testing them in even more different conditions. In the noise detection task, we have focused on indoor environments that were reasonably well-lit. It would be interesting to test it on outdoor images or in rather dark environments. This would also provide insight into how specific the parameter settings of ϑ_r and ϑ_l are to the application. For the horizon detection task more field testing is required in order to better identify the conditions in which the sky segmentation fails. This would hopefully lead to insights into how the segmentation can be ameliorated, possibly leading to the extraction of novel visual features.

Future work regarding sub-sampling includes the application of sub-sampling techniques to other tasks and the development of even smarter sampling strategies. In this article, the focus has been on passive sampling strategies in which the sample locations can be determined in advance. An active sampling strategy could use the information extracted from the image so far to determine the best location for the next sample. In this manner, it may be able to obtain even higher computational efficiencies.

References

[1] T. Andersen and T. Martinez. The little neuron that could. In *International Joint Conference on Neural Networks*, volume 3, pages 1608 – 1613, 1999.

[2] J.-Y. Audibert, R. Munos, and C. Szepesvári. Tuning bandit algorithms in stochastic environments. In *Algorithmic Learning Theory 2007*, pages 150–165, 2007.

[3] D. H. Ballard. Animate vision. *Artificial Intelligence*, 48(1):57–86, 1991.

[4] C. Barnes, E. Shechtman, A. Finkelstein, and D.B. Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. In *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 2009.

[5] Antoine Beyeler, J.-C. Zufferey, and D. Floreano. Optipilot: control of take-off and landing using optic flow. In *European Micro Air Vehicle conference and competitions (EMAV 2009)*, 2009.

[6] J.K. Bradley and R. Schapire. Filterboost: Regression and classification on large datasets. In *NIPS*, volume 20, pages 185–192, 2008.

[7] J. Byrne and R. Mehra. Wireless video noise classification for micro air vehicles. In *2008 Association for Unmanned Vehicle Systems International (AUVSI) Conference*, 2008.

[8] R. Carnie, R. Walker, and P. Corke. Image processing algorithms for UAV 'sense and avoid'. In *IEEE International Conference on Robotics and Automation 2006 (ICRA)*, pages 2848–2853, 2006.

[9] K. Celik, S.J. Chung, and A. Somani. Mono-vision corner slam for indoor navigation. In *(EIT 2008)*, pages 343–348, 2008.

[10] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Worst-case analysis of selective sampling for linear classification. *Journal of Machine Learning Research*, 7:1205–1230, 2006.

[11] S. Chikkerur, C. Tan, T. Serre, and T. Poggio. An integrated model of visual attention using shape-based features. Technical report, MIT CSAIL, CBCL-278, 2009.

[12] O. Chum and J. Matas. Optimal randomized ransac. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(8):1472 – 1482, 2008.

[13] T.D. Cornall and G.K. Egan. Measuring horizon angle from video on a small unmanned air vehicle. In *2nd International conference on autonomous robots and agents*, 2004.

[14] S. Dasgupta, A. Kalai, and C. Monteleoni. Analysis of perceptron-based active learning. *Journal of Machine Learning Research*, 10:281–299, 2009.

[15] G.C.H.E. de Croon, K.M.E. de Clerq, R. Ruijsink, B. Remes, and C. de Wagter. Design, aerodynamics, and vision-based control of the delfly. *International Journal on Micro Air Vehicles*, 1(2):71 – 97, 2009.

[16] G.C.H.E. de Croon, C. de Wagter, B.D.W. Remes R., and Ruijsink. Local sampling for indoor flight. In *Belgium-Netherlands Artificial Intelligence Conference (BNAIC 2009)*, 2009.

[17] G.C.H.E. de Croon, C. de Wagter, B.D.W. Remes, and R. Ruijsink. Random sampling for indoor flight. In *International Micro Air Vehicle conference, Braunschweig, Germany (2010)*, 2010.

[18] G.C.H.E. de Croon, E. de Weerdt, C. De Wagter, B.D.W. Remes, and R. Ruijsink. The appearance variation cue for obstacle avoidance. *IEEE Transactions on Robotics*, in press.

[19] G.C.H.E. de Croon, E.O. Postma, and H.J. van den Herik. A situated model for sensory-motor coordination in gaze control. *Pattern Recognition Letters*, 27(11):1181–1190, 2006.

[20] G.C.H.E. de Croon, B.D.W. Remes, C. de Wagter, and R. Ruijsink. Sky segmentation approach to obstacle avoidance. In *IEEE Aerospace Conference, Big Sky, Montana, USA*, 2011.

[21] J. Denzler and C.M. Brown. Information theoretic sensor data selection for active object recognition and state estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):145–157, 2002.

[22] P. Domingos and G. Hulten. A general method for scaling up machine learning algorithms and its application to clustering. In *ICML*, pages 106–113, 2001.

[23] S. Ertekin, J. Huang, L. Bottou, and C.L. Giles. Learning on the border: active learning in imbalanced data classification. In *CIKM'07*, 2007.

[24] S.M. Ettinger, M.C. Nechyba, P.G. Ifju, and M. Waszak. Vision-guided flight stability and control for micro air vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems 2002 (IROS)*, volume 3, pages 2134 – 2140, 2002.

[25] S. Fefilatyev, V. Smarodzinava, L.O. Hall, and D.B. Goldgof. Horizon detection using machine learning techniques. In *5th international conference on machine learning and applications (ICMLA'06)*, 2006.

[26] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), 2004.

[27] H. He. Learning from imbalanced data. *IEEE transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

[28] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.

[29] D. Hoiem, A. A. Efros, and M. Hebert. Geometric context from a single image. In S. Ma and H.-Y. Shum, editors, *10th IEEE International Conference on Computer Vision (ICCV 2005), Beijing, China*, volume 1, pages 654–661, Washington, DC, 2005. IEEE Computer Society.

[30] S. Jodogne and J. Piater. Closed-loop learning of visual control policies. *Journal of Artificial Intelligence Research*, 28:349–391, 2007.

[31] T. Kato and D. Floreano. An evolutionary active-vision system. In *Congress on Evolutionary Computation (CEC 2001), Seoul, South Korea*, volume 1, pages 107–114. IEEE Computer Society, 2001.

[32] O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *NIPS*, volume 6, pages 59–66, 1993.

[33] T.G. McGee, R. Sengupta, and K. Hedrick. Obstacle detection for small autonomous aircraft using sky segmentation. In *ICRA 2005*, 2005.

[34] V. Mnih, C. Szepesvári, and J.-Y. Audibert. Empirical bernstein stopping. In *25th International Conference on Machine Learning (ICML 2008)*, volume 307, pages 672–679, 2008.

[35] I.F. Mondragón, M.A. Olivares-Méndez, P. Campoy, C. Martínez, and L. Mejias. Unmanned aerial vehicles UAVs attitude, height, motion estimateion and control using visual systems. *Autonomous Robots*, 29:17–34, 2010.

[36] C. Monteleoni and M. Kriinen. Practical online active learning for classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007.

[37] N.Frietsch, J. Seibold, J. Gut, T.Schaich, O. Meister, and G.F. Trommer. Cooperative navigation aiding in heterogeneous uav/ugv teams. In *International Micro Air Vehicle conference and competitions (IMAV 2010)*, 2010.

[38] J.R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[39] C. Rasmussen. Superpixel analysis for object detection and tracking with application to UAV imagery. In *3rd international conference on Advances in visual computing*, volume 1, pages 46–55, 2007.

[40] D. Roth and K. Small. Active learning with perceptron for structures output. In *ICML '06: workshop on learning in structured output spaces*, 2006.

[41] B.C. Russell, A. Torralba, K.P. Murphy, and W.T. Freeman. Labelme: a database and web-based tool for image annotation. *International Journal of Computer Vision*, 77(1–3):157–173, 2008.

[42] J. Shotton, J. Winn, C. Rother, and A. Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *ECCV 2006*, 2006.

[43] Nathan Sprague and Dana Ballard. Eye movements for reward maximization. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

[44] J. Sturm and A. Visser. An appearance-based visual compass for mobile robots. *Robotics and Autonomous Systems*, 57:536–545, 2009.

[45] S. Thurrowgood, D. Soccol, R.J.D. Moore, D. Bland, and M.V. Srinivasan. A vision based system for attitude estimation of UAVs. In *IEEE / RSJ International Conference on Intelligent Robots and Systems*, pages 5725–5730, 2009.

[46] S. Todorovic, M.C. Nechyba, and P. Ifju. Sky / ground modeling for autonomous MAV flight. In *IEEE International Conference on Robotics and Automation 2003 (ICRA)*, pages 1422–1427, 2003.

[47] J. Vogel and N. de Freitas. Target-directed attention: sequential decision making for gaze planning. In *IEEE International Conference on Robotics and Automation (ICRA 2008)*, pages 2372–2379, 2008.

[48] L. Xu and E. Oja. A new curve detection method: Randomized hough transform (rht). *Pattern Recognition Letters*, 11:331 – 338, 1990.

[49] B. Zafarifar, H. Weda, and P.H.N. de With. Horizon detection based on sky-color and edge features. In W.A. Pearlman, J.W. Woods, and L. Lu, editors, *Visual Communications and Image Processing 2008 (SPIE)*, volume 6822, pages 1–9, 2008.

[50] J. Zhu and E. Hovy. Active learning for word sense disambiguation with methods for addressing the class imbalance problem. In *2007 Joint conference on empirical methods in natural language processing and computational natural language learning*, pages 783–790, 2007.