An Easy-to-Use System for Tracking Robotic Platforms Using Time-of-Flight Sensors in Lab Environments

André Kirsch and Jan Rexilius
Bielefeld University of Applied Sciences and Arts, 32427 Minden, Germany
{*andre.kirsch, jan.rexilius*}*@hsbi.de*

Keywords: Tracking, Robot, Drone, MAV, External, Time-of-Flight, LiDAR.

Abstract: The acquisition of accurate tracking data is a common problem in scientific research. When developing new algorithms and AI networks for the localization and navigation of mobile robots and MAVs, they need to be evaluated against true observations. Off-the-shelf systems for capturing ground truth data often come at a high cost, since they typically include multiple expensive sensors and require a special setup. We see the need for a simpler solution and propose an easy-to-use system for small scale tracking data acquisition in research environments using a single or multiple sensors with possibly already available hardware. The system is able to track mobile robots moving on the ground, as well as MAVs that are flying through the room. Our solution works with point clouds and allows the use of Time-of-Flight based sensors like LiDAR. The results show that the accuracy of our system is sufficient to use as ground truth data with a low-centimeter mean error.

1 INTRODUCTION

Ground truth data is an important requirement to correctly assess the accuracy of navigation and localization algorithms and is also used for training reinforcement learning networks for mobile robots and micro aerial vehicles (MAVs), hereafter summarized as robotic platforms. The data contains positional information and sometimes also information about the orientation of the robotic platform. But capturing valid tracking data typically requires some sort of motion capturing system like OptiTrack[1], which come at a high cost. There are other cheaper alternatives available, each having different advantages and disadvantages, like the Lighthouse positioning system (Taffanel et al., 2021) and Ultra-Wideband (UWB) (Shule et al., 2020), which require hardware mounted on the tracked object.

We focus on Time-of-Flight (ToF) based technologies like LiDAR, as they do not require any lighting in the environment and can track unknown objects that have no tags or special hardware attached to them. Time-of-Flight is a technology that computes distance based on how much time light or sound needs to travel until it hits something and bounces back. For light, a typical wavelength is infrared. It is often either emitted using LEDs for low-range ToF devices or using a laser in case of LiDAR. Sending out light waves as a laser has the advantage of the light reaching farther distances and therefore covering a larger area.

In this paper, we present a system for tracking robotic platforms using Time-of-Flight for tracking data acquisition, as shown in figure 1. The system focuses on lab environments and can be used with different types of Time-of-Flight-based sensors that capture point clouds. Its main goal is to make capturing position data more accessible since it does not rely on a specific sensor. This enables the use of sensors already available to a researcher, which might have become unused after finishing an earlier research project. This omits the need to invest into a costly motion capturing system. The proposed system should be easy to install and already be useable with a single sensor, while being scalable to multi sensor setups. Multi sensor setups can increase the recording area or capture the scene from a different angle. Please note, that the tracking accuracy, frequency, and range highly depend on the selected sensor. Furthermore, there is no need to install additional hardware on the robotic platform when using our system.

2 RELATED WORK

Camera-based motion capturing systems are the gold standard for tracking objects in a confined space and are often used to record ground truth data due to their

[1]https://optitrack.com/

Kirsch, A. and Rexilius, J.
An Easy-to-Use System for Tracking Robotic Platforms Using Time-of-Flight Sensors in Lab Environments.
DOI: 10.5220/0013110500003905

In *Proceedings of the 14th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2025)*, pages 571-578
ISBN: 978-989-758-730-6; ISSN: 2184-4313

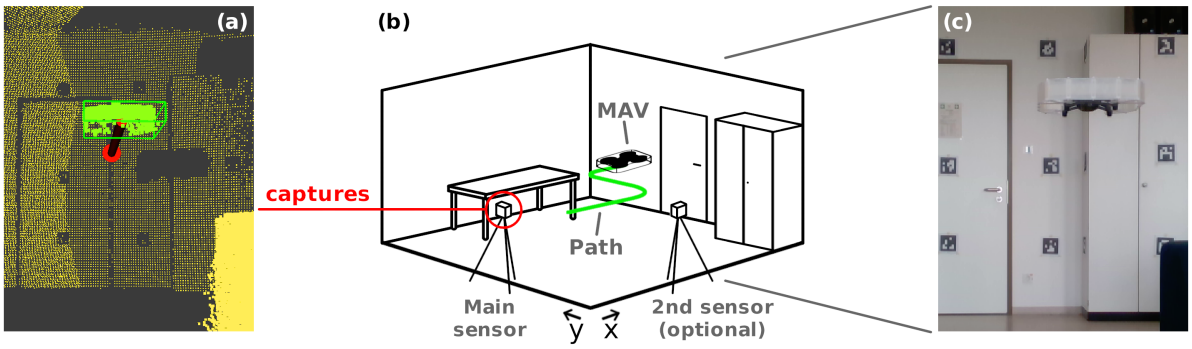

Figure 1: Visualization of the setup of the proposed system. Subfigure (b) shows the recording area with two installed sensors. At least one sensor is required. Additional sensors can be used to either increase the size of the recording area or to capture the recording area from a different angle, as shown in the image. The captured data is visualized in subfigure (a), where our system is tracking the detected MAV. Subfigure (c) shows a real-life view of a part of the recording area captured from the view of the main sensor.

millimeter-level accuracy, as shown in (Furtado et al., 2019). Popular systems include OptiTrack, Vicon[2] and Qualysis[3], which come at a high cost and require a complex setup of multiple sensors in a designated area. They either use active or passive markers attached to the tracked object that emit or reflect infrared (IR) light, respectively. This light is captured by multiple special cameras. Through triangulation of the captured data, it is possible to calculate the position of the markers in 3D space. A cheaper open-source alternative that uses markers is Easy RoCap (Wang et al., 2024), which is specifically designed for tracking mobile robots and MAVs. It uses infrared sensors from multiple Intel RealSense depth cameras. There are also commercial markerless motion capturing systems but these mainly focus on the tracking of humans.

Another IR-based system for tracking MAVs is the Lighthouse Positioning system (Taffanel et al., 2021). Multiple base stations emit IR light through moving sweep planes. This light is captured by multiple IR receivers mounted on the tracked MAV. The position is calculated on-board the MAV. With lower velocity and in a confined space, this method can reach accuracies of less than a centimeter. Ultra-Wideband is another alternative, requiring active components on the tracked hardware. As the name suggests, it utilizes radio technology to determine the location of tracked objects. An outline of this technology is given by (Shule et al., 2020). As we want the location estimation to be computed independently of the tracked object, we focus on a different method. We use ToF sensors that are part of the environment and capture data without communication with the tracked object required.

[2]https://www.vicon.com/

[3]https://www.qualisys.com/

2.1 Time-of-Flight-Based Methods

Compared to the aforementioned methods, Time-of-Flight and especially LiDAR are used in a variety of use cases. Its advantage over other approaches is that it does not require any active or passive hardware mounted on the tracked object. Therefore, it is a preferred technology for tracking unknown objects and has formed the research area of 3D single object tracking (3DSOC), including the pioneering works of SC3D (Giancola et al., 2019) and P2B (Qi et al., 2020). SC3D uses a Siamese tracker, while P2B leverages PointNet++ (Qi et al., 2017). More recent solutions to the problem include PTTR and PTTR++ (Zhou et al., 2022), as well as P2P (Nie et al., 2024), which also use some sort of AI network. The networks are commonly evaluated and compared using outdoor LiDAR sequences of a driving car, as in the KITTI dataset (Geiger et al., 2013). Object classes therefore include different types of cars, pedestrians, and cyclists. None of the mentioned AI-based research has been designed and tested with robotic platforms in mind, which in our case are typically smaller compared to their evaluated object classes.

The research on tracking mobile robots is much more sparse. This might be due to the large variety in forms and sizes, but also because swarms of mobile robots are typically connected through a network and can exchange their position information. (Yilmaz and Bayindir, 2022) and (Waşik et al., 2015) use a 2D LiDAR sensor to find mobile robots of the same type as the carrier of the sensor in the environment, also known as kin detection. (Pleterski et al., 2023) do kin detection for miniature mobile robots using an ultralow-resolution ToF sensor. They train a CNN on the depth images to determine if a mobile robot is present in the image.

With the recent publications of the LiDAR-based

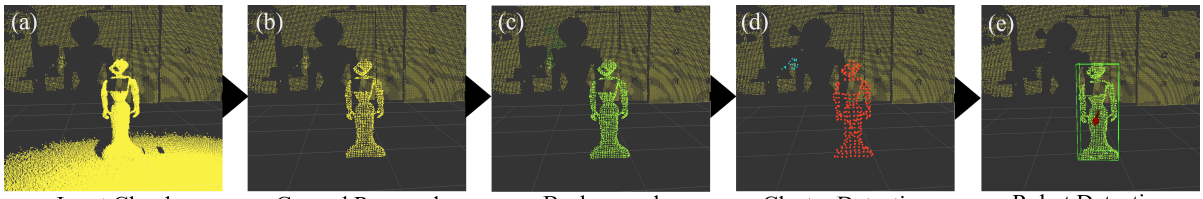

Figure 2: Overview of the proposed pipeline using the Pepper mobile robot as an example. An input cloud is shown in (a). The other subfigures (b)-(e) show the different stages of the pipeline as described in the sections 3.2-3.5. For visualization reasons, the background is shown in all images in yellow. Note that we downsampled the point cloud for performance reasons.

MAV datasets by (Catalano et al., 2023b) for general MAV detection and tracking, and by (Yuan et al., 2024) for drone threat detection, there is a stronger focus on detection and tracking of drones in point clouds compared to mobile robots. Compared to 3DSOC, both AI-based and traditional methods are researched to solve this problem. Initial groundwork has been done by (Dogru and Marques, 2022) and (Wang et al., 2021). Other traditional methods include works done by (Qingqing et al., 2021) and (Catalano et al., 2023a), who integrate multiple scans using different frequencies to improve accuracy, as well as (Gazdag et al., 2024), who use a particle filter for tracking and utilize the scanning pattern of the LiDAR sensor. For AI-based methods, (Sier et al., 2023) fuse 2D signal image provided by the sensor and point clouds to detect MAVs. Point clouds are used in a consecutive step to extract the MAV pose. (Chen et al., 2024) detect MAVs in RGB image data and again use point clouds for position estimation. The tracking of MAVs by (Deng et al., 2024) is primarily based on LiDAR captures and incorporates Radar data. MAVs are detected through a combination of an attention-based LSTM, a PointNet-based module, and an MLP. Since detection and tracking of robotic platforms using AI-based methods is still in its early stages and often merges different sensor types, we focus on traditional approaches in our implementation.

3 CONCEPT

This paper proposes a system for tracking and detecting mobile robots and MAVs in lab environments. The main use case of the system is to capture position data of a specified robotic platform. The user provides a captured or a live stream of point clouds to the system. Since the system is implemented using the Robot Operating System (ROS), different sensors can be used directly. Compared to other solutions, our system does not rely on specific hardware and can use any type of Time-of-Flight based sensor available to the user. Please note that the accuracy and frequency heavily depend on the selected sensor. It is also possible to combine the recordings of multiple sensors and feed the result into the system. Furthermore, the user only needs to specify the dimensions of robotic platforms to track. The system does not require any other knowledge and does not make any additional assumptions about the platform to avoid falsifying the input data. Figure 2 shows an overview of our approach. The following subsections explain each part in more detail.

3.1 Input Data

The input is a point cloud representing the tracking area. For a single sensor, this input can be directly forwarded to the next step. For multiple sensors, their point clouds need to be aligned. The user needs to specify an initial coarse alignment for every additional sensor, which is refined through point cloud registration. This requires some overlap between the point clouds. When working with any number of sensors, note that different parts of an object are captured at the edges of the field of view compared to the center, which can lead to inaccuracies. With multiple sensors, a similar situation can occur when the tracked object enters the field of view of an additional sensor.

3.2 Ground Removal

Ground removal is the first step in the pipeline. It mainly consists of a pre-filtering step and standard RANSAC-based plane fitting. Pre-filtering ensures that plane fitting is only applied to points close to the ground. Furthermore, it is possible to downsample the point cloud. This can negatively influence the tracking accuracy, but allows for real-time computation times in all cases.

3.3 Background Removal

Background removal is used to split moving foreground points from static background points. A back-

ground cloud is built over time, storing the probability of points being part of the background. The background cloud contains points in 3d space that store the background probability for its immediate surroundings and are matched against points from an input cloud. The background probability from the background cloud point defines whether the corresponding input point is part of the foreground or background.

Data: $cloud_{new}$, $cloud_{bg}$
Result: $cloud_{fg}$
$U \leftarrow [false, \ldots]$;
foreach $p_{new} \in cloud_{new}$ **do**
  $p_{corr}, \theta_{corr} \leftarrow$
   $nearestPoint(point_{new}, cloud_{bg})$;
  if $\|p_{new} - p_{corr}\| < corr\_dist$ **then**
    $U[p_{corr}] \leftarrow true$;
    if $\theta_{corr} < bg\_thresh$ **then**
      add p_{new} to $cloud_{fg}$;
    end
  else
    add p_{new} to $cloud_{bg}$;
    add p_{new} to $cloud_{fg}$;
  end
end
update $cloud_{bg}$ using U;

Algorithm 1: Algorithm for background removal.

Given a new input cloud and a background cloud, each point of the input cloud is checked for being in the background by finding its corresponding background point, as shown in Algorithm 1, where θ_{corr} is the background probability for p_{corr}. If no corresponding point is in close distance or if the corresponding point is not part of the background, the current point is added to the foreground cloud. Depending on the case, either a new point is added to the background cloud, if no corresponding point was found, or the background probability of the corresponding point is increased. This update of the background cloud is done at the end.

$$\theta^t_{corr} = \begin{cases} \min(\theta^{t-1}_{corr} + t_{in}\Delta t, 1), & U[p_{corr}] \\ \max(\theta^{t-1}_{corr} - t_{out}\Delta t, 0), & \neg U[p_{corr}] \end{cases} \quad (1)$$

The probability update for the background cloud points is shown in Equation 1. If a background point was accessed in the current step ($U[p_{corr}] = true$), the probability of the point being part of the background is increased. t_{in} is the time in seconds that is required for a point to reach maximum background probability. Δt is the difference between the capture times of the current and previous input clouds. In the event, that the input cloud did not contain a corresponding point for a point in the background cloud, its probability is decreased, with t_{out} being the fade out time.

The background cloud is initialized using a downsampled version of the first input cloud. For faster initialization, new points are added to the background cloud with an initial probability value. This initial value is the sum of the background threshold and t_{in}. Note that robotic platforms are initially part of the background and need to move to become detectable.

3.4 Cluster Detection

The cluster detection is based on Euclidean distance clustering, but differentiates between a foreground and a background. The point clouds are provided by the prior step. To save computation time, both input clouds are downsampled. New clusters are only created for points in the foreground and can grow using points from both foreground and background. Clustering is finished when every foreground point is part of a cluster. To counteract inaccuracies in background removal, i.e. having single foreground points that are part of a large background cluster, we added a foreground-to-background ratio. This classifies clusters as invalid, if the ratio between foreground and background points exceeds a certain threshold (here 50 %). Invalid clusters are discarded. For each valid cluster, the centroid and the oriented bounding box are calculated. Since dimensions in datasheets are typically axis-aligned, we only consider the rotation on the up-axis of the bounding box.

3.5 Robot Detection and Tracking

The tracking of robotic platforms is done by tracking-by-detection, using only the dimension and position information of detected clusters and a model. A model of a robotic platform is provided by the user and consists of the three values width, height, and length, as stated in the datasheet of the robotic platform. It is used to find the robotic platform within a list of detected continuous clusters. The detected clusters are compared against the model and against each other using three figures of merit for the error. The first error E_a is the difference in aspect ratio (Equation 2), inspired by (Kaku et al., 2004). Instead of the principal axis length, we use the bounding box dimensions.

$$E_a = \sqrt{\left(\frac{A_2}{A_1} - \frac{B_2}{B_1}\right)^2 + \left(\frac{A_3}{A_1} - \frac{B_3}{B_1}\right)^2} \quad (2)$$

A_1, A_2, and A_3 are the bounding box dimensions of the first cluster sorted by size with A_1 being the

largest dimension. Thus, B_1, B_2, and B_3 are either the dimensions of the model or the second cluster.

$$E_v = \frac{1}{3} \times \left| \left(\frac{A_1 \times A_2 \times A_3}{B_1 \times B_2 \times B_3} \right) - 1 \right| \tag{3}$$

The second error E_v is for the difference in volume between the first cluster and either the model or second model (Equation 3). The inputs for this equation are the same as in Equation 2. We divide this error by three to reduce its influence on the overall error, as small changes can already lead to a high error value. Also note that compared to using Intersection over Union, we do not require a position and our error is more emphasized when the first cluster is larger than the model or the second cluster.

$$E_d = \|p_A - p_B\| \tag{4}$$

The third error E_d is the Euclidean distance between the centers of the two clusters (Equation 4).

$$\begin{aligned} E &= E_a + E_v + E_d \\ E_s &= E_a + E_v \end{aligned} \tag{5}$$

As shown in Equation 5, the errors are summed to form the overall error E. When calculating the error E_s using the model, the position error is omitted, as the model does not provide position data.

To track robotic platforms, we use continuous clusters. A continuous cluster is a data structure that stores information about a tracked object, including position, bounding box, and error data. It is updated using detected clusters in new point clouds. For every new cluster, we try to find the best-fitting continuous clusters using the error formula as shown in Algorithm 2. If a fitting continuous cluster was found, we update the continuous cluster based on the new cluster. If no fitting continuous cluster was found, we return false and the cluster is inserted as a new continuous cluster. We use a constant maximum error to prevent clusters from being inserted into a wrong continuous cluster. Also note that we differentiate between normal clusters and sparse clusters, which do not contain more than eight points. As they contain little information, they need to be handled differently. This means that 1. they can not form a new continuous cluster and 2. can only update a continuous cluster if that cluster is the tracked robotic platform.

For a continuous cluster to be included in the search for the robotic platform, it must be valid. A continuous cluster is considered valid when it has been detected in every frame for a certain amount of time by the cluster detection. If a continuous cluster is valid, it is evaluated against the model by calculating a mean error value for every cluster that was assigned to it using the cluster detection method described above. The bounding boxes of the clusters are

Data: $cluster_{new}$, $clusters_{cont}$
Result: bool
$cc_{best} \leftarrow None$;
$error_{best} \leftarrow MAX\_ERROR$;
foreach $cc \in clusters_{cont}$ **do**
  $error_{cc} \leftarrow calculateError(cluster_{new}, cc)$;
  if $error_{cc} < error_{best}$ **then**
    $cc_{best} \leftarrow cc$;
    $error_{best} \leftarrow error_{cc}$;
  end
end
if $cc_{best} \neq None$ **then**
  if cc_{best} *has not been updated this step* **then**
    update cc_{best} with $cluster_{new}$;
  else
    revert cc_{best} update;
    update cc_{best} with $cluster_{new}$;
    rerun this algorithm for the reverted cluster;
  end
  return true;
end
return false;

Algorithm 2: Algorithm for updating a continuous cluster.

compared against the model dimensions. We limit the number of evaluated clusters to avoid an infinitely increasing cluster count. The continuous cluster with the smallest error value is selected as the tracked robotic platform. We avoid false selections by using a minimum required error value. If a continuous cluster has not been updated or has not been marked as valid after a specified amount of time, it is removed from the list of continuous clusters.

It is possible that the best continuous cluster is not updated for the current input cloud when no fitting cluster is found. There are two main reasons for this to happen: 1. the tracked object has moved out of the field of view, and 2. the tracked object has not moved for a certain amount of time and therefore faded into the background. While, for the first reason, the object can no longer be tracked, the tracked object is still visible in the second case, and we need to update it in case of small movements. Similar to the cluster detection part, we can use the previous position of the cluster as the single foreground point and find its corresponding cluster using the additional background information. If a cluster was found in close proximity, we can update the continuous cluster with the new cluster. We can assume that the detected cluster is the tracked object, as the chosen proximity value only allows for minimal offset.

Position refinement is the last step of the robot detection and tracking procedure before publishing the

Table 1: Position error (RMSE) for the Avia LiDAR sensor using the Multi-Lidar Multi-UAV dataset (Unit: meter, Δt: 0.2 s).

MAV	Sequence									Mean
	Stnd01	Stnd02	Stnd03	Stnd04	01	02	03	04	05	
Holybro	0.05	0.043	0.0446	0.0503	0.0731	0.0992	0.1263	0.0681	0.0777	**0.0702**
Autel	-	-	-	-	0.1313	0.1074	0.1007	0.0845	0.1089	**0.1066**
Tello	-	-	-	-	0.1032	0.0991	0.1223	0.1008	0.0965	**0.1044**

Table 2: Position error (RMSE) for the Avia LiDAR sensor using the Multi-Lidar Multi-UAV dataset (Unit: meter, Δt: 0.1 s).

MAV	Sequence									Mean
	Stnd01	Stnd02	Stnd03	Stnd04	01	02	03	04	05	
Holybro	0.0748	0.0701	0.0612	0.0854	0.0844	0.0837	0.1008	0.0611	0.0802	**0.078**
Autel	-	-	-	-	0.0844	0.0666	0.0713	0.0637	0.0658	**0.0703**
Tello	-	-	-	-	0.0578	0.0489	0.0585	0.0435	0.0493	**0.0508**

final position. As from cluster detection until now, a downsampled version of the input cloud has been used, the accuracy of the estimated position can be improved. Therefore, based on the estimated position, the cluster is again extracted from the high resolution cloud provided by the ground removal part. This cluster now includes edge points that might not have been considered with the downsampled version. Finally, the cluster center is calculated using this new cluster.

4 EVALUATION

We evaluate our proposed system in terms of accuracy using an MAV dataset (Catalano et al., 2023b) for comparison with a motion capturing system and a custom dataset for comparison with the Vive Tracker system. By using these two datasets, we show that our system is applicable to MAVs as well as mobile robots and that it is able to process LiDAR data as well as regular ToF data. In addition, the custom dataset highlights requirements in the system setup.

4.1 MLMU Dataset

The Multi-LiDAR Multi-UAV dataset (Catalano et al., 2023b) is a dataset designed for detecting and tracking MAVs using LiDAR sensors. It contains MAV flights recorded by three different LiDAR sensors, as well as the OptiTrack motion capturing system for ground truth data. We only focus on the Livox Avia sensor. Due to the distance between the sensors and the MAVs, the point density is too low for the other two sensors to provide accurate results. This could be mitigated by reducing the distance between the sensor and the tracked object, which we were unable to test. The three MAVs captured in the dataset are the Holybro X500, the Autel Evo II, and the Tello. We omitted the outdoor captures, since we focus on tracking in indoor environments. Since a single LiDAR capture only contains partial information about the environment, we preprocess the point clouds by merging them over a certain timespan. We decided to test the values of $\Delta t = 0.1\ s$ and $\Delta t = 0.2\ s$. The merged point clouds are published at a frequency of 16 hz in both cases, therefore adding single captures to multiple point clouds.

The results of our evaluation are shown in Table 1 for $\Delta t = 0.1\ s$ and in Table 2 for $\Delta t = 0.2\ s$. While the error increased by 11 % with $\Delta t = 0.1\ s$ for the Holybro, the results of the Autel and Tello are better (-43 %) with a smaller timespan. While the larger timespan doubles the point density, we noticed that higher values leave a point trail, which is especially noticeable with the smaller MAVs. On the other hand, while the point density is sufficient for the smaller MAVs Autel and Tello with $\Delta t = 0.1\ s$, at some occasions, the clustering fails for the Holybro, which selects only parts of the MAV and therefore leads to inaccuracies. Compared to the base results in (Catalano et al., 2023b), we achieve similar accuracies without the focus on a specific sensor type. Using only their evaluated sequences, they perform better on the standard sequences, possibly due to the use of a kalman filter, while we achieve a better overall mean error of 0.0589 m compared to their result of 0.0655 m.

4.2 Custom Dataset

The custom dataset has been captured using two pico Monstar Infrared ToF sensors, with ground truth data being provided by a Vive Tracker system. The two sensors have been placed as shown in figure 2 (b), with the second sensor capturing a side view. The dataset contains recordings of the mobile robots Pepper and Turtlebot 2. An overview of the types of paths recorded for both platforms is given in Figure 3. Sequences 7-8, which are not shown, also include ran-

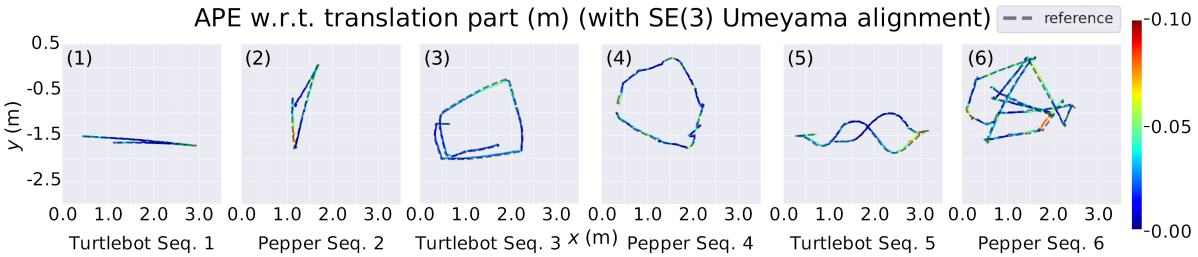

Figure 3: Overview of some of the sequences recorded in the custom dataset for each robot using the Turtlebot and Pepper robots. The path types include *forward-backward (1)*, *left-to-right (2)*, *square (3)*, *circle (4)*, *slalom (5)*, and *random (6)*.

dom movement with a focus on longer durations, like in sequence 6. Sequence 0 is a mobile robot standing still after a small initial forward movement to capture noise recorded by the sensor. Compared to the other dataset, this dataset has a stronger focus on the use case. This includes a smaller distance between the sensors and the tracked objects for a higher point density, as well as using two sensors to evaluate the difference between using one or more sensors. The individual point clouds have been captured with a frequency of 5 *hz* and downsampled using a voxel size of 0.02 *m*. The downsampling is necessary in some cases to ensure real-time performance, e.g. when the mobile robot is close to the sensor. With smaller robotic platforms, i.e. MAVs, this might not be necessary.

The results for the custom dataset are shown in Table 3. The mean position errors are between 3 *cm* and 4 *cm*. The mean error is 10 % higher for the Pepper robot. This is due to the more complex shape compared to the Turtlebot. Furthermore, small uncontrollable upper body movements of the robot can also negatively impact the accuracy. The results also show a smaller error of -14 % when using two sensors, as multiple sensors can better capture the full shape of the tracked object. Note that for sequence 1 (see figure 3a), there is an improvement in accuracy using a single sensor, as the second sensor alters the recorded shape of the robotic platform due to the respective left-right movement leading to inaccuracies. Another reason for inaccuracies, as can be seen in figure 3c and 3e, are the accuracy drops on the outer parts of the tracking area as the visible part of the robot changes. This is also present with the Pepper robot and is amplified through the difference in shape, when rotating the robot, as shown in figure 3b, 3d, and 3f. The bounding box shape depends on the orientation, which can lead to a shift in the estimated center of the robot.

Table 3: Position error (RMSE) for the pico Monstar sensor using the custom dataset for a single and for two sensors (Unit: meter).

Sequence	Turtlebot		Pepper	
	Multi-view	Single-view	Multi-view	Single-view
0	0.0105	0.0054	0.0104	0.0138
1	0.0328	0.0306	0.0364	0.0361
2	0.0284	0.0316	0.0352	0.0429
3	0.0255	0.0305	0.0267	0.035
4	0.0175	0.0209	0.0292	0.0358
5	0.0337	0.0328	0.0284	0.0343
6	0.0289	0.0349	0.0394	0.0461
7	0.0418	0.059	0.0357	0.0463
8	0.04	0.0498	0.0426	0.0412
Mean (1-8)	**0.031**	**0.0362**	**0.0342**	**0.0397**

5 CONCLUSION

In this paper, a system for detecting and tracking robotic platforms for capturing tracking data in lab environments was presented. The main purpose of the system is to allow for easy capturing of position data using already available sensors instead of investing in a costly motion capturing system. It uses Time-of-Flight based sensors, like LiDAR, to capture point clouds. Tracking is already possible using a single sensor, which allows for a fast setup. Only based on the dimensions of a model, robotic platforms are detected in these clouds and their positions are estimated. There is no need to mount additional hardware or markers onto the robotic platform.

The evaluation shows that the system is capable of detecting and tracking different robotic platforms with accuracies in the low-centimeter range using a single or two sensors and with low double-digit frequencies. While this does not match the accuracies and frequencies of high-quality motion capturing systems, we argue that our solution is still applicable to many use cases. Future work will include an in-

tegration of orientation information provided by the robotic platforms, as well as refinements on the tracking accuracy by including the model dimension information in the final position estimation. Supplementary documentation and the source code are available at https://github.com/IoT-Lab-Minden/RP_Tracking.

REFERENCES

Catalano, I., Sier, H., Yu, X., Westerlund, T., and Queralta, J. P. (2023a). Uav tracking with solid-state lidars: Dynamic multi-frequency scan integration. *arXiv preprint arXiv:2304.12125*.

Catalano, I., Yu, X., and Queralta, J. P. (2023b). Towards robust uav tracking in gnss-denied environments: a multi-lidar multi-uav dataset. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*.

Chen, H., Chen, X., Chuanlong Xie and, S. W., Zhou, Q., Zhou, Y., Wang, S., Su, H., and Quanfeng Xu and, Y. L. (2024). Uav tracking and pose-estimation - technical report for cvpr 2024 ug2 challenge. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Deng, T., Zhou, Y., Wu, W., Li, M., Huang, J., Liu, S., Song, Y., Zuo, H., Wang, Y., Wang, H., and Chen, W. (2024). Multi-modal uav detection, classification and tracking algorithm - technical report for cvpr 2024 ug2 challenge. In *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Dogru, S. and Marques, L. (2022). Drone detection using sparse lidar measurements. *IEEE Robotics and Automation Letters*, 7(2).

Furtado, J. S., Liu, H. H., Lai, G., Lacheray, H., and Desouza-Coelho, J. (2019). Comparative analysis of optitrack motion capture systems. In *Advances in Motion Sensing and Control for Robotic Applications*.

Gazdag, S., Möller, T., Filep, T., Keszler, A., and Majdik, A. L. (2024). Detection and tracking of mavs using a lidar with rosette scanning pattern. *arXiv preprint arXiv:2408.08555*.

Geiger, A., Lenz, P., Stiller, C., and Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*.

Giancola, S., Zarzar, J., and Ghanem, B. (2019). Leveraging shape completion for 3d siamese tracking. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kaku, K., Okada, Y., and Niijima, K. (2004). Similarity measure based on obbtree for 3d model search. In *International Conference on Computer Graphics, Imaging and Visualization (CGIV)*.

Nie, J., Xie, F., Zhou, S., Zhou, X., Chae, D.-K., and He, Z. (2024). P2p: Part-to-part motion cues guide a strong tracking framework for lidar point clouds. *arXiv preprint arXiv:2407.05238*.

Pleterski, J., Škulj, G., Esnault, C., Puc, J., Vrabič, R., and Podržaj, P. (2023). Miniature mobile robot detection using an ultralow-resolution time-of-flight sensor. *IEEE Transactions on Instrumentation and Measurement*, 72.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *arXiv preprint arXiv:1706.02413*.

Qi, H., Feng, C., Cao, Z., Zhao, F., and Xiao, Y. (2020). P2b: Point-to-box network for 3d object tracking in point clouds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Qingqing, L., Xianjia, Y., Queralta, J. P., and Westerlund, T. (2021). Adaptive lidar scan frame integration: Tracking known mavs in 3d point clouds. In *International Conference on Advanced Robotics (ICAR)*.

Shule, W., Almansa, C. M., Queralta, J. P., Zou, Z., and Westerlund, T. (2020). Uwb-based localization for multi-uav systems and collaborative heterogeneous multi-robot systems. *Procedia Computer Science*.

Sier, H., Yu, X., Catalano, I., Queralta, J. P., Zou, Z., and Westerlund, T. (2023). Uav tracking with lidar as a camera sensor in gnss-denied environments. In *International Conference on Localization and GNSS (ICL-GNSS)*.

Taffanel, A., Rousselot, B., Danielsson, J., McGuire, K., Richardsson, K., Eliasson, M., Antonsson, T., and Hönig, W. (2021). Lighthouse positioning system: Dataset, accuracy, and precision for uav research. *arXiv preprint arXiv:2104.11523*.

Waşik, A., Ventura, R., Pereira, J. N., Lima, P. U., and Martinoli, A. (2015). Lidar-based relative position estimation and tracking for multi-robot systems. In *Robot 2015: Second Iberian Robotics Conference*.

Wang, H., Chen, C., He, Y., Sun, S., Li, L., Xu, Y., and Yang, B. (2024). Easy rocap: A low-cost and easy-to-use motion capture system for drones. *Drones*, 8(4).

Wang, H., Peng, Y., Liu, L., and Liang, J. (2021). Study on target detection and tracking method of uav based on lidar. In *Global Reliability and Prognostics and Health Management (PHM-Nanjing)*.

Yilmaz, Z. and Bayindir, L. (2022). Lidar-based robot detection and positioning using machine learning methods. *Balkan Journal of Electrical and Computer Engineering*, 10(2).

Yuan, S., Yang, Y., Nguyen, T. H., Nguyen, T.-M., Yang, J., Liu, F., Li, J., Wang, H., and Xie, L. (2024). Mmaud: A comprehensive multi-modal anti-uav dataset for modern miniature drone threats. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Zhou, C., Luo, Z., Luo, Y., Liu, T., Pan, L., Cai, Z., Zhao, H., and Lu, S. (2022). Pttr: Relational 3d point cloud object tracking with transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.