Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2012, Article ID 240476, 14 pages
doi:10.1155/2012/240476

Research Article

Robot Navigation Control Based on Monocular Images: An Image Processing Algorithm for Obstacle Avoidance Decisions

William Benn and Stanislao Lauria

Department of Information Systems and Computing, Brunel University, Uxbridge UB8 3PH, UK

Correspondence should be addressed to Stanislao Lauria, stasha.lauria@brunel.ac.uk

Received 21 June 2012; Accepted 10 August 2012

Academic Editor: Zidong Wang

This paper covers the use of monocular vision to control autonomous navigation for a robot in a dynamically changing environment. The solution focused on using colour segmentation against a selected floor plane to distinctly separate obstacles from traversable space: this is then supplemented with canny edge detection to separate similarly coloured boundaries to the floor plane. The resulting binary map (where white identifies an obstacle-free area and black identifies an obstacle) could then be processed by fuzzy logic or neural networks to control the robot's next movements. Findings show that the algorithm performed strongly on solid coloured carpets, wooden, and concrete floors but had difficulty in separating colours in multicoloured floor types such as patterned carpets.

1. Introduction

Autonomous mobile robots need the capability to navigate along the hallway avoiding walls in indoor environments. A number of methods have been proposed to solve the navigation problems, based on different sensor technologies such as odometry, laser scanners, inertial sensors, sonar, and vision. Missing information due to sensor temporal failure or communication delay is one of the critical aspects when dealing with sensory data for robot navigation. In [1, 2] some solutions to tackle the missing information and filtering problems have been proposed. Also a multisensor architecture could be used to design robots. While combining different sensor types, such as ultrasound, vision, and infrared, may collectively result in a more accurate decision, it could also pose increasing costs and complexity [3].

This paper will be focusing on how effective vision alone can be used as a tool for navigation and collision avoidance. One notable challenge is providing autonomous

navigation in a dynamic environment, which Saffiotti [4] describes as *real world environments that have not been specifically engineered for the robot*.

Vision is one of the most important senses to a human being and in the past decade there has been an increased interest to use images in robotics. Machines may lack the vast knowledge of object recognition that a human brain can provide but the amount of computational power available in modern times make such machine vision a viable choice of input, and unlike a human eye, machine vision does not degrade over time providing consistent image capture. Images from a coloured web camera are used here as the source of information for this task. Visual sensors can provide plenty of information, however the environment they capture is often very dynamic and elements and features to be detected can change with the environment (i.e., the floor, door colour). Still to navigate successfully, a robot needs to distinguish between what is and is not navigable. By analysing each image frame, the system should be able to identify (if any) the available navigable areas.

Broadly speaking, there are two navigation strategies: *map-based navigation* and *map-less navigation*. In this paper, we focus on the latter. In indoor environments, the robot has often to navigate along the hallway while avoiding obstacles. Then, the navigation strategies are determined by capturing and extracting relevant information about the elements in the environment. These elements can be the walls, edges, doorways, and so forth and it is not necessary to calculate the absolute positions of these elements of the environment. The navigation problem is well studied (see, e.g., [5]). Whereas, [6, 7] are some examples of strategies developed for the detection of obstacles or edge detection using vision systems. However, often these methods are dependent on the environment around a robot. For example, in [7] it is not clear how the system would react to changes in the floor patterns, whereas in [6] Neural Networks strategies are considered during the camera calibration phase to tackle this issue. References [8, 9] have investigated the use of optical flow to identify a dominant ground plane. However, their assumption is that the floor is the dominant plane.

In our paper, we extend these ideas on using the floor to calculate the correct values for the parameters necessary to extract the required information from each image. Then, to identify the obstacles, the sequential use of *colour iImage segmentation* and then *edge detection* strategies has been investigated. That is, a two steps strategy has been used.

Step 1: Image Segmentation

Step 2: Edge Detection

Each of the steps above is based on one of two basic properties of intensity values: discontinuity and similarity. Once the image has been processed in this way, fuzzy logic, neural networks, and so forth could then be considered to optimise decisions and control the robot navigation strategies.

Colour segmentation will determine obstacles from the floor while canny edge detection supplements the colour segmentation by finding sharp changes in colour gradients. Colour image Segmentation is based on partitioning an image into regions that are similar according to a predefined criteria. Whereas the aim of the edge detection stage is to partition an image based on abrupt changes in intensity. In similar research, these two steps are not always applied independently or only one of the two is applied (see, e.g., [10, 11]). In our paper both steps are applied as a consecutive sequence to the image. A measure on the effect on the success rate of each step is also investigated. Each of the steps mentioned above is discussed in detail below. Then, results are presented and discussed.

1.1. Image Colour Segmentation

For robot navigation, image segmentation is the process of decomposing an image into parts which should be meaningful to identify obstacle-free areas.

A more formal definition of segmentation can be given in the following way [12]. Let I denote an image and let H define a certain homogeneity predicate. Then the segmentation of I is a partition P of I into a set of N regions R_n, $n = 1, \ldots, N$, such that

(1) $\bigcup_{n=1}^{N} R_n = I$ with $R_n \bigcup R_m \neq 0; n \neq m$,

(2) $H(R_n)$ = true for all n,

(3) $H(R_n \cup R_m)$ = false if R_n and R_m adjacent.

Condition (1) states that the partition has to cover the whole image, condition (2) states that each region has to be homogeneous with respect to the predicate H, and condition (3) states that the two adjacent region cannot be merged into a single regions that satisfies the predicate H. The desirable characteristics that a good image segmentation should exhibit have been defined in [13].

Several colour representations are currently in use in colour image processing. The most common is the RGB space where colors are represented by their red, green, and blue components in an orthogonal Cartesian space. Most cameras will capture an image using the RGB colour space.

However, colour is better represented in terms of hue, saturation, and intensity. An example of such a kind of representation is the HSV space. HSV rearranges the geometry of RGB in an attempt to be more intuitive and perceptually relevant (see e.g., [14]).

The main approaches in image colour segmentation are based on partitioning an image into regions that are similar according to a set of predefined criteria. These segmentation methods are based on sets of features that can be extracted from the images such as pixel intensities. Thresholding, clustering, and region growing are examples of such approaches. Extensive work has been done in this area (see e.g., [10]).

Thresholding is one of the simplest and most popular techniques for image segmentation. The threshold can be specified using a heuristic technique based on visual inspection of the histogram but this approach is operator-dependent. If the image is noisy, the selection of the threshold is not trivial. Thus, more sophisticated methods have been proposed. The Balanced Histogram Thresholding (BHT), see for example [15], is a histogram-based thresholding method. The BHT approach assumes that the image is divided in two main classes: the background and the foreground. The BHT method tries to find the optimum threshold level that divides the histogram in two classes. In general, thresholding creates binary images from grey-level ones by turning all pixels below some threshold to 0 and all pixels about that threshold to 1.

In this paper, a pre-defined area of the image (red area in Figure 5) has been used to calculate the threshold values. The selected rectangular area used is located at the bottom of the image because this area is likely to contain the floor. Within this area of selection colour thresholds are calculated to define the criteria to process in a meaningful way each pixel of the image during the successive phases of the segmentation process discussed below. Further details of the thresholding step are discussed in Section 3.

There is extensive work investigating different algorithms to segment regions by identifying common properties in order to separate an image into regions corresponding to objects. (see e.g., [16]).

In [17] a multiphase image segmentation model for color images is discussed. It mainly focuses on homogeneous multiphase images. It only considers the global information of the given image, thus it cannot deal with images with inhomogeneity.

Refernce [18] applies relative values for R, G, and B components on each pixel for image segmentation. He observed traffic signs in an open environment and segmented the red color in such a way that if green and blue colors in a pixel are summed up and compared with red color, it gives relatively 1.5 times higher values for the red component in pixel. If the pixel has relatively higher red component, it determines as the featured pixel. A binary segmented image is then created using the known coordinates of the featured pixels.

Refernce [19] proposed a detection and recognition algorithm for certain road signs. Signs have the red border for warning signs and a blue background for information signs. A car has a mounted camera that gets images. Colour information can be changed due to poor lighting and weather conditions such as dark illumination and rainy and foggy weather. To overcome these problems they proposed two algorithms by using RGB color image segmentation.

Refernce [20] focused on identifying similar colour domains from human skin and vegetables. The advantage of this solution is that it does not require converting from the RGB colour space (allowing the source of the captured image to be worked on directly) and was robust against various illumination. To avoid converting RGB to other colour spaces such as HSV, [20] devised a method which uses 5 constant threshold variables (α, $\beta 1$, $\beta 2$, $\gamma 1$, and $\gamma 2$) to determine whether an RGB pixel is within a specific colour zone. Assuming the following variable values:

r = Red value of the pixel
g = Green value of the pixel
b = Blue value of the pixel
α = Minimum red threshold value (0–255)
$\beta 1$ = Minimum red-green component value (0–255)
$\beta 2$ = Maximum red-green component value (0–255)
$\gamma 1$ = Minimum red-blue component value (0–255)
$\gamma 2$ = Maximum red-blue component value (0–255).
The algorithm considers a pixel to be within a certain colour range if

(1) $r > \alpha$,

(2) $\beta 1 < r - g < \beta 2$,

(3) $\gamma 1 < r - b < \gamma 2$.

Some initial evaluations of the [20] technique applied to indoor navigation domains have shown that it captured a too broader amount of the threshold from the target floor surface. Therefore, in the present paper a modification of the above algorithm has been investigated. In particular, an additional constant ($\alpha^{\max}$) has been added to hold the maximum red threshold while the existing red constant ($\alpha^{\min}$) was used to hold the minimum red threshold. The first rule was then modified as follows:

(1) $\alpha^{\min} < r < \alpha^{\max}$.

After a few tests, the optimal settings found for the α parameters were the following. In higher illuminated conditions and pastel coloured environments $\alpha^{\max}$ is most efficient at being set to higher values such as a range between 170 and 200. $\alpha^{\min}$ is best set to a midrange value between the 75 to 90 range. In low illumination conditions $\alpha^{\max}$ is most

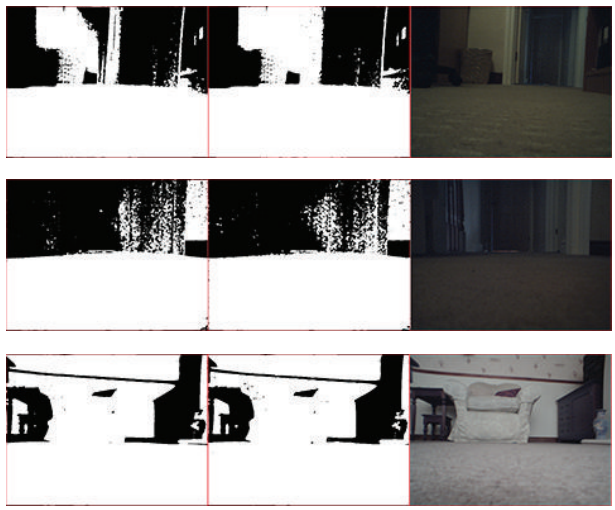


Figure 1: Comparison of original colour segmentation technique against modified rule set. Left image modified rule set, middle image original rule set, and right image raw capture.

efficient between low midrange values such as 60 to 80. $\alpha^{\min}$ should be set to a low range between 20 to 40. A higher broader range is needed under high illumination as it is most likely that obstacles will reside in the lower colour ranges. This broad range can be a downfall when obstacles are of a similar colour to the surrounding environment, which is where the edge supplementation is expected be of a great aid.

Figure 1 shows that the modified rule set investigated in the present paper picks up less noise from the image. It is also better at picking up colours that are similar to the floor plane, although the effect of this varies depending on the difference of change. From some initial tests, there is evidence that the modified algorithm keeps the floor threshold values correct when there are some small illumination changes caused by the robot's movement.

1.2. Edge Detection

Colour segmentation alone is not enough to fully segment an image, gaps were left by noise and areas of a similar colour to the floor plane were misinterpreted as traversable space. To eliminate this issue a separate edge map was produced from the captured image which was then processed by a probabilistic Hough algorithm to identify strong lines in an image.

It was decided that the best edge detection method for the project was the canny edge implementation. It excels in identifying strong edges with a lower number of line disconnections, it also picks out major details from an object [21, 22], while it is weaker at identifying minor details, we are only interested in the silhouette of an obstacle.

Once the canny edge map has been generated the probabilistic Hough transform can be applied to the image. The principle of this procedure is to scan through each pixel in a binary image finding all lines that could fit through this point. If a line fits through enough points then it is considered significant enough to keep [23]. Each point is picked randomly and once enough points have been passed through by a line, then they are removed from any subsequent scanning. This is then repeated until all points are eliminated or there are not enough points left to identify a significant line. The implementations used for this solution are

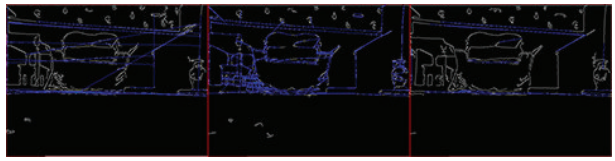


Figure 2: Comparison of Hough line parameters. Left image large maximum pixel gap value, middle image low line votes, and segment length value, right image optimal found settings.

from the OpenCv library, there are various parameters that can be passed to the probabilistic Hough transform.

Line votes number: of points a line must pass through to be considered significant

Minimum segment: length Minimum length a line must be to be selected

Maximum pixel gap: The biggest gap between points on a line that there can be.

After a few tests, the optimal settings found for the above parameters were the following:

Line votes = 80 lines

Minimum segment length = 20 pixels

Maximum pixel gap: Minimum pixel gap value.

In Figure 2 it is possible to note that when a large pixel gap is allowed lines will often extend across multiple disjointed edges from the canny map. This is not desirable for our purposes as it could fill legitimate gaps that a robot would be able to pass through. However, when a small segment length value and a low line vote count is set we end up with many short lines that could easily be combined into a single long line, this again is not desirable as the more lines there are, the larger the processing time that is required to apply the line information to the colour segmentation map. See Figure 3 for a list of images with the optimal found probabilistic Hough line parameters (with blue lines indicating the Hough lines).

Output of the probabilistic Hough transform was an array of lines. To apply this information to the colour segmentation map, a polygon was drawn from the start and end points of each line to the top of the image. Figure 4 shows a comparison between edge supplemented and nonedge supplemented segmentation maps.

2. Implementation

The algorithms have been implemented in `C++` because of the high-performance libraries available for this language. Additional processing was completed using the OpenCV library.

To calculate thresholds for the rules defined in Section 1.1, a rectangular area is selected from the image (Figure 5). Within this *mask*, thresholds are calculated as shown in Listing 1. In particular, each pixel is iterated, updating a colour threshold only when it is less than or bigger than the current threshold (depending on if it is a minimum or maximum threshold). Once the minimum and maximum thresholds have been calculated they can be compared against all the pixels in the captured image. If a pixel's RGB value is between the desired threshold, then the pixel can be marked as white otherwise as black as shown in Listing 2. To apply the Hough line information from the line array is simply a matter of drawing a black area onto the existing binary map, this is achieved by plotting a 4 sided polygon. Care must be taken to determine the correct winding order to avoid a twisted hourglass like shape; this

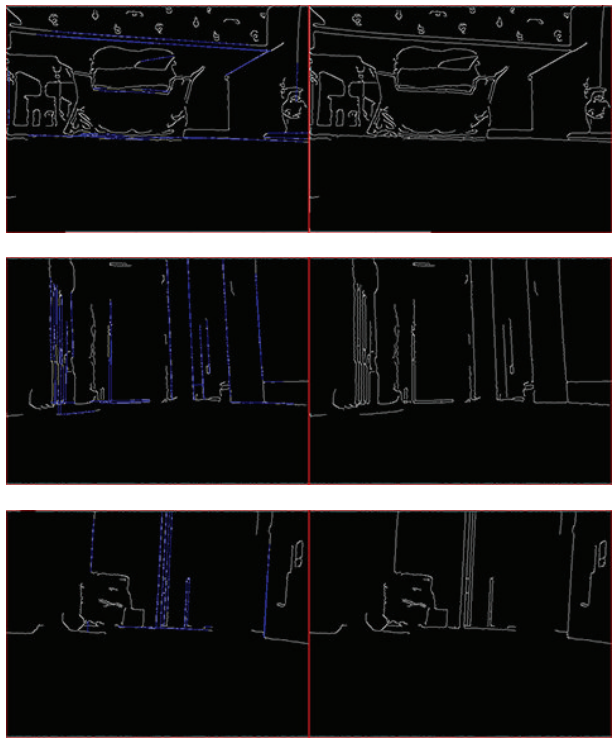


Figure 3: Optimal Hough line parameters applied to canny edge map.

is easily rectified by checking whether the first point of the line is to left or right of the end point and changing the point drawing order as shown in Listing 3.

3. Results

The algorithm discussed above has been tested using images from an indoor environment. The same set of images has been used to test different settings. For each setting, every image has been processed by a different combination of algorithms. The following four different settings have been tested:

Original. The image segmentation algorithm [20].

Original and edge. The original and the edge detection algrithms.

Modified. The modified image segmentation algorithm presented in this paper.

Modified and edge. The modified and the edge detection algorithms.

Once the image has been processed following one of the settings listed above, a decision algorithm has been applied to the produced binary map (*produced result*). The same decision algorithm (based on a fuzzy logic algorithm) has been applied irrespective of the setting used to obtain the binary image. Then, the *produced result* has been compared with the decision that humans would produce in those situations (*expected result*).

A match between the *produced result* and the *expected result* has been considered correct, whereas a discrepancy between the *produced result* and the *expected result* has been counted as an error. Six different possible outputs have been defined as the range of the possible

```
(1) for (int i = start Y; i < height; ++i)
(2) {
(3)                for (int j = start X; j < width; ++j)
(4)            {
(5)                unsigned char blue = pixel Data [i * step + j * channels];
(6)                unsigned char green = pixel Data [i * step + j * channels + 1];
(7)                unsigned char red = pixel Data [i * step + j * channels + 2];
(8)
(9)                if (red == 0 && green == 0 && blue == 0)
(10)               {
(11)                       continue;
(12)               }
(13)
(14)           //Find thersholds
(15)           if (red < mMinimumRed)
(16)           {
(17)                   mMinimumRed = Red;
(18)           }
(19)
(20)           if (red > mMaximumRed)
(21)           {
(22)                       mMaximumRed = Red;
(23)           }
(24)
(25)           int redGreenRange = red – green;
(26)           int redBlueRange = red – blue;
(27)
(28)           if (redGreenRange < mRedGreenRangeMin)
(29)           {
(30)                       mRedGreenRangeMin = redGreenRange;
(31)           }
(32)
(33)           if (redGreenRange > mRedGreenRangeMax)
(34)           {
(35)                   mRedGreenRangeMax = redGreenRange;
(36)           }
(37)
(38)           if (redBlueRange < mRedBlueRangeMin)
(39)           {
(40)                   mRedBlueRangeMin = redBlueRange;
(41)           }
(42)
(43)           if (redBlueRange > mRedBlueRangeMax)
(44)           {
(45)                       mRedBlueRangeMax = redBlueRange;
(46)           }
(47)       }
(48) }
```

Listing 1: Threshold. The code calculating the thresholds for the Image Segmentatation step.

```
(1) for (int i = 0; i < inImage − > height; ++i)
(2) {
(3)         for (int j = 0; j < inImage − > width; ++j)
(4)         {
(5)             int bluePos = i ∗ step + j ∗ channels;
(6)             int greenPos = i ∗ step + j ∗ channels + 1;
(7)             int redPos = i ∗ step + j ∗ channels + 2;
(8)
(9)             unsigned char red = inPixel Data [redPos];
(10)            unsigned char green = inPixel Data [greenPos];
(11)            unsigned char blue = inPixel Data [bluePos];
(12)            int redGreen = red − green;
(13)            int redBlue = red − blue;
(14)
(15)            if ((red > mMinimumRed && red < mMaximumRed)
(16)                && (redGreen >= mRedGreenRangeMin
(17)                    && redGreen <= mRedGreenRangeMax)
(18)               &&(redBlue >= mRedBlueRangeMin
(19)                    && redBlue <= mRedBlueRangeMax))
(20)            {
(21)                //pixel is within floor range set to white
(22)                outPixelData [redPos] = 255;
(23)                outPixelData [greenPos] = 255;
(24)                outPixelData [bluePos] = 255;
(25)                ++total Pixels;
(26)            }
(27)            else
(28)            {
(29)                outPixelData [redPos] = 0;
(30)                outPixelData [greenPos] = 0;
(31)                outPixelData [bluePos] = 0;
(32)            }
(33)        }
(34)}
```

LISTING 2: Image Segmentation. The code applying the thresholds.

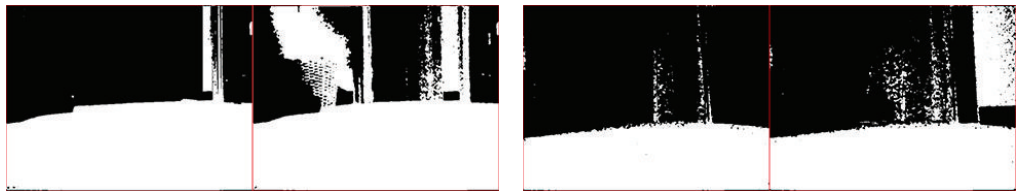
decisions. `move forward` or `turn left` are examples of some of the six *produced, expected results* output.

Results in Table 1 show that both the use of the modified algorithm and the two steps strategy are very significant ($\chi^2(3,\ N = 21) = 12.4,\ p = 0.006$). That is, when the performance of the modified red threshold rule is compared with the original rule in both settings (*original* versus *modified* and *original and edge* versus *modified and edge*) a higher correct success rate is obtained for the modified algorithm. Moreover, the discrepancy between *produced result* and *expected result* is reduced with the introduction of the *edge step* when the image is processed.

Different floor patterns have also been tested to investigate the modified algorithm under different conditions. Figure 6 shows the outcome of using, respectively, the *modified* in (a) and the *original* in (b) for a given floor pattern raw image in (c). From Figure 6 it is possible to conclude that the modified rule set seems to perform specifically strongly with white colours compared to the original rule set. Moreover, the modified rule set handles the nonuniform floor patterns better. That is, with the introduction of the $\alpha^{\max}$ value it is possible

```
(1) for (auto it = Filtered Lines. Begin (); it != Filtered Lines. end (); ++it)
(2) {
(3)          CvPoint poly Points [4];
(4)
(5)          //First point is to the left of the right point
(6)          if ((∗ it) [0] <= (∗ it) [2])
(7)          {
(8)                    polyPoints [0] = cvPoint ((∗ it) [0], 0);
(9)                    polyPoints [1] = cvPoint ((∗ it) [2], 0);
(10)                   polyPoints [2] = cvPoint ((∗ it) [2], (∗ it) [3]);
(11)                   polyPoints [3] = cvPoint ((∗ it) [0], (∗ it) [1]);
(12)         }
(13)         else //First point is to the right of the right point
(14)         {
(15)                  polyPoints [0] = cvPoint ((∗ it) [2], 0);
(16)                  polyPoints [1] = cvPoint ((∗ it) [0], 0);
(17)                  polyPoints [2] = cvPoint ((∗ it) [0], (∗ it) [1]);
(18)                  polyPoints [3] = cvPoint ((∗ it) [2], (∗ it) [3]);
(19)         }
(20)
(21)         cv Fill ConvexPoly (inImage, &polyPoints [0], 4, cvScalar (0, 0, 0));
(22) }
```

LISTING 3: Edge Segmentation. The code implementing the Edge Segmentation algorithm.



(a) (b)

Figure 4: Comparison of applied edge supplementation to colour segmentation map. In both (a) and (b), left image applied edge supplementation, right image nonapplied.

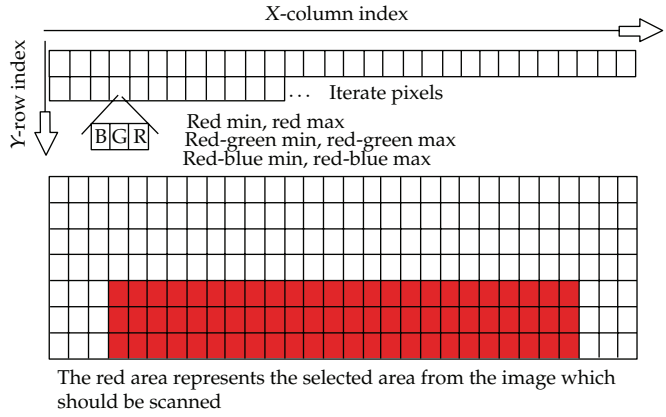


Figure 5: Selecting area for colour thresholds.

Table 1: Algorithm testing. Correct matching results between the *produced result* and the *expected result* for the different settings.

Setting	Success rate (%)
Original	42
Original and Edge	76
Modified	57
Modified and Edge	90

Table 2: Algorithm processing time. Processing time values (in ms) for the *image segmentation* and *edge detection* settings.

Setting	Processing time (ms)
Modified	25.6
Modified and Edge	27.6

to notice that the algorithm performs particularly strongly against white colours compared to the original algorithm (see, e.g., Figure 6). The original algorithm would have a much larger threshold, based on our tests the original algorithm has a threshold range 38.75% greater than that of the modified algorithm under highly illuminated environments.

Table 2 shows the processing time for the different settings described at the beginning of the section. The time difference between the image segmentation algorithm [20] (i.e., *original*) and the modified version presented in this paper (*modified*) is too negligible to show in the results, therefore only results for the *Modified* configuration have been shown.

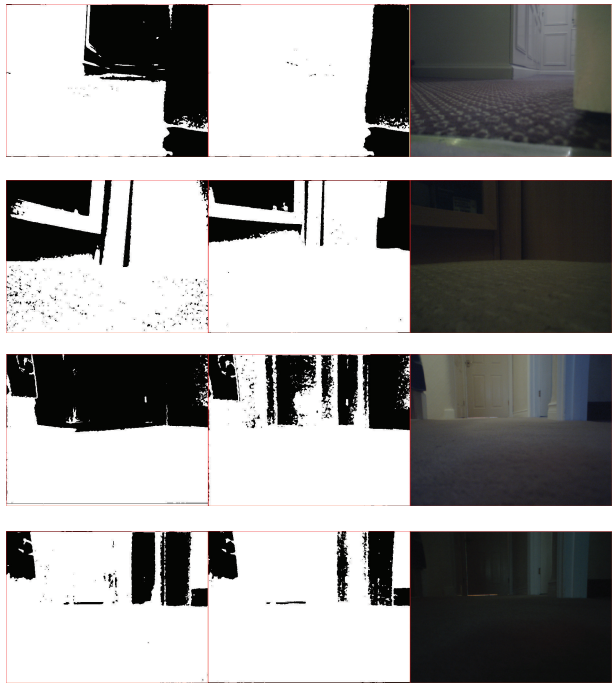
The CPU used for all tests was a *Phenom II X4 955* and CPU clock was set to 3.6 GHz. The time without edge detection step and the time with the edge detection step has been measured, respectively, in row 1 and 2. The values in the table indicate the time in milliseconds it took to apply the algorithms indicated in the *Setting* column to the image and then to generate the binary collision map. In all cases the mean was taken from a sample of 10 measurements.

Results in Table 2 show that the modified algorithm does not increase the processing time (since as stated above *modified* and *original* settings produce similar processing time). Further, it shows a percentage increase of 7.8% over the no edge configuration. That is, as expected, the time to extract the required information increases by including the edge detection step in the algorithm. However, from Table 2 it is possible to observe that a percentage increase of 7.8% in processing time has produced a percentage increase of 57.9% in the success rate for the algorithm.

4. Discussions and Conclusions

The sequential use of *colour image segmentation* and then *edge detection* strategies has been investigated. A novel color image segmentation algorithm and a probabilistic Hough algorithm have been implemented and tested. The novelty introduced here has been demonstrated to improve an existing algorithm for image processing.

In particular, the modified red threshold rule considered for the image segmentation algorithm helped in keeping the learnt thresholds stable. Moreover, the use of both colour segmentation and edge detection techniques complemented each other by removing the weaknesses that each method separately presented. In particular, with the colour



(a) (b) (c)

Figure 6: Comparisons between the 2 different image segmentation algorithms under different floor patterns. Columns (a) and (b) show the outcome of the processing for the raw image in (c).

segmentation technique identifying the main obstacle-free area and the edge detection filling any remaining gaps in the detected segments in the output binary map.

Although the approach discussed in this paper has been demonstrated to be independent of floor changes, further development is needed to cope with patterned floor surfaces. This is because the edges detected from the patterns and the wider colour thresholds that are learnt from such floors make it difficult to produce an accurate binary map. A possible solution could be to make use of the canny edge map to identify the patterned areas in the floor surface and combine that with the information from the threshold learning algorithm so that it would ignore colours within that area of the image. As a consequence the learnt colour thresholds would be narrower and would not erroneously detect obstacles as obstacle free areas.

The scalability of this algorithm for a distributed architecture (with several robots involved) is another aspect that could be investigate further. Each robot will produce a (slight) different image of the same environment to extract the required features. Therefore each robot can receive not only its own information but also the information from its neighboring robot according to the topology of the given robot network. Then to deal with the complicated coupling between one sensor and its neighbors, a filtering approach such as the ones in [24, 25] could be considered. However, some further investigation is required to analyse how these paradigms would perform with these types of data.

References

[1] H. Dong, Z. Wang, and H. Gao, "Distributed filtering for a class of time-varying systems over sensor networks with quantization errors and successive packet dropouts," *IEEE Transactions on Signal Processing*, vol. 60, no. 6, Article ID Article number6168290, pp. 3164–3173, 2012.

[2] Z. Wang, B. Shen, H. Shu, and G. Wei, "Quantized H_∞ control for nonlinear stochastic time-delay systems with missing measurements," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, Article ID Article number6082385, pp. 1431–1444, 2012.

[3] J. Esteban, A. Starr, R. Willetts, P. Hannah, and P. Bryanston-Cross, "A review of data fusion models and architectures: towards engineering guidelines," *Neural Computing and Applications*, vol. 14, no. 4, pp. 273–281, 2005.

[4] A. Saffiotti, "The uses of fuzzy logic in autonomous robot navigation," *Soft Computing*, vol. 1, no. 4, pp. 180–197, 1997, http://aass.oru.se/~asaffio/.

[5] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 2, pp. 237–267, 2002.

[6] A. M. Zou, Z. G. Hou, M. Tan, and D. Liu, "Vision-guided mobile robot navigation," in *Proceedings of the IEEE International Conference on Networking, Sensing and Control (ICNSC '06)*, pp. 209–213, April 2006.

[7] W. Shi and J. Samarabandu, "Corridor line detection for vision based indoor robot navigation," in *Proceedings of the Canadian Conference on Electrical and Computer Engineering (CCECE '06)*, pp. 1988–1991, May 2006.

[8] N. Ohnishi and A. Imiya, "Dominant plane detection from optical flow for robot navigation," *Pattern Recognition Letters*, vol. 27, no. 9, pp. 1009–1021, 2006.

[9] K. Van Workum and R. Green, "Smart wheelchair guidance using optical flow," in *Proceedings of the 24th International Conference Image and Vision Computing New Zealand (IVCNZ '09)*, pp. 7–11, November 2009.

[10] N. Senthilkumaran and R. Rajesh, "Edge detection techniques for image segmentation a survey of soft computing approaches," *International Journal of Recent Trends in Engineering*, vol. 1, no. 2, pp. 844–846, 2009.

[11] S. Al-amri, N. V. Kalyankar, and S. D. Khamitkar, "Image segmentation by using edge detection," *International Journal on Computer Science and Engineering*, vol. 2, no. 3, pp. 804–807, 2010.

[12] L. Lucchese and S. K. Mitra, "Color image segmentation : a state-of-the-art survey," *Citeseer*, vol. 67, no. 2, pp. 207–221, 2001.

[13] R. M. Haralick and L. G. Shapiro, "Image segmentation techniques," *Computer Vision, Graphics, & Image Processing*, vol. 29, no. 1, pp. 100–132, 1985.

[14] R. C. Gonzales and R. Woods, *Digital Image Processing*, Addison-Wesley, Reading, Mass, USA, 1992.

[15] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[16] H. D. Cheng, X. H. Jiang, Y. Sun, and J. Wang, "Color image segmentation: advances and prospects," *Pattern Recognition*, vol. 34, no. 12, pp. 2259–2281, 2001.

[17] Y. Yang and B. Wu, "A new and fast multiphase image segmentation model for color images," *Mathematical Problems in Engineering*, vol. 2012, Article ID 494761, 20 pages, 2012.

[18] S. Varun, S. Singh, R. Sanjeev Kunte, R. D. Sudhaker Samuel, and B. Philip, "A road traffic signal recognition system based on template matching employing tree classifier," in *Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA '07)*, pp. 360–365, December 2007.

[19] V. Andrey and K. H. Jo, "Automatic detection and recognition of traffic signs using geometric structure analysis," in *Proceedings of the International Joint Conference (SICE-ICASE '06)*, pp. 1451–1456, October 2006.

[20] C. Lin, C. H. Su, H. S. Huang, and K. C. Fan, "Colour image segmentation in various illumination circumstances," in *Proceedings of the 9th WSEAS International Conference on Circuits, Systems, Electronics, Control and Signal Processing (CSECS '10)*, pp. 179–184, Stevens Point, Wis, USA, December 2010.

[21] E. Nadernejad, S. Sharifzadeh, and H. Hassanpour, "Edge detection techniques: evaluations and comparisons," *Applied Mathematical Sciences*, vol. 2, no. 29–32, pp. 1507–1520, 2008.

[22] D. Ziou and S. Tabbone, "Edge detection techniques - an overview," *International Journal of Pattern Recognition and Image Analysis*, vol. 8, pp. 537–559, 1998.

[23] R. Laganiére, *OpenCV 2 Computer Vision Application Programming Cookbook*, Packt, 2001.

[24] Z. Wang, B. Shen, and X. Liu, "H_∞ filtering with randomly occurring sensor saturations and missing measurements," *Automatica*, vol. 48, no. 3, pp. 556–562, 2012.

[25] B. Shen, Z. Wang, and X. Liu, "A stochastic sampled-data approach to distributed H_∞ filtering in sensor networks," *IEEE Transactions on Circuits and Systems. I. Regular Papers*, vol. 58, no. 9, pp. 2237–2246, 2011.