An Efficient Approach for Detecting Moving Objects and Deriving Their Positions and Velocities*

Andreas Gustavsson

Mälardalen University, Högskoleplan 1, 722 20 Västerås, Sweden

Abstract. Well-functioning autonomous robot solutions heavily rely on the availability of fast and correct navigation solutions. The presence of dynamic/moving objects in the environment poses a challenge because the risk of collision increases. In order to derive the best and most foreseeing re-routing solutions for cases where the planned route suddenly involves the risk of colliding with a moving object, the robot's navigation system must be provided with information about such objects' positions and velocities.
Based on sensor readings providing either 2-dimensional polar range scan or 3-dimensional point cloud data streams, we present an efficient and effective method which detects objects in the environment and derives their positions and velocities. The method has been implemented, based on the Robot Operating System (ROS), and we also present an evaluation of it. It was found that the method results in good accuracy in the position and velocity calculations, a small memory footprint and low CPU usage requirements.

Keywords: Computer Vision Algorithms, Object Detection, Robotics, ROS.

1 Introduction

Fast and correct navigation, including object avoidance, is a crucial ability for any autonomous agent (e.g. autonomous robot, human, etc.) working in a dynamic environment. Navigation is often based on a global (i.e. known and static) map of the environment. When the environment might be dynamically changing, account is usually also taken to sensor readings, which are used to create a local (i.e. temporarily valid) map, on top of the global map, to avoid collisions with dynamic objects.

Based on the idea of using a global and a local map, a novel method, introducing navigation based on magnetic flow fields, has been proposed [24]. This method relies on the existence of a base flow field (cf. a global map), showing a collision-free path (if any) from the start point to the goal point. In case objects can move, or be moved, they cannot simply be static parts of the base flow field. The method therefore relies on information about the position and velocity of such objects. Each object is seen as having a dipole flow field, whose magnitude is proportional to the speed of the object and whose direction is aligned with its *velocity*. The total flow field, and thus the path,

*This is a pre-copyedited version of a contribution published in K. Arai and S. Kapoor (editors), Advances in Computer Vision, published by Springer International Publishing. The definitive authenticated version is available online via https://doi.org/10.1007/978-3-030-17798-0_25.

is updated in accordance to how the *position* of each detected object affects it. This strategy provides a very foreseeing and effective solution to the object avoidance re-routing problem. The proposed method [24] is destined to be implemented within the Robot Operating System, ROS [22].

In this paper, we present an effective and efficient method for detecting objects and deriving the position and velocity of these. The method relies on receiving a sequence of either 2-dimensional (2D) polar range scans or 3-dimensional (3D) point clouds. Our method thus serves to satisfy the prerequisite knowledge about moving objects for the mentioned path-planning algorithm, and for a multitude of other systems relying on the information provided by our system.

An implementation of the method as a ROS library and a few executable ROS nodes is provided open source [13]. Two of the nodes take as input a stream of `LaserScan` [2] messages and a stream of `PointCloud2` [5] messages, respectively, and use the library for detecting objects and deriving their positions and velocities. The position and velocity of each detected object are derived in several ROS coordinate systems (from here on referred to as *frames*) [18], including `map`, so that an agent's findings can be shared with other agents in a globally interpretable way. Thus, an agent could easily be made to benefit from other agents' findings. The library also provides the possibility for the user to define a confidence value for each found object, if desired. The implemented method has been verified and evaluated, both using simulations and physical experiments.

The rest of this paper is organized as follows. Section 2 details the foundations of our approach for finding moving objects, based on a single sensor data stream, while Sect. 3 presents our approach for utilizing several sensor data streams. Section 4 briefly explains the ROS-based implementation of our method. Section 5 explains the experiment setup we have used to perform the evaluation (based on the implementation) which is presented in Sect. 6. Some related research is presented in Sect. 7, and Sect. 8 concludes the paper and states some future tracks for our research.

2 Finding Moving Objects in 2D Polar Range Scan Data

This section presents our method for finding the positions and velocities of detected moving objects (no object classification is performed). The method is founded on the data having the form of consecutive 2D polar range scans of the environment (cf. Fig. 1). The sensory source is assumed always to produce the same number, n, of scan point ranges. Each range should represent the distance to the closest obstructing object at a given and known angle, around the sensor z-axis and relative the sensor x-axis (cf. the data produced by a 2D laser range scanner). Note, however, that the above assumptions only represent the internal structure of our method and that an actual sensory source could deliver for example 3D data and have the z-axis pointing forward instead of the x-axis (cf. an *optical* frame [11]), as will become apparent in Sect. 4. The frame of the sensor is assumed to be Cartesian and right-handed.

Each scan is assumed to have an attached timestamp, representing the time at which the scan was acquired. The scans are assumed to be streamed in messages (an actual system need not have this type of setup, this is just the terminology we use to describe

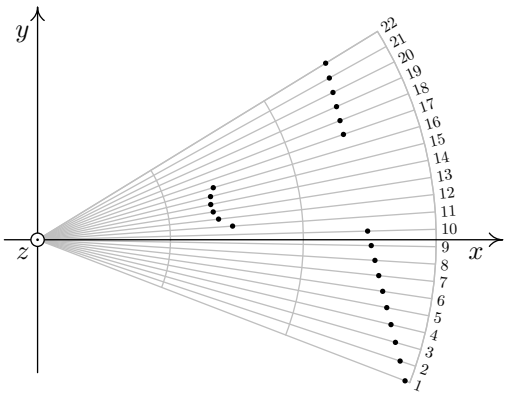


Fig. 1. 2D polar range scan example—a circular object in front of a wall. Scan rays are numbered.

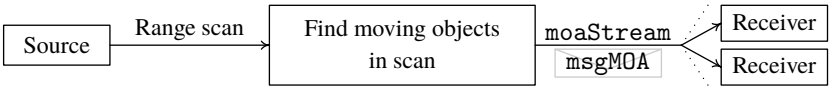


Fig. 2. Architectural description.

our method), from a *single* source (i.e. sensor). The data related to the set of found moving objects is, in turn, streamed in a message, to a set of receivers (cf. Fig. 2).

The main structure of our method is outlined in Algorithm 1. We keep a history of data scans in a bank, `bank` (cf. Fig. 3), which is similar to a circular buffer, except that incoming data is immediately handled. Setting the size, m, of the bank appropriately allows us to derive accurate velocities for the found objects, as discussed below. The oldest and newest scan messages in the bank are pointed to by `oldest` and `newest`, respectively. When a message is received on the stream, `dataStream`, the (timestamped) range data of that message replaces the oldest range data in the bank. To allow for reducing the influence of high-frequent noise, the received range data can be adapted using exponential moving average (EMA; row 7). In Algorithm 1, the parameter $\alpha \in [0, 1]$ represents the degree of weighting decrease—a higher value of α decreases the influence of old range data faster.

Objects are detected in the incoming scan data message by iterating through, comparing and grouping consecutive range values. First, the next valid scan point is found (rows 11–15). To be valid, its range value must lie wihtin $[\mathtt{th}^{\min}_{\text{dist}}, \mathtt{th}^{\max}_{\text{dist}}]$. Second, all consecutive valid scan points, whose ranges are not pair-wise separated by more than $\mathtt{th}^{\max}_{\delta_{\text{edge}}}$, are counted (rows 16–24). The derived scan points are considered to constitute one unique object. If the considered object includes the scan point with the lowest index and the sensor delivering the messages is viewing 360°, then the high end of the scan points must also be accounted for, since they might also belong to this object

Algorithm 1. Find moving objects

Input: dataStream, bank, oldest, newest, α, $\mathtt{th}^{\max}_{\delta_{\text{edge}}}$, $\mathtt{th}^{\min}_{\text{pts}}$, $\mathtt{th}^{\min}_{\text{dist}}$, $\mathtt{th}^{\max}_{\text{dist}}$, $\mathtt{th}^{\max}_{\delta_{\text{dist}}}$, $\mathtt{th}^{\max}_{\delta_{\text{width}}}$, $\mathtt{th}^{\min}_{\text{conf}}$
Output: msgMOA on moaStream

```
1  procedure
2      loop
3          Create new msgMOA
4          Receive msg with timestamp, timestamp, and array of distances, data, from dataStream
5          bank[oldest].timestamp ← msg.timestamp
6          for all i iterating msg.data do                ▷ Parallel operation—replace oldest data (EMA:ed)
7              bank[oldest].data[i] ← α × msg.data[i] + (1 − α) × bank[newest].data[i]
8          Update oldest and newest pointers
9          i ← 1
10         limit ← |bank[newest].data|
11         while i ≤ limit do                             ▷ Sequential operation
12             range_i ← bank[newest].data[i]             ▷ First point of object
13             if range_i < th^min_dist or th^max_dist < range_i then
14                 i ← i + 1
15                 continue
16             nrObjectPoints ← 1                         ▷ Object consists of scan point i so far
17             rangePrev ← range_i
18             for j ← i + 1 ; j ≤ |bank[newest].data| ; j ← j + 1 do   ▷ Count object scan points
19                 range_j ← bank[newest].data[j]
20                 if th^min_dist ≤ range_j ≤ th^max_dist and |rangePrev − range_j| ≤ th^max_δedge then
21                     nrObjectPoints ← nrObjectPoints + 1
22                 else
23                     break
24                 rangePrev ← range_j
25             if i = 1 and sensor views 360° then        ▷ Valid first point, must also account high end of scan
26                 rangePrev ← range_i
27                 for k ← |bank[newest].data| ; j < k ; k ← k − 1 do
28                     range_k ← bank[newest].data[k]
29                     if th^min_dist ≤ range_k ≤ th^max_dist and |rangePrev − range_k| ≤ th^max_δedge then
30                         nrObjectPoints ← nrObjectPoints + 1
31                         limit ← limit − 1              ▷ Do not account for this point from lower end
32                     else
33                         break
34                     rangePrev ← range_k
35             if th^min_pts ≤ nrObjectPoints then
36                 Calculate position, position_new, and width of object
37                 Try to track object through bank given th^min_dist, th^max_dist, th^max_δedge th^max_δdist, th^min_pts, th^max_δwidth
38                 if object could be tracked then
39                     Calculate position, position_old, in oldest scans
40                     δ_t ← bank[newest].timestamp − bank[oldest]).timestamp
41                     velocity ← (position_new − position_old)/δ_t
42                     Calculate confidence
43                     if th^min_conf ≤ confidence then
44                         Create MO from timestamp, position_new, velocity and confidence
45                         Add MO to msgMOA
46             i ← j                                      ▷ Skip already-considered scan points
47         Publish msgMOA on moaStream if we found at least one object, otherwise discard msgMOA
48 end procedure
```

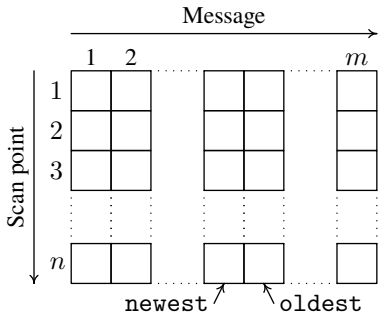


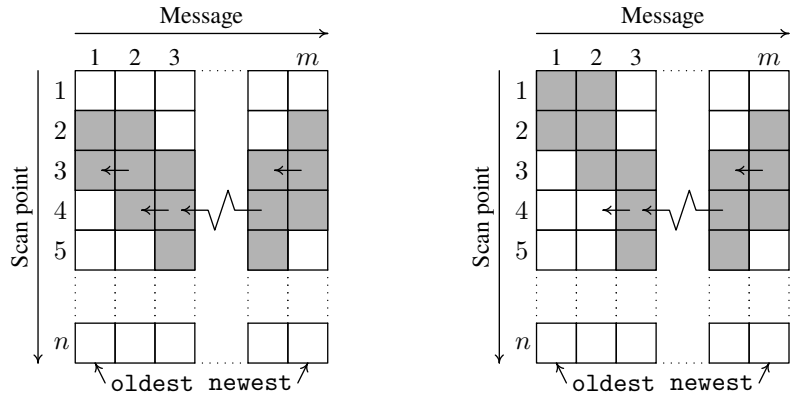
Fig. 3. The bank of m consecutive messages, each containing n scan points.

(rows 25–34). If the found object is constituted by less than $\texttt{th}^{\text{min}}_{\text{pts}}$ points, then the object is discarded and the search continues. Otherwise, the found object is further considered.

Since the angle, relative the sensor, for each scan point is known, it is straightforward to calculate the coordinates, in the frame of the sensor, of the closest point and the estimated center point of the found object. It is also straightforward, using the law of cosine, to calculate the seen width of the object.

The velocity of the found object is calculated based on how it has moved, relative the sensor (more on this in Sect. 4), from its position in the oldest scan data message in the bank to its position in the newest scan data, and on the duration of this movement. However, for this calculation to be possible, the object must exist in the oldest scan data. Therefore, we try to track the object through each scan data message stored in the bank, from the newest to the oldest (cf. Fig. 4). The basic assumption while tracking the object is that its center scan point in the newer data covers some part of the object in the older data. There are several unrelated reasons for not succeeding in tracking an object: the center scan point in the newer data tracks to a scan point in the older data whose range value lies outside the interval given by $[\texttt{th}^{\text{min}}_{\text{dist}}, \texttt{th}^{\text{max}}_{\text{dist}}]$; the center scan point in the newer data tracks to an object constituted by less than $\texttt{th}^{\text{min}}_{\text{pts}}$ scan points (given $\texttt{th}^{\text{max}}_{\delta_{\text{edge}}}$); the distance to the object (from the sensor) differs more than $\texttt{th}^{\text{max}}_{\delta_{\text{dist}}}$ between two consecutive scans; or the difference in seen width of the object between two consecutive scans is larger than $\texttt{th}^{\text{max}}_{\delta_{\text{width}}}$.

The size of the bank can, clearly, affect the accuracy of our method. For example, a relatively small size, in combination with a sensor which is producing scans with a relatively high frequency, means that $\texttt{position}_{\text{new}} - \texttt{position}_{\text{old}}$ and δ_t are, most likely, too small in relation to the noise in the data, to accurately estimate the current velocity of the object. On the other hand, a relatively large size, in combination with a sensor which is producing scans with a relatively low frequency, means that $\texttt{position}_{\text{new}} - \texttt{position}_{\text{old}}$ and δ_t, most likely, do not cover the instantaneous changes in movement well enough. For the latter case, it is easy to see that the estimated velocity of a constantly accelerating (or retarding) object could be very imprecise—the change in position between two consecutive scans is larger (or smaller) for the newer scans than for the older scans. Thus, to yield the best result, the bank size should be adapted to the scan rate of the



(a) Tracking success. **(b)** Tracking failure.

Fig. 4. Tracking an object through the bank—gray scan points indicate actual location.

sensor and the environmental context, if possible. (We, of course, also have the case where the sensor per se is not fast enough to satisfactory handle the given environmental context—e.g. if the environment contains objects moving very rapidly—but, this issue is very difficult to solve by means of adapting the bank size.)

If we are confident enough about our calculations, then the object is added to `msgMOA`, which is published on `moaStream`, provided that at least one object was found, after the entire newest scan data has been searched for objects. How the confidence value, `confidence`, should be calculated depends on the given context: how fast objects in the environment move, how fast our robot moves, with what rate the source produces scans, how large the bank is etc. This is further discussed in Sects. 4 and 5.

The details of a found object are stored as a `MO` object. `MO` should be viewed as containing four data fields: a timestamp, `timestamp`; a position, `position`; a velocity, `velocity`; and a confidence value, `confidence`. The `msgMOA` should be viewed as an array, list or the like, containing `MO` objects.

3 Handling Several Sources

As previously stated, the method, outlined in Algorithm 1, can receive input from a single source only. If we desire to use s sensory sources for finding moving objects, then one instance of the method must be applied for each source (cf. Fig. 5).

Whenever the method is instantiated for several sources, and we know how the sources are physically located in relation to each other, then we can determine whether an object seen by one source is the same object which is seen by another source. If several sources see the same object, then that object should be assigned a higher confidence value (cf. Fig. 5). This section presents a method for achieving this.

Our strategy is architecturally visualized in Fig. 5 and outlined in Algorithm 2. $\texttt{moaStream}_1, \ldots, \texttt{moaStream}_s$ are the streams on which `msgMOA` messages are received.

We keep a cache containing the latest message from each source. For each incoming `msgMOA`, and each `MO` it contains, a matching object is searched for in the cached message from all other sources. An object is considered to match another object if the two objects' timestamps, positions and velocities do not differ by more than $\mathtt{th}^{\max}_{\delta_t}$, $\mathtt{th}^{\max}_{\delta_{\text{pos}}}$ and $\mathtt{th}^{\max}_{\delta_{\text{vel}}}$, respectively. The considered object's confidence value is increased by the average confidence of all objects, seen by other sources, found to match it (but upper-bounded by 1). All objects in `msgMOA`, with possibly increased confidence values, are then further reported on `moaStream`.

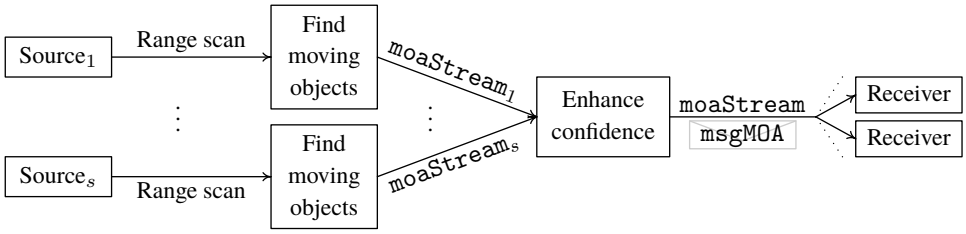


Fig. 5. Enhancing the confidence by using s sources.

Algorithm 2. Enhance confidence of `MO`

Input: $\mathtt{moaStream}_1, \ldots, \mathtt{moaStream}_s$, $\mathtt{th}^{\max}_{\delta_t}$, $\mathtt{th}^{\max}_{\delta_{\text{pos}}}$, $\mathtt{th}^{\max}_{\delta_{\text{vel}}}$
Output: `msgMOA` on `moaStream`

```
1  procedure
2      Create cache of s messages, one per source
3      loop
4          Receive msgMOA from moaStream_i, where i ∈ {1, ..., s}
5          Replace cached message for source i with msgMOA
6          for all MO in msgMOA do                                   ▷ Parallel operation
7              confidenceSum ← 0
8              matchingSources ← 0
9              for all sources, j, such that j ≠ i do                ▷ Parallel operation
10                 for all MO_j in j's cached message do             ▷ Parallel operation
11                     if |MO.timestamp − MO_j.timestamp| ≤ th^max_δt and
                           |MO.position − MO_j.position| ≤ th^max_δpos and
                           |MO.velocity − MO_j.velocity| ≤ th^max_δvel then
12                         confidenceSum ← confidenceSum + MO_j.confidence
13                         matchingSources ← matchingSources + 1
14                         break
15             if 0 < matchingSources then
16                 MO.confidence ← min(1, MO.confidence + confidenceSum/matchingSources)
17         Publish msgMOA (with possibly updated MOs) on moaStream
18 end procedure
```

4 Implementation

The confidence-enhancing method of Algorithm 2 has been implemented [13] in C++ as a ROS node [22]. Since ROS provides a communications layer, structured the way assumed by our algorithms, the implementation of the confidence enhancer node is very straightforward.

The functionality of the bank (i.e. storing a history of incoming range scan data, and deriving the position and velocity of objects found based on that data), as outlined in Algorithm 1, has been implemented [13] as a ROS C++ library. The ROS transformation system, `tf` [10], is used for transforming the object position and velocity from the sensor frame into the `base_link`, `odom` and `map` frames [18]. This automatically accounts for how the sensor has moved, as long as such information is available. The output messages (i.e. `msgMOA`) are of the type `MovingObjectArray.msg` [13]. This message type contains the name of the sending node and an array of `MovingObject.msg` [13] messages, where each such message hence corresponds to a `MO` object.

The bank can be told to publish messages meant for visualization in RViz [15]. We use several types of "visualization" messages: a `LaserScan` [2], which visualizes each scan point range, adapted using EMA, from the newest received message (the scan points of found objects have an intensity distinct from scan points not belonging to a found object); a `LaserScan`, which marks the point on each found object closest to the sensor; a `MarkerArray` [4], which adds an arrow (using a `Marker` [3] message) for each found object, where the tail of the arrow marks the position of the object and the direction and length of the arrow shows the velocity of the object; a `MarkerArray`, which adds a line (using a `Marker` message) for each found object, showing the change in position (from the oldest scan to the newest scan in the bank) for that object; and a `MarkerArray`, which adds a line (using a `Marker` message) for each found object, showing the width of the object (in the newest scan). The velocities can be shown in any of the four available frames but, the default frame is `map`.

The bank implementation provides some functionality for handling `LaserScan` or `PointCloud2` [5] input messages in a convenient manner (the user of the library can simply provide the bank with a message of any of these types and the data of that message is added to the bank automatically). An instance of the bank should, of course, only be used for a single source, sending only one of these message types, as discussed in Sects. 2 (cf. Fig. 2) and 3 (cf. Fig. 5).

The ranges in a `LaserScan` message can be handled without modification, since the bank expects a 2D polar structure. On the other hand, each point in a `PointCloud2` message must be extracted and projected, from a 3D volume onto a 2D plane, before it can be stored in the bank. The projection, in our implementation, is made from (x, y, z) volume coordinates to (x, y) plane coordinates. A 2D polar range scan structure (as expected by the bank) is achieved, by mapping each resulting (x, y) coordinate onto one or several (based on a given voxel grid leaf size) scan point(s) of the bank data structure. If several (x, y) coordinates map to the same scan point(s), then the chosen range value for that scan point is the distance to the coordinate closest to the sensor. All this, of course, makes the handling of `PointCloud2` messages more costly.

A sensor-interpreting application, using our bank library, is required to implement a function for calculating the confidence values for the found objects. The function should

take as input: the information derived for that object; the arguments to the bank (such as the bank size, thresholds etc.); the difference in time between the newest and oldest data in the bank; and whether transformations of the object position and velocity, from the sensor frame into the `base_link`, `odom` and `map` frames, were possible. This allows for the user to make a context-based implementation of the confidence calculation. Note that all found objects can be reported, using the calculation `confidence` $\leftarrow 1$.

Along with the bank library, we provide [13] two sensor-interpreting executable ROS nodes, one for handling `LaserScan` message streams and one for handling `PointCloud2` message streams, which use the bank for finding moving objects. Each of these two nodes can take any data stream of its handled type as input which allows for them to be used on data streams from a large variety of sensors. The size of the bank (i.e. the number of scan messages it stores) can be automatically calculated by the nodes, based on measuring the rate of received scan messages and a desired time interval which the messages in the bank should cover.

We also provide [13] several ROS launch files for running our sensor-interpreting nodes. The sensor-interpreting nodes can be launched to run in a live setting, with actual physical sensors providing the data streams. Or, they can be launched in a simulated setting, with the sensor data coming from a provided recording of a 360° LIDAR and a depth camera (more about these sensors in the next section).

5 Experiment Setup & Verification

The moving-object-finding algorithm and the confidence-enhancing algorithm have been evaluated on a research laptop receiving input from a Slamtec RPLIDAR A2M8 [6] 360° LIDAR (providing a `LaserScan` [2] message stream) and an Intel RealSense D435 [1] depth camera (providing a `PointCloud2` [5] message stream) over USB 3.0 ports. The sensors were physically mounted close together with overlapping fields of view. The laptop had an Intel i7 quad-core processor clocked at 2.9 GHz and 32 GB of DDR4 (2.4 GHz) Random Access Memory (RAM).

The evaluation environment consisted of an outdoor area with varying sunlight conditions. An object, about 70 cm wide and moving with speeds between $0\,\mathrm{m\,s^{-1}}$ and $2\,\mathrm{m\,s^{-1}}$, was used for detection purposes.

Experiments showed that when the bank covered a time period of 0.5 s, good accuracy in the position and velocity calculations resulted. The bank size was therefore automatically calculated by the sensor-interpreting nodes to cover a time period of 0.5 s. This resulted in a bank size of 6 messages for the `LaserScan` interpreter and 10 messages for the `PointCloud2` interpreter (the rate of received messages on the two streams were about 11 Hz and 18 Hz, the latter after voxel-grid-filtering, respectively).

The points of the `PointCloud2` messages were mapped onto a 2D polar range scan structure with a field of view of 180° and a total of 360 scan points.

The confidence value for each found object, `MO`, was derived using Eq. 1. Here, the resulting confidence value is cropped to lie within $[0, 1]$; α is the degree of EMA weighting decrease; $expr_1$? $expr_2$: $expr_3$ is $expr_2$ if $expr_1$, and $expr_3$ otherwise; `transformSucces` is a boolean value representing whether we could transform `MO`'s position and velocity from the sensor frame into the `base_link`, `odom` and `map` frames [18];

δ_t is the difference between the timestamps of the newest and oldest scans in the bank; `width(M0)` is the current seen width of `M0` and `widthOld(M0)` is the width of `M0` as found in the oldest scan in the bank; and `baseConfidence` is a measure for how well we trust the given sensor (we used 0.3 for the LIDAR and 0.4 for the camera).

$$\begin{aligned}\max(0, \min(1, \alpha \times ((&\texttt{transformSucces} \; ? \; 0.5 \; : \; 0.0) \\ &+ \tfrac{-10}{3}(\delta_t - 0.2)(\delta_t - 0.8) \\ &- 5.0 \times |\texttt{width(M0)} - \texttt{widthOld(M0)}| \\ &+ \texttt{baseConfidence})))\end{aligned} \tag{1}$$

The expression involving δ_t in Eq. 1 is visualized in Fig. 6 and results in a maximum confidence increase when $\delta_t = 0.5\,\text{s}$, which is the time period that the bank is expected to cover. If $\delta_t < 0.2\,\text{s}$ or $0.8\,\text{s} < \delta_t$, then the resulting confidence value is decreased. Thus, successful transformations of the object's position and velocity, a well-adapted bank size in combination with a well-functioning data stream from a trustworthy sensory source without much high-frequent noise, and a similar old and current object width result in high confidence.

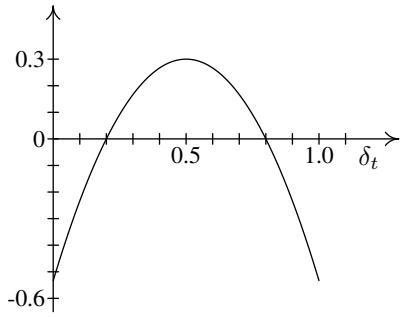


Fig. 6. The effect of δ_t on the confidence: $\frac{-10}{3}(\delta_t - 0.2)(\delta_t - 0.8)$

To verify the correctness of Algorithms 1 and 2, we logged all found objects and verified the correctness of the derived position and velocity in the different frames, for both sensors. We also used RViz [15] to visually verify the correctness of the derived information. The used EMA weighting decrease factor and thresholds for the two different cases of using Algorithm 1 are presented in Table 1 and the used thresholds for Algorithm 2 are presented in Table 2.

All experiments showed good accuracy in the position and velocity calculations. The Valgrind [20] heap profiler, Massif, was used for some long-running experiments to successfully verify the lack of memory leaks in our implementation. It is hence our opinion that the algorithms and implementations can be considered sufficiently and successfully verified.

Table 1. Used EMA weighting decrease factor and thresholds for Algorithm 1.

`LaserScan`		`PointCloud2`	
α	1.0	α	1.0
$\mathtt{th}^{\max}_{\delta_{\text{edge}}}$	0.2 m	$\mathtt{th}^{\max}_{\delta_{\text{edge}}}$	0.15 m
$\mathtt{th}^{\min}_{\text{pts}}$	4 points	$\mathtt{th}^{\min}_{\text{pts}}$	4 points
$\mathtt{th}^{\min}_{\text{dist}}$	0.01 m	$\mathtt{th}^{\min}_{\text{dist}}$	0.01 m
$\mathtt{th}^{\max}_{\text{dist}}$	8 m	$\mathtt{th}^{\max}_{\text{dist}}$	6.5 m
$\mathtt{th}^{\max}_{\delta_{\text{dist}}}$	0.4 m	$\mathtt{th}^{\max}_{\delta_{\text{dist}}}$	0.4 m
$\mathtt{th}^{\max}_{\delta_{\text{width}}}$	20 points	$\mathtt{th}^{\max}_{\delta_{\text{width}}}$	50 points
$\mathtt{th}^{\min}_{\text{conf}}$	0.5	$\mathtt{th}^{\min}_{\text{conf}}$	0.5

Table 2. Used thresholds for Algorithm 2.

Confidence Enhancer	
$\mathtt{th}^{\max}_{\delta_t}$	0.1 s
$\mathtt{th}^{\max}_{\delta_{\text{pos}}}$	0.1 m
$\mathtt{th}^{\max}_{\delta_{\text{vel}}}$	0.1 m s^{-1}

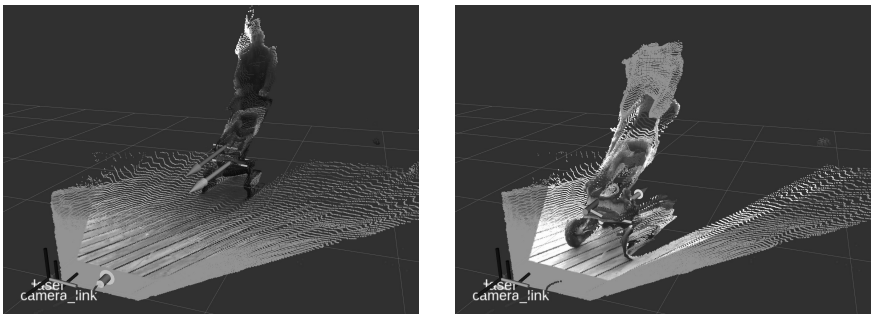
6 Evaluation

To evaluate the presented method, we have measured several quantities and qualities. The first such is the resident set size, i.e. memory usage (the RAM page size was 4 kB). For the `LaserScan`-interpreting node, this was constant and showed 11 896 kB in our experiments. For the `PointCloud2`-interpreting node, this varied, depending on the size of the incoming messages (voxel-grid-filtered point clouds vary in size depending on how close objects are to the sensor) but never exceeded 40 048 kB in our experiments. For the confidence-enhancing node, this was constant and showed 10 436 kB in our experiments. Thus, we find the memory footprints of these different nodes to be of reasonable size and provide a good opportunity for achieving a high-performing system.

The second measured quantity is the average CPU usage. For the `LaserScan`-interpreting node, this was about 1.3%. For the `PointCloud2`-interpreting node, this was about 25%. For the confidence-enhancing node, this was about 0.2%. The CPU usage is of course dependent on the rate of the incoming messages and the complexity of the presented algorithms. However, the `LaserScan` messages arrive at a rate of about 11 Hz (see the previous section) and are of constant size, so the CPU usage required by the implementation of Algorithm 1 to find objects seems to be quite low. The higher CPU usage visible for the `PointCloud2` part thus seems to stem from reading and mapping the points in each received cloud. The requirements of the confidence-enhancing node is clearly dependent on how many instants of Algorithm 1 are sending moving object information to it, and on how many objects each source has found. In our experiments, the requirements have thus shown to be very low while successfully handling the two incoming message streams. There is also a risk of the confidence-enhancing node

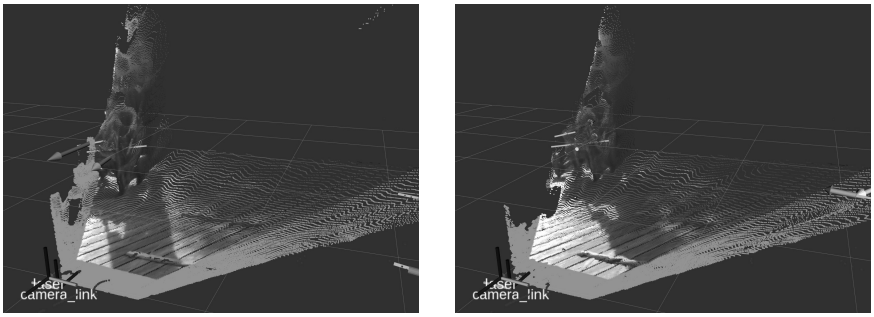
becoming a bottleneck, due to the architecture of the system (cf. Fig. 5). For our setup, the experiments show that this is not the case, however.

The third measured quantity is the output (`MovingObjectArray`) message rates. For the `LaserScan`-interpreting node, this was about 11 Hz. For the `PointCloud2`-interpreting node, this was about 17 Hz. Comparing these rates to the rates of the incoming messages, it is obvious that the `LaserScan`-interpreting node seems to be able to handle the incoming message stream without having to drop any messages. For the `PointCloud2`-interpreting node, it seems that a few incoming messages have to be dropped. This conclusion is rather vague, however, because of the constantly (to a small extent) varying message rates. For the confidence-enhancing node, the rate of the outgoing message stream seemed to be equal to the total rate of the incoming messages. Thus, it seems like the confidence-enhancing node was able to handle the incoming message streams without having to drop any messages.



(a) Toward. **(b)** Away from.

Fig. 7. Object moving toward and away from the sensors.



(a) Meeting. **(b)** Overtaking.

Fig. 8. Object meeting and overtaking the sensors on their left.

The fourth measured quality is how well the system is able to correctly calculate the position and velocity of objects moving in different directions, relative the sensors. The results of the position and velocity calculations (arrows; as discussed in Sect. 4), along with the width of the detected object (green/gray lines), are visualized in Figs. 7–15 for an occurrence within the following cases. Note that the point cloud sometimes covers (parts of) the arrows and lines in the figures. For clarity, only the output of the sensor-interpreting nodes is shown in the figures. For the first five cases, the sensors remain in a static position (i.e. do not move).

The first case is when the object is moving toward/away from the sensors along their depth-viewing axes (Fig. 7), and the second case is when the object is meeting/overtaking the sensors on one side (Fig. 8).

The third case is when the object is moving across the field of view of the sensors (Figs. 9–12). This case poses a challenge, because the object is moving across the sensors'/banks' scan rays, not along them, which makes tracking the object from the newest scan in the bank to the oldest more difficult. When the object is crossing the field of view of the sensors at a distance of 1 m with a speed of $2\,\mathrm{m\,s^{-1}}$, the system mostly fails to correctly calculate its position and velocity. For speeds around $1\,\mathrm{m\,s^{-1}}$, the `PointCloud2`-interpreting node successfully calculates the position and velocity at a distance of 1 m, though (cf. Fig. 9). For speeds around $0.5\,\mathrm{m\,s^{-1}}$ and below, both interpreting nodes successfully calculate the position and velocity at a distance of 1 m. When the object is crossing the field of view of the sensors at a distance of 2–4 m with a speed of $2\,\mathrm{m\,s^{-1}}$, the `PointCloud2`-interpreting node is able to derive the position and velocity of the object, while the `LaserScan`-interpreting node mostly fails to do so. For speeds around $1\,\mathrm{m\,s^{-1}}$ and below and at distances of 2–4 m, the `LaserScan`-interpreting node is able to successfully calculate the position and velocity of the object, though. Thus, we have established somewhat of a soft limit for at what distances and speeds the sensor interpreters fail to correctly calculate the position and speeds of the object because the rate at which the sensor produces scans is too low.

The fourth case is when the object is moving diagonally across the field of view of the sensors (Fig. 13), while the fifth case is when the object is moving in an arc in front of the sensors (Fig. 14).

The sixth case is when the *sensors are moving* forward while the object is moving perpendicularly across their path (Fig. 15).

All cases (except for the mentioned exceptions) show good accuracy in the calculation of the position and velocity of the object, and the rates of the `MovingObjectArray` output message streams are very stable.

The fifth measured quality is how well instantaneous changes in how the object moves (e.g. rapidly stopping/starting or changing direction) are interpreted. Both the `LaserScan`- and `PointCloud2`-interpreting nodes handle the instantaneous changes very well when the object is moving at speeds up to about $1\,\mathrm{m\,s^{-1}}$. For speeds higher than this, the interpretation of the instantaneous changes becomes more difficult, but the result can still be seen as quite satisfactory. The best result was achieved for speeds between 0.5 and $1\,\mathrm{m\,s^{-1}}$. The confidence-enhancing node could successfully handle its two incoming data streams very well for all speeds and movement patterns.

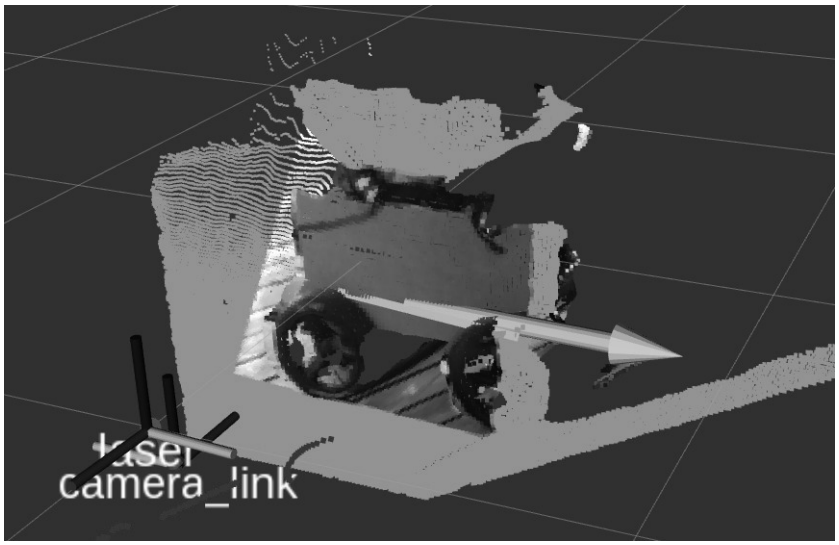
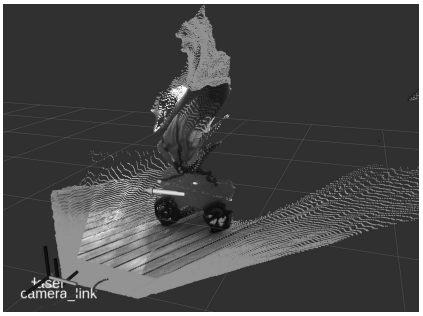
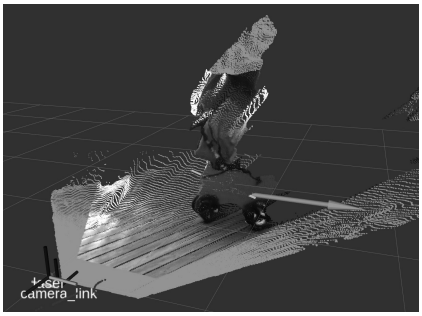


Fig. 9. Object crossing the sensors' field of view with a speed of $1\,\mathrm{m\,s^{-1}}$ at a distance of 1 m.

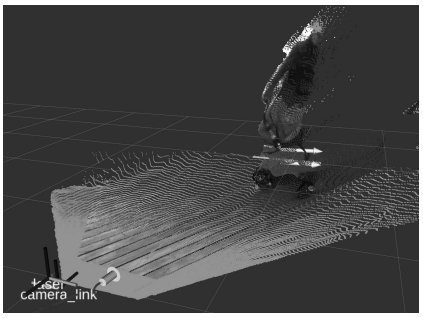


(a) Crossing with a speed of $0.5\,\mathrm{m\,s^{-1}}$.

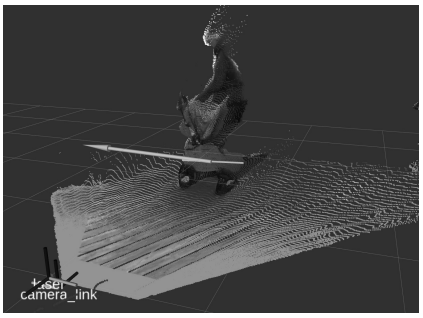


(b) Crossing with a speed of $2\,\mathrm{m\,s^{-1}}$.

Fig. 10. Object crossing the field of view of the sensors at a distance of 2 m.

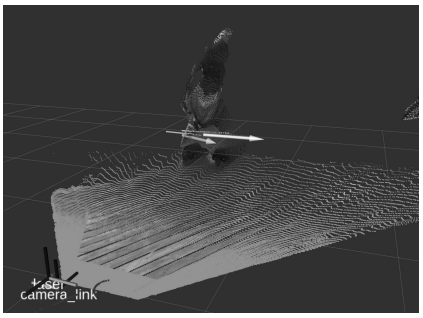


(a) Crossing with a speed of $0.5\,\mathrm{m\,s^{-1}}$.

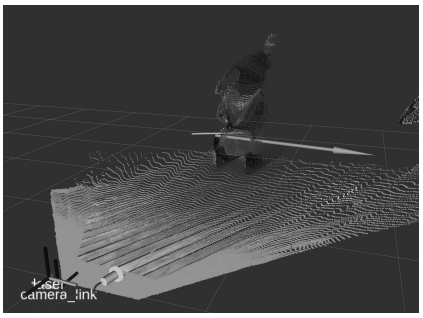


(b) Crossing with a speed of $2\,\mathrm{m\,s^{-1}}$.

Fig. 11. Object crossing the field of view of the sensors at a distance of 3 m.

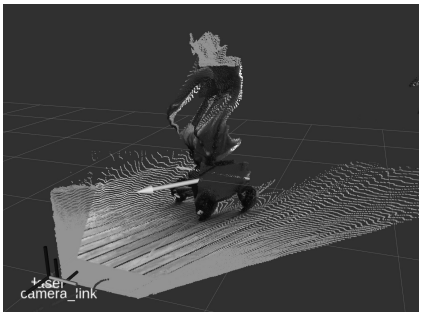


(a) Crossing with a speed of $0.5\,\mathrm{m\,s^{-1}}$.

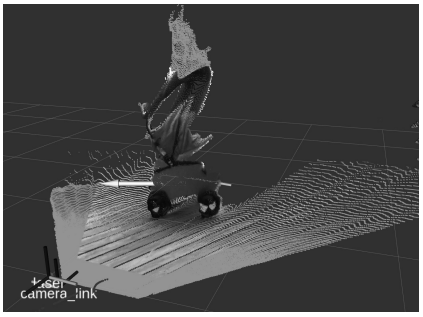


(b) Crossing with a speed of $2\,\mathrm{m\,s^{-1}}$.

Fig. 12. Object crossing the field of view of the sensors at a distance of 4 m.

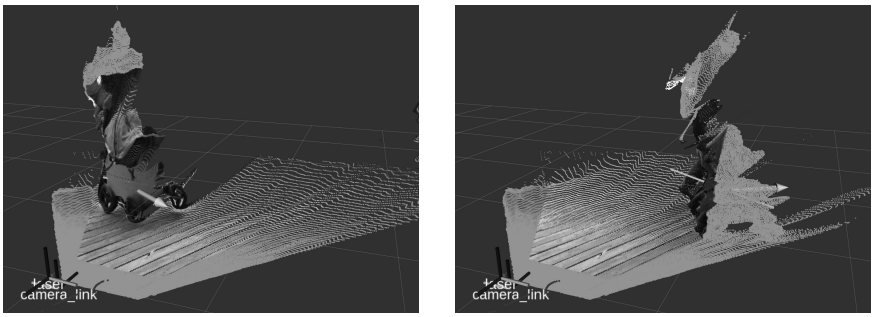


(a) 30°.



(b) 60°.

Fig. 13. Object moving diagonally across the field of view of the sensors.



(a) First position. **(b)** Second position

Fig. 14. Object moving in a convex arc in front of the sensors.

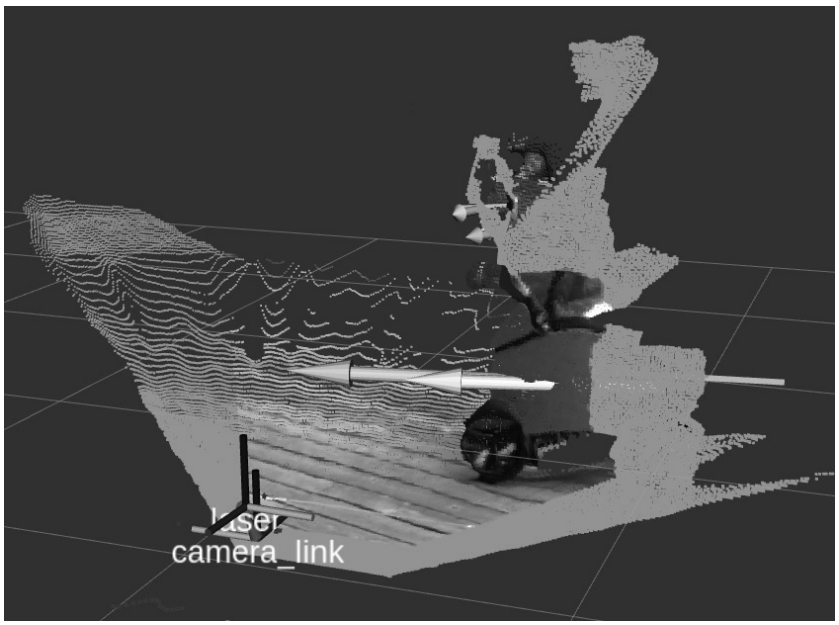


Fig. 15. Sensors moving forward at $0.5\,\mathrm{m\,s^{-1}}$ and object crossing their path at $1\,\mathrm{m\,s^{-1}}$.

The overall results are hence: good accuracy in the position and velocity calculations for reasonable object speeds at reasonable object distances; small memory footprints; and low CPU usage requirements for the implementations of Algorithms 1 and 2.

7 Related Work

To the best of the author's knowledge, there is no other work trying to achieve the exact same results as those presented herein. However, this work has some similarities to work on localization and map building [8,12,17] (e.g. simultaneous localization and mapping—SLAM [7,9,17]) and on detecting and tracking moving objects [14,16,19,21,23,25,26].

Many localization and map-building solutions rely on feature-to-feature, point-to-feature or point-to-point matching techniques for determining changes in consecutive environment scans. Our method could be said to rely on a very simple feature-to-feature, or rather segment-to-segment, technique for tracking an object through the bank. The flat structure of the bank, only consisting of range readings at known directions, makes our solution quite effective and efficient.

We do not focus on tracking the detected moving objects in the environment, but only provide an instantaneous view of the surroundings. One could argue that the tracking instead will be done "automatically" by the navigation system when fed with information by our system. Like most approaches to tracking objects [26], our approach works the best for rather smooth motions of the objects in the environment (i.e. an object is assumed not to move too far between two consecutive sensor readings). Some techniques (applicable to for example street surveillance applications) rely on a static sensor with a known default/background image input, then the deviations from this known default image and the current input image represent objects.

We use partly the same strategies as those presented by Diosi and Kleeman in 2007 [8] where they try to estimate the motion of a laser range scanner using its range readings (thus, they are primarily targeting localization/odometry- and mapping-related problems but the technique could also be applied for tracking moving objects). Like Diosi and Kleeman, we too work in a polar coordinate system and two consecutive scan points are considered to belong to the same object if their ranges do not differ more than a specific threshold value. However, we do not group points whose ranges differ more than this threshold, but which lie on a straight line, into objects. The latter grouping of scan points is a very effective strategy when trying to detect walls [8] and other large plane-shaped surfaces. We do not specialize on detecting walls, however, but try to be quite general in our object detection and focus on moving objects. Therefore, we do not include such a specialized condition.

8 Discussion, Conclusions & Future Work

In this paper, we have presented an efficient and effective method for detecting moving objects and deriving their positions and velocities, based on 2D polar range scans of the environment. The method has been implemented as a C++ ROS library along with two executable nodes using the library/method. One of the nodes takes a (2D) `LaserScan` [2]

message stream as input, while the other node takes a (3D) `PointCloud2` [5] message stream as input. Thus, great flexibility in which type of sensor can be used to feed our method with a data stream is provided.

We have also provided a very efficient method for increasing the confidence value for objects which are seen/detected by several sensors. Note that this method can be used to easily perform sensor fusion with the goal of providing information about moving objects (since this method only relies on the output of the first method, and the first method can be applied to a great variety of sensor types).

In this paper, the presented methods were evaluated using a 360° LIDAR from Slamtec (RPLIDAR A2M8) [6] and a depth camera from Intel (RealSense D435) [1]. It was found that the simplicity of our approaches resulted in good accuracy in the position and velocity calculations for reasonable object speeds at reasonable object distances, and small memory footprints and low CPU usage requirements for the implementations of the presented methods.

The complexity of Algorithm 1 is dependent on the number of messages that are stored in the bank, m, and on the size of each such message, n. The complexity of caching the range values of an incoming message, and performing EMA on them, is $O(n)$. The object detection (rows 11–34) loops through the range values of the newly arrived message only once. Thus, the object detection complexity is $O(n)$. Tracking a detected object through the messages in the bank (row 37) is of $O(mn)$ in the worst case (i.e. when all scan points constitute the object in all cached messages). The rest of the operations are of constant complexity. Thus, the total complexity is $O(mn^2)$ in the worst case. However, setting the thresholds used by the algorithm appropriately will drastically reduce this worst-case complexity, like our evaluation experiments show.

The complexity of Algorithm 2 is dependent on the number of sources, s, and on the number of objects detected by each source, which is n_i, where $i \in \{1, \ldots, s\}$, in the worst case. When receiving a message from source i, each of its detected objects is compared to each object detected by all other sources. Thus, the complexity is $O(n_i \prod_{j \in \{1,\ldots,s\} \setminus \{i\}} n_j) = O(\prod_{j=1}^{s} n_j)$ in the worst case. For our evaluation setup, the worst-case complexity is hence $O(n_1 n_2)$. Again, setting the thresholds used by the algorithms appropriately will drastically reduce this worst-case complexity, since then it will not be that each scan point for each source represents an object. This is evident in our evaluation experiments.

We consider making parallel implementations of some strategically chosen parts of the algorithms and compare their performance to the sequential versions presented in this paper. It should be noted that there are several pitfalls to consider, though. Since the number of scan points, n, in an incoming polar scan message is typically quite small, it is expected that the overhead from using several threads might not be covered by the parallel computations. It should be noted that detecting objects and tracking them through the bank are inherently sequential operations in our approach. The detection could be done in parallel by several threads, but that would require special handling of the threads' respective search boundaries. This approach is not expected to out-perform the sequential approach for realistic values of n. The objects could be tracked through the bank in parallel if changing the structure of the algorithm to first identify all objects in an incoming message and then tracking them as a subsequent operation.

Since the number of scan points might be too small for an efficient parallel implementation of Algorithm 1, it should also be that the number of found objects in an incoming `msgMOA` to the confidence-enhancing node is too small as well. In case we have a very large number of sources, though, then it might be efficient to compare an incoming message to the cached messages in parallel.

Acknowledgements

The research presented herein was funded by the Knowledge Foundation through the research profile "DPAC - Dependable Platforms for Autonomous systems and Control."

References

1. Intel® RealSense™ Depth Camera D435, https://click.intel.com/intelr-realsensetm-depth-camera-d435.html (Accessed September 29, 2018)
2. LaserScan message, http://docs.ros.org/api/sensor_msgs/html/msg/LaserScan.html (Accessed September 29, 2018)
3. Marker message, http://docs.ros.org/api/visualization_msgs/html/msg/Marker.html (Accessed September 29, 2018)
4. MarkerArray message, http://docs.ros.org/api/visualization_msgs/html/msg/MarkerArray.html (Accessed September 29, 2018)
5. PointCloud2 message, http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html (Accessed September 29, 2018)
6. RPLIDAR A2, https://www.slamtec.com/en/Lidar/A2 (Accessed September 29, 2018)
7. Bailey, T., Durrant-Whyte, H.: Simultaneous localization and mapping (SLAM): Part II. IEEE Robotics & Automation Magazine 13(3), 108–117 (2006)
8. Diosi, A., Kleeman, L.: Fast laser scan matching using polar coordinates. Intl. Journal of Robotics Research 26(10), 1125–1153 (2007)
9. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: Part I. IEEE Robotics & Automation magazine 13(2), 99–110 (2006)
10. Foote, T.: tf: The transform library. In: Intl. Conf. on Technologies for Practical Robot Applications (TePRA). pp. 1–6. IEEE (2013)
11. Foote, T., Purvis, M.: REP 103: Standard Units of Measure and Coordinate Conventions (2010), http://www.ros.org/reps/rep-0103.html, http://www.ros.org/reps/rep-0103.html (Accessed September 29, 2018)
12. Gonzalez, J., Gutierrez, R.: Direct motion estimation from a range scan sequence. Journal of Robotic Systems 16(2), 73–80 (1999)
13. Gustavsson, A.: Find moving objects repository on github, https://github.com/andreasgustavsson/find_moving_objects (Accessed November 9, 2018)
14. Hue, C., Le Cadre, J.P., Pérez, P.: Tracking multiple objects with particle filtering. IEEE transactions on aerospace and electronic systems 38(3), 791–812 (2002)
15. Kam, H.R., Lee, S.H., Park, T., Kim, C.H.: Rviz: a toolkit for real domain data visualization. Telecommunication Systems 60(2), 337–345 (Oct 2015), https://doi.org/10.1007/s11235-015-0034-5
16. Khan, Z., Balch, T., Dellaert, F.: Mcmc-based particle filtering for tracking a variable number of interacting targets. IEEE transactions on pattern analysis and machine intelligence 27(11), 1805–1819 (2005)

17. Kohlbrecher, S., Von Stryk, O., Meyer, J., Klingauf, U.: A flexible and scalable SLAM system with full 3D motion estimation. In: 2011 IEEE Intl. Symposium on Safety, Security, and Rescue Robotics (SSRR). pp. 155–160. IEEE (2011)
18. Meeussen, W.: REP 105: Coordinate Frames for Mobile Platforms (2010), http://www.ros.org/reps/rep-0105.html, http://www.ros.org/reps/rep-0105.html (Accessed September 29, 2018)
19. Montesano, L., Minguez, J., Montano, L.: Modeling the static and the dynamic parts of the environment to improve sensor-based navigation. In: Proc. Intl. Conf. on Robotics and Automation (ICRA). pp. 4556–4562. IEEE (2005)
20. Nethercote, N., Seward, J.: Valgrind: a framework for heavyweight dynamic binary instrumentation. In: ACM Sigplan notices. vol. 42, pp. 89–100. ACM (2007)
21. Pu, S., Rutzinger, M., Vosselman, G., Elberink, S.O.: Recognizing basic structures from mobile laser scanning data for road inventory studies. ISPRS Journal of Photogrammetry and Remote Sensing 66(6), S28–S39 (2011)
22. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. vol. 3, p. 5. Kobe, Japan (2009)
23. Schulz, D., Burgard, W., Fox, D., Cremers, A.B.: Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In: Proc. Intl. Conf. on Robotics and Automation (ICRA). vol. 2, pp. 1665–1670. IEEE (2001)
24. Trinh, L., Ekström, M., Çürüklü, B.: Toward shared working space of human and robotic agents through dipole flow field for dependable path planning. Frontiers in Neurorobotics 1, 1–24 (June 2018), http://www.es.mdh.se/publications/5128-
25. Wang, C.C., Thorpe, C., Suppe, A.: Ladar-based detection and tracking of moving objects from a ground vehicle at high speeds. In: ProcİntelligentVehicles Symposium. pp. 416–421. IEEE (2003)
26. Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. ACM Computing Surveys (CSUR) 38(4) (Dec 2006), http://doi.acm.org.ep.bib.mdh.se/10.1145/1177352.1177355