On-board real-time tracking of pedestrians on a UAV

Floris De Smedt, Dries Hulens, and Toon Goedemé
ESAT-PSI-VISICS, KU Leuven, Belgium
firstname.lastname@kuleuven.be

Abstract

Recent technical advances in Unmanned Aerial Vehicles (UAV) made a realm of applications possible. In this paper we focus on the application of following a walking pedestrian in real-time, using optimised pedestrian detection and object tracking. For this we use an on-board embedded system, offering an optimal ratio of computational power and weight. We extend the commonly used ground plane estimation technique, used to reduce the search space, based on the sensor data off the UAV. The integration of the ground plane constraint obtains a significant speed-up over the already optimised Aggregate Channel Feature (ACF) detector. To compensate for the frames without detections, we use a particle tracker based on color information. We successfully validated our system on a flying UAV.

1. Introduction

During the past years, UAVs have gained the attention for many applications, both for industrial and consumer use, such as aerial photography, surveying, cinematography, surveillance applications and rescue operations. Commonly the UAV is manually controlled using a remote control. In this paper we focus on the application of automatically following pedestrians using a camera connected to an on-board embedded system, which is selected based on its computational power and weight criteria. To obtain an accurate pedestrian location in each frame, we combine a pedestrian detector with a particle tracker based on color information. Since these algorithms are executed on an on-board embedded system, the results can directly be used to steer the UAV without human intervention. To improve both accuracy and processing speed of our pedestrian detection algorithm, we extend the commonly used ground plane estimation technique to a more generalised form based on the UAV sensor data, which is used as a search constraint. The technique described in this paper can be used in a lot of applications such as: cinematography (automatic filming of *e.g.* extreme outdoor sports), automatic patrolling for intruder detection, following in surveillance applications and guardian angel or outdoor monitoring of elderly and children.

In this paper we propose three contributions to the state-of-the-art. Firstly, we selected an optimal embedded system based on both required computational power and weight criteria to execute our optimised state-of-the-art algorithms in real-time, on-board on a UAV. On the second hand we propose a reformulation of the ground plane assumption such that it elegantly takes into account constraints and is parametrisable with instantly measured altitude and rotation values of the UAV. Finally, by combining our optimised and constrained pedestrian detection algorithm with a color based particle filter, we allow a continuous PID-controlled steering of a real drone, which we validated with real-life experiments.

This paper is structured as follows. In section 2 we discuss the current state-of-the-art on pedestrian detection, ground plane estimation and UAV applications. In section 3 we describe the pedestrian detection algorithm we use, and our approach to integrate a ground constraint to improve accuracy and detection speed. Next to this, we describe the extension of it to the 6 DoF we are dealing with when the video is taken from a UAV. In section 4 we describe the implementation of the particle filter we use. In section 5 we describe both the hardware and the software construction for our application. In section 6 we validate both the impact of using a ground constraint on the Caltech dataset [9] and our complete system to automatically follow pedestrians on real-life experiments. Finally we conclude our work in section 7.

2. Related work

Pedestrian detection receives a lot of attention in current literature, concerning both improvements in speed and accuracy. In 2004, Viola and Jones [15] proposed a technique of using Haar wavelets to detect faces in real time. The use of Integral Images made the processing speed independent of the wavelet-size. In 2005, Dalal and Triggs [4] proposed a technique where pedestrians where modelled based on contrast information in the images. The two latter techniques are still used as a reference for accuracy comparison. In 2009, Dollár *et al*. [8] published their work on *Integral*

Channel Features which extends the gradient features with color information. In this work, the features are represented by *Channels*. From a pool of randomly generated rectangles from inside the model size, the best features are selected using AdaBoost. To improve the speed of this detector, they proposed to approximate features at multiple sizes from features at other sizes instead of calculated from image data [7]. In 2012, R. Benenson *et al*. proposed a technique [2] where instead of evaluating a single model on multiple feature layers, a set of models is created which are evaluated on the same features, hereby using the approximation approach of [7] for model approximation instead of feature approximation. This multi-model approach was combined with the use of stixels, indicating possible positions of pedestrians in the image based on ground plane estimation. Using GPU hardware to exploit the created parallellisation opportunities leaded to pedestrian detection at 100 frames per second. In 2013, the training process of *Integral Channel Features* was optimised with the focus on accuracy by [3], leading to state-of-the-art detection results. Recently, Dollár proposed his *Aggregate Channel Feature* (ACF) approach [6], which was a combination of a generalised feature approximation technique based on [7] and using all possible rectangles of a rigid grid instead of a randomly generated pool of rectangles. The reduction of the amount of features, combined with the improved AdaBoost training method described in [1] resulted in a detector which was both accurate and fast in both evaluation and training, even on single core hardware. Complementary to the pedestrian detection approach used, multiple techniques can be used to improve evaluation speed, including the use of GPU hardware, which exploits data parallellisation opportunities, and on the other hand limiting the search space. As mentioned before, the technique described in [2] uses both GPU hardware and ground plane estimation based on stereo images. In 2013, De Smedt *et al*. proposed an approach [5] where the pedestrian detection was optimised by using both GPU and CPU together as a hybrid system. By combining this with the warping window approach [14] as a generalised ground plane estimation approach and object tracking, they reduced the searching process to only a minimum, obtaining 500 detections per second. In 2011, Sudowe and Leibe proposed a technique in [13] to generalise the use of a ground plane in combination with a sliding-window object detector. Based on the homograpy of the ground plane and the real-world size of the object, the area in the image where the object can be found is selected, and used as the input for the object detector. They validate this approach by using a CUDA implementation of the HOG algorithm. In this paper we use a similar approach of exploiting the ground plane information as a constraint. Hulens *et al*. [10] designed a tool to determine the best suited hardware platform for an algorithm at a specified evaluation speed. Next to that, the tool estimates the maximum possible activity of an embedded system, taking into account the battery and power consumption. The requirement of minimum weight and power usage hinders the use of GPU-platforms. Therefore we focus on CPU-based embedded systems and suitable algorithms in this paper. In [11], Naseer *et al*. describe a system to follow pedestrians using a quadrocopter. They use two cameras, one for determining the 3D position of the UAV based on markers on the ceiling. The second camera is a depth camera, which is used to detect a person in 3D. The image from the depth camera is warped based on the calculated 3D position. Part of the calculations are performed on a ground station. Recently Pestana *et al*. [12] proposed a system similar to ours where a UAV is used to track and follow objects of various kinds (pedestrians, cars, ...). They still use a wirelessly connected computer to do the necessary calculations to perform the steering of the UAV, and also require an online learning stage. In our system, the UAV is fully autonomous, since all processing is performed on an on-board embedded system using a single camera. Next to that we use a general applicable pedestrian model (although models for other objects could be used) such that we do not require a learning stage. Our system is not restricted to labo environment.

3. Pedestrian detection

In this section we describe how we optimise the pedestrian detection algorithm. In subsection 3.1 we give a more detailed overview of the ACF algorithm we use. In subsection 3.2 we describe how ground plane information can be exploited by restricting the search space based on the scale the searched object can appear on each location. This general applicable approach will be extended in subsection 3.3 where we reformulate the calibrated approach by parametrizing it with measured sensor data from the UAV, making calibration superfluous.

3.1. Aggregate Channel Features

As described in section 2, the *Aggregated Channel Features* approach proposed in [6] reaches high detection speed, even with a limited amount of computational resources. Therefore it forms a suitable starting point for our application. A matlab implementation of this algorithm is publicly available as part of *Piotr's Computer Vision Matlab Toolbox*[1].

For our application we ported this implementation to C++ for performance considerations. Hereby we reused the mex-implementations of the different channel features as used in the original implementation. Since *ACF* uses a rigid grid to select the feature locations, it avoids the need for integral image calculations. The features are formed by single pixel lookups from the channels.

[1] http://vision.ucsd.edu/~pdollar/toolbox/doc/index.html

Below we describe how a ground plane constraint can be integrated with this algorithm. Take in mind that our approach is independent of the pedestrian detector used.

3.2. Exploiting ground plane constraint

Ground plane estimation is the technique to approximate the orientation of the ground plane in 3D relative to the position of the camera. This allows to approximate the height of an object for each position in the image and thus enables a restriction on the search space. In [13] is described how the object height in an image can be predicted based on the homography of the ground plane and the real world size of the object to detect. Based on the desired variance on the real world size, *e.g.* detecting pedestrians between 160cm and 185cm, rectangular regions are cut from the image and transformed to a standard size, where each region should be evaluated at the same scale.

As a first approximation, we model the ground plane constraint of the Caltech dataset [9] by a first order linear relationship between the y-position of the feet of the pedestrian in the image and their height, instead of using a homography of the ground plane. Figure 1 visualises the ground plane estimation function in black, which is obtained by fitting a curve through the annotations from the Caltech training set (blue circles). As can be observed, a lot of annotations do not comply exactly with this function. This is due to two main reasons: the variation in object height and the tilt movement of the camera relative to the ground plane (pitch angle around the y-axis in figure 3). For small angles, this can be compensated by using an offset of our calibrated function (visualised as the cyan lines). The red lines visualise the compensation of the pedestrian height variation on top of the compensation for camera movement.

As can be seen in figure 1, we now can determine the image region each specific scale should be searched in. The boundaries of this region are defined by the intersection of the object height with the red lines, which indicate the combination of both compensations (the height variation and camera movement). Take in mind that these y-values indicate the location on the ground plane, and so this region is extended with the object size at that location. This is visualised in figure 2 for an object size of 150px. Since this process reduces the search space from a full feature layer to only a limited area, this evidently leads to improved speed results.

The use of a ground plane constraint benefits both the detection speed, by reducing the search space, and the accuracy of the detection process, since false detections that do not fall within the searching space for that scale, are avoided. In section 6 we validate this using the Caltech dataset (see figure 9).

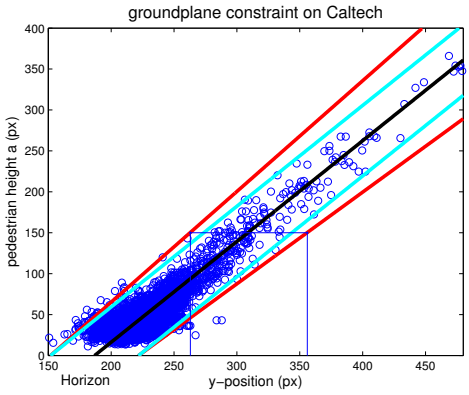

Figure 1: The ground plane estimation as used in our Caltech experiment. The black line is our estimation function, while the red lines indicate the variance allowed on the scaling.

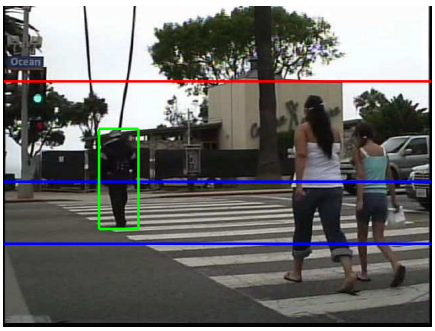

Figure 2: The region we crop to search for a pedestrian size of 150px. The blue lines indicate the boundaries on the ground plane, while the red line takes the object size itself also into account. An example detection is indicated with a green bounding box.

3.3. Using a ground plane constraint with 6 DoF

In our application we perform pedestrian detection on images taken from a flying UAV. The 6 degrees of freedom (DoF), visualised in figure 3, make it impossible to apply a pre-calibrated ground plane estimation function. Happily enough the ground plane function is only influenced by 3 DoF: rotation around the x- and y-axis (called the roll and pitch respectively) and translating along the z-axis (altitude). Translating along the x- and y-axis and rotation around the z-axis do not influence the ground plane estimation function and so they can be ignored. The IMU of the autopilot measures the values corresponding to these DoF, enabling us to compensate for them and derive a linear

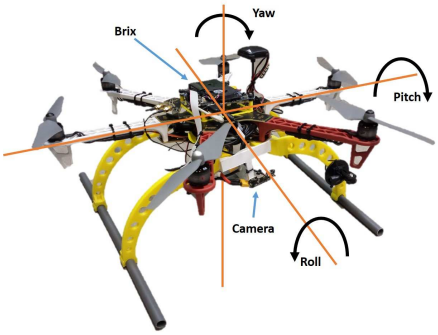

Figure 3: Our UAV system with the degrees of freedom shown, translating and rotating over the 3 axes.

ground plane constraint function.

We will first reformulate the ground plane constraint such that the UAV's altitude can be used as an input parameter. Figure 4 visualises the scenario of a flying UAV with a forward looking camera based on the pinhole model. The further away the pedestrian is located, the smaller, and the closer to the horizon, it will appear in the image. Figure 5 shows the corresponding frame as captured by the camera on the drone. The y-position of the pedestrian's feet (position on the ground plane) can be calculated by:

$$y = \frac{h_{im}}{2} + \frac{a}{A}B \tag{1}$$

where h_{im} is the height (in pixels) on the image. Hereby the assumption of a flat ground plane is made, which imposes the y-position of the horizon in the middle of the image.

For this equation, we know all parameters up front except for the real-world flying height (B) of the UAV. By measuring the altitude of the UAV, we can obtain the parameters of the first order function and reuse the approach we described in subsection 3.2.

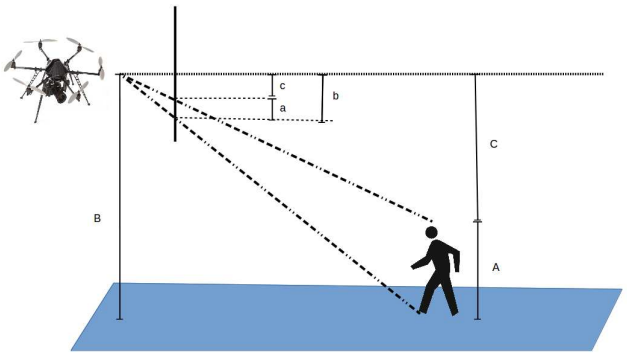

Figure 4: Side view of the scene with a forward looking camera.

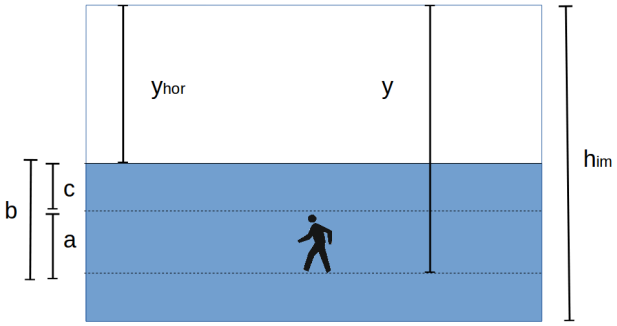

Figure 5: The image content as captured by the camera

The *roll* of the UAV is easy to compensate for, since this only requires rotating the image with the inverse angle, making the horizon to lie horizontally again. The last degree of freedom we have to compensate for is the *pitch*, with the camera looking slightly upwards (or downwards) instead of straight forward. The equations of the two red lines in figure 1 become now:

$$a = y\frac{A_{\text{MIN}}}{B} - h_{\text{hor,MAX}} \tag{2}$$

$$a = y\frac{A_{\text{MAX}}}{B} - h_{\text{hor,MIN}} \tag{3}$$

, where B is the measured altitude of the UAV, A_{MIN} and A_{MAX} the minimum and maximum real-world height of the pedestrians to be detected, and $h_{hor,MIN}$ and $h_{hor,MAX}$ the y-position of the horizon in the image, deviating from the ideal $h_{hor} = \frac{h_{im}}{2}$ value due to pitch angle effects.

When the pitch angle (α) can be measured, as in our UAV application, another solution to cope with this deviation is transforming the image back to a zero-pitch image. Again using the pinhole camera model (as shown in figure 6), this transformation boils down to:

$$y_2 = f\frac{f.sin(\alpha) + y_1.cos(\alpha)}{f.cos(\alpha) - y_1.sin(\alpha)} \tag{4}$$

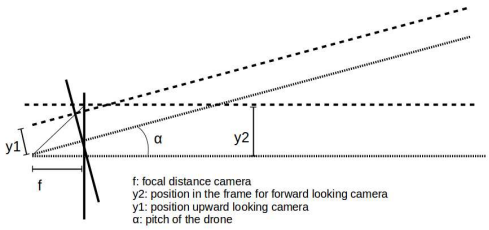

Figure 6: Compensate for the *pitch* by projecting a point to the forward looking view.

Based on the previous equations, we extended our ground plane constraint from subsection 3.2 to cope with the extra degrees of freedom of a UAV. In section 6 we will evaluate this approach for our application.

4. Particle tracker

Although the pedestrian detection methodology we described in the previous section obtains good results, it does not form a reliable methodology on its own. Due to the variations in appearance a pedestrian can have, it is impossible to obtain a 100% reliable detection in each frame. Lowering the threshold will increase the recall and thereby the chance of detecting the pedestrian to follow, but will also increase the amount of false detections. Moreover, detecting at low thresholds requires a less strict rejection threshold used for the cascade, which implies less pruning during the detection pipeline, with an increased evaluation time as a consequence. Evidently this is to be avoided for time-critical applications such as ours. Therefore we use a tracker to compensate for the missed detections.

The implementation we created is based on the public available implementation of Kevin Schluff[2]. Each particle represents a state: $S = \{x, y, v_x, v_y, s\}$, where x and y represent the position of the particle, v_x and v_y represents the moving speed of the particle, and s represent the relative scaling from the initial object size.

A state-update is performed on each frame. The state of each particle is updated using a constant velocity transition matrix as given by equation 5. The new position of the object is determined by taking the weighted mean of the particles. The confidence (C) of each particle is calculated using equation 6, which uses the Bhattacharyya distance (D) between the color histogram of the model and the color histogram of the particle's position.

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$C = \exp(-20.D^2) \tag{6}$$

To avoid the influence of background noise while tracking, we initialise the tracker not on the full detection rectangle, but on the chest. The chest does only move a little compared to for example the legs, and it is fairly easy to select an area on the chest which contains almost solely pixels belonging to the detected pedestrian.

[2] https://bitbucket.org/kschluff/particle_tracker

5. Integration

5.1. System overview

Figure 8 gives an overview of the hardware we use. To demonstrate our framework we used the F550 hexa-copter from DJI with the Pixhawk as stability controller. We equipped the F550 with the Brix computing module (Intel-I7 processor, 8G RAM and 100G HD with 172g of weight) to run our framework on. The choice of using the Brix is based on the tool developed by Hulens *et al*. [10]. For our application, the flight-time is estimated on 12 minutes when the algorithm runs at 15fps. As seen in figure 8, the Brix is communicating with the Pixhawk using the *Mavlink protocol*. By this communication protocol the sensor data including pitch, roll and yaw-angles as well as the altitude are retrieved from the Pixhawk. To control the F550 from the Brix, several changes in the Pixhawks firmware were made to receive Mavlink control messages from the Brix instead of from the remote control.

To combine all previous steps we developed a software framework, of which an overview is shown in figure 7. Each of these blocks is executed in a separate thread. The *Capture Frames* retrieves the frames from the camera. When the frame is retrieved, this thread requests the measured sensor-data from *Read Sensordata UAV*, which maintains communication with the Pixhawk using *Mavlink messages* which can be seen in figure 8. The *roll* is corrected by rotating the retrieved frame based on the measured sensor-data. Both the roll-corrected image and the sensor-data are passed to *Person Detection* and *Person tracking* using a queue.

The *Person Detection* performs the detection of pedestrians, using the ground constraint based on the sensor-data, as described in subsection 3.3. At the same time, the *Person Tracking* performs a state-update to obtain the most probable position of the person using color-information based on 150 particles, as described in section 4. The results of both the person tracker and the person detector are passed to *Combine Results Control UAV*, which combines them to a single location. This is described in more detail in subsection 5.2.

The task of our UAV demonstrator is to steer the UAV such that a tracked pedestrian is kept in view, and therefore in the center of the camera image.

The difference between the desired position and the calculated position of *Combine Results Control UAV* is used to steer the UAV using a *PID control loop*. This latter calculates a smooth control value to steer the UAV, depending on the size of the error and the speed the errors changed in time. *PID control loop* maintains a continuous communication with the *Pixhawk*, which on his turn controls the motors of the UAV.

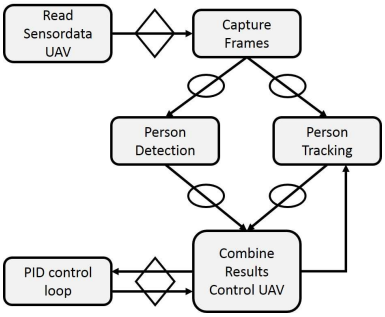

Figure 7: Overview of framework. Oval=queue, Diamond=mutex

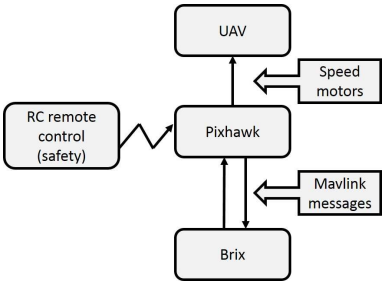

Figure 8: Overview of UAV.

5.2. Combination of pedestrian detection and object tracking

We combine the detections of the pedestrian detector and the particle filter into a single hypothesis of the pedestrian's location. The tracker is controlled from *Combine Results Control UAV*. After the UAV takes off, the tracker is initialised on the strongest detection found. Hereby, the target pedestrian to follow is selected. From then on, the tracker is reinitialised using the pedestrian detections found by the pedestrian detector, which is performed in two cases:

1. When the tracking confidence drops below a 50% confidence, we search for detections with an overlap of at least 50% with the current tracking rectangle. The confidence score is calculated using the same method as for the particles, given by equation 6.

2. When the confidence of the tracker becomes very low we suspect the tracker to be drifted away. Therefore we calculate the confidence on all detections in the image. The tracker is reinitialised on the detection with the highest similarity between the detection box and the model.

In both cases, when no detection is found matching the criteria, no reinitialisation of the tracker is performed. Take in mind we are currently focussing on tracking just one pedestrian, and this approach should be altered slightly to cope when multiple pedestrians are walking through the scene.

As we described in section 4 we do not use the original dimensions of a detections to initialise the tracker, but only a small rectangle focussing on the chest of the pedestrian for improved detection results. To perform the overlap criterion, we invert this operation to obtain the dimensions of the tracking rectangle as it would be for a full body tracking rectangle.

6. Experiments

In this section we discuss the experiments we have performed. First, in subsection 6.1 we validate the use of a ground plane constraint using the Caltech pedestrian dataset. In subsection 6.2 we present the results we obtained when using the framework described in section 5 on a flying UAV. All speed results are obtained using the Brix embedded system.

6.1. Ground plane constraint validation

In subsection 3.2 we described how ground plane constraints can be integrated with an object detector to improve both detection accuracy and model evaluation. Figure 9 visualises the accuracy improvement of using a ground plane constraint as compared to an evaluation using all scales on the whole image. These experiments are performed using a model which is trained on the Caltech training set.

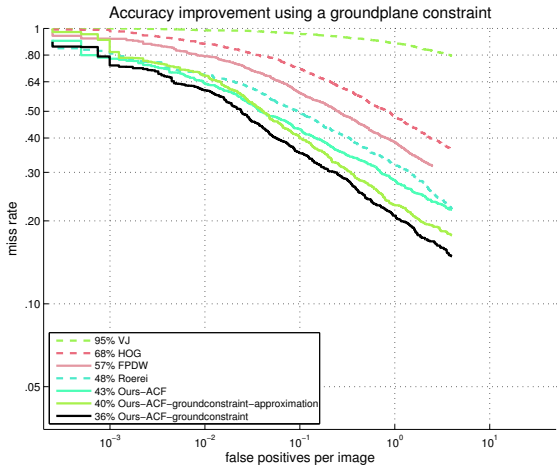

Figure 9: Accuracy improvement on the Caltech dataset by using a ground plane constraint.

In table 1 we compare the detection speed of our implementations. As we can observe, the ground plane constraint brings a significant improvement in evaluation speed considering the combined accuracy improvement. The choice

Technique	Evaluation speed	Speed-up
Matlab-ACF	5.8 fps	0.72×
Ours-ACF	8.1 fps	1×
Ours-ACF-groundconstraint-approximation	31.1 fps	3.9×
Ours-ACF-groundconstraint	27.2 fps	3.4×

Table 1: Comparison of detection speed of the algorithms we provide evaluated on Caltech.

Technique	Evaluation speed	Speed-up
Ours-ACF	14.9 fps	1×
Ours-ACF-groundconstraint-approximation	33 fps	2.2×
Ours-ACF-groundconstraint	21.9 fps	1.47×

Table 2: Comparison of detection speed with the different implementations used for the UAV application. The ground plane constraint is calculated based on he measured sensor data.

of using approximation of features, which is one of the speed improvements of ACF, is dependent on the importance of speed versus accuracy.

6.2. UAV application

To evaluate the system we described in section 5, we performed a number of test flights. For two sequences, we will show detailed results. Example frames of these are shown in figure 10. The first sequence of 751 images shows the performance of our system while following a walking pedestrian from behind. The second sequence of 823 images keeps the UAV more or less at the same spot while following a pedestrian walking in front of the UAV. All the data (the frames, the tracked position and the detections) where saved to disk to allow comparison of different approaches. All the frames where manually annotated. Based on these sequences we will discuss the evaluation speed of the different detection options, as described in section 3, and the accuracy of the position determination of our system. We used a pedestrian detection model trained on the INRIA training set.

In table 2 we present the detection speed of our detector implementations. As can be seen we reach decently high processing speeds on the on-board embedded system we use. Since we are using a 15fps camera, we chose the most accurate implementation, which uses the ground plane constraint without the approximation of features.

By adding the roll-angle to rotate the image, we make sure the frame can be processed while the pedestrians are straight up instead of rotated. This is visualised in figure 11. The use of roll-correction has a high impact on the amount of correct detections. To determine the infuence of roll-correction, we annotated a dedicated experiment of

(a) Before roll-correction (b) After roll-correction

Figure 11: The visual effect of applying roll-correction on the captured frames.

	Sequence 1	Sequence 2
Detection	4.05%	4.59%
Tracking	10.47%	7.51%

Table 3: Comparison of percentage center x-position error w.r.t. the width of the person. Less as 50% means we are still tracking inside the annotation bounding box.

1367 frames. The amount of times the target pedestrian could correctly be detected without roll-correction is 115 times, or 8.4% of the images in the sequence. This is drastically increased to 692 times, or 50% of the images in the sequence when applying the correction. Since the detections are used to improve the tracking, and thus the steering of the UAV, this has a big influence on the resulting tracking performance.

In table 3 we give the accuracy of our system. The accuracy is measured using the deviation of the tracked person's center coordinate with respect to the corresponding annotation, but relative to the width of the annotation, which gives a normalised accuracy measure of how accurately the position of the pedestrian is passed to the PID-loop to steer the UAV.

During processing, the Brix embedded system has a workload of 95% and has a 22W power use.

7. Conclusion

In this paper we described a system to automatically follow pedestrians with a UAV using an on-board embedded system. The location of the pedestrian is determined with a combination of a pedestrian detector and a color based particle filter. The pedestrian detector is used to help the particle filter remain on target. The detection speed and accuracy are improved by using a ground constraint, which we reformulate to be fully parametrisable by the sensor data measured by the IMU of the UAV. Our complete system is successfully validated using a real flying UAV. Hereby we reach real-time processing on the on-board embedded system, while obtaining a normalised accuracy of 10.47%.

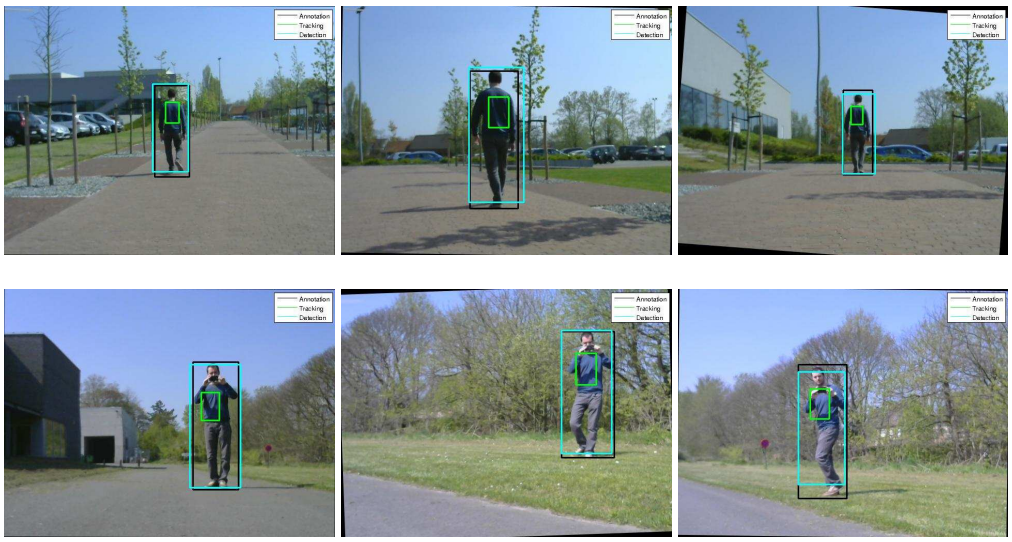

Figure 10: Example images from the two sequences we use to validate our system, black: annotation, cyan: detection, green: tracked position. Top row: sequence 1, bottom row sequence 2.

Acknowledgements

This work is partly funded by KU Leuven via the CAMETRON project.

References

[1] R. Appel, T. Fuchs, P. Dollár, and P. Perona. Quickly boosting decision trees-pruning underachieving features early. In *JMLR Workshop and Conference Proceedings*, volume 28, pages 594–602. JMLR, 2013.

[2] R. Benenson, M. Mathias, R. Timofte, and L. Van Gool. Pedestrian detection at 100 frames per second. In *CVPR, 2012 IEEE Conference on*, pages 2903–2910. IEEE, 2012.

[3] R. Benenson, M. Mathias, T. Tuytelaars, and L. Van Gool. Seeking the strongest rigid detector. In *CVPR, 2013 IEEE Conference on*, pages 3666–3673. IEEE, 2013.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[5] F. De Smedt, K. Van Beeck, T. Tuytelaars, and T. Goedemé. Pedestrian detection at warp speed: Exceeding 500 detections per second. In *CVPRW, 2013 IEEE Conference on*, pages 622–628. IEEE, 2013.

[6] P. Dollár, R. Appel, S. Belongie, and P. Perona. Fast feature pyramids for object detection. *PAMI, IEEE Transactions on*, 36(8):1532–1545, 2014.

[7] P. Dollár, S. Belongie, and P. Perona. The fastest pedestrian detector in the west. In *BMVC*, volume 2, page 7. Citeseer, 2010.

[8] P. Dollár, Z. Tu, P. Perona, and S. Belongie. Integral channel features. In *BMVC*, volume 2, page 5, 2009.

[9] P. Dollár, C. Wojek, B. Schiele, and P. Perona. Pedestrian detection: A benchmark. In *Proc. of CVPR*, June 2009.

[10] D. Hulens, J. Verbeke, and T. Goedemé. How to choose the best embedded processing platform for on-board uav image processing ? In *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. IEEE, 2015.

[11] T. Naseer, J. Sturm, and D. Cremers. Followme: Person following and gesture recognition with a quadrocopter. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 624–630. IEEE, 2013.

[12] J. Pestana, J. L. Sanchez-Lopez, S. Saripalli, and P. Campoy. Computer vision based general object following for gps-denied multirotor unmanned vehicles. In *American Control Conference (ACC), 2014*, pages 1886–1891. IEEE, 2014.

[13] P. Sudowe and B. Leibe. Efficient use of geometric constraints for sliding-window object detection in video. In *Computer Vision Systems*, pages 11–20. Springer, 2011.

[14] K. Van Beeck, T. Goedemé, and T. Tuytelaars. A warping window approach to real-time vision-based pedestrian detection in a trucks blind spot zone. In *Proceedings of the ninth international conference on informatics in control, automation and robotics*, volume 2, pages 561–568, 2012.

[15] P. Viola and M. J. Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.