Horizontal-based Attitude Estimation for Real-time UAV control

Maiko Arakawa, Yuichi Okuyama, Shunsuke Mie, Ben Abdallah Abderazek
University of Aizu
m5221116@u-aizu.ac.jp, okuyama@u-aizu.ac.jp, okuyama@u-aizu.ac.j, benab@u-aizu.ac.jp

Abstract

An attitude estimation is an important process for auto-piloting unmanned aerial vehicles (UAVs) control. We implemented the attitude estimation system from the visible horizon in the images. We use images from the camera to compensate IMU (Inertial measurement units). We used FPGA since the image processing needs hard computation and UAV's control needs hard real-time processing. The horizon is detected through the use of morphological smoothing, Sobel filter and Hough transform. Run-length encoder is used before Hough transform to reduce memory bandwidth. The system is implemented by high-level synthesis and on Zynq UltraScale+MPSoC ZCU102. The result shows the system is faster than the implementation using desktop computer four times.

1. Introduction

Multiple companies develop auto-piloting systems for UAVs in the field of agriculture, transportation, surveying, and rescue. In the general approach, UAVs estimates their attitude for self-control with GPS and inertial measurement units (IMU) such as an accelerometer, gyro sensor, and magnetism sensor. However, IMU accumulate their errors and cannot correct them by information of IMU. Magnetism sensor is influenced by magnetism and GPS depends on locations [1]. A vision-sensor solve these problems. The vision-sensor can compensate for other sensors weakness because images from the sensor are not affected by magnetism and location, and images can correct errors. Image processing needs high calculation costs, and attitude estimation must be satisfied hard real-time processing. For both high computation costs and hard real-time constraint, we used FPGA that has parallelism and enables efficiency numeric operation [2].

If the horizon is visible, it is the strongest world reference. A sky and ground segmentation are a critical step for detection of the horizon. The detection algorithms use color and texture in the image as critical features [1] [3-5]. D.Dusha el.al [6] proposed new front-end image processing algorithm for the horizon detection and estimated an attitude. They showed high accuracy of 90.7, however time is not mentioned about the calculation time. H.Guo el. al [7] also used the horizon with the Kalman filter and LSD algorithm [8]. The results showed higher tracking accuracy and speed than [6], but not real-time.

In this paper, we describe the horizontal-based attitude estimation system combined front-end image processing and Hough transform implemented on FPGA. We evaluate the system with desktop by the speed and show how much faster. Section 2 explain the horizontal-based attitude estimation method. In section 3, hardware implementation of the system is provided. Section 4 show evaluation of the system, and a conclusion is given in section 5.

2. Attitude Estimation from Images

In this section, the algorithm of the horizontal-based attitude estimation is explained. The horizon is decided using pre-filtering, run-length encoding and Hough transform [9]. Pre-filtering step remove a noise and get the only pixels related the horizon. Run-length encoding step compress the image, so not related pixels are skipped. Hough transform decides the horizon. The image processing front-end which is proposed in [6] is used in pre-filtering step. The method has robustness for changing light, scenery, and glare. Hough transform with run-length encoder is described in [10]. The architecture can reduce memory access and memory bandwidth.

2.1. Pre-filtering

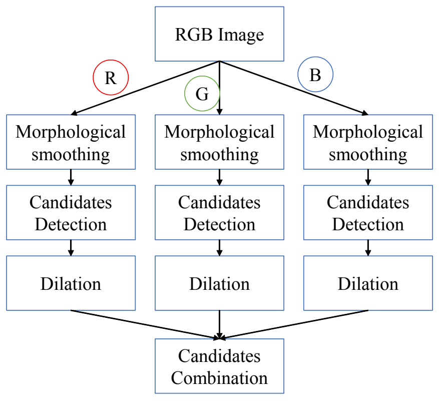

Figure 1. Pre-filtering overview

Figure 1 shows the flow of pre-filtering step. This step consists Morphological smoothing [11], edge detection, dilation, and edge combination. Three filters are applied to each RGB channels of the image, and finally, all edges are combined. Morphological smoothing reduce noise in the image and has edge-preserving properties and not strong response will be curved with circular structure element. Sobel filter and thresholding are performed for the horizon candidate detection. The output is binary image and one represents a candidate(feature), zero represents nonfeature. All candidates of RGB channels are combined with pixel-wise AND operation leaves only the horizon since the horizon is correlated between each RGB channels.

2.2. Run-Length Encoding

A run-length encoding can encode an input binary image into a zero-symbol stream. Most of the binary image is nonfeature, then skipping them leads effective calculation.

An image is divided into blocks, and zero-symbols are produced for all blocks. A symbol is represented by *{rb, code, zl}* triplet. The *rb* is whether the current block is the first block in a line or not, the *code* is the pixel values in the block, and the *zl* counts zero blocks number after the current block. A zero block means all the pixels in the block are nonfeature pixels. In the pre-filtering step, only a strong line is left as feature pixels, and feature pixels are crowded. The run-length encoding can reduce memory bandwidth and computation time using the symbol and skipping the zero blocks.

2.3. Horizon Detection

Hough Transform is an algorithm for detecting straight lines. Each line is represented in the ρ-α space, where the ρ is a perpendicular distance of the origin to the line and the α is the angle between the ρ and x-axis. A line can be drawn for each feature point. The ρ value is calculated using (1) for all feature pixels and for all K, interval of angles. Once the ρ value is calculated, the specific (ρ, α) is voted. Voting means counting the number of the each produced (ρ, α) value. Completing voting for all feature points, the horizon is represented as (ρ, α) that has largest vote number. From the ρ and α, the gradient of the horizon is lead from (2).

$$\rho = x\cos\alpha + y\sin\alpha \qquad (1)$$

$$y = \frac{x}{\tan\alpha} + \frac{\rho}{\sin\alpha} \qquad (2)$$

2.4. Attitude Estimation

The gradient and the position of the horizon is related to the attitude. The horizon is on the image, on the camera-coordinate. The attitude of UAV is on the world-coordinate. The coordinate transform from camera into world is needed.

First, we define world-coordinate and camera coordinate in Figure 2. The z-axis of the world frame is defined as the line from the origin to the center of the horizon. The y-axis is the direction of aircraft's gravity. The x-axis works according to the right-hand coordinate system. The z-axis of the camera frame is the optical axis of the camera. The x-axis is parallel to the top edge of the image plane. The y-axis is according to the right-hand coordinate system.

The UAV's attitude is the rotation from the world coordinate frame to the camera coordinate frame. The rotation about the x-axis is a pitch angle θ, the z-axis is a roll angle ϕ and the y-axis is a yaw angle ψ. Using only the horizon, the yaw angle is assumed zero.

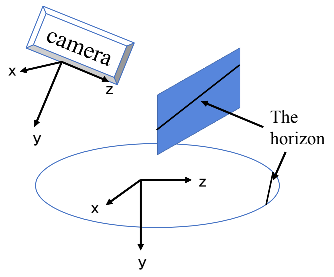

Figure 2. Coordinate definition

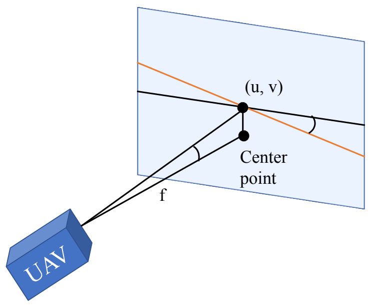

Figure 3. Attitude of UAV's

The roll angle is dependent on only the gradient of the visible horizon line obviously in Figure 3. (3) is carried out.

$$\phi = \operatorname{atan}(-a) \qquad (3)$$

(u, v) is a point on the horizon, f is a focal length, and h is the distance from the center point of the image-plane to the point *(u, v)*. In Figure 3, we can represent as:

$$\tan\theta = \frac{h}{f} \qquad (4)$$

$$\theta = \arctan\left(\pm\frac{u\sin\phi + v\cos\phi}{f}\right) \qquad (5)$$

From (5), the pitch angle is dependent on the roll angle and the position on the image plane that the horizon falls.

3. Hardware Implementation

We implemented horizontal-based attitude estimation system using high-level synthesis. The system the parameter of the horizon from the image. From the camera, RGB image is captured and stored them in the memory. Proposed circuits access to the memory, read image data, apply to pixels pre-filtering and horizon detection. ARM processor calculates own attitude from the ρ and α of the horizon. The system is shown in Figure 4.

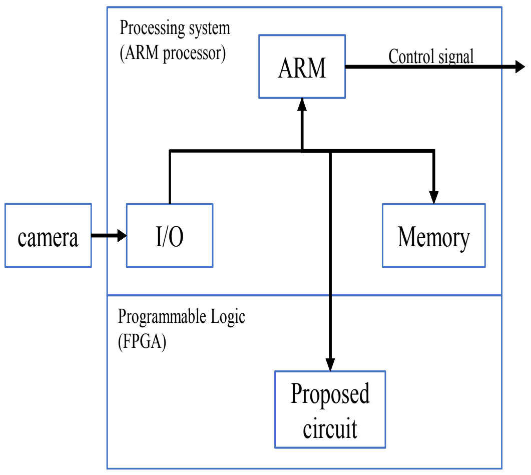

Figure 4. System overview

Overview of the proposed circuit is shown in Figure 5. Pre-filtering function read image data from memory and sends pixel after filtering to Run-length encoding function. Run-Length Encoding function sends the zero-symbol every four lines. Hough transform function gets the zero-symbol and updates the vote number in the memory. Subsequently, it sends *ρ* and *α* to the ARM processor.

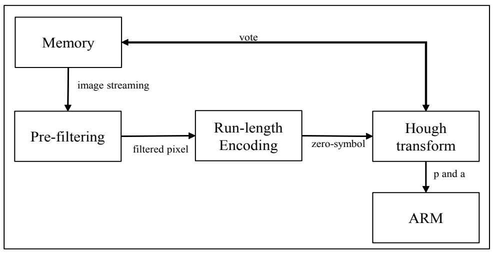

Figure 5. Proposed circuit

3.1 Pre-filtering

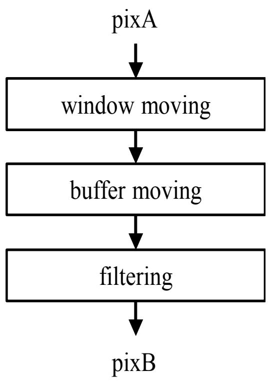

Figure 6. Flow of pre-filtering

Pre-filtering function separates pixels related to the horizon and pixels not related. The line buffer and the window buffer are used to reduce hardware resource requirement. The line buffer stores pixels for one line. The window buffer stores pixels for filter applying. The kernel size of filter is five by five, thus five lines ($lineBuf_0$ to $lineBuf_4$), and five by five window are used. Figure 6 shows flow of pre-filtering function. First, in widow moving, all pixels in the widow buffer move to left column and pixels from the line buffer and a new pixel (pixA) are stored in the end of the window line. Next, in the buffer moving, the pixel move from $lineBuf_n$ to $lineBuf_{n-1}$ and a pix A is stored in $lineBuf_4$. Finally, filtered pixB is send to next filter. Pre-filtering have the erosion filter, the dilation filter and Sobel filter. For all the filter, the difference is only kernel and the architectures are same.

3.2 Run-Length Encoding

Run-length encoding function compresses the feature image into zero-symbol. The pre-filtered pixel is stored to buffer for four lines and divided into two by four blocks. If the block is not a zero-block, the zero-symbol is produced. Counting the zero-block, if sum of zero-block is over than maximum number, the zero-symbol is created. Pre-filtering function reduce unnecessary pixels, thus skipping zero-blocks reduces memory bandwidth. Figure 7 shows a flow of run-length encoding.

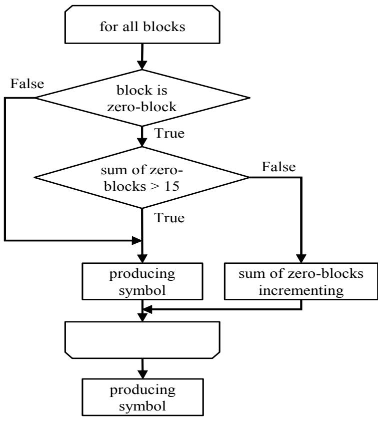

Figure 7. Flow of run-length encoding

3.3 Hough transform

Hough transform function calculate ρ value and vote for all angles and zero-symbols. The largest accumulated votes will correspond to the horizon. In the implementation, the votes for a specific (ρ, α) value can be stored in a memory addressed by the specific (ρ, α) value. The step-size is one.

A flow of Hough transform function is shown in Figure 8. We have five functions, pixel arrangement, reset variable, block incrementing, pixel incrementing, vote consolidation and vote. First, pixel arrangement function decides pixel order. There are two way of pixel arrangement by the angle. Reset variable function reset a variable value zero. Block incrementing function calculates the smallest ρ_α value in a block. The fractional part of the ρ_α is used in the pixel incrementing function. Pixel incrementing function calculates other ρ_α value in a block. After that, the vote consolidation function decides vote number for each specific (ρ_α, α) in a block and the vote function votes. Finally, the find max function finds the largest voted (ρ_α, α) value as the horizon.

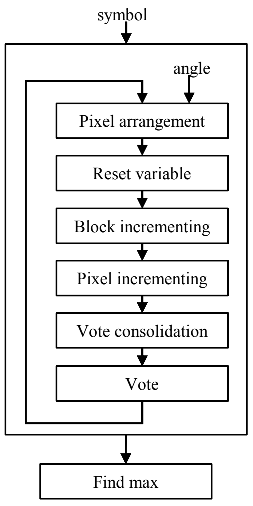

Figure 8. Hough transform

3.3.1 Pixel arrangement

All pixels in a block are labeled from p_0 to p_7 and, p_0 has smallest ρ value in the block. Figure 9 shows an arrangement of pixels, (A) is for 0°≤angle<90 and (B) is for 90°≤angle<180.

p_0	p_1
p_2	p_3
p_4	p_5
p_6	p_7

(A)

p_1	p_0
p_3	p_2
p_5	p_4
p_7	p_6

(B)

Figure 9. Pixel arrangement:(A)0°≤angle<90° (B)90°≤angle<180°

3.3.1 Block incrementing function

Algorithm 1. Find the smallest $\rho_\alpha(p_0)$

```
function Find the smallest ρα(p0) (rb, zl, angle, sinval, cosval)
{
  if the angle is less than 90 degree
    initialize row[angle] = 0
  else
    initialize row[angle] = cosval

  if this is not a first block
    // move to next block address
    col[angle] += step[angle]
  else
    // move to a first block in a next line
    col[angle] = row[angle]
  // calculate next row block address
  row[angle] += N*sinval
  // skip the zero blocks
  step[angle] = (zl+1)*M*cosval
  return col[angle]
}
```

The block incrementing function calculates the smallest ρ_α value in a block. An algorithm is shown in Algorithm 2. The block size is 2 by 4, and so M and N are 2 and 4. The variable row calculates $\rho_\alpha(p_0)$ values for the first blocks in the y-direction and the variable col calculates $\rho_\alpha(p_0)$ value for the nonzero blocks in the x-direction. First, initialize row if the angle is less than 90 degrees, the angle α otherwise $\cos\alpha$. Next, the current block is checked whether the current block is the first block in the line or not. If the current block isn't the first block, zero blocks will be skipped. If the current block is the first block, col is equal to row. The col is only responsible for calculating the ρ_α value of the first pixel in a block. The ρ_α values of other pixels are calculated by the next pixel incrementing function.

3.3.2 Pixel incrementing function

Algorithm 2. Calculate all ρ_α in the block

```
function Calc all ρ_α in the block ($vo$, $f$, $\cos\alpha$, $\sin\alpha$) {
    $vo_0=0$
    $vo_1=f+\cos\alpha$
    $vo_2=f+\sin\alpha$
    $vo_3=f+\sin\alpha+\cos\alpha$
    $vo_4=f+2\sin\alpha$
    $vo_5=f+2\sin\alpha+\cos\alpha$
    $vo_6=f+3\sin\alpha$
    $vo_7=f+3\sin\alpha+\cos\alpha$
}
```

The pixel incrementing function calculates $\rho_\alpha$values of other pixels in a block using $\rho_\alpha(p_0)$. The calculation method is shown in Algorithm 3. The variable f is a fractional part of $\rho_\alpha(p_0)$ and the variable vo has difference integer value between $\rho_\alpha(p_0)$ and other pixels. The vo_n is relative to nth pixel in the block. The efficient circuit is obtained since the only fractional part is needed.

3.3.3 Vote consolidation function

Algorithm 3. Vote consolidation

```
function Vote consolidation (v, code)
{
    for all $vo_n$
        $v_{vo_n} += code_n$
    endfor
}
```

Vote consolidation function decides vote numbers for each specific (ρ_α, α) in a block. For the block-size of 2 by 4, there are at most five different ρ_α values in the whole block. The range is i to $i+4$ (i is the integer part of $\rho_\alpha(p_0)$). The v_i is the vote number of $(\rho_\alpha(p_0)+i, \alpha)$, and so if a pixel is a feature pixel, vote number is incremented.

3.3.4 Vote function and find max function

The vote function vote v_i to v_{i+4} number. We only need to access once (with the address i to $i+4$), and each time we can accumulate the memory contents in parallel. At the end of Hough transform, we search the largest voted (ρ_α, α). We assume the most voted (ρ_α, α) is the horizon. The (ρ_α, α) is sent to ARM processor, and ARM processor calculates own attitude.

4. Evaluation

We use Vivado HLS 2018.1 for a synthesis. The proposed Zynq system is targeted by the Zynq UltraScale+MPSoC ZCU102 xczu9eg-2ffvb1156. The Zynq system is compared with desktop system implemented by the software with Intel core i5-6400 CPU @2.70GHz 8GB RAM. We assume a typical camera frame rate is 30 [Hz]. We evaluate the hardware resources and the execution time and show this system satisfy hardware resource limitation of the target device. We also compared the execution time of our proposed system and desktop system.

4.1 Performance of each function

Table 1. Resource utilization (%) of functions

	BRAM	DSP	FF	LUT
Pre-filtering	2	0	1	2
RLE	~0	0	~0	34
Hough	~0	2	33	54
Available	1824	2520	548160	274080

Table 2. Calculation cycles of functions

	Latency(cycles)		Interval(cycles)	
	Min	Max	Min	Max
Pre-filtering	8	8	9	9
RLE	1	322	1	322
Hough	3456	3456	3456	3456

Table 1 shows resource utilization of each function, pre-filtering, run-length encoding and Hough transform. Max frequency of pre-filtering, run-length encoding, and Hough transform are 123MHz, 196MHz, and 114MHz.

Table 2 shows the synthesis results of functions. Latency is a number of clock cycles required for the function to compute all output values. Interval is a number of clock cycles before the function can accept new input data. Input image size is assumed 640x480. Pre-filtering function read 24bit pixel value every nine cycles, and output result pixel value after eight cycle. However, pre-filtering function stores pixels into line buffer and waits line buffer becomes full. Actually, latency is four line, 20480 cycles are assumed. Run-length encoding read one bit pixel every cycle. Run-length encoding function stores pixel for four line buffer, and latency is one. Once line buffers are full, zero-symbols are calculated and latency is 322 because of a width of the image and setup time. Hough transform function read one zero-symbol every 3456 cycles. In the function, angle calculation is looped from 0 degree to 180 degree with one degree interval. Cycle time of 1 angle is assumed 19 cycles.

4.2 Performance of overall circuit

Table 3. Number of resource requirement of overall circuit

	BRAM	DSP	FF	LUT
Total	75	60	198921	253067
Utilization (%)	4	2	36	92

Table 4. Latency of overall circuit

Latency(cycles)	
Min	Max
2719485849601	40792330444801

Table 5. Interval of overall circuit

Interval(cycles)	
Min	Max
2719485849601	40792330444801

Table 3 shows resource requirement. Table 4 and Table 5 shows latency and interval of overall circuit. Maximum frequency is 114MHz. Maximum latency is when all the pixels in the image are feature pixels. In actually, pre-filtering extract horizon and latency will be small.

4.3 Comparison of computation time

Table 6. Comparison of computation time[ms]

num of symbol	desktop system	Zynq system
5101	4441	1229

Table 5 show comparison result of computation time between the Zynq system and the desktop system using practical image. The image drawn only horizon can be extracted a few but strong candidates of the horizon, however the image drawn extra object cannot be extracted only the horizon. The number of the candidates decides the number of the symbol, so the computation time depends on number of the symbol. The practical image has only a strong horizon. The desktop system takes 4441[ms] and the Zynq system takes 1229[ms] with 5101 symbols.

5. Discussion

The result shows the Zynq system is four times faster than the desktop system, but not real-time. Pre-filtering and run-length encoding have lower latency than Hough transform. In this evaluation, one Hough transform function calculate, so calculation with some Hough transform functions in parallel will be improve speed. The variability of resource requirement is big. LUT is used thirty times as much as Block RAM. The Zynq system will be more effective performance reconsidering memory management.

6. Conclusion

In this paper, we describe horizontal-based attitude estimation system on FPGA using high-level synthesis. This system uses the image from the camera, finds the horizon in the image, and then estimates attitude from the horizon. The image includes an unrelated information to the horizon. First of all, pre-filtering function reduces unnecessary pixels and extracts only feature pixels. Next, run-length encoding function compresses the feature image into zero-symbol. Last, Hough transform function finds out the horizon. Run-length encoding function reduces the memory bandwidth, and the resources are used efficiently by Hough transform function.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number JP18K11413.

References

[1] E. R. Shabayek, C. Demonceaux, O. Morel, D. Fofi, “Vision Based UAV Attitude Estimation: Progress and Insights,” Journal of Intelligent and Robotic Systems, Springer Verlag, pp. 295-308, 2011

[2] W. S. Fife, J. K. Archibald, “Reconfigurable On-Board Vision Processing for Small Autonomous Vehicles,” Journal on Embedded System, 2007

[3] Y. WE, G. LI, W. LI, X. ZHANG, J. CHE, F. QIAO, “A Monocular Vision-based Attitude Estimation Approach for small Unmanned Aerial chicles and its Experimental Verification,” Navigation and Control Conference, 2016

[4] S. Todorovic, C.Nechyba, P. G. Ifju, “Sky/Ground Modeling for Autonomous MAV Flight,” International Conference on Robotics & Automation, pp14-19, 2003

[5] S. Thurrowgood, D.Soccol, R. J. D. Moore, D. Bland, M. V. Srinivasan, “A vision based System for Attitude estimation of UAVs,” in Intelligent Robots and Systems, 2009

[6] D. Dusha, W. Boles, R. Walker, “Attitude Estimation for a Fixed-Wing Aircraft Using Horizon Detection and Optical Flow,” Digital Image Computing Techniques and Applications, Australian Research Center for Aerospace Automation, 2007

[7] H.Guo, Y. Zhang, J. Zhou, Y. Zhang, “A FAST AND ROBUST VISION-BASED HORIZON TRACKING METHOD,” Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 12th International Computer Conference, 2015

[8] R. G. Von, J. Jakubowicz, J. Morel, G. Randall, "LSD: A fast line segment detector with a false detection control", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 4, pp. 722-732

[9] R. O. Duda and P. E. Hart, “Use of the Hough transformation to detect lines and curves in pitures,” Commun. ACM vol. 15, pp. 11-15, 1972

[10] Z. Chen, A. W. Y. Su, M. Sun, “Resource-efficient FPGA Architecture and Implementation of Hough Transform”, IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 20, pp. 1419-1428, Aug. 2012

[11] R. C. Gonzalez, R. E. Woods, "Digital image processing" in Upper Saddle River, N.J.:Prentice Hall, 2002