Real-time moving object detection and removal from 3D pointcloud data for humanoid navigation in dense GPS-denied environments

Prabin Kumar Rath¹ | Alejandro Ramirez-Serrano² | Dilip Kumar Pratihar³

¹Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India
²Department of Mechanical and Manufacturing Engineering, University of Calgary, Calgary, Alberta, Canada
³Department of Mechanical Engineering, Indian Institute of Technology - Kharagpur, West Bengal, India

Correspondence
Alejandro Ramirez-Serrano, University of Calgary, Calgary, AB, Canada.
Email: aramirez@ucalgary.ca

Funding information
MITACSGlobalink2019 program; Natural Sciences and Engineering Research Council of Canada (NSERC)

Abstract
Robot perception in dynamic confined unstructured environments is a challenging task due to unanticipated changes that take place in the surroundings. Although 3D perception sensors are able to capture terrain topology with high precision, the interim variations between collected sensor data that are caused due to the motion of moving entities with respect to the robot lead to noisy mappings of the environment. In this article, a real-time 3D perception filter is presented that is capable of detecting and eliminating moving point clusters from the input pointcloud data collected in an indoor environment. Using LiDAR and IMU sensors the proposed mechanism can help in precise 3D pointcloud map generation in dynamic and unstructured GPS-denied environments. In this article, a novel approach has been proposed based on the concepts of data clustering, relative motion, pointcloud change detection and confidence tracking. The novelty of this approach lies in its ability to detect within cluster movements and the proposal of a generic tracking method for handling inconsistent motion of objects typically found in indoor environments. For the detection of moving objects, the proposed mechanism does not require any prior knowledge about the target entity. For pointcloud preprocessing, a ground plane removal approach has been proposed based on voxel grid covariance along the axis normal to the ground. The approach was experimented on a humanoid robot in indoor office environments using Velodyne VLP-16 LiDAR and Intel T265 IMU. The results show that the proposed approach is efficient in detecting indoor moving objects in real time.

KEYWORDS
confidence tracking, DATMO, euclidean clustering, pose transformation, relative motion, SLAM, voxel grid covariance

[Correction added on 21 October 2020, after first online publication: In Funding information, 'Natural Sciences and Engineering Research Council of Canada (NSERC)' has been added to this version.]
1 | INTRODUCTION

Perception in robotics, the task of getting acquainted with the environment using diverse sensors such as day/night cameras, 2D range finders and 3D light detection and ranging (LiDAR)s, is paramount to intelligent robotic functionalities. With the advancement in sensor technology seen in the last decade, there have been significant contributions to the field of robot perception that has led to the development of state of the art simultaneous localization and mapping (SLAM) algorithms. These algorithms have been developed for accurate mapping of static or quasi-static environments where position of objects around the sensor does not change with respect to the inertial frame (often referred as the world frame or fixed frame). Errors in the sensor’s motion estimation and measurement greatly affect the quality of mapping process. In dynamic environments these estimation errors are increased due to the motion of objects around the sensor, and have an even greater effect on the generated map. In this article, the term “dynamic” is referred to frequent changes in the surroundings that occur constantly due to a number of independent moving entities (eg, people, vendors and other mobile robots). Due to these characteristics, 3D mapping is typically done for evacuated facilities such that the robot is only required to perceive static components of the environment.

However, in real-world indoor scenarios such as offices, shopping malls and sporting events the environment is dynamic in nature. When using typical SLAM algorithms, the presence of moving objects particularly people (ie, walking, running, jumping, etc) leads to inaccurate map generation. Furthermore, in cluttered and dynamic spaces if moving objects are not properly identified there is a risk to have a potentially large accumulation of undesirable artifacts in the generated map. A reason for this is that conventional algorithms such as iterative closest point (ICP) fails to accurately register the incoming pointcloud. Under such conditions accurate pose estimation of the robot (and its sensors) becomes problematic. In some cases, accuracy of SLAM can be improved when motion of the robot or identification of features in the environment meets certain conditions such as loop closure which is also difficult to detect in a dynamic environment. Hence, dynamic/moving objects need to be detected and effectively handled in real time to generate maps that are as good as the ones generated in a purely static environment. This is crucial for mobile robots targeted to operate in unknown dynamic real-world (potentially unstructured) GPS-denied environments which require to perform real-time (ongoing) mapping and planning.

In the field of robot perception, the problem of detecting and tracking moving objects from dynamic pointcloud scenes has been researched earlier and is known as detection and tracking of moving objects (DATMO). In the pioneering work on simultaneous localization, mapping and moving object tracking (SLAMMOT), Wang et al define the eight requirements of a DATMO algorithm as:

- Detect and initiate new dynamic objects
- Model dynamic object trajectories
- Perform data association (ie, find data correspondence)
- Combine two or more dynamic objects that correlate with each other
- Omit dynamic objects that are no longer in the sensor’s field of view
- Account for objects that may be occluded
- Attempt to rectify incorrect measurements
- Operate robustly over long observation sequences

The algorithm proposed in Reference 7 is an attempt to develop a DATMO-integrated SLAM model for pointclouds captured in dynamic outdoor environments. In the work presented in this article a generic DATMO pipeline has been proposed specifically for GPS-denied indoor environments. It can be used with various autonomous systems (eg, rovers), including humanoids performing complex motion maneuvers (eg, walking, climbing, jumping, running). Humanoids with their typical agility have the potential to navigate/map in such environments with ease. While navigating and mapping such environments they require frequent/fast knowledge about the spatial locations of nearby moving entities to get aware of the changes taking place in their surroundings. The proposed algorithm has the aim to be used as an intermediary process in SLAM and for navigation of humanoids in dynamic cluttered spaces where objects of various geometries move in diverse (chaotic) ways at varying speeds.

In the proposed research (Figure 1), first the LiDAR sensor data are segregated into moving and static point clusters. Then the motion states of independent entities present in the robot's surroundings are determined. For this, the incoming
real-time pointcloud and IMU odometry data are time-synchronized to determine the pose (position and orientation) of the LiDAR at the time of capturing the corresponding pointcloud data frame. Subsequently, a transformation matrix between two consecutive pointclouds (LiDAR pose) is used to transform the pose of the pointcloud data taken at time \( t - 1 \) to that of the pointcloud data taken at time \( t \). The transformation takes place in LiDAR’s frame and provides an overlapping pointcloud data, which are later analyzed to identify the point clusters representing moving objects around the robot. The identified clusters (static and dynamic) are tracked for a given number of frames to generate a confidence for each point cluster in the frame. All clusters within the pointcloud frame satisfying a confidence constraint, are then classified as moving objects/entities. The proposed approach resembles the idea of a classification problem based on historical observations taken over a very small finite time-horizon, similar to the approach used by humans’ cognition abilities. Figure 1 provides a schematic flowchart of the proposed DATMO pipeline.

2 | RELATED WORK

State-of-the-art in SLAM algorithms as discussed in References 1, 2 are effective. However, work in SLAM has mainly focused on static or sparsely dynamic environments that do not change rapidly such as caves and empty office spaces or large open spaces where very few objects move at any given time (e.g., warehouse). A few studies consider the mapping of dynamic environments.

To solve the problems of detection of moving objects during SLAM, Wang et al.\(^7\) provided the first mathematical model of SLAM integrated with DATMO (SLAMMOT). The authors used a Bayesian formulation using extended Kalman filter (EKF) to estimate posteriors, iterative closest point (ICP) to represent data in grid-maps, multiple hypothesis tracking (MHT) for finding data association and an interacting multiple model (IMM) algorithm to track and model moving objects. Following the work of Wang, Vu et al.\(^9\) used an incremental scan matching method instead of ICP to determine vehicle odometry in dynamic outdoor environments.

Hahnel et al.\(^4\) used a probabilistic method (i.e., sample-based joint probabilistic data association filter, SJPDFs) to track multiple people and incorporated the estimates of the tracking technique into the mapping process. They showed that the resultant map contained less spurious objects. In contrast to Reference 4 Wolf et al.\(^10\) used two occupancy grids, one to model the static parts of the environment, and the other to model the dynamic parts. They also used a third map that contained information about static landmarks. An EKF was used to incrementally estimate the correct position of the robot and landmarks (i.e., corners) in the environment. Their work also shows that localization based on corner detection can be used to improve the performance of SLAM algorithms.

Mertz et al.\(^11\) proposed a 2D scan matching method for detection of moving objects using a 2D projection of the 3D scanned data. Such method segments out different point clusters from the 2D scanline to extract line or corner features out of it for data association. For tracking of the moving objects a Kalman filter is used. However, use of elevation maps for projecting 3D sensor data into 2D leads to loss of essential information about the objects perceived by the sensor. Luo et al.\(^12\) use a radially bounded nearest neighbor (RBNN) approach to cluster distinct objects in the pointcloud. They propose a covariance-based Kalman filter to estimate correct centroid of moving objects for tracking and use global nearest neighbor (GNN) for data association. In their work they show that global cluster centroids can be reliably used as a feature for tracking.

In contrast to Reference 11 Asvadi et al.\(^13\) proposed the use of 2.5D motion grids in which each cell stores the height of objects above the ground level. A threshold based segmentation on average height and height variance of the grid cells is used to remove the ground plane from the grid. Moving objects are detected by the use of a local map update algorithm using historical pose set of the sensor. However, use of multiple historical pose is prone to localization errors which the
authors have tried to suppress using the 2.5D motion grid representation of data. They used a nearest neighbor approach for data association and Kalman filters with constant velocity model for tracking moving objects around the robot. In a proceeding work by Asvadi et al., a 2D/3D fusion-based moving object tracking was used.

Azim and Ayardi propose the use of octrees for detection and classification of moving objects, due to their efficiency and low memory consumption. They monitor the state of leaf node voxels between consecutive pointcloud frames to determine potential moving voxels which are later clustered to determine the points corresponding to a moving object. The objects are classified based on the features of their bounding box and associated using the GNN approach. Similar to other approaches, tracking is performed using a Kalman filter.

Takabe et al. proposed a moving object detection method with the use of cameras by minimizing an energy function based on photometric and depth consistencies. Although such approach is good for indoor spaces it presents challenges in outdoor environments due to brightness issues with camera capturing devices. The approach also fails to work in dark spaces. Dewan et al. proposed a model-free approach for detection in outdoor environments based on motion cues of the objects. They use the popular random sample consensus (RANSAC) algorithm to determine motion models for the moving objects. However, they also admit that their approach cannot be used for pedestrian tracking. A reason for this, as mentioned in their work is slow and inconsistent motion behavior of people walking around the sensor.

Based on the above developments, moving object detection has shown limitations for GPS-denied dense environments where the sensitivity of the detection algorithm needs to be high, robust and accurate for perceiving low-speed moving objects. These limitations arise from the fact that when objects move with a high velocity (e.g., bikes, cars, buses, motorcycles in outdoor environments) the spatial change between consecutive pointcloud frames (those sampled at high frequency) is distinct and easy to detect. In an indoor environment the moving object’s relative velocity is significantly low (e.g., walking, running, jumping) and hence with a high sampling frequency it is difficult to determine the pointcloud changes between consecutively captured frames.

Litomisky and Bhanu presented a method for detecting moving objects in indoor environments using RGB-D sensors and spatial relationships between point clusters sampled from two consecutive pointcloud frames. For data association, they use viewpoint feature histogram (VFH) descriptors of the moving clusters and match them using dynamic time wrapping (DTW). Although this approach is well formulated and sound, the change in viewpoint over time is not significant when the data sampling frequency is high, hence it does not provide accurate results when objects move at relatively slow speeds (e.g., detecting a toddler crawling around the robot). The use of VFH for cluster correspondence, however, works well for dense pointclouds but fails to provide satisfactory results for sparse pointclouds. Thus, the approach proposed in Reference is suitable with dense pointclouds (short range) coming from sensors such as the Kinect and Intel Realsense but fails for sparse (i.e., long-range sensor data) pointclouds captured with high-end LiDAR sensors such as Velodyne and Riegl LiDARs.

Previously developed algorithms do not consider a cluster-based approach for detecting objects with both static and dynamic aspects simultaneously (i.e., within cluster movements). In such cases typical algorithms may lead to classifying an object having multiple (potentially independent) moving parts as both moving and static at the same time. For example, let a person wave his/her hands while standing at a fixed position. In such a situation, not only the hand but the entire point cluster corresponding to the person should be detected as moving. Examples of such cases may also arise when sensing humanoid robots (or people) moving their arms or head but not their legs. In such cases, the humanoid might be considered as both a static and a moving object at different times which may lead to ambiguity. Thus, partial detection of moving parts within a stationary object in the environment may be problematic as a given robotic artifact may plan its trajectories based on considering an object to be static when in reality might move and affect the robot’s actions as well as the map generation performance.

Previous works have used different varieties of Kalman filters for tracking moving objects. However, moving objects found in an indoor environment have inconsistent motion cues. Hence, it is difficult to predict the motion of the entities with a conventional Kalman filter. In the work proposed in this article it has been assumed that reliable odometry data are available to get the motion states of the LiDAR with respect to the inertial frame. The main contributions of this article are:

1. An approach for moving object detection based on relative motion of objects with respect to sensor and mathematical formulation of associated intuition.
2. A novel confidence tracking approach for track maintenance to handle inconsistencies in the motion of moving objects in indoor environments.
3. A new ground plane removal method independent of translational variations, that can be used with a gimbal stabilized humanoid head.

The article ahead is organized as follows. In Section 3, the proposed DATMO pipeline is presented in detail. In Section 4, the hardware and software setup used for testing the proposed work are described. Section 5 shows the obtained experimental results on a sample pointcloud data set collected in an indoor office environment along with experimental validation on a publicly available data set.

3 | PROPOSED APPROACH

In the proposed work the focus is on DATMO as a separate block rather than an integrated version with SLAM. The proposed approach detects and removes moving objects from the incoming pointcloud frames in real-time which can later be used by any available SLAM algorithm. The principal idea is to observe relative motion between sensor (LiDAR) and the objects within its surroundings. Herein, the words “sensor” and “LiDAR” have been used alternatively throughout the text and mean the same.

The proposed work is concerned with cluttered and unstructured indoor environments. Long-range data from the LiDAR which usually has a lot of outliers are superfluous for detection of nearby moving objects. Hence, the raw cloud provided by the LiDAR is cropped to a $6 \times 6 \text{ m}^2$ around the sensor which can still be considered as a large range for indoor cluttered environments. With the cropped pointcloud considered as input, an eight-step process has been proposed for detecting and removing moving entities present in the sensed environment.

3.1 | Cloud and pose synchronization

Each sensor data sample has its own timestamp holding the time at which the corresponding data were captured. At any instant of time, sensor’s odometry data are used to obtain pose of the sensor with respect to the inertial frame. As the odometry data and the pointcloud data are sampled independently, it becomes essential to synchronize them with respect to their time of capture. A synchronized pointcloud-pose pair provides the information about pose of the sensor while capturing the corresponding pointcloud. In this work, ROS approximate time synchronization policy\textsuperscript{21} has been used for generating pointcloud-pose pairs from the two data streams (pointcloud and odometry). The method provides an average maximum time difference of 3.14 milliseconds between the timestamps of synchronized data.

3.2 | Ground plane removal

Figure 2 shows an example of a typical pointcloud captured in a cluttered indoor office environment where the position of LiDAR is shown as a black circular dot. The captured data are shown after the $6 \times 6 \text{ m}^2$ cropping around the sensor. Different objects within the pointcloud are numbered for better understandability where the actual objects are 1-closet, 2-bicycle, 3-whiteboard, 4-person, 5-table and 6-chair, respectively. As it can be observed from the figure, points corresponding to the ground become connecting links between individual distinct objects observed in the pointcloud. Due to this, clustering the points into distinct groups of point clusters (ie, each cluster corresponding to a specific object around the robot) becomes increasingly difficult. In the presence of the ground plane, euclidean clustering, for example, cannot be used, as multiple independent items would be typically grouped as a single cluster due to their connectivity through the ground plane points. The color of a given point in Figure 2 represents the intensity of the reflected laser scan.

As the concerned environments are indoor, it has been assumed that the ground plane is typically flat and does not possess any significant elevations. Such elevations are found in pointclouds captured in outdoor environments. Furthermore, use of a gimbal stabilized humanoid head for data collection stabilizes the sensor against any rotational variations. These assumptions provide the flexibility to consider ground plane to be lying parallel to the xy plane with the normal of ground aligning with the $z$-axis of the sensor.
One of the conventional and well-accepted ground plane removal methods is the RANSAC\textsuperscript{22,23} algorithm. It is an iterative plane fitting procedure that randomly selects points from the pointcloud to fit a 2D plane according to a provided threshold. The fitting requires availability of adequate ground plane points within the pointcloud which is abundantly found for data captured in outdoor environments (e.g., roads, parking areas). However, for pointclouds captured in indoor cluttered environments, the availability of ground plane points is less and sparse. Due to this, conventional RANSAC algorithm fails in extracting the ground plane satisfactorily from such pointclouds.

To overcome the challenges associated with unavailability of adequate ground plane points, a modification of the ground plane removal method proposed by Douillard et al\textsuperscript{24} has been used. The ground plane is removed using a voxel grid covariance constraint as discussed in the following three steps:

1. The entire pointcloud is represented as a collection of aligned 3D boxes, also known as voxels. Voxels comprising the data set having more than three data points are selected for use in the next step.\textsuperscript{25} The size of voxels (selected based on the density of the pointcloud) is an important parameter that determines the time taken to complete the ground removal process.

2. For all selected voxels in Step 1, the spatial $z$-axis covariance of all the points within each voxel is computed. The $z$-axis covariance constitutes three values $\text{cov}(x,z)$, $\text{cov}(y,z)$ and $\text{cov}(z,z)$ computed using Equation (1) (shown only for one of the covariances where $n$ is the number of points within a voxel).

$$\text{cov}(x,z) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(z_i - \bar{z})}{n}. \tag{1}$$

It is intuitive to assume that the points corresponding to the ground plane will have very little variation along the $z$-axis. Since the $\text{cov}(x,z)$ and $\text{cov}(y,z)$ represent the variation of the points in the $xy$ plane (where the ground plane lies) with respect to the variation in $z$, the group of voxels encompassing the potential ground plane points will tend to have very small values of $\text{cov}(x,z)$ and $\text{cov}(y,z)$. Thus, all voxels comprising the flat surfaces are segmented (i.e., clustered) into a group of voxels known as ground voxel plane (GVP) using a threshold value $\delta$. Such voxel satisfy the condition shown in Equation (2). The resultant GVP has a collection of voxels that represent planes parallel to the ground including the ground plane.

$$\text{cov}(x,z) < \delta \quad \text{and} \quad \text{cov}(y,z) < \delta$$

$$\delta = \text{Covariance threshold}. \tag{2}$$
For an indoor environment, the clustered voxels may contain points corresponding to diverse plane surfaces parallel to the ground such as the surfaces of tables, stairs and other flat surfaces. The presence of a true ground plane in the GVP is detected by extracting the dominant plane among all the plane surfaces segmented in Step 2. The dominant plane is determined based on the point count within the voxels of the corresponding plane. The true ground plane within the GVP is herein indicated by a plane having the largest number of points. The dominant plane surface (herein considered as the true ground plane in the GVP) is identified by binning the points within all the clustered voxels with respect to their z-coordinate (elevation) values. The binning process divides the entire z-axis domain \((z_{\text{min}} \text{ to } z_{\text{max}})\) of the point cloud into a number of equal subdivisions (bins) \((\text{bins})\) \((\text{Equation (3))}\) and distributes the points among the bins according to their z-coordinate values \((\text{Equation (4))}\). Herein, the term bin gap \(G\) is utilized to denote the size of each bin. The value of the parameter \(G\) depends on the sensor used for sampling the point cloud. After the binning process is completed, the bin with the highest number of points is considered as the ground plane bin. These points are finally removed from the raw point cloud which completes the ground plane removal process.

\[
\text{No of bins} = \text{ceil}\left(\frac{z_{\text{max}} \text{ - } z_{\text{min}}}{G}\right),
\]

\[
\text{Range of } i\text{th bin } = [z_{\text{min}} + (i - 1) \times G, \quad z_{\text{min}} + i \times G].
\]

The binning of points in Step 3 can be represented as a histogram. Figure 3A shows the histogram generated by the algorithm with \(G = 10\) cm when applied on a point cloud captured (Figure 2) using Velodyne VLP-16 LiDAR. In Figure 3A the bin with range \([-0.5, -0.4)\) has the highest number of points (225 points). As expected, when the points within this bin are visualized it represents the ground plane shown in Figure 3B. Bins with range \([0.0, 0.1)\) and \([0.1, 0.2)\) also have a relatively large number of points (107 and 80, respectively) when compared with the other 24 remaining bins. The reason for this is that they contain points representing a tabletop surface within the point cloud. It must be noted that the set of points corresponding to the table’s surface are preserved in the point cloud. Hence, the proposed method is applicable to variable height flat and horizontal surfaces that is usually required when humanoid robots attempt to move up a staircase.

The results of ground plane removal (for point cloud shown in Figure 2) using conventional RANSAC and voxel grid covariance is shown in Figure 4A,B, respectively. It can be observed that the proposed approach is much more effective in removing ground plane compared with the conventional RANSAC method. For the experiment conducted, RANSAC distance threshold was chosen as 20 cm and voxel size for the proposed method was chosen as 10 cm. Run time of both the methods were analyzed and it was observed that the proposed method takes an average of 32 milliseconds per point cloud.
frame while RANSAC takes an average of 26 milliseconds. Run time for the proposed method can be greatly reduced with the use of a relatively larger voxel size. However, with larger voxels, removal accuracy is reduced as a few nonground plane points also get removed along with true ground plane.

### 3.3 Euclidean clustering

Once the set of points comprising the ground plane have been removed, the remaining points are clustered with respect to the euclidean distance between them. This process separates the pointcloud into clusters where each cluster denotes an independent object/entity (e.g., person, table, chair, animal, car) within the perceived environment. The clustering technique described in Reference 26 has been used for this process. The threshold for minimum distance between two points to be considered belonging to the same cluster and minimum number of points within a cluster are parameters that depend on the perception sensor used and application environment.

Figure 5 illustrates the point clusters obtained after applying euclidean clustering to the point cloud in Figure 4B using a minimum distance threshold value of 0.11 m where six entities (as mentioned in Section 3.2) have been identified within the given environment. After the clustering step is completed, all the detected clusters in the pointcloud frame are stored in a cluster vector \( \mathbf{V} \) as a collection of point clusters. A pointcloud frame can have multiple point clusters depending on how cluttered and dynamic the given environment is. The number of detected clusters in a pointcloud captured frame at time \( t \), that is, the number of clusters in \( \mathbf{V}_t \) is denoted with \( k \). Each cluster \( i \) within \( \mathbf{V}_t \) is denoted as a 3D vector \( \mathbf{PC}^i_t \) having \( n_{PC}^i \) number of points (Equation (5)). It must be noted that all the \( k \) point clusters detected in a pointcloud frame have the same timestamp \( t \) and associated sensor pose (discussed in Section 3.1). In Equation (5), subscript of PC denotes the time of capture while superscript denotes the index of the point cluster within \( \mathbf{V}_t \).

\[
\mathbf{PC}^i_t = \begin{pmatrix}
x_1 & x_2 & x_3 & \cdots & x_{n_{PC}^i} \\
y_1 & y_2 & y_3 & \cdots & y_{n_{PC}^i} \\
z_1 & z_2 & z_3 & \cdots & z_{n_{PC}^i} \\
1 & 1 & 1 & \cdots & 1
\end{pmatrix},
\]

\[
\mathbf{V}_t = \begin{pmatrix}
\mathbf{PC}_1^1 & \mathbf{PC}_1^2 & \mathbf{PC}_1^3 & \cdots & \mathbf{PC}_1^k
\end{pmatrix}.
\]
3.4 | Cluster correspondence

Following the identification of all objects within the environment, a cluster correspondence is performed. It is the task of finding association between clusters of two consecutively captured pointclouds. This is an essential step in the proposed mechanism which enables the tracking of a particular cluster in a series of pointcloud frames. A point cluster sampled at time $t$ is considered as $PC_t$ and the subsequently corresponding sampled cluster is considered as $PC_{t+1}$ (identifying the same object in the next pointcloud frame). When the following two constraints are satisfied, the identified object can be tracked:

1. The sensor has a high sampling rate (ie, difference between timestamps of consecutively captured data frames is small). For the Velodyne VLP-16 LiDAR such time is 0.1 second.
2. The sensor’s relative velocity with respect to the ground is within a specific limited range (eg, between 0 and 6 m/s).

Furthermore, it is assumed that the spatial positions of the clusters (ie, viewpoints) with respect to the sensor do not change drastically between two consecutive frames. This assumption can be used to find the correspondence between the clusters of two consecutive frames by using the spatial location of the clusters’ centroid as the correspondence feature. If the above two constraints are not satisfied, the proposed association method fails to conserve the information of fast-moving objects (in similar way Digital Particle Image Velocimetry in computational fluid dynamics fails to correlate the correspondence of moving particles when particles move at high-speed w.r.t. the camera capture frame rate\textsuperscript{27,28}). As these centroids’ locations are assumed to not change too much, a nearest neighbor approach can be used to match them.

A global nearest neighbor (GNN) approach implemented with Kd-tree data structure\textsuperscript{29} is used for matching the centroids of clusters between two consecutive pointcloud frames. The matching is reciprocal in nature and hence a correspondence between clusters $PC_t$ and $PC_{t+1}$ is considered correct, only if the nearest neighbor of $PC_t$’s centroid is $PC_{t+1}$’s centroid and vice versa. Results of the correspondence are stored as mappings between indices of cluster vectors $V_t$ and $V_{t+1}$, respectively (discussed in Section 3.3). That is, every pair of consecutive pointcloud frame has an associated map $M_t$ that stores the correspondence relations between the clusters of the two frames. An example of such correspondence map is shown in Equation (6). Due to the reciprocal correspondence property, map $M_t$ contains unique one-to-one mappings. The number of correspondences in a given map is less than or equal to the number of clusters in each of the two frames.

$$M_t = \left( [1 \rightarrow 3] \ [4 \rightarrow 8] \ldots \ [p \rightarrow q] \right)$$

$$1 \leq p \leq k_{t-1}; \ 1 \leq q \leq k_t \quad (k_t \text{ is the number of clusters in } V_t).$$

(6)
Although the assumption of small change in cluster’s position holds most of the time, but the possibility of wrong/false-positive correspondence cannot be ignored. In order to avoid false-positive correspondence, a volume constraint is applied on the bounding box (BB) of clusters matched by the Kd-tree. For this, two matched clusters must have a nearly equal volume of their bounding box. This condition must hold true due to the assumption that the viewpoint does not change drastically between two consecutive pointcloud frames. Hence, a normalized volume constraint as shown in Equation (7) is used to filter out false-positive correspondences from the generated map $M_t$. $\delta$ is an appropriate threshold for ensuring correct correspondence of clusters.

$$v_1 = \text{Volume}(BB(PC_t)),$$

$$v_2 = \text{Volume}(BB(PC_{t+1})),$$

$$\frac{|v_1 - v_2|}{|v_1 + v_2|} < \delta. \quad (7)$$

### 3.5 Cluster pose transformation

Theoretically, a moving object can be defined as an entity whose pose changes continuously with respect to the inertial frame. The clusters are analyzed by utilizing pose of the sensor capturing the pointclouds. The synchronized pointcloud-pose pair (discussed in Section 3.1) provides such information. That is:

1. Sensor pose difference between two consecutively captured pointclouds has information about the motion of sensor with respect to inertial frame.
2. Spatial change in pose of point clusters as perceived by the sensor has information about relative motion of objects with respect to the sensor.

These two pieces of information are combined mathematically to determine the motion state of objects with respect to the inertial frame, forming the backbone logic for the proposed DATMO approach. Point cluster of an object sampled at time $t$ is $PC_t$. Sensor pose while sampling the point cluster is represented by a $4 \times 4$ homogeneous matrix $P_t$, defined with respect to the inertial frame. Motion of the object in inertial frame affects the local pose $P_{Lt}$ of point cluster, as observed from the sensor’s frame. Motion state (moving or nonmoving) of the object is determined by the change in global pose $P_{Gt}$ of the corresponding point cluster, as observed from the inertial frame.

All the corresponding cluster pairs in $M_t$ are examined to find out clusters of moving objects around the sensor. Each corresponding cluster pair $PC_t$ and $PC_{t+1}$ (discussed in Section 3.4) has associated sensor pose matrices $P(t)$ and $P(t+1)$, respectively. The local pose of these two point clusters as observed from the sensor’s frame are $P_{Lt}$ and $P_{Lt+1}$, respectively. Herein, a transformation matrix $T$ is defined that changes the local pose of point cluster $PC_t$ from $P_{Lt}$ to $P_{Lt+1}$ and generates a transformed cluster $PC_t^*$ (shown in Equation (8)).

$$P_{Lt+1} = TP_{Lt},$$

$$T = P_{Lt+1}P_{Lt}^{-1},$$

$$PC_t^* = TPC_t. \quad (8)$$

$T$ represents a 3D transformation in the sensor’s frame that transforms all the points within $PC_t$ (as shown in Equation (5)) to generate a new point cluster $PC_t^*$. In simple terms, $PC_t^*$ is an approximation of $PC_{t+1}$ in sensor’s frame as both have the same local pose $P_{Lt+1}$. The global pose of the point cluster $P_{Gt}$ in terms of the local pose $P_{Lt}$ and sensor pose $P_t$ is shown in Equation (9).

$$P_{Gt} = P_tP_{Lt}. \quad (9)$$

Assuming that point clusters $PC_t$ and $PC_{t+1}$ belong to a nonmoving object in the surrounding, their global pose $P_{G(t)}$ and $P_{G(t+1)}$ must remain identical at time $t$ and $t + 1$, respectively. Considering this assumption, the transformation matrix $T$ can be represented solely in terms of sensor pose matrices $P(t)$ and $P(t+1)$ as shown in Equation (10).
\[ P_{G(t+1)} = P_{G(t)}P_{L(t+1)} \quad \text{(using Equation (9))}, \]
\[ P_{G(t+1)} = P_{G(t)}P_{L(t+1)} \quad \text{(using Equation (9))}, \]
\[ P_{G(t+1)} = P_{G(t)}P_{L(t+1)} \quad \text{(as } P_{G(t)} \text{ equals } P_{G(t+1)}), \]
\[ P_{G(t)} = P_{G(t+1)}P_{L(t)}^{-1}, \]
\[ P_{G(t+1)} = P_{G(t)}P_{L(t)}^{-1}, \]
\[ T = P_{G(t)}^{-1}P_{G(t+1)} \quad \text{(from Equation (8))}. \]

The transformation matrix \( T \) encapsulates motion information of the sensor and is used to approximate \( PC_{t+1} \) as \( PC_{t}^{*} \).

An example demonstration of a similar 2D pose transformation is shown in Figure 6A,B. A sensor (shaded triangle) moves toward a static/nonmoving object (black circle) identified as \( PC_{t} \). \( S(x,y) \) denotes the sensor position and \( PC_{t}(p,q) \) denotes the position of cluster centroid with respect to the reference frame. After the sensor moves “d” units toward the object, the cluster appears to have moved “d” units closer to the sensor (represented as \( PC_{t+1} \)) when observed from the sensor’s frame. Figure 6A shows the local pose of the point clusters in sensor’s frame and Figure 6B shows the global pose of the point clusters in the inertial frame. Matrix \( T \) in the example is \( 3 \times 3 \) as the transformation takes place in 2D. The same logic is also applicable to 3D spaces where the matrix \( T \) becomes a \( 4 \times 4 \) homogeneous transformation matrix.

A mathematical derivation for the above example is presented below showing that the transformation matrix for a nonmoving object’s cluster can be approximated from consecutively sampled sensor pose matrices.

\[
P_{L(t)} = \begin{pmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \quad P_{L(t+1)} = \begin{pmatrix} 1 & 0 & p-d \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \quad P_{G(t)} = \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} \quad P_{G(t+1)} = \begin{pmatrix} 1 & 0 & x+d \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}
\]

\[
T = P_{G(t+1)}P_{L(t)}^{-1} = P_{G(t+1)}P_{L(t)}^{-1} \quad \text{(from Equation (10))}
\]

\[
\Rightarrow T = \begin{pmatrix} 1 & 0 & p-d \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -x-d \\ 0 & 1 & -y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -d \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{pmatrix}
\]

\[
PC_{t}^{*} = TCP_{t} \quad \text{(from Equation (8))}
\]

\[
\Rightarrow \begin{pmatrix} p-d \\ q \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -d \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p \\ q \\ 1 \end{pmatrix}.
\]
The approximation of PC_{t+1} from PC_t as PC^*_t using the matrix T is valid only if the corresponding object is static/nonmoving with respect to inertial frame. As the local pose of PC^*_t is same as that of PC_{t+1} (ie, P_t(t+1)), both the point clusters overlap/superimpose in the sensor’s frame (shown in Figure 6A). However, if the object is moving, then the transformed pointcloud PC^*_t will not overlap with PC_{t+1} as T does not account for the motion of object in inertial frame. Thus, after such a transformation, the extent of overlap between PC_{t+1} and PC^*_t can be used to determine whether the corresponding object is moving or nonmoving with respect to inertial frame.

3.6 Moving object detection

The extent of overlap between PC_{t+1} and PC^*_t is determined with respect to the amount of spatial similarity detected between them. This approach of comparison detects the following two types of motion for an object’s cluster:

1. Movements of entire cluster with respect to the inertial frame.
2. Movement of a subset of points within the cluster with respect to the cluster itself (ie, within cluster movements).

For this, the octree pointcloud change detection algorithm proposed by Kammerl et al. has been used. The method ensures structural consistency while comparing the point clusters and adaptively matches the voxels of two octrees in a recursive manner. The change detection algorithm to determine spatial changes between PC^*_t and PC_{t+1} works in the following three steps:

1. A double buffered octree of resolution 30cm is constructed to represent the point clusters PC^*_t and PC_{t+1}.
2. As the two point clusters can potentially be different, the total number of points within all the voxels that spatially differ between the two point clusters are counted to determine \( \varphi \).
3. If the point count \( \varphi \) is more than a parameterized threshold \( \Phi \) (Equation (11)) then the cluster PC_{t+1} is identified as representing a moving object/entity.

\[
\varphi > \Phi(\eta),
\]

\[
\Phi(\eta) = \frac{n_{PC^*_t} + n_{PC_{t+1}}}{\eta}.
\]

Equation (11) shows the moving object detection constraint where \( n_{PC^*_t} \) (from Equation (5)) is the number of points in the cluster PC^*_t. The normalization parameter \( \eta \) in Equation (11) controls the sensitivity of moving object detection. With a small value of \( \eta \), it fails to detect small within-cluster changes (eg, a person standing at a fixed place and waiving hand) as the threshold \( \Phi \) tends to be too large to detect such small changes. It may also fail to detect the motion of objects moving very slowly. On the other hand, with a high value of \( \eta \), any small change between the point clusters can be easily detected; however, it also increases the number of false-positive results (ie, static objects might get detected as moving). False positive detections occur when threshold \( \Phi \) tends to be too small, due to which nonmoving clusters erroneously get detected as moving. Hence, the parameter \( \eta \) is properly tuned with respect to the targeted environment and type of sensor used.

The detection results for all the clusters within a pointcloud frame captured at time \( t \) is herein represented as a boolean vector \( R_t \) (Equation (12)). Number of elements in \( R_t \) is equal to the number of detected clusters in that frame (ie, \( k \) from Equation (5)). Each index value \( r^i \) is either true or false representing whether the cluster at the corresponding index in \( V_t \) is moving (dynamic) or nonmoving (static).

\[
R_t = \begin{bmatrix} r^1 & r^2 & r^3 & \ldots & r^k \end{bmatrix},
\]

\[
r^i = \begin{cases} 
    \text{True} & \text{if } PC^*_t \text{ is moving} \\
    \text{False} & \text{otherwise} 
\end{cases}.
\]

3.7 Confidence based cluster tracking for reducing false positive detections

Despite of false positive detections, \( \eta \) must be assigned a high value for accurate and sensitive moving object detection in indoor environments. By analyzing the detection results from Section 3.6, it was found that even with a high value of \( \eta \) the following two observations always hold:
(1) Truly moving objects get detected as moving on multiple consecutive frames (however, occasionally they get erroneously identified as static due to motion inconsistencies).
(2) Static objects get occasionally detected as moving, due to the variation in the sensor’s motion.

Intuitively, by observing the detection results for a finite number of consecutive pointcloud frames a consistency relation is found to reduce the number of false positive detections (i.e., clusters detected as moving in a consistent manner are highly likely to be actually moving than the clusters those are occasionally detected as moving). Following such intuition, a confidence based tracking algorithm is used to filter out any false positive detections present in the result vector $R_t$. The proposed method uses result vectors and correspondence maps from a series of historical frames to identify the clusters that are consistently detected as moving. The number of frames to track for analyzing the presence of moving clusters, is denoted as a parameter $\zeta$. A cluster in the latest frame is considered as moving only if it has been detected as moving in all the last $\zeta$ frames (including the latest frame).

With each incoming pointcloud, the computed correspondence map $M_t$ and detection result vector $R_t$ are appended to two separate buffers $M_b$ and $R_b$, respectively. Every pointcloud frame is associated with one result vector, and every consecutive frame pair is associated with one correspondence map. For analyzing $\zeta$ frames, the tracking algorithm requires $\zeta$ result vectors and $\zeta - 1$ correspondence maps. Equation (13) shows the structure of $M_b$ and $R_b$ where $R_{t-1}$ refers to the result vector of the $i$th historical frame and $M_{i+1}$ is the correspondence map between $i$th and $i+1$th historical frames (e.g., $M_{t+1}$ stores correspondence relations between $R_{t+2}$ and $R_{t+1}$). This result buffer $R_b$ is analyzed using a recursive algorithm that utilizes the correspondence relations from $M_b$ to determine clusters getting continuously detected as moving in the last $\zeta$ frames.

$$M_b = \left( \begin{array}{cccc} M_{t-(\zeta-2)} & \ldots & M_{t-2} & M_{t-1} & M_t \end{array} \right),$$
$$R_b = \left( \begin{array}{cccc} R_{t-(\zeta-1)} & \ldots & R_{t-2} & R_{t-1} & R_t \end{array} \right).$$

(13)

Moving clusters are tracked using their centroids and these moving centroids are stored in a tracking vector $T_m$. However, if a new moving object appears in the range of the sensor, initial $\zeta$ detections are used to build up the confidence that the corresponding cluster is truly moving following which the cluster centroid is added into $T_m$. If a particular cluster is found to be moving then its centroid is appended to $T_m$ with a nonmoving confidence value $\beta = 0$ (zero). Equation (14) shows the tracking vector $T_m$ as a metacentroid object where a cluster $X$ with nonmoving confidence $\beta$ is denoted as $X(\beta)$. Each $X(\beta)$ holds the 3D (x,y,z) coordinate of the cluster centroid and associated nonmoving confidence value. In order to prevent adding redundant centroids (centroids of moving clusters which have previously been added to $T_m$), a new centroid is added to $T_m$ only if its euclidean distance is more than a threshold $\Delta_{\text{catch-up}}$ from all the existing centroids in $T_m$.

$$T_m = \left( \begin{array}{cccc} X_1(\beta) & X_2(\beta) & X_3(\beta) & \ldots \end{array} \right).$$

(14)

For every newly captured pointcloud, all the centroids in $T_m$ are matched to the cluster centroids within the new pointcloud using a nearest neighbor approach. This determines the new positions of moving clusters with respect to the sensor following which centroids in $T_m$ are updated with their matched counterparts. In real-world environments however, a moving cluster can go out of sensor’s range, a cluster may also disappear momentarily due to partial or complete occlusion of the moving object. For such edge cases, the nearest neighbor approach can match to a nearby nonmoving cluster, thus updating a centroid in $T_m$ with an ambiguous nonmoving centroid. To handle such ambiguities, $\beta$ value for each centroid in $T_m$ is updated after the centroid matching procedure. $\beta$ for each centroid is updated according the following three policies:

(1) If the matched cluster is detected as nonmoving then $\beta$ is incremented by one unit ($\beta = \beta + 1$). Such an increment is observed when a moving cluster goes out of the sensor’s range or a detection miss occurs due to partial or complete occlusion of the cluster.
(2) If the euclidean distance to the matched cluster’s centroid is greater than a threshold $\Delta_{\text{leave-off}}$ then $\beta$ is incremented by one unit ($\beta = \beta + 1$). If the distance to the matched cluster centroid is large, it is assumed that the matching process has produced a wrong correspondence of cluster centroids.
(3) If the matched cluster is detected as moving, then $\beta$ is decremented by one unit ($\beta = \beta - 1$). Such a decrement ensures that moving clusters which occasionally get detected as nonmoving due to motion inconsistencies or occlusions continue to be classified as moving.

From above policy 1 and 2, it can be inferred that nonmoving confidence $\beta$ is a value that identifies the number of consecutive frames in which a tracked cluster has been detected as nonmoving. A centroid retains its existence in $\mathbf{T}_m$ until its maximum value condition $\beta \leq \beta_{\text{max}}$ is satisfied (ie, if a centroid’s $\beta$ exceeds the threshold then it is removed from $\mathbf{T}_m$). $\beta_{\text{max}}$ is thus defined as the maximum number of consecutive frames in which a tracked cluster can be detected as nonmoving. $\beta$ also holds a minimum value condition of $\beta \geq 0$ (ie, if $\beta$ is equal to 0 then its value is no longer decreased). It must be noted that for edge cases, a centroid in $\mathbf{T}_m$ gets ambiguously updated with a nonmoving cluster centroid. Thus, the corresponding nonmoving cluster gets falsely tracked as moving. However, within the next $\beta_{\text{max}}$ pointcloud frames, $\beta$ for such ambiguous centroid exceeds the threshold and hence the centroid gets removed from $\mathbf{T}_m$. $\beta$ is a parameter used to ensure that all the cluster centroids within $\mathbf{T}_m$ are continuously detected as moving by the detection algorithm. As a result the proposed approach keeps track of the moving clusters in real-time.

The tracking algorithm has been designed to handle dense indoor spaces that are common in real-world scenarios. Specifically, the algorithm considers people walking around within the sensor’s range. Situations like sudden disappearance of moving objects due to partial or complete occlusions are handled by the nonmoving confidence $\beta$. On the other hand, the moving confidence $\zeta$ ensures better detection output by reducing false positive results. The proposed approach is also suitable for multiple object tracking when the moving objects are well separated from each other. For an environment with multiple objects moving very close to each other, the confidence tracker might face difficulties in distinguishing one object from another due to the use of nearest neighbor approaches.

### 3.8 Moving object removal

The last step in the proposed methodology is to remove the tracked moving objects from newly captured pointcloud frames. As discussed in Section 3.7, when a new pointcloud arrives, all the centroids within $\mathbf{T}_m$ are matched with the cluster centroids of the new pointcloud using a nearest neighbor approach. These matched clusters are then removed to generate a filtered pointcloud without any moving objects in it. Pointcloud thus obtained does not have any dynamic clusters in it which can later be used by SLAM methods to generate maps for indoor environment without any spurious artifacts.

### 4 EXPERIMENT SETUP

To test and validate the proposed methodology, experiments were conducted using a life-size humanoid robot. Head of the robot was mounted on a tilt/pan gimbal neck and sensor data were collected while manually navigating the robot in a cluttered indoor office environment. Figure 7A shows a picture of the office corridor where sensor data were collected. The environment was setup to represent a cluttered space with objects of various shapes and dimensions such that the proposed work could be tested against typical challenges faced by robots navigating in GPS-denied environments. Sensors mounted on the head included a Velodyne VLP-16 LiDAR and an Intel RealSense T265 tracking camera. Figure 7B shows the experimental head of the humanoid robot mounted with sensors. The RealSense T265 sensor was used for collecting odometry (pose) information of VLP-16 LiDAR with respect to the inertial frame. A Linux system with 8 GB RAM running on Intel-i5-9th generation processor was used as the computing unit for the experiments. Using the abovementioned hardware, the proposed algorithms were implemented in C++ using Point Cloud Library (PCL) and Robot Operating System (ROS). RViz was used as the primary visualization tool. Available ROS sensor packages were used as drivers to access data from the respective sensors. The source code of the work has been open-sourced at https://github.com/prabinrath/dynamicslamtool for easy replication of the results discussed in this article.
RESULT ANALYSIS

A number of experimental tests were conducted to analyze and validate the proposed methodology. Such tests consisted of static and dynamic confined spaces which included one or more moving entities. In what follows, an illustrative test representative of the obtained results is described. Pointcloud data set used for the selected experiment was collected for 28.1 seconds at 10 frames per second. Simultaneously, the humanoid robot was moved through the office corridor, mimicking the motion during a typical office task such as distributing the mail. The environment had one moving object (ie, a person walking toward and away from the sensor’s field of view) and had multiple nonmoving (static) objects (eg, tables, chairs, cubicles, a bicycle, a white board). A sample of the pointcloud captured within such an environment was shown in Figure 2.

Data collected from the experimental environment comprised 279 pointcloud frames. The ground truth for moving objects were obtained by manually marking (hand-labeled) the moving entities within each of the pointcloud frames. Following the euclidean clustering step (Figure 5), the result of moving object being detected and tracked is shown in Figure 8A where the moving person’s point cluster is found to be enclosed within a pink bounding box. Figure 8B shows the raw pointcloud after removing the tracked cluster of the moving person.

As discussed in Sections 3.6 and 3.7, moving object detection without cluster tracking generates a large number of false positive results. However, the use of cluster tracking algorithm significantly reduces the number of false positive detections. In the analytical results discussed ahead, false positive detections were computed as a cumulative sum of false positive detections from all the frames (ie, sum of all the false positive detections in 279 frames for the discussed experiment). The variation in the number of false positive detections with respect to the sensitivity parameter $\eta$ was analyzed. The results of the experiment with and without using cluster tracking are shown in Tables 2 and 1, respectively. The four column headings in the tables represent the following information:

- **Total**: Total number of moving object detections (for the discussed experiment it is the sum of all the detections in 279 frames).
- **$m \rightarrow m$**: Number of true positive detections, that is, moving object detected as moving (for the discussed experiment it is the number of times the walking person’s cluster got detected as moving).
- **$m \rightarrow s$**: Number of detection misses, that is, moving object failed to be detected as moving (for the discussed experiment it is the number of times the walking person’s cluster got detected as nonmoving).
- **$s \rightarrow m$**: Number of false positive detections, that is, a static object gets detected as moving. ($\text{Total} - [m \rightarrow m]$)

For the experiments performed with cluster tracking, the parameters $\Delta_{\text{catch-up}}$ and $\Delta_{\text{leave-off}}$ were fixed to 0.2 and 0.5 m, respectively.
Figure 9A shows a graphical variation in number of false positive detections with respect to the variation in the sensitivity parameter $\eta$. The reason for the reduction in false positive detections after using the cluster tracking can be attributed to the confidence criteria used by the tracking algorithm. Such figure also shows the increasing trend in the number of false positives with respect to the increase in the detection sensitivity as discussed in Section 3.6.

The moving confidence $\zeta$ and the nonmoving confidence threshold $\beta_{\text{max}}$ are the two critical parameters for the cluster tracking algorithm. With a high moving confidence, nonmoving clusters rarely satisfy the confidence criteria and hence false positive results reduce significantly. A high nonmoving confidence threshold increases the lifetime of ambiguous centroids inside tracking vector $T_m$ and hence the number of false positive detections are amplified. The results of the experiment with variation in moving confidence and nonmoving confidence threshold are shown in Table 3. Figure 9B shows a graphical variation of false positives with respect to variation in tracking parameters.

Accuracy of moving object detection for the proposed approach can be defined as shown in Equation (15). True positive detections (ie, “m $\rightarrow$ m”) are the number of moving cluster detections against truly moving clusters. The accuracy of moving object detection achieved without cluster tracking and with cluster tracking are shown in Figure 10A using yellow and blue bars, respectively. It can be observed that the cluster tracking algorithm reduces the number of false positives with the cost of reduction in detection accuracy. A reason for this is, after a cluster gets detected as moving by the detection algorithm, initial $\zeta$ detections are used to build up the confidence and confirm that the cluster is actually moving.

\[
\text{Accuracy (\%)} = \frac{\text{True positive detections}}{\text{Ground truth detections}} \times 100. \quad (15)
\]
The variation of moving object detection accuracy with respect to variation in tracking parameters is shown as a heat map in Figure 10B. A heat map is a graphical representation of data where the individual values contained in a matrix are represented as different shades of a color. The nonmoving confidence $\beta_{\text{max}}$ improves the consistency in moving object detection (Section 3.7) and hence a higher value helps to achieve a higher detection accuracy.

An analysis of the experimental results can be used to select the optimal set of parameters for a given application environment. For the indoor office environment discussed in this article, the optimal set of parameters is $\zeta = 4$, $\beta_{\text{max}} = 3$, and $\eta = 20$ which provide a 91.42% accuracy with six false positive detections. Choice of optimal parameter set is subject to the existing trade-off between accuracy and false positive detections.

For a more extended validation of the research work, experiments were also conducted with the publicly available KITTI Campus data set. The proposed work has been specifically curated for cluttered indoor environments. However, the campus data set from KITTI can be considered as a similar dynamic environment that a humanoid may come across while navigating in indoor environments. The data set is collected using a Velodyne HDL-64E lidar attached to the roof of a moving car. Odometry data are obtained from an OXTS RT 3003 inertial and GPS navigation system. Figure 11A shows an RGB image of the scene, obtained from the front camera of the car. Figure 11B shows three persons (two pedestrians and one cyclist) being detected and tracked by the proposed approach. As the work only considers an area within $6 \times 6$ m$^2$ around the sensor, the presented pointclouds have been zoomed in for providing a clear and enlarged view of the concerned region. Figure 11C shows the pointcloud obtained after removing tracked moving objects from the raw data. The results of the experiment show that the proposed method is able to detect and track multiple moving objects successfully.

**TABLE 3** Results with variation in cluster tracking parameters (detection sensitivity $\eta = 20$)

| Value of $\zeta$ | Value of $\beta_{\text{max}}$ | Total  | $m \rightarrow m$ | $m \rightarrow s$ | $s \rightarrow m$ |
|------------------|-------------------------------|--------|-------------------|-------------------|-------------------|
| 4                | 0                             | 53     | 53                | 52                | 0                 |
| 4                | 1                             | 93     | 86                | 19                | 7                 |
| 4                | 2                             | 99     | 93                | 12                | 6                 |
| 4                | 3                             | 102    | 96                | 9                 | 6                 |
| 3                | 0                             | 79     | 66                | 39                | 13                |
| 3                | 1                             | 131    | 85                | 20                | 46                |
| 3                | 2                             | 152    | 90                | 15                | 62                |
| 3                | 3                             | 170    | 101               | 4                 | 69                |
| 2                | 0                             | 177    | 76                | 29                | 101               |
| 2                | 1                             | 311    | 92                | 13                | 219               |
| 2                | 2                             | 401    | 98                | 7                 | 303               |
| 2                | 3                             | 476    | 102               | 3                 | 374               |
FIGURE 10  Accuracy variations with respect to algorithm parameters

(A) Accuracy achieved with cluster tracking is found to be less than the one achieved without using cluster tracking. (Tracking parameters: $\xi = 3$, $\beta_{\text{max}} = 2$)

(B) Heatmap showing the variation of moving object detection accuracy with respect to tracking parameters. (Detection sensitivity $\eta = 20$)

FIGURE 11  Results of experiment with KITTI data set

(A) RGB image of a scene from KITTI Campus dataset.

(B) Moving objects tracked using confidence based cluster tracking.

(C) Pointcloud after removal of identified moving objects from raw data.
Computation time is a major constraint while processing 3D pointcloud data. The proposed approach for moving object detection and removal has eight steps which are executed serially. On an average, total computation time per pointcloud frame is about 60-70 milliseconds. Major computationally intensive tasks are ground plane removal ($\approx 32$ milliseconds) and euclidean clustering ($\approx 10$ milliseconds) while the other remaining steps collectively take about 20-25 milliseconds. Many methods discussed in Section 2 use Bayesian formulations for tracking of moving objects which is computationally expensive. The advantage in computation time when compared with these methods is achieved using GNN-based approaches and confidence-based cluster tracking algorithm.

6 | CONCLUSIONS AND FUTURE WORK

A DATMO pipeline was proposed for humanoid navigation in GPS-denied environments. Although the work has been developed in context of humanoid robots, it is not limited to that and can be used with a variety of mobile robots which try to navigate and map in cluttered indoor environments. The contributions and experimental results of the proposed research work is recapitulated below:

- Cluster pose transformation approach can be reliably used for moving object detection provided that accurate odometry data of the sensor is available. With a proper tuning of sensitivity parameter, it can detect within cluster movements which is typically found in dynamic indoor environments.
- Confidence-based cluster tracking algorithm can be used for reducing false positive results significantly. The confidence constraints employed by the proposed algorithm are suitable for handling inconsistencies in motion of the moving objects found in indoor environments.
- Voxel grid covariance based ground plane removal overcomes the limitations of conventional plane fitting methods which do not perform well due to unavailability of enough ground plane points. Such unavailability is typical to indoor cluttered environments.
- The entire pipeline takes less than 100 milliseconds per frame while running on a standard computing unit. Hence, it is potentially suitable for use in real-time robot navigation and mapping.

As a scope of future work the research will be focused on improving flexibility and performance of the proposed approach. The algorithms discussed have a few number of parameters that need to be tuned for getting optimal results in different application environments. The tuning performed during the experiments discussed in this article were based on a trial and error approach. An adaptive approach will be developed for the parameters such that they can be changed in real time, depending on the spatial features of the pointcloud. Efficient pointcloud clustering methods such as Reference 36 will be explored for a faster real-time usage. Detection of moving objects will be attempted using machine learning algorithms based on 3D feature comparison between point clusters. Such developments will be focused toward reducing false positive results as much as possible and to improve overall detection accuracy.

ACKNOWLEDGEMENT
The research is supported by MITACS Globalink 2019 and NSERC grant programs.

PEER REVIEW INFORMATION
Engineering Reports thanks Jonathan Lwowski and other anonymous reviewer(s) for their contribution to the peer review of this work.

PEER REVIEW
The peer review history for this article is available at https://publons.com/publon/10.1002/eng2.12275.

CONFLICT OF INTEREST
The authors declare that there is no conflict of interest regarding the publication of this article.
AUTHOR CONTRIBUTIONS
Alejandro Ramirez-Serrano equally contributed to the conceptualization, investigation, project administration, validation and writing—review and editing. Prabin Rath equally contributed to the formal analysis, methodology and validation. Dilip Kumar Pratihar equally contributed to the investigation.

ORCID
Alejandro Ramirez-Serrano https://orcid.org/0000-0003-0562-0121

REFERENCES
1. Cadena C, Carlone L, Carrillo H, et al. Past, present, and future of simultaneous localization and mapping: towards the robust-perception age. IEEE Trans Robot. 2016;32(6):1309-1332.
2. Taketomi T, Uchiyama H, Ikeda S. Visual SLAM algorithms: a survey from 2010 to 2016. IPSJ Trans Comput Vis Appl. 2017;9(1):16.
3. Sriranawan P, Prasanjith MD, Wang D. 3D feature points detection on sparse and non-uniform pointcloud for slam. Paper presented at: Proceedings of the 2017 18th International Conference on Advanced Robotics (ICAR), Hong Kong; 2017:112-117; IEEE. http://dx.doi.org/10.1109/ICAR.2017.8023504.
4. Hähnel D, Schulz D, Burgard W. Mobile robot mapping in populated environments. Adv Robot. 2003;17(7):579-597.
5. Zhang Z. Iterative point matching for registration of free-form curves and surfaces. Int J Comput Vis. 1994;13(2):119-152.
6. Llamazares Á, Molinos EJ, Ocaña M. Detection and tracking of moving obstacles (DATMO): a review. Robotica. 2019;38(5):1-14.
7. Vu T-D, Burlet J, Aycard O. Grid-based localization and local mapping with moving object detection and tracking. Inf Fusion. 2011;12(1):58-69.
8. Wolf DF, Sukhatme GS. Mobile robots simultaneous localization and mapping in dynamic environments. Auton Robot. 2005;19(1):53-65.
9. Litomisky K, Bhanu B. Removing moving objects from point cloud scenes. Paper presented at: Proceedings of the 2011 IEEE International Conference on Pattern Recognition (ICPR), Cancun, Mexico; 2011:561-566; IEEE.
10. Wurm KM, Hornung A, Bennewitz M, Stachniss C, Burgard W. OctoMap: a probabilistic, flexible, and compact 3D map representation for robotic systems. Paper presented at: Proceedings of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation, Anchorage, Alaska; vol. 2, 2010.
11. Endres F, Hess J, Sturm J, Cremers D, Burgard W. 3-D mapping with an RGB-D camera. IEEE Transactions on Robotics. 2013;30(1):177-187.
12. Fischler MA, Bolles RC. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun ACM. 1981;24(6):381-395.
13. Ayala D, Brunet P, Juan R, Navazo I. Object representation by means of non-minimal division quadtrees and octrees. ACM Trans Graph. 1985;4(1):41-59.
14. Russ RB. Semantic 3D object maps for everyday manipulation in human living environments. KI-Künstliche Intelligenz. 2010;24(4):345-348.
15. Adrian RJ. Twenty years of particle image velocimetry. Exp Fluids. 2005;39(2):159-169.
16. Termini D, Di Leonardo A. Efficiency of a digital particle image velocimetry (DPIV) method for monitoring the surface velocity of hyper-concentrated flows. Geosciences. 2018;8(10):383.
29. Friedman JH, Bentley JL, Finkel RA. An algorithm for finding best matches in logarithmic expected time. *ACM Trans Math Softw.* 1977;3(3):209-226.
30. Kammerl J, Blodow N, Rusu RB, Gedikli S, Beetz M, Steinbach E. Real-time compression of point cloud streams. Paper presented at: Proceedings of the 2012 IEEE International Conference on Robotics and Automation, Saint Paul, Minnesota, USA; 2012:778-785; IEEE.
31. Hall DS. *High definition lidar system*. US Patent 7,969,558, Velodyne Lidar Inc; 2011.
32. https://www.intelrealsense.com/tracking-camera-t265.
33. Rusu RB, Cousins S. 3d is here: point cloud library (pcl). Paper presented at: Proceedings of the 2011 IEEE International Conference on Robotics and Automation, Shanghai, China; 2011:1-4; IEEE.
34. Quigley M, Conley K, Gerkey B, et al. ROS: an open-source robot operating system. Paper presented at: Proceedings of the ICRA workshop on open source software; vol. 3, 2009:5; Kobe, Japan.
35. Geiger A, Lenz P, Stiller C, Urtasun R. Vision meets robotics: the KITTI dataset. *Int J Robot Res (IJRR)*. 2013;32(11):1231–1237.
36. Bogoslavskyi I, Stachniss C. Fast range image-based segmentation of sparse 3D laser scans for online operation. Paper presented at: Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea; 2016:163-169; IEEE.

**AUTHOR BIOGRAPHIES**

**Prabin Kumar Rath** received his bachelor in technology (BTech) degree from Computer Science and Engineering Department, National Institute of Technology, Rourkela (India). He has a firm experience in the field of robot perception and control. He completed his research internships from University of Calgary (Canada) and Indian Institute of Technology, Kharagpur (India) during which he worked on developing efficient perception algorithms for humanoid robots and control algorithms for rehabilitation robots, respectively. He also has a keen interest in computer networks and its application within the domain of Internet of Things.

**Alex Ramirez-Serrano** has a BSc degree in Mechanical Eng. (Mexico), two Master of Science degrees, one in Mechanical & Aerospace Eng. (USA) and a second in Computer Science/Artificial Intelligence (Mexico) and a PhD in Mechanical & Industrial Eng. (Canada). Dr Ramirez-Serrano is a full-time professor at the University of Calgary where he has served on diverse roles including a former director of the manufacturing program and director of the graduate program within the Department of Mechanical and Manufacturing Engineering. He is the founder and director of the Unmanned Vehicle Systems (UVS) Robotarium research laboratory where he performs R&D activities in-ground, aerial and humanoid robotics. His expertise are in the areas of design, control, navigation and modeling of UVS. He is also the founder and CEO of 4Front Robotics, a Canadian based robotics company developing UVS for operation in confined complex spaces.

**Dilip Kumar Pratihar** received his PhD from Indian Institute of Technology (IIT) Kanpur, India, in 2000. He received University Gold Medal, A.M. Das Memorial Medal, Institution of Engineers’ (I) Medal and others. He completed his postdoctoral studies in Japan and then, in Germany under the Alexander von Humboldt Fellowship Programme. He is working now as a Professor (HAG Scale) of Mechanical Engineering Department and PIC, Centre for Excellence in Robotics, IIT Kharagpur, India. His research areas include robotics, soft computing and manufacturing science. He has published more than 270 research papers. He has authored three textbooks, edited one book and coauthored four reference books. He has guided 22 PhDs. He is in editorial board of 12 International Journals. He has been elected as FIE, MASME and SMIEEE. He has completed a number of sponsored and consultancy projects.

**How to cite this article:** Kumar Rath P, Ramirez-Serrano A, Kumar Pratihar D. Real-time moving object detection and removal from 3D pointcloud data for humanoid navigation in dense GPS-denied environments. *Engineering Reports*. 2020;2:e12275. https://doi.org/10.1002/eng2.12275