GPU-based Point Cloud Recognition using Evolutionary Algorithms

Roberto Ugolotti, Giorgio Micconi, Jacopo Aleotti, and Stefano Cagnoni

Department of Information Engineering, University of Parma, Italy
{rob_ugo,micconi,aleotti,cagnoni}@ce.unipr.it

Abstract. In this paper, we describe a method for recognizing objects in the form of point clouds acquired with a laser scanner. This method is fully implemented on GPU and uses bio-inspired metaheuristics, namely PSO or DE, to evolve the rigid transformation that best aligns some references extracted from a dataset to the target point cloud. We compare the performance of our method with an established method based on Fast Point Feature Histograms (FPFH). The results prove that FPFH is more reliable under simple and controlled situations, but PSO and DE are more robust with respect to common problems as noise or occlusions.

Keywords: Particle Swarm Optimization, Differential Evolution, Pattern Recognition, GPGPU, Point Clouds

1 Introduction

The recent spread of 3D sensors has strongly increased the number of systems that operate on 3D data to perform operations like motion planning, human-robot interaction, manipulation and grasping.

In this paper, we consider a system which is part of an architecture whose goal is to help users program robotic tasks. To reach this goal, a sub-system for object recognition is required (see Figure 1). It receives input data from a high-resolution planar laser scanner mounted on the wrist of a six degrees of freedom robot arm. The estimated accuracy of the whole measurement chain is about 1.5 cm, the main sources of error being the variable remission of objects and the angle of incidence of the laser. Data undergo several preprocessing steps to refine the acquisition and are then passed to the FPFH (Fast Point Feature Histograms) based recognizer, along with a list of models stored in a database. The output of the recognizer indicates which objects are present in the scene and in which pose. A thorough description of a preliminary version of this system can be found in [1].

In [2] we have shown that bio-inspired metaheuristics like Particle Swarm Optimization (PSO) [3] or Differential Evolution (DE) [4] can successfully perform object recognition and registration.

The main goal of this paper is to present an implementation of the method proposed in [2] to solve the problem of 3D point-cloud registration and recognition. We will assess its performance in several situations and compare our results

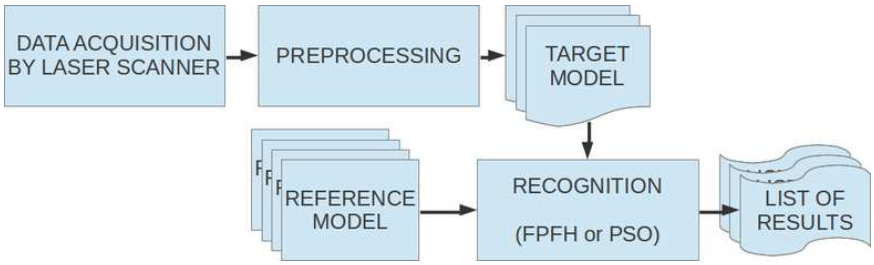

Fig. 1: Representation of the system within which the FPFH recognizer, or the one based on PSO as an alternative, is used.

to those obtained using FPFH features. Figure 1 shows that this recognizer can be easily embedded into the existing system.

2 Theoretical Background

In this section, we briefly describe the background of our approach to object recognition without re-introducing PSO and DE, whose descriptions can be found in [5] and [6], respectively.

Point Clouds

A point cloud is a set of three-dimensional points expressed within a certain coordinate system. A point cloud can have several meanings but is usually interpreted as a discrete representation of the external surface of an object. It can be generated artificially, using CAD or 3D editing tools, or by several types of sensors, like, for instance, RGB cameras, depth cameras or laser scanners. Point clouds are often used in object recognition and in many other problems related to the understanding of the environment.

Point-cloud registration is a well-known problem for which many solutions have been proposed. Approaches that inspired our work can be found in [7], where Li et al propose a function based on a Gaussian Mixture distance map and use PSO to optimize it; in [8], where registration of partially overlapped point clouds is achieved by estimating their Extended Gaussian Images; and in [9] where DE is used to register a triangular mesh to a point cloud by minimizing their relative distance.

Model Based Object Recognition

The approach used in this work is an application of the method presented more in depth in [2]. The general process is quite straightforward:

1. A template of the object to recognize is created off-line, defining the available range of deformations to which it can be subject;

2. This model is rotated and deformed during the evolutionary process in order to match, as much as possible, a target under consideration (we used PSO and DE, but other metaheuristics can be used);
3. The process stops when a convergence criterion (e.g. alignment reached, time) is met.

The main goal of this work is to recognize the pose of a known object, so the first step just consists of reading a point cloud from a database of available models. Moreover, the model can only be subject to a rigid transformation, so the search space is defined only by six degrees of freedom (translations and rotations around the three axes). This means that the dimensionality of the search space in which DE and PSO operate is six.

CUDA

Graphic Processing Units (GPUs) contain up to several thousands of cores that can execute the same code at the same time on different data. While originally used only in gaming and computer graphics, their use has recently spread to a very large number of applications [10] following the GPGPU (general-purpose computing on GPU) paradigm, within environments like CUDA or OpenCL.

CUDA (Compute Unified Distributed Architecture) [11] is a general purpose parallel computing environment distributed by nVIDIA^{TM} which exploits the massively parallel computation capabilities of its GPUs. CUDA C/C++ is an extension of the C language that allows development of GPUs routines (named *kernels*), that run in parallel as a number of different CUDA threads, following the Single Instruction Multiple Thread (SIMT) model. Each kernel is executed on different threads, which run all the same code, but on different data. These threads may be grouped into *blocks*. A block can be seen as a group of threads that share the same information and can exploit fast, local memory instead of using the slow, global one.

Algorithms with high arithmetic intensity, low memory requirements and few interactions between independent threads, like evolutionary algorithms (EAs), are very well suited for GPGPU. Therefore, in the last years, many GPU-based implementations of EAs have been presented. The first implementations of PSO and DE based on CUDA were developed in 2009 and 2010, respectively [12, 13]; after that, several other implementations have been proposed. Two comprehensive reviews regarding GPU implementations of PSO [14] and DE [15] have been recently presented by Kromer et al.

3 FPFH

Fast Point Feature Histograms [16] (an evolution of PFH [17]) are pose-invariant local features which represent the underlying surface model properties for all the elements composing a point cloud. These features form a full description of a point cloud, therefore they can be used for several tasks, like aligning a

target to a reference (registration). These descriptors are computed for each point of a given point cloud and are generated by comparing the normal of a specific point with the normals of the points within a certain radius, which is a fundamental parameter of the algorithm. For a more detailed description, please refer to [17]. Once all descriptors of the two point clouds (target and reference) have been computed, a particular version of the RANSAC algorithm (RANdom SAmple Consensus) [18] is used to find a raw alignment between the clouds. This version is called SAC-IA (SAmple Consensus - Initial Alignment) and is followed by a second step, which attempts to refine the previous alignment, using the Iterative Closest Point algorithm. Eventually, the two transformations found by the algorithms are composed in order to compute the full transformation needed to align the two clouds.

4 Evolutionary Implementation

In this section we describe the fitness function used by PSO and DE, as well as the system's GPU-based implementation. From now on, we will refer only to PSO, but DE could also play exactly the same role.

4.1 Fitness Function

The fitness function used by PSO is relatively straightforward. We compare the target cloud T to be recognized (composed of N_T points), with a reference cloud R extracted from a database, composed of N_R points. This reference is subject to a transformation M encoded by a PSO particle, to obtain $R' = M(R)$. The fitness of a particle is the average of the minimum distances of each point of T to the closest point of the roto-translated reference R'. More formally:

$$F(T, R') = \frac{1}{N_T} \sum_{p \in T} \min_{q \in R'} \left(dist(p, q) \right)$$

where $dist()$ is a valid distance metric between points; in this case we selected the squared euclidean distance.

Each point cloud is expressed within a local reference frame centered around its centroid. A model can do a full rotation around each axis while the range of translation is limited to 10 cm in each direction, which is good enough to satisfy the requirements of the environment we are considering.

4.2 GPU Implementation

The entire system, including the computation of the fitness function, has been implemented on GPU. Several implementation designs have been tested. In the final one, two degrees of parallelism are exploited:

1. The i-th PSO particle represents a possible transformation M_i of the reference R and relies on a CUDA block, so all M_is can be computed in parallel;

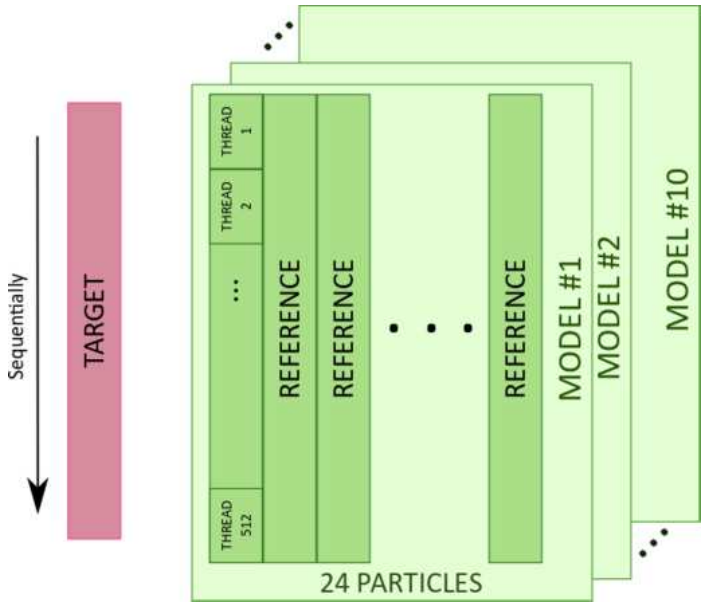

Fig. 2: Scheme of the implementation of the fitness function on CUDA. Target points are computed sequentially. The parallel implementation relies on the fact that each point is compared (potentially) in parallel to all reference models (10 in this case), where each of the 24 PSO particles represents a possible transformation; 512 points are processed simultaneously for each particle.

2. Within each particle (so, within each block), each of many parallel threads processes a limited number of points of R, by firstly computing a portion of the transformed point cloud R' and then comparing it with all points of T.

The points of T are actually processed sequentially, but a significant speedup can be obtained anyway because each of them is compared at the same time to several points of the reference cloud, and to different transformations of R. A further level of parallelization has been tested where each particle is represented by more blocks, and each block considers a sub-portion of T, since different parts of the target can be computed independently. This choice was discarded because it does not bring any speedup of the fitness function computation. This is probably caused by the large amount of resources (especially number of threads) needed for its computation, which prevents a full parallelization and forces the GPU to schedule some CUDA blocks sequentially.

If the target is compared with more than one reference (for instance, to recognize which object has been scanned), a further level of parallelism can be added: several optimization processes can be executed in parallel using different reference models. For the same reason explained in the previous paragraph, the parallelism is not perfect and the difference with respect to a version in which

all references are analyzed sequentially is not very significant. Figure 2 outlines how the work is subdivided among CUDA blocks and threads.

The GPU-based implementation of the metaheuristics employed in this paper has been presented in [19][1]. The parallel PSO implementation is structured as three distinct kernels: (i) the first one generates the solutions that is going to be evaluated, (ii) the second one computes the fitness function described before, and (iii) the last one updates the population.

5 Results

We performed the experimented tests on a PC equipped with a 64-bit Intel Core i7 CPU running at 3.40 GHz using CUDA v. 5.0 on an nVidia GeForce GTX680 graphics card with 1536 cores working at 1.20 GHz and compute capability 3.0.

The PSO and DE parameters (unless specified otherwise) were set as in Table 1. They have been chosen by manually generating 40 possible combinations, and testing them on the problem described in the next subsection. The configuration that gave the best average fitness was finally selected. We compared DE and two PSO versions (with global and ring topologies).

Table 1: Parameters used by DE and PSO. Refer to [5] and [6] for the meaning of the parameters.

DE	PSO_r	PSO_g
$Cr = 0.9$	$\phi_1 = 1.19$	$\phi_1 = 1.8$
$F = 0.5$	$\phi_2 = 1.19$	$\phi_2 = 0.7$
Exponential Crossover	$\omega = 0.5$	$\omega = 0.72$
Target-to-best Mutation	Ring Topology ($K = 1$)	Global Topology
Population Size = 24	Population Size = 24	Population Size = 24
Generations = 90	Generations = 90	Generations = 90

5.1 Error vs Fitness

We performed several experiments under different conditions. Firstly, we wanted to prove that our fitness function is correct, i.e., a good fitness value actually corresponds to a good match between the reference and the target. In these tests (and in all the following, except the ones presented in Section 5.4), we used the same model (a wooden mallet) as target and as reference, with random roto-translations applied to the target. So, it was actually possible to achieve a perfect matching if the recognition process identified the correct transformation. We define the translation applied to the target as t_T and the rotation as r_T to show that, the closer to (t_T, r_T) the transformation applied to the reference, the

[1] The code is available online at http://sourceforge.net/projects/libcudaoptimize/

better its fitness, i.e. there exists a direct correlation between error and fitness values.

Figure 3 shows the relationship between errors in the transformation and fitness values. Each point represents an independent repetition of the recognition task. Its position on the graph represents the error in terms of translation (euclidean distance between translation obtained at the end of the experiment and t_T) and rotation (angle between rotation computed and r_T). The color is related to the fitness value: dark colors stand for good (low, since this is a minimization problem) fitness values and light colors represent bad values. As can be seen, the closer a point is to the optimum $(0, 0)$, the darker it is. Computing the correlation coefficient between these two distances and the fitness, the results are 0.825 (translation error) and 0.726 (rotation error) showing a significant direct relationship.

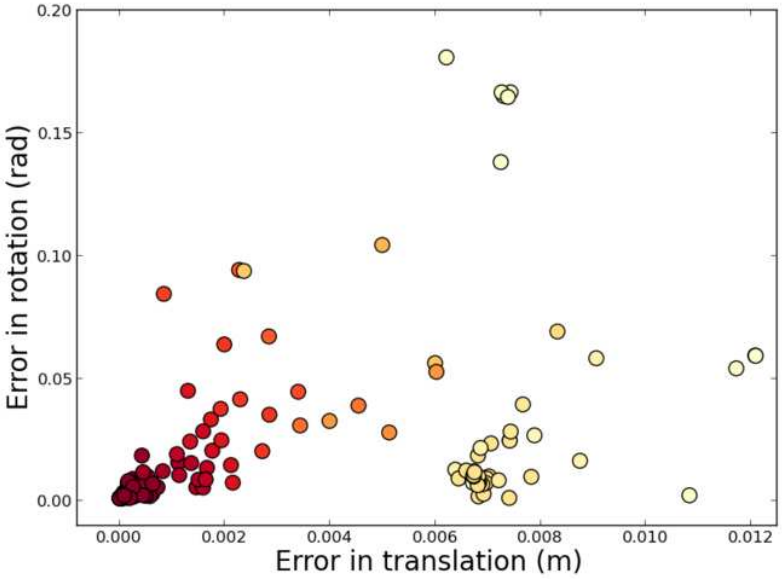

Fig. 3: Relation between error in translation (shown on the x axis of the graph) and rotation (on the y axis) and fitness value (color).

5.2 Time Comparison

We tested different PSO, DE and FPFH parameters (varying the number of generations in the first two, of RANSAC and ICP iterations for the other) in order to see how they behave within different time constraints. We set four different time limits: 0.7 s, 1.3 s, 2.3 s and 3.2 s. Figure 4 shows that FPFH reaches

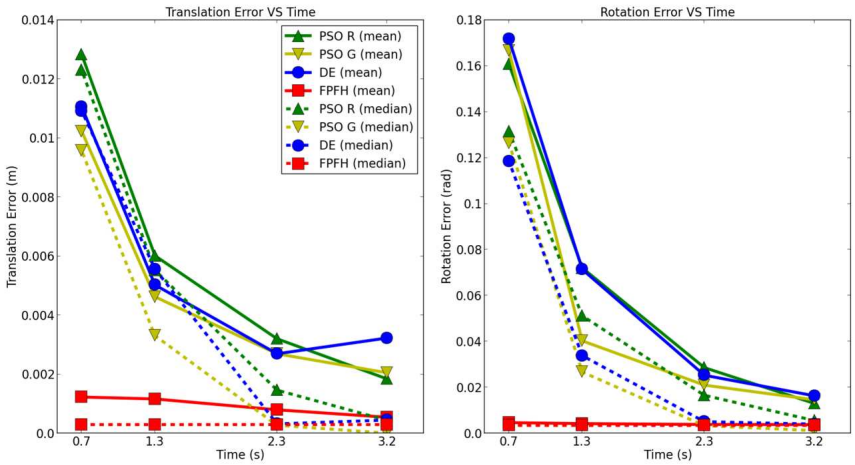

Fig. 4: Error versus processing time allowed for optimization, computed over 100 experiments. Solid lines represent average values, while dotted lines represent medians.

good results very quickly, but cannot improve them any further, while metaheuristics use their exploitation abilities to constantly refine their results. This is confirmed by statistical tests (Friedman test with the Dunn-Sidak correction, $p = 0.01$) which show that within the first two time limits FPFH is statistically better than the other methods considering translation and rotation errors.

Moreover, PSO/DE have usually a lower median and higher average when compared to FPFH. This result (that will be confirmed in all other tests) proves that evolutionary methods have a better ability of finding more precise solutions, but sometimes they fall in local minima and fail completely. On the contrary, FPFH steadily obtains good results, though worse than the ones obtained in the successful runs of the metaheuristics.

The sequential single-thread CPU implementation of the PSO recognizer takes an average of 60.5 s for 90 generations, which means it is 18.9 times slower than the GPU version. If we parallelize the evolutionary process over the 8 cores available on the CPU, the time needed is reduced to 16.4 s, so the GPU is still 5.1 times faster.

5.3 Noise and Occlusions

In this section, we simulated some situations that can hamper object recognition, like noise and occlusions. We simulated the former by adding to each point of T a random value from a uniform distribution (we chose ranges of 0.001, 0.002, 0.005, 0.01 m), and the latter by removing all points above a certain percentile along a given dimension (we "occluded" 20%, 40%, 60% and 80% of the target). Figure 5 shows that FPFH is less robust to this kind of difficulties than PSO.

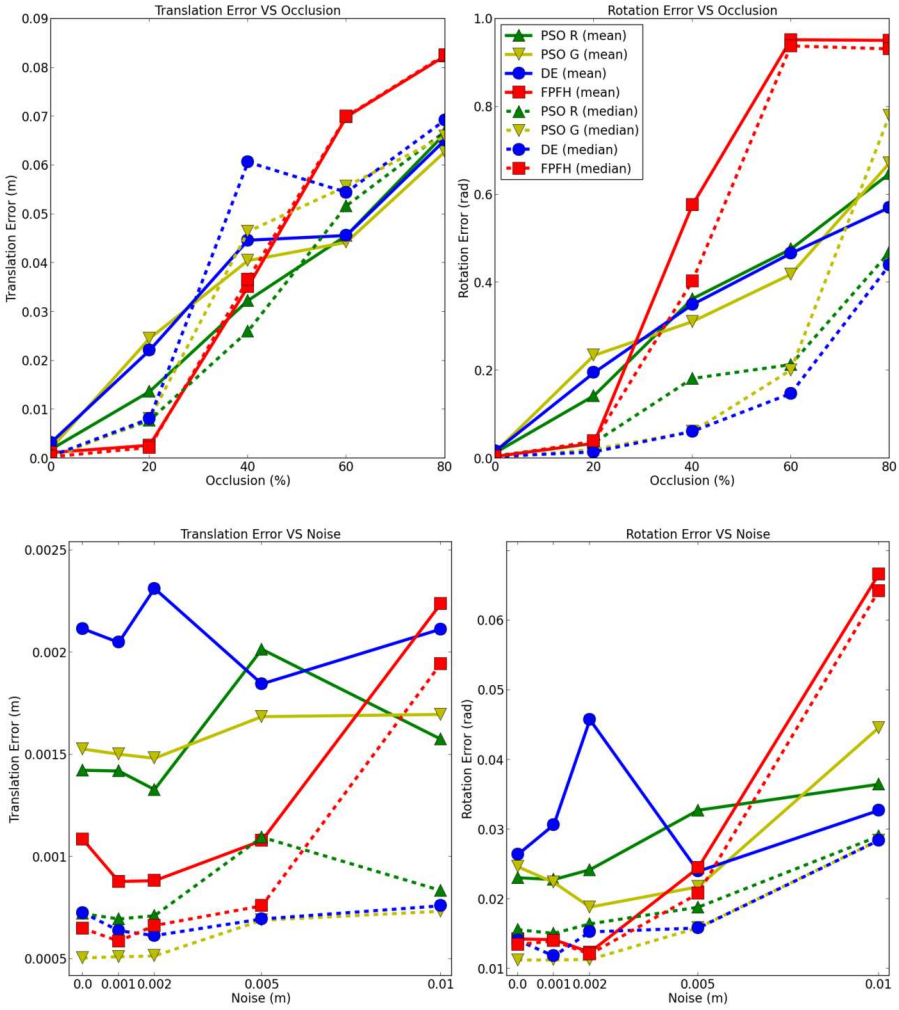

Fig. 5: Variation of errors in the presence of occlusions (top) and noise (bottom) added to the target over 100 experiments. Solid lines represents average values, while dotted lines represents medians.

Starting from an occlusion level of 60%, and for a noise range of 0.01 m, FPFH is significantly worse than all the EA-based methods in translation and rotation errors.

Figure 6 shows, as in Figure 3, how PSO and FPFH react to noise. Each color represents a different value of noise added to the target. It can be seen on the left that there is no clear difference between the different levels of noise when using PSO, while, on the right, the points corresponding to different noise levels can be easily clustered as different clouds. In particular, when the noise is low, there is almost no difference among the solutions found but, when noise increases, the dots are scattered over a large area.

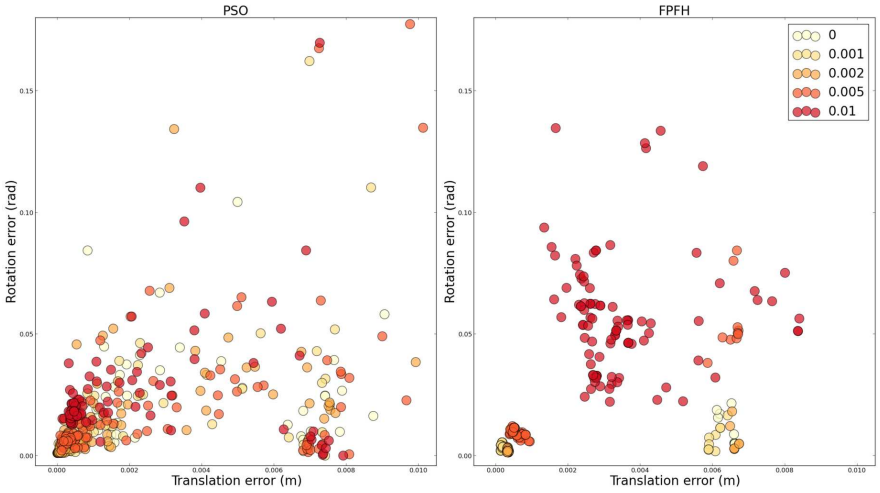

Fig. 6: Errors versus noise level (color) using PSO (left) and FPFH (right) over 100 experiments for each level. The results of FPFH are more consistent, while the ones obtained by PSO are more scattered.

5.4 Object Recognition

After assessing the behavior of these two methods under different conditions, we performed some tests on a different problem: object recognition. In this case, the goal was not only to understand where the object was located, but also to recognize the target object, within a set of ten reference objects: the wooden mallet previously used, a ewer, a burner, a toy horse, a mug and five different boxes of different shapes and sizes. We performed 50 independent tests in which each object was used as target and compared to all the others both under normal conditions and simulating the presence of noise and occlusions. Results are presented in Figure 7.

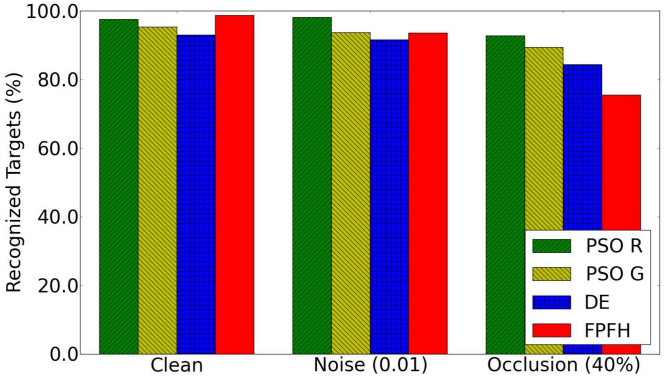

Fig. 7: Percentage of correct recognitions over 500 experiments (50 repetitions for 10 different objects) for every entry of the bar chart. Again, one can see how FPFH performance degrades in the presence of occlusions.

6 Conclusions

We applied a method based on Particle Swarm Optimization or Differential Evolution to recognize objects acquired with a laser scanner in the form of a point cloud. Each PSO or DE particle encodes a possible roto-translation of a point cloud used as reference; its optimization process tries to minimize the squared euclidean distance between the points of the target and the reference. We compared our method with a well-known method used for this task, FPFH. The main conclusions can be summarized as follows:

- FPFH reaches good results in a very short time, but it is not able to further improve them. Vice versa, the longer the time allowed to run EAs, the better the results they obtain;
- FPFH reaches good results almost always in ideal conditions, while EAs are able to achieve higher precision most of the times, but sometimes fail;
- EAs are more robust to noise and occlusions than FPFH.

As previously stated, PSO and DE parameters were selected among a few manually selected alternatives. It has been largely proved that, in many tasks, a good parameter setting can improve the performance of metaheuristics significantly. As future work, we will try to see if better performance can be achieved by automatically selecting such parameters.

The fitness function currently implemented for PSO can work properly only when the point cloud represents a single target. In some situations, it may be useful to recognize and localize more than one object in the scene at the same time. This can be obtained by moving the focus on the reference instead of focusing on the target: in other words, instead of finding the pose that minimizes the average distance of the points of the target cloud, one should find the transformation that minimizes the same metric regarding the points in the reference cloud. This will be the next step of our work.

References

1. Oleari, F., Lodi Rizzini, D., Caselli, S.: A low-cost stereo system for 3D object recognition. In: IEEE International Conference on Intelligent Computer Communication and Processing (ICCP). (2013) 127–132
2. Ugolotti, R., Nashed, Y.S., Mesejo, P., Ivekovič, Š., Mussi, L., Cagnoni, S.: Particle Swarm Optimization and Differential Evolution for model-based object detection. Applied Soft Computing **13**(6) (2013) 3092–3105
3. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proc. IEEE International Conference on Neural Networks. Volume 4. (1995) 1942–1948
4. Storn, R., Price, K.: Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute (1995)
5. Das, S., Suganthan, P.: Differential Evolution: A Survey of the State-of-the-Art. IEEE Transactions on Evolutionary Computation **15**(1) (2011) 4–31
6. Poli, R., Kennedy, J., Blackwell, T.: Particle Swarm Optimization. Swarm Intelligence **1**(1) (2007) 33–57
7. Li, H., Shen, T., Huang, X.: Approximately global optimization for robust alignment of generalized shapes. IEEE Transactions on Pattern Analysis and Machine Intelligence **33**(6) (2011) 1116–1131
8. Makadia, A., Patterson, A., Daniilidis, K.: Fully automatic registration of 3D point clouds. In: Conf. on Computer Vision and Pattern Recognition. (2006) 1297–1304
9. Urfalolu, O., Mikulastik, P., Stegmann, I.: Scale Invariant Robust Registration of 3D-Point Data and a Triangle Mesh by Global Optimization. In: Advanced Concepts for Intelligent Vision Systems. Volume 4179 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2006) 1059–1070
10. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A Survey of General-Purpose Computation on Graphics Hardware. Computer Graphics Forum **26** (2007) 80–113
11. nVIDIA Corporation: nVIDIA CUDA Programming Guide v. 5.0. (2012)
12. de Veronese, L., Krohling, R.: Swarm's flight: Accelerating the particles using C-CUDA. In: Proc. IEEE Congress on Evolutionary Computation. (2009) 3264–3270
13. de Veronese, L., Krohling, R.: Differential evolution algorithm on the GPU with C-CUDA. In: Proc. IEEE Congress on Evolutionary Computation. (2010) 1–7
14. Kromer, P., Platos, J., Snasel, V.: A brief survey of advances in Particle Swarm Optimization on Graphic Processing Units. In: IEEE World Congress on Nature and Biologically Inspired Computing (NaBIC). (2013) 182–188
15. Kromer, P., Platos, J., Snasel, V.: A brief survey of differential evolution on Graphic Processing Units. In: Symposium on Differential Evolution (SDE). (2013) 157–164
16. Rusu, R.B., Blodow, N., Beetz, M.: Fast point feature histograms (FPFH) for 3D registration. In: IEEE International Conference on Robotics and Automation (ICRA). (2009) 3212–3217
17. Rusu, R.B., Marton, Z.C., Blodow, N., Beetz, M.: Learning informative point classes for the acquisition of object model maps. In: IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV). (2008) 643–650
18. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM **24**(6) (1981) 381–395
19. Nashed, Y.S.G., Ugolotti, R., Mesejo, P., Cagnoni, S.: libCudaOptimize: an open source library of GPU-based metaheuristics. In: Proc. of the Genetic and Evolutionary Computation Conference (GECCO) companion, ACM (2012) 117–124