Liborg: a lidar-based Robot for Efficient 3D Mapping

Michiel Vlaminck[a], Hiep Luong[a], and Wilfried Philips[a]

[a]Image Processing and Interpretation (IPI), Ghent University, imec, Ghent, Belgium

ABSTRACT

In this work we present Liborg, a spatial mapping and localization system that is able to acquire 3D models on the fly using data originated from lidar sensors. The novelty of this work is in the highly efficient way we deal with the tremendous amount of data to guarantee fast execution times while preserving sufficiently high accuracy. The proposed solution is based on a multi-resolution technique based on octrees. The paper discusses and evaluates the main benefits of our approach including its efficiency regarding building and updating the map and its compactness regarding compressing the map. In addition, the paper presents a working prototype consisting of a robot equipped with a Velodyne Lidar Puck (VLP-16) and controlled by a Raspberry Pi serving as an independent acquisition platform.

Keywords: 3D mapping, ICP, lidar, multi-resolution, octree

1. INTRODUCTION

Many of today's applications require accurate and fast 3D reconstructions of large-scale environments such as industrial plants, critical infrastructure (bridges, roads, dams, tunnels), public buildings, etc. These 3D reconstructions allow for further analysis of the scene, e.g. to detect wear or damages on the road surface or in tunnels. The 3D models can also be used to organize or monitor events in conference venues or in other event halls. Finally, in the domain of intelligent vehicles, 3D maps of the environment can facilitate autonomous driving. Unfortunately, the current process of 3D mapping is still an expensive and time-consuming process as it is often done using static laser scanning, hence needing a lot of different viewpoints and a lot of manual intervention and tuning. Often times it is also difficult to map the entire area in detail; there are always parts that are too difficult to reach. Motivated by these shortcomings, we present our Liborg platform, a spatial mapping and localization system that is able to acquire 3D models on the fly by using lidar data. By means of a prototype, we built our own four-wheel robot consisting of a Velodyne Lidar Puck (VLP-16) controlled by a Raspberry Pi to serve as an independent acquisition platform. The robot is able to drive autonomously but can also be controlled by a remote control. Currently, the data is streamed to a server where the processing is done. In the future we plan to integrate a nVidia Jetson TX1 on the robot in order to be able to do the processing on-board. Figure 1 shows two images of our Liborg robot with the scanner being mounted using different tilt angles.

In previous work,[1] a mobile mapping system was presented that operates online and gives autonomous vehicles the ability to map their surroundings. In that work, a Velodyne HDL-32e lidar scanner was used to capture the environment. The focus was mainly on the lidar odometry and no attention was given to the map data structure itself. This work will extend the former by keeping a compact *global* map of the environment in memory that will be continuously updated by fusing newly acquired point clouds. This fusion will improve the map by reducing noise and correcting small errors made during the pose estimation. It will also help to estimate future poses more accurately. In order to guarantee fast execution times, we propose to organize the map as a hierarchical octree that serves as a compact representation of the environment. This paper will explain how the octree-based map can be exploited to speed up the estimation of the current pose of the robot without sacrificing accuracy. In addition, we will discuss how our solution is generic in the sense that no specific sensor set-up is needed. The sensor can thus be put in any orientation without any additional requirements. Also, no additional assumptions are made on the type of the environment. Finally, we conducted an experimental study using our Liborg robot and evaluated our system on both processing time and accuracy.

Further author information:
Michiel Vlaminck: E-mail: michiel.vlaminck@ugent.be

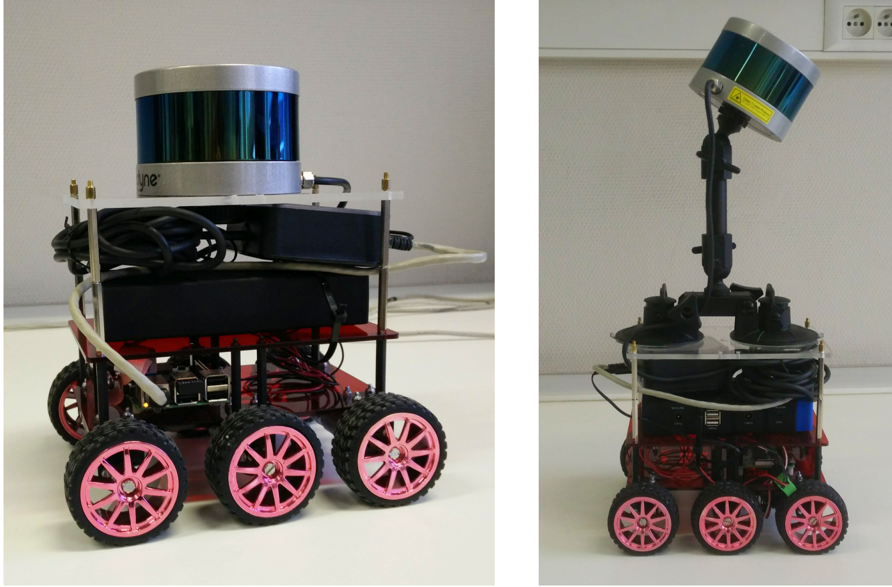


Figure 1. Picture of our prototype Liborg. A Velodyne Lidar Puck (VLP) is mounted on a remote control vehicles steered by a raspberry pi. Currently, the data from the Velodyne is streamed to a server where the processing is done. On the left picture, the VLP is mounted horizontally, whereas on the right picture the VLP is tilted in order to be able to capture overhanging objects (e.g. the ceiling) with sufficient detail.

2. RELATED WORK

The topic of mobile mapping has been widely studied in the past. Many of the early mapping robots used a single laser beam mounted with a fixed tilt angle[2] and as a result, they only provided a 2D map or ground plan of the environment. Unfortunately, 2D maps have limited use in the domain of autonomous driving or damage assesment as they are not detailed enough to allow further analysis. In a nutshell, the problem of mobile mapping can be split into two sub problems. The first one deals with the estimation of the current pose of the robot - also referred to as odometry - and is often conducted using scan matching. The second problem deals with the fusion of the sensor data into a coherent global map. Regarding the scan matching, the main registration approach is the Iterative Closest Point (ICP) algorithm or one of its many variants. ICP is an algorithm in which the transformation between two scans is estimated by assuming closest point correspondences in the two scans and subsequently minimize the ℓ_2-norm between all corresponding pairs. The work of Bosse and Zlot,[3,4] well known for their 3D mapping with a 2-axis laser scanner, is an example of a ICP-based approach. A second approach for 3D scan matching is based on the (3D) normal distribution transform (NDT). Magnusson *et al.*[5] were the first to introduce 3D scan matching using this 3D NDT. They used it on an autonomous mining vehicle. A third approach was presented by Zhang *et al.*[6] Just like Bosse and Zlot the authors used a 2-axis laser scanner that is moving in 6-DOF to map the environment. To find the accurate transformation that aligns all points, two low-level features, 'edges' and 'planar patches' are extracted and used to find the correct transformation.

However, many of the aforementioned solutions are not scalable as the number of (raw) data points in the map becomes intractable. For that reason, many other papers have been focusing on mapping the 3D environment with so called *occupancy grids* based on octrees.[7–9] The *Octomap* framework presented in[10] is a nice example of such a mapping solution that deals with sensor noise by means of probabilistic occupancy estimation. However, their work only describes the map data structure itself and not how it can help to speed up the map building or map updating process. Also, in their framework it is also not possible to spatially adapt the resolution to the desired detail. In contrast to this work, Droeschel et al.[11,12] presented a mobile mapping system based on laser data that uses a 3D map with locally varying resolutions. In their work, the authors suggest to choose the resolution based on the distance to the sensor, which correlates with the sensor's characteristics in relative distance accuracy and measurement density. Altough their method proofs effective for laser registration, the resolution scheme remains fixed and does not give the freedom of adapting it based on other criteria. In[13] on

the other hand, the authors propose a technique to find the adequate resolution for grid mapping and to adapt cell sizes locally on the fly. The splitting of the cells is based on a statistical measure they derive in their paper and in contrast to other approaches the adaptation of the resolution is done online during the mapping process itself. In contrast to the former studies, in which one 3D map is used with varying resolution, the authors of[14] suggest to use a hierarchy of octree-based models instead. Doing so, they keep a collection of octrees in memory each with their own leaf resolution size. As a result, objects of interest could be modelled with a lot more detail then unimportant elements. Alltough this later approach is very promising, building and maintaining hierarchies of octrees is a lot more complex to efficiently update them with newly acquired 3D points. In this work we will exploit the advantages of an octree-based map to quickly find an accurate estimate of the current pose of the robot and fuse newly acquired points with it.

3. APPROACH

The key idea of our solution is to combine the popular ICP algorithm with an octree-based map to enhance the speed of the scan matching while preserving the quality (in terms of accuracy) of the obtained map. One of the main drawbacks of an ICP-based solution is data storage, since all the raw 3D points have to be saved. In case the estimation of the transformation is solely conducted using the current and the previous scan, only the prvious scan has to be saved. However, in case we want to use larger parts of the current *map*, or a larger history, data storage might become an issue. Thus, a different data structure for the map structure is required. A second drawback of the ICP algorithm is the need for nearest neighbour searches, an inherent time-consuming task. Both drawbacks of the ICP algorithm can be alleviated by adopting an octree-based map structure. Our entire mapping approach consists of three main parts: 1) (pre)-processing the raw point clouds, 2) finding the alignment of two consecutive point clouds and 3) finding the alignment of the point cloud with the map built so far and fuse them together.

3.1 Preliminaries

As stated before, the Liborg platform is using the Velodyne Lidar Puck (VLP), a lidar scanner containing 16 lasers that is continuously spinning its head. We call one rotation of the head a *sweep* and we denote the point cloud acquired during sweep k as $\mathcal{P}_k$. The VLP is spinning its head 10 times per second and hence every 100 ms a point cloud is *acquired*. The captured point clouds can contain up to 29k points. During one experiment, the scanner is mounted with a fixed tilt angle. This means that the scanner, in normal situations, will not experience a rotation around its roll angle. Rotations around the pitch angles are possible if the robot is *climbing* a tilted surface. Our Liborg robot is able to climb inclines of up to 45°.

3.2 Point cloud (pre)-processing

The point clouds acquired by the VLP scanner are quite accurate, but obviously not perfect. Some outliers are still present and the accuracy in terms of measured distance approximates 3 cm, compared to 2 cm of its bigger brother, the Velodyne HDL-32E. For that reason, a first step in the whole mapping process is to filter out the outliers and *correct* small errors. The second purpose of the preprocessing is to convert the point cloud into a 2D grid image in order to obtain a notion of adjacency in the 2D domain. Doing so, we can exploit algorithms from the image processing literature to quickly perform certain operations. An overview of the different steps in the preprocessing part is given by figure 2.

3.2.1 2D grid conversion

As the VLP scanner is using 16 lasers, all firing more or less simultaneous while it is spinning its head, we can construct a 2D grid by *projecting* the 3D points into a cylindrical image plane. Each element in the 2D grid contains a 3D point as well as its distance to the sensor origin hence serving as a *range image*. The 2D grid has 360° degrees of horizontal FOV and will give us a notion of adjacence, hence helping us in conducting *nearest neighbor* queries in a fast manner.

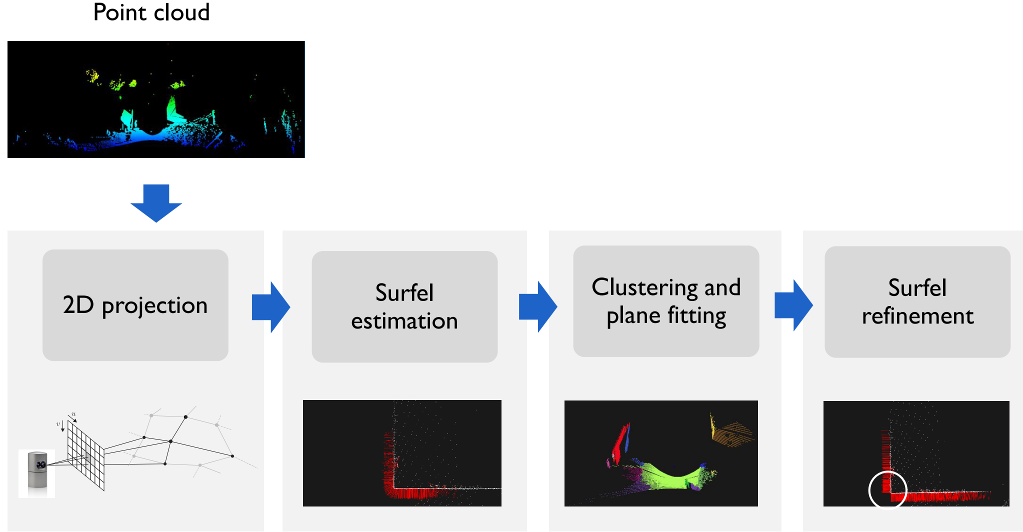


Figure 2. Overview of the point cloud processing process prior to alignment.

3.2.2 Surfel estimation

In essence, the lidar scanner is sampling a world composed of surfaces and a logic step is hence to reconstruct this underlying surface. The surfel estimation step will for each point estimate low level features describing the underlying surface. The word *surfel* is a concatenation of 'surface' and 'element' and hence it can be seen as a building block (element) to represent a small surface patch. In essence, it consists of the point $\mathbf{p}_k$ itself, together with its normal vector $\mathbf{n}_k$ and the set of neighbours $\mathbf{N}_k$ that was used to compute the surface normal. We will denote the surfel of a point $\mathbf{p}_k$ by $\mathbf{S}_k = \{\mathbf{p}_k, \mathbf{n}_k, \mathbf{N}_k\}$. Often times, an easy neighbourhood function is chosen, such as the k nearest neighbours or all the points lying within a certain radius r. However, this overly simple function causes the estimation of normal vectors to be inaccurate as points can be separated by an object border and can be lying on different surfaces. This can on its turn jeopardize the correct alignment of two point clouds and hence the estimation of the sensor pose. In literature, a popular way of estimating normal vectors is by using the principal component analysis of the neighbour set of a point. We extended this method by incorporating an additional point clustering based on the initial estimation of the surfels. We refine them afterwards using the neighbours resulting from this clustering. This initial estimation is conducted using an *approximate* k-nearest neighbour set. More specifically, we use a window set on the 2D grid we have computed earlier. The window has a width of 11 pixels and a height of 5 pixels, giving 54 potential neighbours. From these 54 candidates, we select the 30 nearest neighbours.

Thus, given the set of 30 neighbours $\mathbf{N}_k$ of a point $\mathbf{p}_k$, we compute the eigenvectors λ_1, λ_2 and λ_3 corresponding to the eigenvectors $\mathbf{v}_1$, $\mathbf{v}_2$ and $\mathbf{v}_3$. We then define the standard deviation along an eigenvector as $\sigma_i = \sqrt{\lambda_i}$ for $i \in 1, 2, 3$. Now, we can consider the three values $\psi_1 = \frac{\sigma_1 - \sigma_2}{\sigma_1}$, $\psi_2 = \frac{\sigma_2 - \sigma_3}{\sigma_1}$, $\psi_3 = \frac{\sigma_3}{\sigma_1}$ which represent respectively how linear, planar or volumetric the underlying surface is. We then define the *dimensionality* label as $l = \underset{i \in [1,3]}{\operatorname{argmax}}(\psi_i)$. More specifically, points lying on *lines* or the intersection of two planes get the label '0', points lying on planar surfaces get the label '1' and finally points belonging to volumes or scatter get the label '3'. A final feature denoted by the Shannon Entropy is computed as $E(\Psi) = -\psi_1 \ln(\psi_1) - \psi_2 \ln(\psi_2) - \psi_3 \ln(\psi_3)$. This measure gives a notion of the certainty about the dimensionality label. The eigenvectors, along with the values ψ_i, the dimensionality label and the Entropy gives us a feature vector that will be used later to guide the alignment process.

3.2.3 Clustering and plane fitting

In section 3.2.2 we described how low-level geometric features are computed. However, using the easy *approximate* k-nearest neighbour criterion, many surface normals are wrongly computed as some of their neighbours may lie

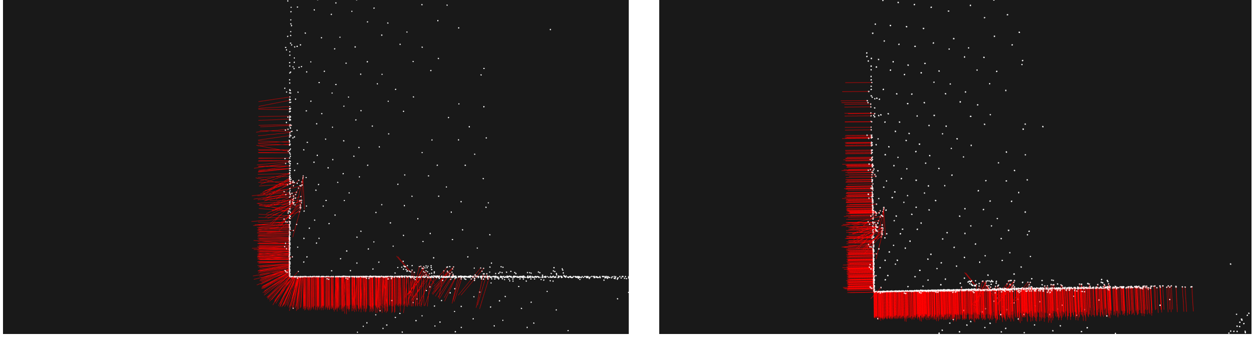


Figure 3. Image depicting the estimation of normal vectors using the k-nearest neighbor criterion. As can be seen in the left image, on the corner of the two planes, normal vectors are forming an arc. The normal vectors depicted in the right image do not suffer from this artefact.

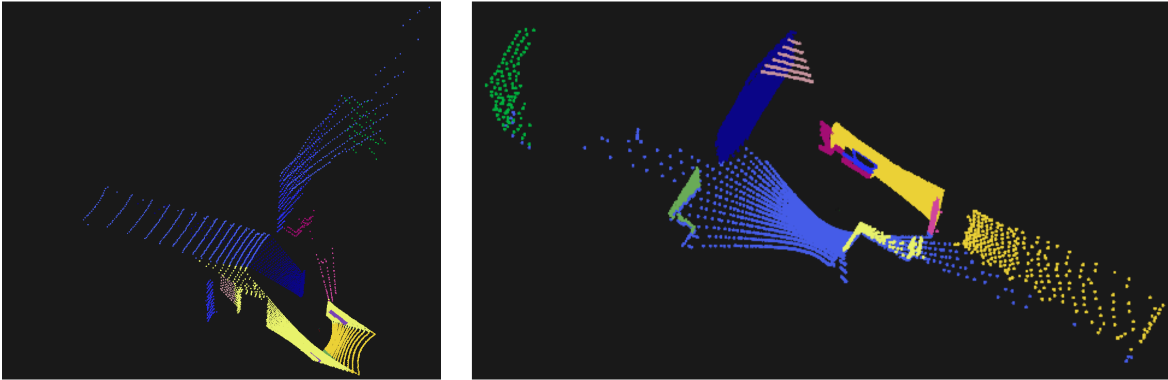


Figure 4. Image depicting the plane fitting and clustering algorithm.

on different surfaces or objects. The result of this artefact can be seen on the left image of Figure 3. In that image, the normal vectors are shown using red lines. We can notice that they form an arc-shape around the corner of the the planes (note that the image is a bird-eye view of two planes). To overcome this issue, we propose to cluster the points in order to find a better neighbourhood estimate. For each point, only the neighbours that are part of the same object or plane cluster will be taken into account to estimate the more accurate point normal. In order to find the clusters, we are adopting a two-step strategy. As many man-made environments are composed of large planar surfaces, we propose to estimate the dominant planes in the scene and cluster their points. As a second step, we cluster the remaining points being part of other non-planar objects.

For this process to be quick, we make use of the 2D grid that was computed before. More precisely, we perform a *region growing* algorithm that is guided by a few *comparisor* functions. The first *scan* of the image will use two different comparisor functions, one that is comparing the difference in angle of the two neighbouring points, $\mathbf{n}_i \cdot \mathbf{n}_{i-1} < \cos\theta$, and one that is comparing the Euclidean distance between the points in 3D space, $||\mathbf{p}_i - \mathbf{p}_{i-1}||_2 < \epsilon$. The values for θ and ϵ were respectively set to 5° and 5 cm. For each of these clusters we subsequently try to fit a plane through all of it points. If the *curvature* constraint, given by $\frac{\lambda_0}{\sum_i \lambda_i}$ is sufficiently low - in the experiments set to 0.001 - we accept it as a plane. Once the planes are detected, we *discard* them from the point cloud. Doing so, the remaining objects are behaving like *islands*, clusters that are separated from other objects by a gap caused by discarding the planes. Subsequently, we repeat the procedure, but this time only using the Euclidean comparisor function. In Figure 4, two images are depicted of this point cloud clustering. The point clouds are acquired with our Liborg platform and were recorded in a corridor of a building belonging to Ghent University.

Once the clusters are determined, the surfel estimation is *redone* using the newly estimated neighbourhood. More specifically, we keep the *approximate* k-nearest neighbors, but we only take points into account that are belonging to the same cluster. Figure 3 shows the estimation of the surface normals using the old neighbourhood

(left) and using the new neighbourhood (right). As can be seen, the normal vectors in the right image do not have the arc-shape in the corner any more. This is the result of only considering points belonging to the same cluster (or plane), leading to more accurate estimates. As a result of this process, some points will have an insufficiently number of neighbours and will form a cluster on their own. Obviously, we can not derive any valuable information from them and for that reason they will be discarded in the next processing steps dealing with the alignment of consecutive point clouds.

3.3 Point Cloud Alignment

Once the point cloud preprocessing is done, the next step consists of aligning consecutive point clouds with each other. The alignment or registration is based on the well-known *iterative closest point* algorithm, for which we applied small modifications. The five main steps in this algorithm are 1) point selection, 2) correspondence estimation, 3) correspondence rejection, 4) correspondence weighting and 5) transformation estimation. Regarding point selection, we use *all* points except those with a limited support, i.e. points with too few neighbours to compute reliable surface features, as determined in the preprocessing step. The correspondence estimation is lead by the *closest* points in the Euclidean 3D domain. Regarding the correspondence rejection and weighting, we apply the Beaton-Tukey robust M-estimator. The weighting function is given by equation 1. As can be noted, this estimator will fully exclude outlier correspondences.

$$w(r) = \begin{cases} (1 - \frac{r^2}{c^2})^2, & \text{if } |r| \leq c \\ 0, & |r| > c \end{cases}. \tag{1}$$

It is important to note that in this equation, r is the distance in *feature* space instead of their true distance in 3D. The distance metric that is used is the Mahalanobis distance, in order for the metric to be scale-invariant. The main motivation not to use the Euclidean distance in 3D for the weights is because two closest points between two consecutive clouds can have a large physical distance between them while still being a correct corresponding pair. This is usually the case for points lying at a far distance from the sensor. These correspondences can however be of great value for a correct and robust estimation of the transformation. The value for c is dependent on the median value of the *error* of *all* correspondences in feature space. Every iterations this value is recomputed, hence leading to an iteratively re-weighted least squares estimate. Finally, to estimate the transformation, we minimize the distance between all points and the tangent plane of its corresponding point, also referred to as the *point-to-plane* error metric. Its formula is given by equation 2:

$$E(\mathcal{P}_k, \mathcal{P}_{k-1}; \mathrm{T_{k,k-1}}) = \sum_{i=1}^{N} \mathbf{w}^i ((\mathrm{T_{k,k-1}} \mathbf{p}_k^i - \mathbf{p}_{k-1}^{\mathbf{c}(i)}) \cdot \mathbf{n}_{k-1}^{\mathbf{c}(i)})^2. \tag{2}$$

In this equation, $\mathrm{T_{k,k-1}}$ is the estimated transformation matrix, $\mathcal{P}_k$ is the *current* point cloud, $\mathcal{P}_{k-1}$ is the *previous* point cloud, $\mathbf{n}_{k-1}^i$ is the surface normal according to point $\mathbf{p}_{k-1}^i$, $\mathbf{w}^i$ is the weight vector and $\mathbf{c}$ is the vector containing the indices of the N corresponding points. The final transformation matrix $\mathrm{T}_{k,k-1}$ is then given by Eq. 3:

$$\mathrm{T_{k,k-1}} = \begin{bmatrix} \mathrm{R}_{k,k-1} & \mathbf{t}_{k,k-1} \\ \mathbf{0}_3^\top & 1 \end{bmatrix} = \underset{\mathrm{T}_{k,k-1}}{\operatorname{argmin}} E(\mathcal{P}_k, \mathcal{P}_{k-1}; \mathrm{T}_{k,k-1}). \tag{3}$$

3.4 Map Alignment

The main novelty of this work is in the way we keep track of the map that has been built up to a specific moment in time and more specifically in how we use it to speed up the alignment process while preserving highly accurate estimates.

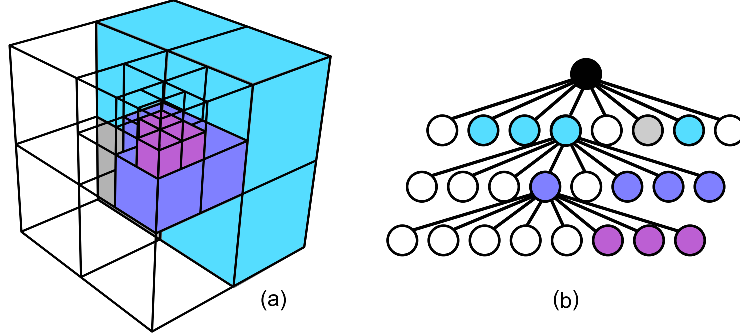


Figure 5. Picture depicting the octree data structure. Blue and purple voxels are *occupied* and different shades denote different resolutions.

3.4.1 Octree-based map

As mentioned before, we model the map as a hierarchical octree, an axis-aligned tree in which every node has up to eight children as can be seen in Figure 5. Starting at the root voxel (a bounding box in the shape of a cube) of the tree, at every level, the corresponding voxel is divided into eight smaller sub-voxels of the same size in case it is non-empty. This non-emptiness criterion is fulfilled as soon as there is one point present in the voxel. Thus, when a voxel does not contain any points, it is not split and no children are attached. This octree representation has many benefits regarding to an efficient alignment of consecutive point clouds. First, the known spatial relationship of the points in the map enables fast (approximate) nearest neighbour queries which on its turn facilitates correspondence estimation between newly acquired points and the current map. Second, an initial guess of the current pose can be estimated using a lower resolution representation of the scene derived from this octree. This guess can then be refined using a higher resolution resulting in fewer iterations and hence yielding another speed-up. This process involves lower resolution representations of the map which is based on a surface reconstruction technique described in the next session. The proposed multi-resolution procedure also has the advantage of being more robust against a less accurate initialisation.

3.4.2 Surface estimation

As mentioned in the previous section, we will estimate the underlying surface of the point cloud to derive a lower resolution representation. More precisely, we will estimate the surface for every point in the map and then re-sample the point cloud based on this surface. Doing so, the point clouds are improved in the sense that noise is removed, surfaces are smoothed out and small gaps are filled. However, if we would perform this operation on the *unstructured* point cloud, it would take a huge amount of time as this operation involves a lot of nearest neighbour searches. Fortunately, as we are maintaining the octree-based map, we have access to a highly efficient space partitioning of the scene and as a result we can conduct this surface estimation in a very fast manner. The main idea is to conduct the surface estimation on a voxel per voxel base. For every point we therefore only consider its nearest neighbours in the *same* voxel. This decreases the search space tremendously and thanks to the non-overlapping nature of the voxels, this process can be done for every voxel in parallel.

The surface reconstruction technique itself is based on the *moving least squares* algorithm, a method that estimates higher-order polynomials through data points. In the pre-processing step, we already determined for each point $\mathbf{p}_k$ its neighbours in $\mathbf{N}_k$ as well as its tangent plane defined as $H_k \triangleq [\mathbf{n}_k, d_k]$ in which d_k represents the distance to the origin. For all points lying in $\mathbf{N}_k$, we can compute the distance to this tangent plane. Subsequently, we fit a polynomial in the set of distances from these points to the surface. To this end, we define a local approximation of degree m by a polynomial $\tilde{p}_k \in \Pi_m$ minimizing, among all $p \in \Pi_m$, the weighted least-squares error:

$$\tilde{p}_k = \underset{p}{\operatorname{argmin}} \sum_{\mathbf{p}_i \in \mathbf{N}_k} (p(\mathbf{x}_i) - f_i)^2 \theta(||\mathbf{p}_i - \mathbf{p}_k||). \tag{4}$$

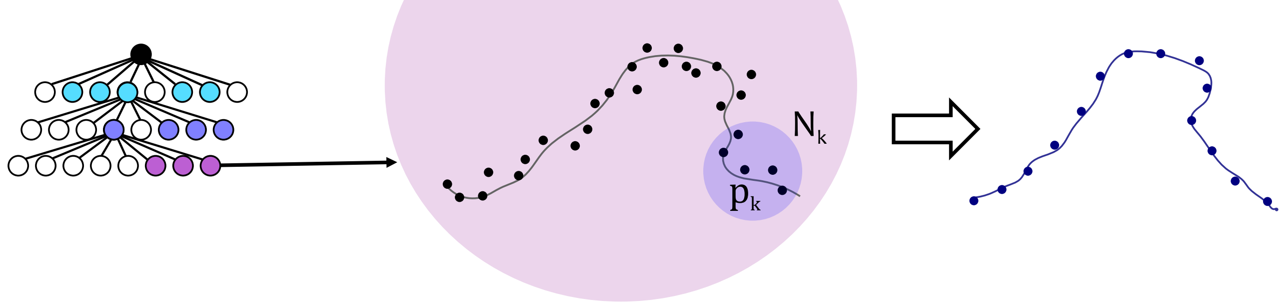


Figure 6. Drawing showing the principle of the surface reconstruction step, which is based on the moving least squares algorithm. Using the neighborhood $\mathbf{N}_k$ we try to fit a polynomial in the set of distances from the neighbours to the tangent plane of $\mathbf{p}_k$. These polynomials are then used to re-project *all* points to this surface. This process improves the point cloud by reducing noise, smoothing out surfaces and filling small gaps. The computed polynomials are also used to derive point clouds with a lower resolution.

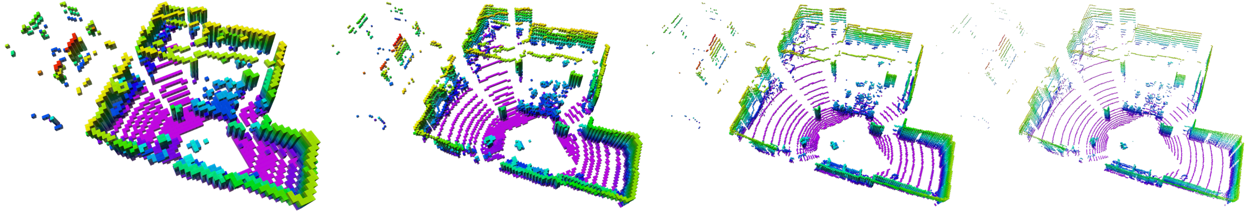


Figure 7. The same map at different resolutions. The alignment algorithm is based on a coarse-to-fine approach in which the lowest resolution representation is used to compute an initial guess, which is then refined using higher and higher resolutions.

In this equation, $\boldsymbol{I}$ is the vector containing the indices of the points in $\mathbf{N}_k$, $\{\mathbf{x}_i\}_{i\in\boldsymbol{I}}$ are the orthogonal projections of the points $\{\mathbf{p}_i\}_{i\in\boldsymbol{I}}$ onto the tangent plane and $f_i \triangleq \langle \mathbf{p}_i, \mathbf{n}_k \rangle - d_k$ is the distance of $\mathbf{p}_i$ to the tangent plane. Finally, $\theta(x) = e^{-(\frac{x}{\sigma_r})^2}$ represents the weighting function that is based on the distances to the tangent plane and the average separation σ_r of the 3D points. Simply put, points lying further away from the surface are getting a lower weight. Once the parameters of the polynomials $\tilde{p}_k$ are known, we project the points on the *moving least squares (mls)* surface. This procedure manipulates the points in such a way that they represent the underlying surface in a better way. In addition, we improve the point cloud using *voxel grid dilation* in order to fill small *gaps*. This latter process first dilates a voxel grid representation of the point cloud, built using a predefined voxel size. After that, the newly created points are projected to the *mls* surface of the closest point in the point cloud.

3.4.3 Multi-resolution alignment

Now that we estimated the underlying surface of the point cloud, we can exploit it in our proposed multi-resolution alignment algorithm. As mentioned before, the idea of the multi-resolution approach is to estimate the transformation using different point clouds derived by evaluating the octree at different resolution levels. The point cloud extraction process will construct a point cloud that contains exactly one point for every node or voxel in the tree. This point is derived as follows. First, we compute the 3D centroid of all points lying in the voxel. Next, we project this 3D centroid to the estimated underlying surface defined as the surface corresponding to the closest point of the 3D centroid in the voxel. Altough this point is a *virtually* created one, it is part of the estimated underlying surface. When using this *centroid*-cloud, a rough estimate of the transformation can be estimated using a similar error metric as in Eq. 3. This first transformation estimate will not be accurate yet, but it will serve as a good initial guess. Now, we can repeat the procedure by extracting another *centroid*-cloud at a higher resolution. In the end, we see that the whole process needs less iterations compared with an ICP-based approach using all the points in every iteration while giving the same accurate estimate. Figure 7 shows an example of the same map at different resolution levels. For clarity, the bounding boxes of each voxel or drawn in order to get a clear view on the actual resolution.

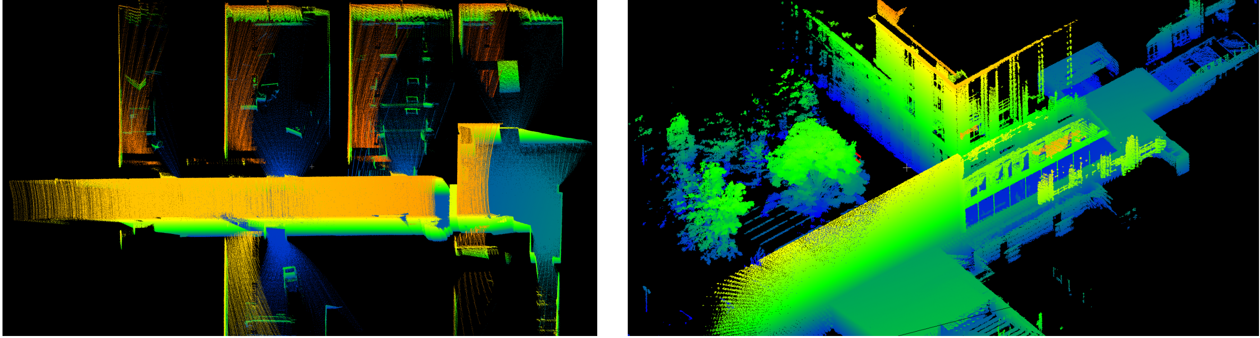


Figure 8. Two 3D reconstructions obtained by our Liborg platform having a tilted sensor as can be seen on the right image of Figure 1. Left, a 3D reconstruction of a corridor in a building of Ghent University, right a 3D reconstruction of another building reconstructed a part indoor and outdoor.

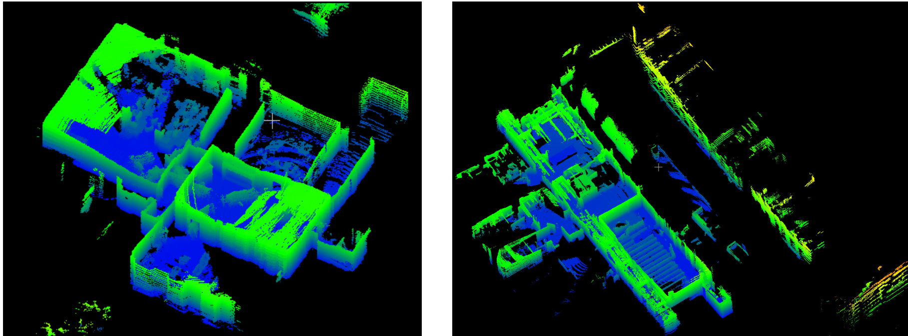
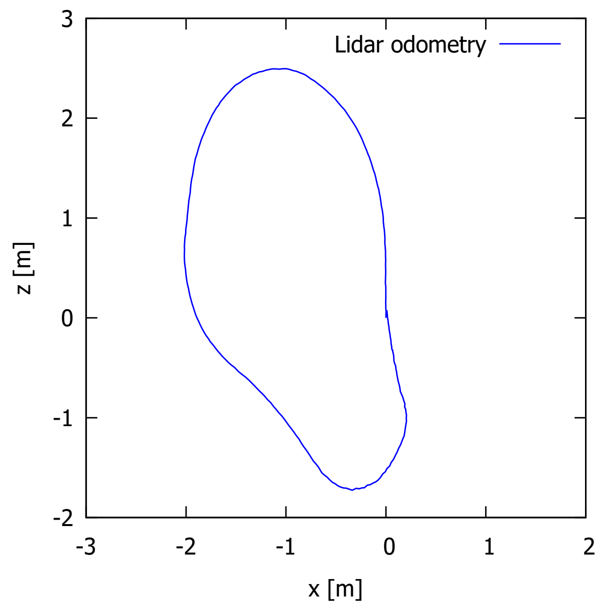


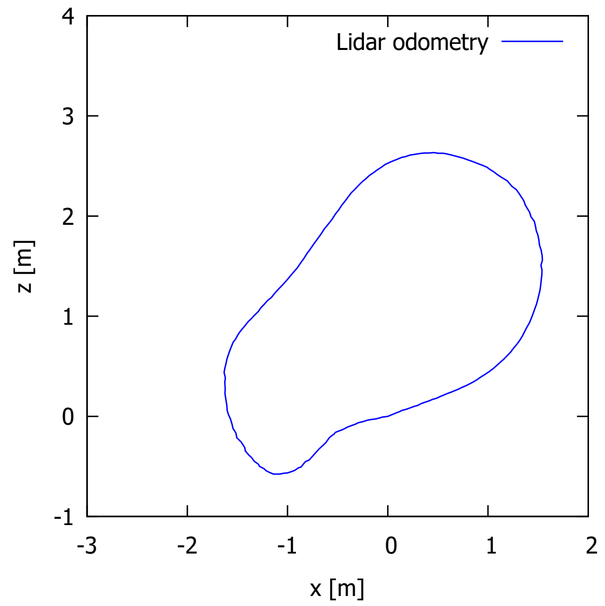
Figure 9. Two 3D reconstructions obtained by our Liborg platform with the sensor put horizontally as can be seen on the left image of Figure 1.

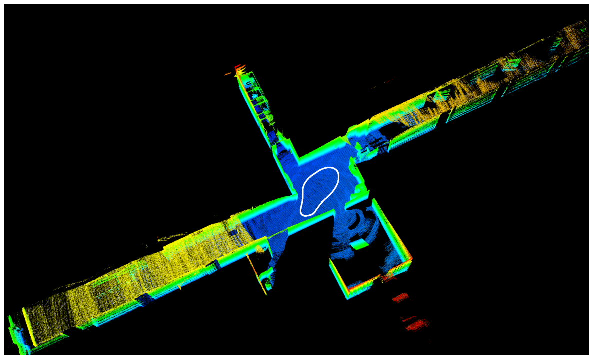
4. EVALUATION

Our Liborg robot has been used in several environments, both indoors and outdoors. In Figure 8, two 3D reconstruction results are depicted. The left one was obtained in a corridor of a Ghent University building while the other one was obtained by scanning a part indoor and another part outdoor. Both sequences were recorded with a tilt angle as shown on the right image of Figure 1. Figure 9 on the other hand shows two reconstructions obtained by putting the sensor horizontally. On all of the four reconstructed models, no discrepancies could be noticed visually. However, in order to be able to evaluate the accuracy of our system quantitatively, we performed two additional experiments. To show that our solution is independent of the choice of sensor set-up, we have put the lidar scanner under two different tilt angles. In the first experiment we have put the scanner horizontally while for the second experiment we have put the scanner under a certain tilt angle (cfr. Figure 1). This latter set-up has the advantage that 'overhanging' objects are also captured in detail and that the level of detail is more or less equal for every part of the scene. The second set-up has the advantage that the FOV is larger and it might benefit from seeing a larger part of the scene. For both experiments, we used 10 centimeter as the highest resolution during the multi-resolution map alignment process. We controlled the robot in such a way that a loop was formed but we did not conduct any kind of loop closure algorithm to correct for drift. As we do not have exact ground truth for the trajectories, we thus computed the difference in position of the start and estimated end point. The distance between both ends exactly represents the error as it should ideally be zero. The results are listed in table 1. The length of the trajectories were approximately 11 and 12 meter while the differences in the start and estimated end points are respectively 5.5 and 4.0 millimeter. Figure 10 shows the results of both experiments. The left graph and image shows the trajectory and reconstruction obtained by putting the sensor horizontally. The right graph and image shows the result obtained using the second set-up with the sensor mounted under a specific tilt angle. As can be seen, the estimated trajectories are indeed forming a loop, hence indicating the high accuracy of the system.

Regarding processing time, we examined the different sub modules separately. Currently, the data acquired







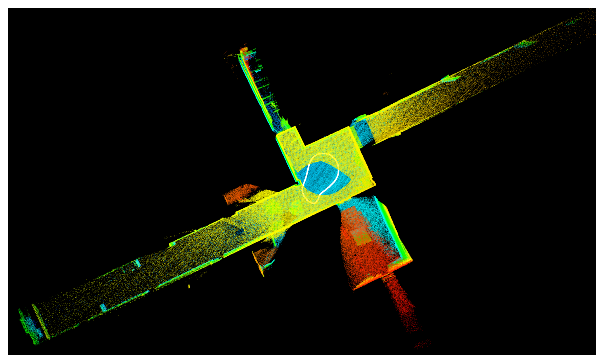


Figure 10. On top, the plots of the resulting trajectories of the two experiments that were conducted. On the left is the trajectory of the experiment with the scanner put horizontally, on the right the trajectory of the experiment with the scanner tilted. The bottom images show the 3D reconstruction. As can be seen, a large part of the ceiling is missing in the left reconstruction, while the right reconstruction covers the whole 3D space (except for a small hole in the ceiling). The Liborg platform is able to deal with both sensor set-ups as the two loops are nicely closing without a large error.

experiment	length (m)	error(m)
1	11.02	**0.0055**
2	11.95	**0.0040**

Table 1. The total error of two experiments conducted in a building of Ghent University. The result of the first experiment was obtained by putting the sensor horizontally while for the second experiment the sensor was tilted (cfr. Figure 1). For both tests, the Liborg robot was controlled to ride a loop. The error denotes the distance between the start point and the 'estimated' end point. The final error is equal or a bit lower than 5 millimeters.

sub-process	t(ms)
2D projection	12
Surfel estimation	8
Clustering and plane fitting	5
Point cloud alignment	50-70*
Map alignment	50-400**
Map fusion	5
Total	**130-500**

Table 2. Processing times for all the different sub-modules obtained for the recordings depicted in figure 10. Regarding the point cloud alignment (*), we note that one iteration takes approx. 10ms and on average 5 to 7 iterations are needed to converge. Regarding the map alignment process (**), we have to mention that the highest resolution for the map was 10 cm. Further, the processing time is also depending on the number of iterations on this highest resolution (in general lying between 3 and 5) and the size of the map. In order to bound the size of the map, a box filter was used keeping on average 40k to 80k points on a resolution of 10 cm. Of course this is depending on the type of environment.

by the Liborg platform is streamed to a server (or a laptop) doing the processing as the Raspberry Pi is not powerful enough and only CPU power is used at this time. All of the experiments were executed on a laptop with an Intel Core i7-4712HQ, 2.30Ghz CPU inside and 16GB RAM. The experiments were conducted using OpenMP using the 4 cores of the CPU at the same time for the surface reconstruction and correspondence estimation step. Table 2 shows the average execution times for every sub-module. These values were obtained by averaging out the execution times of all the frames of the two recordings shown in Figure 10. We can state that the preprocessing of the point clouds only takes approximately 25 milliseconds. The point cloud alignment module takes an additional 50 to 75 milliseconds, depending on the number of iterations. The two modules together can already act as a odometry solution that is running real-time. Recall that the VLP is outputting a point cloud every 100ms and the two 'sub-modules' take 75 to 100 milliseconds to process. If we add the map alignment and fusion, the results are more accurate but it comes at an additional computation cost. This additional computation cost is growing as the size of the map is growing. For that reason we apply a *bounding box* filter on the map, to bound the size and keep the number of points on the highest resolution constant. The results listed in Table 2 were obtained using 10 cm as the highest resolution and a bounding box filter keeping on average between 40k and 80k points for the highest resolution. We see that using these parameters, the map alignment takes 50 to 400 ms, depending on the number of iterations, typically between 3 and 5 for the highest resolution. In summary we can state that the whole mapping process takes 130 to 500 ms per frame.

5. CONCLUSION AND FUTURE WORK

In this work our spatial mapping and localization system Liborg was presented. The platform is able to acquire large-scale 3D models of both indoor and outdoor environments on the fly. The system allows to reconstruct the environment in 3D with minimal manual intervention. Experiments demonstrated that the accuracy is high while the execution time remains below 500 milliseconds per frame. In future work, we will improve the execution time even further by offloading some of the operations to the GPU, while of course still preserving the current accuracy. Second, we plan to mount a nVidia Jetson TX2 on the Liborg in order to make it able to do all of the processing on board instead of streaming it to a laptop or server. Finally, we want to add the automatic detection of loops and the adoption of loop closure to correct the small errors that are present.

REFERENCES

[1] Vlaminck, M., Luong, H. Q., Goeman, W., Veelaert, P., and Philips, W., "Towards online mobile mapping using inhomogeneous lidar data," in [*IEEE Intelligent Vehicles Symposium IV*], (June 2016).

[2] Thrun, S., Burgard, W., and Fox, D., "A real-time algorithm for mobile robot mapping with applications to multi-robot and 3d mapping," (2000).

[3] Bosse, M. and Zlot, R., "Continuous 3d scan-matching with a spinning 2d laser.," in [*ICRA*], 4312–4319, IEEE (2009).

[4] Zlot, R. and Bosse, M., [*Efficient Large-Scale 3D Mobile Mapping and Surface Reconstruction of an Underground Mine*], 479–493, Springer Berlin Heidelberg, Berlin, Heidelberg (2014).

[5] Magnusson, M., Duckett, T., and Lilienthal, A. J., "Scan registration for autonomous mining vehicles using 3D-NDT," *Journal of Field Robotics* **24**, 803–827 (Oct 24 2007).

[6] Zhang, J. and Singh, S., "Loam: Lidar odometry and mapping in real-time," in [*Robotics: Science and Systems Conference*], (July 2014).

[7] Jessup, J., Givigi, S. N., and Beaulieu, A., "Robust and efficient multi-robot 3d mapping with octree based occupancy grids," in [*Proc. of the IEEE International Conference on Systems Man and Cybernetics Conference*], 3996–4001 (2014).

[8] Wilkowski, A., Kornuta, T., Stefanczyk, M., and Kasprzak, W., "Efficient generation of 3d surfel maps using RGB-D sensors," *Applied Mathematics and Computer Science* **26**(1), 99–122 (2016).

[9] Steinbruecker, F., Sturm, J., and Cremers, D., "Volumetric 3d mapping in real-time on a cpu," (2014).

[10] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W., "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Auton. Robots* **34**, 189–206 (Apr. 2013).

[11] Droeschel, D., Stueckler, J., and Behnke, S., "Local multi-resolution representation for 6d motion estimation and mapping with a continuously rotating 3d laser scanner," in [*Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*], 5221–5226 (May 2014).

[12] Droeschel, D., Stückler, J., and Behnke, S., [*Local Multi-resolution Surfel Grids for MAV Motion Estimation and 3D Mapping*], 429–442, Springer International Publishing, Cham (2016).

[13] Einhorn, E., Schrter, C., and Gross, H.-M., "Finding the adequate resolution for grid mapping - cell sizes locally adapting on-the-fly.," in [*ICRA*], 1843–1848, IEEE (2011).

[14] Wurm, K. M., Hennes, D., Holz, D., Rusu, R. B., Stachniss, C., Konolige, K., and Burgard, W., "Hierarchies of octrees for efficient 3d mapping.," in [*IROS*], 4249–4255, IEEE (2011).

[15] Beno, P., Pavelka, V., Duchon, F., and Dekan, M., "Using octree maps and RGBD cameras to perform mapping and a* navigation," in [*2016 International Conference on Intelligent Networking and Collaborative Systems, INCoS 2016, Ostrawva, Czech Republic, September 7-9, 2016*], 66–72 (2016).