15th IAPR International Conference on Machine Vision Applications (MVA)
Nagoya University, Nagoya, Japan, May 8-12, 2017.

A Raspberry Pi 2-based Stereo Camera Depth Meter

James Cooper, Mihailo Azhar, Trevor Gee, Wannes Van Der Mark, Patrice Delmas
Georgy Gimel'farb
Department of Computer Science, The University of Auckland, Auckland, New Zealand

Abstract

The Raspberry Pi single-board computer is a low cost, light weight system with small power requirements. It is an attractive embedded computer vision solution for many applications, including that of UAVs. Here, we focus on the Raspberry Pi 2 and demonstrate that, with the addition of a multiplexer and two camera modules, it is able to execute a full stereo matching pipeline, making it a suitable depth metering device for UAV usage. Our experimental results demonstrate that the proposed configuration is capable of performing reasonably accurate depth estimation for a system moving at a rate of 1 ms^{-1} when in good lighting conditions.

1 Introduction

The goal of this work is to investigate the potential of using a Raspberry Pi 2 as a depth meter with respect to a slow moving system. This work is the first step of a long-term plan to investigate the usage of a Raspberry Pi or similar devices as an embedded collision avoidance system for unmanned aerial vehicles.

UAV technology is proving to be enormously useful, making significant contributions to environmental science [9], agriculture [20], search and rescue [2], and surveillance [16], to name a few. The addition of embedded systems to UAVs leads to automation of these craft's capabilities, thus reducing the dependency on human controllers, and therefore potentially increasing the scope of uses for these devices.

The Raspberry Pi is especially suited to usage with a UAV due to its light weight and low power requirements. It is also relatively cheap and widely available. The availability of cheap, light-weight cameras make the Raspberry Pi suitable for computer vision systems.

While some previous work has found the Raspberry Pi to be insufficient for UAV-based stereo [6][14], it should be noted that these works were completed prior to the availability of the Raspberry Pi 2. In this work, we assert that the faster processor and light-weight of the Raspberry Pi 2 make it an ideal candidate for our applications. The more-powerful Raspberry Pi 3 has subsequently been released, but at the time of the initial experiments for this paper, the Raspberry Pi 2 was the latest available. As such, it is exclusively considered here, though future work will likely use a Raspberry Pi 3.

One critical limitation of the Raspberry Pi 2 for stereo image processing is that it only has one CSI port for high speed data flow from a camera system. We found however that this could be overcome by the addition of an IVPort multiplexer [8].

2 Literature Review

The various Raspberry Pi (RPi) models have been used repeatedly for vision research, where a low-profile on-board processor has been needed. Dziri *et al.* [3] and Tu [18] *et al.* used cameras with RPis to track humans and honeybees, respectively. Neves and Matos [12], and Valsan and Patil [19] used dual USB cameras and OpenCV to create depth estimation systems. Da Silva [1] *et al.* captured images with a UAV and processed them on an RPi, simulating live capture and processing, but did not put the RPi in the field. Pereira and Pereira [14] tested mounting a RPi onto a UAV in the hopes of performing near-real-time image processing, but concluded that the RPi did not have the processing capabilities required for their application. They used an older version of the Raspberry Pi though.

The first version of the IVPort multiplexer was investigated by Pikkarainen [15]. In their work, they reported that a 400 ms delay between image captures was necessary. They recommended the use of two synchronised RPis for stereo image capture, which is a significant limitation for UAV usage due to weight and mounting considerations.

To the best of our knowledge, only [15] has previously investigated the use of a multiplexer with a Raspberry Pi, and no one has previously investigated the combination for use with a moving set-up.

3 Methodology

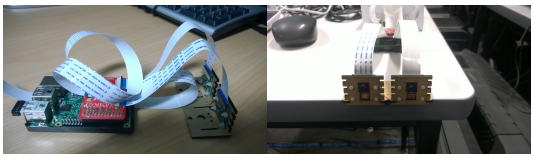

Figure 1: The Raspberry Pi with IVPort mounted, and camera modules attached

The equipment used in this work (Fig. 1) was (i) A Raspberry Pi Model 2, running the Raspbian OS; (ii) An IVMech IVPort multiplexer, revision three; (iii) Two Raspberry Pi camera modules, version one.

Table 1: Raspberry Pi Camera Setup Specifications

Resolution	5 Megapixels
Video Modes	1080p30, 720p60, 640x480p90
Pixel Size	1.4 µm x 1.4 µm
Focal Length	3.60 mm ± 0.01
Base Line	41.00 mm

3.1 The Multiplexer Solution

The Raspberry Pi has a dedicated port for high-speed camera access, the camera serial interface (CSI) port. As there is only one of these installed on the Raspberry Pi 2, it is problematic for stereo image processing. One way to get around this is to use USB webcams [12][19], however this has worse performance that the CSI port [5]. Also, this solution means that the cameras occupy two of the Raspberry Pi's USB ports, which may be desired for other peripherals.

Thus the solution followed in this work is to attach a camera multiplexer, which connects to the Raspberry Pi via its general purpose input-output (GPIO) pins. The multiplexer then operates between the cameras and the Raspberry Pi's CSI port, switching camera feeds based on signals passed via the GPIO pins. The multiplexer we use in our experimental work is the IV-Port multiplexer [8].

3.2 Stereo Matching Pipeline

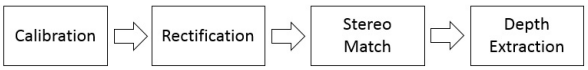

Figure 2: Stereo Matching Pipeline

The goal of the stereo matching pipeline is to determine a depth map. The proposed steps of this pipeline are as follows:

1. **Calibration**: Calibration is performed to determine the intrinsic parameters of each camera in the system, as well as the relative positions and orientations of those cameras relative to each other. In this work we used Tsai calibration [17].

2. **Rectification (and Distortion Removal)**: The calibration parameters allow for the determination of image transforms that remove distortion and remap the images to canonical epipolar geometry (e.g. Fig. 4). The rectification algorithm used here was derived from [4].

3. **Stereo Matching**: Once a stereo pair has been rectified, the disparity map may be derived using a stereo matching algorithm (see Fig. 4 bottom). In this work, we use the Block Matching [10] algorithm from OpenCV [13] to achieve this.

4. **Depth Extraction**: A disparity map is easily converted into a depth map using the relation $Z = (f \times baseline)/disparity$, where Z is the depth and f is the camera's focal length.

The stereo matching pipeline was implemented on the RPi using Python and OpenCV [13]. While some investigation was done towards the usage of high speed, low memory algorithms tailored for embedded systems (such as [7]), it was found that this was ultimately not necessary, as it was found that in most cases the OpenCV implementation was fast enough relative to the acquisition rates of the system.

4 Experiments and Results

Three different experiments were performed on the system to assess its performance. Depth map acquisition is implemented using a typical calibration stereo pipeline [11]. Calibration is performed using Tsai calibration [17]. In the first experiment, the goal was to assess the raw capturing rates associated with a single camera attached to the Raspberry Pi rig. In the second experiment, the goal was to establish the expected rate of acquisition when capturing stereo pairs using the proposed multiplexer solution. The final experiment verified the full pipeline with respect to depth acquisition and motion of the rig.

4.1 Single Camera Capture Rate Assessment

The rationale behind initially assessing the performance of only a single camera was to establish an expected upper-bound on the reliable performance of the system. This upper-bound was used to assess the plausibility of subsequent results.

Experiments were performed by executing continuous image capture sessions for 5 second intervals, saving the captured images to disk. A frame rate for each experiment was determined by counting the number of images captured during the session and dividing the amount by the 5 second interval.

The main result of this experiment was that the fastest frame rate was achieved for a resolution of 640×480 pixels, which was 63.2 frames per second (fps). The main surprise that came from this set of experiments, was the discovery of cases where higher resolution images achieved better frame rates than lower resolution images (1025×768 $\mapsto$ 32.2 fps and 1280×960 $\mapsto$ 38.6 fps for example). Further investigation revealed the cause to be that the Raspberry Pi captures images in a limited set of predefined resolutions, and then maps to new resolutions by scaling the captured images. Frame rates associated with one of the predefined settings tended to be higher than those associated with other resolutions.

4.2 Stereo Camera Capture-rate Assessment

The goal of this set of experiments was to determine the fastest strategy for capturing frames on the proposed system. Experiments were conducted by executing continuous capture sessions for 5 second intervals. Here, we restricted the resolution to 640×480 pixels. The strategies tested were:

1. **Sequence Capture with Toggle Thread (SCTT)**: This strategy makes use of a capture thread and a toggle thread. The capture thread continuously captures from the active camera and saves to disk, while the toggle thread switches the active camera after a predefined interval. Switching between the capture and toggle mode allowed to test various toggle interval. It was found that toggle rates below 100 ms resulted in corrupt images (see Fig. 3).

2. **Synchronized Capture and Switch (SCS)**: In this strategy an image is captured from the first camera and saved to disk and then from the second camera and saved to disk and so forth. The

switching call is made immediately after an image is captured, to give the longest time possible between a switch and the next image capture.

3. **Capture to Memory Stream and Switch (CMSS)**: This strategy is similar to the SCS strategy, however images are captured and saved to memory instead of disk. A separate thread slowly unpacks the memory and writes the images to disk.

(a) 10 µs (b) 10 ms

Figure 3: Corrupted images captured with too high toggle rates using the SCTT strategy, with different periods between switch calls.

It was found that the SCTT, SCS and CMSS strategies respectively required 200 ms, 271 ms and 193 ms to perform a capture of two images for a 'stereo pair', resulting in respective framerates of 5.0, 3.7 and 5.2 frames per second.

The minimum switching time necessary to ensure that the vast majority of images captured are uncorrupted was found to be approximately 100 ms. This places an upper bound of around five stereo images per second (10 per second, five for the left, five for the right). The synchronized nature of the switching employed in the SCS and CMSS strategies ensured that the switching time occurred at the best point possible, when there would be the longest time until a new image was captured, significantly reducing the incidence of corrupted image captures.

Considering that the CMSS strategy yielded a framerate at the rough maximum imposed by the multiplexer, and avoided issues of images becoming severely corrupted, such as with Figure 3, the CMSS strategy was employed in the later experiments.

4.3 Full pipeline evaluation

In order to evaluate our full system, we conducted a set of experiments to assess the behavior of our system with respect to depth acquisition with the rig in motion. Time and space constraints prevented us from performing further experiments for this paper.

Our experimental setup included a backboard towards which the Raspberry Pi rig moved at an approximate rate of 1 ms^{-1}. This speed was assessed via a pedometer and a human operator walking slowly. Incremental markers were placed so that the distance between the camera and the backboard were known at 10 millimeter intervals. Depth measurements for the backboard were acquired manually from the depth map. A final measurement was acquired by performing a plane fitting on the depth values and acquiring the depth value at the central position of this plane.

The experiment was repeated a second time, however this time the cameras were kept static at each interval. For comparison, a static GoPro stereo rig was also used. The purpose of this experiment was to assess the effect of motion on the system. The experimental results are shown in Table 2.

Table 2: Actual Distance vs System Distance (mm) acquired from RPi rig moving at 1 meter per second and a static system, in good lighting conditions.

Actual	Moving	Static
500	631	504
1500	1621	1506
2000	2109	2027
2500	2635	2510
3000	3100	3012
3500	3635	3514
4000	4055	4055
4500	4583	4583

For comparison, results were also gathered with the cameras looking outdoors. In order to be able to accurately measure distances for the ground truth, this was performed indoors, but with the backboard against a window looking onto a busy street. Lighting conditions were unfavourable, with only moderate lighting inside, but significant glare behind the backboard (see Figure 4). Results were collected with both the RPi and a stereo pair of GoPro cameras for comparative results with a commonly used set-up, and are presented in Table 3. Precise round actual distances were difficult to achieve in the experimental location, so the actual depths measured with a laser depth-meter are shown. Due to the poor lighting conditions, the backboard was too dark in the image for features to be detected at 4500 mm in the static scenario, so results are not included.

Table 3: Groundtruth distance vs system distance (in mm). Acquired from our RPi rig moving at $1m.s^{-1}$ and from a static system, in suboptimal lighting conditions.

Static			Moving		
GT	RPi	GoPro	GT	RPi	GoPro
1009	892	929	1018	1038	778
1502	1532	1367	1503	1559	1711
2002	2077	1865	2014	2077	2164
2494	2532	2160	2492	1933	2675
3008	2971	2880	2988	2373	2901
3494	3998	3248	3496	3693	3043
4025	4090	3881	3995	3127	3647
-	-	-	4498	4090	4128

5 Conclusions

In this work, we describe the construction of a Raspberry Pi 2 system that is capable of acting as a depth measurement system for slow forward moving systems. We investigated the notion of attaching two cameras to a single CSI port with the addition of an IVPort multiplexer with respect to a Raspberry Pi 2 system in motion. It was found that a simple OpenCV based stereo

Figure 4: The RPi processing pipeline. From top to bottom: Indoor and outdoor rectified stereo pairs; corresponding disparity maps

matching pipeline was adequate to acquire the respective depth maps when its performance was assessed relative to the memory requirements of the Raspberry Pi and its speed with respect to the acquisition rates of the images from the camera system. A resolution of 640×480 pixels was found to yeld the fastest image acquisition rate at an effective 63.2 frames per second from a single camera. It was also found that the fastest rate that we could acquire stereo pairs was 5.2 *stereo* frames per second. Any attempt to use the multiplexer to switch between cameras at a faster rate resulted in interlacing and colour artifacts within images. It was found that motion mostly affected our system performance at distances smaller than 4 meters from the target object. Greater distances however, had roughly equivalent depth measurements. It is suspected that the reason for this is that the error due to motion had less effect at greater distances due to the lower depth resolution per disparity level at this distance.

When in less-than-ideal lighting conditions, results varied. At close distances, the RPi system produced more accurate results than a stereo pair of GoPro cameras, but became less accurate at greater distances, especially when in motion. The results suggest that the RPi with a multiplexer may be an acceptable substitute for a GoPro pair at low speeds, while being much more practical for mounting on a lightweight vehicle.

In future work, we aim to attach our system to an actual UAV and perform field testing of the system.

References

[1] J. F. da Silva, A. V. Brito, J. A. da Lima, and H. N. da Moura. An embedded system for aerial image processing from unmanned aerial vehicles. In *2015 Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 154–157, 2015.

[2] P. Doherty and P. Rudol. A uav search and rescue scenario with human body detection and geolocalization. In *Australasian Joint Conference on Artificial Intelligence*, pages 1–13. Springer, 2007.

[3] A. Dziri, M. Duranton, and R. Chapuis. Real-time multiple objects tracking on Raspberry-Pi-based smart embedded camera. *Journal of Electronic Imaging*, 25(4): 041005, 2016.

[4] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. *Machine Vision and Applications*, 12(1):16–22, 2000.

[5] B. Horan. *Hardware Overview*, pages 1–16. Practical Raspberry Pi. Apress, Berkeley, CA, 2013.

[6] D. Hulens, T. Goedem, and J. Verbeke. How to choose the best embedded processing platform for on- board uav image processing? In *Proceedings 10th international conference on computer vision theory and applications*, pages 1–10, 11-14 March 2015.

[7] M. Humenberger, C. Zinner, M. Weber, W. Kubinger, and M. Vincze. A fast stereo matching algorithm suitable for embedded real-time systems. *Computer Vision and Image Understanding*, 114(11):1180–1202, 2010.

[8] IVMech. Raspberry pi camera module multiplexer. `https://github.com/ivmech/ivport`, 2016. [Online; accessed 25-September-2016].

[9] A. Jaakkola, J. Hyyppä, A. Kukko, X. Yu, H. Kaartinen, M. Lehtomäki, and Y. Lin. A low-cost multi-sensoral mobile mapping system and its feasibility for tree measurements. *ISPRS journal of Photogrammetry and Remote Sensing*, 65(6):514–522, 2010.

[10] K. Konolige. Small vision systems: Hardware and implementation. In *Robotics Research*, pages 203–212. Springer, 1998.

[11] W. Li, T. Gee, H. Friedrich, and P. Delmas. A practical comparison between zhang's and tsai's calibration approaches. In *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand*, pages 166–171. ACM, 2014.

[12] R. Neves and A. C. Matos. Raspberry PI based stereo vision for small size ASVs. In *2013 OCEANS - San Diego*, pages 1–6, 2013.

[13] L. OpenCV. Computer vision with the opencv library. *GaryBradski & Adrian Kaebler-OReilly*, 2008.

[14] F. C. Pereira and C. E. Pereira. Embedded image processing systems for automatic recognition of cracks using uavs. *IFAC-PapersOnLine*, 48(10):16, 2015.

[15] M.-P. Pikkarainen. Raspberry pi-pohjainen rgb-& ir-kuvantamis-ja mittausjrjestelm, 2015.

[16] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek. Autonomous uav surveillance in complex urban environments. In *Web Intelligence and Intelligent Agent Technologies, 2009. WI-IAT'09. IEEE International Joint Conferences on*, volume 2, pages 82–85, 2009.

[17] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, 1987.

[18] G. J. Tu, M. K. Hansen, P. Kryger, and P. Ahrendt. Automatic behaviour analysis system for honeybees using computer vision. *Computers and Electronics in Agriculture*, 122:10–18, 3 2016.

[19] V. N. Valsan and C. Y. Patil. A system on chip based stereo vision approach for disparity measurement. In *Industrial Instrumentation and Control (ICIC), 2015 International Conference on*, pages 1284–1287, 2015.

[20] C. Zhang and J. M. Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6):693–712, 2012.