Tracking a depth camera: Parameter exploration for fast ICP

François Pomerleau, Stéphane Magnenat, Francis Colas, Ming Liu, Roland Siegwart

Tracking a Depth Camera: Parameter Exploration for Fast ICP

François Pomerleau[1] and Stéphane Magnenat[1] and Francis Colas[1] and Ming Liu[1] and Roland Siegwart[1]

***Abstract*— The increasing number of ICP variants leads to an explosion of algorithms and parameters. This renders difficult the selection of the appropriate combination for a given application. In this paper, we propose a state-of-the-art, modular, and efficient implementation of an ICP library. We take advantage of the recent availability of fast depth cameras to demonstrate one application example: a 3D pose tracker running at 30 Hz. For this application, we demonstrate the modularity of our ICP library by optimizing the use of lean and simple descriptors in order to ease the matching of 3D point clouds. This tracker is then evaluated using datasets recorded along a ground truth of millimeter accuracy. We provide both source code and datasets to the community in order to accelerate further comparisons in this field.**

I. Introduction

Laser-range sensors were a cornerstone to the development of mapping and navigation in the last two decades. Nowadays, rotating laser scanners, stereo cameras or depth cameras (RGB-D) can provide dense 3D point clouds at a high frequency. Using the Iterative Closest Point (ICP) algorithm [1], [2], these point clouds can be matched to deduce the transformation between them and consequently the 6 degrees-of-freedom motion of the sensor.

ICP is a popular algorithm due to its simplicity. This leads to hundreds of variations around the original algorithm that were demonstrated on various experimental scenarios. Because of the lack of a common comparison framework, the selection of an appropriate combination is difficult.

The chief assumption of ICP is that the association between points is mainly correct using the closest point. If not, the computed transformation may be irrelevant. There are typically two ways to ensure that the association is correct: attaching descriptors to the points to ease disambiguation, or applying the ICP algorithm fast enough to limit the magnitude of changes. Descriptors are widely used in the vision community to match images and recently 3D descriptors have been introduced to help the association step of ICP (see [3] or [4] as recent examples). While this approach is promising, most elaborate descriptors are still too costly to compute for online processing.

The first contribution of this paper is an open-source modular ICP library, which allows to compare several "flavored" ICP solutions within the same framework. This library is released together with our implementation of nearest-neighbor search with kd-tree, called libnabo[1], which has comparable to slightly superior performance to ANN[2], thanks to more compact data structures. Moreover, libnabo features a modern, template-based interface.

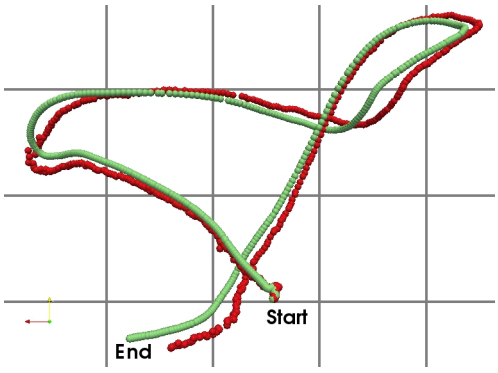

Fig. 1. One path of depth camera, tracked at 30 Hz. Projection on the xy-plane of the tracked position (red) versus the measured ground truth (light green). Each grid square is half a meter.

Our second contribution is to optimize the use of lean and simple descriptors to produce an ICP-based 3D pose estimator at the frame rate of modern RGB-D sensors. This pose estimator, or tracker, could in turn be used to feed more precise algorithms requiring more time to model the word, for instance SLAM. In this paper, we focus on improving the speed of the tracker while keeping the pose estimation in a usable range. This is done by exploiting the modularity of our ICP library to adapt different filters.

We show statistical analysis of the tracker behavior in the context of indoor navigation using a Microsoft Kinect. We performed this evaluation using datasets recorded along a ground truth of millimetric accuracy that we make available[3]. Fig. 1 presents an example of one of the 27 paths recorded.

II. Related Works

Several recent works have focused on the speed of ICP algorithms. The search for the closest point is one of the bottlenecks of ICP. Using an approximate kd-tree decreases computational time of ICP [5]. Approximate kd-trees employ distance thresholds to limit the search at the risk of returning sub-optimal neighbors [6]. This allows to increase the overall speed of the algorithm while the redundancy between points prevents a decrease in performance. Additionally, Zlot et al. compare kd-tree, locality-sensitive hashing and spill-trees [7]. They conclude that kd-tree is better in terms of accuracy, query time, build time, and memory usage. They also observe that huge approximations can reduce the query time by two orders of magnitude while keeping a sufficient accuracy.

(1) Autonomous Systems Lab. – ETH Zurich, Tannenstr 3, 8092 Zürich, Switzerland firstname.lastname@mavt.ethz.ch

[1] http://github.com/ethz-asl/libnabo

[2] http://www.cs.umd.edu/~mount/ANN/

[3] Datasets used for this article can be downloaded at: http://www.asl.ethz.ch/research/datasets

Another research direction explore multiple resolutions. Jost et al. compute several times the ICP while varying the resolution from coarse to fine [8]. At a coarse resolution, i.e. with a limited number of points, ICP converges faster but with less accuracy than at a fine resolution. However, by initializing a finer-resolution ICP with the result of the coarser one, the convergence of the fine-resolution ICP is much faster than single-shot ICP as the initial alignment is mostly correct. These authors also use a pre-computed list of nearest neighbors to approximate the matching step. With both of these techniques, they show a significant increase of the speed of ICP while maintaining an adequate robustness. For the same absolute performance as standard ICP, Li et al. [9] obtain less iterations at higher resolution which decreases the total time of a factor of 1.5 in 2D and 2.5 in 3D. The multi-resolution approach can also increase the search speed for the closest point by using a hierarchical-model point selection with a stereo camera [10]. By subsampling the space and with the help of the structure of the sensor, this solution can achieve a speed gain of 3 with respect to standard ICP with kd-tree search. In this case, the use of the structure of the depth image increases the speed of matching. In the same direction, the specificity of a 2D laser scanner can help optimize search [11]. However, these optimizations are oriented toward specific sensors, which makes them hard to generalize and are not suitable for a multi-sensor setup.

III. Modular ICP

ICP is an iterative algorithm performing several sequential processing steps, both inside and outside the main loop. For each step, there exist several strategies, and each strategy demands specific parameters.

To our knowledge, there is currently no easy way to compare these strategies. To enable such a comparison, we have developed a modular ICP chain (see Fig. 2), called libpointmatcher, that we provide as open-source software[4]. This chain takes as input two point clouds and estimates the translation and the rotation that minimize the alignment error. We call the first point cloud the *reference* and the second the *reading*. The ICP algorithm tries to align the reading onto the reference. To do so, it first applies some filtering to the clouds, and then iterates. In each iteration, it associates points in reading to points in reference and finds a transformation of reading minimizing the alignment error. The ICP chain consists of several steps. A *data filter* takes a point cloud as input, transforms it, and produces another cloud as output. The transformation might add informations, for instance surface normals, or change the number of points by randomly removing some of them for example. Data filters can be chained. A *matcher* links points in the reading to points in the reference. Currently, we provide a fast k-nearest-neighbor matcher based on a kd-tree, using libnabo. A *feature outlier filter* removes (hard rejection) and/or weights (soft rejection) links between points in reading and their matched points in reference. Criteria can be a fixed maximum

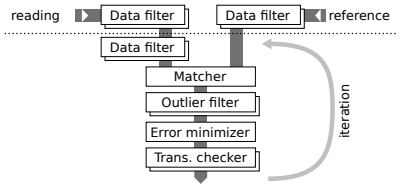

Fig. 2. The modular ICP chain as implemented in libpointmatcher

authorized distance, a factor of the median distance, etc. Points with zero weights are ignored in the subsequent minimization step. Feature outlier filters can be chained. An *error minimizer* computes a transformation matrix such as to minimize the error between the reading and the reference. There are different error functions available, such as point-to-point or point-to-plane. A *transformation checker* can stop the iteration depending on some conditions. For example, a condition can be the number of times the loop was executed or it can be related to the matching error. Transformation checkers can be chained.

This chain provides standardized interfaces between each step. This allows to add novel algorithms to some steps to evaluate their impact on the global ICP behavior.

IV. Tracker

A. From ICP to Tracking

Using libpointmatcher, we implement a fast tracker that we also provide as open-source software[5]. This tracker takes as input a stream of point clouds and produces as output an estimation of the 6D pose of the sensor. To avoid drift, the tracker holds a single reference and matches every incoming point cloud against it. If the ratio of matching points drops below a pre-defined threshold, the tracker creates a new reference with the current cloud. This keyframe-based mechanism allows a higher frame rate by reducing the number of kd-tree creation and limits drift if the sensor stay at the same position. To easily explore the different parameters that affect the performance of the ICP algorithm, the ICP chain is completely configurable at run time.

We provide two versions of the tracker, an online one integrated with ROS and an offline one. The ROS version provides real-time tracking of the sensor pose, and publishes the latter as `tf`, the standard way to describe transformations between reference frames in ROS. The offline version ensures that no cloud would be dropped and therefore improves the consistency of the measurements. This also allows us to run experiments in batch without being limited by the frame rate of the sensor. This version takes as input a dataset file and a text-based list of configurations and parameters. The offline tracker uses these to reconfigure the ICP chain for each experiment. We use the offline version to produce the results shown in this paper.

B. Experimental Setup

We want to quantify parameters that affect the tracking speed and accuracy. To do so, we employ the Kinect sensor.

[4] `http://github.com/ethz-asl/libpointmatcher`

[5] `http://www.ros.org/wiki/modular_cloud_matcher`

(a) (b) (c)

Fig. 3. Experimental environments of (a) low complexity, (b) medium complexity, and (c) high complexity

We acquire several datasets in a ROS environment, using the Kinect OpenNI driver[6] and `rosbag` to record the data. We run the experiment in a room equiped with a Vicon tracking system[7] [12]. The later provides ground-truth position in the order of millimeter. We thank Prof. Raffaello D'Andrea who has given us access to this tracker-equiped room.

In their comparison of ICP performance, Rusinkiewicz and Levoy used three synthetic environments composed of low-frequencies, all-frequencies, and high-frequencies surfaces with some added noise [13]. We reuse this concept and transpose it in a real indoor experimental setup. We assemble 3 different static environments of increasing complexity (Fig. 3). For each complexity, an operator performs 3 types of motions: translations on the three axes (for about 10 s per axis), rotations on the three axes (for about 10 s per axis), a free fly motion over the scene (for about 15 s). We perform each type of motions, for all environments, at 3 different speeds: slow motion with speed in the range of indoor ground robots (around 0.3 m/s), medium motion with speed in the range of agile robots (around 0.5 m/s), fast motion with a challenging speed (around 1.3 m/s).

This gives us 27 datasets with point clouds produced by the Kinect at 30 Hz and its pose tracked by the Vicon at 100 Hz. We use a resolution of 160×120 depth pixels to generate the point clouds, which creates clouds containing at most 19200 points, as some points from the sensor are invalid.

C. Measurement Method

To compare the various parameters affecting the quality of the registration, we define an error metric. To provide robustness against noise, we cumulate the path over 30 registrations and then compute the error in translation e_t and in rotation e_r. The error in translation corresponds to the Euclidean distance between the pose estimated through ICP and the Vicon pose. The error in rotation corresponds to the absolute angular difference.

The tracker only relies on environmental information without any prior. Thus, the registration might fail depending on what the sensor sees. The modular ICP detects such cases and outputs an identity transform. We keep track of those failures N_{fail} over a dataset having a number of registrations N_{icp}. In the case of free-fly–motion datasets, $N_{\text{icp}} = 447$. In the case of translation and rotation datasets, $N_{\text{icp}} = 838$. We define an ICP performance metric for a given dataset:

$$\text{perf} = \frac{N_{\text{icp}} - N_{\text{fail}}}{N_{\text{icp}}} \frac{1}{\text{median}(e_t)} \tag{1}$$

The first fraction gives the success ratio while the second one is the inverse of the median error of the dataset. The intuition behind the use of this performance metric, instead of directly using the error, is that we expect time and performance curves to have similar trends. If the evolution of the curves follow the same tendency, it is difficult to devise a clear parameter optimum. The success ratio compensates the fact that the library returns an identity transformation if a failure happens, which could be close to the ground-truth value when the movement is slow. With this ratio, the performance will be 0 if all registrations fail and equal to the inverse of the translation error if all registrations succeed. We can define a similar metric using the rotation error e_r; experiments show similar results as with e_t.

Along the performance, we also measure the time. We divide the time to register the whole dataset by N_{icp} to compute the average time by ICP call. This provides a better accuracy than measuring time at every ICP call individually.

V. Experiments

The modular ICP allows many possible combinations of algorithms and parameters. In this paper, our aim is to enable a fast registration while keeping a reasonably precise pose estimation. Thus, we focus on simple solutions regarding sensor-noise modeling, point selection, and matching.

A typical experiment on a dataset implies a single value of time and performance over N_{icp} for a given parameter. Then, we repeat the computation N_{test} times to increase statistical significance, as some filters introduce randomness. We again repeat these over a range of parameters N_{par} for different datasets. Such experimentation gives us a graph like Fig. 4a. Then, to ease interpretation of results, we use robust estimators (i.e. median or quantiles as opposed to mean or variance) to extract the mode and the dispersion of the distribution for a given parameter. Fig. 4b shows the extraction of the median in blue with the 10% and 90% quantiles in dashed red. We observed that, in our experiments, quantiles follow the same tendency as the median so in further graphs, we only present the median for the sake of readability.

We first explore parameters related to sensor noise (Section V-A), subsampling (Section V-B), and nearest-neighbor (NN) approximation (Section V-C). We use the datasets with the free-fly motion at low speed within the three types of environments. Given the resulting optimized parameters, we evaluate the robustness against all 27 datasets and also look at the effect of the hardware on the processing speed (Section V-D). All these experiments use a different number of tests and parameters. Table I summarizes the configuration of each experiment, with the final column representing the total number of ICP computed per experiment expressed as a factor of 1'000'000. The total number of registration required for the experimental section of this article is around 11 millions.

[6] `http://www.ros.org/wiki/ni`
[7] `http://www.vicon.com/`

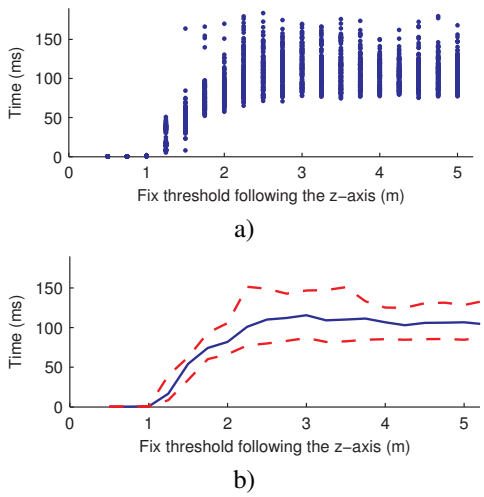

Fig. 4. Example of result processing (see Section V-A). a) raw results and b) median of the results in solid blue, 10% and 90% quantiles in dashed red.

TABLE I
NUMBER OF ICP PER EXPERIMENT

Experiment Names	N_{icp}	N_{par}	N_{test}	Total (M)
Sensor noise (fixed)	3×447	20	45	1.2
Sensor noise (ratio)	3×447	19	45	1.2
Subsampling (ratio)	3×447	39	30	1.6
Subsampling (step)	3×447	39	30	1.6
NN approximation	3×447	20	60	1.6
Robustness	9×447 $+18 \times 838$	1	20	3.8
Hardware speed	1×447	39	20	0.4

Additionally, we fix the error minimization solution being the point-to-plane error [2] and the outlier filter being the median distance [14] for all experiments.

A. Sensor Noise

The first experiment tackles how to handle the sensor noise in the registration. Based on parallax, the Kinect has an accuracy on the depth inversely proportional to the distance. Moreover, it has a dead zone of 0.4 m close to the sensor. We explore 2 techniques: a fixed threshold to prune points over a certain depth, a ratio of points to keep with the smallest depths. Both these techniques eliminate points farther then a certain distance. One could also employ weighted minimization to handle sensor noise, but as we wish to optimize processing time, dropping points is more efficient.

The results for the fixed threshold (Fig. 5a) show that below 1.5 m, this method does not yield enough points to ensure registration. As the threshold increases from 1.5 m to 5 m, the performance and the time follow a similar curve, essentially monotonic. The reason is that the average depth of what is being seen changes, and setting a fixed threshold leads to a lack of points in some situations. On the contrary, using a percentage of points has a different behavior. As Fig. 5b shows, between a ratio of 0.4 and 0.6, the performance is higher than using all the points (i.e. with a ratio of 1) while the time is divided by half. Indeed, keeping less than 40 % of the points reduces chances to take advantage of important constraints and using all the points does not cut off any noise. Therefore, for further experiments we select the second technique with a ratio of 0.4.

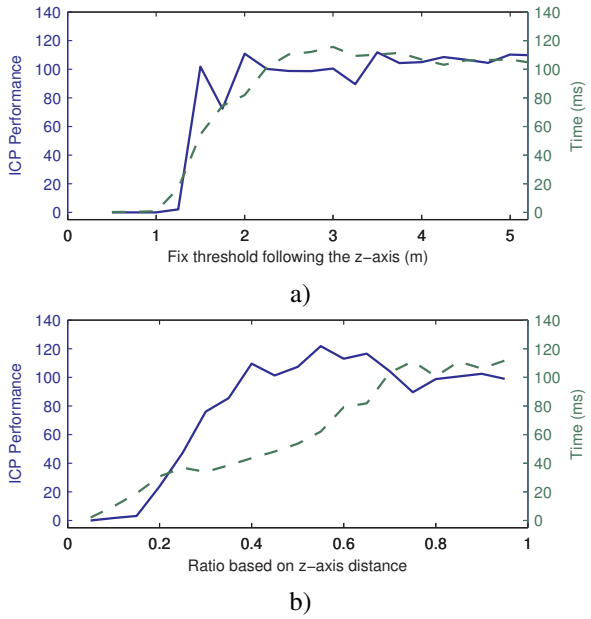

Fig. 5. Performance and time for sensor-noise thresholds with a) parameters based on fixed distances, and b) parameters using quantiles. Solid blue represents ICP performance and dashed green represents time for convergence.

B. Subsampling

The second experiment evaluates how much we can subsample the cloud without loosing too much performance. Again, we compare two techniques: randomly selecting points using a uniform distribution, and keeping only one point every n points. More complex subsampling techniques exist, to compensate the radial distribution of 3D scanners [15] or to select points leading to more constrained environments [16], but these are too slow to fit in the scope of this work.

From Fig. 6a, we observe that the time follows linearly the ratio of points used while the performance follows an exponential convergence. The step technique results (Fig. 6b) show an exponential reduction of the time while the general tendency of the performance is to reduce linearly. It is worth noting that parameters of the step technique are discrete which is shown using the filled dots on the time curve. The performance of the step subsampling shows more jitters than the one of the random selection. We attribute this to artificial patterns in scans due to the fixed-step nature of this technique.

We conclude that the random-subsampling technique gives us more control on the desired computation time and is less likely to produce artifacts in the resulting scans than the fixed-step technique. Moreover, we compare time for both techniques in relation with the number of points kept: the extra computation required for the random sampling does not augment the computational time significantly. Since there is no optimum for that parameter we accept the fact that going fast increases error and we select a subsampling ratio of 0.3.

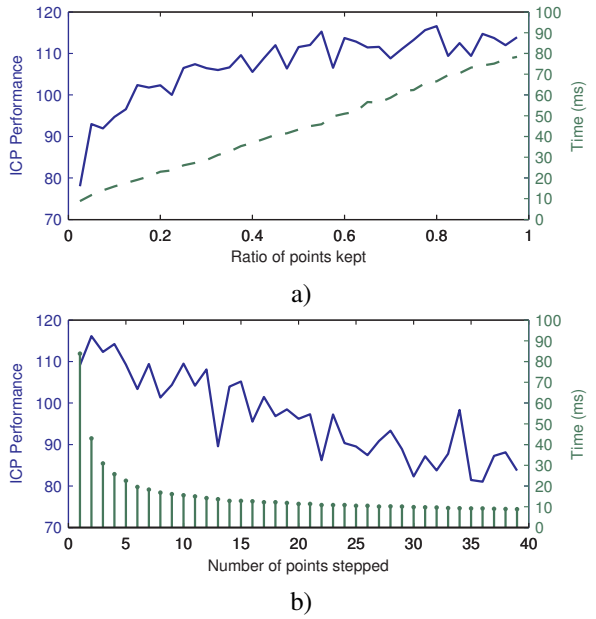

Fig. 6. Performance and time for subsampling methods with a) random selection, and b) fixed step based on fixed skip. Solid blue represents ICP performance and dashed green represents time for convergence.

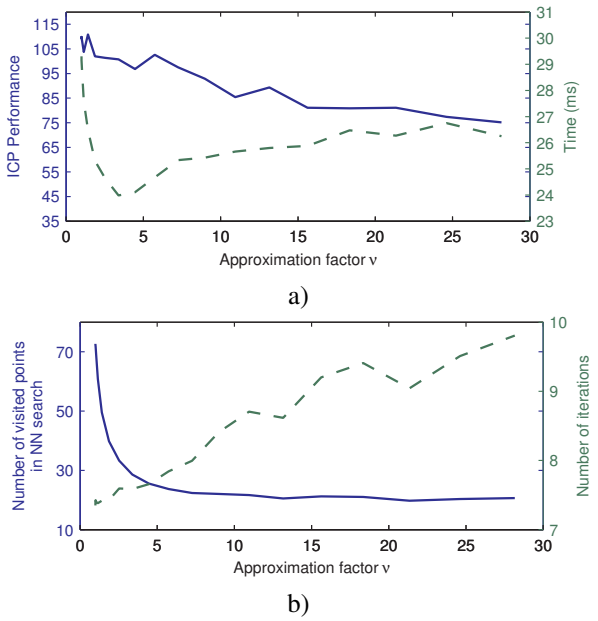

Fig. 7. Performance and time for approximate search using a kd-tree. a) ICP performance in Solid blue and convergence time in dashed green. b) The average number of visited points per NN request expressed in 1000 in solid blue and the number of iterations per ICP in dashed green.

C. Nearest-Neighbor Approximation

This experiment stems from the observation that using an approximate NN-search leads to faster registration without affecting the error much [5] [7], compared to an exact search. We implement the NN-search using an approximate kd-tree as in [6] and vary ν, the approximation factor[8].

In Fig. 7a, we see that when ν increases, both the time and the performance decrease, but the latter decreases slower than the former. Moreover, the time decreases rapidly to a minimum and then increases again. The reason is that while the number of points visited in the kd-tree decreases exponentially with ν, the number of iterations required by the ICP to converge increases linearly (Fig. 7b). Given those results, we selected $\nu = 3.3$. It is interesting to note that this is the same optimal value as reported briefly in [7].

D. Robustness Evaluation

Using the selected parameters, we compare the tracking error for different motion velocities, motion types, and environment complexities (27 trajectories). Fig. 8 presents the results of the tracker translation error directly in meter for each tracking second instead of the performance metric used in former experiments. The error on translation for the three graphs is represented following a common log scale on the y-axis to highlight differences at low value. The box plots represent quartiles with the vertical red line being the median and the "+" symbols being outliers over 99.3% coverage of the distribution.

The most important relation is that the error increases significantly as a function of the motion velocity with the median being outside the first quartile of each velocity clusters.

[8]We defined $\nu = \sqrt{1+\epsilon}$ where ϵ is the approximation constant defined in [6].

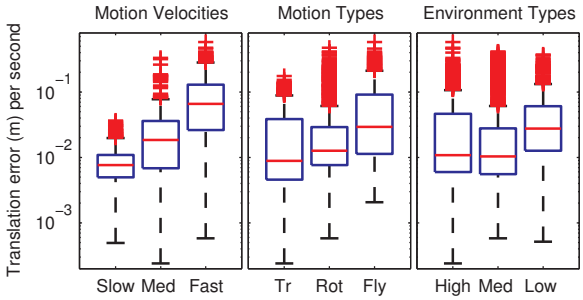

Fig. 8. The error as function of motion velocities, motion types, and environment types

We also observe this effect with the percentage of failures, which has a median value of 0% for the slow motion and going up to 32% for the fast motion (not present in the graph). Translation motions are the easiest to register followed by rotational movements and free-fly movements where larger accelerations are present. We note that the low-complexity environment is harder to register than the high and medium complexity ones. The reason is that the low-complexity environment contains very few planes and they are rarely all in the field of view of the Kinect, leading to some under-constrained dimensions.

In our experience, the main factor influencing registration speed is the number of points randomly subsampled. Demanding a low-processing time means accepting more error from the registration. Since this processing time highly depends on the computer, we tested three different processors with an increasing number of points kept. Note that the algorithm is not multi-threaded and does not employ any GPU acceleration,

which allows us to compare the performance with embedded systems. The systems are: a recent laptop with an Intel Core i7 Q 820, an old desktop PC with an Intel Xeon L5335, an embedded system with an Intel Atom CPU Z530.

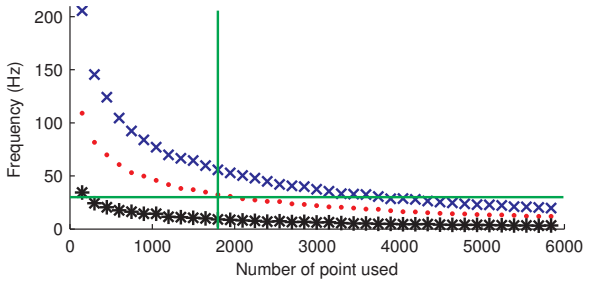

Fig. 9. Comparison of time vs ratio of points used for different hardwares: Intel Core (blue "x"), Intel Xeon (red "."), and Intel Atom (black "*")

Results from Fig. 9 show a significant difference in the range of frequencies between different systems. To help interpreting the graph, the horizontal green line represents 30 Hz which (i.e. the minimum frequency available for real-time operations using a Kinect) and the vertical green line represents a minimum number of points selected in the subsample experiment (Section V-B). Recent processors can process up to 3700 points at 30 Hz or one can reduce the number of points to free processing time for other programs. Note that the Atom can run at most at 10 Hz, with the minimal number of points. Based on former experiment with quadcopters [17], a control loop needs to run between 5 and 10 Hz to cope with the dynamic of such a system. This shows that our tracker is usable on Unmanned Aerial Vehicles; we will conduct further tests in this direction.

VI. Conclusions and Future Works

We presented first an efficient and modular open-source ICP library. Its modularity allows to quickly test and compare different variants of ICP algorithms. Based on this library, we designed and optimized a 3D pose tracker for dense depth cameras running at 30 Hz on standard laptop with thousands of points. As it does not use GPU acceleration, the tracker can also run on embedded system (10 Hz on a Atom board). Finally we proposed a sound performance evaluation using datasets recorded with a ground truth of millimeter accuracy.

It is very difficult to find a general solution to all problems using ICP. We can optimize a particular ICP implementation by identifying environmental characteristics and typical motions expected for a given application. One must also take into account sensor frequency, noise, and field of view to devise a robust registration strategy. From a robotic-application point of view, the robustness experiment shows that pose-tracking in cluttered rooms, typically encountered in apartments or offices, is easier than tracking in corridors of public buildings or in places with few furnitures. To cope with this, one could adjust the speed of the robot as a function of the complexity of the environment. One should also limit the rotational velocity when the curvature of the sensor path is large.

Hierarchical subsampling also increases speed as highlighted in the introduction. However, further investigation is required to optimize according to specific applications.

We also intend to augment the diversity of building blocks available in the modular ICP library to increase the space of possible algorithm comparisons.

VII. Acknowledgements

The research presented here was supported by the EU FP7 IP projects *Natural Human-Robot Cooperation in Dynamic Environments (ICT-247870;* `http://www.nifti.eu`*)* and *myCopter (FP7-AAT-2010-RTD-1,* `http://www.mycopter.eu`*)*. François Pomerleau was supported by a fellowship from the Fonds québécois de recherche sur la nature et les technologies (FQRNT). We thank Prof. Raffaello D'Andrea for letting us use the Flying Machine Arena and its Vicon system.

References

[1] P. Besl and H. McKay, "A method for registration of 3-D shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 2, pp. 239–256, 1992.

[2] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," *Robotics and Automation. Proc. of IEEE International Conf. on*, vol. 3, pp. 2724 – 2729, 1991.

[3] K. Pathak, A. Birk, N. Vaskevicius, and J. Poppinga, "Fast registration based on noisy planes with unknown correspondences for 3-D mapping," *Robotics, IEEE Transactions on*, vol. 26, no. 99, pp. 1–18, 2010.

[4] Y. Zhuo and X. Du;, "Automatic registration of partial overlap three-dimensional surfaces," *Mechanic Automation and Control Engineering, Proc. of the International Conf. on*, pp. 299–302, 2010.

[5] A. Nuchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6D SLAM with approximate data association," *Advanced Robotics, Proc. of the International Conf. on*, pp. 242–249, 2005.

[6] S. Arya and D. Mount, "Approximate nearest neighbor queries in fixed dimensions," *Discreet Algorithms. Proc. of the fourth annual ACM-SIAM Symposium on*, pp. 271–280, 1993.

[7] R. Zlot and M. Bosse, "Place recognition using keypoint similarities in 2D lidar maps," *Experimental Robotics*, 2009.

[8] T. Jost and H. Hugli, "A multi-resolution ICP with heuristic closest point search for fast and robust 3D registration of range images," *3-D Digital Imaging and Modeling, Proc. of the International Conf. on*, pp. 427–433, 2003.

[9] C. Li, J. Xue, S. Du, and N. Zheng;, "A fast multi-resolution iterative closest point algorithm," *Pattern Recognition, Proc. of the Chinese Conf. on*, pp. 1–5, 2010.

[10] D. Kim, "A fast ICP algorithm for 3-D human body motion tracking," *Signal Processing Letters, IEEE*, vol. 17, no. 4, pp. 402–405, 2010.

[11] A. Censi, "An ICP variant using a point-to-line metric," *Robotics and Automation. Proc. of IEEE International Conf. on*, pp. 19–25, 2008.

[12] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," may 2010, pp. 1642–1648.

[13] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," *3-D Digital Imaging and Modeling. Proc. of International Conf. on*, pp. 145–152, 2001.

[14] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," *Pattern Recognition. Proc. of the International Conf. on*, vol. 3, pp. 545 – 548, 2002.

[15] D. Gingras, T. Lamarche, J. Bedwani, and E. Dupuis, "Rough terrain reconstruction for rover motion planning," *Computer and Robot Vision, Proc of the Canadian Conf. on*, pp. 191–198, 2010.

[16] N. Gelfand, L. Ikemoto, S. Rusinkiewicz, and M. Levoy, "Geometrically stable sampling for the ICP algorithm," *3-D Digital Imaging and Modeling, Proc. of the International Conf. on*, pp. 260–267, 2003.

[17] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments," *Robotics and Automation. Proc. of IEEE International Conf. on*, 2011.