J Real-Time Image Proc
DOI 10.1007/s11554-013-0356-z

SPECIAL ISSUE PAPER

Real-time monocular image-based path detection

A GPU-based embedded solution for on-board execution on mobile robots

Pablo De Cristóforis · Matías A. Nitsche · Tomáš Krajník · Marta Mejail

Received: 18 December 2012 / Accepted: 13 May 2013
© Springer-Verlag Berlin Heidelberg 2013

Abstract In this work, we present a new real-time image-based monocular path detection method. It does not require camera calibration and works on semi-structured outdoor paths. The core of the method is based on segmenting images and classifying each super-pixel to infer a contour of navigable space. This method allows a mobile robot equipped with a monocular camera to follow different naturally delimited paths. The contour shape can be used to calculate the forward and steering speed of the robot. To achieve real-time computation necessary for on-board execution in mobile robots, the image segmentation is implemented on a low-power embedded GPU. The validity of our approach has been verified with an image dataset of various outdoor paths as well as with a real mobile robot.

P. D. Cristóforis (✉) · M. A. Nitsche · M. Mejail
Laboratory of Robotics and Embedded Systems, Computer Science Department, Faculty of Exact and Natural Sciences, University of Buenos Aires, Pabellón I-Ciudad Universitaria, Ciudad Autónoma de Buenos Aires, Buenos Aires, Argentina
e-mail: pdecris@dc.uba.ar

M. A. Nitsche
e-mail: mnitsche@dc.uba.ar

M. Mejail
e-mail: marta@dc.uba.ar

T. Krajník
Intelligent and Mobile Robotics Group, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, Karlovo náměstí 13, Prague 2, Czech Republic
e-mail: tkrajnik@labe.felk.cvut.cz

1 Introduction

One of the current challenges of mobile robotics is to achieve complete autonomy, i.e. to develop a robot that can carry out its tasks without the need of a human operator. The ability to safely move in one's environment is fundamental for an autonomous mobile robotic system. To navigate in the real world, a mobile robot needs not only to avoid collisions, but also to detect those portions of the world that are forbidden, dangerous or impossible to traverse. For a large class of terrestrial reactive navigation problems, the world in front of the robot can be modeled as a flat plane, and any detected point that deviates from the planar model can be considered to be an obstacle. Many obstacle avoidance algorithms use active sensors such as sonars [4], laser range finders [33], radars [15] and 3D cameras based on time of flight [21] and structured light [14]. These sensors are inherently suited for the task of obstacle detection and can be used easily because they directly measure the distances from obstacles to the robot.

However, ultrasonic sensors suffer from specular reflections and poor angular resolution. Standard laser range finders are precise, but they only provide measurements in one plane. Three-dimensional laser rangefinders, as well as most radars, are not suitable for small robot applications because of size, weight and energy consumption. Most 3D cameras illuminate the perceived scene by infrared light and do not work outdoors due to the presence of sunlight. Since All active sensors transmit signals, these might interfere with each other if multiple sensors or multiple robots are present in the same environment. Moreover, the distance measurements provided by these sensors are not suitable to distinguish between different types of ground surfaces or recognize the shape of a road without the presence of bounding structures such as

Published online: 22 June 2013

surrounding walls. For all these reasons, active range sensors are not sufficient by itself for solving the problem of outdoor road navigation.

On the other hand, visual sensors are affordable, small and can provide higher resolution data and virtually unlimited measurement ranges. They are passive and, therefore, do not interfere with each other. Most importantly, visual sensors can not only detect obstacles, but also identify forbidden areas or navigate mobile robots with respect to human-defined rules (i.e. keep off the grass). Such forbidden areas are not obstacles, since they are in the same plane as the path, but should be considered as non-traversable.

There are several issues in using vision for path following. For one, road appearance varies from place to place, which makes difficult to define what a road is. Consequently, many approaches rely on the teach-and-replay paradigm [6, 18, 30] to navigate through previously visited areas. These systems require the robot to traverse a specific environment during a human-guided training step so it learns the environment before it can navigate it autonomously. Typically, the robot extracts visual features such as SIFT [20] or SURF [1] and stores them in a map. During the replay or test stage, the robot compares the stored visual features with what the ones currently seen to estimate its position and determine its control variables.

The teach-and-replay paradigm has some drawbacks. First the robot workspace is limited only to the regions visited during the training step. In addition, several training steps have to be performed to deal with variable environment appearance caused by varying illumination or seasonal changes. The training process requires human intervention every time a robot moves to a new workspace, which can become tedious for large outdoors environments. Moreover, systems based on the teach-and-replay paradigm assume that the environment does not change over time. All these mentioned drawbacks limit the applicability of the teach-and-replay approach. Therefore, to achieve a completely autonomous navigation system, it is desirable to remove any human intervention and cope with unknown environments, without any a priori information [3, 9].

The aim of this paper is to facilitate a mobile robot equipped only with a monocular camera to autonomously drive through semi-structured outdoors paths, avoiding obstacles should they appear. The proposed method is based on segmenting the images captured with the camera and classifying each region to infer a contour of traversable space. The robot performs all processing on-board, given real-time constraints imposed by the robot motion. At this point, the use of specific hardware to achieve real-time image processing becomes critical. In this paper, we present an embedded solution based on a low-power Graphics Processing Unit (GPU) that it is mounted on-board the robot. A depiction of the path detection system running on-board the mobile robot can be seen on Fig. 1.

GPUs have lately gained considerable popularity as cheap, powerful and programmable general purpose processors outside their original application domain. Recent models are able to sustain over hundreds of GFLOPs and due to their highly parallel architecture, GPUs are attractive platforms for intensive data-parallel algorithms. This kind of hardware is, therefore, very well suited for on-board mobile robot with visual-based perception. Although general-purpose computing on graphics processing units (GPGPU) has been an active area of research for decades, the introduction of Compute Unified Device Architecture (CUDA) and CTM has finally brought it within reach of a broader community, giving programmers access to dedicated application programming interfaces (APIs), software development kits (SDKs) and GPU-enabled C programming language variants.

This paper will consider the optimization of an existing GPU segmentation algorithm using CUDA. This segmentation results in a numbers of *super-pixels* or *segments* that can be classified as traversable or non-traversable space given an example of what the traversable space looks like (using a sub-region of the image corresponding to the area directly in front of the robot). The implementation of the

Fig. 1 ExaBot robot performing autonomous navigation in an unknown outdoor environment. The horizon line (*red*) is detected as well as contour of navigable space (*blue*). Middle points of the road are computed (*yellow*) to guide the robot. To achieve real-time constraints imposed by robot motion, the image segmentation is implemented on a low-power embedded GPU (on *top* of the robot)

optimized segmentation algorithm is compared to the original GPU version and with another commonly used algorithm running on a recent CPU.

After individual classification, the resulting positively classified group of super-pixels nearest to the example area is assumed to correspond to the most likely navigable path. Finally, by computing the contour of this area, a motion line can be extracted and used for guiding the robot through the path. A control law is defined that restricts the robot to remain inside this detected navigable area. From these steps, the most computationally demanding step corresponds to the image segmentation. Therefore, we focus on optimizing this portion of the method to allow real-time execution.

The outline of the paper is as follows: in Sect. 2, the related works are mentioned. Section 3 gives a brief overview of the proposed method. In Sect. 4, the horizon detection algorithm is described. Section 5 discusses different approaches for image segmentation, justifies the choice of Quick shift and presents an implementation optimized for GPU. Section 6 deals with the problem of the classification of each super-pixel in traversable and non-traversable. Section 7 explains how to extract the road contour from the classified image. Section 8 proposes a control law to maintain the robot in the middle of the road. Section 9 shows experimental results with an image dataset of various outdoor paths as well as with a real mobile robot. Finally, Sect. 10 concludes the paper.

2 Related work

There are several previous works that propose autonomous vision-based path following methods targeted towards outdoor environments using monocular cameras as their main sensor and without requiring a training step. Most of these try to recognize the road's appearance as well as its boundaries without any *a priori* information. Some basic methods rely on recognizing road edges that separate the road from its surroundings by identifying the lines in the image (for example, using the Hough transform) [37]. But this approach is not valid for semi-structured outdoor roads where the edges are not easily distinguishable. In some cases, the road is simply estimated as a geometrical shape as in [28] where an histogram is utilized for differentiating between the road region defined as a triangle and its two flanking areas. In addition, there is also a group of techniques that try to recognize the road using the road's vanishing point [16, 22, 24, 27]. In general, these techniques rely on detecting converging edge directions to vote for the most likely vanishing point. The biggest problem of this type of approach appears when the road is not straight but curved or its edges are not well structured. Others systems detect the road by performing color and texture segmentation [2, 19]. Again the downside of these methods is that they usually involve computationally expensive algorithms that could not be implemented on a mobile robot with limited computational equipment. Finally, there are some works that merge both image segmentation and vanishing point detection. In [5], better results are reached and computation is performed on-board in a mobile robot, but the robustness of the system depends on the correct computation of the vanishing point.

Other group of works performs visual path following using pixel classification. By assuming that the robot is operating on a flat surface, the problem can be reduced to classifying pixels into two classes: traversable or non-traversable. This approach that is suitable for robots that operate on benign flat terrains has been used in a variety of works for indoor navigation. In [34], classification is done at the pixel level. First, the image is preprocessed with a Gaussian filter. Second, the RGB values are transformed into the HSV (hue, saturation, and value) color space. In the third step, an area in front of the mobile robot is used for reference and valid hue and intensity values inside this area are histogrammed. In the fourth step, all pixels of the input image are compared to the hue and intensity histograms. A pixel is classified as an obstacle if its hue or intensity histogram bin values are below some threshold. Otherwise the pixel is classified as belonging to the ground. This method is fast, as no complex computation is involved. The main drawback of this method is its low robustness to variable illumination and noise.

Due to this inconveniences, the idea of segmenting the image into a number of super-pixels (i.e. contiguous regions with fairly uniform color and texture) arises. In [29], a graph-based image segmentation algorithm [11] is used for indoor navigation in structured environments. Initially, an image is represented simply by an undirected graph, where each image pixel has a corresponding vertex. The edge is constructed by connecting pairs of neighboring pixels. Each edge has a corresponding weight, which is a non-negative measure of the dissimilarity between neighboring pixels. Beginning from single-pixel regions, this method merges two regions if the minimum intensity difference across the boundary is greater than the maximum difference within the regions. Once the image is over-segmented in super-pixels, each one is labeled as belonging to the ground or non-ground region using the HSV histogram approach. While this method is more stable and robust, it is quite computationally expensive, so the exploration algorithm that uses this method has to stop the robot periodically to capture and process images of the workspace. Using the same image segmentation algorithm, in [36] super-pixels that are likely to be classified equally are grouped into constellations. Constellations can then be labeled as floor or non-floor with an estimator of

confidence depending on whether all super-pixels in the constellation have the same label. This is a more robust method, but computationally expensive. Again, the robot must be stopped periodically to perform computations.

Finally, there is another group of systems that perform visual path following using a probabilistic approach. In [8], a visual model of the nearby road is learnt using an area in front of the system as a reference. A model is built which is subsequently used to score pixels outside of the reference area. The basic model of the road appearance is a mixture of Gaussians (MOG)-model in RGB space. This approach was used by the navigation system, which won the DARPA Grand Challenge in 2005.

In this paper, some ideas from previous works mentioned above are used to develop a new method for real-time image-based autonomous robot navigation for semi-structured outdoor roads. In outdoor semi-structured environments, the traversable area is often cluttered by small objects with a color of the forbidden area, e.g. grass, tree leaves, water or snow in the middle of the road. In this case, most classification methods working at a pixel level would perform worse than methods which first segment the image to several areas with similar texture or color. Since such image segmentation is computationally expensive, we propose to use a GPU-based embedded solution to achieve the real-time constraints imposed by the robot motion. To achieve a robust classification of the superpixels, a probabilistic approach similar to [8] is used. The main difference of the proposed method is that it works in HSV color space instead of RGB.

3 Method overview

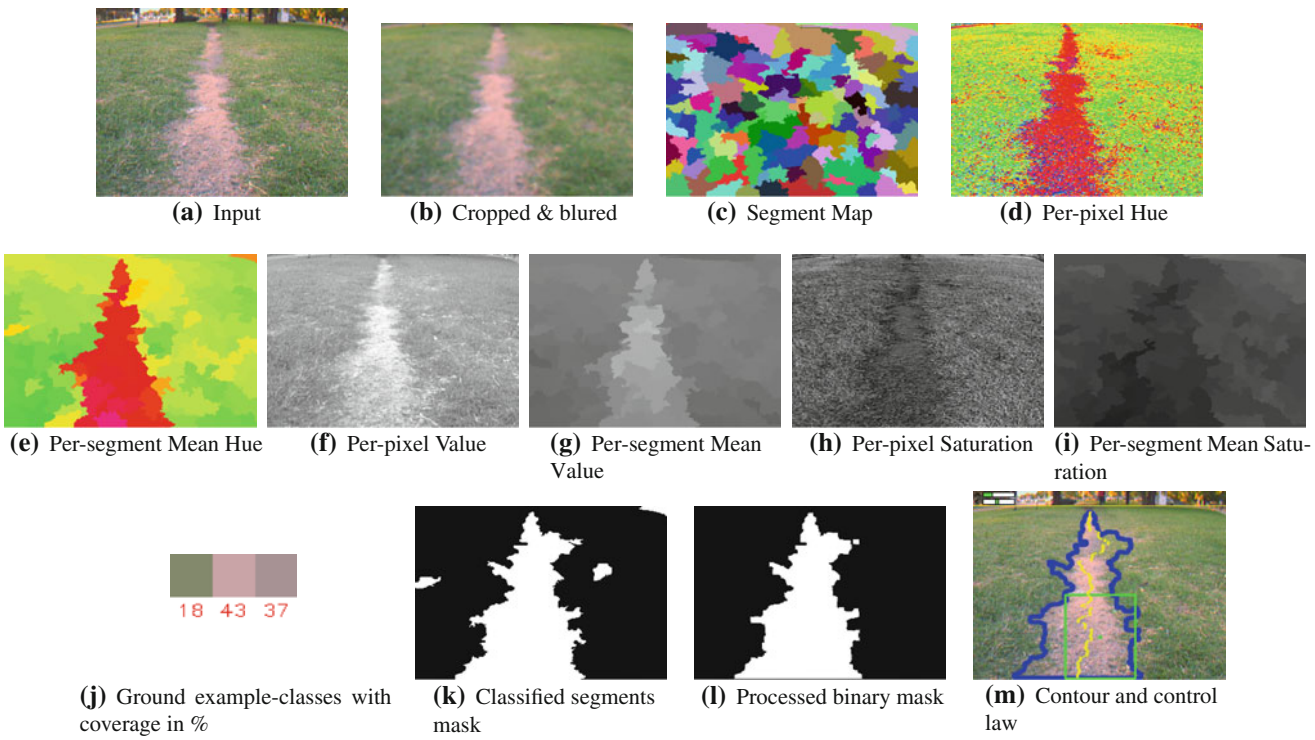
The proposed method processes an input image to obtain a robot control command as its final output (Fig. 2). Each step of the pipeline is briefly described here and in detail in the following sections:

3.1 Horizon detection

The input image (Fig. 2a) is analyzed to find the horizon (Fig. 3) and then cropped such that only the region below it is fed to the rest of the pipeline.

3.2 Color space conversion and smoothing

Along the RGB cropped image, an HSV version is also obtained. In the following steps, the HSV version is almost always used, unless specified. A median-blur filter is applied

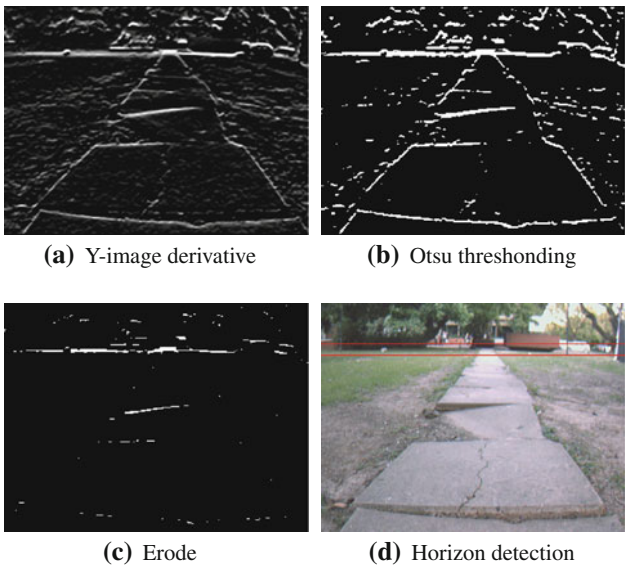


(a) Input **(b)** Cropped & blured **(c)** Segment Map **(d)** Per-pixel Hue

(e) Per-segment Mean Hue **(f)** Per-pixel Value **(g)** Per-segment Mean Value **(h)** Per-pixel Saturation **(i)** Per-segment Mean Saturation

(j) Ground example-classes with coverage in % **(k)** Classified segments mask **(l)** Processed binary mask **(m)** Contour and control law

Fig. 2 Pipeline description of the complete road-detection method: **a** input image as acquired from the camera, **b** cropped image below automatically detected horizon, **c** segment map. For each segment, mean and covariance are computed: **e**, **i**, **g** per-segment hue, saturation and value means (per-pixel hue, saturation and value on **d**, **h** and **f**, for reference). Binary mask (**k**) obtained from classification using ground models from ROI (**j**) is processed (**l**) and used for extracting road-contour and control values (**m**)



(a) Y-image derivative **(b)** Otsu threshonding

(c) Erode **(d)** Horizon detection

Fig. 3 Pipeline description of the horizon detection algorithm: **a** Sobel operator is first applied to the input image to obtain *Y*-image derivative, **b** Otsu thresholding is used to transform *Y*-image derivative to binary, **c** an erosion filter is applied to reduce noise and remove thin lines, **d** a histogram is computed to find the sub-image where the horizon line is located

to the RGB image (Fig. 2b), to reduce noise, without smoothing edges. A blurred HSV version is also obtained.

3.3 Segmentation

The RGB version of the blurred image is segmented. This step produces a *segment map* (Fig. 2c), which consists of an image of labeled pixels. *Segment processing*. The map of segments is processed to build a *list of segments* or *super-pixels*, which describes groups of pixels with the same label. While building this list, mean and covariance (both in RGB and HSV color spaces) are computed for each segment (Fig. 2d–i).

3.4 Classification

A rectangular region of interest (ROI) corresponding to the area directly in front of the robot gives the system an example of the road appearance (Fig. 2j). The RGB/HSV information of this region is modeled by a mixture of gaussians (MOG). By comparing each super-pixel in the image to this MOG model, a classification criteria are applied to decide whether a super-pixel belongs to the road or non-road classes. The result of this classification is a binary mask (Fig. 2k).

3.5 Contour extraction

From all closed regions classified as road, the most likely one is selected and a morphological opening filter is applied to "close" small gaps (Fig. 2l). The contour of this final region is extracted.

3.6 Control law

From the previous road contour, middle points are computed (Fig. 2m). These points are then used to compute the linear and angular speeds of the robot.

4 Horizon detection

A fast horizon detection algorithm is used to find the area of the image where the horizon line is located. This horizon line implies that the portion of the image above does not need to be processed and, therefore, can be ignored for the rest of the computation, considerably reducing execution time. While it is possible to fix this value in advance, automatic horizon detection allows the robot to face uphill or downhill paths without either disrupting road detection or negatively affecting performance. While on outdoor roads side motion may also be present, this is case is not considered on this work and was not found during experiments.

In [8], authors propose an image-based horizon detection algorithm which rests on two basic assumptions: (1) the horizon line will appear approximately as a straight line in the image; and (2) the horizon line will separate the image into two regions that have different appearance, i.e. sky pixels will look more like other sky pixels and less like ground pixels, and vice versa [10]. The same assumptions are used in other image-based horizon detection algorithms [23] and are valid for the case of autonomous vehicles that cross open areas or for unmanned aerial vehicles. However, the second hypothesis may not be true for semi-structured outdoor roads, where it is common that pixels above the horizon line correspond to objects, such as plants, trees, cars, buildings or even people rather than only sky, and cannot be represented with the same model.

Thus, instead of using the second hypothesis, in this work we assume that the horizon line will be more or less parallel to the bottom of the image, which is valid for robots that operate on flat or small slope terrains. Based on this assumption we define the horizon as the area of the image where the greatest change in pixel intensity over the *Y* axis direction is detected. We use the Sobel filter as a discrete differentiation operator to compute an approximation of the gradient of the image intensity in *Y* axis direction. Then, we apply the Otsu thresholding method (OTM) to the image derivative to segment the image into two classes and, therefore, obtain a binary image. OTM finds the best threshold which maximizes interclass variance and minimizes intraclass variance. This is very proper

for road detection, because OTM overcomes the negative impacts caused by environmental variation, without user assistance. Moreover, its low computational complexity makes it suitable for real-time applications and it still remains one of the most referenced thresholding methods [31]. There are some previous works that use OTM for horizon detection [23], but in this paper we apply it to the image derivative.

Once we have a binary image, we apply an erosion filter to reduce noise and remove thin lines. Afterwards, we divide the image in a number of sub-images of equal heights and compute how many pixels are above the Otsu threshold for each sub-image, which can be thought of as a histogram of pixels that belong to the horizon, following the idea of [23]. For our tests, we use ten sub-images. The sub-image with the highest amount of foreground pixels is where the horizon is expected to be. In this work, we introduced an improvement which consists in performing two passes: in the second iteration, we start to divide the image at an offset corresponding to half the height of the sub-images to overlap the first iteration. This is very useful for cases where the horizon is in between two sub-images. With the second pass, in this case, we can find a better sub-image candidate containing the horizon. To choose between the two sub-images detected during each pass, the candidate sub-image with the highest ammount of foreground pixels is chosen as the winner. To optimize computation, we only process the upper half of the image because we know that the horizon is never below. Inside the sub-image containing the horizon, the middle horizontal line is assumed to be the most likely horizon in the image. Figure 3 resumes the horizon finding algorithm.

5 Segmentation methods

As it will be shown, segmentation is the most time-consuming step from the complete road detection method. To satisfy real-time and on-board execution constraints, several segmentation methods were considered and tested. In particular, the Quick shift resulted to be the most adequate for the computing platform utilized, which consists of an embedded GPU processor on-board a mobile robot.

5.1 Graph-based

In [11], an efficient graph-based segmentation method is presented. In general terms, the algorithm first constructs a fully connected graph where each node corresponds to a pixel in the image. Pixel intensities between neighbors are analyzed and edges are broken whenever a threshold is exceeded. The resulting unconnected sub-graphs define the segments.

The method works as follows. An undirected graph $G = (V, E)$ is defined, where V is the set of vertices (pixels) and E the set of edges. Each edge $e_{i,j} \in E$ has an associated weight w_{ij}, which indicates the dissimilarity between vertices v_i and v_j. In image segmentation, this weight is obtained by a difference in pixel intensity, color, location, etc. The segmentation S can be defined as a partition of V into connected components $C \in S$ of a graph $G' = (V, E')$, where $E' \subseteq E$. The end result of the segmentation is such that edges between two vertices in the same component have relatively low weights, and edges between vertices in different components have higher weights.

The main steps of the algorithm are

1. Sort E into $\pi = (e_1, \ldots, e_m)$, by non-decreasing edge weight
2. Define initial segmentation S_0, such that $C_i = v_i$
3. Repeat for $q = 1, \ldots, m$.
   (a) Define v_i, v_j as the vertices connected by edge e_q.
   (b) Construct S_q as follows: if v_i and v_j are in disjoint components of S_{q-1} and $w(e_q)$ is small compared to the internal difference of both those components, then merge the two components.

As demonstrated by the authors, the proposed algorithm is found to have $O(m \log(m))$ complexity for the case of non-integer weights.

5.2 Quick shift

Quick shift [35] is an example of a non-parametric mode-seeking algorithm, which aims to estimate an unknown probability density function. Density estimation is performed by associating data points to modes of the underlying probability density function.

One commonly used approach for the estimation is to use a *Parzen-window* for the density estimation [25]. Formally, given N data points $x_1, \ldots, x_N \in \chi = \mathbb{R}^d$, the Parzen-window approach estimates the probability as:

$$p(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x_i), \tag{1}$$

where $\boldsymbol{\delta}(\boldsymbol{x})$ is commonly referred to as the *kernel*, which is generally written as a Gaussian. The mode-seeking algorithm evolves data points x_i towards a mode of $p(x_i)$, by means of gradient ascent over the kernel. All points associated with the same mode form a cluster.

There are several mode-seeking methods that differ in the strategy used to evolve that data points towards a mode. Quick shift is actually an improvement over Medoid shift

[32] which, in turn, is an improvement over the original Mean shift [7] method. Medoid shift simplifies this evolution of the data points by restricting the point trajectories to only move over the data points x_i themselves. The downside of this approach is mainly its slow speed in practice [35]. Finally, Quick shift simplifies trajectories even further by not analyzing the gradient and simply moving data points to their nearest neighbor for which there is an increment in the density $p(x)$.

In [35], Quick shift is applied to the problem of image segmentation and it is shown that their proposed method is considerably faster than Mean shift, and marginally faster than Medoid shift.

The first step of the Quick shift algorithm involves the computation of the density for each pixel, by means of analyzing a local neighborhood inside which contributions to the density are the most significative. The second step links every pixel to its nearest neighbor with higher density. Pseudo-code is presented in Algorithm 1.

Algorithm 1 Quick shift image segmentation algorithm in pseudo-code

```
function COMPUTEDENSITY(image)
    for x in all pixels of image do
        P[x] ← 0
        for y in all pixels less than 3σ away from x do
            P[x] ← P[x] + e^{−(f[x]−f[y])^2 / 2σ^2}
        end for
    end for
    return P
end function

function LINKNEIGHBORS(image, P)
    for x in all pixels of image do
        for y in all pixels less than τ away from x do
            if (P[y] > P[x] and distance(x,y)
                is smallest amog all y) then
                d[x] ← distance(x,y)
                parent[x] ← y
            end if
        end for
    end for
    return d, parent
end function
```

5.3 Quick shift on the GPU

In [12], an implementation of the Quick shift algorithm for execution on GPUs is presented. Due to the independence in the computation of the density between different pixels, the authors exploit the parallel execution capabilities of GPUs for this computation. Using several features of this kind of computing platforms, the cost associated with the type of access pattern associated with a parallel implementation of Quick shift is greatly reduced. Since for the computation of the density over a single pixel requires accessing a local neighborhood of pixel densities, the redundant access of neighbors is managed using the *texture memory space* of the GPU which includes efficient caching.

Compared to a CPU implementation, the proposed algorithm implementation is between 10 and 50 times faster when running on medium-range hardware. For the case of 256×256 pixel resolution images, the GPU implementation works at 10 Hz.

In this work, the Quick shift algorithm running on a GPU was chosen due to its speed and simplicity. However, the hardware utilized for experiments by the authors is still more powerful than what can be found on low-power embedded platforms commonly present on mobile robots. Therefore, in this work several simple optimizations were included in the originally available source-code [13] which implements the GPU version of the Quick shift algorithm.

The first optimizations that were introduced consist of careful tuning of the number of concurrent threads executed, the registers required for code compilation (which affects the efficiency of the thread scheduling and thus the parallelization level), taking into account the capabilities of the specific card to be used. The second main optimization that was introduced involves a simpler handling of out-of-bound accesses which arise when searching the neighborhood of pixels near the edges of the image. In the original implementation, these were avoided explicitly, where in our case, the *clamping mode* of the texture memory is used. Here, these accesses simply return the nearest valid pixel in the image (effectively repeating pixels in the image outwards). By introducing this change, the code can be simpler and more efficient. Finally, memory accesses in general were reduced by delaying them up to the point where they were for certain to be required.

These optimizations reduce computation to the point of allowing real-time execution on the chosen embedded platform.

6 Classification method

To achieve a robust classification of the segments, a probabilistic approach method is used, based on Dahlkamp et al. [8] with some differences.

In abstract terms, the classification step aims to determine if a population sample, modeled by a Gaussian probability distribution $\mathcal{N}(\mu, \Sigma)$, with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ, represents an instance of a more general model of the road class or not. This road class, in turn, is represented by a mixture-of-gaussians (MOG) model. However, in this work, the gaussian elements of this mixture model are readily available and, therefore, the global mixture model is not explicitly computed.

In the following sections, the classification method is presented in detail.

6.1 Segment model computation

Given a segment composed of N pixels $P^{\mathrm{RGB}}_{i=1\ldots N}$ represented as threedimensional vectors in RGB color-space, mean and covariance are computed as follows:

$$\begin{aligned}\mu^{\mathrm{RGB}} &= \sum_{i}^{N} \frac{P_i^{\mathrm{RGB}}}{N} \\ \Sigma^{\mathrm{RGB}} &= \frac{(P_i^{\mathrm{RGB}} - \mu^{\mathrm{RGB}})(P_i^{\mathrm{RGB}} - \mu^{\mathrm{RGB}})^{\mathrm{T}}}{N-1}\end{aligned} \tag{2}$$

Since individual segments are to be classified using the HSV color space, mean and covariance need to be computed for pixels in this representation. However, computing the average of Hue values from a sample is, in principle, ill-posed since this channel is actually a circular measure (can be interpreted as an angle). The solution utilized was to compute first the RGB mean, and then convert this RGB value to HSV. Finally, in order to compute the HSV covariance matrix, the distance between a population sample and the mean was restricted to angles less than 180°.

It should be noted that this approach requires the original RGB representation of the image, along with the computed HSV version. However, the RGB covariance matrix is not required.

The reason for using the HSV color space is that, in contrast to RGB, the chrominance and luminance information are maintained in separate channels. This provides better control when selecting thresholds over each HSV channel and also provides some degree of invariance between chrominance and luminance.

6.2 Road class model computation

To compute a model for the road class, a rectangular ROI corresponding the area directly in front of the robot is used as an example of the road appearance. From this ROI, a model is extracted. However, this area may contain very dissimilar information due to textures (e.g. tiles) or outliers (e.g. grass on the side of the cement road). Therefore, this area is represented with a MOG model. Given that the image was already segmented and each segment is modeled with a Gaussian distribution, the elements of the mixture model are already computed.

The classification step starts by computing the intersection of the segments of the image and the rectangular ROI. Then, segments in this intersection are re-grouped by similarity by iteratively joining similar models. This similarity is defined in terms of the Mahalanobis distance, defined in this case for two Gaussian distributions. Two distributions are said to be similar when:

$$(\mu_1 - \mu_2)^{\mathrm{T}}(\Sigma_1 + \Sigma_2)^{-1}(\mu_1 - \mu_2) \leq 1 \tag{3}$$

The process of joining similar segments inside this ROI works iteratively, by merging two segment's mean and covariance matrices into one. On each step, for each segment inside this region, among all other segments that satisfy Eq. (3), the nearest neighbor in Mahalanobis space is chosen for merging. This merging by pairs continues until no more segments can be merged. To merge two segments with means μ_1, μ_2, covariance matrices Σ_1, Σ_2 and number of pixels N_1, N_2, a new segment is obtained by:

$$\begin{aligned}N_f &= N_1 + N_2 \\ \mu_f &= (\mu_1 N_1 + \mu_2 N_2)/(N_1 + N_2) \\ \Sigma_f &= (\Sigma_1 N_1 + \Sigma_2 N_2)/(N_1 + N_2)\end{aligned}$$

With this minimal set of Gaussian models, each segment in the image can now be classified as belonging to the road or non-road classes by comparing them to all elements of this set. Only elements which cover the ROI by more than a specified threshold c (*coverage*) are considered. In this way, small outliers inside the ROI can be ignored. Again, the notion of similarity is defined as in Eq. (3).

When both grouping similar road segments inside the ROI and classifiying segments in the complete image, in Eq. (3) the sum of the covariance matrices is post-processed to include the notion of minimum covariance. These minimums are added in the computation as a way to increase the threshold used for this classifier from (3), but allowing to affect each channel of the HSV color space differently. For the case of re-grouping similar segments, this allows to be more permissive by relaxing the notion of similarity. On the other hand, when classifying segments, these thresholds account for the variance which exists in the complete visible road, as opposed to only the ROI. This issue appears frequently since the color of the road changes smoothly towards the vanishing point.

To apply these minimums to the sum of the covariance matrices, an eigen decomposition of this resulting matrix is first obtained:

$$\Sigma_{\mathrm{sum}} = VDV^{\mathrm{T}}, \tag{4}$$

where V is the matrix of eigenvectors and D the diagonal matrix of eigenvalues. Then, D is modified as

$$D'_i = \begin{cases} D_i & D_i > T_i \\ T_i & \text{else} \end{cases}$$

where T is a diagonal matrix of minimum values. Finally, a new covariance matrix is recomposed using Eq. (4) using V and D'. It should be noted that the thresholds used for

grouping road segments do not generally depend on the environment and can be preset. On the other hand, thresholds used for classification may need adjusting when high color variance exists in the complete road.

7 Road contour extraction

The previous classification step produces a binary mask which distinguishes road from non-road pixels. The goal of the following step is to extract a contour of the road within the captured image. With this contour, a simpler road representation can be used to apply a control law which will maintain the robot centered.

The binary mask, however, may contain several unconnected patches of pixels detected as road, since areas of similar appearance may exist outside but nearby the navigable area. Therefore, the first task is to identify the correct patch. Two possibilities were analized: find the patch which includes the center point of the rectangular ROI or extract the contour with the largest perimeter. Experiments have indicated that the latter approach is more robust.

Once this contour is extracted, a new binary mask consisting only of the selected patch is generated. This second mask is then processed with a morphological opening operation (an erosion followed by a dilation) with the main purpose of removing peninsulas from the main patch, which may appear in naturally demarked roads and should not be included in the following steps. While the erosion operation may disconnect these patches, the area of all road regions will be reduced. Therefore, by dilating the image afterwards, all road areas are again expanded without reconnecting them.

The final road contour is again extracted from the processed binary mask.

8 Control law

From the previously determined contour, a virtual road's center lane is estimated to maintain the robot centered and away from road edges. First, by going row-by-row in the image, the middle point for the current row is obtained by subtracting the leftmost and rightmost pixel's horizontal positions of the road contour. When finished, the list of middle points is used to compute angular and linear speeds with a simple and effective control-law, based on previous [17].

From the list of n horizontal values x_i of the i-th middle point of the detected road region, angular speed $\boldsymbol{\omega}$ and linear speed x are computed as follows:

$$\omega = \alpha \sum_{i}^{n} \left(x_i - \frac{w}{2} \right),$$
$$x = \beta n - |\omega|$$

where w is the width of the image. The effects of this control law are such that the robot will turn in the direction where there is the highest deviation, in average, from middle points with respect to the image's center vertical line. This line can be assumed to be the position of the camera, which is mounted centered on the robot. Therefore, whenever a turn in the road or an obstacle is present, the robot will turn to remain inside the road and avoid the obstacle. The linear speed of the robot is inversely proportional to the angular speed determined by the previous computation. This has the effect of slowing down the robot whenever it has to take a sharp turn.

9 Experimental results

The proposed method was tested experimentally in several aspects. In terms of performance, computation time was measured over two distinct platforms (modern laptop and embedded hardware on-board robot) for the individual segmentation step and for an iteration of the complete method. For qualitative analysis, the road detection capability of the system was evaluated using offline execution over previously captured images, as acquired by the robot's camera during a human-guided step. Finally, online testing was performed by executing the proposed algorithm on-board the robot on real-time to assess the closed-loop behavior.

The robot used for online experiments (and some offline dataset acquisition) was the ExaBot [26] (Fig. 1), which features a differential-drive motion and supports different sensor configurations depending on the needs. For this work, the robot was equipped with an AT3IONT Mini-ITX computer, featuring a dual core Intel Atom 330 processor with an embedded ION nVidia graphics card. This embedded GPU is CUDA enabled, allowing it to be used as a GPGPU platform, with 16 cores running at 1.1 GHz and with 256 MB of RAM. On the other hand, as a reference, a modern Laptop with an Intel Core i5 at 2.30 GHz and 4 GB of RAM was also used for performance measurements.

A firewire camera was used for acquiring images (model 21F04, from *Imaging Source*) with a 3.5–8mm zoom lens, set at its widest focal length. While the camera is capable of capturing images at 640 × 480 px images at 15 fps, a smaller resolution of 320 × 240 px (at 30 fps) was chosen since it was enough for proper road detection. This smaller resolution also decreases computation times.

9.1 Offline testing

To assess the quality of the algorithm in terms of its ability to perform road detection, the proposed method was executed over more than 1,000 images. Images were obtained from different sources: some of them were acquired with the camera mounted on the ExaBot robot (altitude from ground: 40 cm) while others were acquired by a camera mounted on a Pioneer 3AT robot (altitude from ground: 70 cm). These datasets depict different situations with varying difficulty in terms of road distinctiveness from surrounding areas, road shape, texture and color, under many lighting conditions including shadows and reflections and during different seasons.

In Fig. 4, a series of example images acquired over distinct structured and semi-structured outdoor roads are presented as obtained by the final step of the proposed method. In some cases (Fig. 4a–j), the road was clearly demarked from surrounding elements (cement/gravel with grass). However, some of these include more difficult situations consisting of tiles of stones (Fig. 4i, j). Included in these tests, there are several cases of semi-structured outdoor roads, involving wet areas with reflections (Fig. 4k–m), leaves at the sides (Fig. 4n–p), stone roads covered with snow in winter (Fig. 4q, r) and even dirt roads (Fig. 4s).

9.2 Online testing

For the purpose of analyzing the stability of the proposed control law along the road detection method when executed online over a stream of images acquired on real-time by the camera, the robot was placed on outdoor roads and positioned in different starting configurations, some of which consisted on extreme cases which would not be reached without manual displacement. As an example, a series of successive frames are presented in Fig. 5. The robot is initially placed at the side of the road (Fig. 5a), pointing away from it. After enabling robot control, the robot soon

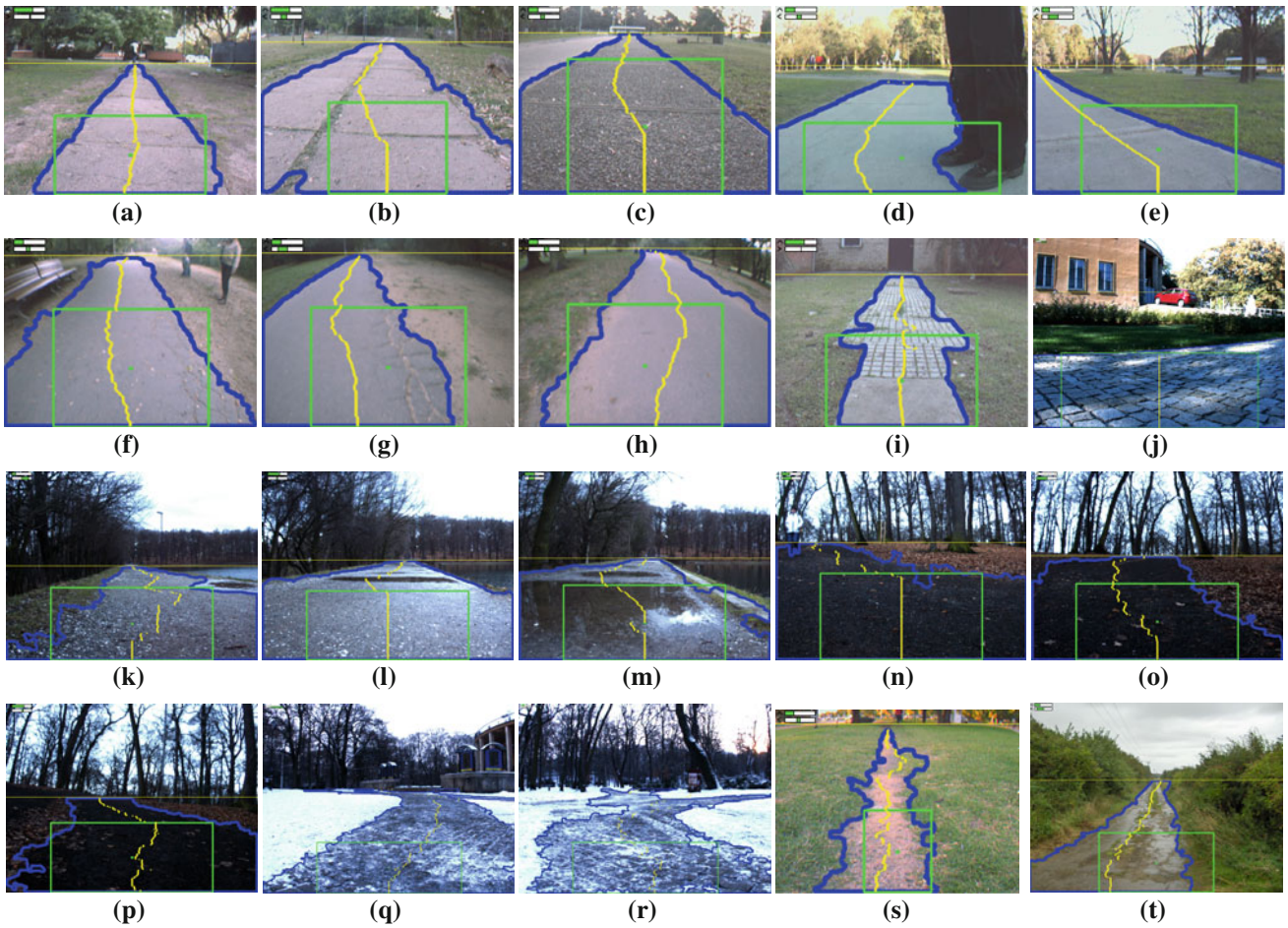


Fig. 4 Example final images processed offline, as obtained from several datasets: the *green rectangle* represents the ROI used for extracting road appearance samples, the *yellow line* corresponds to the automatically detected horizon, the *blue contour* delimitates the detected ground region, *yellow points* correspond to road middle-points from which control law is computed, *small bars* in *top-left* show the control law output for linear (*top bar*) and angular speed (*bottom bar*), going from 0 to 1 and from −1 to 1, respectively

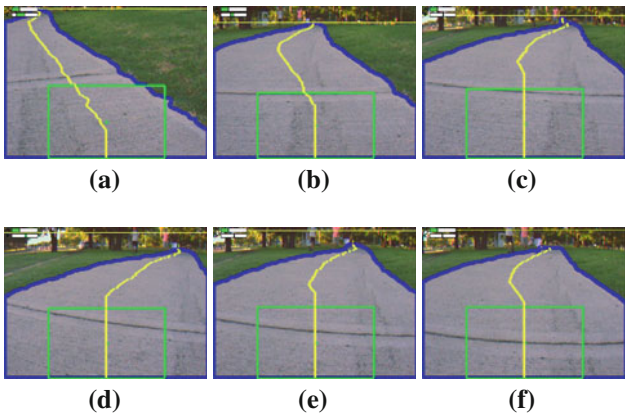


Fig. 5 Example of online testing: series of successive frames where robot was started from a deviated position and ended with robot centered in road

turns towards the road (Fig. 5b–d) advances and then turns again to remain on course (Fig. 5e–f). As can be seen in Fig. 5, the road edges exceed the image frame. Since the road edges are not visible it is assumed that the road is wide enough for safe traversing of the road. The robot will continue forward until sufficiently close to the edge to steer away.

Since real-world testing is a time costly process, system behavior has been first tested thoroughly over off-line datasets and then simply control law testing was performed to insure stability of robot motion.

9.3 Segmentation methods performance

In this work, several image segmentation methods were considered to meet the real-time constraints required for on-board execution on embedded hardware. In this section, the execution time of these algorithms is presented, measured on different platforms. While each method utilizes a different set of parameters, the corresponding values were chosen in each case by maximizing execution speed. Of importance for the classification quality and computation speed of the road detection algorithm as a whole are both the size and the number of segments (as expected, quantities are inversely proportional). In these terms, the differences obtained in the resulting segmentations for the chosen set of parameter values were negligible.

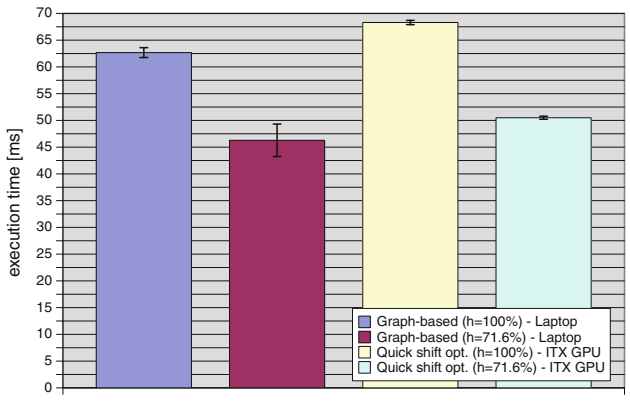
In Fig. 6, mean execution speeds (over 32 iterations) are presented corresponding to different segmentation algorithms, executing platforms and relevant parameters. The algorithms were executed over a test image captured during outdoor experiments.

In Fig. 6a acceptable timings are presented, whereas in Fig. 6b the execution speeds presented do not permit real-time execution and stable control of the robot.

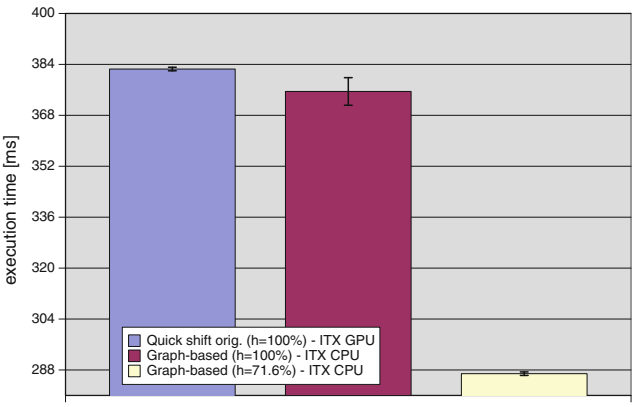
The first set of measurements (Fig. 6a) includes the execution of the Graph-based segmentation algorithm on the Laptop's processor and the optimized Quick shift algorithm running on the GPU of the Mini-ITX computer. These combinations of algorithm and executing platform were found to be the fastest and within real-time constraints. For both of these cases, the complete image was processed (horizon set to 100 %) and only the relevant part according to the test image (horizon set to 71.6 %).

The optimal tradeoff between total number and individual size of segments was found to be around 190 segments. For the case of the Graph-based algorithm, the threshold for segment splitting was $t = 140$. For the Quick shift algorithm, the thresholds used were $\sigma = 2$ and $\tau = 8$.

For completeness, a second batch of measurements was performed (Fig. 6b), comparing the previous measurements to the execution of the unmodified Quick shift algorithm running on the same GPU and the Graph-based algorithm running on the Mini-ITX dual core CPU. On these conditions, the computation times of the segmentation step alone already exceed the tolerance for stable

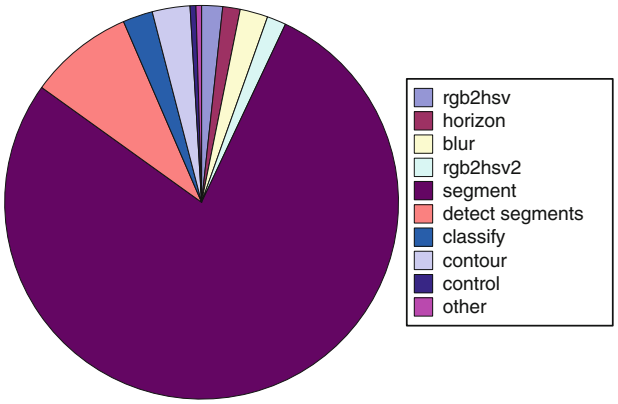


(a) Optimal choices of segmenation algorithm and executing platform

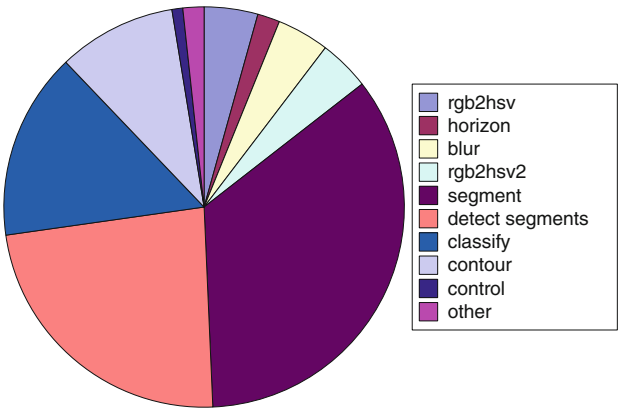


(b) Other combinations of segmenation algorithm and executing platform

Fig. 6 Computation time for the segmentation step, for different algorithms and executing platforms



(a) Computation time of algorithm as executed on the Core i5 processor present on the Laptop



(b) Computation time of algorithm as executed on the GPU present on the Mini-ITX embedded computer

Fig. 7 Execution time for each step relative to the whole algorithm executed on a Laptop (**a**) and the Mini-ITX GPU (**a**): *rgb2hsv* conversion of original image to HSV color-space, *blur* median-blur applied to input image, *horizon* horizon detection, *rgb2hsv2* conversion to HSV of smoothed image, *segment* image segmentation algorithm (Graph-based on CPU, Quick shift on GPU), *detect segments* construction of map of labeled pixels and computation of mean and covariance for each segment, *classify* ground model construction and individual segment classification, *contour* binary mask computation from classified segments and road contour, *control* control law extraction from road middle-points

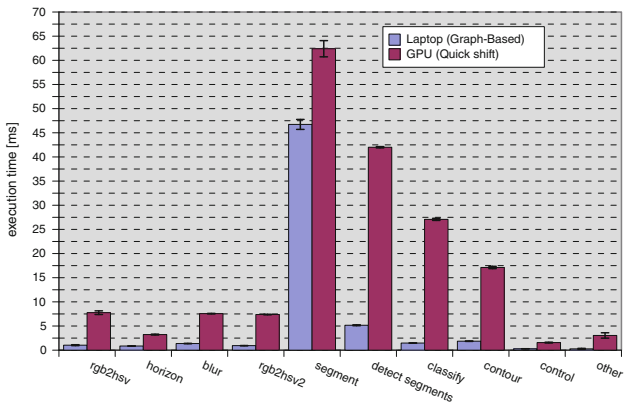


Fig. 8 Absolute execution time for each step of the algorithm, in both platforms

control of the mobile robot, which should be in the order of 4 Hz for the complete algorithm.

Finally, by comparing the execution times (Fig. 6a and b) between the original and optimized (presented in this work) versions of the Quick shift segmentation algorithm running on GPU, it can be seen that with these optimizations a speedup of approximately 5.6× is achieved.

9.4 Complete algorithm performance

The proposed algorithm, as described in previous sections, can be decomposed in several steps. In this section, the time consumed by each step is presented. In Fig. 7 the computational time of each step relative to the total iteration is presented. In Fig. 8 the absolute computational time consumed by each step is shown. In both cases, two series of timings were measured, corresponding to the segmentation algorithm and parameters choice which maximize execution speed, for the two computing platforms considered (Laptop and embedded GPU).

These time measurements correspond to the mean value of $N = 66$ repetitions over the same input image. Standard deviation is presented in Fig. 8 for each step. The total mean time consumed by one iteration of the algorithm was 59.9841 ms (std. dev. 1.2011 ms) as executed on the Laptop, and 179.1447 ms (std. dev. 1.744 ms) as executed on the GPU.

10 Conclusion and further work

In this paper, a new real-time image-based monocular path detection method is presented. It does not require camera calibration and works on semi-structured outdoor paths. The core of the method is based on segmenting images and classifying each segment to infer a contour of navigable space. This method allows a mobile robot equipped with a monocular camera to travel through different naturally delimited paths. The road contour shape can be used to calculate the forward and steering speed of the robot. To achieve real-time computation necessary for on-board execution in mobile robots, the image segmentation is implemented on a low-power embedded GPU.

When analysing the road detection capability of the method, it proved to be very robust handling difficult situations associated with unstructured outdoor roads. Using an example area from which only a subset of modes with enough coverage of the rectangular ROI is used, outliers

can be ignored without requiring precise placement of this region. This also allows the usage of the camera as the sole sensor, in contrast to using a laser scanner for precise road example region detection. Finally, using the segmentation itself and then merging similar segments included in the ROI, similar results to clustering using k-means (as used in related works) can be achieved without requiring to determine the ammount of classes k in advance.

For evaluation of the closed-loop performance of the algorithm, a simple control law was implemented which aims to maintain the robot in the middle of the detected road. By positioning the robot in different initial orientations with respect to the road, the correct behavior of the control law and the method as a whole was assessed. The simplicity of this control law does not require the camera to be calibrated.

Regarding the segmentation step, several algorithms were evaluated on different platforms. For the case of a modern CPU, the Graph-based segmentation method proved to be the fastest. For the case of an embedded computer suited for small mobile robots, this method is not fast enough. An embedded GPU allowed the implementation of the Quick shift algorithm with execution times within required constraints and comparable to execution times of modern CPUs.

As a part of the proposed road detection, we introduced a horizon detector algorithm, with some differences with respect to previous works. The proposed horizon detection method is very robust and can handle difficult scenarios where there is a high contrast sky and background objects like trees. The implementation of this step is also quite simple and fast, which is important to maintain real-time execution constraints. Furthermore, by detecting the horizon, the rest of the algorithm can process only the image below, further speeding up image processing.

10.1 Future work

While the proposed method demonstrated to be robust and stable, there are several aspects that could be improved and further analyzed.

First, while the HSV color space was used due to the separation of the chrominance and luminance information in different channels, there are several other color spaces which perform this separation. One of these corresponds to the LAB space, which has the particular benefit of expressing chrominance in linear (as opposed to the circular Hue channel) coordinates, therefore, possibly simplifying mean and covariance computations.

Also related to color space is the fact that, for HSV in particular, when very low Saturation is present, the Hue channel can contain any value while still producing the same perceived color. A similar effect occurs for very low Value. This fact should be taken into account when comparing two models, for example, by modifying thresholds in-place based on mean values of individual channels. Addressing this issue may increase independence of thresholds used for classification from environment appearance.

Regarding the control law used, a very simple method was proposed in this work. However, it is possible to implement more sophisticated methods, for example, by fitting the list of middle road points with a curve. Also, if camera calibration can be assumed to be always available, the middle points of the road could be reprojected into world coordinates and, therefore, use a control law that precisely takes into account camera position, robot speeds, etc. Implementing such method could also permit more rigorous numerical evaluation of control law stability and performance.

To further improve the algorithm performance when executing on the embedded GPU, several steps could possibly improved by also executing those on the GPU. This corresponds to the case of the detection of segments, contour extraction and classification. Particularly, this last step could be performed completely in parallel for each segment.

Acknowledgments This work has been supported by the Ministry of Education of the Czech Republic by project 7AMB12AR022 and by Ministry of Science of Argentina by project ARC/11/11.

References

1. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robust features (SURF). Comput. Vis. Image Underst. **110**(3), 346–359 (2008)
2. Blas, M., Agrawal, M., Sundaresan, A., Konolige, K.: Fast color/texture segmentation for outdoor robots. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008. IROS 2008, pp. 4078–4085. IEEE, New York (2008)
3. Bonin-Font, F., Ortiz, A., Oliver, G.: Visual navigation for mobile robots: a survey. J. Intell. Robot. Syst. **53**(3), 263–296 (2008)
4. Borenstein, J., Koren, Y.: Obstacle avoidance with ultrasonic sensors. IEEE J. Robot. Autom. **4**(2), 213–218 (1988)
5. Chang, C., Siagian, C., Itti, L.: Mobile robot monocular vision navigation based on road region and boundary estimation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012. IEEE, New York (2012)
6. Chen, Z., Birchfield, S.: Qualitative vision-based path following. IEEE Trans. Robot. **25**(3), 749–754 (2009)
7. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. IEEE Trans. Pattern Anal. Mach. Intell. **24**(5), 603–619 (2002). doi:10.1109/34.1000236
8. Dahlkamp, H., Kaehler, A., Stavens, D., Thrun, S., Bradski, G.: Self-supervised monocular road detection in desert terrain. In: Proceedings of Robotics: Science and Systems (RSS) (2006)
9. DeSouza, G., Kak, A.: Vision for mobile robot navigation: a survey. IEEE Trans. Pattern Anal. Mach. Intell. **24**(2), 237–267 (2002)

10. Ettinger, S., Nechyba, M., Ifju, P., Waszak, M.: Vision-guided flight stability and control for micro air vehicles. Adv. Robot. **17**(7), 617–640 (2003)
11. Felzenszwalb, P., Huttenlocher, D.: Efficient graph-based image segmentation. Int. J. Comput. Vis. **59**(2), 167–181 (2004)
12. Fulkerson, B., Soatto, S.: Really quick shift: Image segmentation on a gpu. In: ECCV2010 Workshop on Computer Vision on GPUs (CVGPU2010) (2010)
13. Fulkerson, B., Vedaldi, A.: Really quick shift: Image segmentation on a gpu. http://vision.ucla.edu/∼brian/gpuquickshift.html, version 0.2 (2010)
14. Henry, P., Krainin, M., Herbst, E., Ren, X., Fox, D.: RGB-D mapping: using depth cameras for dense 3d modeling of indoor environments. In: The 12th International Symposium on Experimental Robotics (ISER), vol. 20, pp. 22–25 (2010)
15. Kaliyaperumal, K., Lakshmanan, S., Kluge, K.: An algorithm for detecting roads and obstacles in radar images.IEEE Trans. Veh. Technol. **50**(1), 170–182 (2001)
16. Kong, H., Audibert, J., Ponce, J.: General road detection from a single image. IEEE Trans. Image Process. **19**(8), 2211–2220 (2010)
17. KoÅanar, K., KrajÃnk, T., PÅŹeuÄil, L.: Visual topological mapping. In: Bruyninckx, H., Preucil, L., Kulich, M. (eds.) European Robotics Symposium 2008, Springer Tracts in Advanced Robotics, vol. 44, pp. 333–342. Springer, Berlin (2008)
18. Krajník, T., Faigl, J., Vonásek, V., Košnar, K., Kulich, M., Přeučil, L.: Simple yet stable bearing-only navigation. J. Field Robot. **27**(5), 511–533 (2010)
19. Kuhnl, T., Kummert, F., Fritsch, J.: Monocular road segmentation using slow feature analysis. In: Intelligent Vehicles Symposium (IV), 2011 IEEE, pp. 800–806, IEEE, New York (2011)
20. Lowe, D.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vis. **60**(2), 91–110 (2004)
21. May, S., Werner, B., Surmann, H., PervÃűlz, K.: 3d time-of-flight cameras for mobile robotics. In: IROS, pp. 790–795. IEEE, New York (2006)
22. Moghadam, P., Starzyk, J., Wijesoma, W.: Fast vanishing-point detection in unstructured environments. IEEE Trans. Image Process. **21**(1), 425–430 (2012)
23. Neto, A., Victorino, A., Fantoni, I., Zampieri, D.: Robust horizon finding algorithm for real-time autonomous navigation based on monocular vision. In: 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), 2011, pp. 532–537. IEEE, New York (2011)
24. Nieto, M., Salgado, L.: Real-time vanishing point estimation in road sequences using adaptive steerable filter banks. In: Advanced Concepts for Intelligent Vision Systems, pp. 840–848. Springer, Berlin (2007)
25. Parzen, E.: On estimation of a probability density function and mode. Ann. Math. Stat. **33**(3), 1065–1076 (1962)
26. Pedre, S., De Cristóforis, P., Caccavelli, J., Stoliar, A.: A mobile mini robot architecture for research, education and popularization of science. J. Appl. Comput. Sci. Methods; Guest Editors: Zurada, J, Estevez p 2 (2010)
27. Rasmussen, C.: Grouping dominant orientations for ill-structured road following. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004, vol. 1, pp. I-470. IEEE, New York (2004)
28. Rasmussen, C., Lu, Y., Kocamaz, M.: Appearance contrast for fast, robust trail-following. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009, pp. 3505–3512. IEEE, New York (2009)
29. Santosh, D., Achar, S., Jawahar, C.: Autonomous image-based exploration for mobile robot navigation. In: IEEE International Conference on Robotics and Automation, 2008. ICRA 2008, pp. 2717–2722. IEEE, New York (2008)
30. Šegvić, S., Remazeilles, A., Diosi, A., Chaumette, F.: A mapping and localization framework for scalable appearance-based navigation. Comput. Vis. Image Underst. **113**(2), 172–187 (2009)
31. Sezgin, M. et al.: Survey over image thresholding techniques and quantitative performance evaluation. J. Electron. Imaging **13**(1), 146–168 (2004)
32. Sheikh, Y., Khan, E., Kanade, T.: Mode-seeking by medoidshifts. In: IEEE 11th International Conference on Computer Vision, 2007. ICCV 2007, pp. 1–8. IEEE, New York (2007)
33. Surmann H, Lingemann K, Nüchter A, Hertzberg J (2001) A 3d laser range finder for autonomous mobile robots. In: Proceedings of the 32nd ISR (International Symposium on Robotics), vol 19, pp 153–158
34. Ulrich, I., Nourbakhsh, I.: Appearance-based obstacle detection with monocular color vision. In: Proceedings of the National Conference on Artificial Intelligence, 1999, pp. 866–871. AAAI Press, Menlo Park; MIT Press, Cambridge (2000)
35. Vedaldi, A., Soatto, S.: Quick shift and kernel methods for mode seeking. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) Computer Vision ECCV 2008. Lecture Notes in Computer Science, vol. 5305, pp 705–718. Springer, Berlin (2008)
36. Wang, Y., Fang, S., Cao, Y., Sun, H.: Image-based exploration obstacle avoidance for mobile robot. In: Control and Decision Conference, 2009. CCDC'09, Chinese. pp. 3019–3023. IEEE, New York (2009)
37. Yanqing, W., Deyun, C., Chaoxia, S., Peidong, W.: Vision-based road detection by monte carlo method. Inf. Technol. J. **9**(3), 481–487 (2010)

Author Biographies

Pablo De Cristóforis is a PhD student at the Robotics and Embedded Systems Laboratory, Computer Science Department, Faculty of Exacts and Natural Sciences, University of Buenos Aires, Argentina. He received his Computer Science degree from that University in 2007. His research interests include vision-based robot navigation and mobile robotics designs.

Matías Nitsche is a PhD student at the Robotics and Embedded Systems Laboratory, Computer Science Department, Faculty of Exacts and Natural Sciences, University of Buenos Aires, Argentina. He received his Computer Science degree from that University in 2008. His research interests include vision-based robot navigation, GPGPU and aerial robots.

Tomáš Krajník received the PhD degree in Artificial Intelligence and Bio-cybernetics from the Czech Technical University, Prague, Czech Republic, in 2012. His research interests include autonomous navigation, reasoning in mobile robotics, and aerial robots.

Marta Mejail is the head of Robotics and Embedded Systems Laboratory. She is a associated professor at Computer Science Department, Faculty of Exacts and Natural Sciences, University of Buenos Aires, Argentina. Her research interests include image processing, pattern recognition and real-time applications.