Choosing between optical flow algorithms for UAV position change measurement

Jasper de Boer, Mathieu Kalksma

Abstract—In recent years optical-flow-aided position measurement solutions have been used in both commercial and academic applications. These systems are used for navigating unmanned aerial vehicles (UAVs) in GPS-deprived environments. Movement in sequential images is detected and converted to real world position change. Multiple approaches have been suggested, ranging from using an optical mouse sensor to the use of a stereo camera setup. Our research focuses on single camera solutions.
Previous research has used a variety of optical flow algorithms for single camera solutions. This paper presents a comparison on three algorithms to check if using different algorithms yield different results in terms of quality and CPU time. This paper also provides insight into the general theory behind using single camera optical flow for UAV navigation.
The compared algorithms are the Lucas-Kanade method, Gunnar Farnebäck's algorithm and Block Matching. A testing framework and custom indoor and outdoor datasets were created to measure algorithms flow estimation quality and computation time.
Characteristic differences were found between the performance of the algorithms in terms of both computation time and quality. Choosing between algorithms therefore can increase flow estimation quality or reduce CPU time usage. Also different winners per test set were found in terms of estimation quality.

Index Terms—UAV navigation, optical flow, Lucas-Kanade method, Gunnar Farnebäck method, Block Matching.

1 Introduction

In recent years the interest in using unmanned aerial vehicles (UAVs) has increased. UAVs can be used for a multitude of applications, for example the inspection of agricultural lands, reviewing annual ditch cleanings, or the inspection of wind turbine blades. An essential part to such systems is position measurement which is needed for point-to-point navigation or for maintaining a position.

One method for navigating UAVs is Global Positioning System (GPS). However, for some applications GPS introduces a significant error to the location [7] and GPS will not work when there is no signal available, for instance in indoor environments.

An alternative to GPS are optical-flow-aided position measurement systems. These systems use sequential images from a camera and computer vision to measure movement. In recent years several single camera solutions have been proposed, which use different optical flow algorithms: SAD Block Matching [4] and Lucas-Kanade [9]. For both algorithms the results are promising.

In 2014 the Twirre architecture for autonomous mini-UAVs using interchangeable commodity components was introduced [14]. The platform is developed for automating inspection tasks. The architecture consists of low-cost hardware and software components. The processing power is positioned on-board the UAV. This allows the platform to run the recognition software on-board. Twirre is able to use optical flow for position measurement.

With architectures like Twirre, CPU time is shared with software that is performing the inspections. Optical flow computation time should therefore be kept to a minimum. This paper therefore investigates if performance in terms of estimation computation time can be improved by choosing the right algorithm. This paper also compares the quality of the algorithms in order to see if optical flow quality can be improved by choosing between algorithms.

The algorithms that will be evaluated in this paper are: SAD Block Matching, the Lucas-Kanade method [8] and the Gunnar Farnebäck technique [5]. The first two algorithms are used in [4] and [9] and the Gunnar Farnebäck technique is used in Twirre. These three algorithms were chosen because they are already applied in the field and because of their availability in the OpenCV library [2]. OpenCV is easy in use and extensively used in the field of computer vision. Its BSD license allows both academic and commercial use [3].

The paper starts with a background chapter which introduces optical flow and how it can be used for measuring position change in terms of real world distance. Section 3 shortly introduces the three used optical flow techniques. The algorithms are compared by looking at their theoretical differences and their efficiency. Section 3 finishes with explaining the test environment, and the data sets that are used for testing the three algorithms. Section 4 provides the results of the tests and section 5 and 6 will give the conclusion and discussion, and some future directions accordingly.

- *Jasper de Boer is a MSc. Computing Science student at the University of Groningen and a Project Engineer at the Centre of Expertise in Computer Vision of the NHL University of Applied Sciences, E-mail: jasper.boer@nhl.nl.*
- *Mathieu Kalksma is a MSc. Computing Science student at the University of Groningen, E-mail: m.kalksma@student.rug.nl.*

2 Background

This section introduces a method for measuring position change with a single camera optical flow system. This section also discusses how the choice of camera and optics affects the optical flow system. Some of the choices made for the research method (Section 3) are based on the method described in this section.

The goal of a optical flow system is to measure x, y and rotational position change on the plane parallel to the ground in a real world measure. The described method assumes the use of a camera that is aimed towards the ground.

2.1 Optical flow

An optical flow algorithm is able to track points across two images. Given a set of points or pixels in the first image, the algorithm tries to locate the points in the second image. From the result two corresponding sets of vectors can be constructed. the source vectors

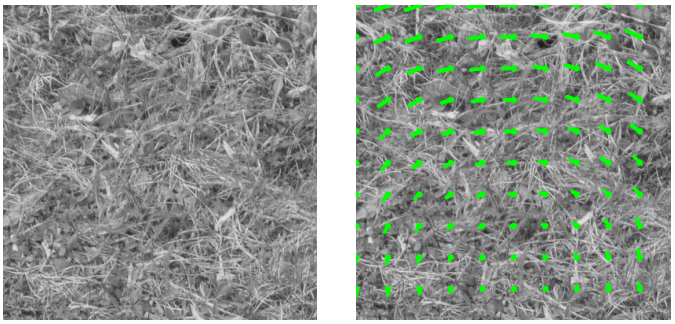
$$\mathbf{v}_{src}[n] = \begin{pmatrix} \mathbf{p}_x[n] & \mathbf{p}_y[n] \end{pmatrix}^T$$

and the destination vectors

$$\mathbf{v}_{dst}[n] = \begin{pmatrix} \mathbf{p}_x[n] + \mathbf{t}_x[n] & \mathbf{p}_y[n] + \mathbf{t}_y[n] \end{pmatrix}^T$$

where $\mathbf{p}[n]$ is the original pixel coordinate and $\mathbf{t}[n]$ contains the translation of the pixel. Figure 1 visualizes this process.

Most optical flow techniques are based on the brightness constancy assumption and the spatial smoothness assumption [1]. With the brightness constancy assumption it is assumed that points keep the same intensity between frames. The spatial smoothness assumption



(a) first frame (b) second frame with the detected flow

Fig. 1: Illustration of detected optical flow between two frames. The second frame is a slightly rotated and translated version of the first frame.

comes from the observation that neighboring pixels generally belong to the same surface and therefore have similar motion [11].

2.2 Angular correction

An UAV has three principal axis as seen in figure 2. The described method uses similarity transformation estimation, and hence angular correction should be applied for both pitch and roll axis. This correction can be done by using a camera gimbal or by transforming the $\mathbf{v}_{src}[n]$ and $\mathbf{v}_{dst}[n]$ vectors.

A gimbal can be used in order to keep the camera perpendicular to the ground. A gimbal uses an inertial measurement unit (IMU)[1] and electric engines to allow the camera to pivot around one axis (yaw), and therewith blocking other rotational actions (rolling and pitching).

The second approach uses the orientation information from an IMU in order to transform both $\mathbf{v}_{src}[n]$ and $\mathbf{v}_{dst}[n]$ vectors.

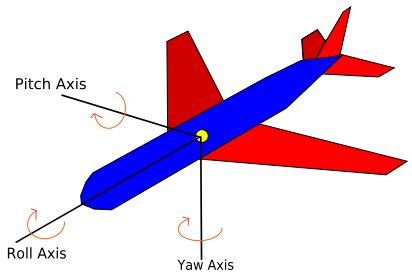


Fig. 2: Principal axis of an UAV. By Auawise (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

2.3 Estimating the similarity transformation

After applying angular correction, the transformation between two frames is reduced to translation $\mathbf{t}$, rotation R and uniform scaling c. This four degrees of freedom similarity transformation can be estimated by reducing the least squares error as in equation 1 [13].

$$e^2(R,t,c) = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{v}_{dst}[i] - (cR\mathbf{v}_{src}[i] + \mathbf{t})||^2 \quad (1)$$

2.4 Pixel distance to real world distance

To calculate real world position change it is necessary to know the distance between the UAV and the ground plane. The height h can be measured e.g. with an ultrasonic distance sensor or with a barometric pressure sensor. Equation 2 is used for calculating the real world traveled distance, where $\mathbf{s}$ is the pixel size of the camera sensor and f the focal length.

$$\mathbf{t}_{rw} = \frac{1}{f}\mathbf{t}\mathbf{s}h \quad (2)$$

From the equation can be seen that a shorter focal length increases the real world traveled distance per moved pixel. Also a greater height and larger size of the camera sensor pixels increase the real world distance. Along with the optical flow algorithm and the frame rate of the camera, these parameters determine the maximum real world speed at which the optical flow system is able to measure position change.

3 Materials and Methods

In this section will provide the setup of the research. The section starts with some theoretical information about the used algorithms. After giving the theoretical background the section continues with the developed testing framework. Finally it concludes by explaining how the experiments are performed and by introducing the used data sets.

3.1 Description of examined optical flow algorithms

In this section the three different algorithms for computing optical flow are presented. They will be presented by explaining their theory, efficiency and showing whether they calculate sparse optical flow or dense optical flow. The difference between sparse optical flow and dense optical flow is that sparse flow only calculates the flow for certain specified pixels, while dense optical flow calculates the flow for all the pixels. This makes sparse algorithms often faster. Since dense flow results in more flow vectors, more data is used to minimize the error function in equation 1, which can lead to a better estimation of the overall transformation.

In case of a large movement between the two images the algorithms sometimes can not detect the movement. A solution for this, which both Lucas-Kanade and Gunnar Farnebäck use, are pyramids. With pyramids you take l pyramid levels and you run the algorithm for each level. For each level L the image is shrunk L times resulting in a 2^L times smaller image. With the image shrunk, the initial too large movement is now detectable.

First the Block Matching algorithm is described, followed by the Gunnar Farnebäck method and then the Lucas-Kanade method.

3.1.1 Sum of absolute differences

The sum of absolute differences (SAD) algorithm is measures similarity between image blocks. It starts by having two images of size $M \times N$. For each pixel in the first image a window of BxB is placed over the image, with the current pixel as middle point. In the new frame the the algorithm searches within a given search area around the position of the original pixel. The algorithm moves the window over every possible position within the search area. For every window position in the new frame the SAD value is calculated. The position with the lowest SAD value is returned as the new position of the current pixel. This process can also be described as minimizing in the following function:

$$\sum_{i=0}^{1-b}\sum_{j=0}^{b-1} |(f_k(x+i,y+i) - f_{k-1}(x+i+u,y+j+v))|. \quad (3)$$

In this equation f_k and f_{k-1} are the two images and b is the block size. By minimizing the result the flow of the current pixel can be estimated. This process is repeated for every pixel in the original image.

This algorithm is very time consuming. The operation will require $O(MNR^2B^2)$ time for a $M \times N$ image, $R \times R$ search region and a $B \times B$ block size. The operation is a dense optical flow method, as it calculates the flow for all the pixels in the image.

[1] An IMU uses accelerometers and gyroscopes to measure orientation and gravitational forces.

3.1.2 Gunnar Farnebäck method

The Gunnar Farnebäck method [5] is a two-frame motion estimation algorithm. Gunnar Farnebäck uses quadratic polynomials to approximate the motion between the frames. This can be done efficiently by using the polynomial expansion transform.

In the case of Gunnar Farnebäck the point of interest is quadratic polynomials that produces the local signal model expressed in a local coordinate system such that

$$f(x) \sim x^T A x + b^T x_c, \tag{4}$$

where A is a symmetric matrix, b a vector and c a scalar.

Gunnar Farnebäck is a dense optical flow algorithm because it computes the optical flow for all pixels in the image.

3.1.3 Lucas-Kanade method

The Lucas-Kanade method describes an image registration technique using spatial intensity gradient information to search for the best match [8]. It does this by taking more information about the image into consideration. With this, the method is capable of finding the best match using far less computations than other techniques that use a fixed order to search. The algorithm takes advantage of the fact that in most cases the two images are already close to each other. The registration problem is finding a vector h that minimizes the distance between two images $F(x)$ and $G(x)$ so that the distance between $F(x+h)$ and $G(x)$ is minimized in a region of interest R. Lucas-Kanade suggests a generalization to deal with distortions such as rotation of the image.

To find the disparity h Lucas-Kanade uses,

$$\begin{aligned} h_0 &= 0, \\ h_{k+1} &= h_k + \frac{\sum_x w(x) F(x+h_k)[G(x) - F(x+h_k)]}{\sum_x w(x) F(x+h_k)^2}, \end{aligned} \tag{5}$$

where $F(x)$ and $G(x)$ are the input images and

$$w(x) = \frac{1}{|G(x) - F(x)|}, \tag{6}$$

is a weighting function.

Finally this algorithm will converge in $O(M^2 \log N)$ [8] time. As Lucas-Kanade only uses given pixels of the image to measure the optical flow it is a sparse optical flow algorithm.

3.2 Testing framework

For conducting the experiments a testing application was developed. The program heavily relies on the OpenCV library [2]. The program can perform optical flow on subsequent images with similarity transformations, using one of the three described optical flow algorithms. The desired algorithm and the corresponding parameters can be set. The application estimates the 4 degrees of freedom (DOF) similarity transformation with the corresponding OpenCV function *estimateRigidTransform()*. The application can also use the *findHomography()* function to find the 8 DOF homography transformation.

The program results the pixel translation of the centre point and the global scaling and rotation between subsequent images.

A second program was written for extracting sequential frames from a large picture. This program was used to generate the image data described in Section 3.3. This section describes the frame extraction process in more detail.

3.3 Data sets

The program for extracting frames from a larger image was used for generating test sets. The program moves a window over an image. The pixels within the window are saved as a new image. The advantage of this approach is that perfect ground data can be used to compare the results of the algorithms against. This is because the translation and rotation are known. Pitching and rolling actions are not incorporated in the simulator, because they can be corrected for as described in Section 2.2.

Figure 3 shows the followed path for the dotted floor image. The individual frames were converted to grayscale and white Gaussian noise with a mean of zero and a variance of *0.001* was added to the frames to make the simulation more realistic.

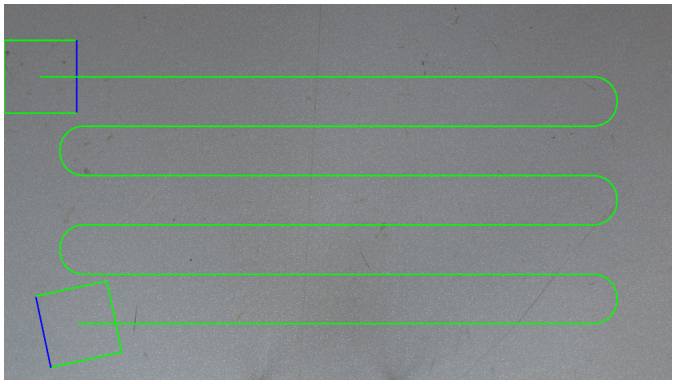


Fig. 3: The path which is followed in order to generate sequential images. The windows of the first and last frame are displayed

Three different images from different types of ground surfaces were used (Figure 4). The first image is an outdoor picture of a grass covered surface. The other two pictures are from indoor floors: a grey floor with small white dots and a floor with red carpet. These three surfaces were chosen in order to have a diversity of data sets where both indoor synthetic floors and an outdoor underground were represented. More data sets could have been generated, but chosen images proved to be diverse enough to answer the research question.

The specifications of the test sets generated from the described images are listed in Table 1. The flow speed can be decreased by binning the frames. By binning an image an area of pixels in the original image is combined into a single pixel for the output image. A two by two binning on an image combines 4 pixels into one pixel and thereby reduces the total number of pixels by a factor 4. The flow speed is then decreased with a factor of 2. The flow speed can be increased by skipping frames.

Table 1: Specifications of the generated and used test sets

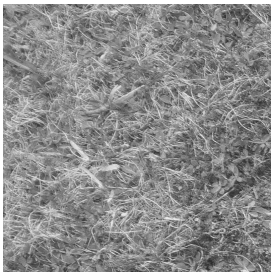
name	original size	crop size	nr. of frames	flow (px)
grass	4608 x 3456	800 x 800	1164	12.3 - 44.3
dotted floor	7360 x 4135	800 x 800	2386	12.3 - 45.2
carpet	7360 x 3856	800 x 800	2056	12.3 - 45.2

3.4 Experiments

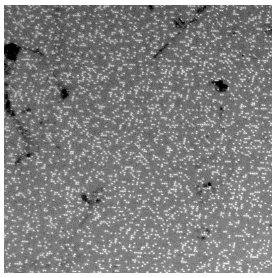
For the experiments, the optical flow algorithm parameters that affect performance were set to the same value for every algorithm. This allows a fair CPU execution time comparison between the algorithms. Other parameters were set according to best practice, as for instance suggested by the OpenCV documentation. Some parameters were adjusted by hand in order to increase flow recognition quality. The range of the scanned neighborhood parameters were set according to known information about the maximum flow range, as listed in Table 1.

The first performed experiment was a CPU processing time comparison. Table 2 lists the parameters used. An Intel i5-4300U based system was used to perform the tests. For the experiment the grass set was used. 50 frames were processed in order to measure the average flow computation time.

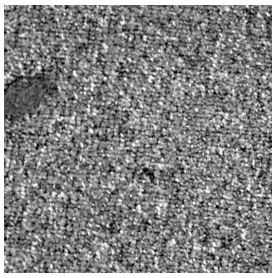
The Lucas-Kanade algorithm requires the locations of features that should tracked. For finding suitable features the Shi-Tomasi corner detector algorithm [10] was used. The execution time of this algorithm was determined for the same binning factors as the optical flow algorithms.



(a) Grass. By Joshua Ezzell (Own work) [CC BY-SA 2.0 (https://creativecommons.org/licenses/by/2.0/)], via Flickr



(b) Dotted floor. The image contrast is enhanced for visibility reasons



(c) Carpet. The image contrast is enhanced for visibility reasons

Fig. 4: First crops of the three ground surfaces

The second experiment was a comparison on the quality of the optical flow algorithms with different types of floor. For determining the quality, the estimated movement of the centre point of the frame was tracked. As error value the average accumulated distance between sequential middle points was calculated, as seen in equation 7. The measure averages out standard normal distributed errors. This measure was chosen since in a real world application the movement of a UAV, in order to get the current location, is accumulated as well.

$$e = \frac{c}{N} || \sum (\mathbf{t}_{et} - \mathbf{t}_{gt}) || \tag{7}$$

In equation 7, N is the number flow estimations, $\mathbf{t}_{et}$ is the estimated translation, $\mathbf{t}_{gt}$ is the ground truth translation and c is the cropping factor.

4 Results

This section describes the results and gives an interpretation. First the results for the CPU-time comparison are given followed by the quality comparison results.

4.1 Computation time comparison

The results of the computation time algorithm are displayed in Figure 5. The results for Block Matching with a smaller binning size than four were left out, since these computation times were a magnitude higher than applicable in the field. The Shi-Tomansi algorithm was added individually to the graph instead of summing the algorithm together with Lucas-Kanade, since the Shi-Tomansi does not have to be used each iteration when applying Lucas-Kanade. It only has to be used when (too many) current good features move out of frame.

The graph shows that the Lucas-Kanade and Farnebäck algorithm outperform simple Block Matching in terms of execution time. The distinction between Lucas-Kanade and Farnebäck is harder to make because of the logarithmic scaling. Table 3 shows the original timings. From this table the differences in computation times are clearly distinguishable. Farnebäck is faster when binning with atleast a factor of 2 is applied.

Table 2: Several of the used parameters for comparing the differences in CPU processing time

	Block Matching	Lucas-Kanade	Farnebäck
binning	-	-	-
search area	43	43	43
block size	3	3	-
pyramid levels	-	3	3
binning	2 x 2	2 x 2	2 x 2
search area	22	22	22
block size	3	3	-
pyramid levels	-	3	3
binning	4 x 4	4 x 4	4 x 4
search area	11	11	11
block size	3	3	-
pyramid levels	-	3	3
binning	8 x 8	8 x 8	8 x 8
search area	6	6	6
block size	3	3	-
pyramid levels	-	3	3
binning	16 x 16	16 x 16	16 x 16
search area	3	3	3
block size	3	3	-
pyramid levels	-	3	3

Table 3: Computation time for several binning factors of Farnebäck and Lucas-Kanade

	computation time in milliseconds	
binning factor	Farnebäck	Lucas-Kanade
1	384.8	327.7
2	53.0	70.0
4	12.3	22.8
8	3.1	5.2
16	0.7	1.2

4.2 Quality comparison

The results of the quality comparison are presented in table 4.

For the dotted floor and carpet data sets the *findHomography()* function was used instead of *estimateRigidTransform()*. The resulting flow for these sets were sometimes too far off for *estimateRigidTransform()*. This occurred specifically for the Lucas-Kanade algorithm and the Block Matching algorithm. Figure 6 shows one of these situations.

All tested algorithms perform reasonably well on the natural grass data set. The errors are reasonably small, since the real average Euclidean flow distance of the grass set is *16.0 pixels*. Figure 7 visualizes the detected movement of the algorithms. The 8 x 8 binning measurements were used to draw the image.

The rotation of the ground truth was used. Further analysis of the results show that rotations are not properly detected, as seen in Figure 8. As soon as a rotation is introduced, the rotational error drastically increases, considering the simulator rotates with 3 degrees per frame, when it rotates. Accumulating the rotation would lead to wrong flight paths. This error is introduced by the functions that estimate the global image transformation. In an real wold UAV application a magnetometer can be used.

The dotted floor proved to be harder to detect for all algorithms. Farnebäck and Lucas-Kanade still perform reasonably well with a binning of 4 and 8, considering the real average Euclidean flow distance is *16.0*. The poor performance of Block Matching can be explained, because most of the dotted floor surface pixels have (near) the same brightness value. Since Block Matching only searches locally, the algorithm will often assign a wrong flow vector.

The Lucas-Kanade algorithm and Farnebäck's algorithm fail when a 16 x 16 binning is applied. Running the Shi-Tomansi separately showed that, without binning, the dots were detected as corners. With

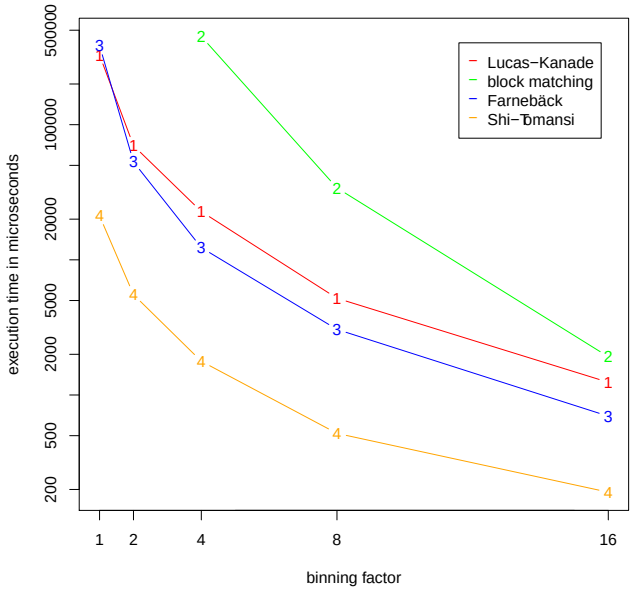


Fig. 5: The computation time according to the binning factor for the three compared algorithms. A logarithmic scaling is used on the execution time axis

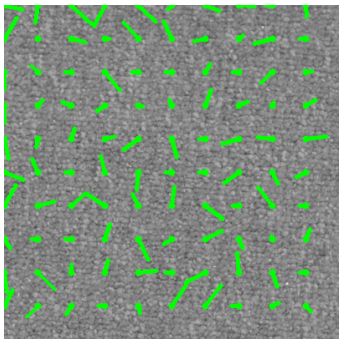


Fig. 6: Wrongfully detected flow by the Block Matching algorithm. The actual movement is a x-axis translation to the right

the highest binning applied however, the dots were to small for the Shi-Tomansi algorithm to detect.

The flow of the carpet data set is well detected at a binning factor of 4. When a binning factor of 8 is applied, Shi-Tomasi does not find enough good points. The Lucas-Kanade algorithm therefore sometimes returns less than 4 flow vectors, which is not enough for estimating the homography transformation. The flow is set to zero in such cases, which leads to high errors in table 4. The same holds true for a binning factor of 16.

Block Matching performs reasonably well with 4 x 4 and 8 x 8 binning, while Farnebäck performs excellent for all binning factors. Figure 9 shows the recognized paths for the algorithms when a 8 x 8 binning is used. The ground truth rotation is used again for the heading of the paths.

5 Conclusion and Discussion

This paper shows that choosing between optical flow algorithms for UAV position change measurement can lead to a decrease of CPU-time usage. The more sophisticated algorithms: Lucas-Kanade and Farnebäck are significantly faster than Block Matching, especially for larger picture sizes.

Table 4: Quality measurements of the algorithms on the different floors. For (R), transformation is calculated with *estimateRigidTransform()* and for (H), transformation is calculated with *findHomography()*.

	binning		
	4 x 4	8 x8	16 x 16
grass			
Block Matching (R)	0.65	0.24	0.19
Lucas-Kanade (R)	0.17	0.18	0.47
Farnebäck (R)	0.17	0.16	0.14
dotted floor			
Block Matching (H)	5.41	2.34	7.70
Lucas-Kanade (H)	0.31	0.63	2.07
Farnebäck (H)	0.75	0.83	1.43
Farnebäck (R)	0.40	0.56	1.80
carpet			
Block Matching (H)	0.17	0.72	2.42
Lucas-Kanade (H)	0.21	18.14	51.44
Farnebäck (H)	0.49	0.49	0.12
Farnebäck (R)	0.27	0.28	0.10

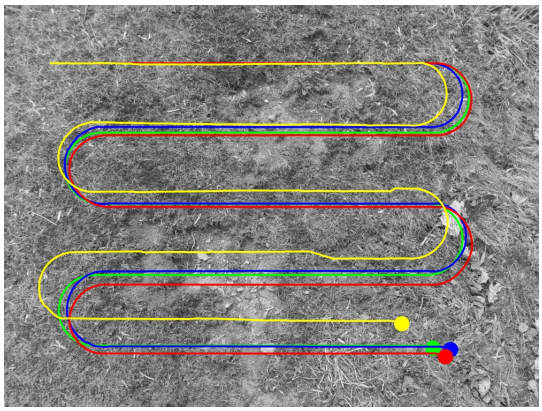


Fig. 7: Visualization of the detected paths for the 16 x 16 binned frames from the grass data set. green is the ground truth, blue is Block Matching, red is Farnebäck and yellow Lucas-Kanade.

Experiment results show that the algorithm's flow recognition quality differ per surface. Also the best algorithm differs per data set and binning factor. Choosing between algorithms can therewith increase optical flow estimation quality. The best optimal flow quality can be obtained by switching between algorithms for different undergrounds.

There is no overall winning algorithm, although in general the Farnebäck method performs best. An surprising result is that the recognition quality does not always degrade by using a higher binning factor. This is particularity interesting because higher binning factors lead to lower CPU-time usage.

During the research a testing framework was developed, which allows fast comparisons between the different optical flow algorithms. Also new data sets for new floor surfaces can be easily generated. The framework can easily be extended in order to test new optical flow algorithms and is therefore also useful for further research.

Due to the low number of examined surface types, the finding that Farnebäck's method performs best cannot directly be generalized to all surface types. Also the simulator did not incorporate scaling. Different height settings may lead to different results.

The compared algorithms were only tested to a certain extent but considering the amount of parameters, performance of in particular Gunnar Farnebäck's algorithm and the Lucas-Kanade algorithm can be further optimized. Also only one feature detection algorithm was used for Lucas-Kanade.

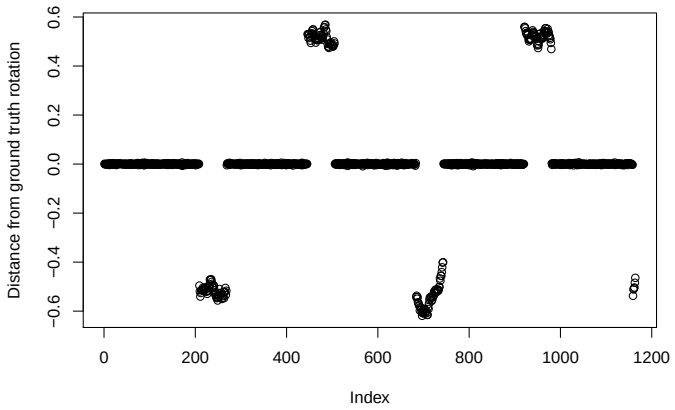


Fig. 8: rotational error in degrees. The index axis corresponds to the index of the frame. The results from Farnebäck with a binning factor of 8 on the grass data set were used

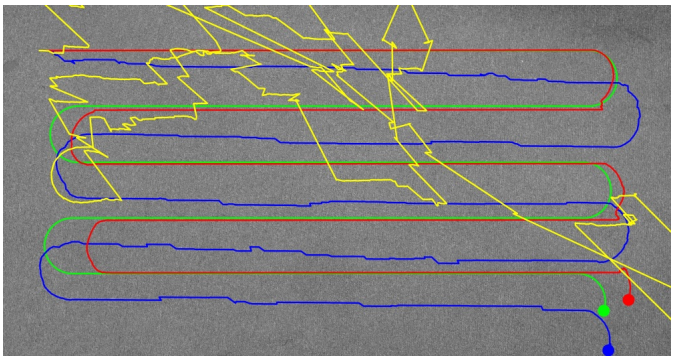


Fig. 9: Visualization of the detected paths for the 8 x 8 binned frames from the carpet data set. green is the ground truth, blue is Block Matching, red is Farnebäck and yellow Lucas-Kanade.

6 Future work

The presented research only compared three algorithms. More algorithms can be taken into consideration. Since the developed framework is designed to be extended, new algorithms can be tested. in a short amount of time. Other interesting algorithms might for instance be simple flow [12] and the Horn-Schunck method [6].

The application that takes frames out of a big picture can be extended to support full affine transformations instead of only similarity transformations. More data sets can be generated and tested to be able to further generalize the results of the experiments.

The results show that the used transformation estimation functions were not able to detect rotations correctly. A magnetometer can be used as an alternative, however, in situations where the earths magnetic field is disturbed by the environment, values from a magnetometer are not reliable. Further research is necessary to see if rotation estimation from optical flow can be improved.

7 Acknowledgments

The authors would like to thank the expert reviewer, prof. dr. M. Biehl and the anonymous reviewers who reviewed drafts of this paper. This research was partially funded by *Nationaal Regieorgaan Praktijkgericht Onderzoek SIA* under project: *Smart Vision voor UAVs*.

References

[1] M. J. Black and P. Anandan. A framework for the robust estimation of optical flow. In *Computer Vision, 1993. Proceedings., Fourth International Conference on*, pages 231–236. IEEE, 1993.

[2] G. Bradski. The opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek. A brief introduction to opencv. In *MIPRO, 2012 Proceedings of the 35th International Convention*, pages 1725–1730. IEEE, 2012.

[4] H. D., M. L., T. P., and P. M. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. *ICRA2013*, 2013.

[5] G. Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*, pages 363–370. Springer, 2003.

[6] B. K. P. Horn and B. G. Schunck. Determining optical flow. *ARTIFICAL INTELLIGENCE*, 17:185–203, 1981.

[7] L. Jayatilleke and N. Zhang. Landmark-based localization for unmanned aerial vehicles. In *Systems Conference (SysCon), 2013 IEEE International*, pages 448–451. IEEE, 2013.

[8] B. D. Lucas, T. Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[9] H. Romero, S. Salazar, and R. Lozano. Real-time stabilization of an eight-rotor uav using optical flow. *IEEE Transactions on Robotics*, 25(4):809–817, August 2009.

[10] J. Shi and C. Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, June 1994.

[11] D. Sun, S. Roth, J. Lewis, and M. J. Black. Learning optical flow. In *Computer Vision–ECCV 2008*, pages 83–97. Springer, 2008.

[12] M. W. Tao, J. Bai, P. Kohli, and S. Paris. Simpleflow: A non-iterative, sublinear optical flow algorithm. *Computer Graphics Forum (Eurographics 2012)*, 31(2), May 2012.

[13] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, 1991.

[14] J. van de Loosdrecht, K. Dijkstra, J. H. Postma, W. Keuning, and D. Bruin. Twirre: Architecture for autonomous mini-uavs using interchangeable commodity components. *IMAV 2014*, 2014.