Nitesh Kumar Dixit, Sandeep Sharma, Surendra Yadav, Lav Saxena / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 2,Mar-Apr 2012, pp.1010-1015

Vision-Based Autonomous Security Robot

Nitesh Kumar Dixit, Sandeep Sharma, Surendra Yadav, Lav Saxena
[1, 3] Deprtment of Electronics and Communication,
[2, 4] Department of Electrical Engineering,
Bhartiya Institute Of Engineering and Technology, Near Sawali Circle, Sikar.

Abstract
The goal of this paper is to develop a computer vision system that enables a robot to navigate the hallways of buildings using a generic webcam as the only sensor. OpenCV2.0 programmed in C++ is the primary tool used to develop the vision system software. Three algorithms were developed to identify the center of the hallway and guide the robot in the correct direction. The first two algorithms use a generic filter (normal, median, or Gaussian) followed by edge detection and then corner detection on the edge- detected image. The first algorithm identifies the strongest vertical lines on an image. Averaging the horizontal coordinates of the vertical lines indicates the location of the center of the hallway relative to the robot. The second algorithm utilizes the trapezoidal shape of the hallway formed where the floor meets the walls, as seen from the perspective of the robot. The y-coordinates associated with the trapezoid's legs are then compared to estimate robot orientation with respect to the walls. The third algorithm uses color to segment the floor from the rest of the features in the image (walls, ceiling, and obstacles). Once again, the trapezoidal shape appears and the center of the hallway is determined based on the location of the highest y-valued pixels identified as floor pixels. Test data indicates that none of these algorithms is singularly sufficient; however, combining their results they can identify the direction a robot must turn to remain in the center of the hallway with 96.6% accuracy. Furthermore, leveraging the results of multiple algorithms produces more robust navigation, where one algorithm covers over the shortcomings of another. The vision system architecture is designed to execute algorithms in parallel. Such a structure enables the addition and removal of algorithms without adversely affecting the system as a whole. Further algorithms may be developed and easily added to improve navigation. Additionally, the system may intelligently ignore results from algorithms that are recognized as inappropriate for certain situations.

Keywords: Generic Filter, OpenCV2.0, Robot, Trapezoidal, Vision.

I. Introduction
VBASR (Vision-Based Autonomous Security Robot) is designed to patrol the floor hallway of the buildings. Essentially, VBASR is a mobile, intelligent security camera able to locate and navigate to specific rooms and photograph any intruders it encounters. VBASR: The primary goal for this paper is a robust, vision-based navigation system. Sage et al. [1] performed similar work using computer vision to detect motion for security systems. DeSouza and Kak [2] present an exhaustive survey of computer vision techniques, which provided inspiration for VBASR. Other excellent resources for familiarization with foundational computer vision concepts and terminology include [3], [4], and [5]. VBASR is primarily a machine vision project; therefore, a chassis that requires little modification is desirable. Figure 1 shows the iRobot Create chassis selected as the robot platform. A simple webcam mounted in the cargo bay and an onboard computer are the only additional hardware necessary for VBASR. Microsoft Robotics Developers Studio (MRDS) is used to control the iRobot Create and OpenCV2.0 computer vision libraries programmed in C++ are utilized to implement the vision algorithms.

Figure 1. iRobot Create and accessories

Author Details
The first author is **Nitesh Kumar Dixit,** he is currently working as Assistance Professor, in BIET, Sikar. He has obtained his M.tech (Embedded System) degree from SRM university, Chennai and B.E. (ECE) from SEC, Dundlod (jhujhunu. Raj). His area of interest is Cryptography, Image

Processing etc.

The Second author is **Sandeep Sharma,** he is currently working as Sr. Lecturer, in BIET, Sikar. He is pursuing his M.tech (Power System) degree from SKIT, Jaipur and B.E. (EE) from Mardhur Engineering College, Bikaner (Raj.). His area of interest is Power system, Robotics, Image Processing etc.

The Third author is **Surendra Yadav,** he is currently working as Lecturer, in BIET, Sikar. He is pursuring M.tech (Digital Communication) degree from SEC, Dundlod and B.E. (ECE) from SBCET, Jaipur (Raj.). His area of interest is Real Time System, Cryptography, Image Processing etc.

The Forth author is Lav Saxena**,** he is currently working as Lecturer, in BIET, Sikar. He is pursuring M.tech (Power System) degree from Poornima Engg. College, Jaipur and B.E. (EE) from KITE, Jaipur (Raj.). His area of interest is Power Electronics, Signal and Image Processing etc.

Problem Description

Given an image of the hallway, such as the one shown in Figure 2, how would a robot choose which direction to turn to travel down the center of the hall? Humans can solve this problem using vision intuitively without a concentrated effort. VBASR must make such decisions using similar information, but must accomplish it within the limitations of electronic circuitry.

One obvious difficulty with the image in Figure 2 is the lack of depth perception. Stereoscopic vision could be used to ascertain depth information [6], but VBASR is designed to use a single webcam, making depth perception on a single image extremely difficult. Three different algorithms were developed to navigate using a webcam as the only sensor. The compilation of those three algorithms constitutes VBASR: A resolver function then considers information from each individual algorithm and determines the final direction of travel.

Figure 2. Example image of hallway

A library of hallway images was used to test the accuracy of the VBASR's vision system. Every image was examined and the desired direction assigned by human observation. The results calculated by the algorithms were then compared to the desired navigation results to evaluate the success rate of each individual algorithm. A final requirement for VBASR is that it should be able to react faster than humans so that it can function properly in its environment. To do so, VBASR must be able to process an image and begin responding within 190ms [7]. Currently, VBASR processes about ten images per second which meets the requirement.

ii. lines algorithm

The first approach attempted was to find the strongest vertical lines in the image. Main vertical lines in a hallway include windows, doors, pictures, etc. All of these are found on the walls.

Figure 3. Lines algorithm theory

Thus, if the wall locations can be determined on either side, then the average of the wall locations should be the approximate center of the hallway, as shown in Figure 3. Red lines represent 'strong' vertical lines and the maroon line represents the average of the x-values of the red lines.

Feature Extraction

To find the strongest vertical lines of the image, a line (edge) detection algorithm is required. A filter must be used on the image as a prerequisite for the line detection algorithm. The edge detection algorithm detects many artifacts that are undesirable, as seen in Figure 4. The best overall VBASR algorithm utilizes a median blur, which returns the median of the neighborhood of the given pixel. Figure 5 shows the desired result when using edge detection on a filtered image. The Canny algorithm [8] is used for all the edge detection required by VBASR.

Figure 4. Edge detection on an un-blurred image

One major problem with the image in Figure 7 is that the computer still has no simple way of identifying the strongest vertical lines. Therefore, corner detection is

used for the final stage of feature extraction. Corner detection performed on the edge-detected image enables the program to obtain data points on the lines, which can be used to find the strongest vertical lines. Figure 8 shows all of the corners detected by the algorithm marked with small white circles. The x and y coordinates of each corner is output to an array for further processing.

Processing

The next step is to find the strongest vertical lines using the corners identified in Figure 6. First, the image from the webcam is split into sixteen vertical bins. These bins allow a histogram-like transformation by counting the number of corners found within the different bins. The result is a sum of the number of corners located in each bin. The number of corners in each bin is compared to a constant value, and, if the number of corners is greater than that threshold, the bin is considered to have a strong vertical line.

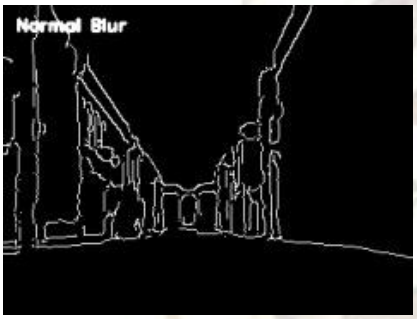

Figure 5. Edge detection on a blurred image

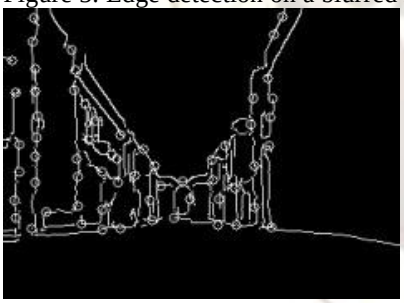
Figure 6. Corner Detection on image in Figure 5

The image shown in Figure 7 is the same image used in all the previous figures of this section. The thin white lines delineate each bin, the black lines represent the strongest vertical lines (as determined by the corners in Figure 6), and the thick white line represents the x-value average of the strongest vertical lines.


Figure 7. Lines algorithm processing example

To determine the direction found by the lines algorithm, the average of the strong vertical lines is found and compared to seven equally distributed direction bins (hard left, left, etc.). If the thick white line in Figure 7 were located on the left edge of the image, then it would evaluate to Hard Left. Likewise, if it were in the center of the image, it would evaluate to Straight.

Results And Shortcomings

In practice, the optimized lines algorithm has an accuracy rating of 79.3%. The algorithm performed worst on images requiring the action of Hard Right, where it achieved a success rate of only 26%.

One shortcoming with this method appears when vertical lines fall directly on the separation line for a bin. When this happens, the corners found on that line may be split in between two bins and the line may be ignored completely.

A second shortcoming occurs when VBASR is oriented directly at a wall (i.e. the image does not contain the center of the hallway at all). In these cases, the algorithm generally finds only one or two strong vertical lines. Depending on where these few lines are found, it may determine a wildly inaccurate direction. If the lines algorithm only detects one or zero strong lines, the algorithm fails and the resolver function ignores the lines algorithm when deciding the final direction for VBASR.

Iii. Corners Algorithm

After observing several images of the hallway, it was noted that the floor in most images forms a trapezoidal shape, as outlined in Figure 8. The trapezoid is created by the intersection between the floor and the walls. Shi and Samarabandu [9] called these lines corridor lines and used the intersection of the corridor lines to aid navigation. VBASR utilizes the corridor lines differently to develop the corners algorithm. If one corridor line is higher on the image than the other, then VBASR is facing the longer, lower corridor line's wall and needs to adjust in the opposite direction. In Figure 8, the corridor lines are marked in orange and the top of the trapezoid is blue. In practice, the edge detection algorithm actually finds the edge of the colored tile rather than the corner of the floor and wall.

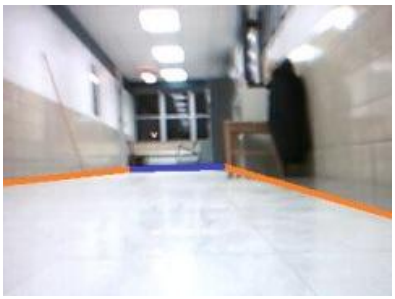
Figure 8. Corners algorithm theory

Feature Extraction

The feature extraction for the corners algorithm is similar to that of the lines algorithm. First, a blur is used on the image to eliminate artifacts from the Canny line-detection algorithm. After the Canny algorithm, corner detection is performed on the line- detected image. In Figure 9, the lower left and right-hand sections of the image are boxed off to aid viewers in understanding how the corners algorithm operates. These boxes denote the regions where the algorithm searches for corridor lines.

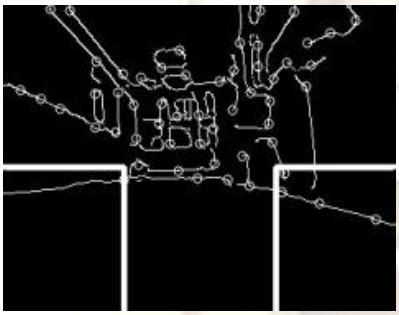
Figure 9. Feature extraction for the corners algorithm

Frequently, the lines detected by the Canny algorithm accurately define the corridor lines, but the corner-detection algorithm fails to find a corner (denoted by white circles) on the corridor lines. (Note that the left-hand box in Figure 9 is an example of such a case.) To aid the corners algorithm, two vertical lines are drawn on the image frame on both sides of the image. The extra vertical lines help the algorithm locate corners on the corridor lines,

Processing

Each of the corners found within the boxed-off sections of the image are generally on the legs of the trapezoid (i.e. along the border where the floor meets the wall). For each of the corners within a box, the x and y-values are averaged to minimize the effect of outliers. The average y-values are then compared and the leg with the higher y-value indicates the direction VBASR should turn. The distance between these final averages also indicates the strength of the turn. Figure 10 shows an example of the complete corners algorithm. The target marks indicate the averages of the corners located in each box. When y-values for the two target marks are compared, Figure 10 evaluates to Slight Left. Improves the overall performance of the system because it sometimes finds the correct direction for images on which the other two algorithms fail.

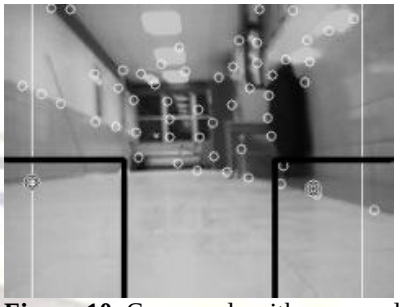
Figure 10. Corners algorithm example

Surprisingly, the corners algorithm fails the most for Slight Right and Slight Left images. Because the corners algorithm averages all the corners found within the boxes it is more likely to find large differences rather than smaller ones. As such, the corners algorithm performs better for large misalignments and, thus, complements the lines algorithm well, since the lines algorithm tends to fail on the Hard Left and Hard Right turns. An obvious shortcoming of this algorithm is that not all of the corners found within the boxed-off regions of the image are directly on the trapezoidal legs.

Iv Colors Algorithm

The third algorithm implemented takes advantage of the color difference between the floor and the walls. In most buildings, the floor color is distinguishable from the wall color. If the floor can be identified and marked, then the image becomes a binary image of "floor" and "not-floor."

Feature Extraction

An OpenCV library command called "flood fill" is used for the colors algorithm. A single pixel is picked as the seed point and then the neighborhood of that pixel is evaluated. If the neighboring pixels are similar enough to the seed point then all of the similar neighboring pixels are set to a predefined value, such as the red shown in Figure 11.

Processing

After a binary image is achieved the resulting image is scanned from the top down. The first row with more than twenty red pixels is selected and the x-values for those pixels are averaged. The result is considered the center of the hallway where the thick pink line indicates the decision line. Finally, the direction is determined by comparing the location of the decision line with the seven direction bins, in the same manner as the lines algorithm discussed above. This particular example evaluates to Straight, as shown in Figure 12.

Figure 11. Example of the flood fill command

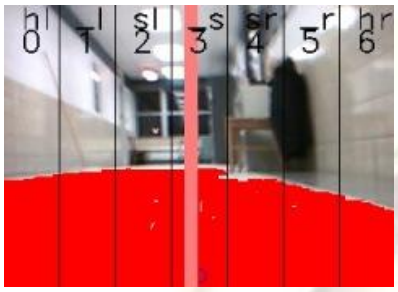

Figure 12. Colors algorithm with direction bins

Results and Shortcomings

Easily the best of the three algorithms, the colors algorithm has an accuracy rating of 94.8%. This algorithm has no particular category of images for which it performs poorly. Unfortunately, many shortcomings still exist for this algorithm.

The first shortcoming is that the seed point cannot be adjusted once it is set. It is possible for the seed point to fall on the wall instead of the floor. If this occurs, the flood fill command will paint the walls red, which is clearly undesirable.

The second shortcoming is that tiles of different colors can confuse the algorithm. To work around these shortcomings several different seed points are used. The pixel value at each seed point is identified and if that point is either white or orange (to catch the occasional orange tile) then it is evaluated using flood fill.

V. Resolver

After all three algorithms independently determine a direction to navigate, they are resolved into a single direction for the entire system. The resolver ignores algorithms when it detects a failure (e.g. the color algorithm is ignored if it paints the walls or ceiling red). Since the resolver evaluates each algorithm in parallel, the system architecture is such that algorithms can be added and removed without compromising the integrity of the system as a whole.

Vi. Results

Resolving the lines algorithm, corners algorithm, and colors algorithm enables VBASR to achieve an overall accuracy rate of 96.6%, as shown in Table 1. Table 1 also details the success rates associated with each algorithm in the seven general navigation directions. Individually, the colors algorithm outperforms both lines and corners algorithms, however, if either of these algorithms is eliminated then the resolver's success rate decreases.

TABLE 1. Final Vision System Results (%)

	Lines	Corners	Colors	Resolved
Hard Left	33.3	13.3	93.3	93.3
Left	87.9	55.2	93.1	100
Slight Left	97.1	28.6	91.4	94.3
Straight	96.4	48.2	96.4	98.2
Slight Right	97.6	29.3	92.7	100
Right	57.1	46.9	96.9	95.9
Hard Right	26.3	21.1	100	94.7
Totals	70.8	34.7	94.8	96.6

Due to the nature of the resolving function, the values for the lines and corners algorithm shown in Table 1 are not the highest percentages achieved for each individual algorithm. This phenomenon occurs because of the interplay between the various algorithms during resolution.

TABLE 2. Optimized Corners Algorithm (%)

	Lines	Corners	Colors	Resolved
Hard Left	73.3	46.7	93.3	73.3
Left	79.3	89.7	93.1	98.3
Slight Left	100.0	68.6	91.4	97.1
Straight	100.0	69.6	96.4	100.0
Slight Right	97.6	51.2	92.7	100.0
Right	45.9	87.8	96.9	90.8
Hard Right	10.5	52.6	100.0	57.9
Totals	**72.4**	**66.6**	**94.8**	**88.2**

The highest accuracy ratings for the lines and corners algorithms are 79.3% and 66.6% respectively. If the parameters are set to optimize the lines or corners algorithms, the total resolved percentage decreases, as demonstrated in Table 2. As stated above, when either of the lower percentage algorithms is removed, the resolved accuracy rating lowers, which demonstrates the benefit of using multiple algorithms in the vision system's parallel architecture. Initial testing of the vision system on the iRobot Create platform yielded promising results. A webcam was mounted to the cargo bay of the robot and the robot was manually controlled to follow the real-time decisions of the resolver. Observing the resulting navigation behavior, it was determined that with the addition

of control software VBASR will be capable of autonomously navigating down the center of the hallway.

Vii. Future Work

The discussion in this paper focuses on navigation of a hallway where obstacles are located only along the walls. Although not generally observed in the halls of the engineering building, it is possible that obstacles may be placed in the center of the hallway, obstructing the path of the robot. To handle these situations, obstacle avoidance will be implemented utilizing an algorithm similar to the colors algorithm. Creating a binary image of "floor" and "not-floor" enables simple detection of obstacles because the obstacles should generally be a different color than the floor. The orange tiles in Figure 11 that are not marked red give a general feel for how the algorithm could identify obstacles. These orange tiles also pose a challenge as they should be identified as "floor" rather than "not-floor." Using the more robust colors algorithm shown in Figure 12, the floor can be identified, and, thus, the obstacles will be isolated. The colors algorithm shown in Figure 12 evaluates the seed points for either orange or white. Any other color would be identified as an obstacle. Other possibilities for obstacle avoidance have been explored by Marques and Lima [10] as well as Ohya et al. [11] After adding obstacle avoidance, the vision system will be integrated with Microsoft Robotics Developers Studio to enable autonomous control of the iRobot Create, completing VBASR's primary navigational requirements. From here, more sophisticated work, such as motion detection (to locate intruders), will begin.

Viii. Acknowledgment

The authors thank BIET's Electronics and Electrical Engineering Department sponsoring VBASR Vision System.

Ix. References

[1] Sage, K., and S. Young. "Security Applications of Computer Vision." *IEEE Transactions on Aerospace and Electronic Systems* 14.4 (1999): 19-29. Aug. 2002.

[2] DeSouza, G. N., and A. C. Kak. "Vision for Mobile Robot Navigation: A Survey." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.2 (2002): 237-67. Aug. 2002.

[3] Davies, E. R. *Machine Vision: Theory, Algorithms, Practicalities*. San Francisco: Morgan Kaufmann, 2005.

[4] Forsyth, D., and J. Ponce. *Computer Vision: a Modern Approach*. Upper Saddle River, N.J.: Prentice Hall, 2003.

[5] Shapiro, Linda G., and George C. Stockman. *Computer Vision*. Upper Saddle River, NJ: Prentice Hall, 2001.

[6] Scott, D., and F. Aghdasi. "Mobile Robot Navigation In Unstructured Environments Using Machine Vision." *IEEE AFRICON* 1 (1999): 123-26. Aug. 2002.

[7] Kosinski, R. J. "Literature Review on Reaction Time." Clemson University, Aug. 2009. 10 Nov. 2009. http://biae.clemson.edu/bpc/bp/Lab/110/reaction.htm

[8] Canny, J. "A Computational Approach to Edge Detection." *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.6 (1986): 679-98. Jan. 2009.

[9] Shi, W., and J. Samarabandu. "CORRIDOR LINE DETECTION FOR VISION BASED INDOOR ROBOT NAVIGATION." *IEEE CCECE* (2006): 1988- 991. Jan. 2007.

[10] Marques, C., and P. Lima. "Multisensor Navigation for Nonholonomic Robots in Cluttered Environments." *IEEE Transactions on Robotics and Automation* 11.3 (2004): 70-82. Oct. 2004.

[11] Ohya, I., A. Kosaka, and A. Kak. "Vision-Based Navigation by a Mobile Robot with Obstacle Avoidance Using Single-Camera Vision and Ultrasonic Sensing." *IEEE Transactions on Robotics and Automation* 14.6 (1998): 969-78. Aug. 2002.