Choosing the best embedded processing platform for on-board UAV image processing ?

Dries Hulens, Jon Verbeke, and Toon Goedemé

EAVISE, KU Leuven,
Sint-Katelijne-Waver, Belgium
{dries.hulens,jon.verbeke,toon.goedeme}@kuleuven.be

Abstract. Nowadays, complex image processing algorithms are a necessity to make UAVs more autonomous. Currently, the processing of images of the on-board camera is often performed on a ground station, thus severely limiting the operating range. On-board processing has numerous advantages, however determining a good trade-off between speed, power consumption and weight of a specific hardware platform for on-board processing is hard. Many hardware platforms exist, and finding the most suited one for a specific vision algorithm is difficult. We present a framework that automatically determines the most-suited hardware platform given an arbitrary complex vision algorithm. Our framework estimates the speed, power consumption and flight time of this algorithm for multiple hardware platforms on a specific UAV. We demonstrate this methodology on two real-life cases and give an overview of the present top processing CPU-based platforms for on-board UAV image processing.

Keywords: UAV, Vision, On-board, Real-Time, Speed Estimation, Power Estimation, Flight Time Estimation

1 Introduction

Nowadays UAVs (Unmanned Aerial Vehicles) are used in a variety of tasks such as surveillance, inspection, land surveying,... They are mostly manually controlled remotely or follow a predefined flight path, while collecting interesting images of the environment. These images are often analyzed offline since the processing power of these UAVs is limited. Otherwise a wireless link is provided to do the processing of the images on a ground station giving the instructions to the UAV. To be more autonomous and operate more robustly, UAVs should be equipped with processing power so that images can be processed on-board. This will ensure that UAVs can analyze and react in real-time on the images and that they can fly much further since a wireless link is not necessary. Recent advances concerning embedded platforms show an ongoing increase in processing power at reasonable power consumption and weight. Currently, it even becomes possible to employ these complex hardware platforms under UAVs. However, since various parameters need to be taken into account, finding an optimal hardware platform

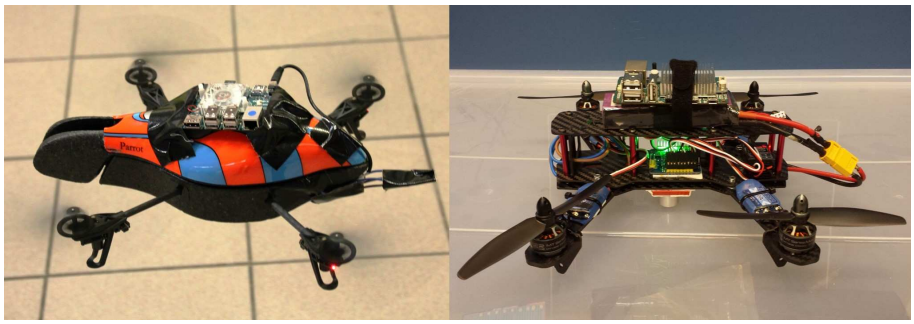

Fig. 1. Parrot AR Drone (left) and XBird 250 (right) carrying an Odroid hardware platform for real-time vision processing.

for a specific algorithm is not trivial. Example applications that need on-board complex image processing are e.g. visual SLAM for 3D sense and avoid, the detection and tracking of people for surveillance purposes, navigating through the corridor between trees in an orchard for counting fruit, the automation of a film crew by UAVs, a vision-based navigation system to automatically clean solar panels,... Determining the optimal trade-off between the processing capabilities and the physical constraints is a daunting task because of their variety. Therefore, in this paper we answer the question: *Which hardware platform is best suited to perform a particular image processing task on a UAV?* A hardware platform can be a simple embedded processor (e.g. a Raspberry PI) or even a small computer like a laptop, depending on the processing power that is needed. Using these under a UAV impose severe constraints on the hardware platforms: they should be lightweight, small and have adequate processing power at low power consumption to maintain long flight times. To determine the effective processing speed of a particular algorithm on a specific hardware platform, one should implement the algorithm on each specific platform. Acquiring a large variety of test platforms to determine the most suitable one evidently is not time nor cost efficient. Therefore, in this paper we present a framework that, given a specific algorithm, estimates the processing speed, power consumption and flight time on a large set of hardware platforms, without the need to acquire any of them. For this we rely on two benchmark algorithms. This paper provides data for a number of hardware platforms only restricted in the fact that they are CPU-based. However since our framework is generic, new platforms can easily be added to the framework. An overview of the platforms that we have included can be found in Table 1. The framework will be evaluated on two real cases. In the first case we track a person with a UAV using a face detection algorithm [2]. For this, we search for a hardware platform that can run the face detector at *4fps* while minimizing the power consumption (e.g. maximum flight time). In our second case the UAV should visually navigate through a fruit orchard corridor, running a vantage point detection algorithm [1] on-board at *10fps*.

The main contributions of this paper are:

Name	Processor	Memory	Weight (gram)	Power (Watt)	Volume (cm^3)
Desktop	Intel I7-3770	20GB	740	107	4500
Raspberry PI	ARM1176JZF-S	512MB	69	3,6	95
Odroid U3	Samsung Exynos	2GB	52	6,7	79
Odroid XU3	Samsung Exynos	2GB	70	11	131
Jetson	Cortex A15	2GB	185	12,5	573
mini-ITX atom	Intel Atom D2500	8GB	427	23,5	1270
mini-ITX I7	Intel I7-4770S	16GB	684	68	1815
Nuc	Intel I5-4250U	8GB	550	20,1	661
Brix	Intel I7-4500	8GB	172	26	261

Table 1. Overview hardware platforms that we have tested for our framework.

- State-of-the-art overview of the current best CPU-based processing platforms for complex image processing on-board a UAV.
- Present experimental results of benchmark computer vision experiments on each of these state-of-the-art platforms.
- We propose a generic model to estimate the processing speed, power consumption and UAV flight time of any given image processing algorithm on a variety of hardware platforms.
- Validation of the proposed generic model on two real cases (people detection/tracking and vision-based navigation).

This paper is structured as follows: in the next section we give an overview of the related work on this topic. In section 3 we briefly discuss the hardware platforms that we used in the framework. In section 4 we present our framework and in section 5 we verify our framework with some experiments and show our results.

2 Related Work

Currently, UAVs are often used to capture images of the environment which are then processed afterwards e.g. surveying [12]. For this the UAVs are controlled manually or by means of GPS. However, our main focus is on autonomously flying UAVs. To enable this, UAVs mainly rely on vision algorithms. Therefore, algorithms like path planning and obstacle avoidance (e.g. object detection) are used to steer the UAV to a certain position [7, 13, 14]. Due to their computational complexity, on-board UAV processing is often practically unfeasible. Therefore, in these approaches, a ground station (with desktop computer) is used to process the images and steer the UAV. However this severely limits their operating range. In cases where on-board processing currently is employed, only light-weight algorithms are used. For example [10] use sky segmentation (color segmentation), running on a Pentium III processor, to detect and avoid objects in the sky. [8] use a marker detection system to follow a predefined path.[17] use line detection, running on a Cortex-A9, for the inspection of pole-like structures. [9] track an IR-LED-pattern mounted on a moving platform, using a ATmega

644P controller and [15] filters laser scanner data on an Atom-based processing platform to estimate crop height. However, our real-life test case algorithms are much more complex. To implement more complex algorithms on a UAV often FPGAs or ASICs are used since they offer an optimal trade-off between weight, power consumption and processing power. [11] designed an FPGA based path planning algorithm, and [6] evaluate other hardware like ASICs as on-board vision processing platform. However, translating e.g. OpenCV code (C, C++ or python) to hardware (using e.g. VHDL) is a tedious and time consuming task. [16] use a high-end processing platform for on-board path planning and obstacle avoidance. This is possible since, in their case, power consumption or weight is less relevant because they use an octacopter with a large carrying capacity. Currently, work exists which achieves real-time performance of complex vision algorithms on UAV mounted embedded platforms [18–20]. However, their algorithms are specifically adapted or designed to perform real-time performance on a targeted hardware platform. We aim to develop a framework that performs the opposite operation; i.e. given a specific algorithm we determine the most suited hardware platform. To resolve all problems mentioned above, in this paper we present a framework that automatically determines the most suitable hardware platform given a user's computer vision algorithm from state-of-the-art, affordable (from $30 to $800), embedded platforms. Our framework enables the use of complex computer vision algorithms which run in real-time on-board of the UAV, directly programmed in OpenCV.

3 State-of-the-art image processing platforms

Nowadays, a number of CPU-based processing platforms are available which are lightweight and powerful and therefore suited for the task at hand. An overview is given in Table 1.We will describe them briefly, in order of ascending processing power (and thus increasing weight). A well-known lightweight processing platform is the Raspberry PI. The PI is a small, low-cost 1GHz ARM11 based hardware platform developed for educational purposes. The main advantage of this small platform is that it runs a linux-based distribution, which allows the compilation and usage of well-known vision libraries e.g. OpenCV. Of course, the processing speed is limited, but simple vision algorithms, like e.g. face detection based on skin color segmentation, run at real-time performance. The PI is equipped with a Broadcom GPU which recently became open-source. A more powerful alternative for the PI is the family of Odroid platforms. One of those platforms is the U3 that is even smaller than the PI and has an ARM based 1.7GHz Quad-Core Samsung processor that is also used in smartphones. Speed tests on the U3 indicated that this platform is *20 times* faster than the Raspberry PI. The XU3 is another Odroid platform which has a Samsung Exynos5422 Cortex-A15 2.0GHz quad core and a Cortex-A7 quad core processor making him two times faster as the U3. The XU3 has a fan to cool the processor where the U3 is passively cooled. Both the U3 and XU3 are equipped with an eMMC slot which is a much faster alternative for the SD card. Another novel and promising

platform is the Jetson TK1 Development Kit with an on-board NVIDIA GPU and a quad-core ARM15 CPU, making the platform especially useful for GPU based vision algorithms. In this paper we only perform experiments on the CPU but in future work the GPU will also be evaluated. The Jetson has several IO ports making it easy to communicate with sensors or inertial measurement units (IMUs), it even has a sata connection for a hard-drive. The CPU speed is comparable with the U3, but when GPU vision algorithms are used this platforms really shines. A more powerful family of hardware platforms are the Mini-ITX platforms. Mini-ITX platforms all have the same dimensions ($17 \times 17cm$) but can be equipped with different processors and IO. They are basically small computers with the same IO as a normal desktop computer. The mini-ITX platforms can be classified into two categories: the Atom platforms that can be compared with netbooks and the I7-3000 platforms that can be compared with desktops. The Atom Mini-ITX platform has a 1.86GHz Fanless Dual Core processor like in many netbooks computers. Its speed is comparable with the U3 and therefore less interesting due to its larger size, power consumption and weight. Unlike the previous, the Intel i7-3770 platform has a quad core processor and is much faster. This platform is one of the fastest platforms we have tested in this paper. It is five times faster than the XU3 and even faster than our reference system that we used (normal desktop computer). Together with a power supply that can be connected to a LiPo battery and a SSD hard drive, this platform can handle complex image processing algorithms on-board a UAV. The disadvantage of this platform is its power consumption and weight. The next family of platforms are the Brix and Nuc barebone mini-computers. These computers are designed to be mounted on the back of a monitor and have a size of $11 \times 11cm$. These platforms consume less power than the Mini-ITX I7 platform but are twice as slow, which is still very fast for such a small computer. The Brix has an Intel I7-4500 quad-core processor and is comparable in speed with the Nuc that has an Intel I5-4250U processor. When stripping down the casing of these two platforms, the Brix only weighs 172g (SSD included) compared to the Nuc that still weigh 550g, giving the Brix the most interesting specs to mount on a UAV for complex image processing algorithms. Section 5.1 gives an overview of the tests we have performed on these platforms.

4 Approach

The goal of our framework is to find the best hardware platform to run a user's new vision algorithm on a UAV. The main criterion we try to optimize is the amount the processing platform reduces the UAV's flight time. Indeed, both because of the hardware platform's own weight and of its electrical power consumption it drains the battery during flight. The best platform is found when a vision algorithm can run on it at a certain *required speed (in fps frames per second)*, while it consumes as little as possible and the weight of the platform is as low as possible in order to extend flight time. The *required speed* can be much lower than the maximum speed that the algorithm can run on a certain

platform, e.g. a face detector that runs at *100 fps* but only *20 fps* is required for a certain application. The power consumption reduces dramatically when reducing the frame rate of the algorithm on the same platform. We propose a generic

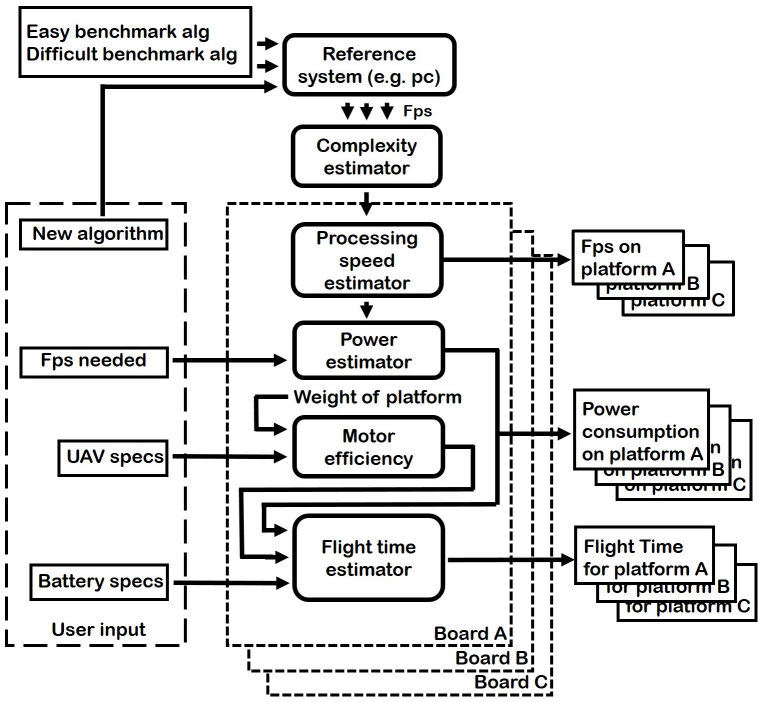

Fig. 2. Overview of our framework

calculation model that estimates the flight time reduction for an arbitrary vision algorithm on a specific embedded processing platform. As seen in Figure 2 this model consists of six blocks. In the first block the user's new algorithm and two benchmark algorithms are executed on a reference system (e.g. the user's desktop computer) and their frame rate is given to the next block where the relative complexity of the new algorithm is estimated. With this, for each hardware platform, its speed is estimated in the next block. Then the power consumption of every platform, while running the new algorithm at a certain required speed, is estimated. In the next block the power consumption of the UAV carrying each hardware platform is calculated. Finally, in the last block the flight-time of the UAV, carrying each hardware platform running the new algorithm at a certain speed, is estimated. In the next subsections these blocks are discussed in detail.

4.1 Complexity and Processing Speed Estimator

To estimate the speed of a new algorithm on every hardware platform we first estimate the complexity of this algorithm. For the sake of simplicity, we assume a linear relation between the processing speed and the complexity of the algorithm. We will validate this linearity assumption in section 5. The speed of the algorithm ($f_{alg} = \frac{1}{T_{alg}}$) on the reference system, e.g. the user's desktop PC, is used as measurement for the complexity (C_{alg}). We empirically measure the relative complexity of the new algorithm with respect to two reference (benchmark) algorithms. The first benchmark algorithm is an easy algorithm that we let correspond with 0% complexity (C_1). For this algorithm we chose the OpenCV implementation of a 3×3 median filter on a color image of 640×480 pixels. The second algorithm is a more difficult algorithm that corresponds to a complexity of 100% (C_2), where OpenCV's HOG person detector is applied to an image of 640×426 pixels. Our *Complexity estimator* uses the execution time of these two benchmark algorithms (T_1 and T_2) and the user's new algorithm (T_{alg}) running on the reference system to calculate the complexity of the new algorithm (see Figure 3). The complexity is then calculated as:

$$C_{alg} = \frac{T_{alg} - T_1}{T_2 - T_1} C_2 + C_1 \tag{1}$$

We assume a linear relation between the computational complexity and the speed of these vision algorithms because they all do mainly the same operations, like applying filters on an image and extracting features. Vision algorithms are always data intensive but most of the time not computationally intensive. Note that code optimizations for specific architectures evidently affect the results. Details like memory usage are not taken into account in this simple model, because the memory on the advanced hardware platforms is easy expandable. Moreover, in our model we only assume CPU-based processing platforms, no other architectures such as GPU or FPGA for which a code translation step would be necessary. In section 5 the validity of this linear relation is verified.

Now that the complexity of the new algorithm (C_{alg}) is known, the speed of the algorithm can be estimated on every platform by following Figure 3 in the other direction, as demonstrated in Figure 4 for two fictitious platforms. The simple and difficult algorithm is run on every platform what results in a T_1 and T_2 for each platform. Because C_{alg} is known from the previous step, T_{alg} can now be calculated for each platform:

$$T_{alg} = \frac{C_{alg} - C_1}{C_2} (T_2 - T_1) + T_1 \tag{2}$$

At this point the speed ($f_{alg} = \frac{1}{T_{alg}}$) of a new algorithm can be estimated for each hardware platform, hence in the next step we can estimate the power consumption of the new algorithm on each platform.

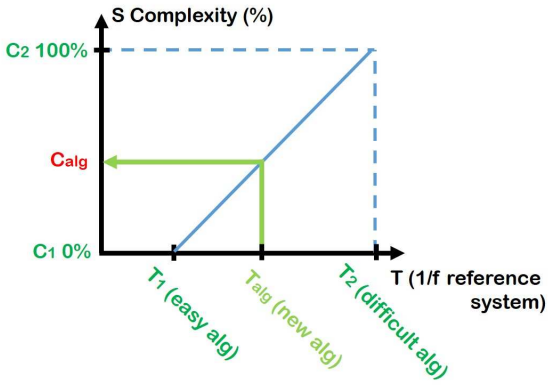

Fig. 3. Linear complexity model. Complexity C_{alg} (red) is estimated with T_1, T_2 and T_{alg} as input (green)

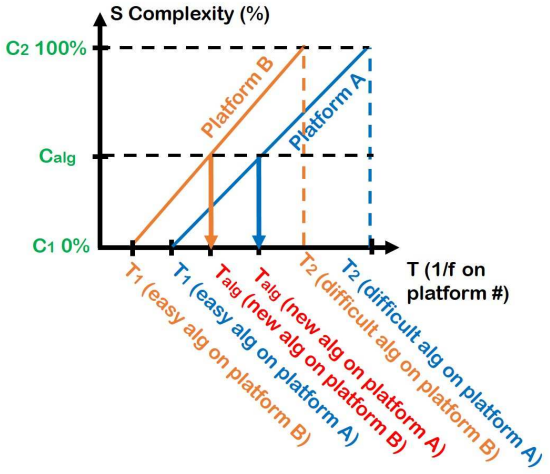

Fig. 4. Calculating T_{alg} (red) for each processing platform (blue and orange) with known C_{alg}, $T1$ and $T2$

4.2 Power Estimator

In UAV applications flight time is of utmost importance. Therefore our framework estimates the power consumption of each hardware platform running the new algorithm at the required speed. We performed experiments to determine the relation between processing speed and power consumption, indicating that a linear model is again a good approximation (see Section 5). When the maximum speed of the algorithm is not required, the power consumption can be lower than when the algorithm is running at full speed. By taking the *required fps* as an input of the *Power Estimation Block* we can estimate the power consumption more precisely for each platform.

To calculate the power consumption P_{alg} of a certain algorithm, the power consumption of each platform is measured when in idle state P_{idle} (doing nothing) and when running all cores at full speed P_{max} (algorithm running at full speed). Together with the required speed (in frames per second) f_{req} and the maximum speed of the algorithm f_{max} the power consumption of the platform can be linearly interpolated as follows:

$$P_{alg} = \frac{P_{max} - P_{idle}}{f_{max}} f_{req} + P_{idle} \tag{3}$$

In this step we also have to eliminate hardware platforms which do not reach the *required fps* (when $\frac{1}{T_{alg}} < f_{req}$). At this point the power consumption of every remaining platform, running the user's new algorithm at a certain speed, is known. In the next step the power consumption of the UAV itself, carrying the platform as payload, is calculated.

4.3 Motor Efficiency

In [4] a model has been developed that enables the user to estimate the power consumption of a multicopter at hover. The performance estimates are based on momentum disk theory and blade element theory of helicopters combined with empirically determined correction factors for multicopters [3]. The model requires the user to input several parameters such as weight, number of propellers n_{props} and propeller radius R. The model uses some empirical parameters such as the Figure of Merit FM (basically the propeller efficiency), the motor efficiency η_{motor} (including the electronic speed controller efficiency) and an installed-to-hover power ratio $\frac{P_{installed}}{P_{hover}}$ of 2 (based on industry standards). The empirical parameters were determined with actual tests on several motors and propellers which are middle grade RC components. The user can (slightly) change these as their multicopter might have higher or lower grade components. We will use this model to estimate the power consumption of the UAV carrying the hardware platform.During hover and slow forward flight it can be assumed that thrust T_{hov}(approximately) equals the total weight force W_{tot} in Newton ($W_{tot} = m_{tot}g = (m_{UAV} + m_{platform})g$) and the hover power per propeller can be calculated through the disk loading DL, induced velocity v_i and air density

ρ:

$$DL = \frac{W_{tot}}{\pi R^2 n_{props}} \tag{4}$$

$$P_{hov_{theo}} = T_{hov} v_{i_{hov}} = W_{tot} v_{i_{hov}} = W_{tot} \sqrt{\frac{DL}{2\rho}} \tag{5}$$

$$P_{hov_{real}} = \frac{P_{hov_{theo}}}{FM\eta_{motor}} \tag{6}$$

Calculating the power consumption of the multicopter based on hover conditions is a rather safe method as during slow forward flight the required power actually decreases by 10% and most multicopter operations take place in this regime [5].

Together with the hardware power consumption P_{alg}, the total electrical power consumption P_{tot} can be calculated as:

$$P_{tot} = \frac{W_{tot} \sqrt{\frac{DL}{2\rho}}}{FM\eta_{motor}} + P_{alg} \tag{7}$$

At this stage the total power consumption of the UAV, carrying the hardware platform that is running a certain algorithm, is known. In the next subsection the flight time is estimated.

4.4 Flight Time Estimator

The flight time for every platform can be estimated since the power consumption of every platform running an algorithm at a certain speed together with the power consumption of the UAV itself carrying each of the platforms is known now. These two values together with the capacity of the batteries are the inputs of this block. Nowadays most UAVs are using lithium polymer batteries because of their good capacity vs weight ratio. Nevertheless the capacity mentioned on the batteries applies only as long as the remaining battery voltage is above a certain value. Therefore most of the time 75% of the battery's capacity is taken as a more fair value to calculate the flight time. Flight time is subsequently calculated as follows:

$$T_{flight}(h) = \frac{0.75 V_{bat} C_{bat}}{P_{tot}} \tag{8}$$

where C_{bat} is the capacity mentioned on the battery in Ah, V_{bat} is the voltage of the battery and P_{tot} is the total power consumption of the UAV at hover (eq. 7).

At this point the main question *"Which hardware platform is best suited to perform a particular image processing task on a UAV?"* can be answered, which we will demonstrate in the next section for our two example algorithms.

5 Experiments and Results

We performed extensive experiments to validate our framework using a wide variety of platforms and multiple algorithms. In the first subsection we performed multiple speed tests of two algorithms to compare the different hardware platforms. In the next subsection we proof that the assumption of a linear complexity and power model holds. Finally we present validation experiments on two computer vision-based real-life cases: face detection and tracking on a UAV for people following and visual navigation to find the corridor in an orchard for fruit counting/inspection.

5.1 Speed tests of two algorithms on each hardware platform

In our first test the processing speed of the OpenCV implementation of a HOG person detector and a Viola and Jones face detector is measured on all platforms. Thereby speed can be compared for every hardware platform. The result can be seen in Figure 5. In Figure 6 we display the ratio of the measured speed of these two algorithms and the power consumption of every platform while running the two algorithms. Figure 7 displays the ratio of the speed and the volume of the hardware platforms and in Figure 8 the ratio of the processing speed and the weight of the platforms is shown.

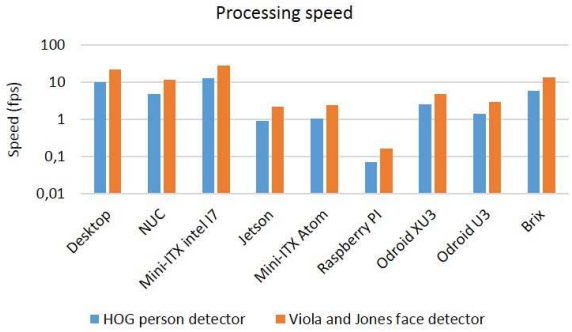

Fig. 5. Speed (logarithmic) of HOG person detector (blue) and Viola and Jones Face detector (orange) for every platform.

As seen in Figures 5 - 8, the Mini ITX Intel I7 platform is one of the fastest but also very heavy. The Jetson and Atom platforms score below average compared to the other platforms because the Jetson is a processing platform designed for GPU implementations and the Atom is already an older generation of CPUs. The Nuc and Brix have a similar speed and power consumption, but the Brix is much lighter and smaller. The two Odroid platforms are similar in power consumption, volume and weight but the XU3 is twice as fast as the U3 platform. Overall, the Brix scores best when all test are taken into account.

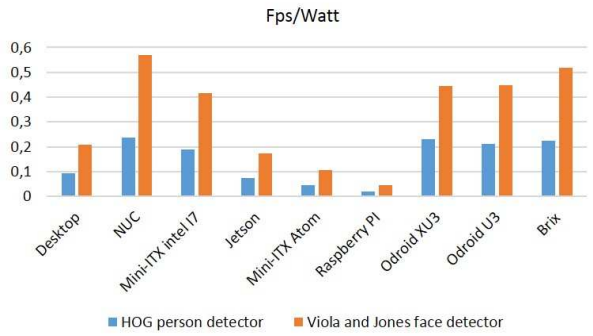

Fig. 6. Processing speed / power consumption ratio for every hardware platform.

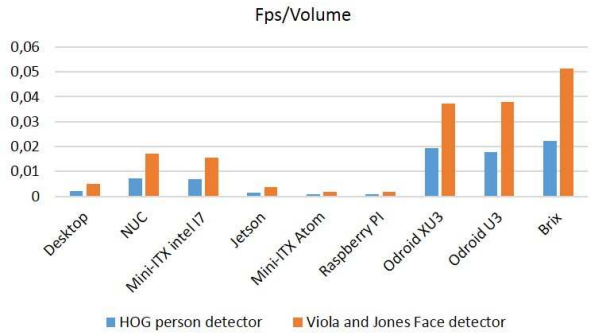

Fig. 7. Processing speed / volume ratio for every hardware platform.

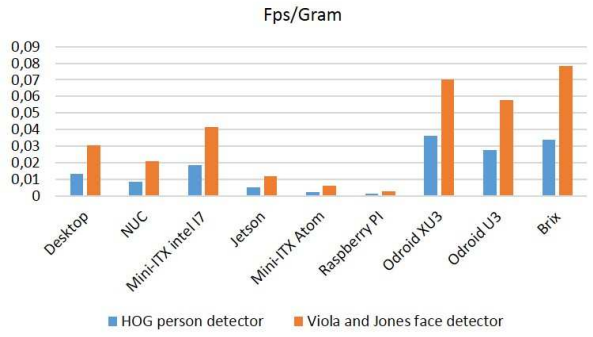

Fig. 8. Processing speed / gram ratio for every hardware platform.

5.2 Validation of Models

In section 4.1 we assumed a linear relation between the complexity of a vision algorithm and the execution speed (the higher the execution time of the algorithm the more complex it is). The linearity is validated by estimating the speed of our two real-case-algorithms, on a desktop computer, for every platform and comparing it with the real speed of these algorithms on every platform. In Figure 9 the percentage deviation between *estimated fps* and *measured fps* is given for the two algorithms. As seen, the error is not greater than 10% which is indicating that the assumption of a linear model for the estimation of the complexity can be taken as valid.

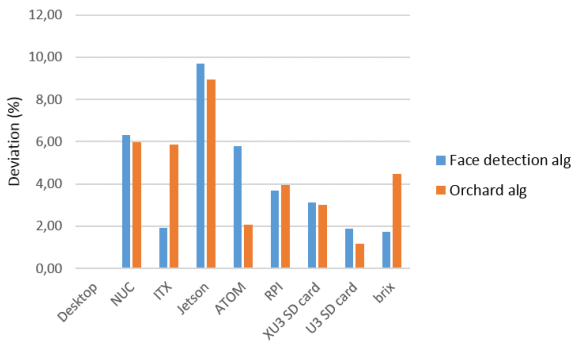

Fig. 9. Deviation between *estimated fps* and *measured fps* of our two real-case-algorithms.

As mentioned in Section 4.2 there is also a linear relation between the power consumption and the processing speed of an algorithm running on a hardware platform. To verify this statement the power consumption of each hardware platform is measured while incrementally increasing the processing speed. As seen in Figure 10, the power consumption increases indeed practically linear with the processing speed for each processing platform.

5.3 Framework Validation on Two Real Cases

For two application cases, we demonstrated the use of the proposed model to find out which hardware platform is best suited for on-board computer vision processing. For both cases a Parrot AR Drone and a XBird 250 is used, of which the forward looking camera is used to capture images for our algorithms. In the first real case, the UAV should follow a single person. The detection of the person is done by using the OpenCV implementation of Viola and Jones face detector [2]. This algorithm should run at least at *4 fps*. In the second case the UAV should navigate through a fruit orchard. Therefore an orchard-path-detection

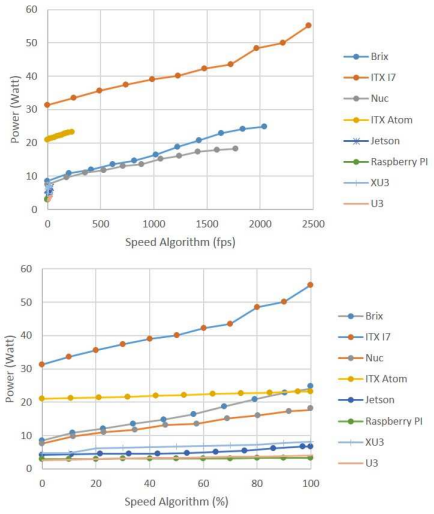

Fig. 10. Power consumption of each platform measured while increasing the speed (top: in fps, bottom: in %) of the easy (Median filter) algorithm.

algorithm is used to find the middle and the vanishing point of the corridor [1]. In this algorithm, filters are applied on the image for preprocessing, followed by a Hough transform to find straight lines (the corridor) and a Kalman filter to predict and track the middle and vanishing point of the corridor. This algorithm should run at least at *10 fps* to fly smoothly through the orchard.

Algorithm	**Speed (fps) Desktop**	**Complexity (%) Desktop**
Benchmark 1 (median)	2040	0
Benchmark 2 (HOG)	9,91	100
Orchard	388	2,08
Face	22,44	43,9

Table 2. Algorithm complexity estimation results.

We ran both algorithms on a normal desktop computer to know their speed with which their complexity is estimated (Table 2). When their complexity is known their speed on every hardware platform is estimated (Equation 2), together with their power consumption (Equation 3) on every platform. At this stage some hardware platforms are discarded because they do not reach the required speed. Thereafter, the total power consumption of the UAV carrying every hardware platform, running the algorithm, is calculated (Equation 7). Finally, flight time is estimated with Equation 8. Results can be seen in Table

Algorithm	Face			Orchard		
Platform	Est. speed (fps)	Est. power consump. (Watt)	Est. flight time (min)	Est. speed (fps)	Est. power consump. (Watt)	Est. flight time (min)
Desktop	22,44			388		
Nuc	10,75	11,48	4,6	199	8,04	4,8
ITX i7	28,88	34,6	3,3	483,9	31,8	3,4
Brix	13,21	13,44	8,3	243,14	9,17	8,9
ITX atom	2,34	24,76		40,28	21,55	5
Jetson	1,98	9,25		16,52	5,71	9,3
RPI	0,16	10,39		1,86	4,61	
XU3	5,06	7,39	11,6	19,5	6,44	11,9
U3	2,95	4,5		14,8	3,55	13,4

Table 3. Results of our framework for the face detection and orchard algorithm. Platforms in red are eliminated because they do not reach the required speed. The platform in green is the best platform to run this algorithm on, on the specific UAV.

	Face		Orchard	
Power consumption	(Watt)	(%)	(Watt)	(%)
Algorithm	7,39	17,2	3,55	9,6
Board weight	7	16,3	5	13,5
UAV weight	26	60,6	26	70
IMU	2,55	5,9	2,55	6,9

Table 4. Power consumption of each part of the system.

Alg.	Est. speed (fps)	Measured speed (fps)	Estimated flight time AR Drone (min)	Measured flight time AR Drone (min)	Estimated flight time XBird (min)	Measured flight time XBird (min)
Face	5,06	4,9	11,6	12,4	7.4	7.12
Orchard	14,8	14,97	13,4	12,7	8,1	7,54

Table 5. Estimated and measured data.

3 and Table 4. Table 4 indicates that the power consumption of the algorithm can't be ignored when using small UAVs.

Secondly, we verified the estimated flight time by attaching the proposed hardware platform on both the AR Drone and XBird while running the specific algorithm. Flight time is measured while hovering, as seen in Table 5 the deviation between estimated and measured data is very small (less than 7%) indicating that our framework indeed finds the best hardware platform for a specific vision algorithm and estimates the speed and flight time very precisely. Note that, when the UAV runs the orchard or face algorithm the flight time reduces with 30.21% and 39.58% as compared to the flight time without payload.

6 Conclusion and Future Work

We developed a framework that finds the best hardware platform for a specific vision processing algorithm that should run at a certain speed on-board a UAV. Furthermore the speed of the algorithm running on each platform is estimated. Thanks to this framework researchers can find a suitable hardware platform without buying them all to test their algorithm on. A second novelty of our framework is that flight time can be estimated for the user's UAV, carrying the proposed platform. We validated the framework with success on two real test cases allowing us to find a suitable hardware platform for our application and to estimate the flight time with our AR Drone and XBird carrying this platform.

Also, we made this model available via an online front end that other researchers can use to find the best platform for their algorithm and even add their own hardware to the framework and expand the database of hardware platforms (*www.eavise.be/VirtualCameraman.html*). In the future we will keep adding new state-of-the-art platforms and extend the framework with GPU platforms.

Acknowledgements

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

References

1. Hulens Dries and Vanderstegen Maarten: UAV autonoom laten vliegen in een boomgaard. Dept of Industr. Eng., College University Lessius (2012)
2. Viola Paul and Jones Michael: Rapid object detection using a boosted cascade of simple features. Computer Vision and Pattern Recognition, CVPR (2001)
3. Prouty Raymond W: Helicopter performance, stability, and control. (1995)
4. Verbeke J., Hulens D., Ramon H., Goedemé T. and De Schutter J.: The Design and Construction of a High Endurance Hexacopter suited for Narrow Corridor. (2014)
5. Theys B., Dimitriadis G., Andrianne T., Hendrick P. and De Schutter J.: Wind Tunnel Testing of a VTOL MAV Propeller in Tilted Operating Mode. ICUAS (2014)
6. Ehsan Shoaib and McDonald-Maier Klaus D: On-board vision processing for small UAVs: Time to rethink strategy. Adaptive Hardware and Systems. NASA/ESA Conference (2009)
7. Suzuki Taro, Amano Yoshiharu and Hashizume Takumi: Development of a SIFT based monocular EKF-SLAM algorithm for a small unmanned aerial vehicle. SICE Annual Conference (SICE), (2011)
8. Meier Lorenz, Tanskanen Petri, Fraundorfer Friedrich and Pollefeys Marc: Pixhawk: A system for autonomous flight using onboard computer vision. Robotics and automation (ICRA), IEEE international conference(2011)
9. Wenzel Karl Engelbert, Masselli Andreas and Zell Andreas: Automatic take off, tracking and landing of a miniature UAV on a moving carrier vehicle. Journal of intelligent & robotic systems, Springer (2011)
10. McGee Tim G, Sengupta Raja and Hedrick Karl: Obstacle detection for small autonomous aircraft using sky segmentation. Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference

11. Kok Jonathan, Gonzalez Luis Felipe and Kelson Neil: FPGA implementation of an evolutionary algorithm for autonomous unmanned aerial vehicle on-board path planning. Evolutionary Computation, IEEE Transactions (2013)
12. Siebert Sebastian and Teizer Joche: Mobile 3D mapping for surveying earthwork projects using an Unmanned Aerial Vehicle (UAV) system. Automation in Construction, Elsevier (2014)
13. Ferrick Allen, Fish Jesse, Venator Edward and Lee Gregory S: UAV obstacle avoidance using image processing techniques. Technologies for Practical Robot Applications (TePRA), (2012) IEEE International Conference
14. Lin Yucong and Saripalli Srikanth: Path planning using 3D Dubins Curve for Unmanned Aerial Vehicles. Unmanned Aircraft Systems (ICUAS), (2014) International Conference
15. Anthony David, Elbaum Sebastian, Lorenz Aaron and Detweiler Carrick: On crop height estimation with UAVs. Intelligent Robots and Systems (IROS 2014), IEEE/RSJ International Conference
16. Nieuwenhuisen Matthias and Behnke Sven: Hierarchical planning with 3d local multiresolution obstacle avoidance for micro aerial vehicles. Proceedings of the Joint Int. Symposium on Robotics (ISR) and the German Conference on Robotics (ROBOTIK) (2014)
17. Sa Inkyu, Hrabar Stefan and Corke Peter: Inspection of pole-like structures using a vision-controlled VTOL UAV and shared autonomy. Intelligent Robots and Systems (IROS 2014), IEEE/RSJ International Conference
18. Shen Shaojie, Mulgaonkar Yash, Michael Nathan and Kumar Vijay: Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor. Robotics: Science and Systems (2013)
19. De Wagter Christophe, Tijmons Sjoerd, Remes Bart DW and de Croon Guido CHE: Autonomous flight of a 20-gram flapping wing MAV with a 4-gram onboard stereo vision system Robotics and Automation (ICRA), (2014) IEEE International Conference
20. Forster Christian, Pizzoli Matia and Scaramuzza Daviden: SVO: Fast Semi-Direct Monocular Visual Odometry. Proc. IEEE Intl. Conf. on Robotics and Automation (2014)