Robust 3/6 DoF self-localization system with selective map update for mobile robot platforms

Carlos M. Costa[a,*], Héber M. Sobreira[a], Armando J. Sousa[a], Germano M. Veiga[a]

[a]*INESC TEC, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal*

Abstract

Mobile robot platforms capable of operating safely and accurately in dynamic environments can have a multitude of applications, ranging from simple delivery tasks to advanced assembly operations. These abilities rely heavily on a robust navigation stack, which requires stable and accurate pose estimations within the environment. To solve this problem, a modular localization system suitable for a wide range of mobile robot platforms was developed. By using LIDAR / RGB-D data, the proposed system is capable of achieving 1-2 centimeters in translation error and 1-3 degrees in rotation error while requiring only 5-35 milliseconds of processing time (in 3 and 6 DoF respectively). The system was tested in three robot platforms and in several environments with different sensor configurations. It demonstrated high accuracy while performing pose tracking with point cloud registration algorithms and high reliability when estimating the initial pose using feature matching techniques. The system can also build a map of the environment with surface reconstruction and incremental update it with either the full field of view of the sensor data or only the unknown sections, which allows to reduce the mapping processing time and also gives the possibility to update a CAD model of the environment without degrading the detail of known static areas due to sensor noise.

Keywords: self-localization, simultaneous localization and mapping, point cloud registration, geometric feature matching

1. Introduction

Humanity has sought a reliable method of navigation ever since it started to explore the world. It began with simple landmark reference points for local travels, then perfected celestial navigation for global journeys, and when it finally conquered space, it deployed a global localization system. Autonomous robots face the same problem, because in order to be able to navigate with precision, they first need to know their location.

Over the years, several localization methods have been proposed and refined, according to the navigation environment and the accuracy requirements. Some are meant for high precision local navigation, while others provide an approximate global position.

A robot capable of operating safely and accurately in a dynamic environment can have innumerous applications, ranging from simple delivery tasks to advanced assembly. Besides improving productivity by performing repetitive tasks with precision and speed, robots can also act as coworkers, helping humans perform their jobs more efficiently and thus, reducing the overall production costs.

Mobile robot platforms have a wide range of localization systems to choose from. Odometry is one of the simplest localization methods and relies on proprioceptive information provided by wheel encoders to incrementally update the known pose. But this approach is very sensitive to wheel drift and can accumulate a significant amount of error over time. This method can be improved with filters that model the odometry error, such as Kalman filters [38], but most localization systems choose to rely on a combination of proprioceptive and exteroceptive information in order to reliably estimate the robot pose. In [18] it is presented a very detailed analysis of the available indoor localization systems and the sensing devices that can be used. Most mobile manipulators that require high precision tracking and safety certified sensors rely on LIght Detection And Rangings (LIDARs), since they can operate in a wide range of atmospheric and lighting conditions and provide very accurate measurements of the environment. Nevertheless, stereo vision and RGB-D systems can also achieve very accurate pose estimation and can perform mapping of the environment much faster.

Most of the available localization systems can be categorized as point cloud registration systems, feature registration systems or probabilistic pose estimation systems. Examples of point cloud registration systems such as the ones presented in [15] and [20] can operate in robots moving at high speeds while the one introduced in [3] can directly register clouds in polar coordinates. A different kind of cloud registration is proposed in [17] in which the points normal distributions are used instead of the points themselves. Other systems perform feature matching either from stereo cameras [13] or RGB-D sensors [14] and can achieve very high update rate, map the environment really fast and integrate color information besides the geometry itself. Particle filters [6, 33] are another group of localization systems

*Corresponding author

Email addresses: carlos.m.costa@inesctec.pt (Carlos M. Costa), heber.m.sobreira@inesctec.pt (Héber M. Sobreira), asousa@fe.up.pt (Armando J. Sousa), germano.veiga@inesctec.pt (Germano M. Veiga)

that rely on probabilistic models in order to provide robust pose estimation even when the robot becomes temporarily lost. Besides localization, other systems such as [9, 12, 34, 30, 32] were developed to perform long term localization and mapping in dynamic environments in order to improve both the pose estimation accuracy and the navigation path planning efficiency. On a more smaller scale, the work presented in [19] showed impressive mapping results even with dynamic and deformable objects.

Some of these localization systems can be used for accurate pose tracking while others provide global pose estimations with less accuracy. The proposed implementation achieves both of these goals and provides an efficient, modular, extensible and easy to configure localization system, capable to operate on a wide range of robot platforms and environments. It is capable of performing high accuracy pose estimation (and robust tracking recovery) using point cloud registration algorithms and it can also reliably estimate the global position using feature matching. It can use several point cloud sensing devices (such as LIDARs or RGB-D cameras) and requires no artificial landmarks. Moreover, it can dynamically update the localization map at runtime and can adjust its operation rate based on the estimated sensor velocity in order to use very few hardware resources when the robot platform is not moving. It also offers a detailed analysis of each pose estimation, providing information about the percentage of registered inliers, the root mean square error of the inliers, the angular distribution of the inliers and outliers, the pose corrections that were performed in relation to the previous accepted pose and in case of initial pose estimation it also gives the distribution of the accepted initial poses, which can be very valuable information for a supervisor when the robot is in ambiguous areas that are very similar in different parts of the known map (and as such, requires the navigation supervisor to plot a path to disambiguate the initial pose before beginning any critical operations).

The next section provides a detailed description of the proposed localization system, explaining each main stage of its processing pipeline. Section 3 describes the testing platforms and environments that were used in the evaluation of the proposed 3/6 Degrees of Freedom (DoF) Robot Operating System (ROS) implementation. Sections 4 and 5 discuss the achieved results in 3/6 DoF and Section 6 finishes with the conclusions.

2. Localization system

This section details the ROS implementation of the proposed 3/6 DoF Dynamic Robot Localization system (DRL)[1]. It starts with an overview of the main processing stages and then details the control flow and algorithms within the processing pipeline.

2.1. Overview

The self-localization system was implemented as a ROS package and extensible uses the Point Cloud Library (PCL) [27] for its processing pipeline. It provides 3/6 DoF localization by publishing *geometry_msgs::PoseStamped* and *geometry_msgs::TransformStamped* messages along with a detailed analysis of the pose estimation and registered point cloud (split into inliers and outliers). Moreover, it also gives detailed analysis of the computation runtime of each of its modules in order to pinpoint which algorithms are using more computation resources (which is very useful information when configuring or upgrading the system).

The ROS implementation can receive sensor data through *sensor_msgs::PointCloud2* messages and as a result it can directly use data from RGB-D and Time of Flight (ToF) cameras. To use LIDARs it provides an assembler that can produce point clouds by merging measurements from several sensor scans using spherical linear interpolation. As such, if the LIDAR sensors are mounted on tilting platforms, they can emulate a 3D sensor and retrieve a very detailed view of the environment.

The self-localization system has a modular software architecture and was implemented as several C++ templated shared libraries that can be easily used for other applications besides robot self-localization. As can be seen in figures 1 and 2, it is an extensible and flexible system able to fit the needs of a wide range of mobile platforms. It can be configured as a tracking system, with or without pose recovery and can also have initial pose estimation using feature detection and matching. Moreover, it can dynamically create and update the map if necessary.

It supports two configurable processing pipelines in order to allow fast deployment of robots in large environments. One to process new reference maps and another to localize a mobile robot platform using live point clouds. This enables the loading of either processed or unprocessed referenced point clouds and allows a navigation supervisor to dynamically provide the relevant map sections based on the robot position (in order to reduce the computation resources needed by lowering the number of kd-tree levels of the reference point cloud).

2.2. Pipeline configuration

The self-localization system was designed to allow fast reconfiguration[2] and parameterization through the use of yaml files and the ROS parameter server. This gives the possibility to quickly tune the localization system to the specific needs of a given mobile platform moving in a particular environment, in order to use the least amount of computational resources possible and without requiring any reprogramming or source code modification. Nevertheless, the system can use a generic configuration if hardware resources are not a concern.

In a typical configuration (following the *Yes* paths of the activity diagram in Figure 2), the first time the localization system is used, it receives a raw reference point cloud that is preprocessed and saved to long term memory along with its associated keypoints and keypoint descriptors. This allows a much faster startup the next time the localization system is initialized. After having a reference point cloud, the localization

[1] `https://github.com/carlosmccosta/dynamic_robot_localization`

[2] `https://github.com/carlosmccosta/dynamic_robot_localization/blob/hydro-devel/yaml/schema/drl_configs.yaml`

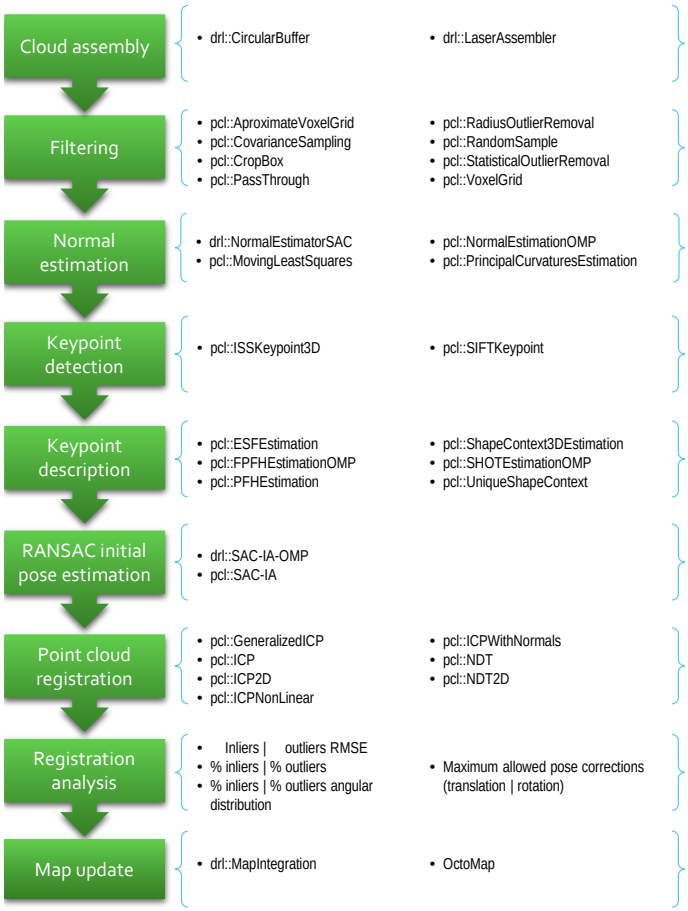


Figure 1: DRL system modules overview

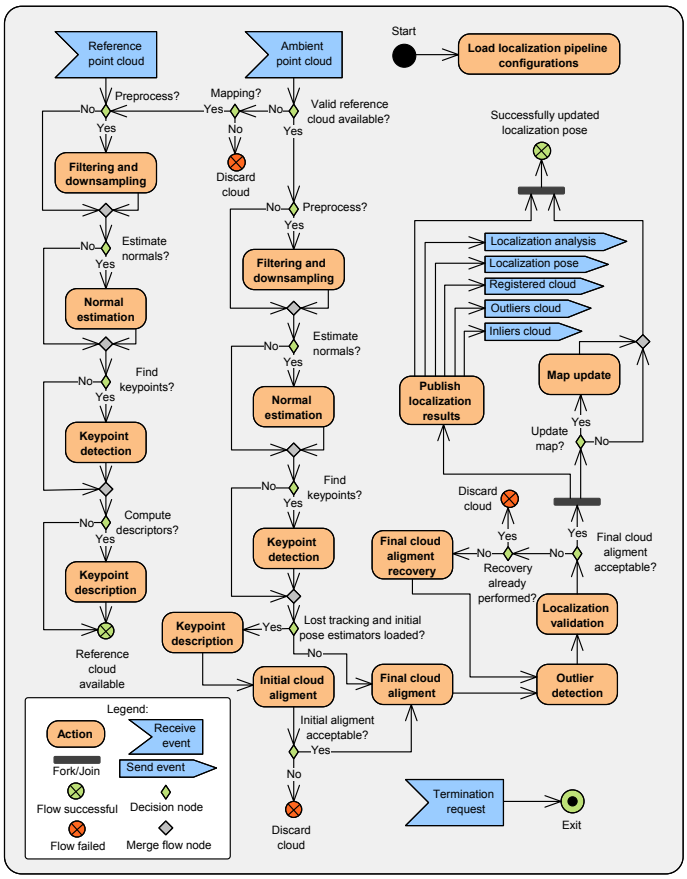


Figure 2: DRL system detailed processing pipeline

system will estimate the robot pose periodically by analyzing the live point cloud sensor data. This data can be preprocessed with several filters and can be associated with computed surface normals. The robot pose estimation is performed by applying a matrix transformation correction to the current robot pose and is based on the registration of the live point cloud with the known map. This registration can use a tracking algorithm configuration tuned for efficiency and a second configuration for tracking recovery purposes. These tracking algorithms require a initial pose estimation, and as such, if one isn't available, a third configuration can be employed to estimate the global position of the robot using geometric features of the environment. The switch between these configuration is based on the analysis of the registered cloud metrics, such as outlier percentage, inliers root mean square error, inliers angular distribution and the registration corrections performed on the live point cloud. After successfully performing the robot pose estimation, the map can be updated by either integrating the full registered point cloud, its inliers or its outliers. Finally, like most ROS nodes, the localization system will stop its execution when it receives a termination signal request.

The next subsections explain in detail the architecture and algorithms used in each of the processing modules present in figures 1 and 2.

2.2.1. *Reference map*

The reference point cloud can be loaded from a Computer Aided Design (CAD) file, point cloud file or dynamically arrive trough a ROS topic as either a 3 DoF *nav_msgs::OccupancyGrid* or 6 DoF *sensor_msgs::PointCloud2*. This allows a localization supervisor to give only sections of a global map in order to use the least amount of memory and processing time (very large maps have deeper search structures, such as kd-trees, and should be avoided in order to keep the number of tree levels within reasonable values).

2.2.2. *Point cloud assembly*

The self-localization system can use any sensor that provides point clouds, namely RGB-D / ToF cameras, LIDARs and stereo vision systems. Each of these types of sensors have very different operation rates and measurements accuracies. As such, the localization system allows the assembly of several live scans using a circular buffer in order to reduce the impact of sensor noise.

For LIDARs, the system provides a *sensor_msgs::LaserScan* assembler[3] that converts laser measurements in polar coordinates into Cartesian coordinates and projects the points using

[3] `https://github.com/carlosmccosta/laserscan_to_pointcloud`

spherical linear interpolation (in order to account for laser scan deformation that occurs when the robot is moving and rotating). It can merge scans from several lasers sensors and it will publish the final point cloud after assembling a given number of scans or periodically after a specified duration. These assembly configurations can be changed at runtime based on the robot velocity (for example, when the robot moves slower, more laser scans are assembled for each published point cloud) or through the use of the ROS dynamic reconfigure Application Programming Interface (API), which allows a navigation supervisor to control the rate at which the localization system operates.

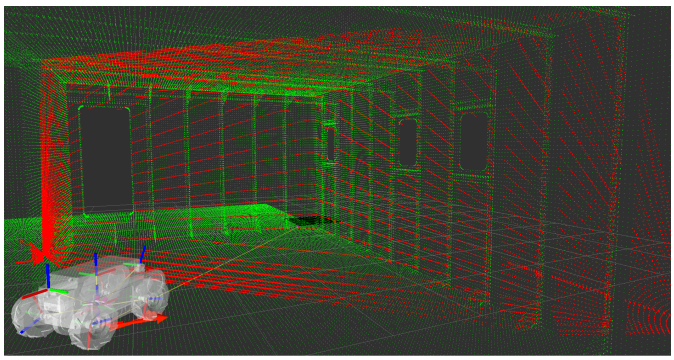


Figure 3: Laser scans assembled with tilting platform (red points)

2.2.3. Filtering and down sampling

The time it takes to perform cloud registration increases considerably when the amount of points in the live point cloud and in the reference map becomes larger. As such, adjusting the level of detail of the point clouds by using voxel grids gives some control over the desired localization accuracy and the computational resources that will be required. This stage is also useful to mitigate the measurement errors of the depth sensors, since the centroid of a voxel that contains points from several scans will be closer to the real surface (if the voxels have dimensions slightly larger than the expected measurement errors).

The localization system allows the application of several preprocessing algorithms (algorithm type / configuration and execution order customizable). The next sections detail some of the methods that can be used to filter and downsample point clouds.

Voxel grid sampling. A voxel grid is a uniform space partition technique that can be used to cluster points according to their Euclidean coordinates. It is a very effective method to control the level of detail of a point cloud because it gives the ability to specify the maximum number of points that a region of space should have.

The point cloud downsampling is achieved by replacing each voxel cluster with a single point. The selection of this point can be very fast if the voxel center is used (pcl::ApproximateVoxelGrid), but computing the centroid of the cluster (pcl::VoxelGrid) yields better results because it represents the underlying surface with more accuracy and it attenuates errors in the sensors measurements.

Random sampling. Random sampling [37] is a fast downsampling method that randomly selects points from the input cloud until the specified number of samples is reached (pcl::RandomSample). This has the advantage of using real measures instead of downsampled approximations, but also means it is more sensible to sensor measurements noise. However, for outdoor environments or very complex scenes, using the real measurements might be preferable than using cluster centroids because the voxels may not have the necessary resolution or may have a prohibitive computational cost.

Covariance sampling. Covariance sampling [8] is a subsampling method that aims to create a stable downsampled point cloud to be registered with Iterative Closest Point (ICP) algorithms. It incrementally builds the downsampled point cloud while trying to keep the 6 eigenvalues of the filtered point cloud covariance matrix as close to each other as possible (pcl::CovarianceSampling).

The resultant point cloud has the desired number of points and is stable enough to be matched with ICP point to plane algorithms.

2.2.4. Outlier removal

Some depth sensors can perform measurements with high accuracy, but they have some limitations that can lead to the creation of outliers [29]. One of those limitations can produce shadow points around objects boundaries. This is due to the fact that a portion of the laser beam may hit the object boundaries and other part may hit areas in the object background. And given that most laser range finders use a weighted sum of several beams, this can yield measurements that are not associated with any real object (outliers). Another issue is related to the angle in which the depth sensor sees the objects areas. If the incidence angle is very low, then it may be difficult to compute its distance and detect if the beam had ambient reflections. This can significant increase the measurements noise or even lead to the creation of outliers. Other common problem is associated with the material properties of the surfaces. For example, objects with very high or very low reflectance, such as metals or glass, can increase the measurements noise. Moreover depending on the combination of surface geometry, material and incidence angle, some objects may even be undetectable by depth sensors.

Given the negative effect that outliers have in surface normal computation and registration algorithms, they should be removed in a preprocessing stage. There are several approaches to perform outlier detection and removal [40], ranging from simple distance thresholds to more robust statistical analysis. The next sections present some of the methods that can be useful in a localization system.

Distance filter. Given that depth sensors have a minimum and maximum recommended distance for their measurements, it is wise to remove points that are close or beyond these limits. Moreover, it may be useful to remove points that are too close to the sensor, because they may belong to the robot itself and not the environment.

This can be achieved by applying a minimum and maximum threshold to the distances returned by the depth sensor (drl::LaserAssembler and pcl::CropBox).

Passthrough filter. A passthrough filter (pcl::PassThrough) can select or remove points according to their properties. For outlier removal, it can be used to select points that are within a given bounding box (useful when we already know what area of the environment we want to analyze) or remove points that don't have the appropriate intensity or color.

Radius outlier removal filter. The radius outlier removal filter (pcl::RadiusOutlierRemoval) deletes points that don't have a minimum number of neighbors within a specified radius distance. It can be useful when the point density is known and is very effective in removing isolated points.

Statistical outlier removal filter. The statistical outlier removal filter (pcl::StatisticalOutlierRemoval) [26] performs a global analysis of the distances between points and discards the ones that don't follow the global distance distribution. It is a robust filter that adapts itself to the point cloud density and is very effective in removing shadow points. To do so, it computes the mean distance that each point has to a given number of neighbors and builds a global distance distribution. Then, assuming that the distribution is Gaussian, it discards the points that have a distance higher than a given threshold (that is a percentage of the standard deviation of the distance distribution).

2.2.5. Surface and object reconstruction and resampling

Depending on the level of sensor noise and amount of outliers present in a given point cloud, it may be necessary to employ surface reconstruction techniques to fill gaps in sensor data or correct measurements errors.

The Moving least squares [1] is a surface reconstruction algorithm that uses higher order bivariate polynomials to fit surfaces to a given set of points. It can be used to fill possible gaps in sensor data, smooth the point cloud, refine surface normals and perform downsampling or upsampling.

Surface reconstruction can also be useful when the point cloud is built from several live scans with different origins and registered with some alignment errors. This allows to improve the points normals by using the surfaces computed by the moving least squares algorithm (pcl::MovingLeastSquares) instead of using the point's neighbors.

2.3. Normal estimation

Most of feature detection, description and matching algorithms along with some registration methods rely on the point's surface normal and curvature. These algorithms analyze the neighborhood of a given point in order to compute the line / surface normal, and as such, the correct specification of what points should be included in the estimation is crucial to achieve accurate results. This depends on the environment geometry and the level of detail that is required, and is usually done by specifying a radius distance or by limiting the number of neighboring points to use.

The next sections describe some of the methods that are supported by the DRL system for 2D/3D sensor data.

2.3.1. Line normal estimation

Line normals give information about the spatial disposition and orientation of a given cluster of collinear points. Each point normal can be computed using Random Sample Consensus (RANSAC) methods [5] by fitting lines to the cluster of neighboring points (bounded by a given distance or limiting the number of neighbors). After knowing the line equation that best fits a given point cluster, its 6 DoF normal is corrected using the sensor origin in order to be on the same plane as the laser scan / point cloud data (drl::NormalEstimatorSAC).

2.3.2. Surface normal estimation

Surface normals provide information about the orientation of the underlying geometry of a given point cluster. They can be computed using plane fitting methods or using more advanced techniques such as Principal Component Analysis (PCA) [11] (pcl::NormalEstimationOMP) or the one presented in Section 2.2.5 (pcl::MovingLeastSquares).

2.4. Keypoint detection and description

Aligning two point clouds with overlapping views of the environment requires the establishment of point correspondences. If both point clouds have similar sensor origins, these can be determined with nearest neighbor's searches and filtered with correspondence rejectors (using other point properties such as reflectance and color). But if they were acquired in two very different positions, then more advanced techniques must be employed.

One of those techniques uses histograms to describe the geometric properties of the environment around a given point. This allows points to be matched even if they have completely different Euclidean coordinates. Also, by using histograms and sampling techniques, these descriptors are much more robust against sensor noise and varying level of point density. However, these advantages come with a heavy computational cost, and as such, point descriptors should only be computed on the most descriptive areas of the environment.

Identifying these environment points is known as feature / keypoint detection [4], and usually involves finding interesting points, such as corners or edges. Besides uniqueness, these points must also be repeatable. This means that the detection algorithms should be able to find the same points even if they are present in different point clouds with sensor noise and varying point density. This is of the utmost importance, because if the same keypoints are not identified on both clouds, then matching the point descriptors will likely fail.

Currently, the localization system can use the Scale Invariant Feature Transform (SIFT) [16] algorithm on the point's curvature or the 3D Intrinsic Shape Signatures (ISS3D) [41] keypoint detector on the point's normals.

Describing a keypoint usually involves analyzing its neighboring points and computing a given metric or histogram that quantifies the neighbor's relative distribution, their normals angular relation, associated geometry or other metrics that are

deemed useful. Several approaches were suggested over the years according to different recognition needs and they are the basis of feature matching algorithms used in the initial pose estimation.

The DRL system can use most of the keypoint descriptors available in PCL, namely the Point Feature Histogram (PFH) [25], the Fast Point Feature Histogram (FPFH) [24], the Signature of Histograms of Orientations (SHOT) [36], the Shape Context 3D (SC3D) [7], the Unique Shape Context (USC) [35] and the Ensemble of Shape Functions (ESF) [39].

2.5. Cloud registration

Point cloud registration is the process of finding the transformation matrix (usually translation and rotation only) that when applied to a given live point cloud will minimize an error metric (such as the mean square error of the live point cloud in relation to a given reference point cloud). Several approaches were suggested over the years and they can be categorized as point or feature cloud registration.

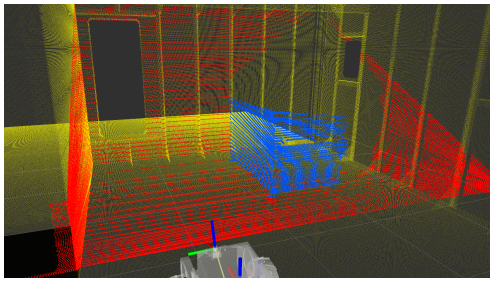
2.5.1. Initial alignment with keypoints descriptor matching

Feature registration is the process of matching keypoint descriptors in order to find an initial alignment between two point clouds. The DRL system uses a feature registration method similar to the Sample Consensus Initial Alignment (SAC-IA) algorithm presented in [24]. It uses a RANSAC approach to select the best registration transformation after a given number of iterations. In each iteration a subsample of randomly selected descriptors from the live point cloud is retrieved. Then for each of these descriptors, k best matching descriptors in the reference point cloud are searched (using a kd-tree) and one of them is chosen randomly (this improves robustness against noise in the sensor data and changes in the environment that are not yet integrated into the map). After having filtered these correspondences between live and reference descriptors, the registration matrix is computed. If this registration matrix results in a point cloud overlap that has a minimum of inliers percentage (a point in the live point cloud is an inlier if it has a point in the reference point cloud closer than a given distance), then it is considered an acceptable initial pose and it is saved (to allow a localization supervisor to analyze the distribution of the acceptable initial poses). In the end of all iterations, the best initial pose (if found) is refined with a point cloud registration algorithm.

2.5.2. Final alignment with point cloud error minimization

Point cloud registration algorithms such as ICP [2] (with its several known variations [23, 20] like ICP point-to-point, ICP point-to-point non-linear, ICP point-to-plane and generalized ICP [28]) and the Normal Distributions Transform (NDT) [17] are among the most popular algorithms to register point clouds. They can achieve very accurate cloud registration but they require an approximate initial pose for the registration to successfully converge to a correct solution (otherwise they may achieve only partial cloud overlap or even fail to converge to a valid solution).

(a) Gazebo overview



(b) RViz overview

Figure 4: Point cloud registration with outlier detection

2.6. Outlier detection

Detecting which points of the environment are not part of the reference point cloud can be very useful to evaluate the quality of point cloud registration as well as to analyze the presence of previously unknown objects. It's computation splits the live cloud into two point sets. One containing inliers (points correctly registered and present in the reference point cloud) and the other having the outliers (points that are either incorrectly registered or not present in the reference cloud).

A given live point can be classified as outlier if the corresponding closest point in the reference cloud is farther away than a given distance threshold. These calculations can be done efficiently (drl::EuclideanOutlierDetector) using the reference point cloud kd-tree. In Figure 4 is an example of a live point cloud retrieved with a tilting laser and registered with an indoor map (yellow points). After registration, the live point cloud was split into inliers (red points) and outliers (blue points).

2.7. Localization validation

After a point cloud is registered by the localization system, several metrics are calculated in order to evaluate if a valid pose can be retrieved using the registration matrix.

The first computed metrics are the percentage of inliers and the Root Mean Square Error (RMSE) of these inliers. If a minimum number of points was registered and the inlier percentage and root mean square error are acceptable (typical values for dynamic environments are at least 35% of inliers with a RMSE lower than 30 mm), then the registration is considered successful. However, these registered points can be agglomerated in a small area and may not be representative of the robot location. As such, a second metric is computed that takes into account the angular distribution (drl::AngularDistributionAnalyzer) of these inliers. This metric gives a measurement of how reliable is this registration and is based on the fact that there is high

confidence in a given pose estimation when there are correctly registered points all around the robot. The recommended parameters for this metric depend on the combined field of view of the sensors and if the robot is in an environment with geometry surrounding it. For typical use cases, a minimum of 60º of inliers angular distribution is reasonable.

The last metrics are the corrections that the registration matrix introduced. Given that the localization system will be in tracking mode most of the time, it is possible to define how far a new pose can be in relation to the previous accepted location and discard new poses that exceed a given threshold. This is useful to discard pose corrections that are very unlikely to happen, such as the robot moving half a meter between poses when it is expected to move only at 30 cm/s. These situations can happen when there is a sudden decrease in the field of view (that can occur due to sensor occlusion or malfunction) or when large unknown objects very similar to sections of the map appear into the field of view of the robot. For a typical robot moving slowly due to safety reasons, a maximum of 10 cm of translation and 10º of rotation between pose estimation might be reasonable (these thresholds depend on the sensors refresh rate and on the expected robot speed).

If all these metrics are within acceptable limits, then the robot pose can be computed by applying the matrix correction to the initial pose associated with the live sensor data.

If any of these metrics are not acceptable, then the system can be configured to simply discard this pose estimation and try to estimate the pose in the next sensor data updates or it can apply a tracking recovery attempt with a different registration algorithm (or the same algorithm with different parameters). This tracking recovery can be activated after a given number of failed point cloud registrations or after a specified timeout (this allows to ignore point clouds that are deformed due to the robot motion during acquisition or that have too much outliers).

If several consecutive pose estimations are discarded (or a given time has passed since the last know pose), the system can have a second level of recovery that can be configured to use the initial pose estimation algorithms in order to finally estimate the global robot pose and reset the tracking state.

2.8. Dynamic map update

After performing a successful pose estimation, the DRL system can update the localization / navigation map by either integrating only the unknown objects or the full registered point cloud (it can also be used in conjunction with OctoMap [10] in order to perform probabilistic map updates).

Integrating only the unknown objects is the recommended approach when there is a known map and the environment is expected to change gradually. This is also more efficient as only the points that need to be integrated are processed and ray traced in OctoMap. Moreover, integrating only new sections avoids map degradation or pollution (by sensor noise) of the detail of static areas (that were provided by CAD models).

On the other hand, integrating the full registered point cloud can be desirable if the map of the environment is very incomplete, very outdated or expected to change considerably during the operation of the robot.

Introducing new elements in the localization map may change slightly the reference coordinate system. As such, the DRL system in conjunction with OctoMap can use a static map for localization and continuously update a different map for the navigation system in order to keep valid previous localization / navigation waypoints and also allow better path planning for longer robot travels (given that some map sections might be obstructed by objects or new pathways might have become available).

3. Testing configurations

This section presents several test scenarios that were devised to evaluate the implemented localization system. They aim to test the accuracy and robustness of the DRL implementation under different environmental conditions and hardware configurations.

3.1. Testing platforms

The localization system was tested on laser sensor data retrieved from three different mobile robot platforms and was executed on the same computer in order to allow a direct comparison of computation time. This computer was a Clevo P370EM3 laptop (with a Intel Core i7 3630QM CPU at 2.4GHz, 16 GB of RAM DDR3, NVidia GTX680M graphics card and a Samsung 840 Pro SSD) and it was running Ubuntu 12.04 along with ROS Hydro, PCL 1.7 and Gazebo 1.9.

The sensor data was recorded into rosbags, and is publicly available in the *dynamic_robot_localization_tests* repository[4] along with all the detailed results, configurations and experiments videos, in order to allow future comparisons with other localization systems.

The hardware specifications of the LIDARs used is presented in Table 1 and the laser points after downsampling and registration can be seen on the movement paths figures as green dots (inliers) and red dots (outliers).

3.1.1. Jarvis platform

The Jarvis platform was used to create 4 tests (two navigation paths with two different sets of movement velocities) within a RoboCup field in order to test the 3 DoF DRL system with a long range LIDAR. It is an autonomous ground vehicle equipped with a SICK NAV 350 laser for self-localization (mounted about 2 meters from the floor) and a SICK S3000 laser for collision avoidance (mounted about 0.20 meters from the floor). It uses a tricycle locomotion system with two back wheels and a steerable wheel at the front.

In Figure 5 the robot is performing a delivery task with the package on top of a moving support.

The 3 DoF ground truth was provided by the SICK NAV350 system and relied on 6 laser reflectors (with 9 cm of diameter) to perform the pose estimations (it is certified for robot docking operations with precision up to 4 millimeters).

[4] `https://github.com/carlosmccosta/dynamic_robot_localization_tests`

3.1.2. Pioneer 3-DX platform

The Pioneer 3-DX shown in Figure 6 was used in 4 tests of the dataset presented in [31] in order to test the 3 DoF DRL system in a challenging environment (industrial hall). It is a small lightweight robot equipped with a SICK LMS-200 laser (mounted about 48 cm from the floor) and a Kinect (mounted about 78 cm from the floor). It uses a two-wheel two-motor differential drive locomotion system and can reach a linear speed of 1.2 m/s and angular velocity of 300°/s.

The 3 DoF ground truth was provided by 8 Raptor-E cameras [5] and according to [31] it had less than 1 cm in translation error and less than 0.5 degrees in rotation error.

Figure 5: Jarvis testing platform

Figure 6: Pioneer 3-DX testing platform [31]

3.1.3. Guardian platform

The Gazebo simulation of the Guardian platform was used to create 8 tests in 4 different environments within a structured environment in order to test the 3 DoF DRL system with perfect ground truth. The Guardian platform is an autonomous mobile manipulator equipped with a Hokuyo URG-04LX laser in the front and a Hokuyo URG-04LX_UG01 laser in the back (both mounted about 0.37 meters from the ground). The front laser had a tilting platform which allows 3D mapping of the environment. The arm is a SCHUNK Powerball LWA 4P and in Figure 7 it is attached to a stud welding machine (in simulation it is attached to a video projector). It uses a differential drive locomotion system and can be moved with wheels or with tracks. This platform didn't have a certified ground truth and as such, the results could not be quantified with an external localization system (the results performed with the Gazebo simulator will be presented instead - robot model shown in Figure 8).

Figure 7: Guardian testing platform

Figure 8: Guardian Gazebo simulation

[5] http://www.motionanalysis.com/html/movement/raptore.html

3.2. Testing environments

The localization system was tested in 4 different environments and used the Jarvis platform in a large room with a RoboCup field, the Pioneer 3-DX in a large industrial hall, the Guardian platform in a simulated indoor environment and a Kinect in a flying arena.

3.2.1. Jarvis in RoboCup field

The RoboCup field (shown in Figure 9) occupies half of a large room (with 20.5 meters of length and 7.7 meters of depth). It has two doors, several small windows and two large glass openings into the hallway. Several tests were performed with the robot at speeds ranging from 5 cm/s to 50 cm/s in this environment. These tests were performed with two different movement paths. The first is a simple rounded path that aimed to test the robot in the region of space that had better ground truth (due to its position in relation to the laser reflectors). The second path was more complex and contained several sub paths with different velocities and shapes (it was intended to evaluate the localization system with typical movements that mobile manipulators require, such as moving forward and backwards with or without angular velocity and stopping at the desired destination).

Figure 9: Jarvis testing environment

3.2.2. Guardian in structured environment

The structured environment simulated in Gazebo is a large room with 12.4 meters of length and 8.4 meters of depth. It has 4 doors, several small windows and the walls have small ledges at regular intervals.

Given that the Guardian mobile manipulator is expected to work on the walls of this environment, several tests were de-

Table 1: LIDARs hardware specifications

Laser model	*Range (meters)*	*Field of view (degrees)*	*Scanning frequency (Hz)*	*Angular resolution (degrees)*	*Statistical error (millimeters)*
SICK NAV350	[0.5..250]	360	8	0.25	15
SICK S3000	[0.1..49]	190	8	0.25	150
SICK LMS200	[0.1..80]	180	10	1.0	35
Hokuyo URG-04LX	[0.06..4.095]	240	10	0.36	10
Hokuyo URG-04LX_UG01	[0.02..4]	240	10	0.36	30

vised with a path following the lower and right wall of the environment.

The first test was done in a static environment clear of unknown objects (shown in Figure 10) and was meant to evaluate the best precision that the localization system could achieve.

The second test was done in a cluttered environment (top of Figure 11) and was designed to test the robustness of the localization system against static unknown objects that were placed in the middle of the environment and close to the walls (to block sensor data from reaching known positions and analyze the robustness of matching unknown points).

In the third test (middle of Figure 11) it was added a large block wall (6.5 meters of width, 1.2 meters of height and 0.1 meters of thickness) very close (0.3 meters) to the CAD reference wall to evaluate the impact of the kd-tree search radius in the robustness of the matching algorithms.

In the last test (bottom of Figure 11) it was added a moving car to the second test environment with the objective of assessing the impact of dynamic objects on the point cloud registration algorithms.

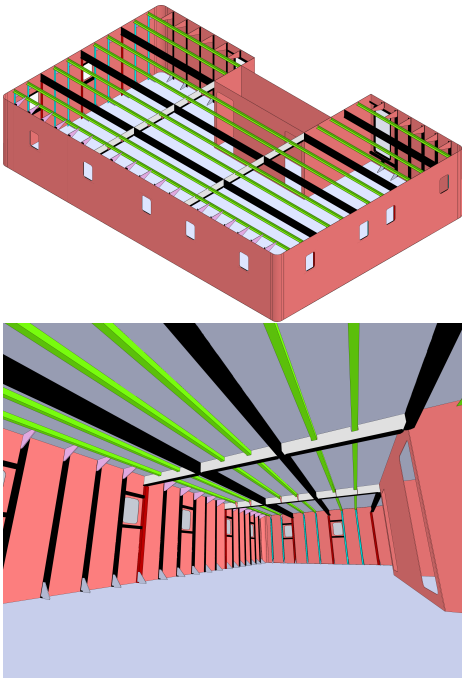


Figure 10: Guardian testing environment

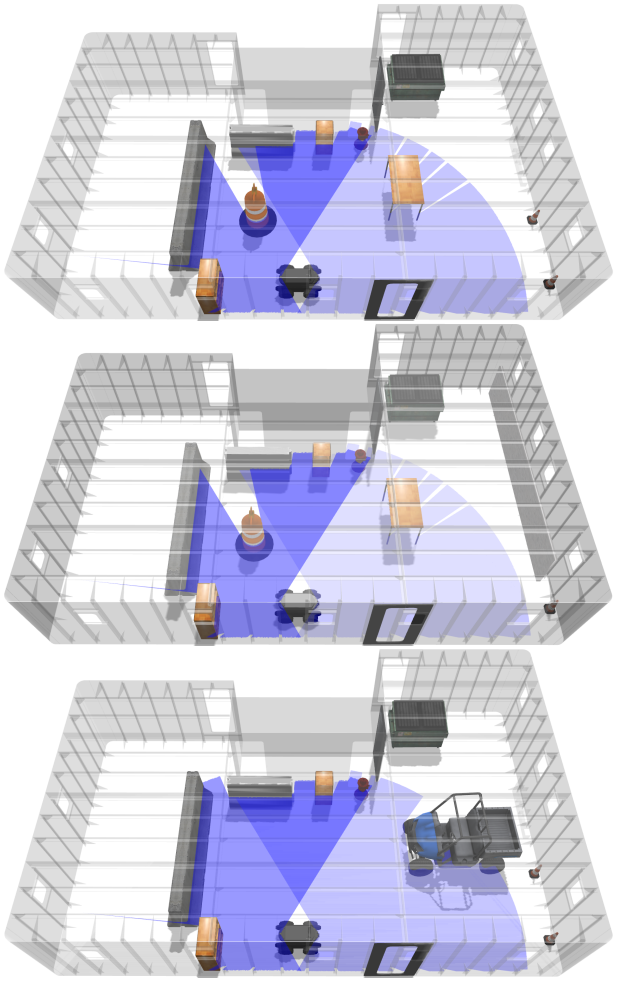


Figure 11: Guardian cluttered (top), cluttered with occluded wall (middle) and dynamic (bottom) testing environments

3.2.3. Pioneer in industrial hall

One of the environments of the dataset presented in [31] is an industrial hall consisting of a large room with 20 meters of length and 12 meters of depth. Four tests were performed with a Pioneer 3-DX in this environment. The first test was a 360º path with a few camera supports in the middle of the room, while the remaining 3 tests were done with several tables and objects spread around in the middle of the room, that significantly reduced the field of view of the robot laser (as can be seen in Figure 12).

Figure 12: Industrial hall with (bottom) and without (top) objects in the center [31]

3.2.4. Kinect in flying arena

The flying arena dataset introduced in [22] and shown in Figure 13 is a large room in which several objects were added in order to test 6 DoF pose tracking. In these tests, the Kinect was moved by the operator in three different paths. The first was a smooth fly movement over the testing scene, while the other two aimed to test paths with mainly translations and rotations. This environment had a ground truth provided by Vicon cameras[6] and according to the authors of the dataset [22], it had sub-centimeter accuracy.

Figure 13: Flying arena environment [22]

[6] http://www.vicon.com/

4. 3 DoF localization system tests

4.1. Overview

The main results of the 3 DoF tests performed with the DRL system (performing localization only with a given initial pose) are presented in tables 2 and 3. They summarize each test by fitting a normal distribution to each evaluation metric. They were retrieved with a known initial pose and used ICP point-to-point as tracking algorithm. The Jarvis and Pioneer tests had a map built using the localization system in mapping mode and were manually corrected to achieve a resolution of 10 and 25 mm respectively (shown in Figure 41 and Figure 32 respectively). The Guardian tests relied on a map built from the CAD model with resolution of 2 mm (shown in Figure 37).

Sensor data preprocessing relied on a voxel grid of 50 mm in order to reduce the impact of sensor measurement noise and also control the level of detail of the live point clouds.

The tests with the initial pose estimation subsystem used SIFT for keypoint selection, FPFH for keypoint description and the feature matching algorithm described in Section 2.5.1 to estimate the initial position and orientation of the robot (Figures 25 and 26 show the accepted initial poses computed by the DRL system when it received as initial pose the large red arrows).

The next sections will provide a brief analysis of the main 3 DoF results achieved with the DRL system. They will start by explaining how laser spherical interpolation can improve localization by mitigating point cloud deformation. Later on it will be analyzed the point cloud preprocessing stage and why it should be carefully tuned to the sensors and ambient geometry. Next it will be presented a global analysis of the test results, in which the pose estimation accuracy achieved by the DRL system will be compared with the ground truth, odometry and the Adaptive Monte Carlo Localization (AMCL) ROS package. Finally it will be analyzed the mapping results and why it is necessary to have accurate environment representations in order to achieve precise localization.

4.2. Laser assembly with spherical linear interpolation

The localization system was designed to operate with any kind of point cloud sensors. For the particular case of data retrieved using LIDARs, there is a very significant problem of point cloud deformation when the robot is moving or rotating at high speeds. This is due to the fact that a typical LIDAR outputs laser scans at a very low rate (8-10 Hz), and as such, assuming that the robot isn't moving when capturing an entire scan will result in deformed point clouds.

Given that ROS outputs laser scans (an entire slice of laser measurements from the start to the end of its field of view), and the localization system doesn't have access to individual measurements as they arrive, then one way to mitigate this scan deformation is by using odometry and / or Inertial Measurement Unit (IMU) information to update the robot pose between the pose corrections performed by the localization system. This allows to use spherical linear interpolation when converting from the raw measurements in polar coordinates into the required Cartesian coordinates in the map frame.

Table 2: 3 DoF DRL localization test results

Test conditions								Translation error (mm)		Rotation error (degrees)		Outliers % [0..100]		Processing time (ms)	
Platform	Ambient	Path shape	Path velocities	kd-tree search radius (mm)	Map cell resolution	Spherical linear interpolation	Nº scans / Nº lasers	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation
Jarvis robot	Cluttered	Rounded	5 cm/s	0.2	10 mm	Yes	1-4/1	4.281	2.264	0.441	0.070	24.35	5.17	11.776	12.265
			30 cm/s	0.2	10 mm	Yes	1-4/1	12.250	8.901	0.535	0.414	16.08	3.80	21.295	8.352
	Cluttered dynamic	Complex	5 cm/s	0.2	10 mm	Yes	1-4/1	4.280	2.264	0.441	0.070	24.35	5.17	11.776	12.265
			5 cm/s	0.2	10 mm	No	1-4/1	4.957	2.473	0.447	0.087	24.34	5.17	11.656	12.997
			50-30-50-10 cm/s	0.2	10 mm	Yes	1-4/1	6.422	3.992	0.397	0.099	23.84	5.59	13.376	13.316
			50-30-50-10 cm/s	0.2	10 mm	No	1-4/1	8.884	8.740	0.422	0.142	24.14	6.18	14.830	23.091
Pioneer robot	Cluttered dynamic	360º	23 cm/s	0.2	25 mm	Yes	1/1	20.105	10.814	5.704	0.662	11.71	3.37	5.046	2.106
		SLAM 1	26 cm/s	0.2	25 mm	Yes	1/1	22.434	12.052	5.429	0.676	4.31	3.84	4.569	2.530
		SLAM 2	19 cm/s	0.2	25 mm	Yes	1/1	19.391	12.568	5.518	0.818	5.09	4.11	4.650	2.180
		SLAM 3	16 cm/s	0.2	25 mm	Yes	1/1	17.480	8.895	5.455	0.732	4.15	4.26	4.388	2.526
Guardian simulator (Gazebo)	Static	Wall follower	5 cm/s	0.05	2 mm	Yes	2/2	2.642	0.463	0.016	0.023	0.0	0.0	5.723	1.868
			30 cm/s	0.15	2 mm	Yes	2-4/2	4.654	3.893	0.044	0.070	0.0	0.0	7.784	3.178
	Cluttered		5 cm/s	0.02	2 mm	Yes	2/2	2.912	0.856	0.020	0.026	25.05	13.30	11.889	9.084
			30 cm/s	0.2	2 mm	Yes	2-4/2	6.092	3.834	0.096	0.085	25.65	11.02	23.925	27.959
	Cluttered dynamic		5 cm/s	0.05	2 mm	Yes	2/2	4.593	3.802	0.069	0.072	33.33	9.88	22.217	32.408
			30 cm/s	0.2	2 mm	Yes	2-4/2	5.897	4.452	0.095	0.104	27.28	13.04	24.409	33.459
			5 cm/s	0.15	2 mm	Yes	2-4/2	12.520	11.100	0.128	0.125	44.18	11.08	10.303	4.865
	Occluded wall		5 cm/s	0.50	2 mm	Yes	2-4/2	237.481	81.445	0.598	0.561	40.60	8.08	19.096	12.670
			30 cm/s	0.15	2 mm	Yes	2-4/2	19.471	23.524	0.144	0.132	46.94	10.36	12.240	6.045
			30 cm/s	0.50	2 mm	Yes	2-4/2	218.534	91.395	0.569	0.423	42.88	8.19	22.170	15.038

Table 3: 3 DoF odometry and AMCL localization test results

Test conditions					Odometry translation error (mm)		Odometry rotation error (degrees)		AMCL translation error (mm)		AMCL rotation error (degrees)	
Platform	Ambient	Path shape	Path velocities	Map cell resolution	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation
Jarvis robot	Cluttered	Rounded	5 cm/s	10 mm	25.353	10.075	0.352	0.206	51.417	20.829	0.442	0.174
			30 cm/s	10 mm	103.676	50.992	2.173	1.916	81.368	36.548	1.388	1.085
	Cluttered dynamic	Complex	5 cm/s	10 mm	21.506	11.937	0.535	0.360	64.300	13.750	0.316	0.214
			50-30-50-10 cm/s	10 mm	124.269	79.775	2.723	1.496	84.230	29.134	0.595	0.581
Pioneer robot	Cluttered dynamic	360º	23 cm/s	25 mm	482.439	163.657	12.810	4.468	135.442	66.445	6.462	1.290
		SLAM 1	26 cm/s	25 mm	370.283	163.734	8.351	2.825	111.470	46.783	6.496	1.386
		SLAM 2	19 cm/s	25 mm	686.983	361.666	11.333	5.423	95.537	47.560	6.957	2.294
		SLAM 3	16 cm/s	25 mm	295.714	151.793	8.858	3.331	108.690	52.341	6.463	1.363
Guardian simulator (Gazebo)	Occluded wall	Wall follower	5 cm/s	2 mm	590.541	486.260	2.610	2.779	538.565	439.726	0.779	1.097
			30 cm/s	2 mm	529.209	386.532	1.056	1.309	464.510	356.744	0.215	0.167

The laser scan deformation is negligible when the robot is moving very slowly. However, when traveling at higher speeds, the point cloud deformation poses a serious issue for map matching algorithms, and the usage of spherical linear interpolation is very important to mitigate it.

Figures 14 and 16 were retrieved using the Jarvis robot with velocities of 50-30-50-10 cm/s and without using the spherical linear interpolation module. Figures 15 and 17 were taken from the same test but now using the spherical linear interpolation module. In the top and bottom of Figure 14 it can be seen that the laser measurements in front of the robot were deformed outwards, causing the points to be projected outside both walls. On the top right corner of Figure 16 there is also a very large deformation, which is now very noticeable, because the first laser measurement isn't close to the last one. These deformations were severely diminished when the laser spherical linear interpolation module was used (as can be seen in figures 15 and 17 respectively.

Analyzing the tests in table 2 associated to the figures 14 to 17 it can be seen that using the spherical linear interpolation module allowed to reduce the mean (from 8.88 mm to 6.42 mm) and standard deviation (from 8.74 mm to 3.99 mm) of the translation error and also reduced the mean (from 0.42º to 0.39º) and standard deviation (from 0.14º to 0.10º) of the rotation error. Moreover, given that the correction of deformation gives a point cloud more similar to the map, the usage of the spherical linear interpolation module also allowed to reduced the global computation time (mean from 14.83 ms to 13.38 ms and standard deviation from 23.09 ms to 13.32 ms).

The laser assembly parameters present in tables 2 and 3 as "Nº scans / Nº lasers" have the meaning of how many laser scans were being merged into each published *sensor_msgs::PointCloud2* and from how many sensors those lasers scans were being generated. As such, a configuration of "2-4/2" means that the laser assembler was merging between 2 (moving fast) and 4 (moving slow) lasers scans depending on the velocity of the robot, and these laser scans were being generated by two LIDARs sensors (one in the front and another in the back of the robot).

Figure 14: Large laser deformation on opposite walls (top and bottom) when the robot is rotating (grid with 1 m spacing)

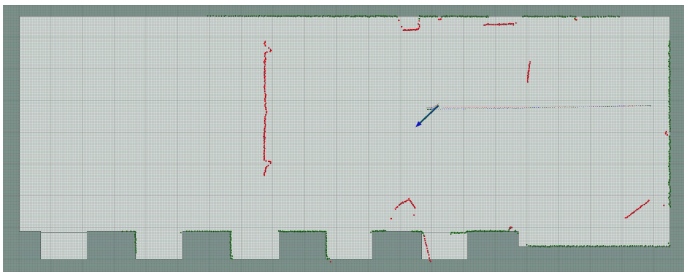


Figure 15: Correction of the large laser deformation on opposite walls (top and bottom) with spherical linear interpolation (grid with 1 m spacing)

Figure 16: Laser deformation on the top right corner when the robot is rotating (grid with 1 m spacing)

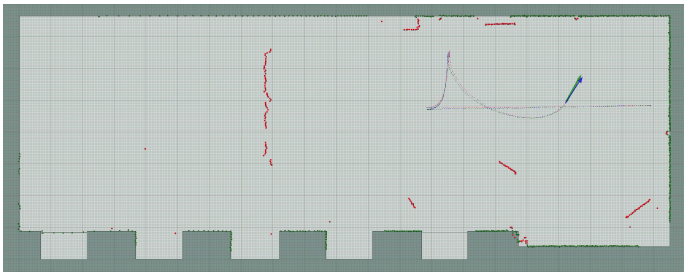


Figure 17: Correction of the laser deformation on the top right corner with spherical linear interpolation (grid with 1 m spacing)

4.3. Point cloud preprocessing

The statistical error of the lasers can be mitigated by merging several laser scans before performing a point cloud registration. This is based on the fact that a considerable amount of laser noise results in measurements oscillating around the true distance to the obstacles, and as such, the effective measurement error can be significantly reduced by retrieving the centroids of a voxel grid applied over the laser data. However, assembling too much laser scans can lead to worse pose tracking if the robot is moving at high speeds, because the lasers would be projected into the map frame with a high position error (because odometry accuracy decreases when the robot speed increases and the localization system will not correct it, since the sensor data is still being assembled and not being used to estimate the robot pose). As such, the implemented laser assembler supports dynamic reconfiguration of the number of laser scans to assemble (or the period of assembly time) based on the robot estimated velocity. This feature besides improving the pose tracking, it also allows a navigation supervisor to regulate the rate at which

the localization system operates. This can be very useful for mobile robot manipulators because the localization system can have a high update rate when the robot is moving fast and a low update rate when it is moving slower or when it is stopped. This allows a more efficient usage of the platform hardware resources while also improving the localization system accuracy.

Besides reducing the sensor measurement errors, this processing stage also allows the removal of laser shadow points caused by veiling, or small cluster of points that may be associated with temporary objects / people. However these filters can take some time to compute (given the intensive usage of point neighbors searches).

For typical usage scenarios of a mobile robot manipulator, the voxel grid and random sampling filters are usually enough to control the computational resources that will be required by the registration algorithms while also improving robustness against sensor noise.

4.4. Point cloud registration

Looking at the results from tables 2 and 3, the poses from Figure 22, the laser assembly figures shown in figures 23 and 24 and the translation / rotation error graphs presented from figures 27 to 30 (from the Jarvis test in the complex path at high velocities), it can be seen that the localization system can register point clouds with much more accuracy than the AMCL ROS package (AMCL was using at most 5000 particles). For the test shown in figure 22, the DRL system was 13.12 times more accurate than the AMCL package in computing the robot position (achieved 6.422 mm of mean translation error while AMCL had 84.23 mm) and was also 1.5 times more accurate than the AMCL package in computing the robot orientation (achieved 0.397^{o} of mean rotation error while AMCL had 0.595^{o}).

The most common problems that seemed to affect the registration algorithms of the DRL system were the point cloud deformation (usually when there is unreliable odometry information, which typically occurs when the robot is moving with high velocities / accelerations), laser measurements errors, unknown objects close to the known reference point cloud and also low resolution maps. The localization system can tolerate these problems by having a default pipeline configuration for the normal operation of the robot, another for temporary tracking recovery and yet another for initial pose estimation.

The switch between the localization system operation modes using the point cloud registration analysis proved to be a very intuitive and fast process to setup and the overall system configurations seemed to be generic enough to be reused in several types of environments, with different robots equipped with varying types of sensors. Such ease of configuration allows the fast deployment of robots and gives a high confidence that the DRL system will remain accurate even in challenging environments. This is not the case with systems such as AMCL, given that they are very dependent on the odometry and laser models, and as such, require constant retuning when one of these models change.

The ability to switch registration algorithms at runtime based on the quality of the estimated pose proved to be an efficient and flexible architecture choice, since it allowed high precision pose tracking with fast registration algorithms and occasional pose tracking recovery with more robust methods / configurations (which happened more often in the tests at higher velocities, in which the odometry was significantly worse).

The recovery configuration is also very useful to adjust the search radius of the ICP algorithms. This parameter can be tuned to avoid wrong correspondences of points from objects close to the reference map, (such as the new wall in front of the reference CAD walls in the middle of Figure 11). Figure 18 is an example of correctly registered point clouds done by the DRL system even when there was a large wall close to the reference map (shown in Figure 37). This was achieved with the ICP point-to-point registration algorithm with a kd-tree search radius of 15 cm, which rejected the point correspondences between the new wall and the reference wall. As can be seen in Figure 20 and Table 2, the DRL system was much more accurate than the AMCL package in this challenging test (achieved a mean translation error of 19.47 mm while AMCL had 218.53 mm). Figure 19 shows the same test but now with a kd-tree search radius of 50 cm. This larger search radius caused the large wall to distort the cloud registration (green dots are considered correctly registered points and red dots are incorrectly registered points) and introduce an offset in the pose estimation (as can be seen in Figure 21). These tests show that a small kd-tree search radius is ideal to track the robot pose in environments with objects close to the reference map. However by restricting the point neighbors search radius, the matching algorithms become much less robust against large errors in odometry, since the initial guess given to the ICP algorithms may cause the correct point correspondences to be rejected since the distance between the live points and the corresponding reference points might be larger than the limited kd-tree search radius, causing the point cloud registration to fail. As such, the ability to have a recovery algorithm with a larger kd-tree search radius allows the DRL system to recover from temporary tracking problems while the main registration algorithm achieves high accuracy tracking even when there are objects close to the known reference points of the map.

The initial pose estimation using feature matching was very reliable and successfully found the robot location even in low feature surroundings (as can be seen in figures 25 and 26). Moreover, the output of the accepted initial pose estimations allows the detection of similar map locations by a navigation supervisor, which can then plot a path to disambiguate the initial pose guess and avoid dangerous operations at a wrong position.

Figures 25 and 26 show the accepted initial pose estimations using the global localization subsystem presented in section 2.5.1. The blue dots presented in figures 25 and 26 represent the keypoints of the live point cloud in the robot start up position, while the violet dots are the reference point cloud keypoints. The green dots are the live point cloud laser measurements after performing the initial pose estimation and registration refinement.

Figure 18: ICP point to point cloud registration performed by the DRL system with a 15 cm kd-tree search radius (grid with 1 m spacing)

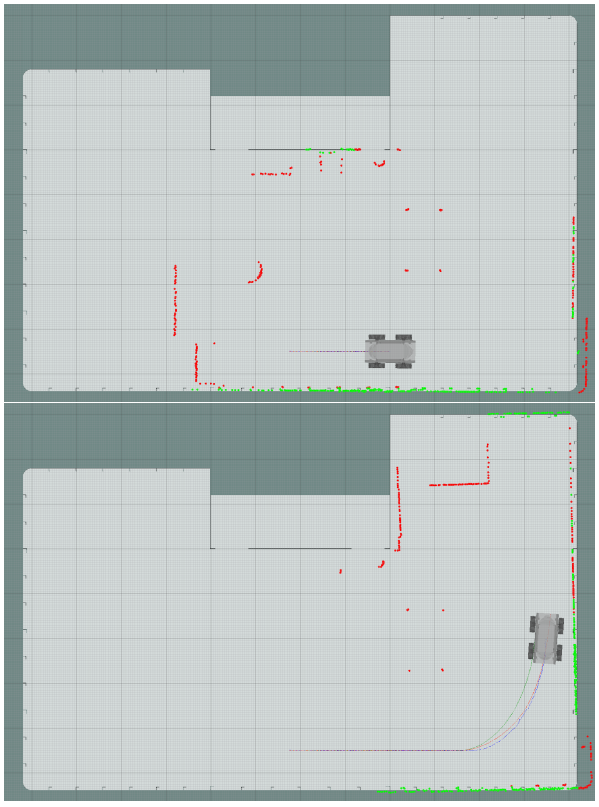


Figure 19: ICP point to point cloud registration performed by the DRL system with a 50 cm kd-tree search radius (grid with 1 m spacing)

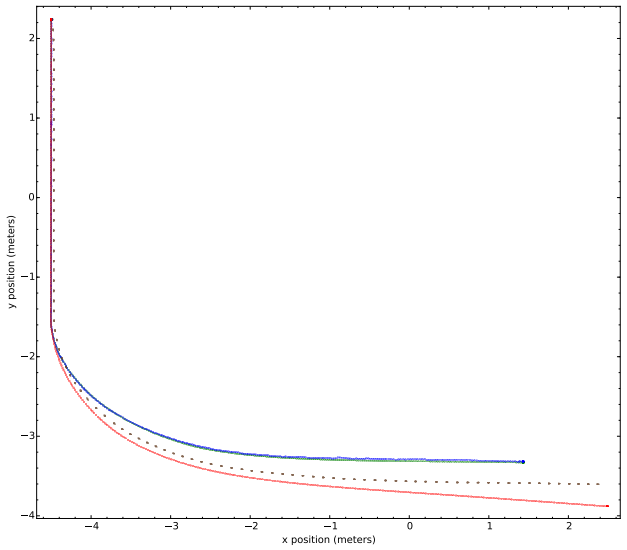


Figure 20: Poses estimated by the ground truth (green arrows), DRL system (blue arrows), AMCL (brown arrows) and odometry (red arrows) in the occluded wall test with the robot moving at 30 cm/s and using a kdtree search radius of 15 cm

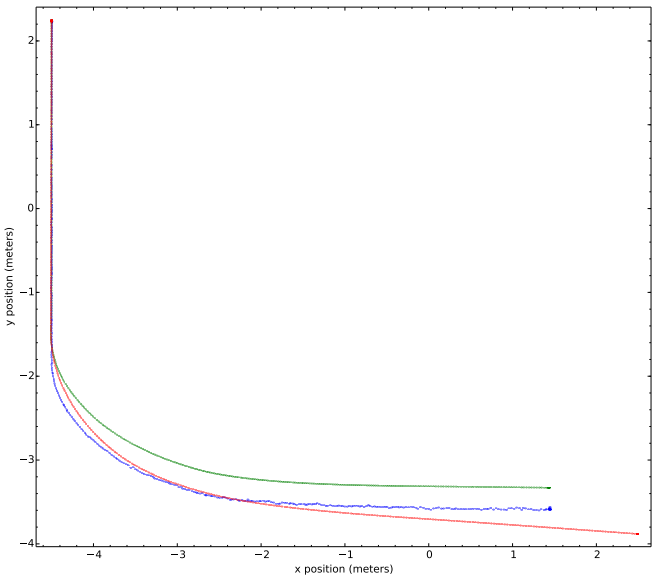


Figure 21: Poses estimated by the ground truth (green arrows), DRL system (blue arrows) and odometry (red arrows) in the occluded wall test with the robot moving at 30 cm/s and using a kdtree search radius of 50 cm

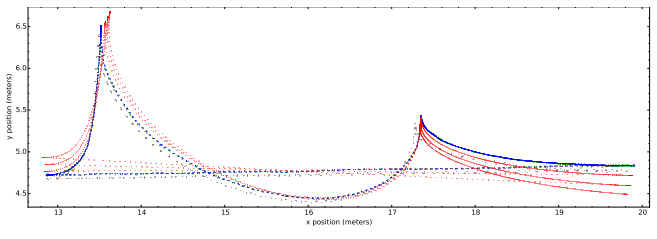


Figure 22: Poses estimated by the ground truth (green arrows), DRL system (blue arrows), AMCL (brown arrows) and odometry (red arrows) in the Jarvis test at 50-30-50-10 cm/s movement velocities

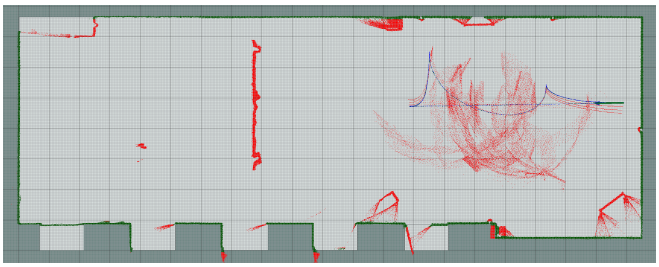


Figure 23: Laser scans assembled on top of the map using the DRL system poses in the Jarvis test at 50-30-50-10 cm/s movement velocities (grid with 1 m spacing)

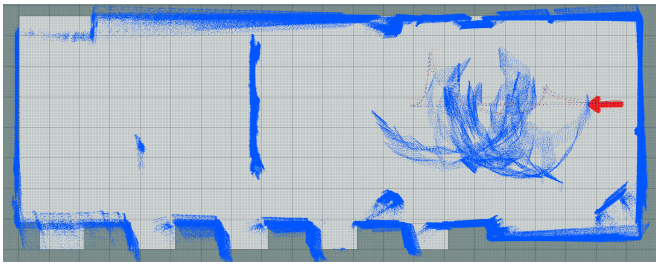


Figure 24: Laser scans assembled on top of the map using the AMCL poses in the Jarvis test at 50-30-50-10 cm/s movement velocities (grid with 1 m spacing)

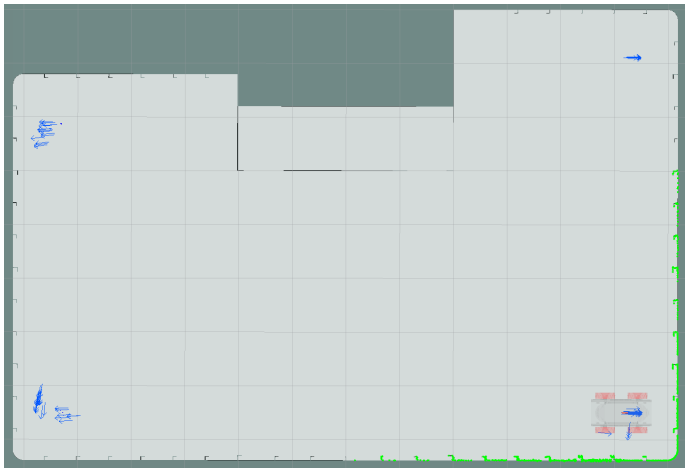
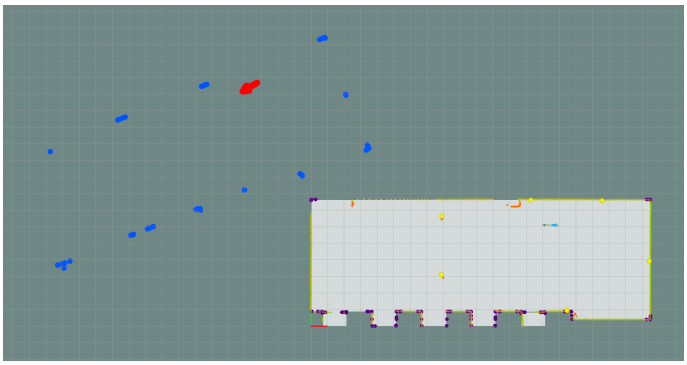


Figure 25: Overview of initial pose estimation using feature matching in the Guardian static environment (grid with 1 m spacing)



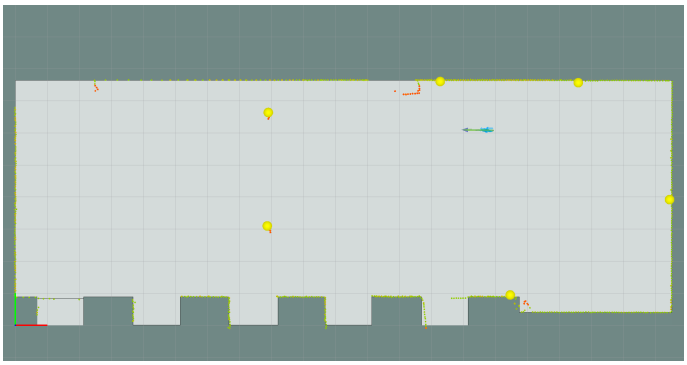


Figure 26: Overview of initial pose estimation using feature matching in the Jarvis environment (grid with 1 m spacing)

4.5. Translation and rotation errors

By analyzing tables 2 and 3 it can be seen that the 3 DoF DRL system can achieve pose tracking with less than 10 mm in translation error and less than 1 degree in rotation error when using a detailed map (10 mm cell resolution) and a sensor with good field of view (360°) and low sensor noise (15 mm). When using a sensor with low field of view (180°) and high sensor noise (35 mm), the DRL system still managed to achieve localization accuracy bellow the map resolution (translation error less than 20 mm and rotation error close to 5 degrees in a map with 25 mm cell resolution).

As expected, the tests at lower velocities had less mean error than the ones at higher speeds while requiring slightly less computation time. Also, the simulator tests had less mean error than the ones performed on the physical platforms. This is due to laser scan deformation that couldn't be simulated and also because the Gazebo simulator was only able to add Gaussian noise to the laser measurements. To compensate these limitations, the error in odometry and laser measurements in simulation was deliberately higher than the expected values for the physical platforms. This explains why the localization system needed more time to perform the pose estimations in the simulator tests. It can also be seen that adding unknown objects increased the mean error and required computation time while adding dynamic objects increased these values even further (given that a moving object appears in a laser scan as a deformed version of its static shape).

These results show that the localization system can reliably achieve high accuracy pose tracking even in dynamic and challenging environments.

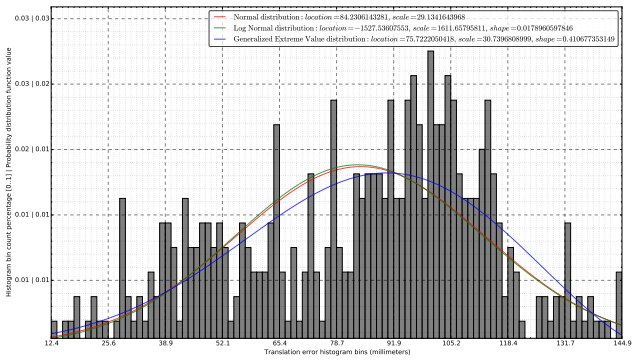


Figure 27: Probability distributions for the AMCL translation errors in the Jarvis test at 50-30-50-10 cm/s movement velocities

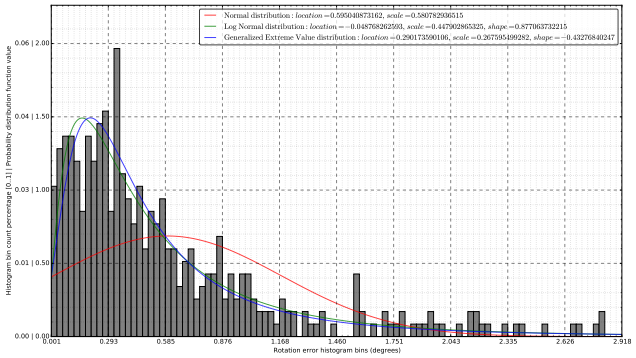


Figure 28: Probability distributions for the AMCL rotation errors in the Jarvis test at 50-30-50-10 cm/s movement velocities

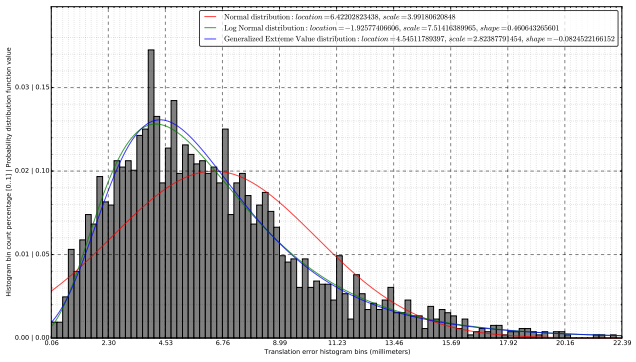


Figure 29: Probability distributions for the DRL system translation errors in the Jarvis test at 50-30-50-10 cm/s movement velocities

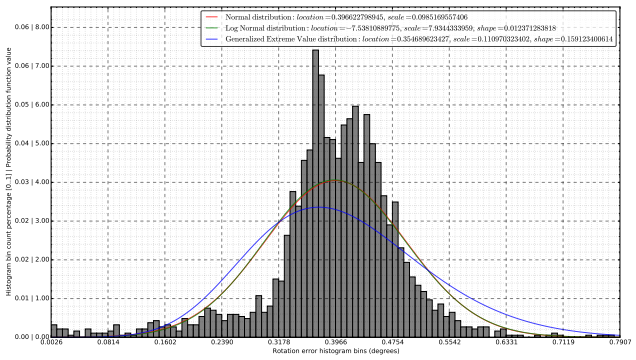


Figure 30: Probability distributions for the DRL system rotation errors in the Jarvis test at 50-30-50-10 cm/s movement velocities

4.6. Computation time

Looking at the computation time graphs in Table 2 and to Figure 31, it can be seen that the localization system can register the point clouds very fast (between 5 and 30 milliseconds), which is low enough to process all the incoming laser scans in real time (typically LIDARs generate scans every 100 milliseconds). Moreover, the computation time seems to be very stable, with occasional peaks due to laser deformation or varying percentage of outliers.

The global computation time is mostly associated with the point cloud registration stage, while the rest of it is due to normal estimation and point cloud preprocessing algorithms.

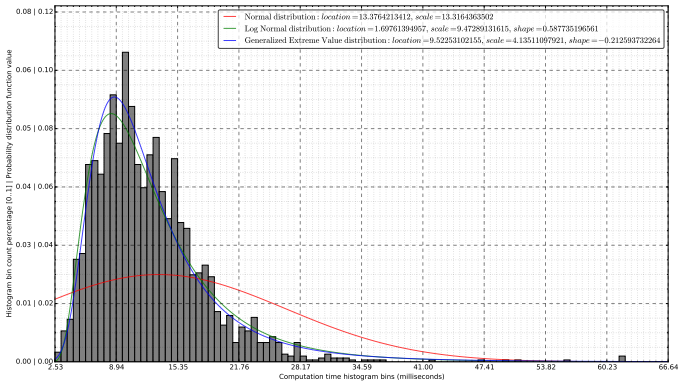


Figure 31: Probability distributions for the DRL system global computation time in the Jarvis test at 50-30-50-10 cm/s movement velocities

4.7. Mapping

Any self-localization system requires a map to estimate the pose of the robot when using exteroceptive information. If such map doesn't exist, then the first live point cloud can be considered as the initial map, and then it can be updated dynamically as new sensor data is processed.

The dynamic and incremental map update capability of the DRL system can be used to register the full sensor point clouds or only the inliers / outliers. Full integration is useful when starting a map from scratch (example in figures 33 and 40). Partial integration can be useful when there is a highly detailed map (for example generated from a CAD model or with other highly accurate mapping system) and we only want to add or remove information from it, such as integrating new large objects and opening doors in the map (example in figure 38). Partial integration besides reducing the computational resources required, it also allows to keep the detail of the original map (by avoiding deformations due to sensor measurement noise). This can be clearly seen in figures 39 and 42 in which the sensor noise polluted the walls of the original map (given that the map cell resolution was much smaller than the mean sensor noise). By performing selective mapping this problem was avoided in the map present in Figure 38 (by inserting on the map only unknown zones while keeping the walls generated from the CAD model untouched).

Looking at planar structures from figures 33 to 36 it can be seen that the DRL system was able to achieve equal or even

better mapping results than the GMapping[7] Simultaneous Localization And Mapping (SLAM) system and even the ground truth provided by the Raptor-E cameras (that seems to be less suitable for mapping than both the DRL system and the GMapping ROS package).

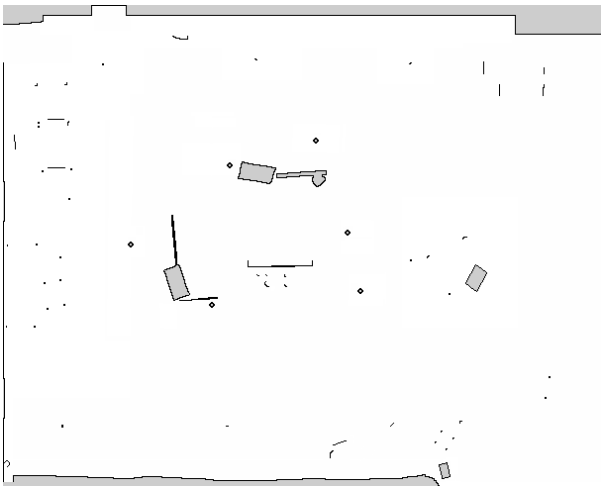


Figure 32: Map manually corrected with 25 mm cell resolution (based on Figure 33)

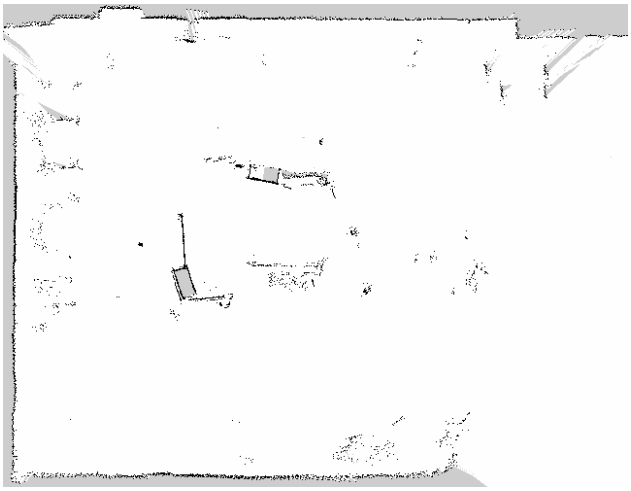


Figure 33: Map made with the DRL system using full integration in conjunction with OctoMap (playing the rosbag in real time)

[7] `http://wiki.ros.org/gmapping`

Figure 34: Map made using the ground truth poses provided by the Raptor-E cameras

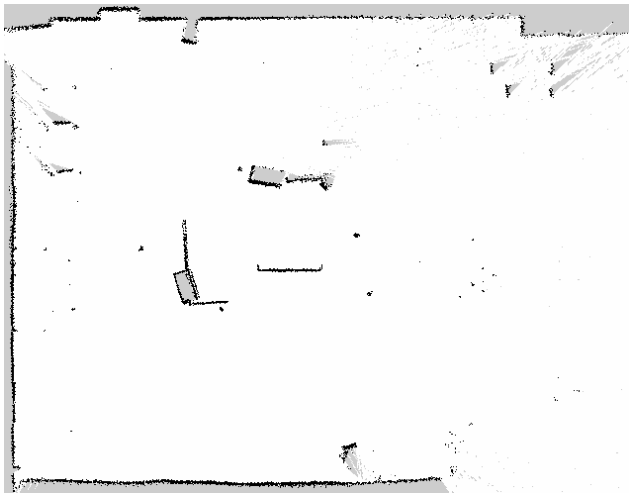


Figure 35: Map made with the GMapping package (playing the rosbag at 10% speed)

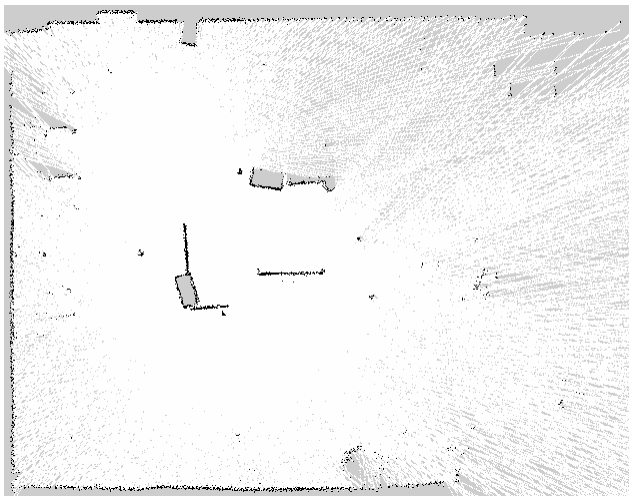


Figure 36: Map made with the GMapping package (playing the rosbag in real time)

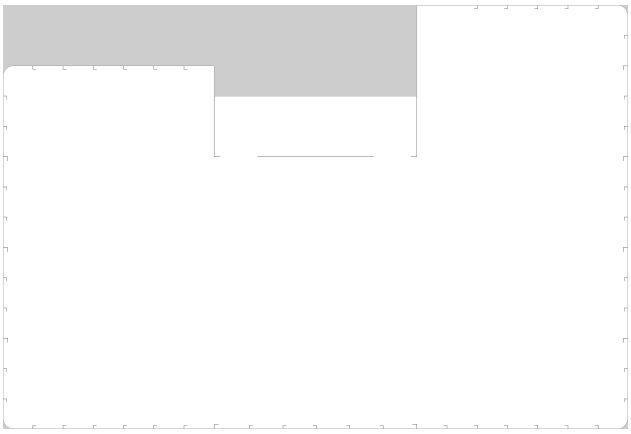


Figure 37: Map of the structured environment generated using a CAD model

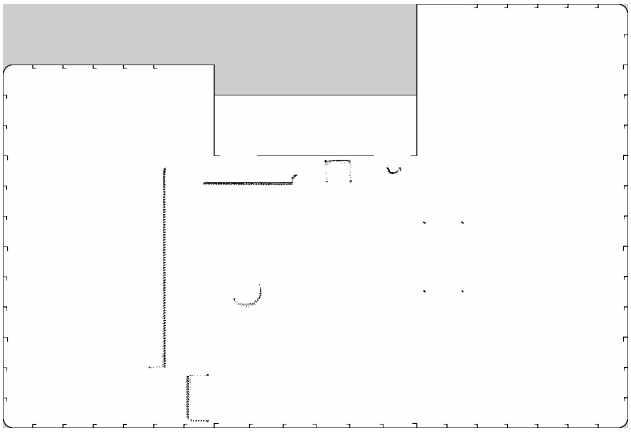


Figure 38: Updated map of the structured environment using the DRL system with partial integration (only points that were not close to the walls of Figure 37 were integrated)

Figure 39: Updated map of structured environment using the DRL system with full integration (all registered points were integrated into the map of Figure 37)

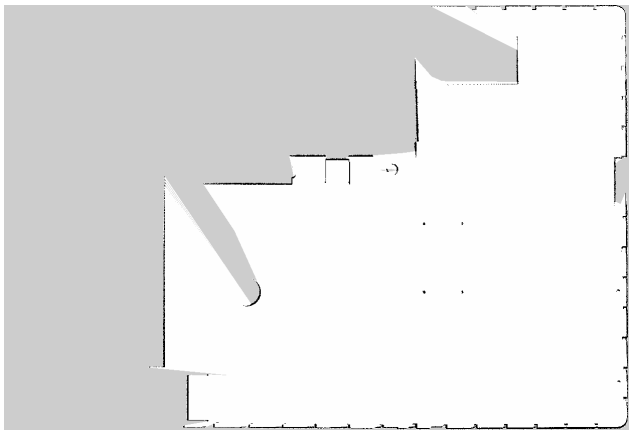


Figure 40: Mapping of the structured environment using the DRL system with full integration (all registered points were integrated)

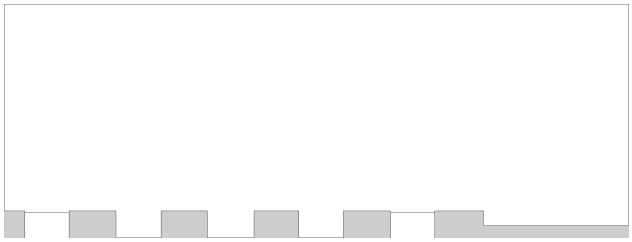


Figure 41: RoboCup field map (manually corrected)

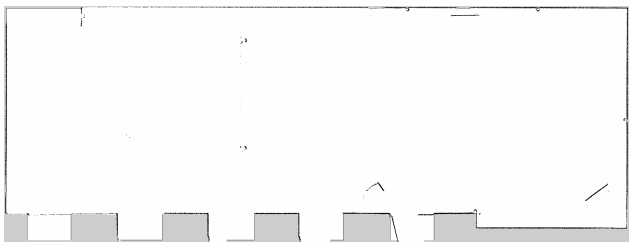


Figure 42: Updated map of the RoboCup field using the DRL system with full integration (starting with the map from Figure 41)

5. 6 DoF localization system tests

5.1. Overview

The main 6 DoF results retrieved with the DRL system (performing localization only with a given initial pose) are shown in Table 4. The first two experiments (fly and translations movements) were performed to evaluate the accuracy of the point cloud registration algorithms while the last one (mainly rotation movements) aimed to test the robustness against temporary absence of valid sensor data (in this test the Kinect had periods in which most of its field of view was outside the map).

These tests were retrieved with a known initial pose and used ICP point-to-point as tracking algorithm and ICP point-to-plane as tracking recovery method. The tests with the ethzasl_icp_mapper used the ICP point-to-plane since the implementation of the ICP point-to-point was consistently losing tracking (the authors of this systems also point out in [21] that

their ICP point-to-plane implementation was more robust and achieved better tracking).

The 3D map was done with the localization system in mapping mode (using the fly test sensor data) and used continuous surface reconstruction in order to create an accurate representation of the environment and reduce the impact of the measurements noise (no manual correction was needed). This map was later downsampled using a voxel grid with 20 mm cells in order to allow real-time processing when performing pose estimation (localization only) of the Kinect sensor.

The next sections will provide an analysis of the 6 DoF results achieved with the DRL system. They will start by explaining the importance of point cloud processing and how it helps reduce the cloud registration time. Then it will be given an analysis of the translation and rotation errors along with the computation time and lastly it will be presented the 3D map building capabilities.

5.2. Point cloud preprocessing

Point cloud preprocessing can have a very significant role when performing 6 DoF cloud registration for self-localization. This is due to real-time requirements and also because the computation time increases substantially when registering large point clouds. One way to control this problem is by preprocessing the point cloud with a voxel grid to adjust the level of detail and also assign a limit to the number of points that come from sensors. This can be achieved with random sampling or similar point selection techniques. Moreover, depending on the sensor used, it may be wise to restrict the points to a given range, and discard the rest that are too far way (given that these measurements will have more errors).

In the tests performed by the DRL system and presented in Table 4, it was applied a voxel grid of 20 mm while keeping only the points from the kinect sensor that were at most at 3 meters in the rotations tests and 2.15 meters in the remaining two tests. Moreover it was applied a random sampling filter in order to limit the number of points to 750 in the rotations tests and to 425 in the remaining two tests. In order to allow a fair comparison in terms of processing time, the tests performed with the ethzasl_icp_mapper also used a random sampling filter with the same limits as the DRL tests.

5.3. Point cloud registration

Looking at Table 4 and figures 43 to 54, it is clear that the DRL system is able to register point clouds with high accuracy, even when they are severely down-sampled (due to real-time processing constraints). Moreover, the localization system is robust against temporary absence of sensor data (when the field of view of the Kinect was outside the known map in the rotations tests) and was able to quickly recover to accurate tracking when valid sensor data was given (analyzing figures 49 to 54 it can be seen that the ethzasl_icp_mapper was less successful in detecting unreliable pose estimations and recovering from then). In these situations the recovery algorithms were activated, switching the registration algorithm from ICP point-to-point to ICP point-to-plane (89 times in 968 Kinect point clouds

Table 4: 6 DoF results

Test conditions				Translation error (mm)		Rotation error (degrees)		Outliers % [0..100]		Processing time (ms)		Cloud registrations	
Path shape	Path velocity	Map cell resolution	System	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	Success ratio	Tracking recovery ratio
Overview fly	30 cm/s	20 mm	DRL	17.926	9.789	3.027	0.638	0.165	0.172	30.714	11.407	473/501	0/501
			ethzasl_icp_mapper	22.810	17.579	3.477	1.958	–	–	324.716	173.477	180/501	–
Mainly translations	20 cm/s		DRL	14.162	7.990	2.486	0.975	0.983	1.874	32.296	10.757	809/926	0/926
			ethzasl_icp_mapper	31.420	52.280	2.915	0.637	–	–	292.265	167.868	358/926	–
Mainly rotations	10 cm/s		DRL	16.576	9.389	2.738	0.583	1.214	3.148	29.818	9.134	541/968	89/968
			ethzasl_icp_mapper	82.010	164.410	3.443	4.451	–	–	516.307	292.159	185/968	–

in the case of the rotations test and 0 on the remaining two tests). Given that these sensor data outages were temporary, the initial pose algorithms weren't necessary. Nevertheless, if the Kinect remained outside the map for a longer period of time, the localization system would alert that the tracking was lost and it would try to find its current pose using feature matching.

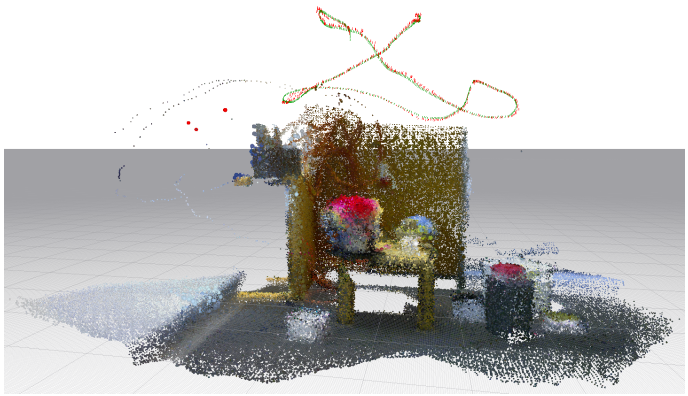


Figure 43: Registered point clouds assembled on top of the map using the DRL system poses (for the top part of the figure: green arrows → ground truth poses, red arrows → DRL poses)

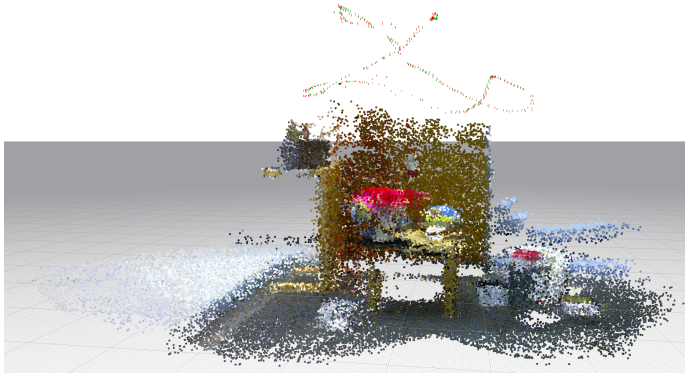


Figure 44: Registered point clouds assembled on top of the map using the ethzasl_icp_mapper system poses (for the top part of the figure: green arrows → ground truth poses, red arrows → ethzasl_icp_mapper poses)

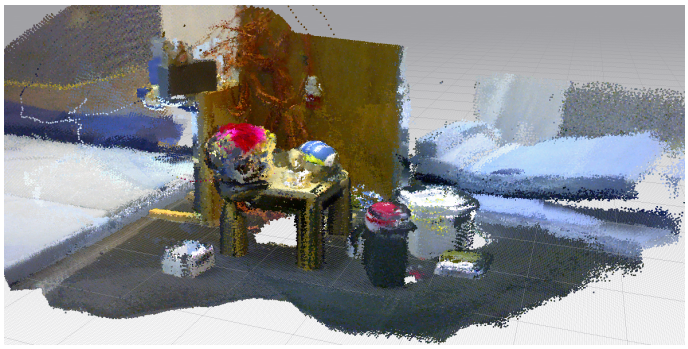


Figure 45: Full Kinect point clouds assembled on top of the map using the ground truth poses

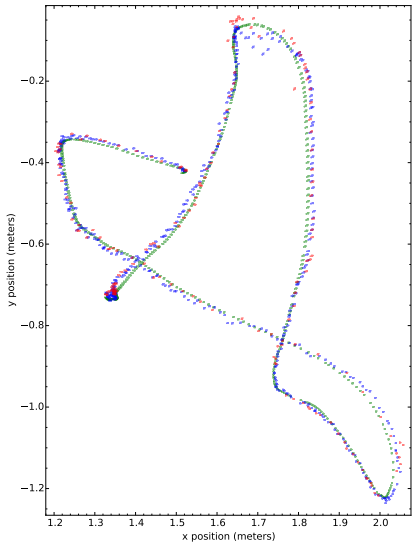


Figure 46: XY 2D plot of poses estimated in the 6 DoF fly test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

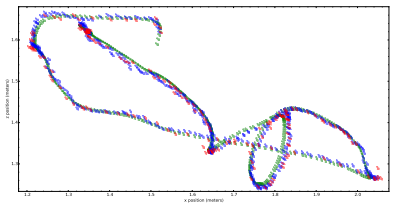


Figure 47: XZ 2D plot of poses estimated in the 6 DoF fly test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

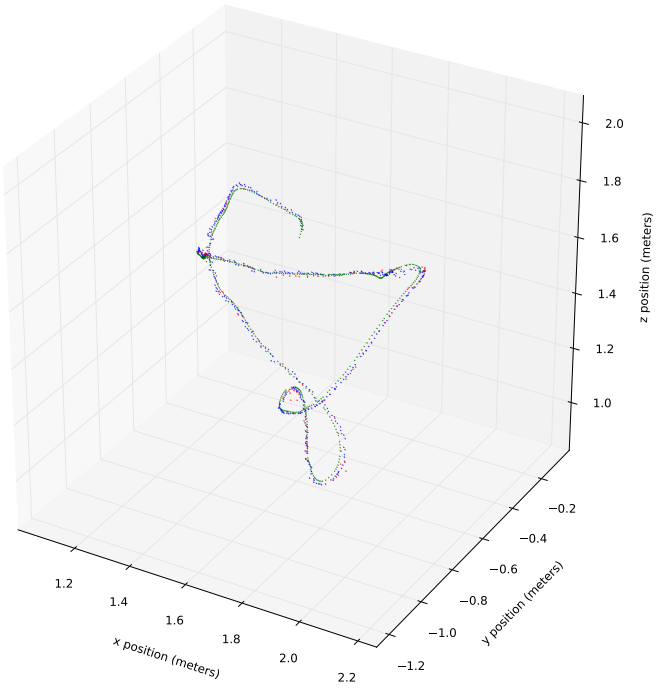


Figure 48: XYZ 3D plot of poses estimated in the 6 DoF fly test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

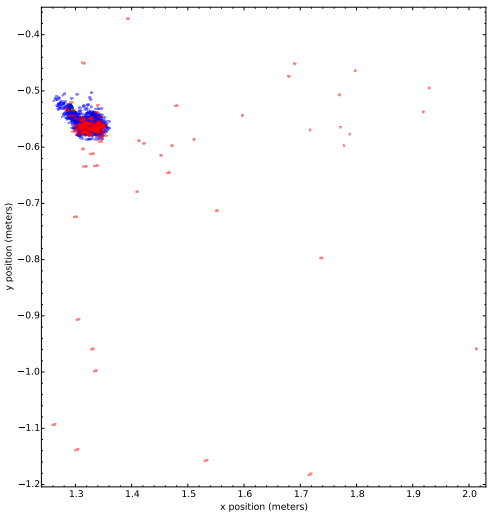


Figure 49: XY 2D plot of poses estimated in the 6 DoF rotations test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

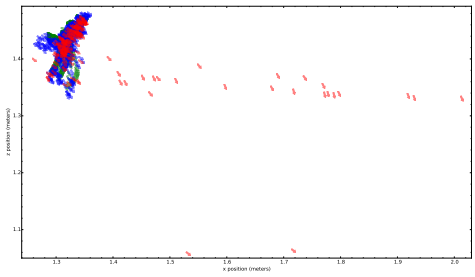


Figure 50: XZ 2D plot of poses estimated in the 6 DoF rotations test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

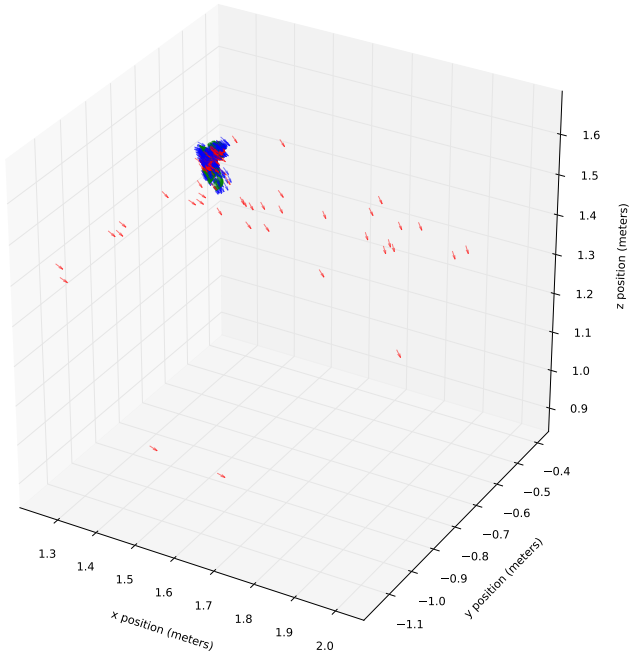


Figure 51: XYZ 3D plot poses estimated in the 6 DoF rotations test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

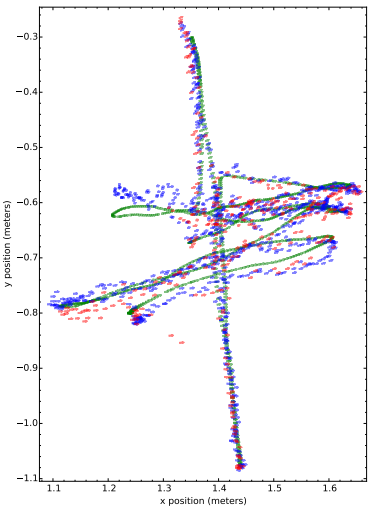


Figure 52: XY 2D plot poses estimated in the 6 DoF translations test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

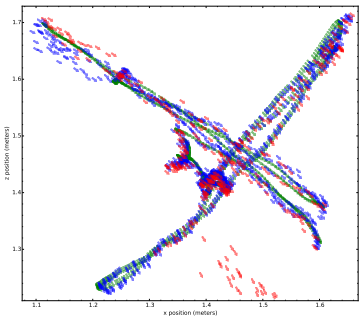


Figure 53: XZ 2D plot of poses estimated in the 6 DoF translations test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

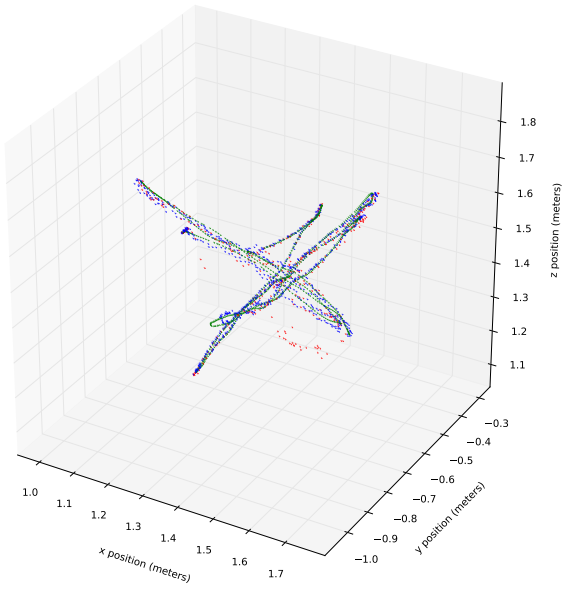


Figure 54: XYZ 3D plot of poses estimated in the 6 DoF translations test by the ground truth (green), DRL (blue) and ethzasl_icp_mapper (red)

5.4. Translation and rotation errors

The results presented in Table 4 and in figures 55 and 56, shown that the DRL system can maintain high accuracy pose tracking, with translation error below two centimeters and rotation error around 3 degrees when good sensor data is available. Moreover it can quickly recover from temporary registration problems caused by occlusions or accelerations.

Analyzing the results in Table 4 and from figures 46 to 58, it can also be concluded that the DRL system achieved better pose tracking than the ethzasl_icp_mapper ROS package (17.9 mm vs 22.8 mm of mean translation error and 3.0º vs 3.5º of mean rotation error in the 6 DoF fly test), while achieving more than double of the refresh rate in the fly test. For the translations and rotations tests, the DRL system achieved much better mean tracking than the ethzasl_icp_mapper system given that it was able to detect much better when the estimated poses were unreliable (Kinect sensor starting to point at unknown map areas). This can be seen in figures 51 and 54 by the large amount of red arrows (ethzasl_icp_mapper poses) far away from the ground truth poses (unreliable estimated poses by the ethzasl_icp_mapper).

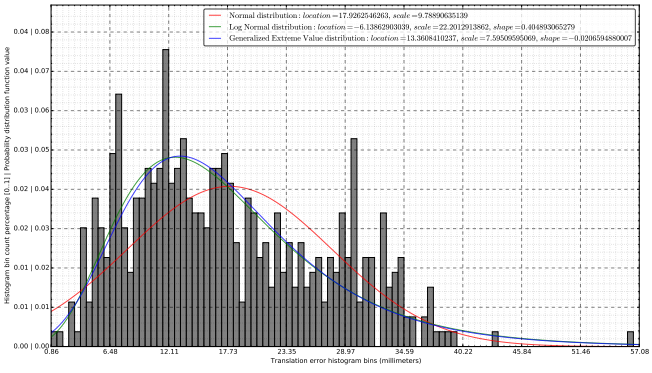


Figure 55: Probability distributions for the DRL translation errors in the 6 DoF fly test

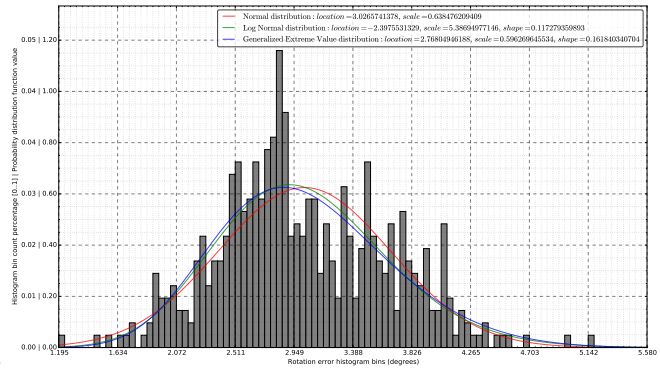


Figure 56: Probability distributions for the DRL rotation errors in the 6 DoF fly test

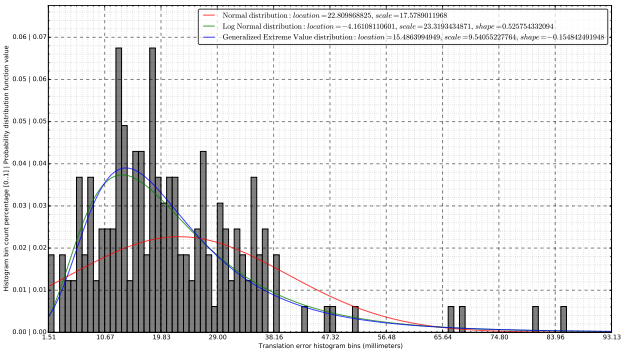


Figure 57: Probability distributions for the ethzasl_icp_mapper translation errors in the 6 DoF fly test

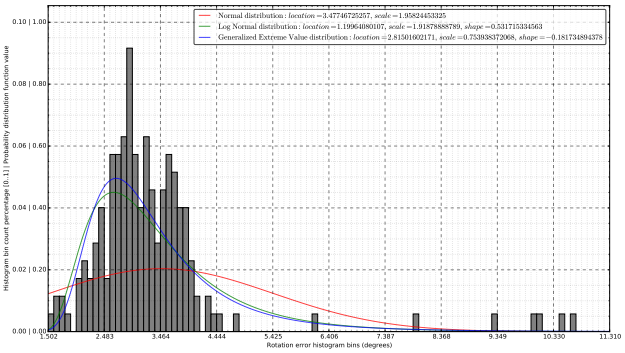


Figure 58: Probability distributions for the ethzasl_icp_mapper rotation errors in the 6 DoF fly test

5.5. Computation time

The localization system was able to achieve a mean computation time low enough to allow real time processing of the Kinect point cloud sensor data (shown in Figure 59). Depending on the accuracy requirements, the computation time can be lowered even further by tuning the preprocessing filters (of both the reference and live point clouds), in order to reduce the number of points used in the cloud registration.

Comparing Figure 59 with Figure 60 it can also be seen that the DRL system required about ten times less processing time in relation to the ethzasl_icp_mapper system (the DRL achieved a mean processing time of 30.71 ms while the ethzasl_icp_mapper required 324.71 ms in the 6 DoF fly test).

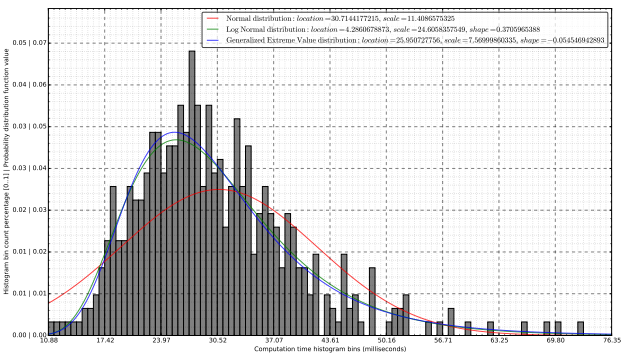


Figure 59: Probability distributions for the DRL global computation time in the 6 DoF fly test

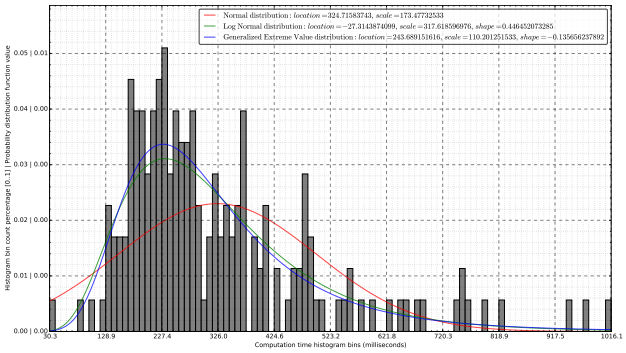


Figure 60: Probability distributions for the ethzasl_icp_mapper global computation time in the 6 DoF fly test

5.6. Mapping

The DRL system can perform mapping of the environment and is able to build very detailed point clouds. These point clouds can be continuously re-sampled with the Moving Least Squares algorithm (example in Figure 62) in order to reconstruct the surfaces of the environment and attenuate the double wall effects due to high sensor noise.

Besides the internal mapping capabilities of the DRL system, it can also be paired with the OctoMap [10] library in order to perform probabilistic integration of sensor data and be able to remove missing objects from the reference point cloud.

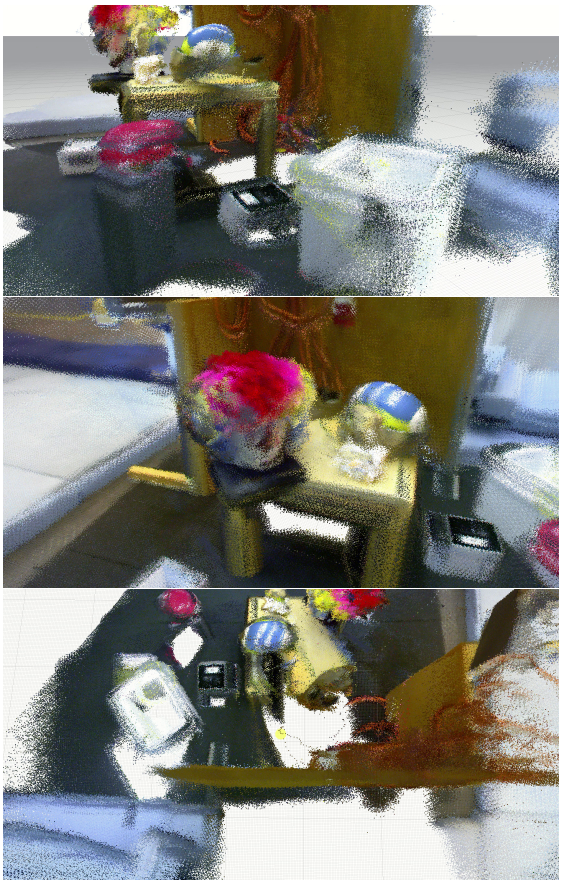


Figure 61: 3D mapping using the ground truth poses

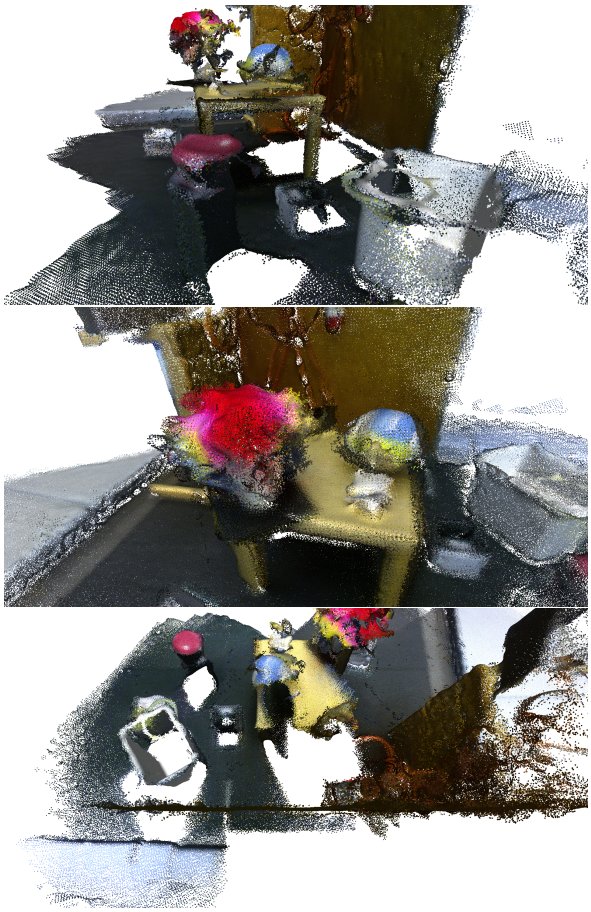


Figure 62: 3D mapping using the DRL system with surface reconstruction

6. Conclusions

The proposed localization system is able to maintain pose tracking with less than 1-2 centimeters of translation error and less than a 1-3 degrees of rotation error (in 3 and 6 DoF respectively) with the robot / sensors moving at several velocities even in cluttered and dynamic environments. Moreover, when tracking is lost or no initial pose is given, the system is able to find a valid global pose estimate by switching to more robust registration algorithms that use feature matching. This approach achieved fast and accurate pose estimation with robust tracking recovery and reliable initial pose estimation while also providing the set of the accepted initial poses before registration refinement, which can be very valuable information for a navigation supervisor when the robot is in an ambiguous region that can be registered in similar zones of the known map. The system also allows dynamic reconfiguration of the number of laser scans to assemble in order to mitigate laser measurement errors and can adapt its rate of operation according to the robot estimated velocity.

The sub-centimeter accuracy achieved by the proposed localization system along with the selective map update capability (using partial or full sensor data integration) and the need of no artificial landmarks / ambient modifications will allow the fast deployment of mobile robots capable to operate safely and accurately in cluttered environments.

Acknowledgments

The authors would like to thank everyone involved in the CARLoS Project. This project has received funding from the European Commission Seventh Framework Programme for Research and Technological Development under the grant agreement number 606363, and from the national project "NORTE-07-0124-FEDER-000060".

References

[1] Alexa, M., Behr, J., Cohen-Or, D., Fleishman, S., Levin, D., Silva, C. T., Jan 2003. Computing and rendering point set surfaces. Visualization and Computer Graphics, IEEE Transactions on 9 (1), 3–15.

[2] Besl, P., McKay, N. D., Feb 1992. A method for registration of 3-d shapes. Pattern Analysis and Machine Intelligence, IEEE Transactions on 14 (2), 239–256.

[3] Diosi, A., 2005. Laser Range Finder and Advanced Sonar Based Simultaneous Localization and Mapping for Mobile Robots. Ph.D. thesis, Monash University.

[4] Filipe, S., Alexandre, L., 2014. A Comparative Evaluation of 3D Keypoint Detectors in a RGB-D Object Dataset. 9th International Conference on Computer Vision Theory and Applications.

[5] Fischler, M. a., Bolles, R. C., 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24 (6), 381–395.

[6] Fox, D., 2003. Adapting the Sample Size in Particle Filters Through KLD-Sampling. The International Journal of Robotics Research 22 (12), 985–1003.

[7] Frome, A., Huber, D., Kolluri, R., Bülow, T., Malik, J., 2004. Recognizing objects in range data using regional point descriptors 3023, 224–237.

[8] Gelfand, N., Rusinkiewicz, S., 2003. Geometrically stable sampling for the icp algorithm. In: Proc. International Conference on 3D Digital Imaging and Modeling. pp. 260–267.

[9] Grisetti, G., Stachniss, C., Burgard, W., Feb 2007. Improved techniques for grid mapping with rao-blackwellized particle filters. Robotics, IEEE Transactions on 23 (1), 34–46.

[10] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., Burgard, W., Feb. 2013. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous Robots 34 (3), 189–206.

[11] Jolliffe, 2002. Principal component analysis. Vol. 2.

[12] Kaess, M., Ranganathan, A., Dellaert, F., Dec 2008. isam: Incremental smoothing and mapping. Robotics, IEEE Transactions on 24 (6), 1365–1378.

[13] Kitt, B., Geiger, A., Lategahn, H., June 2010. Visual odometry based on stereo image sequences with ransac-based outlier rejection scheme. In: Intelligent Vehicles Symposium (IV), 2010 IEEE. pp. 486–492.

[14] Labbe, M., Michaud, F., Sept 2014. Online global loop closure detection for large-scale multi-session graph-based slam. In: Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on. pp. 2661–2666.

[15] Lingemann, K., Nüchter, A., Hertzberg, J., Surmann, H., 2005. High-speed laser localization for mobile robots. Robotics and Autonomous Systems 51 (4), 275–296.

[16] Lowe, D., 2004. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60 (2), 91–110.

[17] Magnusson, M., dec 2009. The three-dimensional normal-distributions transform — an efficient representation for registration, surface analysis, and loop detection. Ph.D. thesis, Örebro University, Örebro Studies in Technology 36.

[18] Mautz, R., 2012. Indoor Positioning Technologies (February 2012), 127.

[19] Newcombe, R. A., Fox, D., Seitz, S. M., June 2015. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time.

[20] Pomerleau, F., 2013. Methodology and Tools for ICP-like Algorithms. Ph.D. thesis, ETH Zürich.

[21] Pomerleau, F., Colas, F., Siegwart, R., Magnenat, S., 2013. Comparing icp variants on real-world data sets. Autonomous Robots 34 (3), 133–148.

[22] Pomerleau, F., Magnenat, S., Colas, F., Liu, M., Siegwart, R., Sept 2011. Tracking a depth camera: Parameter exploration for fast icp. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on. pp. 3824–3829.

[23] Rusinkiewicz, S., Levoy, M., 2001. Efficient variants of the icp algorithm. In: 3-D Digital Imaging and Modeling. IEEE, pp. 145–152.

[24] Rusu, R., Blodow, N., Beetz, M., May 2009. Fast point feature histograms (fpfh) for 3d registration. In: Robotics and Automation, 2009. ICRA '09. IEEE International Conference on. pp. 3212–3217.

[25] Rusu, R., Blodow, N., Marton, Z., Beetz, M., Sept 2008. Aligning point cloud views using persistent feature histograms. In: Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. pp. 3384–3391.

[26] Rusu, R. B., 2009. Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments. Ph.D. thesis, Technische Universität München.

[27] Rusu, R. B., Cousins, S., May 2011. 3d is here: Point cloud library (pcl). In: Robotics and Automation (ICRA), 2011 IEEE International Conference on. pp. 1–4.

[28] Segal, A., Haehnel, D., Thrun, S., June 2009. Generalized-icp. In: Proceedings of Robotics: Science and Systems. Seattle, USA.

[29] Sotoodeh, S., 2006. Outlier detection in laser scanner point clouds. In: International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences XXXVI-5. pp. 297–302.

[30] Steinbrücker, F., Kerl, C., Cremers, D., 2013. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In: Proceedings of the 2013 IEEE International Conference on Computer Vision. ICCV '13. IEEE Computer Society, Washington, DC, USA, pp. 3264–3271.

[31] Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D., Oct. 2012. A benchmark for the evaluation of RGB-D SLAM systems. IEEE, pp. 573–580.

[32] Stückler, J., Behnke, S., Sep. 2012. Integrating depth and color cues for dense multi-resolution scene mapping using rgb-d cameras. In: Proc. of the IEEE Int. Conf. on Multisensor Fusion and Integration for Intelligent Systems (MFI). pp. 162–167.

[33] Thrun, S., Mar. 2002. Probabilistic robotics. Commun. ACM 45 (3), 52–57.

[34] Tipaldi, G. D., Meyer-Delius, D., Burgard, W., Dec. 2013. Lifelong localization in changing environments. The International Journal of Robotics Research 32 (14), 1662–1678.

[35] Tombari, F., Salti, S., Di Stefano, L., 2010. Unique shape context for 3d data description. In: Proceedings of the ACM Workshop on 3D Object Retrieval. 3DOR '10. ACM, New York, NY, USA, pp. 57–62.

[36] Tombari, F., Salti, S., Di Stefano, L., Sept 2011. A combined texture-shape descriptor for enhanced 3d feature matching. In: Image Processing (ICIP), 2011 18th IEEE International Conference on. pp. 809–812.

[37] Vitter, J. S., Jul. 1984. Faster methods for random sampling. Commun. ACM 27 (7), 703–718.

[38] Wan, E., Van Der Merwe, R., 2000. The unscented kalman filter for nonlinear estimation. In: Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000. pp. 153–158.

[39] Wohlkinger, W., Vincze, M., Dec 2011. Ensemble of shape functions for 3d object classification. In: Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on. pp. 2987–2992.

[40] Zhang, Y., Meratnia, N., Havinga, P., Second 2010. Outlier detection techniques for wireless sensor networks: A survey. Communications Surveys Tutorials, IEEE 12 (2), 159–170.

[41] Zhong, Y., Sept 2009. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In: Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on. pp. 689–696.