Visual Monocular Obstacle Avoidance for Small Unmanned Vehicles

Levente Kovács
MTA SZTAKI - Institute for Computer Science and Control
Hungarian Academy of Sciences, Kende u. 13-17, Budapest, Hungary
levente.kovacs@sztaki.mta.hu

Abstract

This paper presents and extensively evaluates a visual obstacle avoidance method using frames of a single camera, intended for application on small devices (ground or aerial robots or even smartphones). It is based on image region classification using so called relative focus maps, it does not require a priori training, and it is applicable in both indoor and outdoor environments, which we demonstrate through evaluations using both simulated and real data.

1. Introduction

In this paper we present and evaluate a monocular obstacle avoidance method, with the intended use on small unmanned vehicles that are limited in the weight of the payload they can carry, or by some other restriction (power, length of flight, computation capabilities, etc.). The main goal is to produce a usable solution that does not require extensive a priori training and the gathering of large training datasets, and can be easily deployed on various micro robot vehicles for indoor or outdoor use. The simulated and real evaluations will show that the proposed approach produces better detections and less false alarms than other methods, resulting in a lower number of collisions.

For outdoor environments, in [11] a single camera obstacle avoidance method was introduced for unmanned ground vehicles using supervised learning to learn depth cues, dividing the images in stripes labeled according to their relative distance and using texture information, followed later by [9], where single image based obstacle avoidance was presented for unmanned aerial vehicles, using Markov Random Field classification modeling the obstacles using color and texture features, training the model for obstacle classes with labeled images. In [2] a visual navigation solution was described, following a path of images acquired in a training phase, avoiding new obstacles using the camera and a range scanner. In [4] a monocular visual approach was presented for recognizing forest trails and navigating a quadrotor micro aerial vehicle in such an environment. The authors use a Deep Learning approach to recognize trail directions, and also provide a large training dataset. For water-based vehicles, in [3] an autonomous watercraft obstacle avoidance approach was presented using a single camera, extracting optical flow to detect and track potential obstacles, based on an occupancy grid approach (using GPS and inertial sensors).

For indoor environments, in [12] a navigation framework was presented using a single image for detecting stationary objects and ultrasonic sensing to detect moving objects, using the difference between the current and expected image for detecting stationary obstacles. [8] used low resolution color segmentation and object detection (trained for 8 object classes) for single camera obstacle avoidance. In [14] obstacle avoidance was created using low resolution images (for color segmentation) to find ground objects and a sonar sensor for extracting depth information, while in [13] indoor obstacle avoidance was produced using optical flow extracted from image series (looking to balance left-right flow quantity) for finding objects and estimating depth.

Although several methods exist that start from some sort of extracted feature map of the captured image, e.g. texture- or color-based segmentation, optical flow-based field segmentation and stationary object detection, they are either not suitable for conveying a sort of relative depth-like information (as the method that is the basis of the current approach does), or are simply not suited for situations where the vehicle (or the camera) is allowed to move freely. E.g., optical flow based methods can be usable in indoor scenarios on a ground-based robot for detecting stationary targets, but they will not work on freely moving cameras, outdoors, in an environment where not all obstacles are stationary. Other approaches that work by feeding datasets specific to a certain application scenario (parks, forests, etc.) into a machine learning framework have the drawback of only working in those specific use cases.

Compared to the above methods, the approach of this paper also uses a single camera - without additional sensor inputs - to detect obstacles to be avoided. However, this method does not require any a priori training either for in-

door or outdoor scenarios, yet it is applicable in both environments. Since it does not target a specific scenario, it can be more versatile than other approaches, easier to deploy, and as it will be shown later, it can perform at acceptable speeds to be usable even on lower capability vehicles or devices. The approach is based on the region of interest extraction method in [7], which uses local blind deconvolution as a means to classify image regions relative to each other (for a short overview see Sec. 2.1 below), producing a feature map that inherently includes localized structural information of the processed images. As we will show later, we will use these feature maps as the basis for the detection and avoidance of possible obstacles. The proposal of using this region extraction method for salient region detection with the possibility of using it for obstacle avoidance was first introduced in [6].

The main contributions of the present paper are the development of a usable and deployable obstacle avoidance method for various platforms (PC, Android smartphones and as ROS - the Robot Operating System `http://www.ros.org` - nodes), and the extensive quantitative evaluations performed in both simulated and real life scenarios.

2. Visual Monocular Obstacle Avoidance

For detecting obstacle regions that we try to avoid, we take the following steps:

1. Capture a current camera frame.
2. Downsize to 320 pixel column width (for faster computation).
3. Extract the feature map.
4. Find the parts of the map - if there are any - that indicate no obstacles.
5. Propose a movement direction towards the found region, or stop.

2.1. Feature map extraction

In [7] the authors describe a local deconvolution based region classification approach, that has been originally introduced for detecting focused image regions, but shown to be usable for segmenting differently textured regions and to produce saliency-like segmentations. The reason for such capabilities is that the employed localized deconvolution process combined with the inclusion of local contrast information produces a feature map that indirectly includes important local structure information (i.e., sharpness, gradients, variance, etc.). We base our obstacle avoidance approach on the latter properties of this method, using the detected regions and the produced maps that generate a weighted map of these regions as a base or processing for trying to avoid them during navigation.

If $g(r) = f(r) * h(r)$, where g is the captured frame, $f(r)$ is a region (r) of the unknown original input image f that is convolved with a h unknown blurring function, then we can get an estimate for f in an iterative deconvolution process $f_{k+1}(r) = f_k(r)[h_k(r) * g/g_k(r)]$ (with the estimation of $h_{k+1}(r)$ having a similar form). Then, after some iterations, a local reconstruction error can be calculated for the image regions using the values of the last iteration as

$$E(g(r), g_k(r)) = |arc\ sin \frac{< g(r) - g_k(r), g(r) >}{|g(r) - g_k(r)| \cdot |g(r)|}| \cdot C_r(g_r), \quad (1)$$

where $C_r(g_r)$ is an optional local contrast term. The obtained error values are then used to classify the image regions relative to each other and produce a feature map.

We will call the generated feature map of the above approach as Dmap for the remainder of this paper. Algorithm 1 shows the main steps of the method, while the second row of Fig. 1 shows some example outputs.

Algorithm 1: Main steps of Dmap generation.

Input: g_c: captured and resized camera frame
Output: d: generated feature map
Initialization:
- w← block size (e.g., 16, 32);
- s← step size (e.g., $w/2$);
- i← iteration count (e.g., $10-20$);
- Convert g_c to grayscale (g);

foreach *$w \times w$ region of g* **do**
  Initialization:
  - initialize f_0 as a $w \times w$ uniform valued image (with the mean of the region of g);
  - initialize h_0 as circular constant unity;

  for *i iterations* **do**
    estimate f_{k+1}, update h_{k+1};
  end
  Calculate reconstruction error (Eq. 1);
  Step to the next $w \times w$ region.
end
Classify the regions, generate d feature map.

2.2. Detecting obstacle regions

For extracting the above feature map, we run the deconvolution process on 32×32 image blocks that overlap by half of their size, for 10 iterations, and map the obtained local errors linearly into a $0-255$ grayscale image (M) with the same size as the downscaled captured frame. Then, we go over the obtained map and try to find a region R_m with an minimal overall intensity. That is, if B_u, $u \in \mathbb{Z}$ are non-overlapping blocks of M, and R_u, $u \in \mathbb{Z}$, are the sums of intensities below a threshold ($20-50\%$) in these blocks: $R_u = \sum_i (B_u(i) | B_u(i) < \varepsilon)$, then we are looking for the region $R_m = min_u R_u$. If we partition the M map into 3×3 regions and find the minimal R_m among them (if it exists), then we can propose a movement direction towards

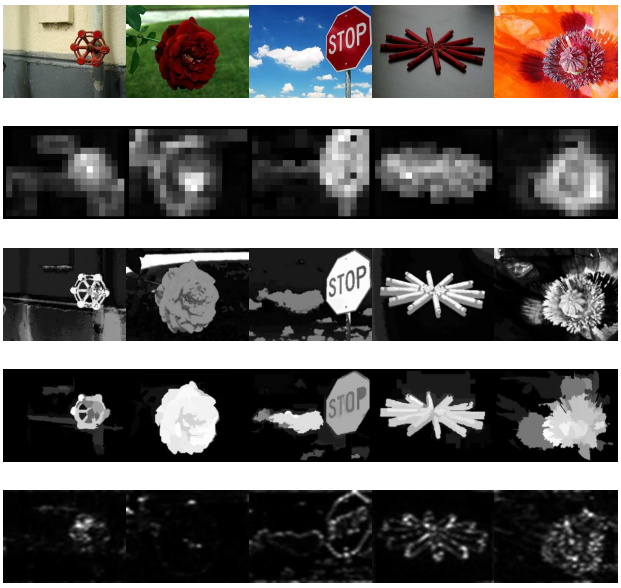


Figure 1. First row: images from the MSRA dataset. Then, from second to fifth rows: Dmap, HC, RC, SR outputs.

that region - i.e., NW, N, NE, E, SE, S, SW, W or Forward. "Forward" stands for going straight ahead, and we do not use a reverse (or backing up) movement since going in this directions can be realized by some turns and then going forward (also, this is a generally accepted behaviour, e.g., `http://wiki.ros.org/move_base`). If there is no R_m region satisfying the above conditions, stopping is proposed, and to look for a way out by turning in some direction. Situations when such a region does not exist are when either there is no intensity on the M map below the threshold, or if there are such regions but their size is too small (in this case smaller than 5% of the frame size).

When such a region R_m does not exist, that does not always mean there are only obstacles in the field of view of the camera - since false detections can occur, as we will show in Sec. 3. However, during navigation our goal is to avoid collisions, thus in such situations we limit the robots' movements to turns. We will describe how robot vehicles could move according to the proposed directions in the following section.

Several approaches exist that have the goal of detecting regions of interest in images (i.e., salient regions). For the purpose of real life comparisons and evaluations, we selected recent methods that have usable and available sources and could be ported to different platforms. The chosen methods are the well-performing Histogram Based Contrast (HC) and the Region Based Contrast (RC) methods of [1], and the Spectral Residual (SR) method of [5] (for a detailed evaluation of these methods from a saliency perspective please visit `http://mmcheng.net/salobj`). Some examples of extracted regions using these methods are shown in Fig. 1, over images from the Microsoft Research MSRA Salient Object Database [10] (where the method used in this paper is labeled as Dmap).

3. Evaluation

For evaluations, several versions of the Dmap, HC, RC and SR methods were produced for PC (Windows and Linux), Android and ROS. The ports were created starting from M.-M. Cheng's saliency algorithm C++ sources available at `https://github.com/MingMingCheng/CmCode` for educational and research use. The subset of these sources containing the HC, RC and SR methods that we ported to Linux, Android and ROS are available online (`http://web.eee.sztaki.hu/~kla/evw16.html`). For performance evaluations a ROS implementation was used running in Gazebo (`http://gazebosim.org`) based virtual environments with simulated robots, an Android implementation was used to perform evaluations in real environments (indoors and outdoors) by a human holding the camera - i.e., replacing the robot in this case. For computational time evaluations a PC (Intel Core i7 930 quad-core 2.80 GHz), a 3DR Iris drone (`https://3dr.com/kb/2014-iris`) with an ODROID-XU board with Cortex-A7 quad-core CPU at 1.2GHz and Android devices (Samsung SM-N9905 with Qualcomm Krait 400/MSM 8974 quad-core 2.3 GHz, HTC One X with ARM Cortex-A9 quad-core 1.5 GHz) were used. These devices will have the labels PC, Iris, And1, And2 in this section.

For the ROS implementation, first a simulation was created with a TurtleBot (with a Kobuki base with a simulated camera with 60 degree field of view, 640x480 resolution, and clipping at 0.05 and 18.0 distances). The basis of the simulated world was the MathWorks 3-room office world from their "Virtual Machine with ROS Hydro and Gazebo for Robotics System Toolbox". An example overview shot of the modified world is shown in Fig. 3. The methods were implemented as ROS nodes (and tested on ROS Hydro and Indigo releases), one node for processing the camera frames, and another for moving the robot based on the obtained movement direction proposals. A similar approach has been followed for a quadrotor-based simulation, a view of which is shown in Fig. 4.

Fig. 2 shows the diagram of the TurtleBot-based processing node-chain in ROS, for taking the images of the bot's camera, processing the images, generating a motion direction and sending the movement commands back to the bot. The quadrotor-based simulation is very similar (with differences in the camera source and the movement commands sent to the simulated robot).

Fig. 3 shows a view of the simulated environment for the TurtleBot-based evaluation. Some stationary obstacles have been added on the floor for this simulation, which generate collision events when the bot runs into them. Fig. 4 shows a

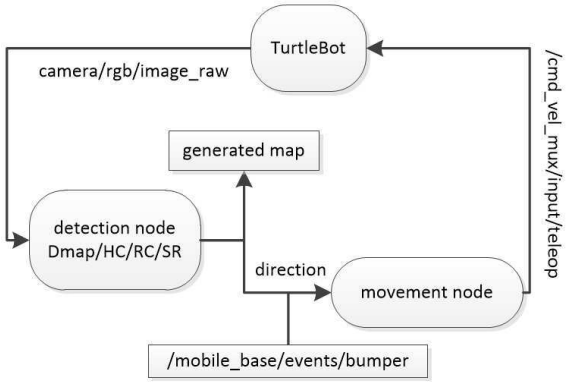


Figure 2. The diagram of the TurtleBot-based ROS simulation node-chain for processing bot camera images, generating a movement direction, and sending movement commands to the bot.

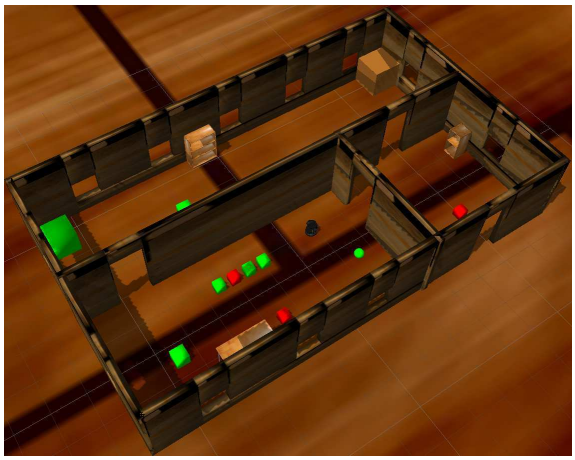


Figure 3. A view of the simulated world for the ROS TurtleBot tests. The bottom room is labeled *room1*, the upper left *room2* and the upper right *room3*.

view of the simulated environment for the quadrotor-based evaluation, where some of the ground obstacles have been removed, and high pillar-like structures were added instead, which also generate collision events when the quadrotor hits them. Fig. 5 shows some example camera views from the two simulations.

For the simulation tests with the TurtleBot, the bot was positioned in one of the rooms (labeled as *room1*, *room2* and *room3*) with the same starting position for all the methods, and was allowed to roam freely for approximately 500 steps. The quadrotor vehicle was tested in *room1*, always placed at the same starting position and height, and was allowed to move around freely. The primary goal of these tests was to see if the methods can drive the bots around by avoiding as many collisions as possible. The point is, that if we have a method that can avoid obstacles without a priori training and in various environments, then it can be integrated into a full navigation chain and be the basis of a

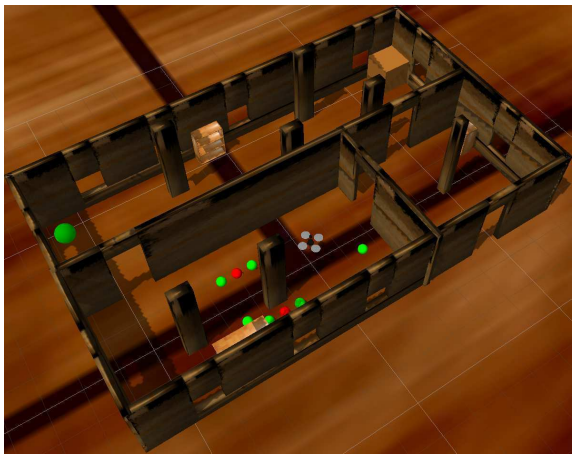
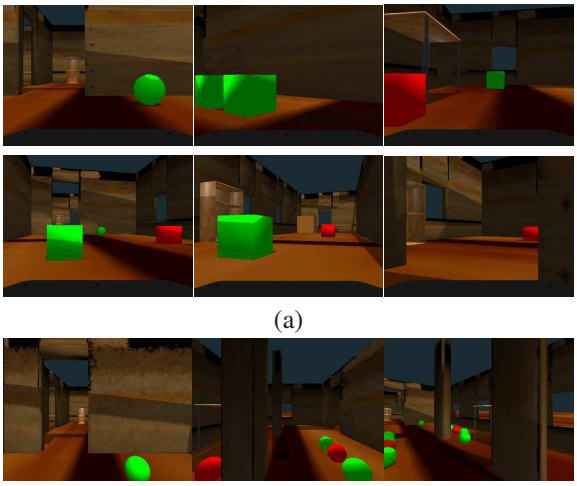


Figure 4. A view of the simulated world for the ROS quadrotor tests.



(a)

(b)

Figure 5. (a) Sample frames from the TurtleBot simulation. (b) Sample frames from the quadrotor simulation.

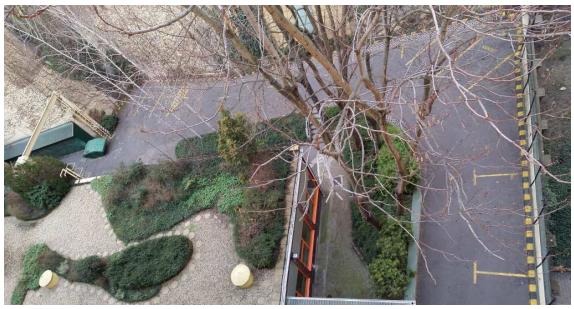
full fledged goal-oriented movement control process.

The quadrotor vehicle was allowed to move in all possible directions, i.e. up-left, up, up-right, right, down-right, down, down-left, left, forward and stop (directions denoted by NW, N, NE, E, SE, S, SW, W, FWD, STOP). In the case of the TurtleBot the following movement rules were applied:

- If the proposed direction is Forward: move forward.
- If the proposed direction is STOP: turn left 30 degrees.
- If the proposed direction is W or E: turn left or right 15 degrees.
- Treat all other proposed directions (NW, NE, N, SW, SE, S) as STOP.

For both simulated vehicles, if a real bump/obstacle hit is signaled by the on-board sensors (bumpers, LIDAR, altitude sensor), we treat it as a STOP signal, but count it as an obstacle hit.

(a)



(b)

Figure 6. (a) The indoor test environment. (b) An top view of the outdoor environment.

To evaluate performance, obstacle hits, or bumps, were detected by using simulated sensors. For the TurtleBot 3 bumpers (left, center, right) were used to detect hits (by listening for events on `/mobile_base/events/bumper`). For the quadrotor a simulated Hokuyo LIDAR sensor (180 degree field of view, 1800 simulated beams) and the z-axis data from `/ground_truth_to_tf/pose` were used to detect horizontal bumps or vertical hits (when descending to hit the ground). In all cases we counted the number and type of actual moves, and the number of sensor-detected collisions.

Real tests were performed with an Android-based smartphone, indoors in a large room with several obstacles, and outdoors in a backyard parking lot with obstacles and vegetation. Fig. 6 shows an overall view of the indoor (a) and outdoor (b) environments used for the real tests, while Figs. 7 and 8 show some example frames from these environments. In these environments the tests were performed by holding an Android device in hand at approximately 1.5m height, and moving/turning in the directions indicated by the method running on the device. All indoor/outdoor tests were performed by picking a method and starting from the same initial position and performing 200 moves. In the indoor/outdoor tests collisions were counted manually by the human holding the device.

3.1. Results

For evaluating the performance of the method and compare it with the others, we concentrated on how well the methods can avoid collisions (i.e., how often do they hit an obstacle in the same environment, during the same time period), what is the ratio of "good" stop signals vs. all stops,

Figure 7. Sample frames from the indoor environment.

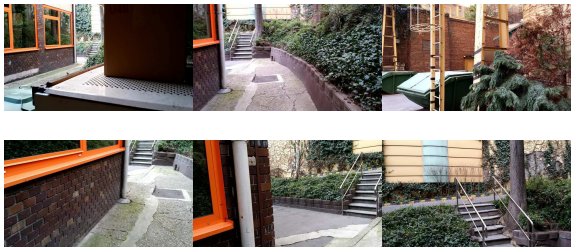


Figure 8. Sample frames from the outdoor environment.

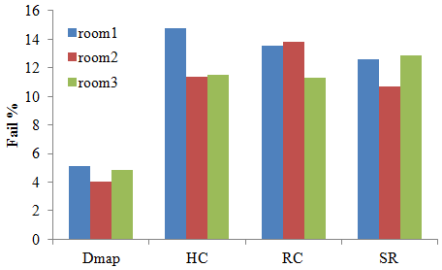


Figure 9. Percentage of hitting obstacles from all the performed moves of the TurtleBot in the 3 rooms of the simulated environment, for all methods.

including collisions (which can indicate whether a vehicle is actually trying to avoid obstacles, or is just moving around blindly in straight lines until hitting something).

Fig. 9 shows performance data of the Dmap method and the compared methods in the rooms of the TurtleBot simulated environment, specifically the ratio of obstacle hitting moves over all the performed moves. This figure clearly shows that the Dmap method hits obstacles less frequently, averaging around 4-5%.

Fig. 10 shows the percentage of actual stop signals for the TurtleBot simulation - i.e., when the respective algorithm indicated the bot should stop because of a detected obstacle - from all the stops, including hitting obstacles. What this figure shows is that in the case of Dmap-based movements a much higher ratio of stop signals is actually indicated by the method. In the case of the RC method none of its stops have been actually algorithm-indicated (i.e., they were either collisions, or directions handled as stops - N, S, etc.).

Fig. 11 shows, as an example, the distribution of bot

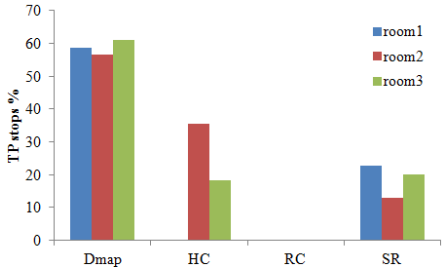


Figure 10. Percentage of real (algorithm-signaled) STOP signals from all the stops (including obstacle hits) of the TurtleBot for all simulated environments and methods.

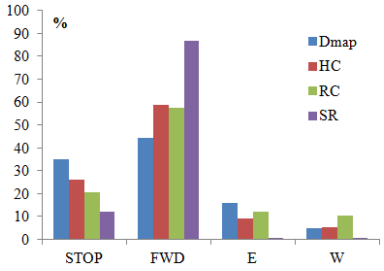


Figure 11. Distributions (in percents) of TurtleBot movement directions in the *room3* segment of the simulated environment.

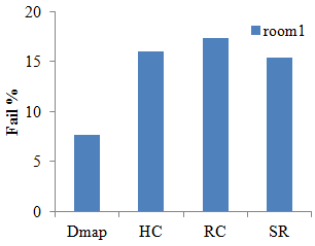


Figure 12. Percentage of hitting obstacles from all the performed moves of the quadrotor in *room1* of the simulated environment, for all methods.

movement directions in one of the rooms of the simulation. Here, the columns of the STOP signal encompass all other signaled directions that are treated as stops for the TurtleBot (i.e., NW, N, NE, SW, S, SE).

The same evaluations were performed for the quadrotor vehicle. Fig. 12 shows performance data of the Dmap method and the compared methods in *room1* of the quadrotor simulated environment, specifically the ratio of obstacle hitting moves over all the performed moves. While in this case the Dmap-based approach performs somewhat worse than on the TurtleBot, it is still more than twice better than the other methods.

Fig. 13 shows the percentage of generated stop signals for the quadrotor simulation - i.e., when the respective algorithm indicated the bot should stop because of a detected obstacle - from all the stops, including obstacle collisions. In this case, since the robot is allowed to move in every possible directions and only algorithm-indicated stops and collisions are treated as stops, this figure shows that the HC and the SR methods never noticed obstacles and only stopped when colliding with something.

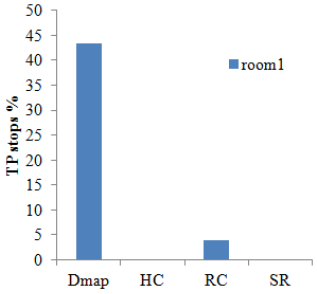


Figure 13. Percentage of real (algorithm-indicated) STOP signals from all the stops (including obstacle hits) of the quadrotor for *room1* of the simulated environment and all the methods.

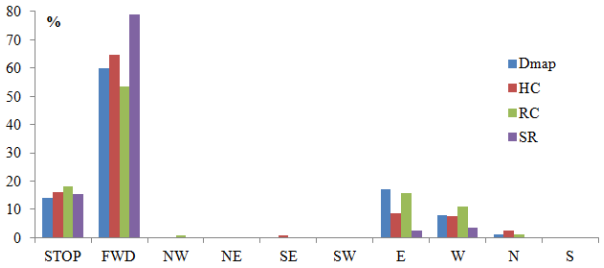


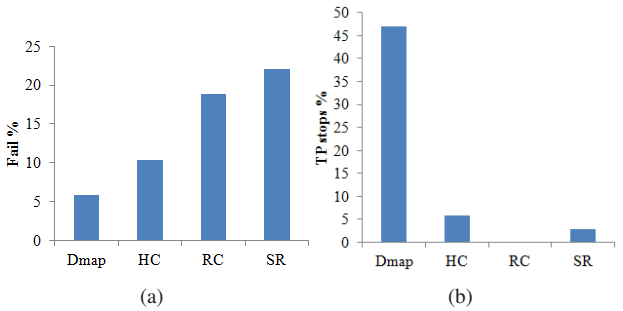
Figure 14. Distributions (in percents) of quadrotor movement directions in the *room1* of the simulated environment.

Fig. 14 shows, the distribution of quadrotor movement directions in *room1* of the simulation. As we can see in the STOP columns of the methods, all methods did stop at several occasions, but if we also look at Figs. 12 and 13 we can see that only Dmap and RC did actually detect obstacles and that the Dmap method's performance was clearly better.

Following the ROS/Gazebo simulations, we also performed real world evaluations in an indoor (office space with placed obstacles) and an outdoor environment as shown in Fig. 6. Figs. 7 and 8 show some example frames that the device captured during the process.

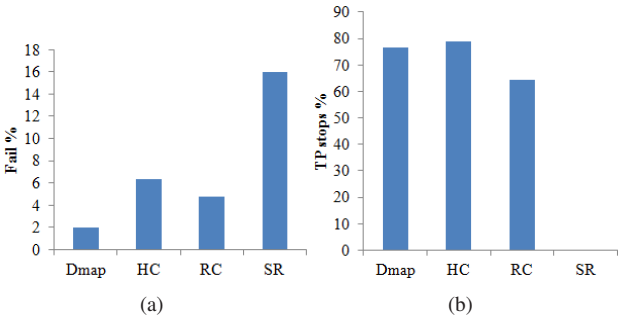
In these cases we performed the same measurements as in the case of the simulations. Fig. 15 (a) shows performance data of the Dmap and the compared methods in the indoor real environment, specifically the ratio of obstacle hitting moves over all the performed moves. Fig. 15 (b) shows the percentage of real STOP signals for the indoors environment - i.e., when the respective algorithm indicated the bot should stop because of a detected obstacle - from all the stops, including obstacle hits.

Fig. 16 (a) shows performance data of the Dmap-based approach and the compared methods in the outdoor real environment, specifically the ratio of obstacle hitting moves over all the performed moves. Fig. 16 (b) shows the percentage of algorithm-indicated STOP signals for the out-



(a) (b)

Figure 15. (a) Percentage of hitting obstacles from all the performed moves in the indoor environment, for all methods. (b) Percentage of real (algorithm-signaled) STOP signals from all the stops (including obstacle collisions) of the indoor environment and all the methods.



(a) (b)

Figure 16. (a) Percentage of hitting obstacles from all the performed moves in the outdoor environment, for all methods. (b) Percentage of real (algorithm-signaled) STOP signals from all the stops (including obstacle hits) of the outdoor environment and all the methods.

door environment - i.e., when the respective algorithm indicated the bot should stop because of a detected obstacle - from all the stops, including obstacle hits.

As the result figures show, all the methods perform better in an outdoor environment, which is especially true for the Dmap approach, since the method is better suited to run on images with varying textures than in environments with large homogeneous surfaces. also, the indoor and outdoor tests still support the usability and higher performance of the Dmap-based approach.

Besides the collision-based measurements, we also tried to visualize the paths the device followed by adhering to the movement directions indicated by the different methods. The point of this visualization is to get an impression about the overall capabilities of the methods, where we expect a better performing method to not get stuck in a small area as opposed to lesser performing ones which might just bounce around from obstacle to obstacle. Fig. 17 shows a visual approximation (created by hand) of the moves performed according to the respective compared methods in the outdoor environment. When creating these figures, the

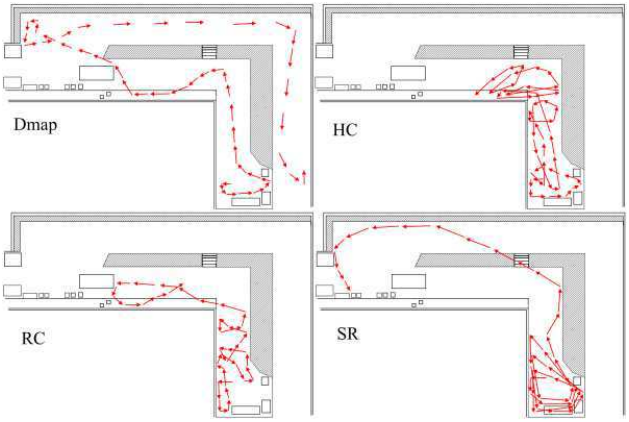


Figure 17. Visual approximations of the movement directions and covered areas in the outdoor environment by all compared methods.

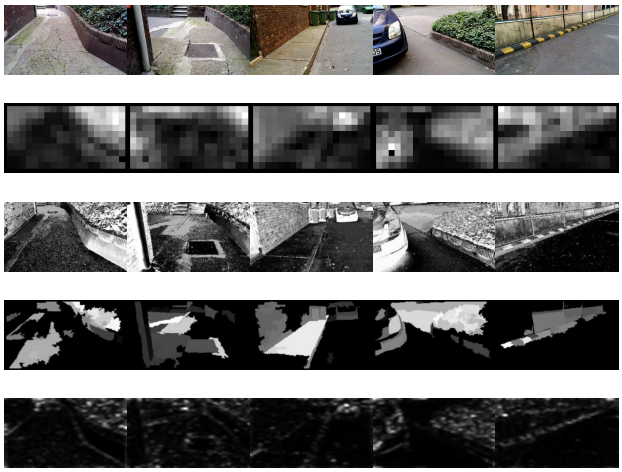
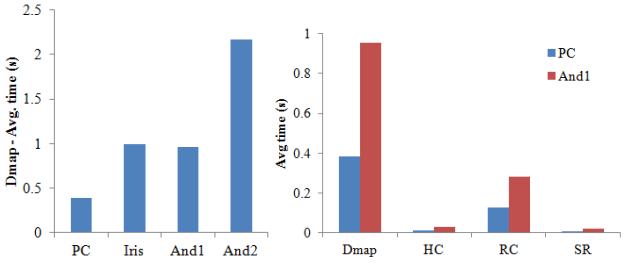


Figure 18. Example input frames (top row) from the outdoors tests, and generated feature maps by the Dmap, HC, RC and SR methods (second, third, fourth and fifth rows respectively).

movements were tracked by viewing the logged frames and indicated movement directions and placing an arrow on the approximate map of the environment to show the respective movements. The visualizations support the previous numerical evaluations in that the Dmap-based approach can be a better basis for integration into a goal-guided navigation framework. The figures show that some algorithms follow a more straight path in obstacle free-regions, while others make turns - the latter is a direct cause of occasional false detections of the respective methods (however, for our purposes false detections are still better than false negatives, which result in collisions). As a visual example, Fig. 18 shows some example input frames from the outdoors environment and the raw generated feature maps from all the methods for these input frames.

As a final step of the evaluation process, Fig. 19 shows average processing times for Dmap and the compared meth-



(a) (b)

Figure 19. (a) Average processing times for the Dmap method on a PC, the Iris drone and two Android devices (described in the text). (b) Average times for all methods on the PC and the And1 Android device.

ods. First in Fig. 19 (a) the times of the Dmap approach are shown on the above described devices. Since the run times on the And1 device and the Iris drone were similar, and since the And2 device was much slower, in Fig. 19 (b) we only compared the methods on the PC and on the And1 device as a reference. What these results indicate is that although the Dmap-based method is slower than the others, given its better overall performance it still could be preferred. Especially so, since it still can run with 1+ frames per second on a 3 years old smartphone and a 2 year-old ODROID-XU board. The desired processing speed on robots is dependent on the targeted application and the movement speed of the robot, but our general goal is to achieve 5-10 frames per second processing times with further future improvements. Both the ROS and Android implementations of Dmap support OpenMP, thus we expect significant time improvements on newer multi-core devices.

4. Conclusions

In this paper we extensively evaluated a single camera/image based obstacle avoidance method that does not require extensive a priori training, can be used in various environments, can be easily ported and deployed on different platforms, is able to perform at practical speeds and has a low enough collision ratio that supports its usability in real life situations. Among our future plans is the implementation of the method - with the possibility of fusing the feature map with other image features - as an integral part of a navigation framework, for either supporting autonomous navigation in a goal-oriented scenario, or free browsing navigation for area surveillance or visual odometry and mapping applications.

Acknowledgements

This work has been partially supported by the Bosch "ERNYŐ_13" project.

References

[1] M.-M. Cheng, N. J. Mitra, X. Huang, P. H. S. Torr, and S.-M. Hu. Global contrast based salient region detection. *IEEE TPAMI*, 37(3):569–582, 2015.

[2] A. Cherubini and F. Chaumette. Visual navigation with obstacle avoidance. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 1503–1598, 2011.

[3] T. El-Gaaly, C. Tomaszewski, A. Valada, P. Velagapudi, B. Kannan, and P. Scerri. Visual obstacle avoidance for autonomous watercraft using smartphones. In *Proc. of Autonomous Robots and Multirobot Systems workshop (ARMS)*, 2013.

[4] A. Giusti, J. Guzzi, D. Ciresan, F.-L. He, J. P. Rodriguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, PP:1–7, 2015.

[5] X. Hou and L. Zhang. Saliency detection: A spectral residual approach. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[6] L. Kovács. Single image visual obstacle avoidance for low power mobile sensing. In *Proc. of Advanced Concepts for Intelligent Vision Systems (ACIVS)*, pages 261–272, 2015.

[7] L. Kovács and T. Szirányi. Focus area extraction by blind deconvolution for defining regions of interest. *IEEE Tr. on Pattern Analysis and Machine Intelligence*, 29(6):1080–1085, 2007.

[8] S. Lenser and M. Veloso. Visual sonar: Fast obstacle avoidance using monocular vision. In *Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2013.

[9] I. Lenz, M. Gemici, and A. Saxena. Low-power parallel algorithms for single image based obstacle avoidance in aerial robots. In *Proc. of IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 772–779, 2012.

[10] T. Liu, J. Sun, N.-N. Zheng, X. Tang, and H.-Y. Shum. Learning to detect a salient object. In *Proc. of IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007.

[11] J. Michels, A. Saxena, and A. Y. Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proc. of the 21st Intl. Conf. on Machine Learning (ICML)*, pages 593–600, 2005.

[12] A. Oh, A. Kosaka, and A. Kak. Vision-based navigation of mobile robot with obstacle avoidance by single camera vision and ultrasonic sensing. In *Proc. of IEEE Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 704–711, 1997.

[13] K. Souhila and A. Karim. Optical flow based robot obstacle avoidance. *International Journal of Advanced Robotic Systems*, 4(1):13–16, 2007.

[14] C. N. Viet and I. Marshall. Vision-based obstacle avoidance for a small, low-cost robot. In *Proc. of IEEE Intl. Conf. on Advanced Robotics (ICAR)*, 2007.