Real Time Vision System for Obstacle Detection and Localization on FPGA

Ali Alhamwi, Bertrand Vandeportaele, Jonathan Piat

Real time Vision System for Obstacle Detection and Localization on FPGA

Ali Alhamwi, Bertrand Vandeportaele and Jonathan Piat

LAAS-CNRS, 7 Avenue du colonel Roche, F-31077 Toulouse Cedex 4, France
University of Toulouse; UPS F-31077 Toulouse Cedex 4, France
aalhamwi@laas.fr, bvandepo@laas.fr, jpiat@laas.fr

Abstract. Obstacle detection is a mandatory function for a robot navigating in an indoor environment especially when interaction with humans is done in a cluttered environment. Commonly used vision-based solutions like SLAM (Simultaneous Localization and Mapping) or optical flow tend to be computation intensive and require powerful computation resources to meet low speed real-time constraints. Solutions using LIDAR (Light Detection And Ranging) sensors are more robust but not cost effective. This paper presents a real-time hardware architecture for vision-based obstacle detection and localization based on IPM (Inverse Perspective Mapping) for obstacle detection, and Otsu's method plus Bresenham's algorithm for obstacle segmentation and localization under the hypothesis of a flat ground. The proposed architecture combines cost effectiveness, high frame-rate with low latency, low power consumption and without any prior knowledge of the scene compared to existing implementations.

1 Introduction

Obstacle detection is a fundamental ability of mobile robots to operate in an cluttered indoor environment and it is essential in order to perform basic functions of mobile robots like avoidance and navigation. This critical task is often addressed with high cost sensors (LIDAR, RADAR) or computation intensive algorithms (Optical Flow, SLAM ...) that prevent to limit the cost of a robotic system.

Sonar based methods prove to be unreliable because of the system noise. LIDAR sensors provide an accurate information and work independently of the ambient light. However, LIDAR sensors are expensive, and provide a performance with low level of vertical resolution.

The detection of obstacles based on images can determine the type of obstacle, and with the reduction of cameras cost it is possible to integrate a large number of cameras; Furthermore, they are compact, accurate and well modelled. However, a software implementation of SLAM and optical flow algorithms in real time requires a great computational load because of the complexity of these algorithms [6]. Vision approaches based on classification need a prior knowledge

about environment to separate ground pixels from obstacle pixels. Vision approaches based on IPM allow obstacle detection under the hypothesis of a flat ground, this method is based on the perspective effect perceived from a scene when observed from two different points of view [3]. This method was introduced in [9] and exploited for obstacle detection in [2]. While this method is based on homographic transformation, it requires an important amount of computations. Architectures based on GPU (Graphic Processing Unit) platform provide a good pipeline performance but they don't meet power requirements. An FPGA (Field Programmable Gate Array) solution can provide better trade-off for power consumption and pipeline requirements of an embedded platform.

This paper is organized as follows: Section 2 describes the theoretical background and related work. Section 3 describes the proposed hardware design. Discussion and conclusion are detailed in sections 4 and 5.

2 Theoretical background

Inverse Perspective Mapping is a technique based on a geometric transformation applied on frames acquired with different point of view (either using multiple camera, or frames acquired at different time). This method belongs to the resampling effect family; an initial image is transformed to generate a view from a different position. Taking advantage of the perspective effect, this generated image is compared to a real image acquired from the new position. This comparison generates high differences for object sticking out of the ground plane. Detecting and localizing these differences in the ground plane allows to compute the object position relative to the camera.

2.1 Inverse Perspective Mapping

In Mono Inverse perspective mapping [3], a single camera is used, two frames are acquired at distinct instants t_n and t_{n+1}, as the robot moves. Odometry sensors are used as input to compute the homography matrix. This matrix encodes the effect on the images of the relative movement of the robot between the two positions for a given plane in the scene which is the ground plane in our case. The camera is considered already calibrated; i.e., it's intrinsic parameters (focal, principal point and distortion coefficients) have been determined off line. Thanks to this knowledge, optical distortions can be removed and it is possible to consider the simple Pinhole camera model to perform IPM. With this model, a 3D point of the scene (X_w, Y_w, Z_w) in the world frame is projected to pixel coordinates (u ,v) in the pinhole image with the equation (1). As noted in equation (1), K is the camera intrinsic matrix and (R ,t) encodes the rotation and translation from the world frame to the camera frame. These former parameters are named the camera extrinsic parameters. As IPM is intended to detect the ground pixels, the world frame which is the robot frame as depicted in figure(1)(a) is chosen such as the X_wY_w plane is the ground plane. Therefore, for 3D points in the world frame laying in the ground plane, $Z_w = 0$ is applied to (1) as shown in the equation (2):

(a) Mono IPM system performing a movement M_vt.

(b) Bird's Eye transformation.

Fig. 1: Inverse Perspective Mapping

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K\ R \begin{bmatrix} X_w \\ Y_w \\ Z_w \end{bmatrix} + K\ t \tag{1}$$

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K\ (\ R \begin{bmatrix} X_w \\ Y_w \\ 0 \end{bmatrix} + t\ ) \tag{2}$$

Applying Algebraic properties to (2) :

$$\begin{bmatrix} su \\ sv \\ s \end{bmatrix} = K\ [r_1\ r_2\ t] \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{3}$$

In the first acquisition, the robot frame is considered as the world frame. Therefore, each pixel coordinates are represented with the equation (4):

$$\begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} = K\ [r_{r1}^c\ r_{r2}^c\ t_r^c] \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = H_1 \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{4}$$

In the second acquisition, the position of the robot frame origin in the second acquisition is represented in the robot frame of the first acquisition.

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = K\ [r_{w1}^c\ r_{w2}^c\ t_w^c] \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} = H_2 \begin{bmatrix} X_w \\ Y_w \\ 1 \end{bmatrix} \tag{5}$$

The transformation (R_w^c, t_w^c) is computed from the equation (6) as shown in the figure $(1)(a)$:

$$\begin{bmatrix} R_w^c & t_w^c \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_r^c & t_r^c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{mvt} & t_{mvt} \\ 0 & 1 \end{bmatrix} \tag{6}$$

from the equation (4) and (5) :

$$\begin{bmatrix} s_2u_2 \\ s_2v_2 \\ s_2 \end{bmatrix} = K\ [r_{w1}^c\ r_{w2}^c\ t_w^c]\ [r_{r1}^c\ r_{r2}^c\ t_r^c]^{-1}\ K^{-1} \begin{bmatrix} s_1u_1 \\ s_1v_1 \\ s_1 \end{bmatrix} \tag{7}$$

Therefore, each ground point is represented in the camera frame I_1 and represented with the coordinates (u_1, v_1) in the image frame will be presented in the camera frame I_2 with the coordinates (u_2, v_2) in the image frame from the equation (7) :

$$H = T_{ipm} = K\ [r_{w1}^c\ r_{w2}^c\ t_w^c]\ [r_{r1}^c\ r_{r2}^c\ t_r^c]^{-1}\ K^{-1} \tag{8}$$

The transformation (8) is only correct for ground points. Therefore, the pixel to pixel value subtraction between $T_{ipm}[I_1]$ and I_2 generates low absolute values for the ground points.

2.2 Segmentation of obstacles

In obstacle detection systems, one of the important steps to extract obstacle pixels is the segmentation of binary image and thresholding is a fundamental tool for segmentation. Otsu's thresholding [10] is known as a good method. The optimal threshold is computed by minimizing the mean square errors between original image and the resultant binary image. A threshold based on Otsu's algorithm is calculated from the equations (9)(10) [10]:

$$\sigma_0^2(t) = \sum_{i=1}^{k} [i - \mu_0(t)]^2 \frac{p_i}{\omega_0} \qquad (9)$$

$$\sigma_1^2(t) = \sum_{i=k+1}^{L} [i - \mu_1(t)]^2 \frac{p_i}{\omega_1} \qquad (10)$$

The threshold produced by the equations (9) and (10) minimizes the weighted within class variance. The problem of searching the optimal threshold can be reduced to search a theshold that maximizes the between-class variance as shown in the equation (11):

$$\sigma_B^2 = \omega_0 \omega_1 (\mu_1 - \mu_0)^2 \qquad (11)$$

For quicker calculation and optimal performance in hardware implementation the equation (11) is used to find the threshold from the histogram extracted of gray-level image.

2.3 Bird's-eye transformation

This transformation allows the distribution of obstacle information among image pixels [2] and leads to efficient implementation of polar histogram in order to localize obstacles. The binarized image is projected on the ground plane in the robot frame as depicted in figure (1)(b), up to a rotation around the vertical axis and a translation in XY. C_1 and C_2 represent camera frames as shown in figure (1)(b). Any point on the ground plane P has a 3D position represented with respect to the camera $C_1$is r_{C1} [7]

$$\frac{n_{C_1}^T . r_{C_1}}{d_{C_1}} = 0 \qquad (12)$$

$$r_{C_2} = R_{C_2C_1}(r_{C_1} - t_{C_1}^{C_2C_1}) \qquad (13)$$

n_{C_1} is the ground plane normal represented in camera C_1 coordinates and d_{C_1} is the distance of the ground plane from the origin of camera C_1. The position vector r_{C_2} of the same point represented in camera C_2 is computed from(13). $R_{C_2C_1}$ is the rotation matrix from C_1 to C_2 and $t_{C_1}^{C_2C_1}$ is the translation from C_1 to C_2 presented in C_1 coordinates. The transformation required from the Original Camera C_1 to the virtual camera C_2 presenting bird's-eye view is:

$$r_{C_2} = H_{C_2C_1} \; r_{C_1},$$
$$H_{C_2C_1} = R_{C_2C_1} - \frac{1}{d_{C_1}} t_{C_2}^{C_2C_1} . n_{C_1}^T \qquad (14)$$

By using the equation (14), the homography matrix of bird's-eye view for the cameras is calculated. To use this matrix in image coordinates(pixels), camera intrinsic matrix K is required as shown in the equation (15):

$$H_{bird} = K(R_{C_2C_1} - \frac{1}{d_{C_1}} t_{C_2}^{C_2C_1} . n_{C_1}^T) K^{-1} \qquad (15)$$

The bird's eye image of the ground plane is generated by applying the homography to each pixel.

2.4 Obstacle localization

Obstacle bearing measurement Straight lines perpendicular to ground plane are parallel in the world frame. By intersecting image plane with a ray parallel to these lines in the camera frame through the camera center [7], vanishing point in the image frame is obtained to represent a point called $focus$ [2]. Thus a beam of lines is originating from $focus$ through the image to represent polar histogram. So binarized image is scanned using polar histogram to localize obstacle shapes in the image, and the tracking of pixels located in each line is done by using Bresenham's algorithm.

$$v = PX_\infty = K\,[I|0]\begin{bmatrix} d \\ 0 \end{bmatrix} = Kd \tag{16}$$

$X_\infty = (d^T, 0)^T$ is the vanishing point represented in the world frame, and v is its projection in the image frame. P being projection matrix, K is intrinsic matrix. Since two points coordinates are required to perform a line equation, the first point(u_0, v_0) is a pixel belonging to the first row of image whereas the second point is $focus$ $(0, 0)$. As noted already, image frame is translated to $focus$. Bresenham's algorithm [4] is initially presented in algorithm 1 for a line having a point coordinates (u_0, v_0) in the first row of image for the octant where $(u_0 < 0, v_0 > 0)$ and the vertical projection $|v_0|$is longer than the horizontal projection $|u_0|$ as depicted in figure (3). $I(u, v)$ is pixel value at the coordinates (u, v). $Dens$ being the number of overthreshold pixels located in line. The implementation of the algorithm is generalized to produce and trace lines in different octants.

Data: u_0,v_0
Result: $Dens$
$dv = v_0$, $du = -u_0$, $D = 2du - dv$, $v = v_0$;
for $v = v_0 \rightarrow 0$ **do**
  if $D > 0$ **then**
    $u \leftarrow u + 1$;
    $D \leftarrow D + 2du - 2dv$;
  else
    $D \leftarrow D + 2du$;
  end
  if $I(u, v) > 0$ **then**
    $dens \leftarrow dens + 1$;
end

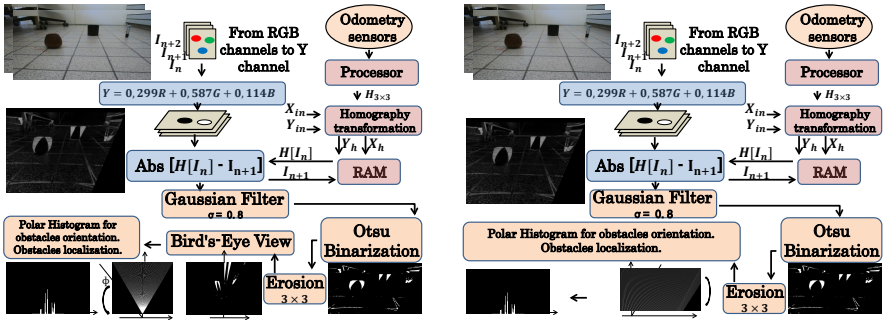
Algorithm 1: Bresenham's method implemented for a specific octant

Obstacles localization in the ground plane Obstacle shapes produced by IPM and binarization often have an isosceles triangular shape where the peak corresponds to the intersection point between ground plane and obstacle object. A method is proposed to find isosceles triangles crossed by Bresenham's lines and extract the peak of each isosceles triangle; meanwhile polar histogram is calculated. For each pixel $I(u, v)$ located in a defined line traced by Bresenham's algorithm, a factor a_r defined as a sum of neighbouring pixels located in the same image row r as shown in the equation (18):

$$a_r = \sum_{k=-l}^{l} I(u + k, v) \tag{17}$$

$$Sc = f(a_r, a_{r+1}) = \begin{cases} Sc + 1, & \text{if } a_{r+1} \leq a_r \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

l being window width, and Sc is a score refering to the possibility whether an isosceles triangle is found or not. Since pixels of lines traced by Bresenham's algorithm don't include all image pixels, extracted points don't represent the ideal points.



(a) First Hardware System design.

(b) Second Hardware System design.

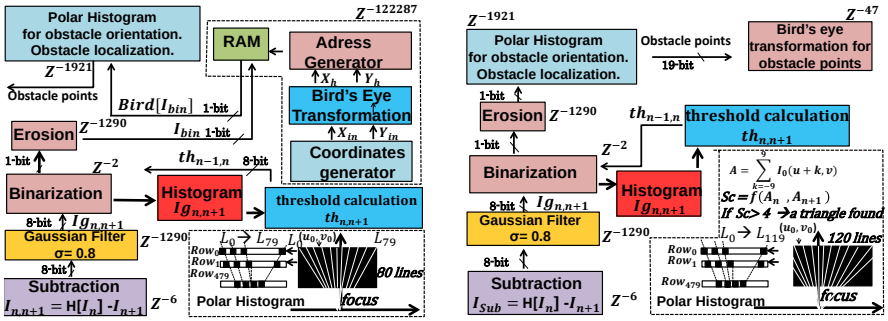
Fig. 2: The two proposed architectures

3 Hardware Design

Two hardware architectures are proposed as depicted in figure (2). In the first architecture, bird's eye transformation is applied to the whole image produced by erosion module, while this transformation is only applied to the contact points in the second architecture. Figure (2) shows the general hardware system design with the differences between the two proposed architectures. Figures (3)(4) show details of an implementation done for images of VGA resolution.

3.1 Homography and Bird's eye transformation

Homography transformation is a pixel-level operation that maps input pixel coordinates to computed output pixels coordinates (mapping operation). This operation requires to store image frames in memory. This transformation can be performed in hardware design by two methods detailed in [3]. As depicted in figure $(4)(a)$, non-sequential reads of the input image and sequential writes of the output image are performed. Each output pixel is mapped to an input pixel. Thus the calculation of inverse matrix of homography matrix produced by the equation (8) is required, this process is performed in software as proposed in [3]. An approximation is done for the non integer output coordinates, and a null pixel value is assigned to the coordinates having no correspondence with the input image. The same process is performed for bird's eye transformation. The main difference between these two transformations is that the values of the transformation matrix can be computed off-line if we assume camera movement only over x, y, ϕ_z. The drawback of this transformation in hardware implementation is the high bit-depth required for fractional of the fixed point elements of the bird's eye matrix elements and the high cost of latency time. Since bird's eye matrix is a constant matrix, the maximum value of $|y_h - y_{in}|$ is calculated off-line and is used to perform the minimum size of Block Random Access Memories (BRAMs) required for the transformation.



(a) First hardware architecture. (b) Second hardware architecture.

Fig. 3: Differences between the two proposed architectures

In our implementation done for VGA resolution, 191 rows of eroded image are stored in BRAMs in order to start bird's eye transformation.

3.2 IPM and Binarization

As shown in figure (3)$(a)(b)$, the image transformed $H[I_n]$ is generated from the previous process. A subtraction is performed between pixels from $H[I_n]$ acquired at t_n and pixels from I_{n+1} at t_{n+1} as shown in the figure (3). A FIFO module is used in order to synchronize the stream of image transformed $H[I_n]$ acquired at t_n to the image I_{n+1} acquired at t_{n+1}. The image I_{n+1} will be written in the memory replacing the image I_n. The output image of subtraction passes through a gaussian filter to remove noise and insure an optimal performance in Otsu's binarization. This gaussian filter for a standard deviation $\sigma = 0.8$ is implemented as 3×3 as a kernel. The image filtered by the gaussian kernel called $Ig_{n,n+1}$ is binarized using Otsu's threshold; This threshold $th_{n-1,n}$ is already computed from the histogram of the image filtered $Ig_{n-1,n}$. As the computation of histogram used to perform Otsu's method to find optimal threshold requires a high latency time, the threshold which binarizes the image filtered $Ig_{n,n+1}$ is a threshold $th_{n-1,n}$ calculated from the image filtered $Ig_{n-1,n}$ as depicted in figure(3). Since there is not a high variation in intensities between two sequential frames, this binarization threshold remains valid and allows to save on processing latency. The resulting binary image is then eroded with a kernel element 3×3. As erosion process and gaussian filter use a 3×3 kernel, two image rows and two pixels are stored in BRAMs to perform these operations. Thus an additional cost to latency time is imposed as depicted in figure(3). In the first architecture, a definite number of eroded image rows is stored in BRAMs to perform bird's eye transformation and provide bird's eye image $Bird[I_{bin}]$. In the second architecture, eroded image pixels simply pass to the localization module.

3.3 Localization

In the first architecture as depicted in figure(3)(a), polar histogram makes use of bird's eye image for obstacles localization. However, the potential presence of two or more obstacles will complicate the process [2]. Therefore, eroded image is used to perform polar histogram, and this is done for the second architecture. For pipeline requirements, two rotating registers of image width size are used to store two image rows, the first register is used to determine which pixels belong to lines drawn by Bresenham's algorithm, count overthreshold pixels, detect isosceles triangles and their peaks, and to compute the next coordinates in the next register. The second register stores pixels read from memory while applying Bresenham's algorithm to the first register. Figure (3)(a) introduces an example how to scan bird's Eye image 640×480 by 80 lines. The output of this module is stored in BRAMs. Each address refers to a scanned sector of the image, and the content of memory represents the number of overthreshold pixels. Figure(3)(b) introduces an example showing how to scan an eroded image 640×480 by 120 lines in the second architecture, and an example is shown to implement the equations (17) and (18) for extracting obstacles contact points with ground plane. A selection process is performed to choose the best points. Extracted points are divided into clusters, each cluster represents points assigned to same obstacle object. In the second architecture, the extracted pixels coordinates which are considered as contact points between ground plane and obstacles are transformed by bird's eye matrix to produce occupancy grid map for robot.

4 Discussion and Results

In the second architecture, Bird's eye transformation is not applied to eroded image, this is advantageous in hardware implementation because this transformation requires a high latency time, many BRAMs to store a specific number of eroded image rows, and a large amount of hardware resources.

The two proposed architecture are implemented using Xilinx Virtex 6 platform. A hardware accelerator for homography transformation and IPM algorithm has been developed in [3]. Table 2 shows the differences between our architecture and [3]. The software part of our architecture includes the calculation of homography matrix produced by the equation (8). A software solution is proposed in [3] to implement this equation by using a soft-processor, this solution is adopted and used in our architecture. In [1], a hardware system based on stereovision is proposed for obstacle detection, this architecture requires two cameras to perform the system while our architecture is a monocular vision system; furthermore, the maximum frequency of our system is better than the maximum achieved frequency in [1]. Latency time in [1] (minimum detection time) is $5.5ms$ for VGA resolution while latency time (computed to produce contact points between obstacles and ground plane) of our architecture is $7.49\mu s$ for the second architecture and $2.05ms$ for the first architecture, the computational latency time is ~ 4610 clock cycles for the second architecture, while ~ 126897 clock cycles are required to perform the first architecture, figure (3) shows the required

clock cycles for each part in the two proposed architectures. Table (1) shows the

Table 1: comparison with other obstacle detection system

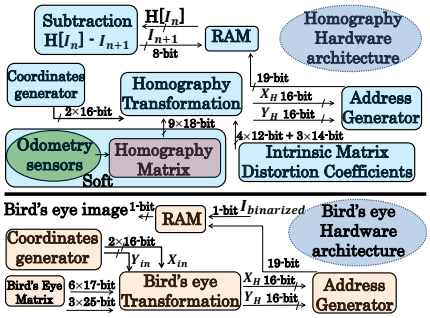
	platform	OD method	Frame rate
[8]	CPU + GPU	optical flow	25 fps 640 × 480
[11]	PC 1.73 GHz	IPM coarse detection	30 fps 720 × 480
[5]	GPU	3D reconstruction	45.8 fps 640 × 480
[1]	FPGA	stereovision	Fmax = 51.7 Mhz,180 fps 640 × 480
ours	FPGA	IPM	Fmax = 61.9 Mhz, 201 fps 640 × 480

Table 2: Comparison of our system with [3](640 × 480)

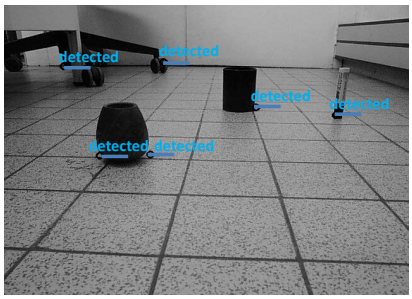
	FPGA [3]	our FPGA
IPM	Yes	Yes
Binarization	No	Yes
Localization	No	Yes
Bird's Eye	No	Yes
Frame rate	30 fps (SDRAM)	201 fps(BRAMs)

Table 3: Resources required for the two architectures

architecture	Slice Reg	Luts	RAMB36E1
First	35469	101479	87
second	34746	153623	78



(a) Implementation of homography and bird's eye.



(b) Results of obstacles localization in the ground plane.

Fig. 4: Results and homography hardware architecture

comparison of the proposed system to other obstacle detection systems. In [8], a system based on Optical flow, a computational intensive method, is used for obstacle detection. The estimation of power consumption in our architecture is around 3.9 *watt* which is clearly less than power consumption in GPU platform as [8]. In [11], a method based on IPM is used to perform a system with localization of obstacles using polar histogram. However, three sequential frames are needed to perform the system; furthermore, the proposed method is limited to vertical edges of obstacles. Figure(4)(b) shows the results of an implementation done for 640 × 480 images of the second architecture. Most of obstacles contact points with ground plane are detected. However, two contact points are detected

as two obstacle objects, this is because of the selection process for contact points. An optimization is still required to overcome that problem.

5 Conclusion

This paper presents a hardware architecture for obstacle detection and localization implemented on FPGA. An efficient solution combines Mono IPM for detection and Otsu's method, plus Bresenham's algorithm for localization. This architecture produces a pipelined design with a high frame rate. The results show the high frame rate of the system. In future, the proposed architecture will be optimized to consume less resources, and extended to a multi-camera system to generate occupancy grid map of the environment around robot.

This work has been performed by Ali Alhamwi, paid by the FUI-AAP14 project AIR-COBOT, co-funded by BPI France, FEDER and the Midi-Pyrénées region.

References

1. H. Bendaoudi, A. Khouas, and B. Cherki. Fpga design of a real-time obstacle detection system using stereovision. In *Microelectronics (ICM), 2012 24th International Conference on*, pages 1–4. IEEE, December 2012.
2. M. Bertozzi, A. Broggi, and A. Fascioli. Stereo inverse perspective mapping: theory and applications. *Image and Vision computing*, 16(8):585–590, May 1998.
3. D. Botero, J. Piat, P. Chalimbaud, and M. Devy. Fpga implementation of mono and stereo inverse perspective mapping for obstacle detection. In *Design and Architectures for Signal and Image Processing (DASIP)*, pages 1–8. IEEE, October 2012.
4. Bresengham.J. Algorithm for computer control of a digital plotter. *IBM Systems*, 4(1):25–30, January 1965.
5. C. Cesar, T. Mendes, F. S. Osorio, and D. F. Wolf. An efficient obstacle detection approach for organized point clouds. In *Intelligent Vehicles Symposium*, pages 1203–1208. IEEE, June 2013.
6. J. Ha and R. Sattigeri. Vision-based obstacle avoidance based on monocular slam and image segmentation for uavs. In *Infotech@Aerospace 2012*, pages 1–9. AIAA, June 2012.
7. R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, UK, 2004.
8. C. Y. He, C. T. Hongand, and R. C. Lo. An improved obstacle detection using optical flow adjusting based on inverse perspective mapping for the vehicle safety. In *Intelligent Signal Processing and Communications Systems (ISPACS), 2012 International Symposium on*, pages 85–89. IEEE, November 2012.
9. H. A. Mallot, H. H. Blthoff, J. J. Little, and S. Bohrer. Inverse perspective mapping simplifies optical flow computation and obstacle detection. *Biological Cybernetics*, 64(3):177–185, January 1991.
10. N. Otsu. Threshold selection method from gray-level histogram. *IEEE TRANSACTIONS ON SYSTEMS.*, SMC-9(1):62–66, January 1979.
11. Z. Yankun, H. Chuyang, and W. Norman. A single camera based rear obstacle detection system. In *Intelligent Vehicles Symposium*, pages 485–490. IEEE, June 2011.