ACT-CORNER: Active Corner Finding for Optic Flow Determination

Guido C.H.E. de Croon[1] and Stefano Nolfi[2]

***Abstract*— A highly efficient corner finding algorithm, named ACT-CORNER, is introduced. The algorithm uses *active sub-sampling*: it searches for corners by exploiting information from local image samples. ACT-CORNER is compared to both the Harris and FAST corner detectors. It is computationally much more efficient than the Harris corner detector. The comparison with FAST depends on the image size, with an advantage for ACT-CORNER for image dimensions of 320×240 pixels or larger. ACT-CORNER's performance is evaluated in the context of optic flow based Time-To-Contact (TTC) estimation. Image zoom experiments show that the accuracy of TTC estimations is similar when ACT-CORNER or FAST are used for corner detection. Finally, experiments with a Parrot AR drone show that the TTC estimates based on ACT-CORNER correspond well to sonar-based estimates.**

I. INTRODUCTION

Optic flow is an important visual cue for the autonomous operation of robots. Especially when the robot is fast and agile, optic flow has to be calculated at a sufficiently high frequency. For this reason, there is a growing body of work on the use of specific optic flow hardware, such as computer mouse sensors [3] or artificial Elementary Motion Detectors [5]. However, the wide availability and multi-functionality of normal digital cameras makes them still of interest for optic flow determination. Typically, optic flow methods determine the flow in camera images by (1) finding locations in the current image that are suitable for being tracked over time (henceforth referred to as 'corners'), and (2) tracking the corners to the subsequent image.

The first step, corner finding, can be computationally intensive. As an example, the well-known Harris corner finding algorithm [7] calculates the first order and approximates the second order derivatives of an image's gray values, while convolving intermediate image representations with a Gaussian filter. Even for small images, an implementation on an embedded processor can require up to a hundred milliseconds of processing time.

Several computationally more efficient methods than Harris have been introduced, of which FAST [11], [12] is the one that is most often used on small robots or mobile devices. FAST is computationally more efficient than Harris, since it employs a simplified definition of what is to be considered a corner, and machine learning is used in order to perform corner classification tests as efficiently as possible.

A perhaps even larger reduction of computational effort is possible if corner detection is not approached as a classification problem but as a visual search problem. The idea of performing a visual search in images on the basis of local image samples is far from new (cf. [2]). However, to date the rare algorithms implementing the idea for corner detection are computationally expensive (cf. [1]).

In this article, a new algorithm called ACT-CORNER is introduced that does display high computational efficiency and an accuracy comparable to existing corner detection algorithms. ACT-CORNER uses *active sub-sampling*: the algorithm performs an informed search for corners by means of 'agents' that process a small local image sample and then either stop or move to explore another sampling location. To understand the potential advantage of this method, one can consider an agent looking at the edge of a table's leg. Shifting its gaze upwards or downwards along the edge will result in finding the corner of the leg with the table plane or with the floor. Of course, the challenge in active sub-sampling lies in finding smart exploration strategies.

The remainder of the article is organized as follows. In Section II, the ACT-CORNER algorithm is explained. Its computational efficiency is compared to that of Harris and FAST in Section III. Subsequently, in Section IV the performance of the three corner finding algorithms is compared on a Time-To-Contact (TTC) estimation task. Conclusions are drawn in Section V.

II. ACT-CORNER

ACT-CORNER is an active sub-sampling method for corner detection. *Sub-sampling* means that only a limited number of local image samples are processed. *Active* means that samples are taken in sequence, with the next sampling location chosen on the basis of the current sample. Since the current sample gives only partial information on the location of surrounding corners, the problem belongs to the challenging class of Partially Observable Markov Decision Problems (POMDPs). The problem is further complicated by the need to identify several good corners.

To approach the problem, an evolutionary swarm robotics methodology is adopted [10], [13]. Under this methodology, the search for corners is performed by a group of agents that move over the image. Their controllers determine the movements by mapping image samples to image shifts. The free parameters of the agents' controllers are optimized by a form of evolutionary algorithm. During the evolutionary optimization, potential solutions are retained or discarded on the basis of the overall performance of the agents in solving the active sub-sampling task. This method permits to: (i) solve POMDPs by exploiting sensory-motor coordination [9], (ii) identify simple and robust solutions that

[1] Micro Air Vehicle lab, Delft University of Technology, Delft, and Advanced Concepts Team of the European Space Agency, Noordwijk, the Netherlands. g.c.h.e.decroon@tudelft.nl

[2] Consiglio Nazionale delle Ricerche (CNR), Roma, Italy. stefano.nolfi@istc.cnr.it

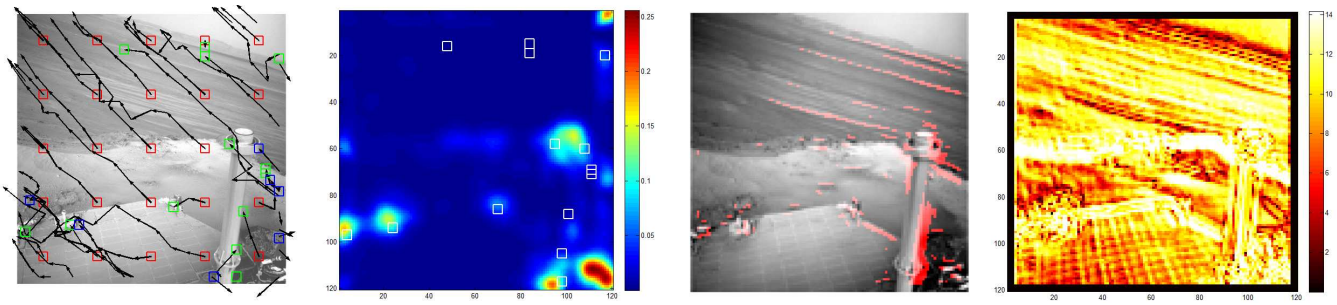

Fig. 1. *From left to right:* (1) example application of ACT-CORNER to an onboard image (at full resolution and brightened for illustration purposes) of the Mars rover Opportunity – details in the text. (2) Harris values at all image locations, with the end locations of stopped agents (squares), (3) the locations at which the agent commands a stop (indicated with a red overlay), (4) the size of the agent's image shifts, $\sqrt{\Delta x^2 + \Delta y^2}$.

might be hard to obtain through explicit design techniques, and (iii) generate coordinated and self-organizing collective behaviours that can allow the agents to explore different parts of the image and to identify several corners. Below we describe the characteristics of the agents, the way in which they are evolved, and the evolved corner finding strategies.

A. Agent

An agent is constituted by a perceptron neural network that has 51 input neurons and 3 output neurons. The first 50 input neurons encode for the horizontal and vertical image gradients I_x and I_y, of a 5×5 pixel window centered on the agent location. The last input neuron is a bias with constant value 1. The output neurons include one binary motor neuron that determines whether the agent will remain in its current location (s = 0) or will move (s = 1), and two additional motor neurons that determine the offset of the movement along the horizontal and the vertical axis $(\Delta x, \Delta y)$ within a $[-10, 10]$ pixel range. All input neurons project connections to all output neurons.

As a consequence of the neural network structure, the agent always reacts in the same way to the same visual inputs. The way in which the agent explores the image and selects a corner point depends on the 153 connection weights $\mathbf{w}$ of the agent's neural controller that are evolved through the algorithm described in the next section. For each image, at the first time step ($t = 1$), 25 agents are initialized on a (5×5) grid over the image. Each agent then interacts with the image until either (i) it issues a stop command, or (ii) it runs out of time ($t = 15$). If an agent moves off the image, it reappears at the other side. The final locations of the agents are the corners found by ACT-CORNER.

B. Optimization of a Swarm of Agents

The free parameters of the agents' neural controllers (i.e., the weights $\mathbf{w}$) have been optimized with the generational Particle Swarm Optimization (PSO) algorithm of the open source project PyGMO[1]. The optimization took place on 6 'islands' (parallel processors), each containing a population of 64 solutions $\mathbf{w}$. Evolution continued for 200 generations. The training set consists of 936 (greyscale 160×120 and 120×120) images collected from five main sources: (1) onboard images of the NASA Mars rover Opportunity[2], (2) photos made on the highway, (3) images made onboard Micro Air Vehicles, (4) Lunar Reconnaissance Orbiter Camera (LROC) images of the moon surface[3], and (5) indoor photos. The images vary significantly in light conditions and amount of texture / contrast. To limit the time necessary for the evolutionary process, each generation of solutions has been tested on a set of 10 images randomly selected from the training set. With different images per generation per island, the entire evolution involved $6 \times 10 \times 200 = 12,000$ randomly selected images. Hence, it is likely that during evolution most training set images have been used.

The goal of ACT-CORNER is to find as many image locations as possible that have a high Harris value. Alternative solutions $\mathbf{w}$ are compared on the basis of the fitness function:

$$f(\mathbf{w}) = \sum_{i \in \mathcal{I}} \sum_{a \in \mathcal{A}} \sum_{x=a_x-2}^{a_x+2} \sum_{y=a_y-2}^{a_y+2} h_i(x, y), \qquad (1)$$

where $\mathcal{I}$ is the set of 10 images, $\mathcal{A}$ the set of agents, (a_x, a_y) is the final position of agent a, and $h_i(x, y)$ is the image i's Harris value at location x, y. In order to discourage agents finding the same corners in the image, $h_i(x, y)$ is set to zero if it has been included in the fitness of one of the agents.

C. Analysis of the Agents' Behavior

The optimization leads to a satisfactory solution. The weight vector optimized by PSO achieves a mean fitness per agent of $2.45 \cdot 10^{-1}$ on a generalization test made on a set of 13 images (different from those used for optimization). For comparison, the mean fitness of random locations on the same set is $1.63 \cdot 10^{-1}$ and that of the first 25 Harris corners is $3.93 \cdot 10^{-1}$. Since these values are difficult to interpret, in Section IV, ACT-CORNER's performance is measured on a time-to-contact estimation task.

In this subsection the agents' behavior is first shown and then analyzed in order to give insight into why ACT-CORNER can successfully locate corners. The leftmost inset of Figure 1 shows an example application of 25 agents to an image from the training set. The agents start on a grid (red squares) and move over the image (black arrows) until they

[1] `http://pagmo.sourceforge.net/pygmo/`

[2] `http://marsrover.nasa.gov/`

[3] `http://www.lroc.asu.edu/`

either issue a stop command (green squares) or run out of time (blue squares). The following inset of Figure 1 shows the Harris values at all image locations. The final locations of stopped agents are indicated with white squares. The main observation from the figure is that the agents end up in areas with considerable texture. Many final locations have high Harris values, although not all of them are located on local maxima. Possibly missing the highest Harris values is inherent to a sub-sampling algorithm such as ACT-CORNER. A secondary observation is that the agents generally move to the top left, modulating their movements on the way. Deviations of the general movement direction typically occur in response to strong edges.

In the two rightmost insets of Figure 1 the following information is shown at all image locations: the locations at which the agent commands a stop $s = 0$ (red overlay), and the size of the image shifts, $\sqrt{\Delta x^2 + \Delta y^2}$. The insets lead to two major observations. First, the stop locations are all located in textured areas, but also include locations that seem to have only one dominant contrast (horizontal or vertical). Only having one dominant contrast can be disadvantageous for determining optic flow because of the aperture problem. Second, the agents move faster in textureless areas such as the sky, and slower in textured areas. There are some textured areas with strong contrast resulting in large shifts, but the direction of these shifts depends on the position of the contrast in the window. As a consequence, in the case of e.g. a horizontal contrast, agents shift up and down while still moving left. This allows the agents to move along edges by then stopping when a corner is found.

III. COMPUTATIONAL EFFORT

The main reason for ACT-CORNER's setup is to achieve a high computational efficiency. In this section ACT-CORNER's computational effort is compared to that of Harris [7] and FAST [11], [12]. First a coarse analysis is performed of the expected number of computations. Then, `C++` implementations of the algorithms are compared.

A. Complexity Analysis

The *Harris* algorithm [7] consists of the following steps:

1) Calculation of I_x, I_y: performed by, e.g., convolution with Sobel masks.
2) Approximation of I_{xx}, I_{xy}, I_{yy} by multiplying I_x, I_y, e.g., $I_{xx} \approx I_x I_x$. The resulting matrices are convolved with a Gaussian mask (e.g., of size 3×3) to render the following step effective.
3) Computation of the Harris value: $h(x, y) = (I_{xx} I_{yy} - I_{xy}^2) - k(I_{xx} + I_{yy})^2$, where typically $k = 0.04$.
4) Non-maximum suppression.

In a $W \times H$ image, the first step requires $2 \times (W-2) \times (H-2)$ multiplications and additions of 3×3 image patches (counted as 17 operations, 9 for multiplication, 8 for adding the results). Here it is assumed that I_x and I_y are calculated with the help of 3×3 Sobel masks. The second step involves $3 \times (W-2) \times (H-2)$ single pixel operations followed by $3 \times (W-2) \times (H-2)$ patch multiplications (again 17 operations). This is followed by $7 \times (W-2) \times (H-2)$ single pixel operations. Nonmaximum suppression requires at least $(W-2) \times (H-2)$ additional pixel operations. Altogether, the above reasoning leads to an estimate of $96 \times (W-2) \times (H-2)$ single pixel operations. This leads to: $1,789,824$ operations for 160×120 images, $7,265,664$ operations for 320×240 images, and $29,276,544$ operations for 640×480 images.

FAST [11] performs the following steps:

1) A corner test for each pixel in the image, involving a decision tree that compares the values from a 16-pixel ring around the center pixel with the value of the center pixel. The test ascertains whether there is a group of n contiguous pixels in the ring that is brighter or darker than the pixel center. A pixel from the ring can be brighter ($p_{\text{ring}} > p_{\text{center}} + T$), darker ($p_{\text{ring}} < p_{\text{center}} - T$), or similar ($p_{\text{center}} - T \leq p_{\text{ring}} \leq p_{\text{center}} + T$).
2) For each corner, first a corner response function is calculated. To find the corner strength v in the online implementations of FAST, a binary search is performed to find the maximum threshold T_{max} for which the corner is still defined as a corner.
3) Non-maximum suppression.

The number of operations of FAST mainly depends on image size and the values of n and T, which influence the number of pixel comparisons. For each pixel first $p_{\text{center}} - T$ and $p_{\text{center}} + T$ are calculated (2 operations), and then the decision tree performs a number c of comparisons. For FAST9 $n = 9$ and $c = 2.26$ on average [11]. This would imply $4.26\,(W-6)\,(H-6)$ operations. Since binary search has an average complexity of $log_2(N) - 1$, with $N = 256$ in the case of the threshold, 7 operations are assumed for the corner response calculation. Another 8 are used to do the nonmaximum suppression of a point with an 8-pixel neighborhood. So after corner detection, 15 operations are performed per corner. With $C = 25$ detected corners and $W \times H = 160 \times 120$, $25 \times 15 + 4.26 \times 154 \times 114 \approx 75,164$. For 320×240 and 640×480 pixels, the estimates amount to $313,383$ and $1,280,573$ operations, respectively.

ACT-CORNER maximally performs t times the steps:

1) Calculation of I_x, I_y at the A agent locations
2) Computation of the perceptron outputs

The first step uses Sobel masks and involves $2\,A$ multiplications and additions of 3×3 image patches (17 operations per patch). The second step involves A times 153 multiplications, 150 summations, and 3 activation function evaluations (estimated in total as 310 operations). The total number of operations is then $t \times A \times 344$. The current setup with $t = 15$ and $A = 25$ would lead to $129,000$ operations.

Although the actual computation times strongly depend on the exact implementation and hardware, this analysis indicates that the computation cost of the ACT-CORNER is lower than the HARRIS algorithms by a factor of 13.9, 56.3, and 226.9, for images of 160×120, 320×240, 640×480 respectively. ACT-CORNER also requires fewer operations than the FAST algorithm for images of 320×240 and 640×480 pixels by a factor of 2.4 and 9.9, respectively.

TABLE I
AVERAGE PROCESSING TIMES OF THE HARRIS, FAST, AND ACT-CORNER ALGORITHMS FOR DIFFERENT IMAGE SIZES.

	160×120	320×240	640×480
Laptop			
Harris	3.47 ms	12.67 ms	46.46 ms
FAST	**0.17** ms	0.84 ms	2.69 ms
ACT-CORNER	0.49 ms	**0.78** ms	**1.02** ms
DSP			
ACT-CORNER	**10** ms	**12** ms	**14** ms

Moreover the computational cost of ACT-CORNER does not directly depend on image size as for the other two algorithms. The number of required operations might increase linearly if the number of agents or number of time steps is increased. However, the algorithm scales rather well to larger images with the same parameters.

B. Empirical comparison computational effort

For the empirical comparison of the computational effort, `C++` implementations of the three algorithms have been compared on a laptop with an Intel Core i7 processor at 2.00 GHz. The FAST-implementation is the one made available at [4], the Harris implementation is the one included in the open source software package OpenCV[5], while ACT-CORNER has been implemented by the authors. Table I shows the average number of milliseconds spent by the three algorithms on processing images of different sizes (results under 'Laptop'). The experimental results qualitatively correspond to the two expectations from the complexity analysis. First, Harris' method is by far slowest on all image sizes. Second, the FAST corner detector is faster on the smallest image size than ACT-CORNER and slower on larger image sizes, although the difference is quantitatively smaller than expected. An unexpected result is that the computation cost of the ACT-CORNER slightly increases with the image size. This is due to the fact that the usage of the same movement range on all images implies the exhibition of a longer movement phase before stopping, in order to traverse the larger texture-poor areas experienced in larger images.

Finally, a `C` implementation of ACT-CORNER has been made for the Digital Signal Processor (DSP) of the Surveyor BlackFin camera system, which can be used on small robotic systems. Table I shows the processing times of ACT-CORNER for different image sizes (results under 'DSP'). The main observation here is that ACT-CORNER can be run at an execution frequency of ~ 71 Hz – 100 Hz (faster than the maximal image capture frequency). Please remark that no DSP-specific optimizations of the code have been performed, so further speed-ups are possible.

IV. PERFORMANCE COMPARISON ON TIME-TO-CONTACT ESTIMATION

The computational efficiency of ACT-CORNER is only of value if the corners are of sufficient quality for downstream vision processing. In this section, the performance of ACT-CORNER is studied and compared with Harris and FAST in the context of a Time-To-Contact (TTC) estimation task[6].

A. Optic Flow Determination

The optic flow determination receives the corners found by one of the three algorithms and tracks them to the next image with the well-known Lucas-Kanade algorithm [4]. A naive implementation would provide the optic flow vectors directly to the TTC estimation algorithm. However, in order to obtain accurate estimates at large TTCs, two additional measures are taken: (1) unreliable optic flow vectors are discarded, and (2) points are followed over time with a Kalman Filter. The first measure involves tracking the features back from the current frame to the past frame, and evaluating whether they are sufficiently close to the original feature locations. The second measure involves a motion model that assumes the optic flow to remain constant over time.

B. Time-To-Contact Estimation Algorithm

When approaching an object, the expanding visual flow provides a measure of the TTC: the distance to the object divided by the approach velocity[7]. The algorithm used in the experiments estimates the TTC by determining the divergence of the optic flow field. It assumes (1) that the object is predominantly planar, and (2) that camera rotations are either not present or are accounted for by means of proprioception (viz. derotation based on gyrometers).

In the explanation of the algorithm a pinhole model is assumed, as in [8]. The following notation is used. Image coordinates of an imaged point P are denoted as $p = (x, y)$, optic flow as (u, v), and the spatial derivatives of optic flow as u_x, u_y, v_x, and v_y. Translational velocity is expressed as $\mathbf{V} = (V_x, V_y, V_z)$. The world point imaged on the image center $(0, 0)$, is named P_C. The distance to P_C is Z. The camera only provides information on the normalized velocity: $\vartheta = (\vartheta_x, \vartheta_y, \vartheta_z) = \mathbf{V}/Z$. Finally, the inclination of the surface around point P_C is represented by z_x, z_y.

Assuming no camera rotation, we have from [8]:

$$u_C = -\vartheta_x, \quad v_C = -\vartheta_y \tag{2}$$

$$u_x = \vartheta_z + \vartheta_x z_x, \quad v_y = \vartheta_z + \vartheta_y z_y \tag{3}$$

$$u_y = \vartheta_x z_y, \quad v_x = \vartheta_y z_x \tag{4}$$

The divergence at p_C can then be written as:

$$\mathrm{div}(p_C) = u_x + v_y = 2\vartheta_z + \vartheta_x z_x + \vartheta_y z_y. \tag{5}$$

[4] `http://www.edwardrosten.com/work/fast.html`
[5] `http://opencv.willowgarage.com/wiki/`

[6] In computer vision, corner detectors are typically tested on measures such as repeatability, i.e., how often the algorithm finds the same world point in images from different angles and positions. Since ACT-CORNER does not process the entire image, it is not to be expected that the same point will be detected in both images. Moreover, repeatability is not a direct measure of how well corners are suited for optic-flow-based downstream processing. Therefore, repeatability tests seem less adequate than the proposed TTC estimation tests for studying ACT-CORNER in a robotic context.

[7] Please remark that the TTC is only equal to the actual time remaining until contact if the approach velocity remains constant.

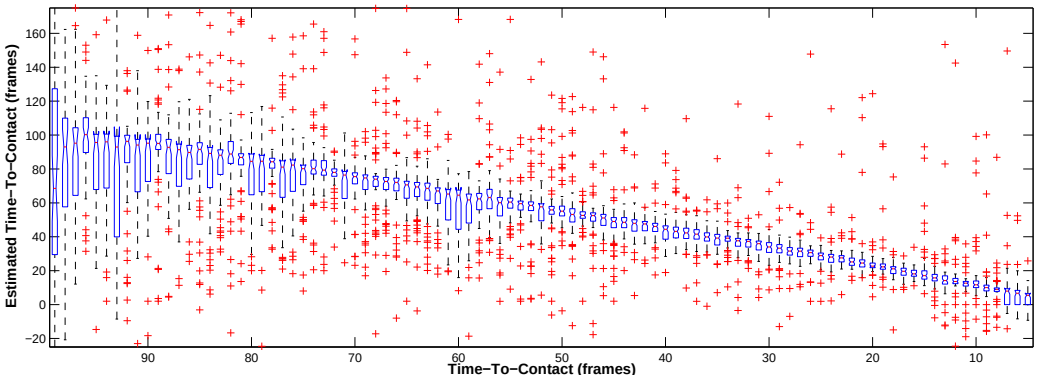

Fig. 2. Box plots representing the estimated TTC distributions per time step for digital zoom approach sequences in the most difficult image set (indoor). The time-to-contact decreases from 100 to 5 frames. The box plots illustrate the interquartile ranges. Red squares represent outliers.

The divergence is inversely related to the TTC (denoted as τ). For example, in the case of $\vartheta_x = \vartheta_y = 0$:

$$\tau = \frac{1}{\vartheta_z} = \frac{2}{\mathrm{div}(p)}, \tag{6}$$

with τ the time at which the camera center will touch point P_C if the velocity V_z stays the same. With nonzero $\vartheta_x z_x$ or $\vartheta_y z_y$, the divergence can still be regarded reciprocal to a time-to-contact, if the surface around P is assumed to extend to the point in which it intersects the direction of motion.

As stated before, the algorithm employed here to estimate the TTC assumes the object to be predominantly planar. It estimates the parameters $\mathbf{p_u}$, $\mathbf{p_v}$ of the equations:

$$u = u_C + u_x x + u_y y = (1, x, y)\mathbf{p_u}^\mathsf{T}, \tag{7}$$

$$v = v_C + v_x x + v_y y = (1, x, y)\mathbf{p_v}^\mathsf{T}. \tag{8}$$

Having the parameter vectors $\mathbf{p_u}$, $\mathbf{p_v}$ permits the calculation of the divergence and therefore the TTC. Estimation of the parameters is done with the help of the set of optic flow vectors $\mathcal{V} = \{(u_1, v_1, x_1, y_1), (u_2, v_2, x_2, y_2), \ldots (u_N, v_N, x_N, y_N)\}$ found as described in the Subsection IV-A. The algorithm finds least-squares solutions to the systems:

$$\mathbf{u} = \mathbf{A}\,\mathbf{p_u}, \quad \mathbf{v} = \mathbf{A}\,\mathbf{p_v}, \tag{9}$$

where $\mathbf{u}$ consists of all u_i, $\mathbf{v}$ consists of all v_i, and $\mathbf{A}$ is a matrix with rows $(x_i, y_i, 1)$. In order to be robust to both noise and deviations from the planar assumption, a RANSAC procedure is used for both fits (using 20 fits of 5 flow vectors). The parameters with lowest error on all optic flow vectors $\mathcal{V}$ are selected. If there are too few points or if all the points are colinear, the TTC is assumed to be 0.

C. Image Zooms

In order to generate a large number of tests while still having access to ground-truth TTC values, experiments have been performed on digital image zooms. Five image sets have been employed, mentioned here with the number of resulting image sequences: (1) moon surface images (30 sequences), (2) indoor images (62 sequences), (3) images of the NASA Mars rover Opportunity (27 sequences), (4) urban outdoor images (106 sequences), and (5) forest outdoor images (66 sequences). The generated images have dimension 160×120. A constant velocity approach is simulated by the zoom, with the TTC decreasing from 100 frames to 5 frames. The results can be seen in Table II. Three main observations can be made from the table. First, the Harris corner detector overall leads to the most accurate downstream vision processing results: it has the lowest mean absolute estimation error $\overline{\tau - \hat{\tau}}$ on four out of five sets. Second, ACT-CORNER achieves slightly more accurate results than FAST and slightly lower mean error than Harris on the Opportunity image set. Its standard deviation is a bit higher than that of the other methods. Third, all three methods obtain the highest error on the indoor image zooms. The reason for this is that indoor environments can have too little texture to reliably determine optic flow and consequently TTC (cf. [6]).

TABLE II

ACCURACY ON ZOOM SEQUENCES, $|\tau - \hat{\tau}|$ (IN FRAMES).

	Harris	FAST	ACT-CORNER
Moon surface	**7.9** ($\pm$25.7)	14.1 ($\pm$60.1)	13.5 ($\pm$57.3)
Indoor	**19.4** ($\pm$67.6)	34.3 ($\pm$76.9)	27.5 ($\pm$99.7)
Opportunity rover	8.8 ($\pm$36.4)	10.5 ($\pm$14.9)	**8.3** ($\pm$40.8)
Urban outdoor	**8.4** ($\pm$34.1)	15.9 ($\pm$43.4)	10.8 ($\pm$50.4)
Forest outdoor	**6.8** ($\pm$20.8)	13.5 ($\pm$48.5)	9.4 ($\pm$41.7)

To further clarify the results in Table II, Figure 2 shows the least accurate results of ACT-CORNER (indoor image set). The box plots illustrate the interquartile ranges of the TTC estimate ($\hat{\tau}$) distribution per time step of the approach. The whiskers extend to the interquartile ranges ± 1.5 times their difference. The crosses illustrate 'outliers', i.e., estimates that are further away from the interquartile ranges than the whiskers. Almost all $\hat{\tau}$-distributions are close to the true TTC, with the box plots forming a line decreasing from ~ 100 to ~ 5. There is a considerable number of outliers though, which are mainly caused by a lack of texture. A subsidiary observation is that the first few time steps have a larger estimate variance than the rest of the approach. This is due to the initialization of the Kalman filtering. Finally, please note that results could further be improved by filtering the TTC estimates over time (e.g., rejecting outliers).

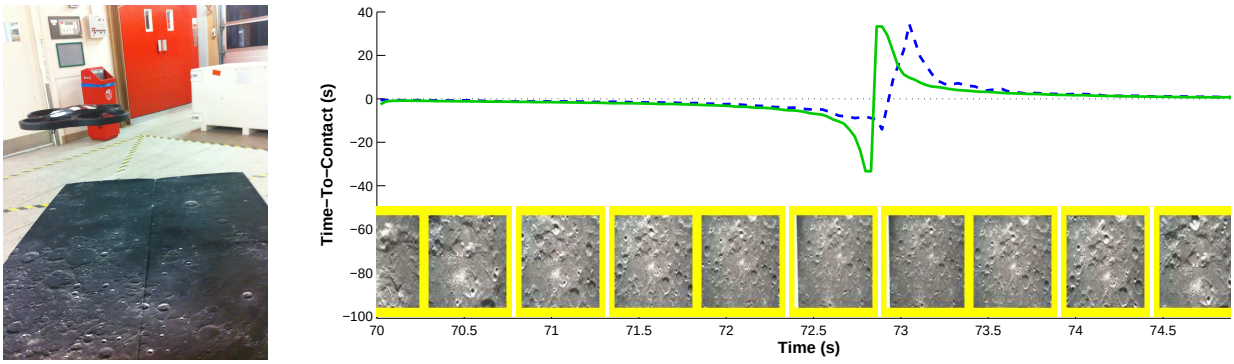

Fig. 3. *Left:* Parrot AR drone during the experiment. *Right:* TTC estimates (blue dashed line) and sonar-based TTC (green line) over time, with corresponding images from the onboard downward pointing camera. The drone first recedes from the moon surface posters and then approaches them.

D. Robotic Experiments

In this Subsection, the TTC estimation algorithm using ACT-CORNER is applied to a real robotic platform, implying perturbing factors such as image noise and motion blur. To this end, experiments are performed with a Parrot AR drone, which is equipped with a downward-pointing camera (sending images at 20 frames per second) and sonar sensor. Given a smooth surface, the sonar sensor provides reliable measurements of the drone's altitude over time. These measurements can be used to determine a 'ground-truth' TTC. We assume though that the sonar cannot reliably measure TTCs outside the interval of $[-1000, 1000]$ frames, and saturate the TTC outside that interval.

1) Experimental Setup: The TTC-measurements are performed while the experimenter decreases and increases the drone's altitude. The drone sends its sonar measurements and bottom camera images to the ground station, where all vision processing is performed. ACT-CORNER is used in combination with the TTC estimation algorithm. During the robotic experiments, optic flow is not tracked back from the current to the past image, but all optic flow vectors are input to the divergence determination. Another notable difference with the image zoom experiments is that the TTC estimates are low-pass filtered over time.

2) Results: Three experiments have been performed with the drone flying above surfaces with different textures: a poster with a moon surface, an empty but slightly textured floor, and the same floor but with the AR drone helipad positioned below the drone. In all cases, the estimated TTC corresponds well to the sonar-based TTC, with a median absolute error of 0.4s – 0.7s during vertical maneuvers. The performance of the TTC estimation is illustrated in Figure 3. It shows the estimated TTC (blue dashed line) and the sonar-based TTC (green line) over time when the drone moves first away and then towards a poster with a moon surface. Below, images are shown at different times. The TTC estimates are close to the sonar-based TTC, except when the movement of the drone reverses. Large positive or negative TTC-values imply slow movements, which are hard to estimate. In addition, when the movement reverses the actual TTC has a singular point: it instantly goes from $-\infty$ to $+\infty$. The filtering results in a delayed response to this singularity and some erroneous TTC-values.

V. CONCLUSIONS

We conclude that ACT-CORNER successfully detects corners suitable for optic-flow based time-to-contact estimation. The method is computationally efficient, performing far fewer operations than the Harris corner detector on any image size and performing fewer operations than FAST on image sizes larger than 320×240. Finally, experiments with a Parrot AR drone show that the TTC estimates based on ACT-CORNER correspond well to sonar-based estimates.

ACKNOWLEDGMENTS

We thank Laurens van der Maaten for proof-reading and Paul Gerke for assisting in the robotic experiments.

REFERENCES

[1] T.L. Arnow and A.C. Bovik. Foveated visual search for corners. *IEEE Transactions on Image Processing*, 16(3), 2007.
[2] D. H. Ballard. Animate vision. *Art. Int.*, 48(1):57–86, 1991.
[3] 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.
[4] Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. description of the algorithm, 2000.
[5] F. Expert, S. Viollet, and F. Ruffier. Outdoor field performances of insect-based visual motion sensors. *Journal of Field Robotics*, 28(4):529–541, 2011.
[6] W.E. Green and P.Y. Oh. Optic flow based collision avoidance on a hybrid MAV. *IEEE Rob. and Autom. Magazine*, 15(1):96–103, 2008.
[7] C. Harris and M. Stephens. A combined corner and edge detector. In *4th Alvey Vision Conference*, pages 147–151, 1988.
[8] H.C. Longuet-Higgins and K. Prazdny. The interpretation of a moving retinal image. *Proceedings of Royal Society, London B*, 208:385–397, 1980.
[9] S. Nolfi. Power and the limits of reactive agents. *Neurocomputing*, 42(1–4):119–145, 2002.
[10] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press / Bradford Books, Cambridge, MA, 2000.
[11] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.
[12] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 32:105–119, 2010.
[13] V. Trianni and S. Nolfi. Engineering the evolution of self-organizing behaviors in swarm robotics: A case study. *Artificial Life*, 17(3):183–202, 2011.