Discrete-Continuous Clustering for Obstacle Detection Using Stereo Vision

Robert Bichsel[1], Paulo Vinicius Koerich Borges[2]

***Abstract*— Efficient obstacle detection is a key requirement for safe robot navigation. We consider the operation of autonomous vehicles in structured industrial environments. In such scenarios, an usual way to perform obstacle detection is to generate an estimate of the ground and detect elements that are on the path of the vehicle, using the ground as a spatial reference. For this task, 3D occupancy grids are a well-know solution. In this work we extrapolate the concept of 3D grids by considering a discrete-continuous representation of the environment. The discrete nature lies in a 2D grid parallel to the ground whereas the continuous aspect represents the height of each cell in the grid. This framework allows for very efficient clustering, for which we also propose a novel algorithm to cluster potential obstacles. Experiments on an autonomous ground vehicle illustrate the applicability of the method.**

***Index Terms*— Obstacle detection, stereo-vision, autonomous vehicles.**

I. INTRODUCTION

The detection of obstacles is an essential task in the operation of autonomous vehicles. The topic has also gained increased attention as part of advanced driving assistance systems (ADAS), which provide safety and comfort technology for human drivers. For robot navigation, obstacle detection is the key element for collision avoidance and local path planning. Proposed solutions range from simple range sensing methods that detect an obstacle and stop the robot, to more complex algorithms that determine the location, trajectory and size of the obstacle, creating strategies to navigate around it.

In the context of ground vehicles (both manned or autonomous), obstacle detection has been tackled with different sensors, such as lasers [1], [2], radar [3], [4], and cameras [5], [6]. One advantage of cameras is their relative affordability and lightweight, aside from being a passive sensor. Cameras can also be setup as a stereo pair, providing 3D information. In this work we propose a novel obstacle detection algorithm using the data available from stereo, combining information from the 2D image and the 3D point cloud generated from stereo. The main application for the proposed method is autonomous vehicle navigation in industrial environments, considering indoor and outdoor scenarios. In this context, the definition of an obstacle is kept simple and general: an obstacle is any object that can interfere with the vehicle path causing a potential collision. Therefore, detecting obstacles allows for (non)passable areas to be identified and path planning can be recalculated accordingly or emergency stops can be activated.

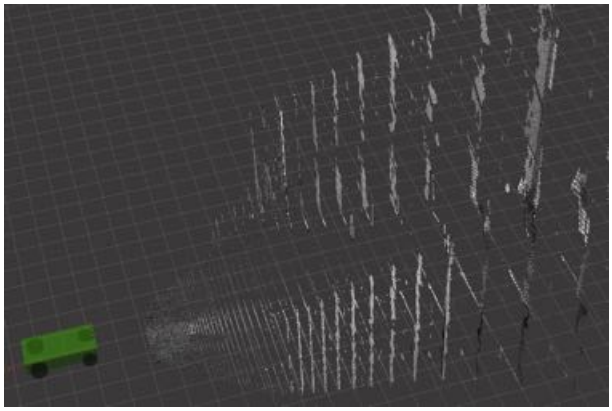


Fig. 1. Sliced structure of 3D range data.

One of the main challenges for obstacle detection based on stereo vision is the quality of the input data. The 3D point clouds are often noisy and contain voids where no matches between the left and right images are found. Depending on the matching algorithm, the 3D data is discretized in the z-direction (i.e. perpendicular to the camera plane) and only slices of data are available for obstacle detection [7], as illustrated in Figure 1. There is a broad variety of algorithms which detect obstacles from stereo vision. Talukter *et al.* [8] require an obstacle to have a steep surface. For every range point, they search for other range points consistent with this definition (in a cone-shaped search space) and thereby develop obstacle surfaces. The definition of an obstacle by a steep surface is also used by Oniga *et al.* [9], who calculate a Digital Elevation Model (DEM) and use the density of range points per DEM cell as a metric to measure the local slope. This allows for the distinction between flat surfaces (roads, traffic isles) and obstacles. A combination of range information and optical flow has also been used to extract obstacles from stereo images [10]. Both range and optical flow information are projected onto a polar grid, then the grid cells are grouped together to identify obstacles, their orientation and their speed. Perrone *et al.* [11] propose an approach where range information and its temporal behavior is extracted by the means of a customized feature detection and tracking module. Detected features are clustered and tracked, so that the obstacle is fully defined. A recent survey by Bernini *et al.* [12] discusses in more detail related

This work was supported by the CSIRO Digital Productivity Flagship.

[1]Robert Bichsel is with the Autonomous Systems Lab at ETH Zurich, Zurich, Switzerland, aralph@ethz.ch

[2]Paulo Vinicius Koerich Borges is with the Autonomous Systems Lab at CSIRO Computational Informatics, Brisbane, Australia. Paulo Borges is also an Adjunct Senior Lecturer at the School of Information Technology and Electrical Engineering, University of Queensland, vini@ieee.org

algorithms.

In contrast to the methods presented above, we consider a discrete-continuous representation of the world in order to identify obstacles. The discrete representation corresponds to a 2D occupancy grid [13] in the $x-z$ plane based on the estimation of a DEM, whereas the vertical component is the continuous variable representing height in the y direction. An illustration of this discrete-continuous framework is shown in Figure 2. This setup presents significant advantages compared to the traditional 3D occupancy grid approaches, in particular regarding the efficient clustering of objects. The continuous nature in the vertical direction enhances the accuracy of the representation of the measured data. Objects are clustered vertically into blocks (of continuos-variable length) representing an obstacle. Our experiments indicate that this strategy reduces significantly the false-positive and false-negative detection error rates of small objects, improving the overall performance of the algorithm compared to the full 3D grid representation. We implement the algorithm on an autonomous all-terrain vehicle (John Deere's Gator) and perform experiments in structured and unstructured areas. We also perform closed-loop experiments feeding the output of the proposed method into the path planning module for obstacle avoidance in autonomous operations.

This paper is organized as follows. Section II presents some background information required for understanding the contributions presented. Section III describes the obstacle detection algorithm. Section IV presents experiments, followed by relevant conclusions in Section V.

II. Background

The proposed method can be summarized as follows:

- Initially, a ground estimation is generated.
- Then, obstacle are clustered in a continuous-discrete representation space.
- Finally, the confidence of each cluster is evaluated, eliminating low likelihood objects.

This section discusses the first step, describing the proposed method for the creation of a DEM and the estimation of a ground plane. The DEM is in essence a down-sampled version of the large amount of range points, which largely reduces for the computational cost. After the DEM, points that belong to the ground are estimated.

A. Digital Elevation Model

A DEM is a discretized representation of a surface based on elevation data. For the creation of a DEM, the robot's ground plane ($x-z$ plane of the detection frame) is divided into a square grid. Grid parameters such as length, width and resolution depend on the environment, size of the vehicle and necessary localization precision. In our implementation we use length = 40m, width = 13m, resolution = 0.1m. The range points are assigned to these squares according to their x- and z-coordinates. The height of the highest point (largest y-value) in a cell is chosen as the cell's height h_{cell}, as depicted in Figure 2. Finally, a new point cloud is created, containing a point for each non-empty grid cell. The coordinates of such a point is given by the x- and z-coordinates of the cell's center and its height as y-component. Figure 3 illustrates the generation of a the DEM from the 3D point cloud.

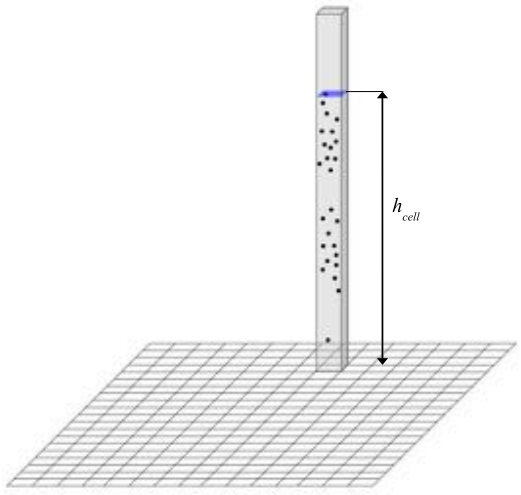
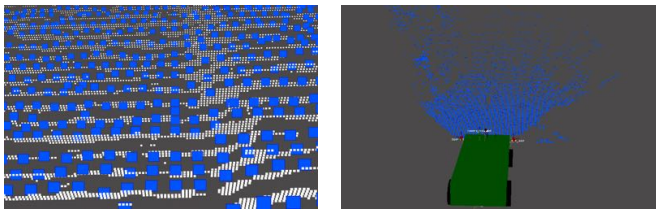


Fig. 2. Schema of the DEM construction: The highest point in falling into a cell defines the cell's height h_{cell}.



(a) Comparison between the raw point cloud (white) and the DEM point cloud (blue).

(b) View of the DEM point cloud.

Fig. 3. Result of the DEM construction. Note the significantly decreased number of points.

B. Ground Point Extraction

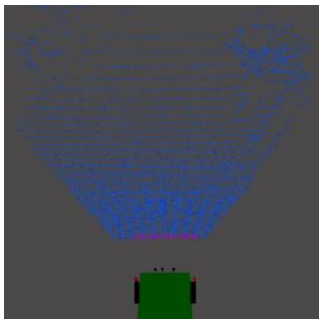
The ground plane estimation is based on a set of 3D points as input. One option for ground plane estimation is to use a Random Sampling Consensus (RANSAC) algorithm [14] on the entire set of DEM points. While this alternative works satisfactorily most of the time, occasionally slanted surfaces next to the road can be wrongly estimated as ground planes. Therefore, we restrict the input of the RANSAC algorithm to a subset of the DEM points, henceforth called ground points.

1) Ground Point Algorithm: The algorithm that identifies the ground points is based on two assumptions:

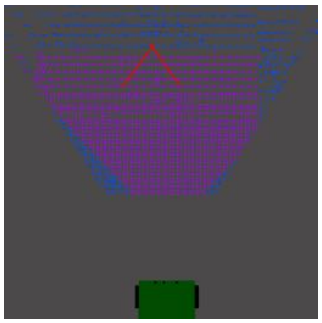
- The vehicle stands on the ground that can be seen in front of it. Hence, the closest points to the vehicle are part of the ground plane. In a structured, industrial environment this assumption usually holds. It does not hold whenever an obstacle is very close to the vehicle, which ideally should never be the case as obstacles are detected and avoided in this framework.
- The slope of the ground is bounded locally by a fixed threshold.

Exploiting these assumptions, the following algorithm is able to detect ground points:

- **Initialization**: The points in the first R rows closest to the vehicle (Figure 4a) are declared as ground points if their height is within $\pm\lambda$ m from the vehicle's ground plane ($z = 0$ in detection frame). Experimental evaluation indicates that effective values for the parameters are $R = 2$ and $\lambda = 0.2$m.
- **Development**: For every point in a successive row (henceforth called candidate point), the ground points in the previous rows are assessed. In a triangular search space (oriented backwards, opening $90°$ degrees as depicted in Figure 4b), the first non-empty row (the first row with ground points in it) is identified. The ground points in this row are divided into two groups: one where the slope between the candidate point and the ground point is below a given threshold (positive vote), and one where the slope is above the threshold (negative vote). If the positive votes outnumber the negative ones, the candidate point is declared as a ground point.

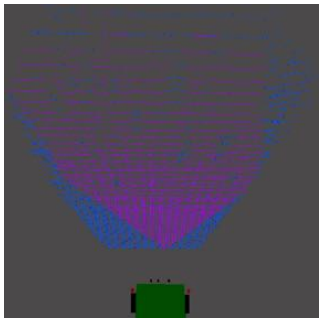


(a) Initial R rows of ground points (purple) in the DEM (blue). In this case, $R = 2$.

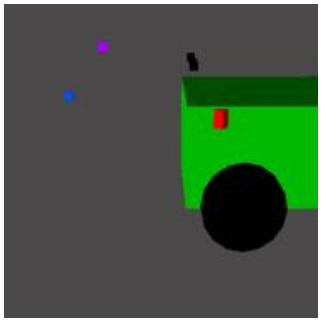


(b) Development of the gound points (purple) from the DEM (blue) using a backwards-oriented search triangle(red).

Fig. 4. Illustration of the detection algorithm for ground points.



(a) When only the first line is used to initialize the algorithm, the choice of ground points is often suboptimal.



(b) Erronneous ground point when height of initialization is not restricted.

Fig. 5. Inadequate choice of ground points due to poor initialization.

C. Ground Plane Estimation

Although roads usually present some degree of curvature, a linear model is a commonly used approximation [15], [16], [17] that is very effective in structured environments and that has the advantage of a substantially lowered computational complexity. A plane is defined by:

$$\mathbf{n}_{plane} \cdot \mathbf{x} + d_{plane} = 0 \tag{1}$$

where $\mathbf{n}_{plane}$ stands for the normal vector of the plane, $\mathbf{x}$ is a generic point in 3D space and d denotes the negative distance of the plane to the origin of the coordinate system.

The parameters $\mathbf{n}_{plane}$ and d can be determine with the RANSAC algorithm, whose input are the ground points extracted from the DEM (Section II-B).

To remove outliers due to sudden jerks, a median filter of length N_{med} is applied in the time domain to the coefficients of the estimated ground plane (sliding window filter). This smoothens the transitions between the ground plane estimations of consecutive frames.

III. Obstacle Detection Algorithm

The proposed obstacle detection algorithm initially creates an occupancy map based on the 3D range data. Then, the occupied space in the map is clustered and tracked over time. Finally, isolated "outlier" clusters are filtered out using a Bayesian approach, such that wrong matches are eliminated. The algorithm is described in detail in this section.

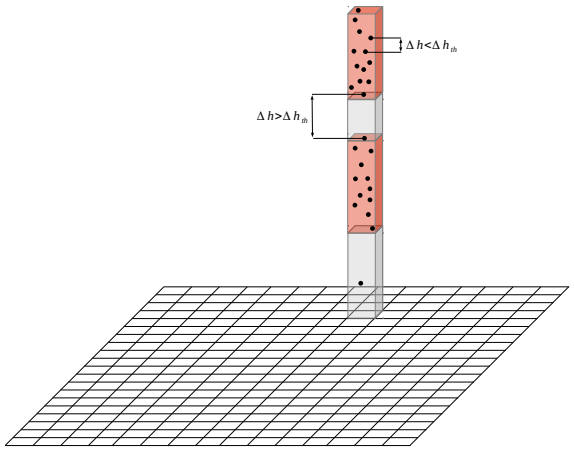
A. Column Occupancy Map

The 3D range data is transformed into a 3D occupancy map according to the following rules:

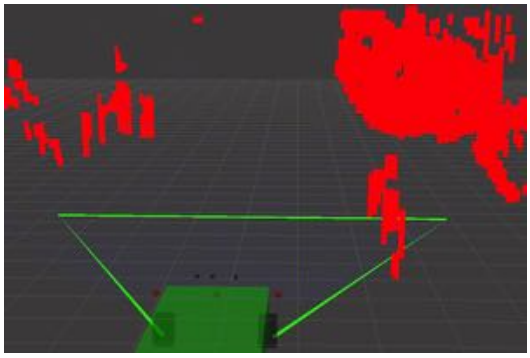
- The same grid used for the DEM (Section II-A) is also used here. It segments the 3D space into columns, with $\{y \in \mathbb{R} \mid -\infty < y < +\infty\}$.
- The points contained in each column (x and z coordinate within the square) are sorted according to their height (y-coordinate).
- The points in each column are assessed from bottom to top: Segments of the column where the distances (in y-direction) between two successive δh points do not exceed the threshold h_{th} are defined as occupied (cf. Figure 6a). However, only points with a minimum height above the ground plane, $h > h_g$, are considered. This ensures that the ground is not defined as occupied.

From this procedure, a column occupancy map with segments of occupied columns is generated, as illustrated in Figure 6b.

The column occupancy map jointly downsamples and clusters the data points. This converts a cloud of points into a semantic representation, where the label "occupied" is attributed to densely populated areas within the point cloud. Once the column occupancy map is created, a further analysis evaluates whether the detected occupied areas are coherent, in order to reduce false-positives. Intuitively, a small speckle of occupied space, for example, with no neighbours in its vicinity and not consistent over time is most likely a false positive due to a faulty match. A large area of occupied space that is visible in multiple successive frames, on the other hand, is likely to represent an actual obstacle.



(a) Schema of the occupancy map creation.



(b) The result of an occupancy map.

Fig. 6. Schema and result of the occupancy map creation.

Therefore, obstacles are accessed based on properties such as size and occupancy of the neighbourhood.

The method proposed here takes advantage of the sliced nature of the data (Figure 1) and consists of four steps: (i) Clustering adjoining columns of occupied space within a slice, (ii) tracking them temporally, (iii) determining the vicinity of clusters by further clustering them into so-called *megaclusters*, and (iv) finally filtering out elements with a high likelihood of being false-positives. These four steps are described in the following sections.

B. Clustering Within a Slice

For the first clustering step, we can exploit the sliced nature of the data (and hence of the column occupancy map). In each slice, and hence in 2D, adjoining occupied columns are clustered together and their size is calculated. The size of a cluster is given by its volume (the area multiplied by the thickness of the slice (10cm)).

To this end, an (initially empty) set of clusters $\mathcal{Q}$ is defined. An individual cluster q ($q \in \mathcal{Q}$) contains a set of occupied elements. Let us focus on one arbitrary slice: The j-th element in column i, $e_{i,j}$, is characterized by the height of its lower and upper boundaries, $y_{low,e_{i,j}}$ and $y_{high,e_{i,j}}$, as illustrated in Figure 7. For each slice, adjoining elements are clustered according to Algorithm 1:

Algorithm 1: Clustering data within a slice.

for $i = 0$ **to** *number of columns in slice - 1* **do**
  for $j = 0$ **to** *number of elements in column i* **do**
    if $e_{i,j}$ *is not in a cluster yet* **then**
      Add new cluster containing only $e_{i,j}$ to $\mathcal{Q}$.
    for $k = 0$ **to** *number of elements in column i+1* **do**
      if $e_{i,j}$ *and* $e_{i+1,k}$ *touch* **then**
        Add $e_{i+1,k}$ to cluster of $e_{i,j}$ or merge the two clusters.

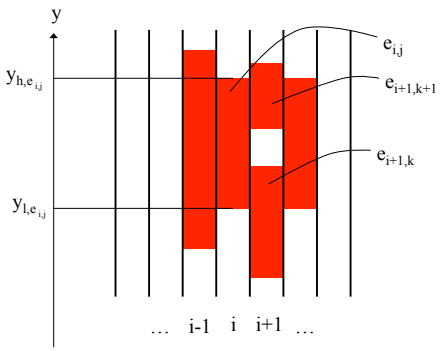


Fig. 7. Illustration of the of the occupied elements within a slice.

where *touch* is `true` if

$$(y_{low,e_{i,j}} \leq y_{high,e_{i+1,k}}) \cap (y_{high,e_{i,j}} \geq y_{low,e_{i+1,k}}) \neq set$$

C. Tracking

Considering stereo matching algorithms that slice the data (and also the clusters as a consequence), it is hard to evaluate the consistency of an obstacle in z-direction (depth) when slices of data are missing. A more accurate representation can be achieved by temporally tracking the obstacles with the help of odometry information. Figure 9 shows a comparison between a tracked and non-tracked representation of clusters, illustrating the more dense information in the tracked version. Tracking is done by simply adding the odometry information to the position of the previously measured data.

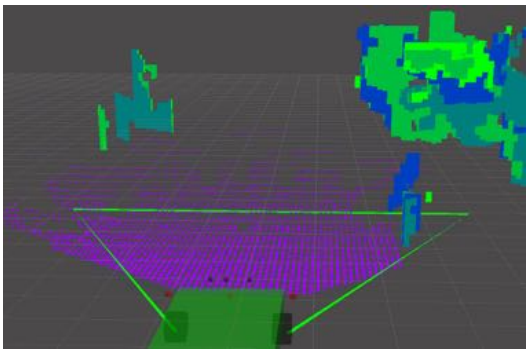
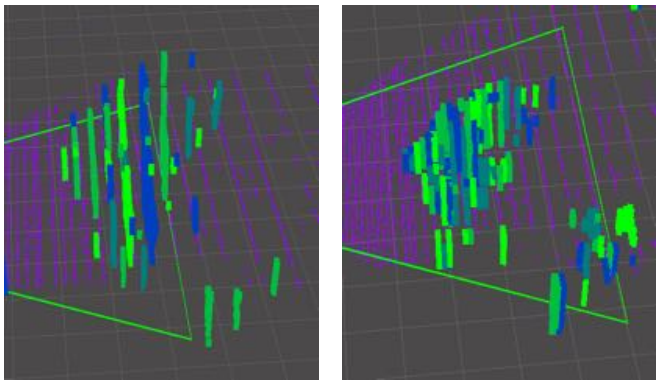


Fig. 8. The column occupancy map grouped into clusters. The different colors are merely there to distinguish the clusters and have no further meaning.



(a) No tracking. (b) Tracking for five iterations.

Fig. 9. Comparison between non-tracked and tracked clusters.

D. Clustering into Megaclusters

Bulks of clusters that are close to each other are further grouped into megaclusters. The *vicinity* of a cluster is defined as all the clusters belonging to the same megacluster. Due to the grid structure of the clusters, a Chebyshev distance d_{Cheb} rather than an Euclidean distance is used to measure the proximity of two clusters as

$$d_{Cheb} = max(|p_x - q_x|, |p_y - q_y|, |p_z - q_z|) \tag{2}$$

where p and q are two points of interest of coordinates x, y and z. Two obstacles are said to be in proximity of each other if the Chebyshev distance between at least one pair of points (p, q) (p from 'obstacle 1' and q from 'obstacle 2') is below a threshold d_{th}. To cluster the obstacles into megaclusters, we define an (initially empty) set of megaclusters $\mathcal{M}$. Each megacluster m in $\mathcal{M}$ contains a set of clusters. The clusters $\mathcal{Q}$ are grouped into megaclusters according to Algorithm 2.

Algorithm 2: Cluster proximity analysis and merging into megaclusters.

for $i = 0$ **to** *number of clusters* **do**
  Choose cluster q_i from $\mathcal{Q}$;
  for $j = 0$ **to** *number of megaclusters* **do**
    Check which megaclusters q_i belongs to $(d_{Chebyshev} < d_{th})$.
  if q_i *belongs to 2 or more megaclusters* **then**
    Merge the respective megaclusters and add q_i
  else if q_i *belongs to 1 megacluster* **then**
    Add q_i
  else
    for $j = i$ **to** *number of clusters* $\mathcal{Q}$ **do**
      Check which clusters q_i belongs to $(d_{Chebyshev} < d_{th})$
    if q_i *belongs to 1 or more clusters* **then**
      Make a new megacluster with all the clusters contained in it
    else if $q_i >$ *size threshold* **then**
      Make a new megacluster with only q_i in it

E. Filtering

In a final step, clusters of low confidence are filtered out. For this evaluation, the size of a cluster and the accumulated size of other clusters in its vicinity (in its megacluster) are considered. Small and isolated obstacles are very uncommon in real scenarios (an obstacle usually does not float in mid-air, but is rather attached to a pole or a branch). For this reason, small and isolated obstacles are discarded.

A Bayesian approach is used to evaluate the cluster confidence value. The cluster size serves as a basis for the prior. The accumulated size of the neighbors (given by the size of the megacluster) is used as a measurement for the likelihood. The posterior can be calculated by:

$$\begin{aligned} P(O|V_{acc}) &= \frac{p(V_{acc}|O) \cdot P(O)}{p(V_{acc})} \\ &= \frac{p(V_{acc}|O) \cdot P(O)}{p(V_{acc}|O) \cdot P(O) + p(V_{acc}|\bar{O}) \cdot P(\bar{O})} \end{aligned} \tag{3}$$

where P represents a probability and p a probability density function. O stands for the event "obstacle", $\bar{O}$ for 'no obstacle" and V_{acc} is the accumulated size (volume) of the clusters in the megacluster. The prior is defined as:

$$P(O) = \frac{V_{cl}}{V_{th}}; \tag{4}$$

where V_{cl} is the size of the cluster and V_{th} is a threshold above which a cluster is always classified as obstacle. Since the events O and $\bar{O}$ are mutually exclusive, $P(\bar{O}) = 1 - P(O)$.

Based on experimental evaluation, $p(V_{acc}|O)$ and $p(V_{acc}|\bar{O})$ are modelled as Gaussian distributions. Since V_{acc} is a purely positive quantity, a truncated Gaussian distribution is used, given by

$$f(x; \mu, \sigma, a, b) = \frac{\frac{1}{\sigma}\phi(\frac{x-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})} \tag{5}$$

where $\phi(\xi) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\xi^2\right)$ is the probability density function of the Gaussian distribution and Φ is its cumulative distribution function. The parameters a and b denote the lower and upper bounds, and μ and σ denote the mean and the variance, respectively. In this case, the distribution $p(V_{acc}|O)$ is centered around μ with variance σ and a lower bound of 0, whereas $p(V_{acc}|\bar{O})$ is centered around 0, but has the same standard deviation σ and also a lower bound of 0, as exemplified in Figure 10. Hence, $p(V_{acc}|O)$ and $p(V_{acc}|\bar{O})$ can be found by

$$p(V_{acc}|O) = \frac{\frac{1}{\sigma}\phi\left(\frac{V-\mu}{\sigma}\right)}{1 - \Phi\left(\frac{\mu}{\sigma}\right)} \tag{6}$$

$$p(V_{acc}|\bar{O}) = \frac{2}{\sigma}\phi\left(\frac{V}{\sigma}\right) \tag{7}$$

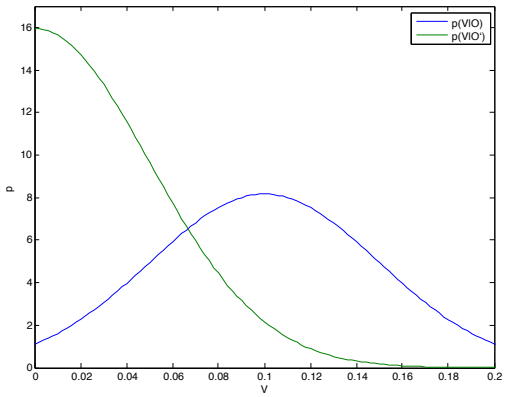


Fig. 10. The truncated gaussian distributions for $p(V_{acc}|O)$ (blue) and $p(V_{acc}|\bar{O})$ (green).

With all the quantities defined, each cluster is assessed according to Algorithm 3:

Algorithm 3: Assessment of the cluster quality.

for $i = 0$ **to** *number of clusters* **do**
  if $V > V_{th}$ **then**
    cluster q_i is an obstacle
  else
    if $P(O|V_{acc}) > 0.5$ **then**
      m_i is an obstacle
    else
      m_i is discarded

A good parameter set was found empirically: $h_g = 0.3m$, $\Delta h_{th} = 0.2m$, $d_{th} = 0.7m$, $\mu = 0.1m^3$, $\sigma = 0.05m^3$ and $V_{th} = 0.07m^3$. The results of this filtering can be seen in Figure 11. Among these parameters, h_g = is the ground height threshold, Δh_{th} is height threshold between consecutive points in a vertical column, d_{th} is the distance threshold for filtering, V_{th} is the volume threshold for a megacluster to be considered an obstacle and μ and σ correspond to the mean and standard deviation, respectively.

IV. Experiments

This section presents the results of the proposed algorithm in an open-loop scenario as well as in the closed-loop system, with the algorithm implemented on the Gator.

A. Practical Considerations

The algorithm is implemented in C++, using the OpenCV Library[1], and the Robotics Operating System (ROS)[2]. The stereo imaging system was formed by two monochromatic Point Grey Grasshopper cameras with resolution of 800×600 pixels fitted with Kowa 2/3" lenses with 5mm focal length and aperture F1.8. The system was setup on a rig with a baseline of 25 cm, as illustrated in Figure 12b.

The stereo pair was mounted on a John Deere's Gator (Figure 12a), an electric utility vehicle, and driven in an industrial park in Australia. The test environment contains both urban and rural characteristics, with trees, roads, sidewalks, curbs, dense buildings and bushland. In this area, the performance of the algorithm was assessed based on both static and moving obstacles. In all scenarios, the algorithm was tested under different lighting conditions considering overcast and sunny weather. The computation was executed on a laptop computer (Intel Core i7, 2.7 GHz Octa-core) and takes 20-65 ms per frame, depending on the amount of ground points (15-25 ms for the calculation of the DEM and the extraction of the ground points, under 1 ms for the ground plane estimation and 30-40 ms for the image processing and the extraction of the obstacles).

[1]www.opencv.org
[2]www.ros.org

(a) (b)

Fig. 12. (a): John Deere's Gator, the test vehicle used for the experiments. (b): The point Grey Grasshopper cameras mounted on their stereo rig, approximately 1 meter above the ground. Obs.: The thermal camera and the inertial measurement unit in the center of the rig were not used in this work.

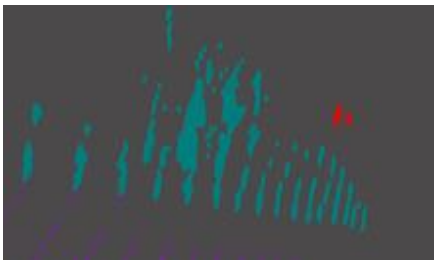
TABLE I
Performance of the visual obstacle detection algorithm. The true positive rate (TPR) and the false positive rate (FPR) quantify the outcome.

	Overcast	**Sunny**
Number of Frames	730	1300
TPR	100%	98%
FPR	4.1%	5.7%

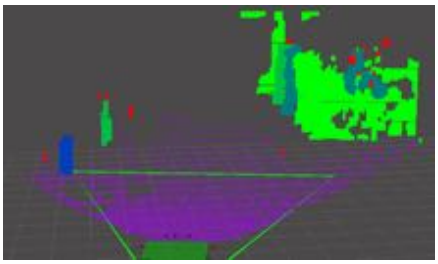
B. Results

A dataset containing a variety of obstacles, such as cars, pedestrians, street poles and bushes, was used to assess the performance of the obstacle detection algorithm. In each frame of the dataset, there is an obstacle (or multiple obstacles) as well as free space. The assessment of the algorithm is done on a frame by frame basis. If all obstacles present in the frame are detected (with an overlap of at least 90% between detection and ground truth), the frame is counted as a true positive. Otherwise, it is considered as a false negative. Analogously, if the free space is detected as such, the frame is counted as a true negative, otherwise as a false positive. Two quantities are calculated: The true positive rate (TPR), which is the ratio of number of true positives over number of total actual positives (here: number of frames), and the false positive rate (FPR), which is the ratio of number of false positives over number of total actual negatives. The results of this assessment can be found in Table I.

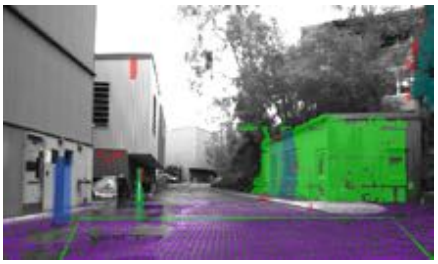
With a high quality disparity point cloud, the obstacles were always detected, with the algorithm performing extremely well regarding the TPR (Figure IV-B). On the



(a) Zoom onto an obstacle. Small speckles (clusters) are only discarded if they are far away from other clusters.

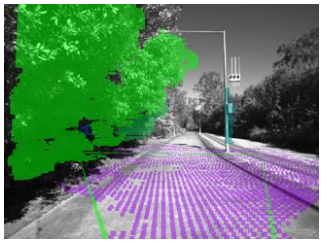


(b) Clusters and filtered out speckles in 3D.



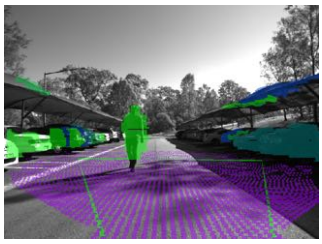
(c) Clusters and filtered out speckles overlaid over the left camera image.

Fig. 11. Results of the filtering. The clusters are displayed in shades of green and blue. Discarded clusters are red.

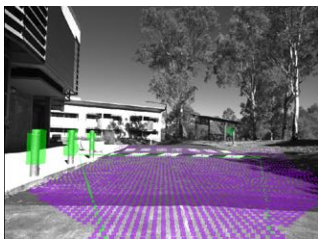


(a) Detection of bushes and light poles.

(b) Detection of cars.



(c) Detection of pedestrians.



(d) Detection of poles.

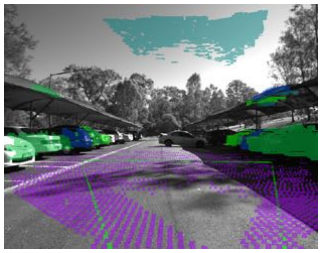
Fig. 13. Snapshots of different scenes in the obstacle detection experiments.

other hand, false-positives arise when the stereo-matching algorithm produces faulty matches between the left and the right image, creating a faulty data range as a consequence. This happens more often in two different scenarios:

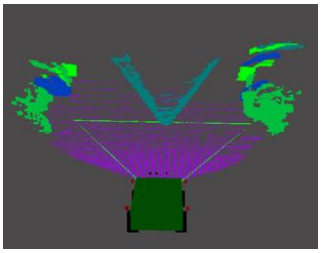
- **Sky:** Occasionally, the matching algorithm falsely matches pixels in the (uniformly blue) sky. The falsely matched points appear close to the camera instead of being at infinity, where they are in reality.
- **Repetitive Patterns:** The stereo-matching algorithm has difficulties finding the appropriate matches between the stereo images in the presence of repetitive patterns. On the test site[3], the facades of several buildings contain vertically oriented corrugated metal sheets, which corresponding to ambiguous feature point description and matching.

As in natural scenarios most of the faulty matches are scattered, the Bayesian filtering proposed in Section III-E successfully mitigates this effect. However, when they occur in big bulks, a false positive can appear as obstacle for a few frames (cf. Figure 14).

[3]CSIRO's QCAT (Queensland Center for Advanced Technology)



(a) Detected obstacles overlaid over left camera image.



(b) 3D view of the detected obstacles.

Fig. 14. Example of a false positive: Points in the sky are falsely matched yielding false positives in the obstacle detection.

C. Closed-Loop Implementation on an Autonomous Vehicle

This section discusses results of the system implementated on an autonomous vehicle, illustrating that the method can effectively assist in the navigation. For these experiments, the Gator (Figure 12a) is also used as the test platform. The vehicle has been automated at CSIRO, and it is equipped with four horizontally mounted Hokuyo 2D laser scanners (mounted approximately 1m above ground), which are used for localization.

The lower-level navigation management is done by the Navigation Package[4] available by ROS. It bases its path planning on a map and on an arbitrary number of sensor inputs. The map is created with the Gmapping Package[5]. A satellite view of the some of the approximate routes traversed is shown in Figure 15(a).

The obstacles detected with the proposed algorithm are used as a second sensor input for the navigation package. The path planning is consequently adapted in order to avoid the obstacles. The following subsections illustrate the applicability of the algorithm embedded in a robotics vehicle in two different scenarios.

1) Goal in unreachable area: This experiment considers the scenario in which the end-position given to the Gator cannot be reached due to an obstacle in its path. As the global path planner sets a path, the Gator tries to follow the global plan as closely as possible with the assistance of a local path planner. When the obstacle is detected, a local plan to follow the global plan cannot be determined and the

[4]www.wiki.ros.org/navigation
[5]www.wiki.ros.org/gmapping

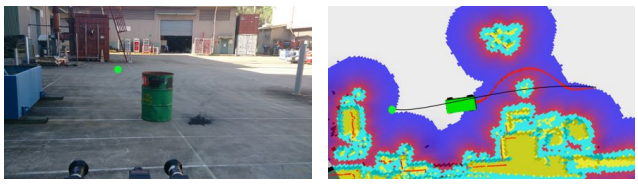
(a) Satellite view of the test area (the drivable routes are highlighted in green), illustrating the structured and unstructured routes. The scale is 410×360 meters.

(b) Map of the test area from 2D laser scans mounted on the vehicle. The laser-based localization is used for the closed-loop experiments.

Fig. 15. Vehicle and cameras used in the experiments.

vehicle stops. In our implementation, a "recovery behaviour" is executed (e.g. reverse and calculate a new plan). In case the obstacle continues to be detected, the recovery behaviour is unsuccessful and the robot aborts the task. We ran this test 15 times under different lighting conditions, achieving successful detection with the robot aborting the mission in all the cases.

2) Obstacle in the way: In contrast to the scenario above, the goal is often reachable, but the path needs to be adjusted due to an obstacle in the way. If the obstacle is detected from far enough, a path around it can be determined. The experimental setup is shown in Figure 16. If the robot is already too close to the obstacle for a smooth path to be found around it, it executes its recovery behaviours (i.e. reverse and calculate a new plan). 15 test runs were performed in different lighting conditions. The robot satisfactorily avoided the obstacle and reached its goal 14 times, failing once due to bright direct sunlight. As the vision system is used in conjunction with the lasers used for localisation, in this failure case, the lasers sensed the object proximity and stalled the vehicle.



(a) (b)

Fig. 16. Experimental setup where an obstacle requires re-planning of the path. (a) The goal (green dot) lies behind the obstacle, but is still reachable. (b) A path plan is initially found in the map (in black), but as the obstacle is detected, the navigation follows an alternative path (in red) around it.

V. Conclusions

We have presented an obstacle detection algorithm using stereo-vision, proposing a novel data representation paradigm and clustering approach. We run several tests exposing the algorithm to many different objects. The low false-positive and false-negative rates illustrate the applicability of the method. In future work, we plan to incorporate a temporal persistence filter and obstacle velocity estimation in order to mitigate the matching error effects described in Section IV-B. One important aspect of the false-positives, is that they generally do not persist in time, and a temporal persistence filter would only accept obstacles that are persistent over multiple frames. However, to accurately define such a filter is not a trivial task, as this requires a diligent spacial description of an object.

References

[1] J. Han, D. Kim, M. Lee, and M. Sunwoo, "Enhanced road boundary and obstacle detection using a downward-looking lidar sensor," *Vehicular Technology, IEEE Transactions on*, vol. 61, no. 3, pp. 971–985, 2012.

[2] R. Manduchi, A. Castano, A. Talukder, and L. Matthies, "Obstacle detection and terrain classification for autonomous off-road navigation," *Autonomous robots*, vol. 18, no. 1, pp. 81–102, 2005.

[3] G. Reina, J. Underwood, G. Brooker, and H. Durrant-Whyte, "Radar-based perception for autonomous outdoor vehicles," *Journal of Field Robotics*, vol. 28, no. 6, pp. 894–913, 2011.

[4] M. S. Darms, P. E. Rybski, C. Baker, and C. Urmson, "Obstacle detection and tracking for the urban challenge," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 10, no. 3, pp. 475–485, 2009.

[5] A. Wedel, U. Franke, J. Klappstein, T. Brox, and D. Cremers, "Realtime depth estimation and obstacle detection from monocular video," in *Pattern Recognition*. Springer, 2006, pp. 475–484.

[6] Y. Dong, Z. Hu, K. Uchimura, and N. Murayama, "Driver inattention monitoring system for intelligent vehicles: A review," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 12, no. 2, pp. 596–614, 2011.

[7] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," in *Asian Conference on Computer Vision (ACCV)*, 2010.

[8] A. Talukder, R. Manduchi, A. Rankin, and L. Matthies, "Fast and reliable obstacle detection and segmentation for cross-country navigation," in *Intelligent Vehicle Symposium, 2002. IEEE*, vol. 2. IEEE, 2002, pp. 610–618.

[9] F. Oniga and S. Nedevschi, "Processing dense stereo data using elevation maps: Road surface, traffic isle, and obstacle detection," *Vehicular Technology, IEEE Transactions on*, vol. 59, no. 3, pp. 1172–1182, March 2010.

[10] C. D. Pantilie and S. Nedevschi, "Real-time obstacle detection in complex scenarios using dense stereo vision and optical flow," in *Intelligent Transportation Systems (ITSC), 2010 13th International IEEE Conference on*. IEEE, 2010, pp. 439–444.

[11] D. Perrone, L. Iocchi, P. Antonello, and C. Fiat, "Real-time stereo vision obstacle detection for automotive safety application," in *Intelligent Autonomous Vehicles*, vol. 7, 2010, pp. 240–245.

[12] N. Bernini, M. Bertozzi, L. Castangia, M. Patander, and M. Sabbatelli, "Real-time obstacle detection using stereo vision for autonomous ground vehicles: A survey," in *Intelligent Transportation Systems (ITSC), 2014 IEEE 17th International Conference on*. IEEE, 2014, pp. 873–878.

[13] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[14] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[15] S. Se and M. Brady, "Stereo vision-based obstacle detection for partially sighted people," in *Computer Vision ACCV'98*, ser. Lecture Notes in Computer Science, R. Chin and T.-C. Pong, Eds. Springer Berlin Heidelberg, 1997, vol. 1351, pp. 152–159.

[16] R. Turchetto and R. Manduchi, "Visual curb localization for autonomous navigation," in *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, vol. 2, Oct 2003, pp. 1336–1342 vol.2.

[17] X. Lu and R. Manduchi, "Detection and localization of curbs and stairways using stereo vision." in *ICRA*. IEEE, 2005, pp. 4648–4654.