Chapter 2

3 DoF/6 DoF Localization System for Low Computing Power Mobile Robot Platforms

Carlos M. Costa, Héber M. Sobreira, Armando J. Sousa and Germano Veiga

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/61258

Abstract

Mobile robot platforms have a wide range of hardware configurations in order to accomplish challenging tasks and require an efficient and accurate localization system to navigate in the environment. The objective of this work is the evaluation of the developed Dynamic Robot Localization (DRL) system in three computing platforms, with CPUs ranging from low to high end (Intel Atom, Core i5, and i7), in order to analyze the configurations that can be used to adjust the trade-offs between pose estimation accuracy and the associated computing resources required. The DRL is capable of performing pose tracking and global pose estimation in both 3 and 6 Degrees of Freedom (DoF) using point cloud data retrieved from LIDARs and RGB-D cameras and achieved translation errors of less than 30 mm and rotation errors of less than 5° when evaluated in three environments. The sensor data retrieved from three testing platforms was processed and the detailed profiling results were analyzed. Besides pose estimation, the self-localization system is also able to perform mapping of the environment with probabilistic integration or removal of geometry and can use surface reconstruction to minimize the impact of sensor noise. These abilities will allow the fast deployment of mobile robots in dynamic environments.

Keywords: Self-localization, point cloud registration, pose tracking, point cloud library, robot operating system

1. Introduction

Autonomous mobile robots capable of operating in dynamic environments have a multitude of applications and can be used to improve the overall speed and efficiency of a wide range of jobs. They can also cooperate with humans to accomplish complex tasks and are able to perform structured and repetitive movements with high precision.

INTECH
open science | open minds

© 2015 The Author(s). Licensee InTech. This chapter is distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Navigation within a dynamic environment requires a robust and accurate self-localization system in order to know where the robot is and which path it should follow to reach the location where it is expected to perform its tasks. The estimation of this global pose can use a varied number of techniques and technologies and may rely on proprioceptive knowledge that the robot has about itself (such as odometry), may incorporate exteroceptive information [1] retrieved from sensors (such as LIDARs, RGB-D cameras, sonars), or may even use infrastructures that are external to the robot (such as GPS). Nevertheless, for fast deployment of autonomous mobile robots on indoor environments, most localization systems rely on a combination of both proprioceptive and exteroceptive information to estimate the robot pose, given that external infrastructures are expensive for large operation areas and don't have either the coverage or the precision required for robot docking operations.

The most used self-localization systems can be categorized as probabilistic pose estimation methods, point cloud registration methods, or feature registration methods. Kalman filters such as the Extended Kalman Filter and the Unscented Kalman Filter [2] along with the particle filters [3] are the most used probabilistic methods but they rely on sensor models, and as such, it is hard to achieve very accurate pose estimations when the models are not well defined or when they change over time depending on the environment on which the robot is moving (which is the case of the odometry model). Point cloud registration methods that rely on the Iterative Closest Point [4, 5] or Normal Distributions Transform [6] can achieve very accurate pose estimations but require an initial alignment of the sensor data with the known map in order to successfully converge. Feature registration algorithms such as the ones presented in [7] and [8] don't require an initial alignment, but need distinctive geometry in the environment in order to successfully estimate the robot pose and are very computational intensive.

The DRL system combined feature registration methods with point cloud registration algorithms in order to provide high accuracy pose tracking along with global pose estimation when the robot starts its operation or becomes lost in the environment. This approach allowed the creation of a more efficient, accurate and robust localization system that uses the most appropriate types of algorithms depending on the knowledge that it has about the robot estimated pose in the environment. Each of these pose estimations have information about the registration of the sensor measurements with the reference point cloud, which includes the percentage of correctly registered sensor points along with their root mean square error and spatial angular distribution. The distribution of the initial poses are also given (when the localization system resets pose tracking, which occurs when the robot starts its operation without knowing where it is in the environment or when it becomes lost) in order to allow a navigation supervisor to detect if the estimated pose is ambiguous (when there are other known areas of the environment with similar geometry) and plot a path to disambiguate the estimated robot location. Besides pose estimation in 3 and 6 DoF, the DRL system can also incrementally build the map of the environment using probabilistic integration or removal of geometry along with surface reconstruction (to reduce the impact of sensor noise). This allows the robot to update an accurate representation of the environment while it performs its tasks, giving the possibility to explore unknown areas or plot paths through regions that where once unavailable.

The DRL system was evaluated using three testing platforms on three different environments and the sensor data was recorded into rosbags in order to allow the tests to be executed and profiled on three different CPUs (from low to high end). The test results showed that the DRL system was able to perform pose tracking with 5–30 mm of mean translation error and 0.4–5.0° of mean rotation error in both 3 and 6 DoF even when running on hardware with low computing capabilities (such as the Intel Atom N2800).

The next section describes the main processing modules of the DRL system. Section 4 details the testing platforms and configurations used, while section 5 provides an analysis of the test results. Section 6 finishes with the conclusions.

2. Brief presentation of the DRL system

The DRL system [9, 10] was implemented as a Robot Operating System (ROS) [11] package[1] and can perform 3 and 6 DoF pose estimations of mobile robot platforms. It was implemented as a set of modular C++ templated shared libraries in order to be reusable for other applications besides robot self-localization. It extensively uses the Point Cloud Library (PCL) [12] for point cloud preprocessing and registration and the OctoMap framework [13] for dynamic map update.

2.1. Processing pipeline configuration

Given the wide range of processing capabilities and sensor configurations that a mobile robot platform can have and the challenging environments in which they might operate, the DRL system can be fully configurable through yaml files in order to meet the specific needs of a given robot application while using the least amount of computational resources. To achieve these goals, it offers a flexible and dynamic pose estimation processing pipeline (brief overview shown in Figure 1) with a range of preprocessing algorithms along with three levels of point cloud registration. The first level is intended for the normal operation of the robot and can be configured for maximum efficiency and precision. The second level can be used for pose tracking recovery and can have algorithms and configurations able to recover from temporary tracking problems, such as partial sensor occlusions or unreliable odometry information. The third level of registration is able to estimate the initial pose of the robot when it starts operating or when it becomes lost in the environment. Besides pose estimation, the system can also incrementally build or update the environment map, allowing mobile robots to explore unknown areas and also leading to better path planning because the navigation system will have an updated view of its surroundings.

The DRL system was designed to operate with sensors capable of generating point clouds of the environment, and as such, it can directly use RGB-D and ToF cameras. For LIDARs, it offers a laser scan assembler that is able to merge measurements from several sensors using spherical linear interpolation (to reduce point cloud deformation). It also allows the usage of a circular

1 https://github.com/carlosmccosta/dynamic_robot_localization

buffer in order to register point clouds containing new measurements from the environment along with points that were previously registered (useful when using several sources of sensor data).

For fine tuning of the processing pipeline configuration, the self-localization system provides a detailed analysis of the estimated poses along with the computation time of each of its main processing stages. This allows to pinpoint the algorithms that require more processing resources, which can be valuable information when the system is running on platforms with very low processing capabilities. Moreover, these computation time logs also allow to identify which processing stages would benefit from a more suitable parameterization or a different algorithmic approach.

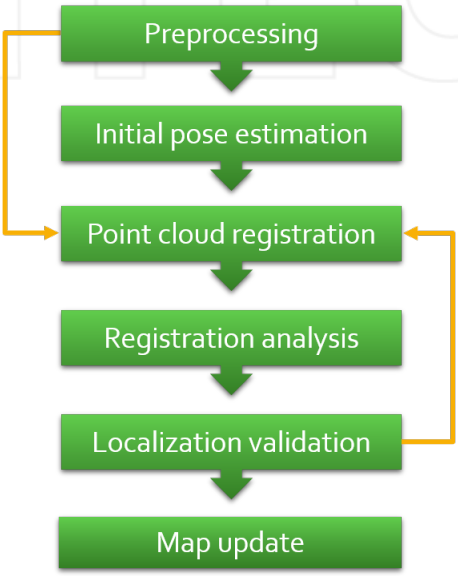

Figure 1. Localization system processing pipeline overview

2.2. Preprocessing

Preprocessing allows to adjust the level of detail of the reference or ambient point clouds and is also very effective in minimizing the impact of sensor noise (by using voxel grids, random sampling, outlier removal, surface reconstruction, sensor distance thresholds and color segmentation). Moreover, it can add information to the sensor data, such as line and surface normals and geometry descriptors, which are required to perform feature registration.

For fast deployment of mobile robots, the self-localization system supports two independent preprocessing pipelines: one for the reference point cloud (map of the environment) and another for the ambient and sensors point cloud. This allows the system to reuse maps from

different sources and with different levels of detail while also giving some control over the time that will be required to perform the pose estimations, given that the amount of time required to register two point clouds decreases considerably when the number of points in both the reference and ambient point clouds are reduced.

2.3. Initial pose estimation

When a mobile robot platform starts its operation without a known initial pose or when it becomes lost in the environment, it requires registration methods capable of estimating the global pose in the known map. One category of algorithms that achieves these goals is the feature matching techniques. Algorithms in this category start with a keypoint detection phase in order to find geometric significant points in the environment and then describe each keypoint by analyzing its surrounding geometry, such as points and normals distribution. These keypoint descriptors (usually histograms) are then matched using a kd-tree and the best correspondences are found using a Random Sample Consensus (RANSAC) approach. With these point correspondences, the correction matrix can be computed and used to estimate the robot pose in the known map.

The keypoint selection phase is used mainly to reduce the computational resources required to perform the initial pose estimation, given that computing descriptors for every point would need a significant amount of processing time and wouldn't improve the feature registration significantly. However, for the geometric feature matching to be successful, the keypoint detector must be able to find the same keypoints when similar geometry is given, even when the sensor data is affected by noise and the point clouds have different level of detail.

The localization system currently supports the Scale Invariant Feature Transform (SIFT) [14] and the 3D Intrinsic Shape Signatures (ISS3D) [15] keypoints detectors. For keypoint description it can use the Point Feature Histogram (PFH) [16], the Fast Point Feature Histogram (FPFH) [17], the Signature of Histograms of Orientations (SHOT) [18], the Shape Context 3D (SC3D) [19], the Unique Shape Context (USC) [20] and the Ensemble of Shape Functions (ESF) [21].

2.4. Point cloud registration

Point cloud registration algorithms such as the Iterative Closest Point (ICP) [22] (with its known variations such as ICP point-to-point, ICP point-to-point non-linear, ICP point-to-plane and generalized ICP [23]) and also the Normal Distributions Transform (NDT) [6] can achieve accurate registration of point clouds by iteratively minimizing the alignment error, but they require the point clouds to be partially aligned in order to converge to a valid solution. As such, by combining the initial pose estimation methods with the point cloud registration algorithms, the DRL system can reliably and accurately estimate the robot pose.

2.5. Registration analysis and validation

Each pose estimation is followed by a post processing stage in which several registration metrics are computed and analyzed in order to determine if the point cloud registration was successful and the estimated pose is good enough to be considered valid. The first metrics

computed are the inliers percentage and Root Mean Square Error (RMSE) (an inlier is a point in the sensor point cloud that has a point in the reference point cloud within a given distance radius). The second metric is the spatial angular distribution of both the inliers and outliers. This metric gives a measurement of confidence in the estimated pose and is based on the fact that when there are correctly registered points all around the robot, the confidence in the estimated pose is higher than when there is only correctly registered points in a small angular region of space. The last metrics are the translation and rotation corrections that were applied to the ambient point cloud in order to minimize the alignment error with the reference point cloud.

The configuration of these metrics thresholds allows to define what is considered a valid pose and may vary depending on the specific robot sensor configurations, the environment in which it will operate, the map resolution, and the required localization precision given the processing and capabilities of the robot.

These registration metrics also control when the self-localization system switches between the three point cloud registration modes. By default, the pose estimations are performed in the normal tracking mode. If the registration fails, the pose tracking recovery methods are activated. If the pose estimation continues to fail with new sensor data for a given period of time and a minimum number of pose estimations has been rejected, then the initial pose algorithms using feature registration are used to estimate the robot global pose and reset the tracking state.

2.6. Incremental map update

After successfully registering the 2D or 3D sensor data with the reference point cloud, the self-localization system can update the 2D or 3D map of the environment by integrating the fully registered point cloud or only the inliers or outliers. This allows to perform full integration of the sensor data when the environment is expected to change drastically or only inlier or outlier integration when small sections of the map need to be updated. By integrating only the outliers, the computation requirements are reduced considerably and also allow to avoid the degradation of the map by not integrating sensor data close to known areas (that could have been generated from CAD models or other accurate mapping systems). For 3D maps, surface reconstruction can be used to reduce the impact of sensor noise and to increase the quality and accuracy of the environment representation.

The self-localization system can also be paired with the OctoMap library in order to perform probabilistic integration of the sensor data and removal of missing ambient geometry.

3. Testing configurations

The DRL system was tested in three environments with three testing platforms in order to assess its accuracy and robustness when both the environment and its sensor configurations change.

The next sections provide a brief overview of the testing platforms used and the environments in which they were deployed.

3.1. Testing platforms

The self-localization system was tested with sensor data retrieved from two different mobile robot platforms (3 DoF tests) and also from a standalone Kinect sensor (6 DoF tests).

In order to allow the repetition and comparison of the tests in different computing platforms, the sensor data was recorded into rosbags and is available in the following repository[2].

The next sections provide a brief description of each of the testing platforms while Table 1 gives an overview of the sensors specifications.

3.1.1. Jarvis robot

The Jarvis robot (shown in Figure 2) is one of our autonomous ground vehicles and was equipped with a SICK NAV350 laser for self-localization (mounted about 2 m from the floor) and a SICK S3000 laser for collision avoidance (mounted about 0.1 m from the floor). It uses a tricycle locomotion system and had a ground truth provided by the SICK NAV350 system (relied on 6 lasers reflectors with 0.09 m of diameter).

Figure 2. Jarvis testing platform

2 https://github.com/carlosmccosta/dynamic_robot_localization_tests

3.1.2. Pioneer 3-DX robot

The Pioneer 3-DX robot (shown in Figure 3 and presented in [24]) is a small autonomous vehicle equipped with a SICK LMS200 laser (mounted about 0.48 m from the floor) and a Kinect (mounted about 0.78 m from the floor). It uses a differential locomotion system and the ground truth was computed using 8 Raptor-E cameras.

Figure 3. Pioneer 3-DX testing platform [24]

3.1.3. Kinect

The Kinect sensor (shown in Figure 4) is a structured light sensor capable of generating 3D colored point clouds of the environment at 30 Hz. It has a range of about 4 meters and has a vertical field of view of 43° and a horizontal field of view of 57°. It was moved within the testing area by a human operator and the ground truth was given by a Vicon motion tracking system.

Figure 4. Kinect sensor

Sensor	Number of measure-ments	Range (meters)	Field of view (degrees)	Scanning frequency (Hz)	Angular resolution (degrees)	Statistical error (millimeters)
SICK NAV350	1440	[0.5..250]	360H	8	0.25	15
SICK S3000	760	[0.1..49]	190H	8	0.25	150
SICK LMS200	180	[0.1..80]	180H	10	1.00	35
Kinect	307200	[0.8..4.0]	57H | 43V	30	0.09	10

Table 1. Sensors hardware specifications

3.2. Computing platforms

The accuracy and computational requirements of self-localization systems can change significantly depending on the environment, sensors used, and movement path of the robot. As such, in order to have representative results, the DRL system was tested in three computing platforms with very different processing capabilities and all running Ubuntu 12.04 along with ROS Hydro and PCL 1.7.

The next sections present a brief overview of each of these computing platforms while Table 2 provides a detailed description of the CPUs used.

3.2.1. High-performance laptop

The Clevo P370EM laptop is a 2012 high-performance laptop equipped with a quad-core Intel Core i7-3630QM CPU, 16 GB of DDR3 at 1600 MHz, and an NVidia GeForce GTX680M GPU.

3.2.2. Low-performance laptop

The Samsung 530U3C is a 2012 low-performance laptop equipped with a dual-core Intel Core i5-3317U CPU, 6 GB of DDR3 at 16000 MHz, and an Intel HD Graphics 4000 GPU.

3.2.3. Low-performance embedded PC

The 2012 low-performance embedded PC was installed in a Robotnik Guardian mobile robot platform and was equipped with a dual-core Intel Atom N2800 CPU, 2 GB of DDR3 at 1066 MHz, and an Intel Graphics Media Accelerator 3650 GPU.

3.2.4. Computing platforms comparison

When building mobile robots, autonomy can be a serious concern, and, as such, a CPU with very low power consumption and a passive or fanless cooling system (such as the Intel Atom N2800 with 6.5 W of Thermal Design Power (TDP)) might be enough to accomplish the desired tasks. When higher workloads are expected, a low-power mobile CPU (such as the Intel Core i5-3317U with a TDP of 17 W) might be a better choice. If power consumption (and cost) isn't

an issue, then a high-performance CPU will be more than enough to run the DRL system in both 3 and 6 DoF along with the perception, planning, and decision modules.

Analyzing the benchmark results present in Table 2[3], it can be seen that the low-performance embedded PC had a CPU (Intel Atom N2800) about 10 times less capable than the high-performance laptop (Intel i7-3630QM), while the low-performance laptop (Intel i5-3317U) had only 2.5 times less computing capabilities. It should also be noted that the Intel Atom N2800 didn't have a level 3 cache, which could significantly improve the performance of the CPU when using applications that make intensive use of memory, which is the case of the DRL system. Moreover, the memory used in the embedded PC had about 50% less frequency (1066 MHz vs. 1600 MHz) and the Direct Memory Interface (DMI) had half the rate of the other two CPUs (2.5 GT/s vs. 5.0 GT/s), which causes cache misses to have high impact on overall CPU performance. Lastly, the Intel Atom had an operating frequency that was 1.29 times lower than the Intel i7-3630QM (and didn't have Intel Turbo Boost functionality).

	Intel Atom N2800	Intel i5 3317U	Intel i7-3630QM	i7 / i5	i7 / Atom	i5 / Atom
Number of cores	2	2	4	2.00	2.00	1.00
Minimum operating frequency (GHz)	1.867	1.7	2.4	1.41	1.29	0.91
Maximum operating frequency (GHz)	1.867	2.6	3.4	1.31	1.82	1.39
Direct Memory Interface (GT/s)	2.5	5.0	5.0	1.00	2.00	2.00
Level 1 instructions cache (KB)	2*32	2*32	4*32	2.00	2.00	1.00
Level 1 data cache (KB)	2*24	2*32	4*32	2.00	2.67	1.33
Level 2 cache (KB)	2*512	2*256	4*256	2.00	2.00	1.00
Level 3 cache (KB)	0	3072	6144	2.00	-	-
Number of transistors (billions)	0.176	0.634	1.4	2.21	7.95	3.60
Hyper-Threading	Yes	Yes	Yes	-	-	-
Manufacturing process (nm)	32	22	22	1.00	0.69	0.69
Thermal Design Power (watt)	6.5	17	45	2.65	6.92	2.62
Launch date	Q4-2011	Q2-2012	Q3-2012	-	-	-
Microarchitecture	Saltwell	Ivy Bridge	Ivy Bridge	-	-	-
Recommended Customer Price (dollars)	47	225	378	1.68	8.04	4.79
3D Mark 06 CPU score	965	2984	6392	2.14	6.62	3.09
Passmark score	636	3097	7766	2.51	12.21	4.87
GeekBench 32-bit score	1033	4039	10196	2.52	9.87	3.91
CINEbench 32-bit R10 Multi CPU score	1829	7337	18091	2.47	9.89	4.01

Table 2. Computing platforms CPU comparison

3 Retrieved from http://cpuboss.com/ and http://www.notebookcheck.net/

3.3. Testing environments

The self-localization system was tested in three different environments using the Jarvis robot in a RoboCup field, the Pioneer 3-DX robot in an industrial hall, and a Kinect sensor in a flying arena.

The next sections describe each of these testing environments.

3.3.1. Jarvis robot in a RoboCup field

The RoboCup field (shown in Figures 5 and 6) is on the right side of a large room with 20.5 m of length and 7.7 m of depth. It has two doors, several small windows, and three large glass openings into the hallway.

Several tests were performed in this environment, with the robot moving with speeds ranging from 0.05 m/s to 0.5 m/s. These tests were done with the robot following either a circular or a complex path containing linear and rotational movements with different speeds.

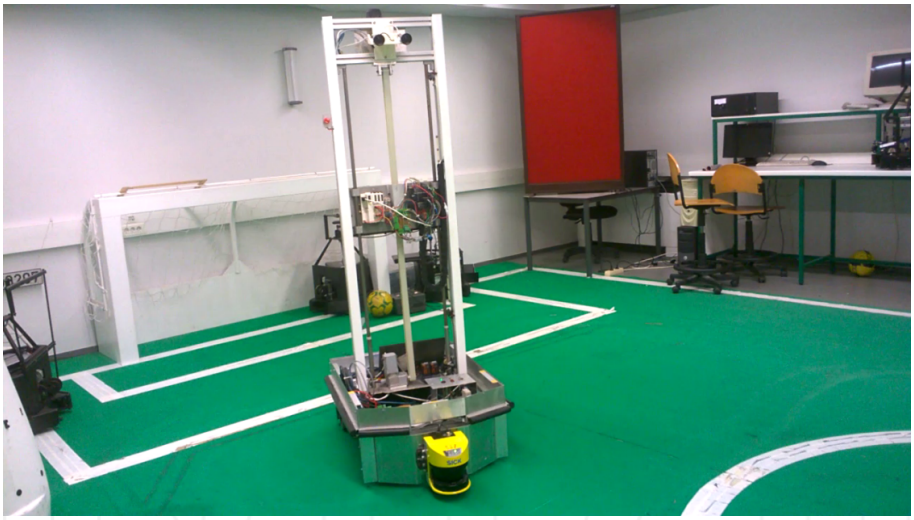

Figure 5. Jarvis testing environment (east side)

3.3.2. Pioneer 3-DX robot in an industrial hall

The industrial hall (shown in Figures 7 and 8 and presented in [24]) is a large room with long and high walls and several occluding objects in the middle.

The first test was done with the robot moving at 0.23 m/s, following a 360° path and without objects in the middle of the environment. The remaining tests were performed with the robot following different paths, with speeds ranging from 0.16 m/s to 0.26 m/s and with occluding objects in the middle.

Figure 6. Jarvis testing environment (west side)

Figure 7. Industrial hall overview [24]

Figure 8. Industrial hall with objects in the middle [24]

3.3.3. Kinect sensor in a flying arena

The flying arena (shown in Figure 9 and presented in [25]) is a large room where several objects with varying dimensions and shapes were added in order to test self-localization systems.

Three different tests were performed in this environment. In the first test, the Kinect was moved in a free-fly motion while the remaining two tests had movements with mainly translations or rotations.

4. Test results

This section presents the results achieved with the DRL system in each of the testing environments (shown from Table 3 to Table 8). They were performed in three different computing platforms with varying processing capabilities and with known initial poses. Each test has information about the conditions in which it was performed (duration, mean velocity of the robot and its path, and also the number of sensor measurements used), and the analysis of the self-localization system error (translation or rotation mean and standard deviation error and

Figure 9. Kinect sensor flying arena [25]

also mean and standard deviation of the computation time used to perform the pose updates) along with the profiling of the computational resources required by the DRL system (using the collectl monitoring tool for the CPU and memory usage and the perf profiler for the remaining metrics).

4.1. 3 DoF test results

The 3 DoF tests relied on LIDAR data and used the 2D ICP (point-to-point) algorithm for pose tracking and 3D ICP (point-to-point) algorithm for tracking recovery (with a larger kd-tree search radius for finding point correspondences, higher limit for the number of iterations allowed in the point cloud registration and a higher convergence time limit).

The 3 DoF maps were generated with the self-localization system in SLAM mode (shown in Figures 10 and 15) and were manually corrected in order to improve the map accuracy and to keep only the most relevant geometry of the environment.

4.1.1. Jarvis robot test results

The 3 DoF tests performed with the Jarvis robot in the RoboCup field (presented in Tables 3 and 4 and from Figures 11 to 14) show that the DRL system was able to track the robot pose (on a map with 10 mm cell resolution) with high accuracy (4 to 12 mm of translation error and 0.4° to 0.7° of rotation error) on the three computing platforms (when using 500 points from sensor measurements). The localization system error (translation and rotation) was consistent across all testing platforms (deviation in translation error bellow 1 mm and in rotation error below 0.11° between tests, which shows the repeatability of the DRL system) and the mean computation time required to update the pose with new sensor data increased on the CPUs with lower processing capabilities. As expected, the computation time ratios between the different CPUs were similar to the benchmark performance ratios presented in Table 2. In this dataset, the Intel Atom had a mean computation time that was 7.69 times higher than the Intel i7, and the Intel i5 required only 1.35 more processing time than the Intel i7. Moreover, the standard deviation of the computation time also increased on the Intel Atom (6.43 times higher than the Intel i7) and in the Intel i5 (1.31 higher than the Intel i7). The CPU usage (percentage in tables are from 0–100% times the number of virtual cores, which in the case of a quad-core with Hyper-Threading would yield a range of [0-800]) followed the same trend as the computation time, while the memory required by the self-localization system remained consistent across all testing platforms (it is related to the map resolution and number of points in the reference or sensor point clouds).

Analyzing the data collected with the perf profiler, it can be seen that the Intel i7 had a mean operation frequency of 3.0 GHz, while the Intel i5 remained at 2.3 GHz (77% of the Intel i7 frequency) and the Intel Atom achieved only 1.82 GHz (62% of the Intel i7 frequency). Moreover, due to the lack of level 3 cache and smaller level 1 cache, the Intel Atom had smaller cache references rate (2.76 times less than the Intel i7) and significantly more cache misses (3.74 times more than the Core i7), which caused an increase of 37% in bus cycles in order to access the main memory. Besides cache misses, the Intel Atom also had much more branch misses (3.46 times more than the Intel i7), which led to less branch instructions per second (6.76 times less than the Intel i7). These memory access bottlenecks and branch mispredictions caused the mean number of instructions per cycle to drop to 0.32 (the Intel Atom has instruction pipelining and Hyper-Threading, which in ideal conditions allows it to process two instructions per clock cycle), which is an indicator that the CPU was spending a considerable amount of time waiting for data from the main memory (which can be several orders of magnitude slower than cache memory) or was wasting processing resources in branches that were not taken (resulting in the execution of instructions that won't be useful and can cause the flushing of the processing pipeline, resulting in even more waste of CPUs resources).

Given that most mobile robot platforms can be used to perform complex tasks in the environment besides localization and navigation, two more configurations were tried in each of the four tests (done with the Intel Atom) in order to assess the trade-offs between localization precision and computation time. This was achieved because the self-localization system can be tuned to specific requirements of accuracy (given the available sensors and computational resources). Looking at Tables 3 and 4, it can be seen that reducing the number of points used

in the point cloud registration (retrieved from sensor data) to half (250) led to a reduction in computation time of 37% at the cost of increasing the localization translation error by 48%. Reducing the number of points to half again (125), resulted in a reduction of computation time of 62% and a translation error increase of 217% (in relation to the original test with 500 points). Despite the large increase in translation error when reducing the number of registration points, some robots may prefer a localization system with low computational requirements and a localization error close to a centimeter instead of having millimeter accuracy with moderate CPU usage.

Analyzing the remaining test results, it can be seen that the Intel i5 didn't suffer from the Intel Atom memory bottlenecks and had much better branch prediction (Intel Atom has a different micro-architecture than the Intel Core processors), having only a small increase in cache misses (14%) and a decrease in branch misses (4%) in relation to the Intel i7. Moreover, the number of instructions per cycle, along with bus cycles rate, was very similar to the Intel i7 (less than 2% difference), while the cache references rate and branch instructions rate decreased by 25% in relation to the Intel i7 CPU.

Comparing the overall results, the Intel Atom was capable of tracking the robot pose with high precision with moderate CPU usage. Nevertheless, for robots designed to perform complex tasks besides navigation, the Intel i5 is probably a better choice, given that the localization system was only using 10% of one of its CPU cores (instead of 50% of one of the Intel Atom cores).

Path	Test duration (seconds)	Velocity (meters / second)	CPU	Number of sensor points	Translation error (millimeters)		Rotation error (degrees)		Computation time for pose update (milliseconds)		Test nº
					Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	
Circular	290	0.05	Intel Atom N2800	125	12.16	8.24	0.71	0.19	18.81	6.26	1.01
				250	5.69	3.40	0.60	0.10	31.20	11.11	1.02
				500	3.83	2.33	0.56	0.09	49.25	17.09	1.03
			Intel i5 3317U	500	4.15	2.38	0.56	0.09	9.05	3.24	1.04
			Intel i7-3630QM	500	3.97	2.38	0.56	0.09	6.57	2.41	1.05
	25.5	0.5	Intel Atom N2800	125	14.31	9.04	0.61	0.46	23.71	6.75	1.06
				250	12.40	8.53	0.57	0.43	43.55	12.83	1.07
				500	12.20	8.72	0.54	0.41	79.92	17.01	1.08
			Intel i5 3317U	500	11.89	8.66	0.54	0.41	14.80	4.61	1.09

Path	Test duration (seconds)	Velocity (meters / second)	CPU	Number of sensor points	Translation error (millimeters)		Rotation error (degrees)		Computation time for pose update (milliseconds)		Test nº
					Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	
			Intel i7-3630QM	500	12.06	8.53	0.55	0.41	10.95	3.51	1.10
Complex	498	0.05	Intel Atom N2800	125	12.60	7.46	0.53	0.16	20.08	7.09	1.11
				250	7.82	4.07	0.49	0.10	34.64	13.10	1.12
				500	5.07	2.71	0.47	0.08	57.78	18.50	1.13
			Intel i5 3317U	500	5.13	2.75	0.45	0.08	9.60	3.39	1.14
			Intel i7-3630QM	500	4.86	2.67	0.46	0.07	7.20	2.66	1.15
	392	0.5-0.3-0.5-0.1	Intel Atom N2800	125	12.86	7.42	0.49	0.14	19.89	6.74	1.16
				250	8.52	4.84	0.45	0.12	35.73	13.48	1.17
				500	6.79	4.04	0.42	0.10	60.79	20.17	1.18
			Intel i5 3317U	500	6.74	3.92	0.42	0.10	10.35	3.88	1.19
			Intel i7-3630QM	500	6.74	3.92	0.42	0.10	7.66	2.95	1.20

Table 3. Self-localization test results performed with the Jarvis robot in the RoboCup field

Path	CPU usage (% 0..100 * nº cores)		Memory usage (MB)		CPU cycles (GHz)	Instructions per cycle	Cache references (M/sec)	Cache misses (% of all cache references)	Branch instructions (M/sec)	Branch misses (% of all branches)	Bus cycles (M/sec)	Test nº
	Mean	Standard deviation	Mean	Standard deviation								
Circular	33.35	14.23	65.46	5.24	1.83	0.22	15.18	27.66	80.43	7.24	130.24	1.01
	41.23	17.57	65.48	5.23	1.83	0.27	13.72	25.64	96.93	6.48	130.86	1.02
	52.10	23.61	65.86	5.34	1.83	0.31	12.32	23.76	109.23	5.95	131.23	1.03
	10.64	8.69	67.53	4.60	2.34	1.31	25.02	7.01	554.37	1.66	94.10	1.04
	7.57	7.59	87.22	4.86	3.02	1.27	36.75	5.47	768.84	1.69	96.66	1.05
	36.61	28.01	59.45	9.68	1.83	0.28	14.61	22.59	99.64	5.86	130.66	1.06

Path	CPU usage (% 0..100 * nº cores)		Memory usage (MB)		CPU cycles (GHz)	Instructions per cycle	Cache references (M/sec)	Cache misses (% of all cache references)	Branch instructions (M/sec)	Branch misses (% of all branches)	Bus cycles (M/sec)	Test nº
	Mean	Standard deviation	Mean	Standard deviation								
	48.55	29.45	59.73	9.91	1.83	0.31	13.14	21.71	107.96	5.71	131.03	1.07
	63.82	37.88	59.82	9.94	1.84	0.35	11.52	20.25	122.21	5.41	131.01	1.08
	12.76	15.50	59.65	4.01	2.20	1.42	23.46	8.25	598.90	1.55	93.53	1.09
	8.86	13.40	78.82	3.22	2.84	1.46	25.72	8.44	788.61	1.47	94.57	1.10
Complex	32.71	13.86	70.32	7.59	1.82	0.22	15.22	28.13	77.47	7.43	130.55	1.11
	41.18	18.86	70.41	7.56	1.83	0.27	13.66	25.84	93.93	6.59	130.98	1.12
	55.16	25.20	70.47	7.58	1.83	0.32	12.26	22.75	108.13	5.96	131.30	1.13
	10.32	8.31	72.72	8.14	2.30	1.37	25.39	6.51	596.54	1.62	95.74	1.14
	8.28	7.40	91.27	7.59	3.04	1.30	37.44	5.12	741.56	1.82	96.07	1.15
	31.69	14.18	67.72	6.23	1.82	0.21	15.44	28.26	75.27	7.58	130.55	1.16
	39.78	19.91	67.97	6.37	1.83	0.26	14.01	25.55	91.91	6.66	130.93	1.17
	53.06	27.91	67.84	6.49	1.83	0.31	12.47	22.85	107.72	5.98	131.19	1.18
	10.38	8.76	67.85	5.93	2.33	1.32	26.37	6.34	589.79	1.63	95.57	1.19
	8.09	7.82	88.94	6.03	3.04	1.28	37.05	6.05	725.08	1.78	95.70	1.20

Table 4. Profiling of self-localization tests performed with the Jarvis robot in the RoboCup field

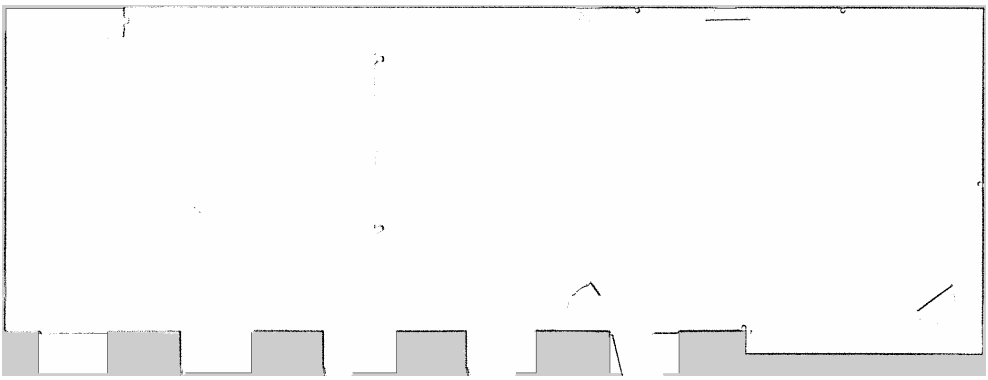

Figure 10. Map of the RoboCup testing environment using the DRL system in SLAM mode

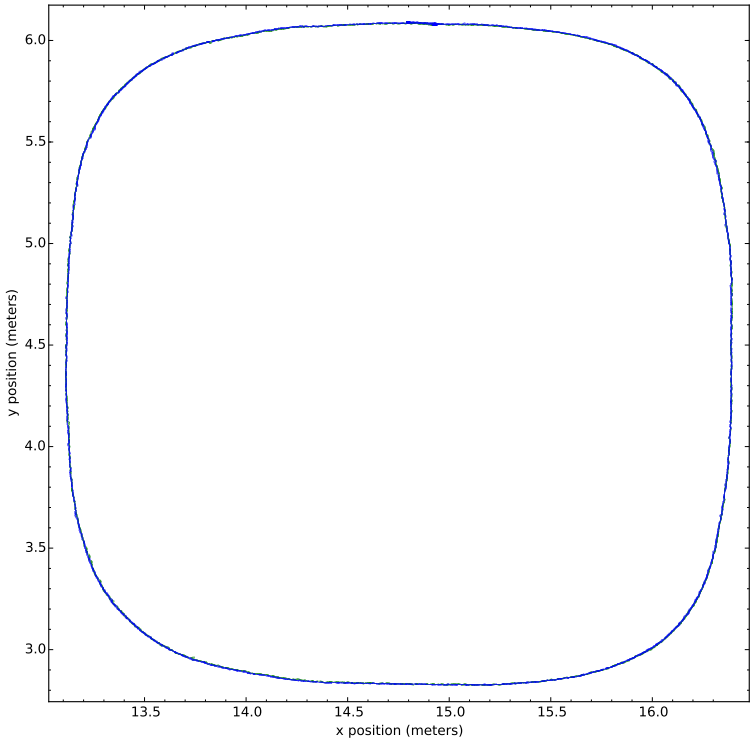

Figure 11. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 1.03

4.1.2. Pioneer 3-DX robot results

The 3 DoF tests performed with the Pioneer robot in the industrial hall aimed to assess the robustness of the localization system on more challenging environments. In this dataset, the LIDAR sensor (SICK LMS200) had half the field of view of the sensor used in the previous dataset (SICK NAV350), and the angular resolution was 4 times lower, which resulted on 180 points for each laser scan (the previous dataset had 1,440 points per laser scan spread across 360°). Moreover, the resolution of the map (shown in Figure 15) was 2.5 times lower (25 mm cell resolution) and the laser scan measurements were limited to 10 m on the tests that had no objects in the middle of the environment (test 2.01, 2.02, and 2.03) and 6 m on the remaining tests (from test 2.04 to 2.12 in which there were objects in the middle of the environment that occluded the walls).

Analyzing Table 5 and 6 and Figures 16 to 19, it can be concluded that decreasing the field of view and reducing the map resolution resulted in a 5-time increase of the translation error and a 9-time increase in rotation error (it should be noted that the translation error remained below the map resolution, which is compelling evidence that the algorithms used can perform point

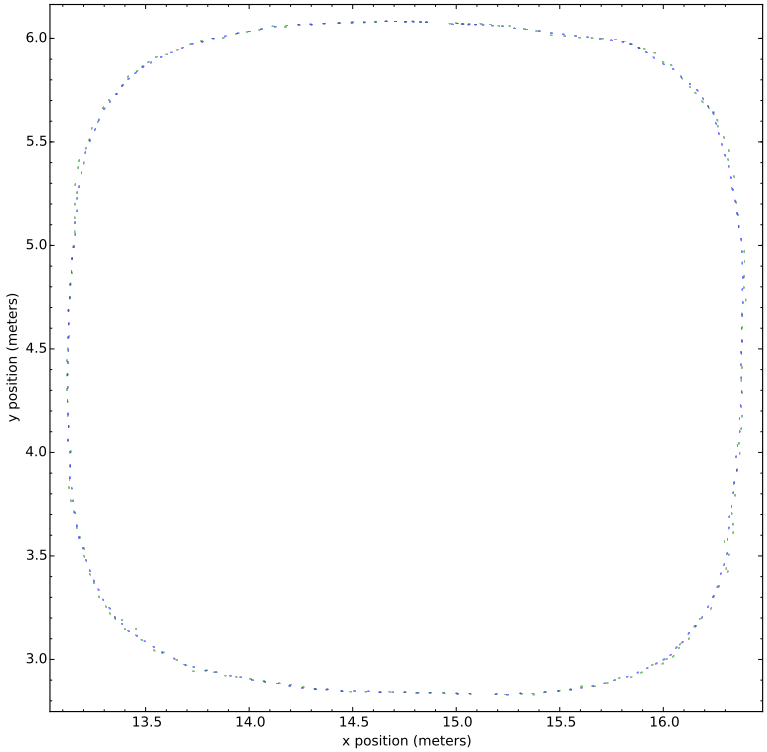

Figure 12. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 1.08

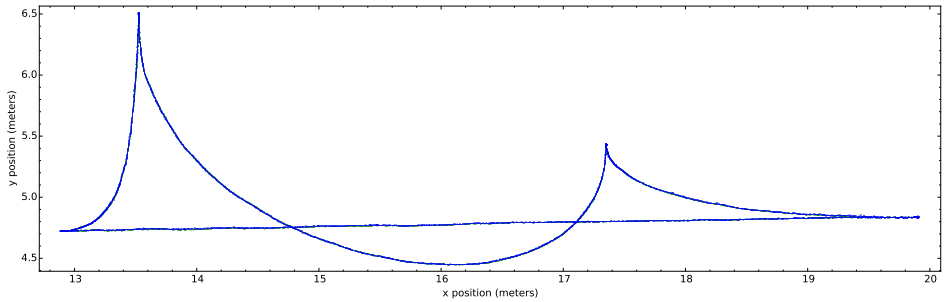

Figure 13. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 1.13

cloud registration with high accuracy). The computation time was reduced about 40% on the Intel Core i7 and i5 (in relation to the previous dataset), and was mostly related to the fact that the number of points in both the reference and sensor point clouds was reduced by 33%. However, the computation time on the Intel Atom remained similar on both 3 DoF datasets

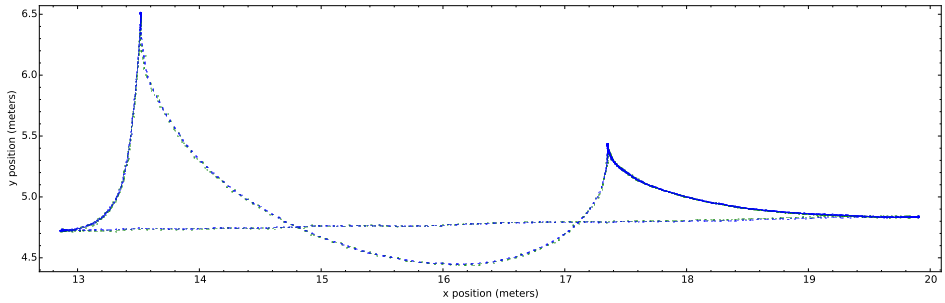

Figure 14. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 1.18

because the percentage of branch misses increased and the mean number of instructions per cycle was reduced (from 0.32 to 0.24).

Comparing the results across the three computational platforms, it can be seen that both translation and rotation error distributions remained consistent across the three computing platforms and had similar results (about 2% change). The Intel i5 had very similar profiling results in relation to the Intel i7, while the Intel Atom required much more time to compute the results (due to its higher percentage of cache and branch misses). As such, similar to the previous dataset, the Intel i5 would be more suitable to run the DRL system since it doesn't suffer from the memory access bottlenecks and was able run the localization system with low CPU usage while requiring much less power than the Intel i7 (17 W vs. 45 W TDP).

Path	Test duration (seconds)	Velocity (meters / second)	CPU	Number of sensor points	Translation error (millimeters)		Rotation error (degrees)		Computation time for pose update (milliseconds)		Test nº
					Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	
360	72	0.23	Intel Atom N2800	180	20.32	11.09	5.71	0.66	59.23	22.76	2.01
			Intel i5 3317U		19.86	10.20	5.71	0.66	6.67	2.31	2.02
			Intel i7-3630QM		19.86	10.19	5.71	0.66	4.69	1.50	2.03
Slam 1	155	0.26	Intel Atom N2800		22.49	12.46	5.44	0.68	51.98	25.26	2.04
			Intel i5 3317U		22.34	12.19	5.43	0.68	5.74	2.53	2.05
			Intel i7-3630QM		22.35	12.20	5.43	0.68	3.92	1.66	2.06
Slam 2	115	0.19	Intel Atom N2800		19.67	13.09	5.52	0.82	54.36	24.60	2.07
			Intel i5 3317U		19.50	12.94	5.53	0.82	6.08	2.55	2.08
			Intel i7-3630QM		19.48	12.91	5.53	0.82	4.16	1.66	2.09

Path	Test duration (seconds)	Velocity (meters / second)	CPU	Number of sensor points	Translation error (millimeters)		Rotation error (degrees)		Computation time for pose update (milliseconds)		Test nº
					Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	
Slam 3	112	0.16	Intel Atom N2800		17.93	9.14	5.46	0.72	49.41	24.59	2.10
	112	0.16			17.93	9.14	5.46	0.72	49.41	24.59	2.10
			Intel i5 3317U		17.68	8.93	5.45	0.73	5.54	2.48	2.11
			Intel i7-3630QM		17.68	8.93	5.46	0.73	3.87	1.67	2.12

Table 5. Self-localization test results performed with the Pioneer robot in the industrial hall

Path	CPU usage (% 0..100 * nº cores)		Memory usage (MB)		CPU cycles (GHz)	Instructions per cycle	Cache references (M/sec)	Cache misses (% of all cache references)	Branch instructions (M/sec)	Branch misses (% of all branches)	Bus cycles (M/sec)	Test nº
	Mean	Standard deviation	Mean	Standard deviation								
360	62.55	28.56	61.94	4.07	1.82	0.24	16.12	19.90	86.44	7.54	130.48	2.01
	14.06	10.99	64.08	4.12	2.24	1.03	33.19	5.96	464.40	1.82	95.02	2.02
	11.79	10.37	85.02	4.08	2.93	0.96	51.91	1.83	586.10	2.03	94.95	2.03
Slam 1	61.84	23.87	67.99	6.85	1.82	0.23	16.19	21.24	82.17	7.77	130.48	2.04
	13.98	9.82	71.11	7.68	2.25	0.93	38.20	4.75	416.97	2.02	94.66	2.05
	11.00	8.69	94.15	7.68	2.99	0.95	59.02	1.71	567.99	2.10	95.57	2.06
Slam 2	61.35	25.76	65.02	5.53	1.83	0.23	16.31	20.40	84.66	7.62	130.23	2.07
	14.13	10.18	67.86	5.98	2.28	0.64	35.70	4.73	437.17	1.89	94.28	2.08
	11.36	9.62	89.02	6.13	3.00	0.92	57.14	1.72	567.34	2.00	94.03	2.09
Slam 3	60.24	25.44	65.16	5.32	1.82	0.23	16.28	21.17	83.64	7.65	130.58	2.10
	13.20	10.53	67.81	5.96	2.14	0.96	35.90	5.50	433.83	1.92	95.01	2.11
	11.00	10.35	88.74	5.84	2.94	0.94	57.45	1.94	579.68	2.04	94.27	2.12

Table 6. Profiling of self-localization tests performed with the Pioneer robot in the industrial hall

4.2. 6 DoF Kinect sensor results

The 6 DoF tests performed with a standalone Kinect sensor in the flying arena (shown in Tables 7 and 8 and from Figure 22 to Figure 27) were meant to evaluate the accuracy and robustness of the self-localization system when moving the sensor in 6 DoF and following challenging paths in cluttered environments.

The 3D map (shown in Figures 20 and 21) was built using only the sensor data from the free-fly test (3.03) with the DRL system in SLAM mode and used surface reconstruction to automatically reduce the impact of sensor noise and improve its accuracy. Given the different paths

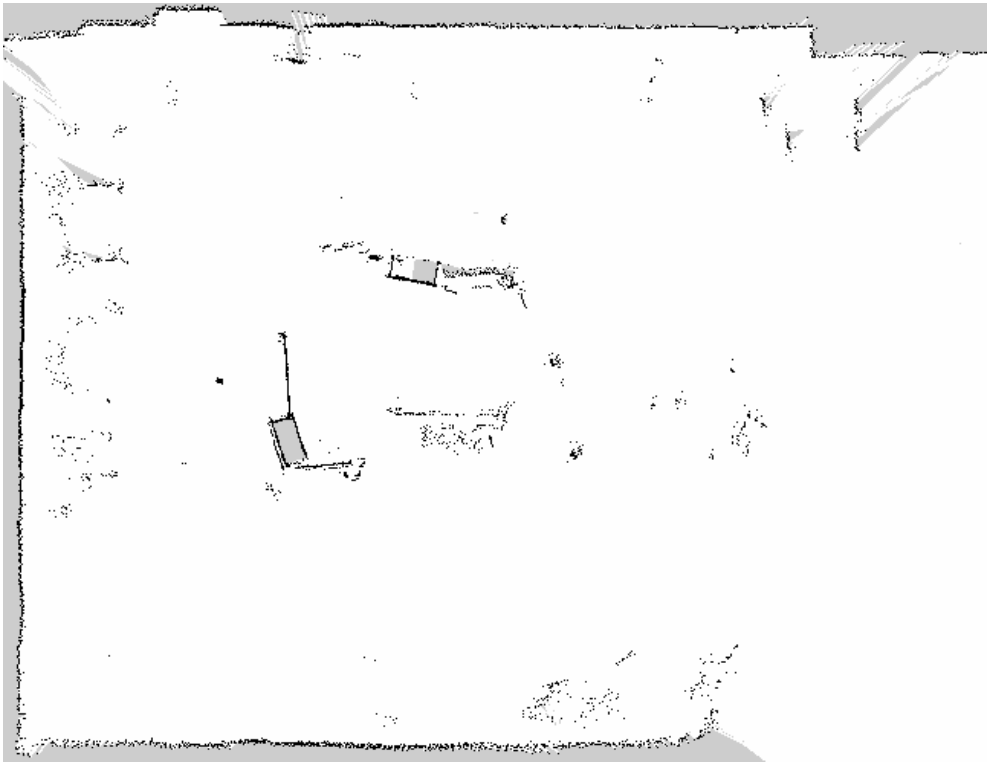

Figure 15. Map of the industrial hall testing environment using the DRL system in SLAM mode

that the Kinect had in each test, there were times in which the sensor field of view was outside of the known map (mainly in the test with rotations). The map was not extended to contain all the areas in the three tests in order to evaluate the robustness of the system against sensor occlusion or malfunctions (which are problems similar to the sensor seeing unknown areas). To allow real-time processing of the Kinect sensor data and keep the CPU usage under acceptable thresholds, the map was downsampled with a voxel grid of 20-mm cell size in both the Intel i7 and Intel i5 tests and a voxel grid of 50-mm cell size on the Intel Atom tests (to allow higher pose update rate).

Unlike the previous 3 DoF tests, estimating a pose in 6 DoF requires a reference point cloud with much more points (in order to represent the environment with the required level of detail), which leads to slower neighbor point searches (because the kd-tree used to perform the neighbor point searches will have more levels and it will take longer to retrieve the desired points). Given that point cloud registration algorithms make heavy use of point searches, the cumulative effect of increasing the search time led to the significant increase of the computation time and CPU usage seen in Tables 7 and 8 (when compared with the 3 DoF tests). Moreover, the more intensive use of memory caused the percentage of cache misses to increase signifi-

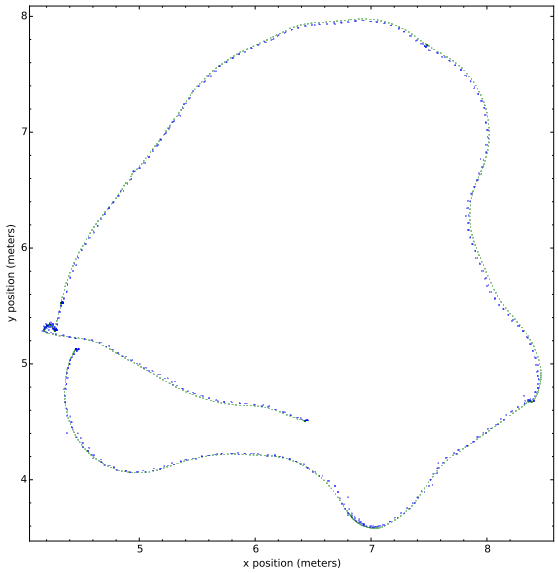

Figure 16. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 2.01

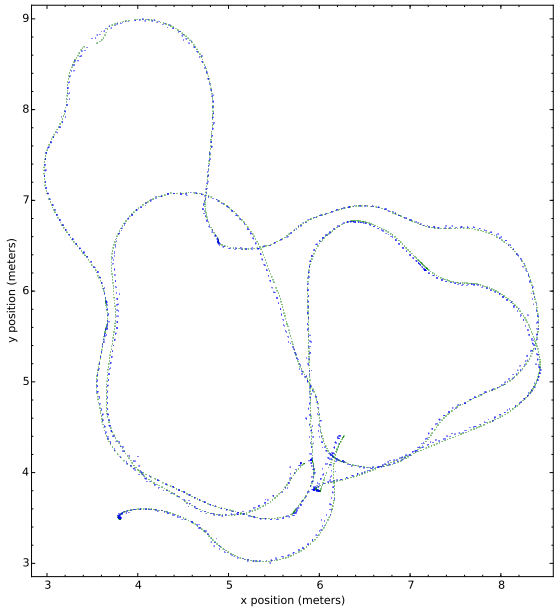

Figure 17. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 2.04

http://dx.doi.org/10.5772/61258

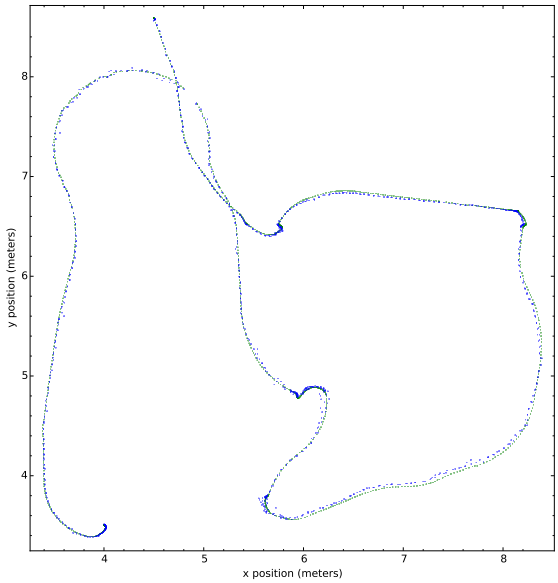

Figure 18. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 2.07

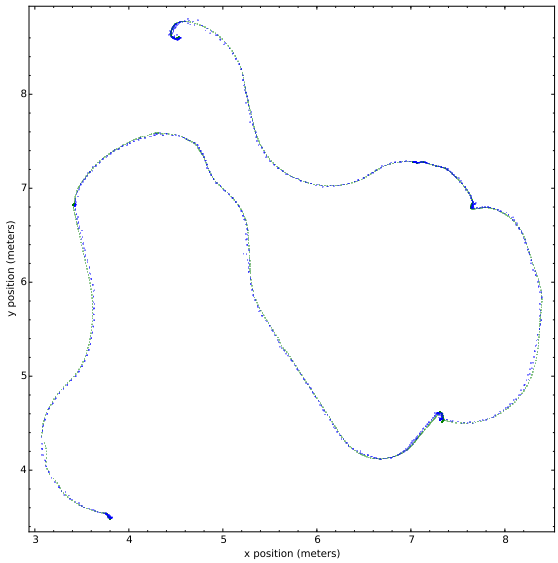

Figure 19. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 2.10

cantly even on the Intel i5 and i7 (from 5–7% to 10–20%). This computational time increase was clearly seen in the test with mainly rotations in which the pose recovery systems were being activated when the sensor had its field of view outside the map. These recovery systems relied on the ICP point-to-plane (with a larger point search radius and higher convergence time limit) in order to be able to reset the tracking state (the normal tracking systems were using ICP point-to-point algorithm). Moreover, the maximum number of points used in this test was increased to make the pose recovery more robust, which also contributed to the increase in computation time (when compared with the other two 6 DoF tests).

Path	Test duration (seconds)	Velocity (meters / second)	CPU	Number of sensor points	Translation error (millimeters)		Rotation error (degrees)		Computation time for pose update (milliseconds)		Test nº
					Mean	Standard deviation	Mean	Standard deviation	Mean	Standard deviation	
Free fly	16.9	0.30	Intel Atom N2800	150	26.90	13.59	3.22	0.88	53.79	20.42	3.01
			Intel i5 3317U	425	19.16	10.19	3.07	0.67	35.03	9.75	3.02
			Intel i7-3630QM	425	19.44	9.63	3.07	0.64	25.94	7.15	3.03
Transla-tions	32.1	0.20	Intel Atom N2800	150	24.09	12.93	2.73	0.71	48.77	16.39	3.04
			Intel i5 3317U	425	16.89	9.83	2.75	0.57	33.58	9.30	3.05
			Intel i7-3630QM	425	16.18	7.84	2.76	0.57	24.83	6.61	3.06
Rotations	33.4	0.10	Intel Atom N2800	150	31.11	17.46	2.96	1.11	58.35	14.49	3.07
			Intel i5 3317U	425	18.67	12.61	2.53	0.96	62.85	25.40	3.08
			Intel i7-3630QM	750	18.82	12.30	2.55	0.92	41.51	17.39	3.09

Table 7. Self-localization test results performed with the Kinect sensor in the flying arena

Overall, the DRL system was able to track the Kinect pose with a mean translation error of 18 mm and a mean rotation error of 2.8° in the Intel i7 and i5 CPUs (it should be noted that the map resolution was 20 mm). For the Intel Atom, the number of points used in the registration (retrieved from the Kinect sensor) needed to be reduced by 60% in relation to the other CPUs, because the excessive computation time was leading to very high CPU usage and causing the localization system to ignore a significant amount of Kinect sensor scans. Adjusting the number of points increased the pose estimation error but allowed the Intel Atom to estimate the Kinect pose more often (even though it couldn´t update it at the same rate as the other CPUs, as can

be seen from Figures 22 to 27). Nevertheless, the Intel Atom was able to track the Kinect pose with 20–30 mm of mean translation error and 2–3° of mean rotation error with an acceptable update rate (30–50% less update rate than the Intel i5 and i7).

Figure 20. Overview of the map of the flying arena environment using the DRL system in SLAM mode

Path	CPU usage (% 0..100 * nº cores)		Memory usage (MB)		CPU cycles (GHz)	Instruc-tionsper cycle	Cache references (M/sec)	Cache misses (% of all cache references)	Branch instruc-tions (M/sec)	Branch misses (% of all branches)	Bus cycles (M/sec)	Test nº
	Mean	Standard deviation	Mean	Standard deviation								
Free fly	81.60	53.07	99.11	11.08	1.84	0.35	11.00	18.51	126.21	5.35	131.62	3.01
	54.42	50.29	107.18	5.28	2.39	1.61	11.33	18.75	734.72	1.96	95.89	3.02
	48.60	41.72	125.92	4.03	3.14	1.65	14.51	11.35	965.25	1.98	97.50	3.03
Transla-tions	84.91	52.01	100.22	8.45	1.84	0.34	10.88	18.12	122.53	5.34	131.73	3.04
	74.40	43.35	108.12	4.85	2.40	1.61	11.73	17.19	720.56	1.97	96.77	3.05
	57.82	36.14	126.76	3.78	3.15	1.61	15.74	11.99	953.30	1.99	98.34	3.06
Rotations	97.25	42.98	100.09	9.83	1.84	0.35	10.94	18.43	121.57	5.66	131.71	3.07
	78.67	44.90	106.99	4.96	2.33	1.64	9.73	18.91	690.02	2.05	96.67	3.08
	76.16	43.72	129.64	4.08	3.15	1.69	13.13	9.93	1000.23	1.98	98.35	3.09

Table 8. Profiling of self-localization tests performed with the Kinect sensor in the flying arena

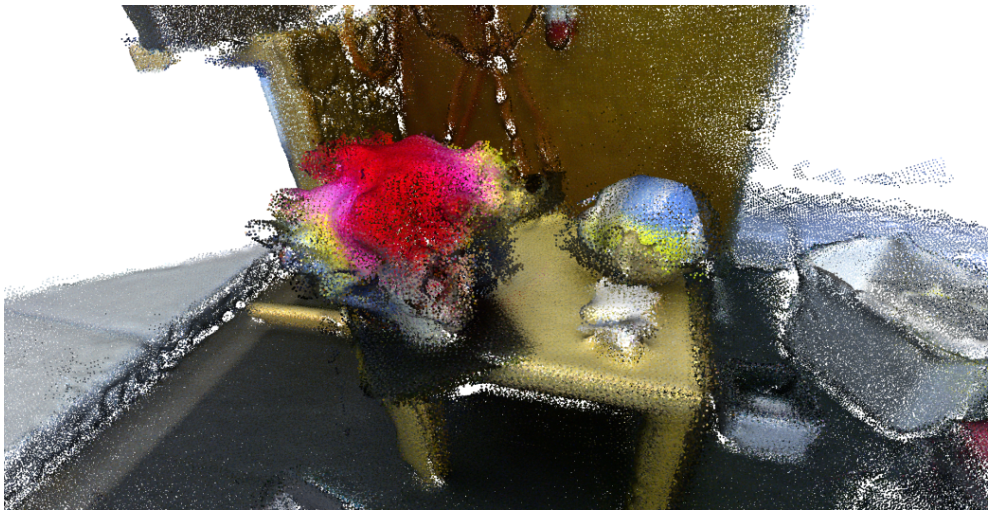

Figure 21. Map of the flying arena using the self-localization system in SLAM mode

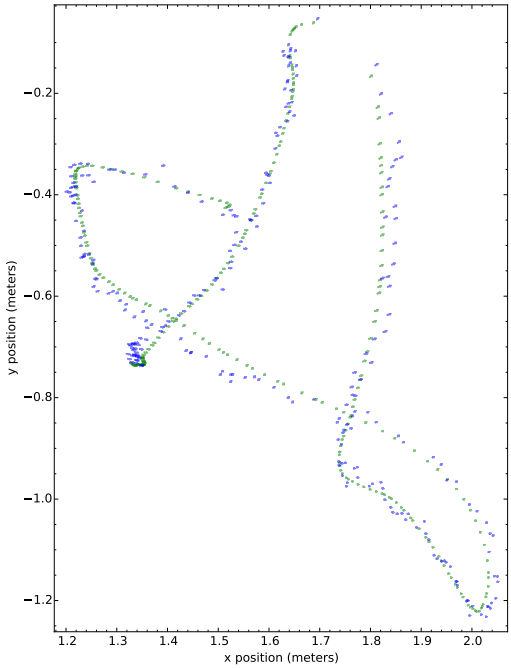

Figure 22. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 3.01

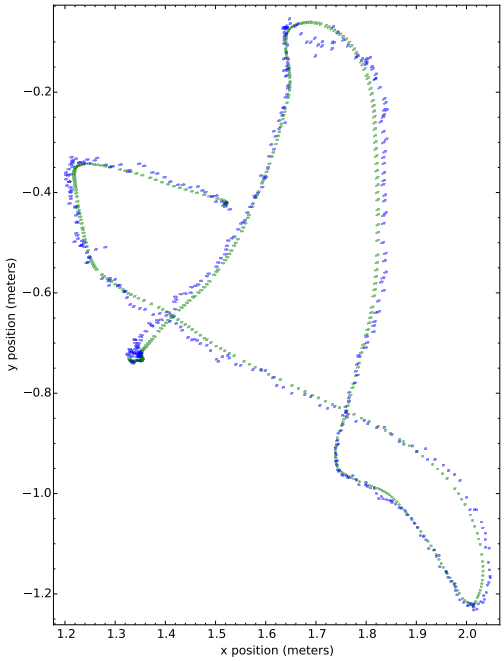

Figure 23. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 3.02

5. Global analysis of test results

Analyzing the test results shown from Tables 3 to 8 and Figures 28 to 31 (with the testing conditions presented in Table 9), it can be concluded that the DRL system can achieve very accurate 3 DoF pose tracking (mean translation error below 1 cm and mean rotation error below 1°) when the sensor data provided has a wide field of view (360°), a high angular resolution (0.25°), and a long range (250 m). These results (configurations j1 to j4) hold even when the Jarvis testing platform was moving at varying speeds (from 0.05 to 0.5 m/s) and with dynamic objects in the environment.

When the DRL system was tested under more challenging conditions (configurations p1 to p4 in the industrial hall), with the sensor data having a reduced field of view (180°), low angular resolution (1°), and short range (6 m), the DRL system still managed to track the robot 3 DoF pose with translation error below the map resolution (mean translation error below 2 cm and mean rotation error below 6°).

The DRL system was also able to track the Kinect 6 DoF pose (configurations k1 to k3) with less than 2 cm of mean translation error and less than 3° of mean rotation error, even when the sensor was aimed at unknown areas, which shows the robustness of the DRL system against temporary sensor problems, which happen more often on sensors with narrow field of view.

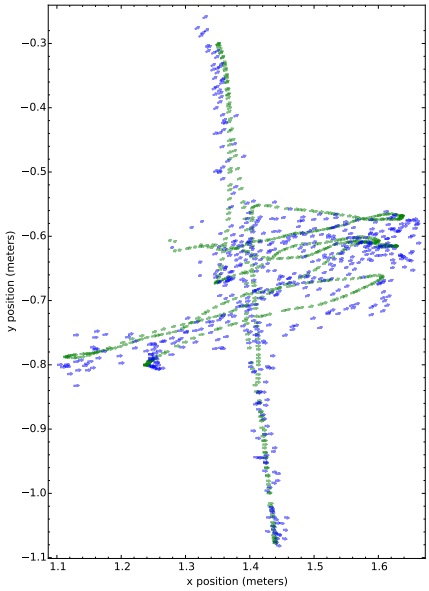

Figure 24. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 3.04

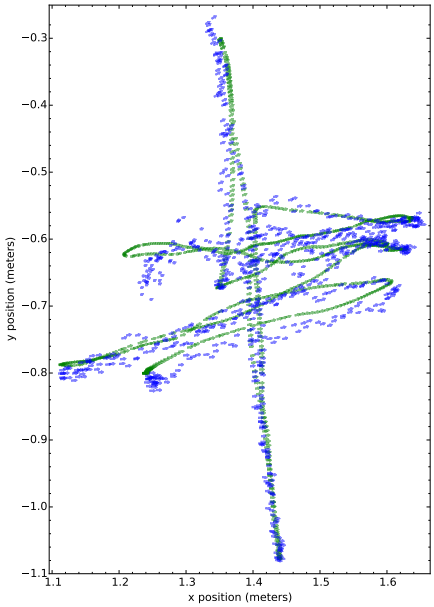

Figure 25. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 3.05

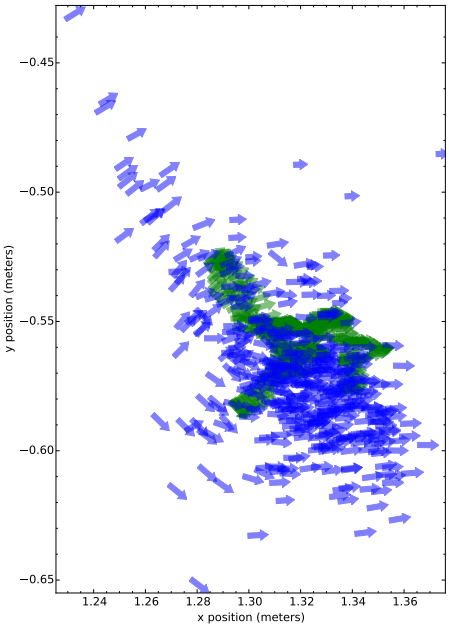

Figure 26. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 3.07

Overall, the DRL system was able to track the 3/6 DoF pose of the testing platforms (equipped with significantly different sensors) with high accuracy and reliability and overcame temporary sensor occlusions and dynamic objects in the environment.

Comparing the profiling results of all tests, it can be concluded that using more points in the reference or sensor point clouds leads to higher computation times (for each pose update) and higher CPU usage. This can be seen in the 6 DoF test results, which had a 5-time increase in the mean computation time and CPU usage when compared to the 3 DoF tests. As such, the 6 DoF pose estimation capabilities of the DRL system should only be used when the mobile platforms are not moving on planar environments.

The DRL system can perform 3/6 DoF pose estimation with CPUs with low power consumption and cost, such as the Intel Atom N2800. However, the standard deviation of the pose estimation, along with the resources required, will grow when compared with better CPUs. Moreover, these low-consumption CPUs may have memory access bottlenecks and given their limited processing capabilities, they may require the tuning of the DRL configurations in order to be able to perform pose estimations with acceptable update rate (which is why Figures 30 and 31 show a similar 3/6 DoF computation time and CPU usage on the tests performed with the Intel Atom N2800). This reconfiguration allowed the DRL system to update the Kinect pose more often at the cost of higher translation and rotation error. Nevertheless, this shows that the DRL system can be used on a wide range of mobile robot platforms with varying computational capabilities.

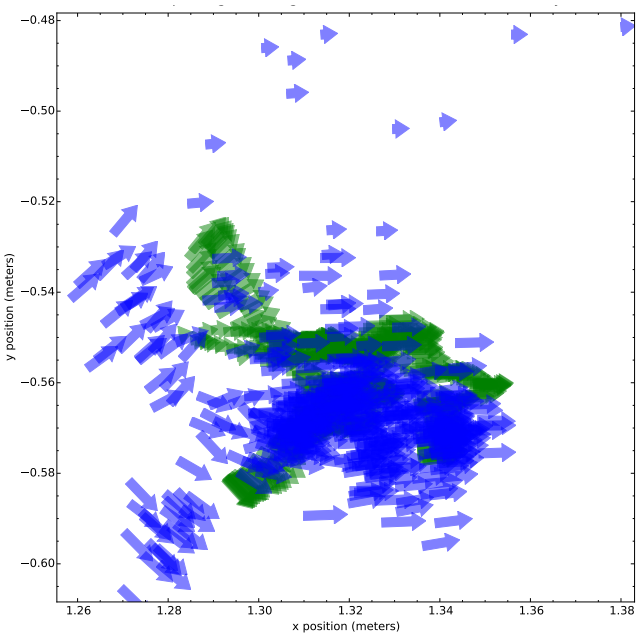

Figure 27. Poses estimated by the DRL (blue) and the ground truth (green) systems for test 3.08

Testing configurations	Sensor platforms	Movement path	Test duration (s)	Mean velocity (m/2)	Map resolution (mm)	Sensor range (m)	Filters	
							Voxel grid (m)	Random Sample
j1	Jarvis	Circular	290	0.05	10	250	0.02	500
j2			25.5	0.5				
j3		Complex	498	0.05				
j4			392	0.5-0.3-0.5-0.1				
p1	Pioneer	360	72	0.23	25	10	-	-
p2		Slam 1	155	0.26		6	-	-
p3		Slam 2	115	0.19			-	-
p4		Slam 3	112	0.16			-	-
k1	Kinect	Free fly	16.9	0.3	20	3.5	Atom -> 0.05 | i5/i7 -> 0.02	Atom -> 175 | i5/i7 -> 425
K2		Translations	32.1	0.2				Atom -> 175 | i5/i7 -> 425
K3		Rotations	33.4	0.1				Atom -> 175 | i5/i7 -> 750

Table 9. Testing configurations shown from Figure 28 to 31

http://dx.doi.org/10.5772/61258

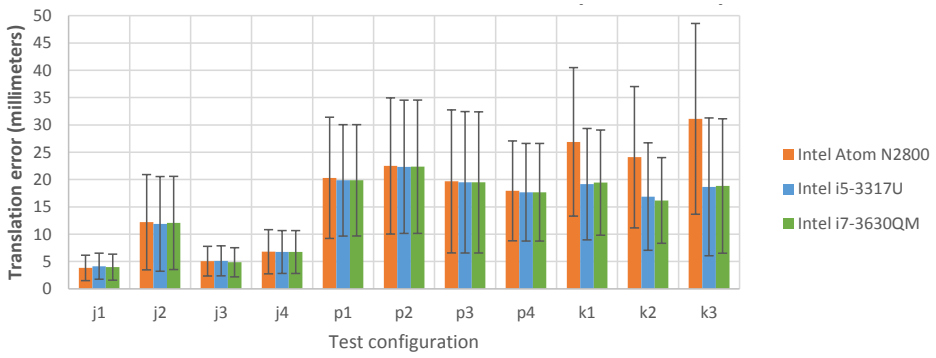

Figure 28. Translation error across all tests (testing configurations shown in Table 9)

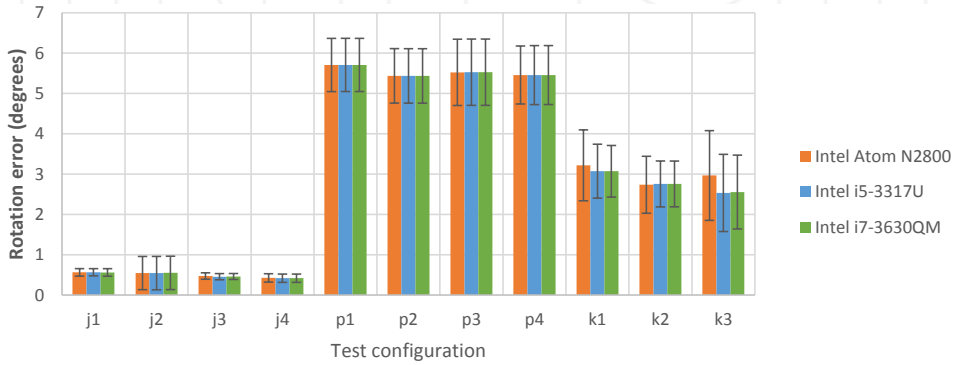

Figure 29. Rotation error across all tests (testing configurations shown in Table 9)

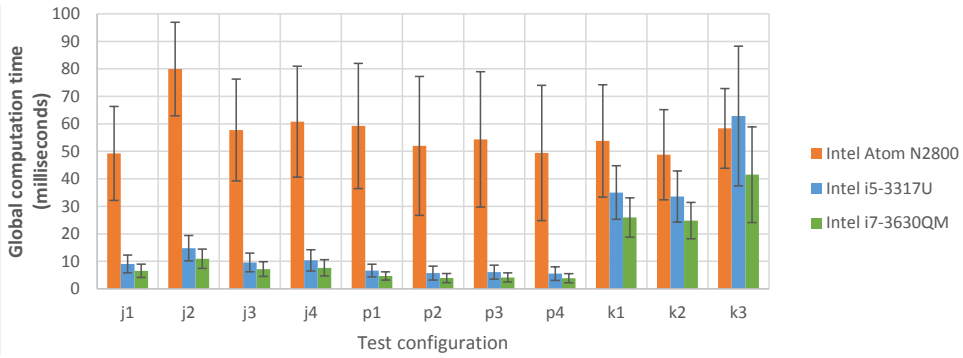

Figure 30. Computation time across all tests (testing configurations shown in Table 9)

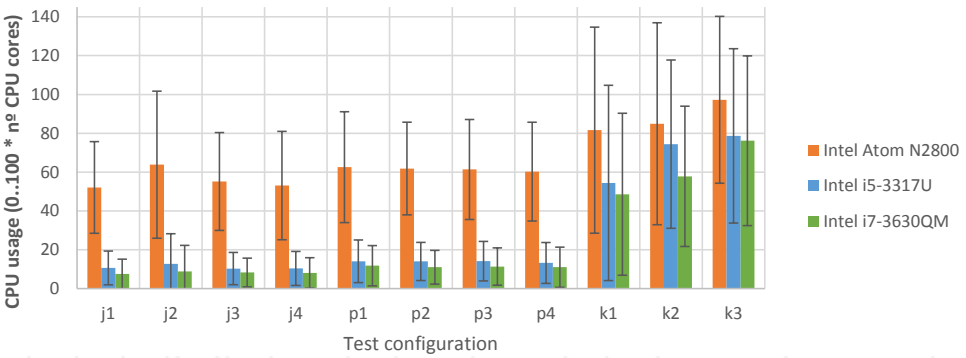

Figure 31. CPU usage across all tests (testing configurations shown in Table 9)

6. Conclusions

Mobile robot platforms require efficient software systems in order to perform their desired tasks without needing expensive and high power consumption hardware. Given the wide range of hardware and sensor configurations and the set of tasks that a mobile robot can perform, this article presented a detailed analysis of the DRL system performance in tests running on three different computing platforms equipped with CPUs ranging from low to high end processing capabilities.

The DRL system was tested in challenging environments and was able to perform high accuracy pose estimation with mean translation error between 5 and 30 mm and mean rotation error between 0.4° and 5° in both 3 and 6 DoF. The trade-offs between pose estimation accuracy and computing resources required can be tuned to the specific needs of the tasks performed by the robot, allowing efficient use of the localization system on low computing power mobile robot platforms.

For the presented tests, and with some configuration fine tuning, the Atom N2800 CPU was able to estimate the 6DoF pose with about 30 mm/3° in 60 ms at over 90% CPU load. The other superior CPUs, Intel Core i5 and i7, were able to estimate 6 DoF poses with about 20 mm/3° in 30 ms at 80% CPU load.

Moreover, several sensors can be used simultaneously in order to increase the field of view seen by the localization system, allowing more accurate and stable estimation of the robot's pose. Besides pose tracking, the self-localization system can also perform initial pose estimation when the robot starts its operation or when it becomes lost in the environment. It can also incrementally build a map of its surroundings with probabilistic integration and removal of geometry and perform surface reconstruction to minimize the impact of sensor noise.

The robust and high accuracy pose tracking capabilities of the DRL system in conjunction with the global pose estimation and mapping modules allow the fast deployment of a wide range of mobile robot platforms in cluttered and dynamic environments.

Acknowledgements

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".

Author details

Carlos M. Costa, Héber M. Sobreira, Armando J. Sousa* and Germano Veiga

*Address all correspondence to: asousa@fe.up.pt

INESC TEC (formerly INESC Porto) and Faculty of Engineering, University of Porto, Portugal

References

[1] Mautz, R., 2012. Indoor Positioning Technologies. Ph.D. thesis, ETH Zürich.

[2] Wan, E., Van Der Merwe, R., 2000. The unscented Kalman filter for nonlinear estimation. In: IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium.

[3] Thrun, S., 2002. Probabilistic robotics. Communications of the ACM.

[4] Lingemann, K., Nüchter, A., Hertzberg, J., Surmann, H., 2005. High-speed laser localization for mobile robots. In: Journal of Robotics and Autonomous Systems.

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

[6] Magnusson, M., 2009. The three-dimensional normal-distributions transform - an efficient representation for registration, surface analysis, and loop detection. Ph.D. thesis, Örebro University.

[7] Kitt, B., Geiger, A., Lategahn, H., 2010. Visual odometry based on stereo image sequences with RANSAC-based outlier rejection scheme. In: IEEE International Conference on Intelligent Vehicles Symposium.

[8] Whelan, T., Johannsson, H., Kaess, M., Leonard, J.J., McDonald, J., 2013. Robust real-time visual odometry for dense RGB-D mapping. In: IEEE International Conference on Robotics and Automation.

[9] Costa, C., 2015. Robot Self-localization in dynamic environments. M.Eng thesis, FEUP.

[10] Costa, C., Sobreira, H., Sousa, A., Veiga, G., 2015. Robust and accurate localization system for mobile manipulators in cluttered environments. IEEE International Conference on Industrial Technology.

[11] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Berger, E. Wheeler, R., 2009. ROS: An open-source Robot Operating System. In: IEEE International Conference on Robotics and Automation.

[12] Rusu, R., Cousins, S., 2011. 3D is here: Point Cloud Library (PCL). In: IEEE International Conference on Robotics and Automation.

[13] Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., Burgard, W., 2013. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Journal of Autonomous Robots.

[14] Lowe, D., 2004. Distinctive image features from scale-invariant keypoints. Journal of Computer Vision.

[15] Zhong, Y., 2009. Intrinsic shape signatures: A shape descriptor for 3d object recognition. In: IEEE International Conference on Computer Vision.

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

[17] Rusu, R., Blodow, N., Beetz, M., 2009. Fast point feature histograms (FPFH) for 3d registration. In: IEEE International Conference on Robotics and Automation.

[18] Tombari, F., Salti, S., Di Stefano, L., 2011. A combined texture shape descriptor for enhanced 3d feature matching. In: IEEE International Conference on Image Processing.

[19] Frome, A., Huber, D., Kolluri, R., Bülow, T., Malik, J., 2004. Recognizing objects in range data using regional point descriptors. In: European Conference on Computer Vision.

[20] 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.

[21] Wohlkinger, W., Vincze, M., 2011. Ensemble of shape functions for 3d object classification. In: IEEE International Conference on Robotics and Biomimetics.

[22] Besl, P., McKay, N., 1992. A method for registration of 3-d shapes. In: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[23] Segal, A., Haehnel, D., Thrun, S., 2009. Generalized-ICP. In: Proceedings of Robotics: Science and Systems.

[24] Sturm, J., Engelhard, N., Endres, F., Burgard, W., Cremers, D., 2012. A benchmark for the evaluation of RGB-D SLAM systems. In: IEEE/RSJ International Conference on Intelligent Robots and Systems.

[25] Pomerleau, F., Magnenat, S., Colas, F., Liu, M., Siegwart, R., 2011. Tracking a depth camera: Parameter exploration for fast ICP. In: IEEE/RSJ International Conference on Intelligent Robots and Systems.

INTECH

INTECH