IEEE Transactions on Intelligent Vehicles, Volume 9, Issue 1, January 2024, pp. 2999-3009
DOI: 10.1109/TIV.2023.3250326

NDT RC: Normal Distribution Transform Occupancy 3D Mapping with Recentering

Hugo Courtois*, Nabil Aouf†, Kenan Ahiska‡ and Marco Cecotti§

***Abstract*—The Normal Distribution Transform Occupancy Map (NDT OM) is a mapping algorithm able to represent a dynamic 3D environment. The resulting map has fixed boundaries, thus a robot with unbounded displacement might fall outside of the map due to memory limitation. In this paper, a recentering algorithm called NDT RC is proposed to avoid this issue. NDT RC extends the use of NDT OM for vehicles with unbounded displacements. NDT RC provides a seamless translation of the map as the robot gets far from the center of the previous map. The influence of NDT RC on the precision of the estimated trajectory of the robot, or odometry, is examined on two publicly available datasets, the KITTI and Ford datasets. An analysis of the sensitivity of the NDT RC to its tuning parameters is carried out using the Ford dataset, while the KITTI dataset is used to measure the influence of the density of the input point cloud. The results show that the proposed recentering strategy improves the accuracy of the odometry calculated by registering the latest lidar scan on the generated map compared to other NDT based approaches (NDT OM, NDT OM Fusion, SE-NDT). In particular, the proposed method, which does not perform loop closure, reduces the mean absolute translation error by $16\,\%$ and the runtime by $88\,\%$ compared to the NDT OM Fusion on the Ford dataset.**

***Index Terms*—NDT, occupancy mapping, KITTI dataset, Ford dataset, recentering.**

I. Introduction

The creation and maintenance of a reliable map of obstacles around a robot is a core problem for many autonomous robotic applications. In particular, maps can provide tools to compute odometry or avoid obstacles. In this paper, odometry is defined as the set of poses of the robot computed from scan registration. Since robots have limited memory and computational power, a stationary map restricts the radius of action of a robot to the size of the map. This is an inherent problem of popular mapping algorithms such as Octomap [1] or NDT OM [2]. In order to overcome this problem, we propose to displace the map so that it remains centered on the robot. This prevents the robot from falling outside of the boundaries of the map.

Desirable properties for a map to be used in robotic applications are 3D capabilities, real time update rate and adaptability to dynamic changes in the environment. A widely used method to represent the environment is the occupancy grid map [3], [4]. The original occupancy grid map was developed in 2D. There are several solutions to extend it to 3D. One of them is the elevation map, which uses a 2D grid and adds an associated height for each cell (e.g., [5]). Although efficient in memory, this approach only allows a single point for a whole spatial column. This makes structures like windows difficult to represent which is a problem for aerial robots.

Octomap [1], as a 3D implementation of occupancy maps, uses octrees to efficiently provide multi-resolution support. Despite this multi-resolution capability, Octomap operates under the assumption that a cell is either fully occupied, or fully empty. This means that to provide an accurate representation of the world, the size of a single cell of the map has to be smaller than a typical feature of the environment.

A solution to use fewer cells is the Normal Distribution Transform (NDT), introduced in the 2D case [6], and extended to 3D by Magnusson, Lilienthal, and Duckett [7]. The NDT operates on a point cloud by computing the mean and covariance of the points inside each spatial cell. The obtained NDT representations of two point clouds can then be matched in order to find the rigid transformation between them. An experimental study on NDT [8] shows that they can be as accurate as occupancy grids of finer resolutions. The combination of the NDT representation with occupancy mapping update is NDT OM [2], which introduces a recursive update of the map. NDT OM allows NDTs to be fused into a unique map. NDT OM is attractive for several reasons:

1) The points from the point cloud are not kept in memory. Only 11 parameters are required per occupied cell (3 for the mean, 6 for the covariance, 1 for the number of points, 1 for the occupancy value).
2) NDT representation can be used to retrieve the odometry. The direct registration of two NDT models is described in [9] and analytical derivatives are provided for the given objective function.
3) Multi-resolution is available due to the recursive update scheme.
4) The obstacle inside a map cell is described by a Gaussian distribution. This results in a map that can represent the environment more accurately and with less cells than a standard occupancy grid map.

Several methods have been proposed to improve the scan registration process with NDTs: the distribution to distribution [9] (D2D) algorithm matches two NDTs together, which is faster and more precise than matching a point cloud with an NDT. The Segmented Region Growing NDT (SRG-NDT) [10] adds two main modifications to the D2D NDT. First, the ground points are removed from the scans. Second, the remaining points are clustered using a region growing algorithm.

Manuscript received on

*Hugo Courtois is with Outsight, 2 rue de Berite, 75006, Paris, France
†Nabil Aouf is with the Department of Electrical and Electronics Engineering, City University of London, London, UK
‡Kenan Ahiska is with the Centre for Electronic Warfare, Information and Cyber, Cranfield University, Defence Academy of the UK, Shrivenham, UK
§Marco Cecotti is with the Advanced Vehicle Engineering Centre, Cranfield University, Cranfield, UK

This work was part of a PhD thesis at the Centre for Electronic Warfare, Information and Cyber, Cranfield University, Defence Academy of the UK, Shrivenham, UK. This work was supported by the Engineering and Physical Sciences Research Council [grant number EP/K504324/1].

Those two operations reduce the runtime and increase the precision compared to D2D NDT. The SE-NDT [11] uses semantic information to improve the matching stage. Two NDTs are created, from points belonging to edges and planes. Then the matching is performed between two NDTs of the same type. An improvement can be brought to the construction of the Gaussians inside a cell: using a probabilistic model of the sensor to define the spatial probability distribution around individual data points, allows Gaussians to be computed from a single point. This increases the precision of the matching process, but slows the algorithm down. The matching process can also be improved indirectly by using NDT OM: as several scans are accumulated into the map, the precision of the map increases. However, since the map used in NDT OM has spatially fixed boundaries, such a strategy is restricted to a fixed spatial zone due to memory limitations.

A solution to this problem is proposed with NDT OM Fusion [12], where the map is divided into tiles. The tiles are saved on the disk when the vehicle is far enough from them. Provided enough disk space, this method allows for greater movement of the robot. In some applications such as obstacle avoidance, it is however not necessary to keep the map of all the traversed areas. In this case, there is no need to use disk space, which would not only reduce the memory requirement, but also speed up the algorithm since retrieving data on the disk is slower than fetching it from the memory. Moreover, a problem noticed with NDT OM Fusion is the loop closure: if the formerly visited areas are stored, when the robot goes back to the same area, the matching is more difficult if the odometry drifted [12]. The proposed algorithm minimizes this issue by considering a smaller map, centered on the robot, meaning that the whole map is constantly updated according to the occupancy scheme. The long term odometry drift is then less relevant, since there is no matching attempt between the current map and former areas falling outside the current map.

Note that the two approaches to improve the quality of the registration, by improving the matching process or by fusing the consecutive NDTs, are compatible, even though combining them is outside the scope of this paper.

A well known weakness of pure odometry methods, which is the case of all NDT based algorithms described above including NDT RC, is a drift over long trajectory, which has been tried to be solved with loop closing mechanisms. Loop closing with NDT has been done through pose graph optimization [13], by accumulating partial NDTs resulting in the accumulation of several scans. Each partial NDT is associated with a vertex in a pose graph, and partial maps are registered if an overlap is likely. In the work of Zaganidis, Zerntev, Duckett, *et al.* [14], NDT are classified in 3 categories: planar, linear or spherical. Histograms of those categories are then built and used as descriptors. The close descriptors to the current robot location are then considered, and if the central descriptor and the considered one are close enough, a registration is performed between the corresponding maps.

In this paper, we focus on the NDT family of algorithms since it uses the same representation for modeling the environment and computing odometry. This is interesting for applications that can make use of both features, such as obstacle avoidance, where the sensor limited field of view and resolution can be compensated by a mapping algorithm [15].

Other approaches having this same advantage have been developed. In the work of Droeschel, Nieuwenhuisen, Beul, *et al.* [16], the 3D points are aggregated to form surfels. Sets of surfels are matched using an expectation-maximization to form a multi resolution map, which is then used for navigation. This map can then be shifted using a ring buffer strategy to maintain the center on the robot. The work of Behley and Stachniss [17] combines several interesting properties: a surfel map is built from laser data, allowing odometry computation and map update at every iteration. Moreover, loop closure is performed to improve global map consistency. This method projects the 3D laser scan into 2D and computes normals, making it more difficult to use with LiDARs exhibiting non regular patterns (e.g., livox avia). It should be noted that although the method runs in real time, it leverages the rendering pipeline of a GPU, making it more costly to implement on hardware limited to a CPU.

It is also possible to perform the odometry computation separately from the map and obstacles representation. The current most precise method on the KITTI odometry dataset [18] using Laser data only is LOAM (LiDAR Odometry And Mapping) [19], [20]. It divides the mapping task in two: a coarse odometry step running at the sensor speed, and a slower matching and registration step. This method uses 3D features, corners and surfaces, to register scans together. The fast odometry step can take advantage of an IMU if available. While this method provides accurate odometry, the mapping step cannot be performed for each scan in real time.

The contribution of this paper is twofold. First, an algorithm, NDT RC, is proposed. It aims at solving the aforementioned problem with fixed boundaries of NDT OM, without using the disk as in NDT OM Fusion. The advantages compared to NDT OM Fusion are faster processing and the absence of a long term loop closure issue. The disadvantage is that the full map is not available anymore, which is not an issue for applications such as obstacle avoidance. In order to test the proposed algorithm, the precision of the odometry is used as a metric. This allows to compare NDT RC with NDT OM Fusion, but also with the algorithms improving the scan registration part, including the D2D NDT, which is used as a baseline. The large scale datasets used are the Ford Campus Vision and Lidar Data Set [21] (Ford dataset) and the KITTI odometry dataset [18]. Second, an analysis of the sensitivity of this algorithm to its tuning parameters and the density of the lidar is carried out. The NDT RC, similar to the NDT OM, depends on several parameters, and to the best of the authors knowledge, the influence of those parameters on the precision of the odometry has not been evaluated.

The NDT RC algorithm has been used in OAST [22], an algorithm for obstacle avoidance during teleoperation of UAVs. This paper focuses on the mapping algorithm, while the aforementioned paper presents the obstacle avoidance algorithm.

The theory behind NDT OM is described in Section II, then NDT RC is described in Section III. NDT RC is tested on two publicly available datasets in Section IV and the influence of its tuning parameters and of the density of the LIDAR is examined in Section V.

II. Background: NDT OM

From a sequence of point clouds, the NDT OM maintains a map representing the environment of the robot by a list of normal distributions.

A. Fusion of two NDTs

The 3D NDT is a spatial representation of a scene built from a point cloud [7]. The space is divided in cubic cells $\mathcal{C}_i$ containing $|\mathcal{C}_i|$ 3D points $\mathbf{p}$. Then for each cell, the 3 by 1 mean vector $\boldsymbol{\mu}_i$ and the 3 by 3 covariance matrix $\boldsymbol{\Sigma}_i$ of the points inside the cell are computed:

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{C}_i|} \sum_{\mathbf{p} \in \mathcal{C}_i} \mathbf{p}, \tag{1}$$

$$\boldsymbol{\Sigma}_i = \frac{1}{|\mathcal{C}_i| - 1} \sum_{\mathbf{p} \in \mathcal{C}_i} (\mathbf{p} - \boldsymbol{\mu}_i)(\mathbf{p} - \boldsymbol{\mu}_i)^T. \tag{2}$$

For a given point $\mathbf{x}$, the probability to belong to the obstacle in cell $\mathcal{C}_i$ is defined by a normal distribution of mean $\boldsymbol{\mu}_i$ and covariance $\boldsymbol{\Sigma}_i$: $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$.

When a new point cloud is available, its NDT representation is computed as described above. The D2D algorithm [9] can then be used to register two NDT representations. Once the rigid transformation between the two NDT representation has been computed, they can be fused using the Recursive Sampled Covariance (RSC) scheme [2]. The RSC scheme aims to fuse two corresponding cells $\mathcal{C}_i$ and $\mathcal{C}_j$ of mean and covariance $(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$ and $(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$ respectively in a new cell $\mathcal{C}_{ij}$ with $|\mathcal{C}_{ij}|$ points and mean and covariance $(\boldsymbol{\mu}_{ij}, \boldsymbol{\Sigma}_{ij})$. The RSC computes $\boldsymbol{\mu}_{ij}$, $\boldsymbol{\Sigma}_{ij}$ and $|\mathcal{C}_{ij}|$ as:

$$\boldsymbol{\mu}_{i+j} = \frac{|\mathcal{C}_i|\,\boldsymbol{\mu}_i + |\mathcal{C}_j|\,\boldsymbol{\mu}_j}{|\mathcal{C}_i| + |\mathcal{C}_j|}, \tag{3}$$

$$\begin{aligned} \boldsymbol{\Sigma}_{i+j} = \frac{1}{|\mathcal{C}_i| + |\mathcal{C}_j| - 1} \Bigg[& \left(|\mathcal{C}_i| - 1\right) \boldsymbol{\Sigma}_i + \left(|\mathcal{C}_j| - 1\right) \boldsymbol{\Sigma}_j \\ & + \frac{|\mathcal{C}_i|\,|\mathcal{C}_j|}{|\mathcal{C}_i| + |\mathcal{C}_j|} \left(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right) \left(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\right)^t \Bigg], \end{aligned} \tag{4}$$

$$|\mathcal{C}_{ij}| = |\mathcal{C}_i| + |\mathcal{C}_j|. \tag{5}$$

A weakness of the RSC is the unbounded growth of the number of points in a cell. In practice, those values will overflow with enough runtime. To solve this, Adaptive Recursive Sample Covariance (ARSC) was developed [2], which applies a sliding average to the mean and covariance of a cell when the number of points hits a user defined threshold. The idea is to define a threshold M_{max} for the number of points. It is considered that above this threshold the influence of n additional points, with $n \ll M_{max}$, bring small changes to the mean and covariance of the cell. When the number of points in a cell goes above M_{max} by n, the mean and covariance are scaled by $M_{max}/\left(M_{max}+n\right)$ and $\left(M_{max}-1\right)/\left(M_{max}+n-1\right)$ respectively, and the total number of point is then capped at M_{max}.

B. Occupancy update

The lidar provides measurements, which are 3D points $\mathbf{z}_t$ at time t. The occupancy is the probability for a cell $\mathcal{C}_i$ to be occupied given the available measurements $\mathbf{z}_{1:t}$ at time t: $p\left(\mathcal{C}_i|\mathbf{z}_{1:t}\right)$. Similar to the occupancy mapping theory [23], the occupancy of each NDT cell on the path of a ray from the LIDAR is reduced while the occupancy of a cell containing a LIDAR point is increased. Using the log odd function $l_{\text{odd}}\left(x\right) = \log\left(\frac{x}{1-x}\right)$, the log odd occupancy is updated using:

$$l_{\text{odd}}\left(p\left(\mathcal{C}_i|\mathbf{z}_{1:t}\right)\right) = l_{\text{odd}}\left(p\left(\mathcal{C}_i|\mathbf{z}_{1:t-1}\right)\right) + l_{\text{odd}}\left(p\left(\mathcal{C}_i|\mathbf{z}_t\right)\right). \tag{6}$$

Since each cell can contain a normal distribution, Saarinen, Andreasson, Stoyanov, *et al.* [2] propose to compute the probability $p\left(\mathcal{C}_i|\mathbf{z}_t\right)$ by estimating the compatibility between the trajectory of the ray and the Gaussian inside the cell. The occupancy is reduced by a greater amount when there is an inconsistency between the map and the LIDAR scan. Let $\mathbf{x}_M$ be the point on the ray defined by the measurement $\mathbf{z}_t$ that maximizes the probability $p\left(\mathbf{x}|\mathcal{N}\left(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)\right)$. This point can be computed analytically [2]. The occupancy update for cells that contain a normal distribution then relies on two quantities: the probability that $\mathbf{x}_M$ can be explained by the distribution inside the cell and the probability that $\mathbf{x}_M$ can be explained by the measurement $\mathbf{z}_t$ ($p\left(\mathbf{x}_M|\mathbf{z}_t\right)$). This last probability is computed by assuming that the distance between $\mathbf{x}_M$ and $\mathbf{z}_t$ can be approximated by a Gaussian distribution centred on $\mathbf{z}_t$, with a variance σ_s^2 coming from the sensor ($p\left(\mathbf{x}_M|\mathbf{z}_t\right)$) [2]:

$$p\left(\mathbf{x}_M|\mathbf{z}_t\right) = \frac{1}{\sqrt{\left(2\pi\right)\sigma_s^2}} \exp\left(-\frac{\|\mathbf{x}_M - \mathbf{z}_t\|^2}{2\sigma_s^2}\right). \tag{7}$$

For instance, a high probability $p\left(\mathbf{x}_M|\mathcal{N}\left(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)\right)$ (the ray is likely to go through an obstacle) paired with a low probability $p\left(\mathbf{x}_M|\mathbf{z}_t\right)$ (this obstacle is far from the endpoint of the ray) means that the confidence in the presence of an obstacle in the cell should be reduced. On the other hand, a high probability $p\left(\mathbf{x}_M|\mathcal{N}\left(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)\right)$ paired with a high probability $p\left(\mathbf{x}_M|\mathbf{z}_t\right)$ is coherent and indicates the likely presence of an obstacle in the cell. The occupancy update of the cell $\mathcal{C}_i$ is then updated as follows [2]:

$$p\left(\mathcal{C}_i|\mathbf{z}_t\right) = \begin{cases} \alpha \text{ if the cell is empty (with } \alpha < 0.5\text{), else} \\ \beta \text{ if } \mathbf{z}_t \text{ is in the cell (with } \beta > 0.5\text{), else} \\ 0.5 - \gamma p\left(\mathbf{x}_M|\mathcal{N}\left(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i\right)\right)\left(1 - p\left(\mathbf{x}_M|\mathbf{z}_t\right)\right). \end{cases} \tag{8}$$

If the cell is empty or contains $\mathbf{z}_t$ (thus is considered occupied), the update is the same as standard occupancy mapping. If the cell contains a distribution, but not $\mathbf{z}_t$, then the log odd occupancy will be lowered by an amount that depends on the consistency between the ray and the distribution inside the cell. The more consistent they are, the closer the occupancy will be to 0.5. The tuning constants α and β are positive in the interval $[0, 1]$ and dictate the speed at which the cell occupancy changes. The tuning constant γ encodes the speed at which a partially occupied cell is penalized. The quantity $l_{\text{odd}}\left(p\left(\mathcal{C}_i|\mathbf{z}_t\right)\right)$ is then clamped to the interval $[-K_{\text{occ}}, K_{\text{occ}}]$ before being used in (6), K_{occ} being a tuning constant. The influence of those parameters, among others, is studied in Section V.

The occupancy update should be performed for each point from the point cloud, using a raytracing algorithm to find each cell on its path. It is however possible to speed this part of the algorithm considerably by using the NDT built from the new point cloud [24]. Instead of considering each point $\mathbf{z}_t$ from the point cloud, the mean $\boldsymbol{\mu}_i$ of each distribution is used $|\mathcal{C}_i|$ times.

The result of this process is an updated NDT map containing normal distributions in the occupied cells and a probability of each cell being occupied.

III. The recentering algorithm: NDT RC

The map resulting from the NDT OM algorithm as presented above is immobile. In order to allow unlimited movement for a mobile robot either ground or aerial, we propose to move the map with the vehicle. The idea is that each time the robot moves further than a threshold d_{thrc} from the center of the map, the map is centered on the new location of the robot. The map can only translate, it cannot rotate.

Three data structures are used to represent the environment: the map M, the list of active cells L_{ac} and the list of deleted cells L_{dc}. An NDT structure built from a point cloud can then be represented by a triplet (M, L_{ac}, L_{dc}). The first element is the map itself, which only hosts indices, but no actual data. Initially, the map is full of zeros. Non-zero indices in the map link each cell to a second array: the list of active cells. Each active cell contains 11 parameters. A cell becomes active if one of those two conditions is met: there is a 3D point to add to the cell or the raytracing algorithm passes through the cell. In the second case, the mean of the cell is set by default to the center of the cell. The coordinates represented by the mean enable the location of active cell on the map, even if there are no 3D points associated to the cell. This creates a bidirectional link between the map and the list of active cells. Once a cell is active, it remains active until it goes out of the map limits.

For performances reasons, it is better to keep the active cells grouped in memory, meaning that they are stored in an array and not a linked list. When an active cell is deleted, its index in the array is added to the list of deleted cells. When a new active cell is added, if the list of deleted cells is non empty, then its first index is chosen to host the new data of the active cell. If the list of deleted cells is empty, then the list of active cells is extended. This allows to keep a list of active cells that is dense. The usage of a list of active cells to store the actual data reduces the memory requirements compared to storing the data directly in the map: for standard sizes of 64 bits for a floating point number and 32 bits for an integer, the proposed scheme is more efficient in memory until the map reaches more than $85\,\%$ of the cells occupied. In a realistic environment, such a number is unlikely to be reached.

The objective of the recentering algorithm is to move all the cells in the map in an efficient way in order to maintain the sensor at the center of the map. Since only translation of the map is performed, this is done by sliding the cells one by one in a fixed direction. Let this direction vector be $\mathbf{d}$. Such a process is illustrated by Fig. 1.

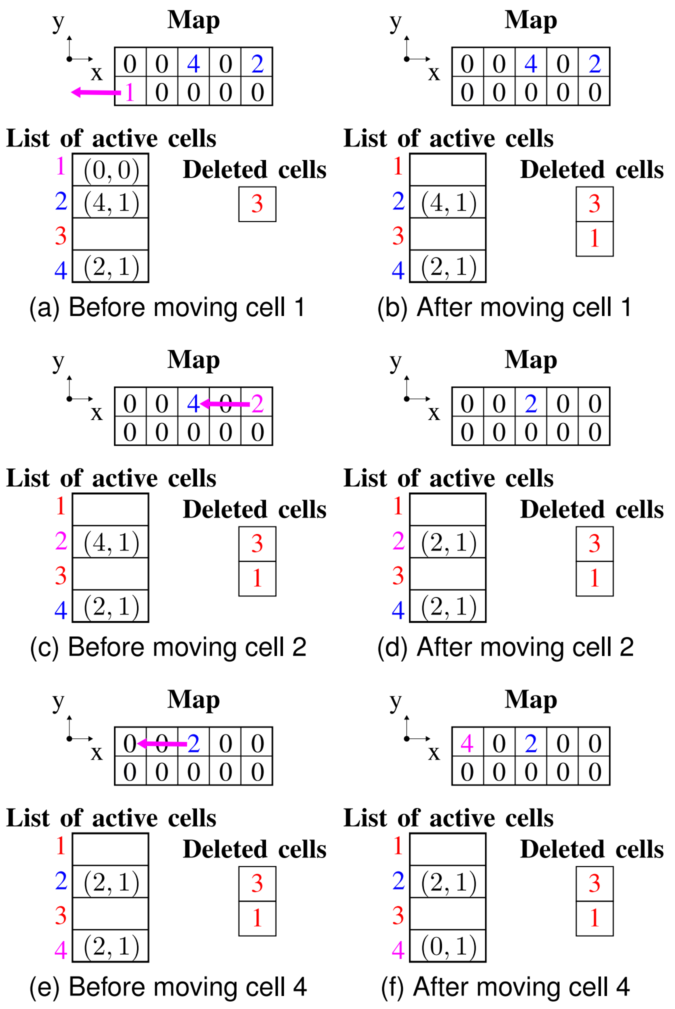


Fig. 1. Illustration of the recentering algorithm for a considered displacement $\mathbf{d}$ of $(-2, 0)$ and a map of 5×2 cells. The active cell to be moved is in pink, the indices of deleted cells are in red, and the other cells are in blue.

The recentering algorithm operates sequentially on individual cells, in the order given by the index of the list of active cells, by updating the values across M, L_{ac}, and L_{dc}. First, the algorithm checks whether the cell should be moved outside of the current limits of the map and, if this is the case, adds the cell index to the list of deleted cells. Then, there are two possibilities: either the index in the map matches the index of the considered active cell, or it does not. 1) If the index stored in the map cell is the actual index of the considered active cells, then it is moved at its new location in the map (note that there can be a new index here, in this case it is overwritten, see the second point). This new location is computed by sliding the map index by $\mathbf{d}$. If the new location is outside the map, the index is instead added to the list of deleted cells. In both cases, the former index in the map is deleted. 2) If the index stored in the map cell does not match the active cell, this implies that the index of this active cell was overwritten by the previous displacement of another cell. In this case, the index of the active cell is still copied to its new location, but the initial value is not erased since it represents the index of a cell that has already been moved. If the new location is outside the map, the index of the active cell is added to the list of deleted cells.

The recentering process is illustrated for the 2D case in Fig. 1 for a map displacement of $(-2, 0)$. Fig. 1a depicts the initial state. There is already one deleted cell, and the list of active cells contains, among others, the mean values to go back to the map coordinates. In order to simplify the depiction, the map coordinates are directly represented in the list of actives cells. The active cell number 1 (Fig. 1a), due to the displacement of the map, goes out of it. Since the cell of coordinates $(0, 0)$ in the map hosts the index 1, this index can be erased. Because this index goes out of the map, it is added to the list of deleted cells. The result is depicted in Fig. 1b. Now, the next cell in the list is cell number 2 (Fig. 1c). Since the cell of coordinates $(4, 1)$ hosts the index 2, actual index of the current active cell, it can be erased and moved to its next destination, $(2, 1)$ where it overwrites 4. The internal coordinates of cell 2 are modified as well (see Fig. 1d). The next active cell, 3 is skipped because the cell is deleted. The remaining cell is the number 4 (Fig. 1e). This time, the map coordinates of cell 4 are $(2, 1)$, but at those coordinates, the index in the map is 2, and not 4. Thus, the content of the map cell $(2, 1)$ is *not* erased, but 4 is still placed into the map cell $(0, 1)$, and the content of the active cell is updated accordingly (see Fig. 1f).

The complete mapping algorithm is illustrated in Algorithm 1. Upper indices indicate a reference frame: s for objetcs in the sensor reference frame (e.g., a LIDAR), m for the map reference frame or w for the world reference frame. The map is always aligned with the world reference frame, but can be translated due to the recentering algorithm shown above, while the sensor reference frame is attached to the robot body frame. A rigid transform $\mathbf{T}_t^{s\to m}$ transforms a point $\mathbf{x}_t^s$ in the sensor reference frame into the map reference frame: $\mathbf{T}_t^{s\to m}\mathbf{x}_t^s = \mathbf{x}_t^m$. For two times t_1 and t_2, the rigid transform $\mathbf{T}_{t_1\to t_2}^{i\to j}$ is defined as: $\mathbf{T}_{t_1}^{i\to j} = \mathbf{T}_{t_2}^{i\to j}\mathbf{T}_{t_1\to t_2}^{i\to j}$.

As future work, loop closure using NDT RC could be implemented in a similar way to [13], using several locally consistent small NDT maps. That would require storing the discarded cells from the list of active cells and a global map storing the location of each fragment. A specific case is to assume that the operating environment of the robot is fully known beforehand, which could be the case in industrial use cases. Then, the NDT of the global map can be computed beforehand, using the memory layout of NDT RC to reduce the memory requirement ofr large areas. Drift can then be avoided without loop closure by matching each new NDT with the global map. This case is of particular importance for applications where reliability and fixed runtime are required.

Algorithm 1: The complete mapping algorithm: NDT RC

```
Input  : A point cloud $\mathcal{P}_t^s$ in the sensor reference
         frame at time step $t$
Output: $M$, $L_{ac}$, $L_{dc}$ are updated and $\mathbf{T}_t^{m\to w}$,
         $\mathbf{T}_t^{s\to m}$, $\mathbf{T}_{t\to t-1}^{s\to m}$ are created
   // Compute initial guess using the
      previous sensor displacement
1  $\tilde{\mathcal{P}}_t^m = \mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}\mathcal{P}_t^s$;
2  Build $\tilde{M}$ and $\tilde{L}_{ac}$ from $\tilde{\mathcal{P}}_t^m$ (see Section II-A for NDT
     building and Section III for the map architecture);
3  Register $\{\tilde{M}, \tilde{L}_{ac}\}$ with $\{M, L_{ac}\}$ using D2D NDT,
     the output is $\mathbf{T}^\Delta$;
   // Update the position of the robot,
      as well as the next inter frame
      transform
4  $\mathbf{T}_t^{s\to m} = \mathbf{T}^\Delta\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}$;
5  $\mathbf{T}_{t\to t-1}^{s\to m} = \left(\mathbf{T}_{t-1}^{s\to m}\right)^{-1}\mathbf{T}^\Delta\mathbf{T}_{t-1}^{s\to m}\mathbf{T}_{t-1\to t-2}^{s\to m}$;
   // Create the actual point cloud that
      will be added
6  $\mathcal{P}_t^m = \mathbf{T}_t^{s\to m}\mathcal{P}_t^s$;
   // Build an NDT representing the new
      scan correctly aligned
7  Build $M^m$ and $L_{ac}^m$ from $\mathcal{P}_t^m$;
   // Fuse $\{M^m, L_{ac}^m\}$ with $\{M, L_{ac}\}$
   // Start with the raytracing to
      update the occupancy
8  for all cells $\mathcal{C}^* \in L_{ac}^l$ do
9      Get the number of points $n$ and mean $\boldsymbol{\mu}$ of cell $\mathcal{C}^*$;
       // $M$ is used to perform the
          raytracing
10     Get all the cells $\mathcal{C}_1, \cdots, \mathcal{C}_k$ in $L_{ac}$ traversed by the
         ray from the sensor to $\boldsymbol{\mu}$;
11     for each cell $\mathcal{C}_{i,i\in 1:k}$ do
12         Update the log odd occupancy of $\mathcal{C}_i$ with
             equation (6) modified as follows:
             $l_{\text{odd}}(\text{occ}_i(t)) =$
             $l_{\text{odd}}(\text{occ}_i(t-1)) + n l_{\text{odd}}(\mathcal{C}_i|\boldsymbol{\mu}, l_t)$ ;
13     end
14 end
   // Then fuse the cells themselves
15 for all cells $\mathcal{C} \in L_{ac}^l$ do
16     Fuse $\mathcal{C}$ into $L_{ac}$ using RSC or ARSC (see
         Section II);
17 end
   // Performs recentering if needed,
      see Fig. 1
18 $recenter\_map(\mathbf{T}_t^{s\to m}, L_{ac}, L_{dc}, M)$;
```

IV. Verification of NDT RC

In this section, the odometry produced by the NDT RC algorithm is tested on two publicly available datasets: the Ford Campus Vision and Lidar Data Set [21] and the KITTI odometry dataset [18]. The Ford dataset is chosen as the competitor technique NDT OM [2] was already evaluated

TABLE I
VALUES OF THE PARAMETERS USED FOR THE NDT RC

Parameter	Value
Cell size	$2.2\,\mathrm{m}$
α	0.45
β	0.9
γ	0.1
Map width and depth	$250\,\mathrm{m}$
Map height	$40\,\mathrm{m}$
d_{thrc}	$10\,\mathrm{m}$
M_{max}	500

on it with a fusion mechanism to allow the robot to have a greater movement range [12]. The KITTI dataset [18] has been chosen because it is popular, allowing our results to be compared to other methods. Both datasets are acquired outdoors using a car. The Ford dataset was acquired in urban conditions, while the KITTI dataset contains 11 sequences acquired in various conditions. The LIDAR sensor used is a Velodyne HDL-64E, with a range of $120\,\mathrm{m}$ and a rotation rate of $10\,\mathrm{Hz}$. The odometry from the D2D NDT [25] is shown as a baseline. The D2D NDT is tuned by making the parameters vary on a grid, similarly to our algorithm. However, the best parameters for D2D NDT are chosen for each dataset, whereas our algorithm uses a single set of parameters for the two datasets. Moreover, to guarantee a fairness of evaluation, we ensured that the results of D2D NDT were at least as good as existing publications applying D2D NDT to those datasets when available [11]. It is important to note that since our algorithm and the D2D NDT are different, using the same parameters would be unfair, which is why they were tuned separately. Our testing indeed shows [26] that the optimal set of parameters for each algorithm are different, thus we have elected to evaluate each of them on the parameters giving the best results and not on the same parameters.

The parameters used for the NDT RC are presented in Table I. They are selected in accordance with the results of the sensitivity analysis in Section V. Note that the results reported on both the KITTI and Ford datasets are obtained with those same parameters. The size of the map allows a whole LIDAR scan to fit inside the map when the vehicle is at the centre. Moreover, the recentering algorithm is called when the vehicle is further than $10\,\mathrm{m}$ from the centre. A study of those parameters is proposed in Section V, where the influence of each of them over the precision and runtime can be seen. The initial guess for initializing the D2D matching process [9] between the current NDT map and the NDT built from the incoming point cloud is the last inter frame odometry resulting from a successful match. The runtime shown is the wall clock time to give a realistic representation of the speed of the algorithm. NDT RC, implemented in C++, runs on a single CPU thread. The computer used to run the tests has an Intel Core i7-6700 CPU and $16\,\mathrm{GB}$ of RAM.

A. Evaluation on the Ford dataset

The main error metric on this dataset is the absolute trajectory error [27] (ATE) per frame. The ATE is obtained after aligning the odometry trajectory obtained using the registration of maps created by NDT RC and the ground truth trajectory [27], and measures the absolute error in translation. Different ATE statistics are shown in Table II. The runtime averaged over the whole sequence is detailed in Table III. The time needed by the recentering algorithm is negligible ($< 0.5\,\%$) compared to the total time. In the rest of this section, the NDT RC algorithm is compared to other algorithms on the Ford dataset. The NDT RC is compared to the D2D NDT [25] as implemented on the Github repository of the authors[1]. The following parameters are used for the D2D NDT: a single 3D matching, a resolution of $1.5\,\mathrm{m}$, a matching neighbourhood size of 2, a map width of $250\,\mathrm{m}$ and a map height of $40\,\mathrm{m}$. For both the D2D NDT and the NDT RC, the initial guess for the matching is the result of the previous successful matching. The comparison between the resulting NDT RC, D2D NDT based odometry trajectories and the ground truth trajectory is shown in Fig. 2a. The trajectory computed by NDT RC is close to the ground truth as suggested by the small ATE.

TABLE II
ERROR STATISTICS APPLIED TO THE ATE OF NDT RC ON THE FORD DATASET (IN METERS)

Statistic	Mean	RMS	Median	Std dev	Min	Max
Value	1.42	1.49	1.49	0.45	0.55	2.38

TABLE III
AVERAGE RUNTIME OF NDT RC ON THE FORD DATASET (IN MILLISECONDS)

Operation	Duration
Scan importation	4
Matching	218
Map update	5
Recentering	<1
Total	229

The second NDT based algorithm is the NDT OM Fusion [12], which is based on NDT OM. The algorithm stores the NDT maps on the disk when the robot leaves the zone, and reload them when needed. Note that the Xsens MTi-G IMU mounted on the car is used with the NDT OM Fusion [12] to provide an initial guess, while our algorithm does not use any additional sensor. The comparison is summarized in Table IV. The runtime of the most precise result is $229\,\mathrm{ms}$ for our method and $263\,\mathrm{ms}$ for D2D NDT. However, since the cell size is the major parameter influencing the runtime, the runtime cannot be compared directly, because the cell sizes are different. In order to get a fair comparison, the runtime with a fixed cell size of $1.5\,\mathrm{m}$ is shown in Table IV. With equal cell size, the D2D NDT is faster than NDT RC, which is expected since NDT RC adds a costly raytracing operation to D2D NDT. However, NDT RC is able to use higher cell size while keeping a low ATE, making it faster than D2D NDT for the most precise result.

The NDT RC outperforms NDT OM Fusion in ATE and is faster. The difference in runtime can be explained by the recentering algorithm, which removes the need for disk accesses. The fact that NDT RC, which discards the cells that move out of the boundaries of the map results, achieves a

[1]https://github.com/OrebroUniversity/perception_oru/tree/port-kinetic, last pull done on Tuesday 27th November, 2018

TABLE IV
COMPARISON BETWEEN NDT RC AND OTHER METHODS ON THE FORD DATASET

Method	Mean ATE (m)	RMS ATE (m)	Runtime (ms)
NDT RC	**1.42**	**1.49**	229 (360)[1]
D2D NDT [25]	10.8	11.65	**263**
NDT OM Fusion [12]	1.7	-	<2000
SLAM method [28]	-	4.48	<300

[1] with $1.5\,\mathrm{m}$ cell size.

more precise odometry compared to NDT OM Fusion, which keeps all the cells, can appear surprising. This improvement is considered to be partly due to the loop closure issue mentioned in the work of Stoyanov, Saarinen, Andreasson, *et al.* [12]. Indeed, keeping all the cells mean that any drift in odometry will make the matching more difficult for NDT OM Fusion in case of loop closure. NDT RC does not have this problem since it does not keep the cells outside the boundaries of the map.

A SLAM method based on the extraction of planar segments from the LIDAR data is proposed by Lenac, Kitanov, Cupec, *et al.* [28]. The authors report the RMSE on this dataset. The RMSE for their method is $4.48\,\mathrm{m}$. The mean runtime reported are around $250\,\mathrm{ms}$ for point cloud segmentation, $1.5\,\mathrm{ms}$ for relative pose computation, and less than $50\,\mathrm{ms}$ for the global map update. However, because this is a SLAM system, the global map runtime can increase with time. A spike in runtime of more than $600\,\mathrm{ms}$ can be seen at the end of the trajectory (Figure 33 [28]). NDT RC has a lower RMSE of $1.49\,\mathrm{m}$, and a faster runtime of $229\,\mathrm{ms}$. This comparison is interesting because it shows that a lower error is obtained with NDT RC despite the absence of loop closure mechanism. Those results are summarized as well in Table IV.

B. Evaluation on the KITTI dataset

The NDT RC is evaluated on the KITTI dataset as well. The resulted odometry translation and rotation errors are computed by using the code provided with the dataset [29]. The details can be found in the original publication [30]. To summarize, the errors in rotation and translation are computed over segments of varying lengths (from $100\,\mathrm{m}$ to $800\,\mathrm{m}$ by increment of $100\,\mathrm{m}$) and then averaged over all segments in all sequences. There are 11 sequences in this dataset.

The NDT RC is compared to the D2D NDT. The D2D NDT is configured with a cell size of $0.5\,\mathrm{m}$. The error obtained with this configuration is inferior to the error reported in the work of Zaganidis, Magnusson, Duckett, *et al.* [11], which validates this choice of parameters. A NDT variation has been proposed with good results on the KITTI dataset: the SE-NDT [11], described in the introduction. Since the SE-NDT has been evaluated on the KITTI dataset by using the standard error measures, the results are comparable to ours. Thus, the results of the SE-NDT paper [11] are printed here. The runtime is not reported directly since the comparison does not appear to be valid. Indeed, in their paper, the SE-NDT algorithm takes $2.82\,\mathrm{s}$ while the D2D NDT takes $4.15\,\mathrm{s}$. In our case, the D2D NDT, with the same set of parameters, runs in less than $0.5\,\mathrm{s}$. Such a difference might be explained by hardware or implementation differences, but this is impossible to validate without access to SE-NDT implementation. For this reason, it was chosen not to report a potentially misleading runtime result. The results are compiled into Table V. The proposed method outperforms both the D2D NDT and the SE-NDT for translational and rotational error while being faster than D2D NDT due to the higher cell size. In order to visualize the results, the trajectory of the proposed method and the D2D NDT are plotted against the ground truth in Figs. 2b to 2l. It can be seen in Fig. 2 that NDT RC produces noticeable improvements over D2D NDT on sequences 0, 1, 2, 3, 5, 8 and 10. NDT RC provides visually accurate odometry for all sequences except the second one. The second sequence contains a lot of large changes in direction, which is challenging for the NDT registration since as it was shown to be sensitive to the quality of the initial guess [25].

TABLE V
RESULTS ON THE KITTI DATASET

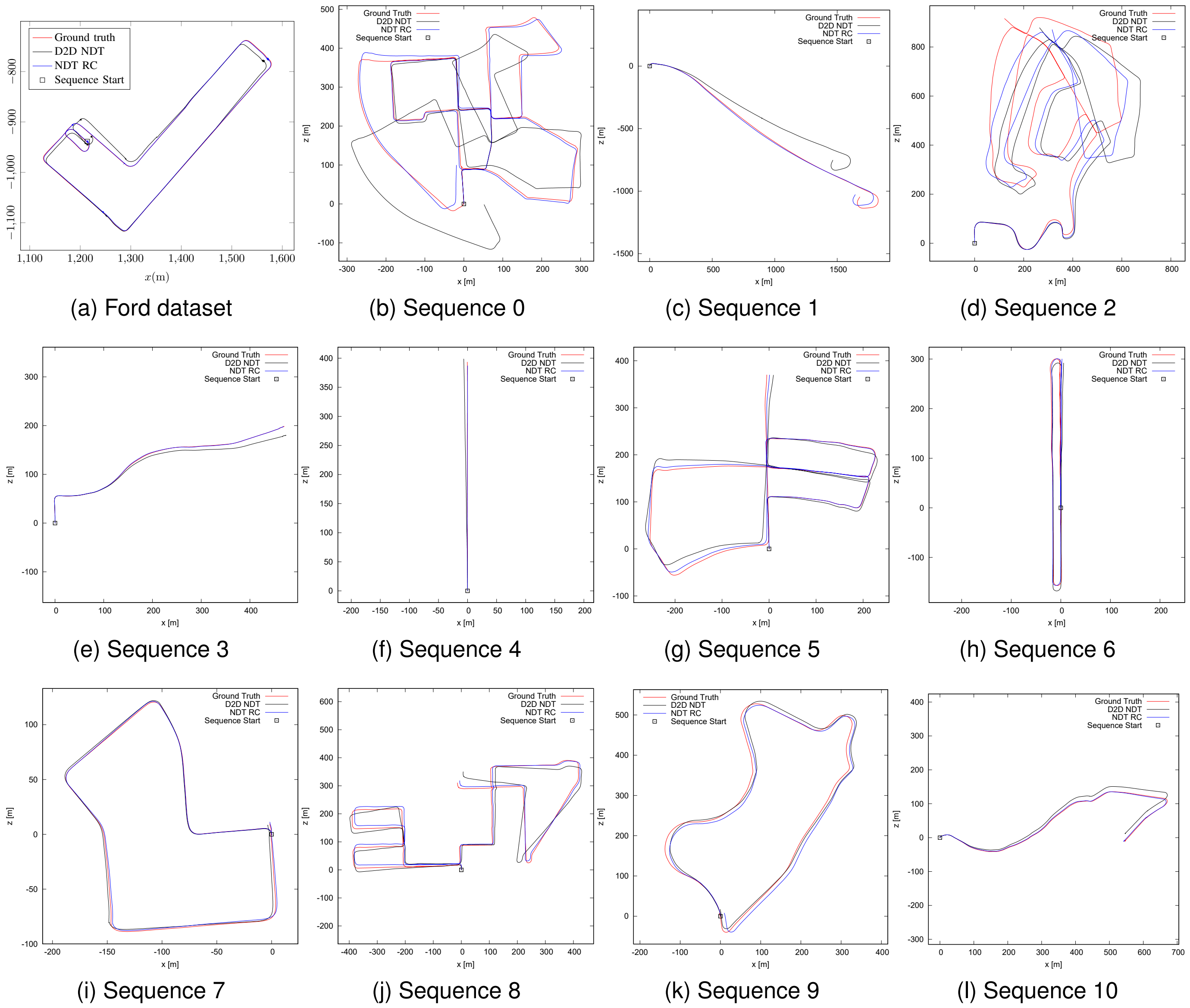
Method	e_t (%)	e_r ($\mathrm{mrad\,m^{-1}}$)	Runtime (ms)
NDT RC	1.54	**0.106**	**187**
SE-NDT [11]	2.60	0.2	See comments
D2D NDT [25]	3.65	0.208	499
LOAM [20]	**0.88**	-	<1000

It is important to note that both D2D NDT and SE-NDT are purely odometry methods: they do not maintain a consistent map between each scan, while the NDT RC method does. It is nonetheless fair to compare ourself to those methods since they are applicable to the same scenarios. Indeed, the recentring algorithm makes it possible to move the map with the robot. NDT OM for instance, could not be applied directly to the KITTI dataset because the map would not fit into memory.

The current top performing laser based method on the KITTI odometry dataset is the LOAM method [20], which achieves $0.88\,\%$ of translation error. The drawback is the runtime, since to achieve those precise results, the slow mapping step was run with each new scan. We can also note that this method uses 2 CPU threads, while NDT RC is entirely monothreaded.

V. SENSITIVITY ANALYSIS

One weakness of the NDT OM algorithm [2], and by extension, of NDT RC, is the numerous tuning parameters involved, as seen in Section II. It might then prove challenging to tune the algorithm for the best results: the precision of the NDT OM Fusion was evaluated with regard to the sensor cutoff distance [12], or the occupancy threshold [24], but to the best knowledge of the authors, no evaluation regarding the influence of the other parameters on the precision of the odometry exists. To address this issue, an evaluation of the precision of the odometry and runtime of the NDT RC is proposed in this section with regards to the five main parameters: the values of α, β, γ (see (8)), which govern the occupancy update during raytracing, the clamping threshold of occupancy per cell, K_{occ} and the limit number of points M_{max} inside a cell. The influence of the size of the map



(a) Ford dataset (b) Sequence 0 (c) Sequence 1 (d) Sequence 2 (e) Sequence 3 (f) Sequence 4 (g) Sequence 5 (h) Sequence 6 (i) Sequence 7 (j) Sequence 8 (k) Sequence 9 (l) Sequence 10

Fig. 2. Trajectory obtained by NDT RC (in blue) compared to the ground truth (in red) on the KITTI dataset and on the Ford dataset (Fig. 2a). Trajectories obtained by NDT RC (in blue) and D2D NDT (in black) compared to the ground truth (in red) on the KITTI dataset (Figs. 2b to 2l)

kept in memory is examined as well. Those parameters are all evaluated for different cell sizes. The Ford dataset [21] is used. Then, the the KITTI dataset [18] is used to evaluate the influence of the LIDAR point cloud density on the precision and runtime of NDT RC. In both cases, the testing conditions are described in Section IV.

Note that the influence of d_{thrc} is not examined since this parameters only affects how often the map is recentered. If set accordingly with the speed of the vehicle and the size of the map, it should have no influence on the precision of the algorithm and as shown in Table III, a neglectible influence on the runtime.

Fig. 3 shows the variation of the ATE and runtime with the value of α and β. α is an indication of the confidence that the cell is empty when a ray goes through it, while β quantifies the probability that a cell is occupied when a point is detected inside it. To keep the figure readable, errors of more that $9\,\mathrm{m}$ are omitted from the graph[2]. Those outliers are indicative of tracking failure, which happens more often at low or high cell sizes. In order for the matching to work reliably, the cell size should be large enough to capture significant features of the environment. However, if the cells are too large, the Gaussian approximation for an obstacle inside the cell might not hold anymore, resulting in matching errors. It is thus expected for the error to diminish up to the optimal cell size, then increase when the cell size is too large to represent the environment accurately. The results show that increasing the cell size results in a fast reduction of the error up to around $1.4\,\mathrm{m}$, with the error starting to increase at a cell size around $2.4\,\mathrm{m}$. The mean ATE is the lowest around a resolution of $2.2\,\mathrm{m}$. A good combination of stability and performance seems to be a couple

[2]Errors occurred at $(\text{Resolution}, \alpha, \beta)$: $(0.6, 0.39, 0.8)$, $(0.6, 0.45, 0.9)$, $(0.8, 0.39, 0.7)$, $(1.2, 0.39, 0.7)$, $(1.6, 0.45, 0.8)$, $(2.2, 0.39, 0.7)$, $(2.6, 0.39, 0.7)$, $(2.8, 0.45, 0.8)$, $(3, 0.45, 0.7)$, $(3, 0.39, 0.8)$, $(3, 0.45, 0.8)$, $(3.2, 0.39, 0.8)$, $(3.4, 0.39, 0.9)$, $(3.2, 0.45, 0.9)$

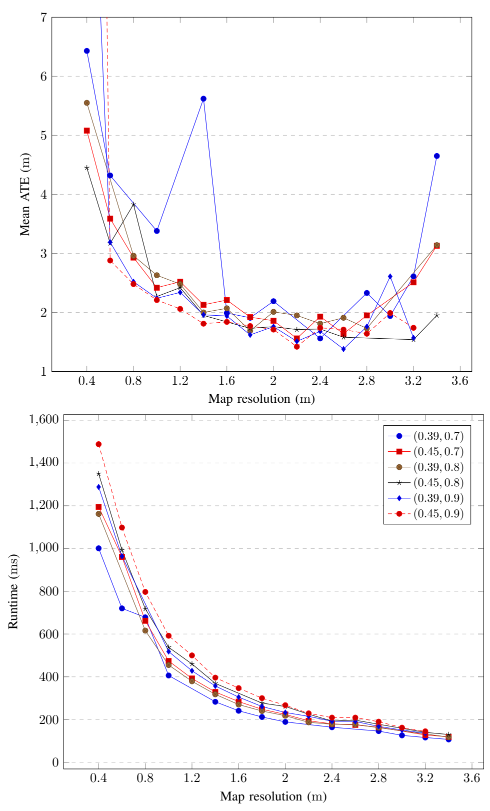


Fig. 3. Evolution of the mean ATE and runtime depending on parameters α and β. The legend indicates a pair (α, β)

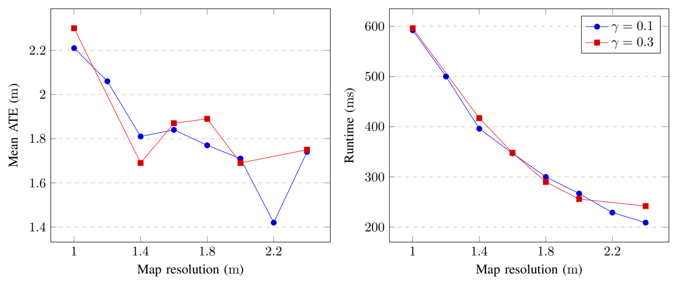


Fig. 4. Evolution of the mean ATE and runtime depending on the γ parameter

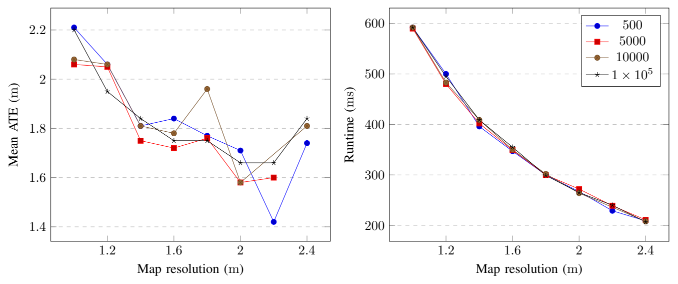


Fig. 5. Evolution of the mean ATE and runtime depending on parameter M_{max}. The legend shows different values of M_{max}

(α, β) of $(0.45, 0.9)$, which gives a mean ATE of $1.42\,\mathrm{m}$ for a resolution of $2.2\,\mathrm{m}$. The runtime is shown to be a function of the map resolution, although the values of α and β have an influence as well. The lower α is, the more punitive the raytracing is. Thus a lower α will tend to suppress more cells. On the other hand, a higher β will tend to create more cells. It is reasonable to consider that the runtime is a function of the number of active cells. Thus, theoretically, a low α and low β should have the lowest runtime, while a high α and high β should have the highest. This is precisely what is observed in Fig. 3.

Fig. 4 illustrates the influence of the parameter γ, which governs how punitive the raytracing is when a ray passes through a cell that is occupied by a Gaussian. There is no visible trend between between the values 0.1 and 0.3 for γ, except that a lower value of γ seems more stable: when $\gamma = 0.3$, the algorithm failed at resolution $1.2\,\mathrm{m}$ and $2.2\,\mathrm{m}$. A higher value of γ means that cells hosting Gaussians are penalised by a larger loss of occupancy when traversed by a ray. Thus in theory, a lower runtime for higher values of γ is expected since the number of active cells should be reduced. This is however not the case, which indicates that the difference in number of invalidated cells hosting Gaussians is low between the two values of this parameter.

Next, the influence of the parameter M_{max} is examined, which governs the sliding average when points are added into a cell: a higher M_{max} indicates a map slower to adapt to dynamic changes. The result is shown in Fig. 5. The difference made by M_{max} is shown to be resolution dependent, with small values of M_{max} preferable at higher cell sizes. It can be hypothesized that smaller values of M_{max} would perform better in a dynamic environment such as the Ford dataset, however this is not observed at lower cell sizes. This parameter should have no influence on the runtime, which is demonstrated in this figure.

The clamping occupancy threshold K_{occ} governs how fast cells are created or eliminated in case of dynamic changes. The influence of this parameters is shown in Fig. 6. The influence of the error is resolution dependent, with no clear trend. The higher value tends to show more stability across changes in resolution. It is difficult to forecast the influence of K_{occ} on the runtime since decreasing this parameter means that cells are both created and deleted more often. Given the results, it looks like this effect balances itself since no significant changes are observed for different values of K_{occ}.

Next, the influence of the size of the map on the mean ATE is investigated. The LIDAR used has a range of $120\,\mathrm{m}$, so the

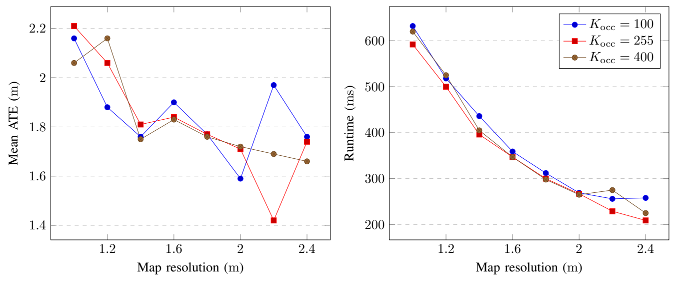


Fig. 6. Evolution of the mean ATE and runtime depending on parameter K_{occ}

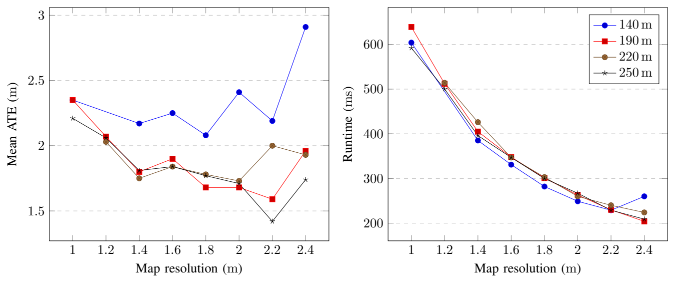


Fig. 7. Evolution of the mean ATE and runtime depending on the size of the map. The legend shows different map sizes (the size is considered as width and depth)

TABLE VI
INFLUENCE ON THE POINT CLOUD DENSITY ON NDT RC USING THE KITTI DATASET

Method	e_t (%)	e_r (mrad m^{-1})	Runtime (ms)
Full cloud	1.54	0.106	187
Partial cloud	15.2	0.897	84

default size, 250 m, should be large enough to host the whole scan. Fig. 7 examines the loss of precision that occurs when the whole LIDAR scan cannot fit into the map anymore. Note that the method fails for a map size of 140 m at the resolution 1.2 m and for the map size of 220 m at the resolution 1.0 m. Those outliers are not including in the graph to keep it readable. A map size of 140 m leads to a larger error compared to bigger sizes. The difference between 190 m, 220 m and 250 m depends on the resolution, with the lowest error achieved by the 250 m map which performs slightly better as the cell size increases. The runtime is barely affected by the size of the map in this dataset, meaning that the number of occupied cells at the edge of the map is small since their disappearance does not lead to a significant reduction in runtime.

Finally, it is interesting to look at the influence of the density of the input point cloud on the quality of the results. The KITTI dataset [18] is used to perform this investigation. It is important to note that the Velodyne HDL-64E LIDAR used in the KITTI datasets is adapted to automotive applications, but not to small UAVs or robots: its weight is around 13 kg and it consumes approximately 50 W of power. A more realistic sensor for small UAVs would be the Velodyne VLP-16 lite [31] which weighs 590 g and draws around 8 W of power. The drawback is that the VLP-16 has only 16 vertical beams, compared to the 64 of the HDL-64E, with a vertical resolution going from 0.4° down to 2°. In order to simulate a VLP-16 output from the HDL-64E, a downsampling method is implemented. Since the VLP-16 has a vertical resolution of 2° for a total vertical field of view of 30°, it is not possible to match exactly the vertical field of view of the HDL-64E, which is 25.2°. It was chosen to respect the vertical resolution of the VLP-16, thus the downsampled cloud has only 13 beams, meaning that the point cloud obtained from an actual VLP-16 would be denser compared to the results presented in this section. The downsampling result is illustrated in Fig. 8. Table VI shows the difference in precision and runtime between the full and downsampled clouds. It can be seen

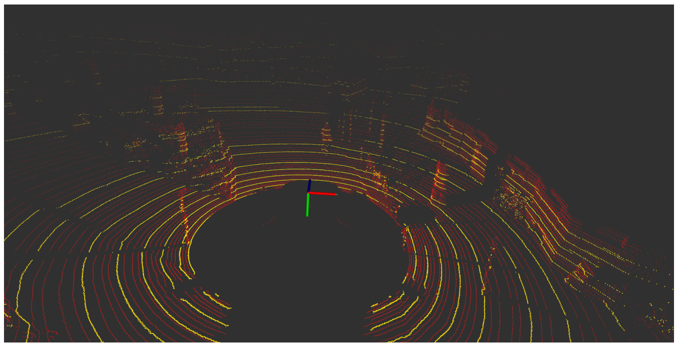


Fig. 8. LIDAR scan acquired from a Velodyne HDL-64E (in red) and a downsampled version matching the characteristics of the Velodyne VLP-16 (in yellow)

that reducing the point density greatly decreases precision, it can be hypothesized that the lower number of Gaussians is not enough to effectively perform matching. The fact that the runtime is only multiplied by 2.2 when the number of points is multiplied by 4 highlights a strength of the algorithm: as shown in Table III, the runtime is dominated by matching, thus it does not scale with the number of points, but with the number of created gaussians..

VI. Conclusion

In this paper, NDT RC, a recentering algorithm applied to the NDT OM framework is proposed. This algorithm introduces a novel recentering mechanism that allows unlimited movement of the robot, therefore enabling usage on larger datasets. A test of NDT RC on two publicly available datasets shows that the recentering algorithm allows to reduce both the registration error and the runtime when compared to other NDT based methods. NDT RC achieves an RMS ATE of 1.49 m on the Ford dataset and a translation error of 1.54 % on the KITTI dataset, with a runtime below 250 ms. A sensitivity analysis is carried out to show the influence of parameters and point cloud density over the algorithm, both in terms of precision and runtime. This analysis shows how the runtime is mainly influenced by the map resolution, and allow a unique set of parameters to be set for both the Ford and KITTI dataset.

For future works, combining the NDT RC with methods improving the registration algorithm such as SE-NDT should yield superior results. Testing could also be extended to larger datasets as well as datasets using UAVs to explore performances with 6 degrees of freedom. Investigation of the application of loop closure methods to NDT RC could provide accuracy benefits by reducing drift.

References

[1] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An efficient probabilistic 3D mapping framework based on octrees,” *Autonomous Robots*, 2013.

[2] J. P. Saarinen, H. Andreasson, T. Stoyanov, and A. J. Lilienthal, “3d normal distributions transform occupancy maps: An efficient representation for mapping in dynamic environments,” *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1627–1644, Sep. 2013.

[3] A. Elfes, “Occupancy grids: A probabilistic framework for robot perception and navigation,” AAI9006205, PhD thesis, Pittsburgh, PA, USA, 1989.
[4] H. P. Moravec, “Sensor fusion in certainty grids for mobile robots,” *AI Magazine*, vol. 9, pp. 61–74, Jun. 1988.
[5] M. Herbert, C. Caillas, E. Krotkov, I. S. Kweon, and T. Kanade, “Terrain mapping for a roving planetary explorer,” in *Proceedings, 1989 International Conference on Robotics and Automation*, May 1989, 997–1002 vol.2.
[6] P. Biber and W. Strasser, “The normal distributions transform: A new approach to laser scan matching,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 3, Oct. 2003, 2743–2748 vol.3.
[7] M. Magnusson, A. Lilienthal, and T. Duckett, “Scan registration for autonomous mining vehicles using 3d-NDT,” *Journal of Field Robotics*, vol. 24, no. 10, pp. 803–827,
[8] T. Stoyanov, M. Magnusson, and A. J. Lilienthal, “Comparative evaluation of the consistency of three-dimensional spatial representations used in autonomous robot navigation,” *Journal of Field Robotics*, vol. 30, no. 2, pp. 216–236,
[9] T. Stoyanov, M. Magnusson, and A. J. Lilienthal, “Point set registration through minimization of the l2 distance between 3d-NDT models,” in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 5196–5201.
[10] A. Das and S. L. Waslander, “Scan registration using segmented region growing NDT,” *The International Journal of Robotics Research*, vol. 33, no. 13, pp. 1645–1663, 2014.
[11] A. Zaganidis, M. Magnusson, T. Duckett, and G. Cielniak, “Semantic-assisted 3d normal distributions transform for scan registration in environments with limited structure,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vancouver, BC, Canada, September 24-28*, 2017, pp. 4064–4069.
[12] T. Stoyanov, J. Saarinen, H. Andreasson, and A. J. Lilienthal, “Normal distributions transform occupancy map fusion: Simultaneous mapping and tracking in large scale dynamic environments,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov. 2013, pp. 4702–4708.
[13] E. Einhorn and H.-M. Gross, “Generic ndt mapping in dynamic environments and its application for lifelong slam,” *Robot. Auton. Syst.*, vol. 69, no. C, pp. 28–39, Jul. 2015, ISSN: 0921-8890.
[14] A. Zaganidis, A. Zerntev, T. Duckett, and G. Cielniak, “Semantically assisted loop closure in slam using ndt histograms,” in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 4562–4568.
[15] H. Courtois, N. Aouf, K. Ahiska, and M. Cecotti, “Oast: Obstacle avoidance system for teleoperation of uavs,” Submitted to IEEE Transactions on Human-Machine Systems (THMS).
[16] D. Droeschel, M. Nieuwenhuisen, M. Beul, D. Holz, J. Stückler, and S. Behnke, “Multilayered mapping and navigation for autonomous micro aerial vehicles,” *Journal of Field Robotics*, 2016.
[17] J. Behley and C. Stachniss, “Efficient surfel-based slam using 3d laser range data in urban environments,” Jun. 2018.
[18] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
[19] J. Zhang and S. Singh, “LOAM: lidar odometry and mapping in real-time,” in *Robotics: Science and Systems X, University of California, Berkeley, USA, July 12-16, 2014*, D. Fox, L. E. Kavraki, and H. Kurniawati, Eds., 2014.
[20] ——, “Low-drift and real-time lidar odometry and mapping,” *Autonomous Robots*, vol. 41, pp. 401–416, Feb. 2017.
[21] G. Pandey, J. R. Mcbride, and R. M. Eustice, “Ford campus vision and lidar data set,” *Int. J. Rob. Res.*, vol. 30, no. 13, pp. 1543–1552, Nov. 2011, ISSN: 0278-3649.
[22] H. Courtois, N. Aouf, K. Ahiska, and M. Cecotti, “Oast: Obstacle avoidance system for teleoperation of uavs,” *IEEE Transactions on Human-Machine Systems*, vol. 52, no. 2, pp. 157–168, 2022.
[23] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005, ISBN: 0262201623.
[24] J. Saarinen, T. Stoyanov, H. Andreasson, and A. J. Lilienthal, “Fast 3d mapping in highly dynamic environments using normal distributions transform occupancy maps,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7*, 2013, pp. 4694–4701.
[25] T. Stoyanov, M. Magnusson, H. Andreasson, and A. J. Lilienthal, “Fast and accurate scan registration through minimization of the distance between compact 3d NDT representations,” *The International Journal of Robotics Research*, vol. 31, no. 12, pp. 1377–1393, 2012.
[26] H. Courtois, “Obstacle avoidance for unmanned aerial vehicles during teleoperation,” PhD thesis, School of Defence and Security, 2020.
[27] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of rgb-d slam systems,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct. 2012, pp. 573–580.
[28] K. Lenac, A. Kitanov, R. Cupec, and I. Petrovic, “Fast planar surface 3d SLAM using LIDAR,” *Robotics and Autonomous Systems*, vol. 92, pp. 197–220, 2017.
[29] A. Geiger, M. Roser, and R. Urtasun, “Efficient large-scale stereo matching,” in *Asian Conference on Computer Vision (ACCV)*, 2010.
[30] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, “On measuring the accuracy of slam algorithms,” *Auton. Robots*, vol. 27, no. 4, pp. 387–407, Nov. 2009.
[31] *Velodyne vlp-16 lite*, http://velodynelidar.com/vlp-16-lite.html, Accessed: 2017-05-05.

Hugo Courtois has received his Ph.D. degree from the Centre for Electronic Warfare information and Cyber in Cranfield University, U.K. in 2019. He is currently working at Outsight on spatial intelligence using lidars. His research interests include mapping, sensor fusion, haptic and lidar technologies.

Nabil Aouf is currently the lead of the Robotics and Machine Intelligence activities at City University of London, U.K. He leads the Robotics, Autonomy and Machine Intelligence (RAMI) group. He has authored more than 180 publications in high caliber in his domains of interest. His research interests include aerospace, information fusion and vision systems,guidance and navigation, tracking, and control and autonomy of systems. Prof. Aouf is an Associate Editor of four journals including an IEEE Transaction Journal.

Kenan Ahiska received B.S., M.S., and Ph.D. degrees in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 2010, 2012, and 2016, respectively. He has background on control theory and its applications. He has works on modeling, guidance control and navigation of unmanned vehicles, optimal and model predictive control of robotic systems. Since 2018, he has been a research fellow in guidance and control in Cranfield University, Defence and Security at Shrivenham, the U. K.

Marco Cecotti received the PhD degree in electrical engineering from Oxford Brookes University, UK, in 2013. He worked for Tata Motors and Dyson on several automotive projects, focused on vehicle control and driver assistance systems. He is now a Lecturer with the School of Aerospace, Transport and Manufacturing at Cranfield University, UK. His research interests include vehicle trajectory control, path planning, localisation and sensor fusion. He is a member of the IEEE.

NDT RC: Normal Distribution Transform Occupancy 3D Mapping with recentering

Courtois, Hugo