A full-3D Voxel-based Dynamic Obstacle Detection for Urban Scenario using Stereo Vision

Alberto Broggi\(^1\), Stefano Cattani\(^1\), Marco Patander\(^1\), Mario Sabbatelli\(^1\) and Paolo Zani\(^1\)

Abstract—Autonomous Ground Vehicles designed for dynamic environments require a reliable perception of the real world, in terms of obstacle presence, position and speed. In this paper we present a flexible technique to build, in real time, a dense voxel-based map from a 3D point cloud, able to: 1) discriminate between stationary and moving obstacles; 2) provide an approximation of the detected obstacle’s absolute speed using the information of the vehicle’s egomotion computed through a visual odometry approach. The point cloud is first sampled into a full 3D map based on voxels to preserve the tridimensional information; egomotion information allows computational efficiency in voxels creation; then voxels are processed using a flood fill approach to segment them into a clusters structure; finally, with the egomotion information, the obtained clusters are labeled as stationary or moving obstacles, and an estimation of their speed is provided.

The algorithm runs in real time; it has been tested on one of VisLab’s AGVs using a modified SGM-based stereo system as 3D data source.

I. INTRODUCTION

A reliable perception of the real world is a key-feature for any obstacle detection system for dynamic environments. When it is possible to use assumptions on road model and on vehicle pose, successful solutions of geometry for obstacle segmentation have been proposed, in some cases working in image coordinates by using v-disparity space [1]; in other cases applying RANSAC fitting to segment the 3D points in road inliers and outliers [2].

The above approaches, however, do not exploit completely the 3D information provided by modern stereo matching algorithms and range finder devices. A notable 3D approach is the so called 6D vision [3], where each 3D point (computed by an FPGA stereo system) is tracked by means of efficient GPU Optical Flow implementation. When such dedicated hardware is not available, the most popular approach tries to provide a full, or partial, 3D reconstruction of the environment projecting depth information into digital elevation maps, containing cells of uniform size. Cells typically store information about the corresponding world portions, depending on the algorithm approach and type of sensor used. The cells just store an occupancy and height information; in this case we talk about 2.5D grid maps [4]. This approach defines an obstacle as anything that stands on the terrain surface; in urban scenarios the terrain surface is typically assumed to be flat, while on off-road an accurate slope estimation is needed [5]. The complex grid maps embed full 3D information using adjacent stack of cells [6], or octrees connected cubes [7], and are able to represent objects at multiple heights located at the same range and azimuth.

Full 3D maps, compared to 2.5D representation of observed scene, have the great ability to deal with obstacles with non conventional shapes, like concave ones. In this case typical elevation maps approaches tends to wrap obstacles with a convex surface, overestimating their volume. Conventional elevation maps also have difficulties in dealing with hanging obstacles, that are often merged together with the below terrain surface.

In this paper we designed an obstacle detector based on a full 3D scene reconstruction to estimate both stationary and moving objects with minimum assumptions of the road model. A modelling of the 3D point cloud, derived from a disparity image, into an accurate voxel reconstruction is necessary to build complex clusters. These data structures contain the geometric and texture information to make a segmentation based on a flood fill approach. A vehicle pose estimation is needed to determinate the velocity and position of obstacles detected; for this reason we developed an egomotion estimator based on the visual odometry approach introduced in [8]. Through a temporal interpolation of previous 3D voxel reconstructions, the objects above the ground can be easily detected and their velocity and position can be estimated using a Kalman filter; more complex and similar approaches exist in literature [3]. Example of the final results is shown in Fig. 1.

\(^1\) A. Broggi, S. Cattani, M. Patander, M. Sabbatelli, and P. Zani are with VisLab-Dipartimento di Ingegneria dell’Informazione, Università degli Studi di Parma \{broggi, cattani, patander, smario, zani\} @vislab.it
II. ALGORITHM OVERVIEW

The approach can be described as a 5 step procedure:

1) 3D world points cloud. A data set of 3D world points \((x_i, y_i, z_i)\) is generated. The data set must provide enough points to support the desired resolution over the area of interest. Implementation details in Section III.

2) Egomotion estimation. A pose estimation of ego-vehicle is needed to distinguish between stationary and moving objects; this is achieved by a visual odometry approach.

3) Voxel-based partitioning. The 3D point cloud is partitioned into voxels at a certain resolution. This structure will contain the points that belong to a voxel, regardless of whether they represent terrain, an object, or spurious noise. In Section IV we will discuss about how this voxel structure quantizes at each frame the 3D point cloud using the egomotion information.

4) Clustering. A colour-space segmentation approach has been used to group together voxels with similar features, considering only those voxels that are above the ground plane; a subsequent refinement and cluster aggregation phase is performed to increase robustness and remove false positives. To each cluster is assigned a unique label \(C\) corresponding to their center of mass \((C_x, C_y, C_z)\).

5) Moving obstacles. Finally, a Kalman filter is applied to prune false positives estimating the predicted speed and pose of candidates. The evaluation of these data determines which of clusters are moving obstacles.

III. PRE-PROCESSING

A. 3D Points Engine

The 3D points engine is based on the processing of the Disparity Space Image (DSI). To perform the stereo reconstruction a modified Semi Global Matching (SGM) algorithm [9] with a multi-resolution analysis scheme [10] has been implemented. In order to reduce computational weight, a multi-threaded SIMD processing scheme has been devised, exploiting the parallel processing capabilities of the target hardware platform.

The engine has been extensively tested during VIAC, the VisLab Intercontinental Autonomous Challenge [11], in a variety of scenarios and in different conditions.

B. Visual Odometry

Visual Odometry (VO) is the process of incrementally determining the position and orientation of a vehicle by examining the changes that motion induces on sequential camera images taken by its onboard cameras [12].

VO implementation used for this work is based on a variation of Geiger’s approach [8] that exploit a different feature descriptor.

![Feature matching using stereo images](image)

Fig. 3 shows the feature matching between two stereo images captured in different times. The top row shows two images of the last observed scene for the left and right camera; same way, the bottom row shows a frame of stereo images captured at the previous time. The similarity metric algorithm tries to detect the features comparing circularly the frame pair in the following order:

- \((\text{last left frame} \leftrightarrow \text{last right frame})\);
- \((\text{last right frame} \leftrightarrow \text{previous right frame})\);
- \((\text{previous right frame} \leftrightarrow \text{previous left frame})\);
- \((\text{previous left frame} \leftrightarrow \text{last left frame})\);

Finally the algorithm computes the roto-translation matrix that minimizes the reprojection error between the supplied correspondences via Levenberg-Marquardt non-linear optimization.

IV. VOXEL APPROACH

The developed approach for obstacle detection receives as input the point cloud from the 3D points engine and returns a list of obstacles, estimated in terms of position, volume and speed. This algorithm can be divided into three main steps. First, the positional 3D information of each point \((x_i, y_i, z_i)\) is used to partition the cloud into voxels. Then, a color-based clustering algorithm is used to obtain the list of obstacles from the set of voxels. Finally, each obstacle is tracked, in order to strengthen the output and estimate the obstacle speed.

A. Voxel-based space partitioning

The perception of the environment is based on the concept of voxel. A voxel is defined as an element of a partition of
the 3D space in cube-shaped blocks, where each block is aligned along the directions of the axes of the Cartesian 3D coordinate system.

Each voxel in a grid is identified by its position and dimension (usually expressed with the center and the edge length, respectively). The set of 3D points spatially belonging to each voxel, as well as some additional information stored for convenience (the centroid of the points, the number of points, the color mean values and component variances, both in the RGB and HSL color spaces), is assigned to the same voxel. These data are stored using an efficient approach [7] saving only the needed voxels: thanks to egomotion information, new voxels are created only when a previously empty volume of the space is now occupied by a new 3D point; similarly, voxels are removed if no 3D points fall into it. Inserting and retrieval costs correspond to an un-ordered associative container; the implementation guarantees an $O(N)$ complexity for the worst case, $O(1)$ for the mean case. Moreover, this implementation allows the movement in the set at a constant cost (plus, eventually, the retrieving of the data), both in a Cartesian and a hierarchical (octree-based) way. This model could be extended to any n-dimensional space.

B. Color-based clustering

The obstacles have been assumed to be above the road plane. So, only the voxels with positive $z$ have to be considered. The set of these voxels is used as the input of a clustering algorithm that segments the same set into clusters, each of them corresponding to an obstacle candidate. To each cluster is assigned the corresponding set and number of contained voxels, the color mean and variance (both in RGB and HSL color spaces), the overall bounding box and its density (where density of a cluster is defined in terms of the ratio between the number of contained voxels and the bounding box volume).

$$f(v, c) = \|\vec{I}_{\text{RGB}}(v) - \vec{I}_{\text{RGB}}(c)\|_2 + \|\vec{I}_{\text{HSL}}(v) - \vec{I}_{\text{HSL}}(c)\|_1$$

(1)

Algorithm IV-B.1 Flood fill color-based clustering

```
foreach not-yet-considered voxel $v$ do
  $c$ ← new cluster containing $v$
  insert $v$ in queue
  while not queue is empty do
    $v'$ ← extract first element from the queue
    foreach $v'' \in N(v') \setminus \{v, v_c \in c\}$ do
      if $f(v'', c) < \varepsilon$ then
        insert $v''$ in $c$
        insert $v''$ in queue
      end if
    end foreach
  end while
end foreach
```

A flood fill color-based clustering algorithm is used for this purpose (Algorithm IV-B.1), using as a similarity check the Equation 1.

The neighborhood $N$ of a voxel $v$ is defined as the voxel-centered closed unit ball in the voxel set, measured with the Chebyshev distance.

The similarity between a cluster and a voxel is based on color. The color of the voxel is compared to the color of the cluster resulting from the colors associated to each already-inserted voxel in the cluster. Various types of color matching have been experimented: the chosen color match is based over the normalized means of the RGB and HSL components, as stated in Equation 1. Empirically, it has been found that the algorithm could produce small clusters, due to the noise over the disparity data, depending on the objects in the scene and on the lighting conditions. So a refinement has been added. First of all, for each small cluster a big neighbour cluster is searched. If any is found, the small one is aggregated to one of them (Algorithm IV-B.2). Every small, or with low density, remaining cluster is treated as noise and it is not considered in further computations (Algorithm IV-B.3).

Algorithm IV-B.2 Cluster aggregation

```
foreach cluster $c$ do
  if number of voxels of $c$ small and there is at least one voxel near $c$
    $c' ←$ cluster with biggest number of voxels near $c$
    aggregate $c$ in $c'$
  end if
end foreach
```

Algorithm IV-B.3 Cluster noise removal

```
foreach cluster $c$ do
  if number of voxels of $c$ small or low density of $c$
    remove $c$ from list of clusters
  end if
end foreach
```

C. Tracking

The reliability of the output is enhanced using a tracking algorithm. The clusters found in the current frame are associated with tracked clusters, according to a greedy policy based on a distance function (Algorithm IV-C.1).

The information of cluster color, dimensions, density and predicted position have been used to create distance functions between clusters. Several linear combinations of these distances have been tested as a distance function for the algorithm.

Each tracked cluster is put in one of these three queues:
- a newly seen cluster is put in the acceptance queue, where it remains until it is seen for enough consecutive frames. If this does not happen, it is discarded.
Algorithm IV-C.1 Tracking associations generation

\[ C \leftarrow \text{set of current frame clusters} \]
\[ T \leftarrow \text{set of tracked clusters} \]

while \( C \) or \( T \) contain not-yet-considered elements do

\[(t^*, c^*) \leftarrow \arg \min_{(t,c) \in T \times C} \ d(t,c)\]

if \( d(t^*, c^*) \) less than threshold then

consider \( t^* \) and \( c^* \) as associated

remove \( t^* \) from \( T \) and \( c^* \) from \( C \)

end if

end while

foreach \( t \in T \) do

consider \( t \) as non associated (ghost)

end foreach

foreach \( c \in C \) do

consider \( c \) as non associated (new cluster)

end foreach

- a cluster seen for enough consecutive frames is put in the tracked queue, where it remains as long as it is visible.
- a cluster that disappears from the view is put in the ghost queue. If it becomes visible again, it is put in the tracked queue. A cluster that remains too long in the ghost queue is discarded.

Moreover, each tracked cluster has a 6-dimensional state associated to it, identified by the position and the speed of the centroid of the cluster. This state is used to predict and correct the spatial information of the cluster by a linear Kalman filter.

V. TESTING CONSIDERED

The input 3D point cloud includes everything that is visible in the images: terrain, obstacles standing on the terrain, sky, etc. The goal of our algorithm is to detect moving obstacles, removing all the remaining points (considered as outliers). The core of algorithm is based on the reconstruction of a voxel map. To figure out the reliability of the developed approach, an evaluation that provides the optimal rendering resolution of voxel partitioning is required. An easy scenario is when all 3D points are processed with no ego-movements (e.g. a flat road in a known position); in this case it is enough to rebuild the whole 3D point cloud into a voxel map at different resolutions. Fig. 4 shows these experiments.

A low resolution indicates a high level detail of the voxel map and vice versa. A high level detail of 3D reconstruction requires a high computational cost in terms of performance; moreover, when the voxel size reduces to the limit where each voxel contains exactly one 3D point, the resulting resolution tends to correspond to the Disparity Map, jeopardizing the advantages of a voxel representation. Hence, it is important to define the right resolution to apply to the clustering algorithm, to achieve the best trade-off between obstacle detection robustness and accuracy in position and speed estimation.

According to this concept, and the experiments shown in Fig. 4, when using a volume of interest of \(28.7m \times 16.4m \times 3.5m\), we have finally found a satisfactory voxel resolution at 0.25m, able to provide good processing performance and real-time computation. Alternative configurations provided less accurate results in terms of detail or several clusters that required a higher computational power to estimate the object’s observed shape; in this case many objects are represented in two or more clusters resulting as truncated due to the little variations in clustering criteria values (e.g. unique color-texture, single point in voxel). Fig. 5 shows the computational costs required to build a complete voxel map. Preliminary results about the moving obstacle detection are shown in Table I. Moving pedestrians and cars have been classified as dynamic. In these tests we have used two annotated image sequences as ground truth with different lighting, cameras and system configurations evaluating the
correct (TP) and false (FP, FN) detections.

<table>
<thead>
<tr>
<th>#Sequence</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>#Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence 1</td>
<td>82%</td>
<td>10%</td>
<td>2%</td>
<td>4500</td>
</tr>
<tr>
<td>Sequence 2</td>
<td>75%</td>
<td>22%</td>
<td>3%</td>
<td>6200</td>
</tr>
</tbody>
</table>

TABLE I
RESULTS ON ANNOTATED IMAGE SEQUENCES

VI. RESULTS

The algorithm has been developed in C++ using VisLab’s GOL D [13] framework; it has been tested in different urban scenarios providing reliable detection of the moving obstacles. With 28.7$m \times 16.4$m $\times 3.5$m volume of interest, 0.25$m$ voxel resolution and 640 $\times$ 480 pixel images, it runs at 10$Hz$ (DSI=$45ms$, Voxel clustering and tracking=$55ms$) on an Intel® Core™ i7. Fig. 6 shows various results in urban scenarios with different cameras, lighting conditions and system configurations. Fig. 6(a) to Fig. 6(g) show only moving obstacles detection results in different urban scenarios; the pose prediction of each obstacle is shown using colored vectors representing the estimated obstacles’ velocity in terms of speed and direction. The next Figures show some examples where also stationary obstacles are highlighted. Fig. 6(l) shows a complex scenario with various obstacles in a construction site, such as trucks and pedestrians, as stationary objects. In this work everything that stands up from a flat ground is detected as an obstacle and no classification or knowledge assumption is applied to estimate the detected object typology.

VII. CONCLUSIONS

In this paper we presented a real-time approach for full 3D obstacle detection, based on stereo vision and voxel-based representation of the environment. A 3D point cloud is sampled into a 3D voxel map that is processed through a color-based clustering technique to build a set of clusters as obstacle candidates for the current frame. A linear Kalman filter is applied to estimate the obstacles motion and pose to distinguish between moving and stationary. The core of this approach is the reconstruction of the voxel map. It is the result of a frame based interpolation between the voxel map generated at the previous frame and the 3D point cloud at current frame; the 3D coordinates of each point are processed in accordance with the ego pose using a visual odometry approach. This represents an additional advantage of our algorithm, because it allows an efficient update of the voxel map: points that fall into an existing voxel just update the corresponding characteristics, while a new voxel is added only when a previously empty volume of the space is now occupied by some points. Another important advantage is based on a fully 3D perception of the object detected without any assumption of shape, providing an accurate estimation of complex obstacles (e.g. hanging or concave).

Future Works

A terrain mapping method could improve the obstacle detector making it independent of any assumptions of ground shape (e.g off-road scenarios). It represents the most important goal for future works. Increasing the robustness of vehicle odometry, considering also the adoption of some hardware solution (i.e. INS and/or GPS). The vision based 3D engine implementation can be negatively affected by poor visibility conditions, leading to low disparity map densities. More testing sessions are needed to characterize the system performance with respect to 3D points resolution and density.

Finally, a quantitative assessment on obstacle detection accuracy is now being considered, running the system on a supervisioned scenario where obstacles are completely known.

VIII. ACKNOWLEDGEMENTS

The work described in this paper has been developed in the framework of the Open intelligent systems for Future Autonomous Vehicles (OFAV) Project funded by the European Research Council (ERC) within an Advanced Investigators Grant.

REFERENCES

Fig. 6. Examples of obstacle detection: (a)(b)(c)(d)(e)(f) moving cars without detection of stationary obstacles; (g) moving motorcycle without detection of stationary obstacles; (h)(i)(j)(k) moving cars and stationary obstacles; (l) stationary pedestrian and others obstacles in a construction site.


