Novelty Search for software improvement of a SLAM system

Víctor R. López-López
Leonardo Trujillo
Instituto Tecnológico Tijuana
Tijuana, Baja California, México
vlopez@tectijuana.edu.mx
leonardo.trujillo@tectijuana.edu.mx

Pierrick Legrand
University of Bordeaux
Institute of Mathematics, Bordeaux, France
IMB UMR CNRS 5251
Inria Bordeaux Sud-Ouest, France
pierrick.legrand@u-bordeaux.fr

ABSTRACT

Genetic Improvement (GI) performs a search at the level of source code to find the best variant of a baseline system that improves non-functional properties while maintaining functionality, with noticeable results in several domains. There a many aspects of this general approach that are currently being explored. In particular, this work deals to the way in which the search is guided to efficiently explore the search space of possible software versions in which GI operates. The proposal is to integrate Novelty Search (NS) within the GISMOE GI framework to improve KinectFusion, which is a vision-based Simultaneous Localization and Mapping (SLAM) system that is used for augmented reality, autonomous vehicle navigation, and many other real-world applications. This is one of a small set of works that have successfully combined NS with a GP system, and the first time that it has been used for software improvement. To achieve this, we propose a new behaviour descriptor for SLAM algorithms, based on state-of-the-art benchmarking and present results that show that NS can produce significant improvement gains in a GI setting, when considering execution time and trajectory estimation as the main performance criteria.

CCS CONCEPTS

• **Software and its engineering** → **Automatic programming**; **Search-based software engineering**; • **Computing methodologies** → *Vision for robotics*; *Genetic programming*;

KEYWORDS

Genetic Improvement, SLAM, Novelty Search

ACM Reference Format:
Víctor R. López-López, Leonardo Trujillo, and Pierrick Legrand. 2018. Novelty Search for software improvement of a SLAM system. In *GECCO '18 Companion: Genetic and Evolutionary Computation Conference Companion, July 15–19, 2018, Kyoto, Japan.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3205651.3208237

GECCO '18 Companion, July 15–19, 2018, Kyoto, Japan

ACM ISBN 978-1-4503-5764-7/18/07…$15.00
https://doi.org/10.1145/3205651.3208237

1 INTRODUCTION

Machine learning (ML) coupled with affordable and powerful computing hardware has brought about a new approach for the development of software systems. Every year new applications are able to automate tasks that formerly required explicit human design. It is now possible to partially automate the development of pattern recognition, image understanding and natural language processing systems. However, until recently, human programming was still an essential part of both system development and maintenance. This has changed, through search-based software engineering, and the genetic programming (GP)-based genetic improvement (GI), at least some of those tasks can be outsourced to an automatic process [27]. In GI, the goal is to use meta-heuristic methods to automatically modify a software system [27]. GI focuses on automatic bug detection and repair [8, 15, 17], and the automatic improvement of non-functional properties [10]. What makes GI different from GP is that it operates directly at the level of source code, exploiting the manner in which source code is developed [6, 10].

We argue that while GI has been shown to be useful at the software systems level [8, 15, 17], probably the most interesting application area is in the automatic improvement of complex programs [10, 11]. In these cases expert human knowledge is scarce, even at the abstract level, while deep source code understanding is usually limited to a small group of developers and close colleagues. Producing a steep learning curve for any developer interested in extending or improving the code.

In this work, we apply GI to vision-based Simultaneous Localization and Mapping (SLAM), one of the most widely studied problems in both computer vision and robotics. The vision-based SLAM problem can be stated as follows. Assume that a robot is provided with a camera as its only sensor, with no additional information regarding the surroundings. The goal for the robot is to automatically construct a map of the environment (usually 3D for vision-based SLAM) and simultaneously localize itself within the map; i.e., determine its pose within the 3D map at every time instance, over time describing a trajectory. When dealing with vision-based SLAM one can distinguish between sparse [4] and dense methods that attempt to fully reconstruct the 3D scene[26]. SLAM is particularly interesting given its wide ranging potential in high-end applications, that include mobile robotics, augmented reality and autonomous vehicle navigation. In fact, the sheer amount of research in SLAM has solved this problem (under certain constraints), at least at theoretical and abstract level. This has lead to many successful SLAM systems, such as the popular KinectFusion [26]. A nice survey is given by Cadena et al. [3] on the past, present and future of SLAM, providing the main theoretical foundations of SLAM, and identifying some of the

main open issues and future perspectives. Concerning the latter, the authors identify the efficiency of modern SLAM systems when executed in *resource-constrained platforms* where execution time or efficiency impose strong constraints on the system.

Indeed, one important contribution has been SLAMBench [20], a comprehensive tool to evaluate dense RGB-D SLAM systems, providing a relatively simple and direct way to evaluate the functional and non-functional properties of a SLAM system. The most important functional property is the absolute trajectory error (of the camera/robot within the environment), while non-functional properties can include execution time, memory consumption or energy consumption. SLAMBench provides tools to perform evaluations on multiple computing platforms and development environments, including C++, OpenMP, OpenCL and CUDA. In other words, SLAMBench provides an open source platform to evaluate the efficiency and accuracy of a vision-based SLAM system.

This is, in our opinion, the perfect research context for the application of GI. Unlike other recent contributions for the automatic tuning of a SLAM system [2, 19], that focus on parameter optimization, GI provides the necessary tools to explore the true underlying design space of the popular KinectFusion algorithm, considering all of the system Kernels as candidates for improvement [26].

Moreover, this work makes a second contribution to GI. Two issues for GI systems are the flatness of the search space, which can be highly neutral, and the loss of diversity in the population. GI systems generate many individuals (list of source code modifications) that produce code that does not compile, and have difficulty finding partial solutions that improve some parts of the code but may degrade performance elsewhere. This work addresses these issues by using Novelty Search (NS) [16], where fitness is not directly computed based on the performance criteria. Instead, NS searches for high-performing individuals indirectly by applying selective pressure to produce novel solutions; i.e., solutions that exhibit unique behaviors relative to others that were previously found. While recent contributions have applied NS to GP systems [22–24, 30], this paper is the first to apply it in a GI setting. To do so, a new behavior descriptor is proposed, that captures the manner in which the SLAM system performs on the set of test cases, a prerequisite to apply NS to any problem. Results show that the NS-GI search is able to improve upon standard GI, and produced significantly improved variants of the KinectFusion SLAM system using a community standard test set [7].

The remainder of this paper is organized as follows. Section 2 reviews related works from computer vision literature. Section 3 provides a brief introduction to GI and to the NS algorithm. The proposed approach is then presented in Section 4. Section 5 presents the experimental work and discusses the main results. Finally, Section 6 presents the conclusions and outlines future work.

2 RELATED WORK

SLAM is a well understood problem, with many successful approaches published over the years. This has lead several research groups to focus on the efficiency of current state-of-the-art systems. That is the reason SLAMBench was developed [21], providing a broad set of tools to evaluate and help detect possible areas of improvement within a SLAM system. Recently, the next logical step was also taken, to use SLAMBench to perform an exploration of the configuration space of a SLAM system.

The first such work is [31], a detailed comparative analysis of two SLAM systems, KinectFusion and LSD-SLAM [5]. The work considers accuracy, energy consumption and frame rate as their comparative measures. It is the first systematic exploration of the the underlying parameter space of each algorithm. The effect that some parameters have on the functional and non-functional properties is studied, such as the minimum pixel gradient in LSD-SLAM, and the voxel grid resolution in KinectFusion. However, the results in [31] provide a partial characterization of parameter space, since each was studied independently. In [31], as well as the papers discussed next, the sequences from the ICL-NUM dataset [7] are used as reference, since the complete ground truth is available and compatible with SLAMBench. The dataset includes the known trajectory of the camera and the full 3D reconstruction of the scene.

Another possibility is to use SLAMBench to guide a learning process to automatically determine the optimal parameter configuration of a SLAM system. This task can be posed as a supervised learning problem, where a benchmark set of test cases (video sequences, such as those in ICL-NUM) can be used to compare the improvement (or performance degradation) caused by a particular parametrization, relative to user-defined default values. That is the goal of HyperMapper [2], a tool to automatically explore the efficiency-accuracy trade-off of a SLAM system. HyperMapper considers execution time (EXT) and power consumption as two non-functional properties that measure efficiency, and the absolute trajectory error (ATE) as the main functional property that measures accuracy. One difficulty in taking this approach is the cost of the objective function that requires the execution of SLAMBench and the SLAM system of interest on the video benchmarks, since to evaluate a particular point in parameter space it is necessary to actually measure its performance. HyperMapper uses ML techniques to train a surrogate objective function online, such that some points in parameter space are not evaluated, and maintains a Pareto set of solutions, each providing a different trade-off between efficiency and accuracy. Parameter space is sampled based on the current solutions on the Pareto set at each iteration. HyperMapper considers a large set of parameters for the KinectFusion algorithm, and also considers compiler flags and architecture-level parameters, comparing the results with the default settings. It is shown that the default configuration is dominated by many solutions on the Pareto Front [2]. In some cases, efficiency can increase from 6 fps to nearly 40 fps, and from 2.77 Watts to 0.65 Watts, while accuracy can increase from 4.41 cm to 3.30 cm of ATE. A big shortcoming of that work, however, is that the algorithm is trained using only half of a single video sequence (400 frames) from the ICL-NUM dataset (living room trajectory 2) without any tests on other video sequences, so generality cannot be confirmed. The search is also extremely costly, with the authors reporting that a total of 3,000 random samples are generated and evaluated with SLAMBench at the start of HyperMapper, with the active learning process generating 1,300 additional samples. To evaluate all these parametrizations the process required about 6 days of computation. HyperMapper also relies on the assumption of locality in parameter space, which might not be the case; i.e., that similar parametrizations will also produce similar performance.

Similarly, [19] presents another application of HyperMapper. They consider two SLAM systems, KinectFusion and ElasticFusion, and ATE and EXT are used for evaluation. A GPU implementation of ElasticFusion is used, and, for instance, a parametrization is found that provides a 1.25-fold speedup and a 2-fold improvement in terms of accuracy, relative to the default parameters. The improvements are tested for generalization by running the same tests on a large set of mobile devices, showing that the same effects are obtained. However, as in [19], performance generalization is not considered across multiple image sequences.

The present work builds upon these contributions in several ways. First, while the authors in [2, 19] claim to be performing design space exploration, they are actually performing, in our opinion, parameter space exploration. On the other hand, by using GI, this work can explicitly modify the algorithmic design of the system, and perform a more complete design space exploration. Second, a much larger training set is considered, two full video sequences from the ICL-NUM dataset are used for training, and the evolved improvements are also applied on the other two video sequences to evaluate generalization. Finally, by integrating NS into GI the system performs a better exploration of the search space, which should be assumed to be highly neutral, with low locality and highly discontinuous. Experimental results will show the beneficial impact of integrating NS on this GI problem.

3 BACKGROUND

3.1 Genetic improvement

GI, unlike GP, starts from a working program that is much larger than those evolved with GP [27]. GI performs a local search of sorts, attempting to find a modified version of a piece of software that is already functioning. It can be seen as a subset of search-based software engineering [9], integrating techniques from test-based software development and GP. Much of the research in GI has focused on the automatic repair of software, bug elimination [15] and detection [17], and has been integrated into online systems that perform continuous repairs [8].

Another common application of GI is to improve non-functional properties of a software system while maintaining the same functional behavior, using a known set of test cases and the behavior of the original system as an oracle. The criteria for improvement can be divided into functional and non-functional properties. The functional properties refer to the logic of the program, while the non-functional properties refer to physical aspects, such as execution time or energy consumption. One noteworthy example of GI is the GISMOE framework by Langdon and Harman [10], which is the basis for the work presented in the present paper. GISMOE has achieved large performance gains for open-source software, speeding-up some very complex pieces of software, such examples have been published in recent years [10, 12–14, 28].

In GISMOE the original source code is represented by a BNF grammar, and a GP search is used to explore alternate versions of the original by generating individuals that represent a list of modifications. These can be the deletions of a line of code, insertion of a line of source code next to another, and replacement of a line of code with another. The reproduction methods of the GP search are mutation and crossover, where mutation consist on applying one of the modifications previously mentioned, and crossover consists on the concatenation of the list of modifications from two individuals to create a new one. However, there are still several issues with GISMOE in particular, and GI in general. One of the most important is the fact the the search can get stagnated and lose diversity quickly. The problem is that it is often difficult to construct complete solutions in a piecewise manner, since partial solutions often break the code or have a negative effect on overall performance. Some of this is due to the fact that the search space is often a large neutral network, where a sufficient gradient to drive the search is lacking. For this reason, the present work presents an alternative way to guide the search, a method called novelty search.

3.2 Novelty search

NS was proposed by Lehman and Stanley as a way to escape from local optima in deceptive problems [16]. It has been applied many times in robotics tasks, with results that show the ability of NS to push the search out of local attractors quite effectively. While any meta-heuristic search can be used, NS only modifies the manner in which fitness is assigned. In standard search, GP and GI follow this approach, fitness of a solution candidate is defined based on the problem objective, directly measuring the degree to which a solution solves the problem of interest. However, if the fitness landscape is deceptive, which is often the case in real-world tasks, then attempting to directly minimize the objective function will surely lead to suboptimal solutions. Conversely, in NS it is required to first define a behavioral descriptor for each solution, to efficiently describe what a solution candidate does when it attempts to solve the problem. To compute fitness, NS then measures the novelty of each solution; i.e. how unique a particular solution is relative to previous solutions produced by the search. In order for NS to be successful, it is assumed that at the beginning of a search most, if not all, of the solutions will be very bad at solving the problem. Hence, the initial behavioral descriptors will be describing low quality solutions. Therefore, when novelty is promoted, the search process will attempt to produce descriptors that are very different from the initial ones. In behavioral space, it should be the case that the behaviors that are the most different from the initial random (bad) behaviors correspond to the desired behaviors of an optimal solution. While changing the selection pressure seems minor, evidence is building that it can be very useful. In the case of GP, NS has been applied to classification [23, 24], symbolic regression [18], and to improve generalization [25].

3.3 KinectFusion

KinectFusion works with RGB-D cameras, such as Kinect, that integrate a depth sensor in addition to a standard camera KinectFusion registers and fuses the stream of measured depth frames as the scene is explored from different viewpoints into a clean 3D reconstruction [2, 26]. A version of the KinectFusion algorithm is provided with SLAMBench, organized into six high-level building blocks referred to as pipelines, that operate in the following order:

(1) **Acquisition**: This input step is included explicitly to account for I/O costs during evaluation of a SLAM algorithm.
(2) **Preprocessing**: Normalizes each depth frame and applies a bilateral filter to reduce noise and invalid depth values.

Test-cases	ICL-NUIM	Partition
1	Living_room_traj0_loop	Testing
2	Living_room_traj1_loop	Training
3	Living_room_traj2_loop	Testing
4	Living_room_traj3_loop	Training

Table 1: Test cases from the ICL-NUIM dataset.

(3) **Tracking**: Computes a point cloud (with normal vectors) for each pixel in the camera frame of reference, and estimates the new 3D pose of the moving camera by registering the point cloud with the current global map using the iterative closest point (ICP) algorithm [1].
(4) **Integration**: Once the new camera pose has been estimated, the corresponding depth map is fused into the current 3D reconstruction.
(5) **Raycasting**: A voxel grid is used as the data structure to represent the map, employing a truncated signed distance function (TSDF) to represent 3D surfaces. This step recovers the 3D surfaces that are present at the zero crossings of the TSDF.
(6) **Rendering**: It is used for visual reconstruction of the map and trajectory of the camera.

Each pipeline contains one or more computer vision kernels, 14 in total. [20] shows that Integration and Raycasting consume most of the processing time, in all implementations and on all different platforms (desktop, laptop and embedded system). The only exception was the CUDA implementation, where Rendering consumed more than Integration on some platforms. In the present paper, we use SLAMBench with the sequential implementation (C++) of KinectFusion, and register the metrics it provides to asses the fitness of the GI approach. The total number of lines of C++ source code for all the KinectFusion kernels is approximately 700. We would like to stress that this is not a particularly large piece of software, but it is rather complex, and tuning or improving the software is not a trivial endeavor [2, 19, 31].

4 PROPOSED APPROACH

In this work, we integrate NS into the GISMOE framework to improve the functional and non-functional properties of the KinectFusion SLAM system, using SLAMBench to compute the objective criteria for improvement. Before providing more low-level details regarding our implementation, it is first necessary to define how NS is applied. All other design issues in GISMOE are kept as defined in [10], except for the fitness function that will be replaced by the NS fitness. As stated above, to apply NS to a particular problem the first step is to define an appropriate behavioral descriptor, which in this case will be based on the performance obtained on each test case, loosely based on the descriptors proposed in [23, 24].

Test Cases. Each candidate solution generated by GISMOE is evaluated on a set of l fitness cases. In particular, for this problem we use four video sequences with known ground truth from the ICL-NUM dataset; see Table 1. Two of them are chosen for training, the other two are used to test the best evolved solution.

Performance Evaluation. First, using the original KinectFusion implementation, we compute the reference performance of the system in terms of the absolute trajectory error $ATE^R(c)$, the functional property we wish to improve, and the execution time of the $i-th$ pipeline (high-level block) $EXT_i^R(c)$ for each test case c, as the non-functional properties, with a total of 6 pipelines in KinectFusion; these are obtained by executing 100 times the original KinectFusion program and calculating the median.

ATE is given by the mean squared error between the real-valued trajectory vector of the ground truth movement of the camera within the scene, and the estimated trajectory computed by the SLAM system. EXT is based on the wall-clock time measure provided by SLAMBench.

Then, during evolution for each individual j we compute the same values, respectively given by $ATE(c,j)$ and $EXT_i(c,j)$. Notice that we are considering the performance of each pipeline individually instead of aggregating over all pipelines. This provides a finer characterization of the performance of each solution candidate in GISMOE. Making it possible to identify individuals that improve specific pipelines, and promoting the recombination of complimentary solutions for the overall improvement of KinectFusion.

Behavior descriptor. The proposed behavior descriptor measures the degree with which a particular individual improves upon the performance of the original system. Thus, we define the functional gain on each test case c as

$$ATE_{Gain}(c,j) = \frac{ATE_R(c) - ATE(c,j)}{ATE_R(c)} \cdot 100\,, \tag{1}$$

and the non-functional gain on each pipeline as

$$EXT_{i,Gain}(c,j) = \frac{EXT_i^R(c) - EXT_i(c,j)}{EXT_i^R(c)} \cdot 100\,, \tag{2}$$

with both values taking on positive values when there is an improvement and negative values when the performance is worse than the reference values.

With these references, the behavior descriptor of the j individual is defined as an integer vector β_j of size $n = (q+p)\times l$, where l is the number of fitness cases, p is the number of pipelines (six) and q is the number of functional measures (only one in this case, ATE). For example, if we assume that $l = 1$ (we use two in the experiments), then the first element $\beta_j(1)$ corresponds to the behavior of a solution with respect to ATE, and the remaining p elements represent the behavior of the non-functional properties, one for each pipeline (each EXT_i) (from $\beta_j(2)$ to $\beta_j(7)$). Each element in β_j is determined by dividing the real line $(-\infty,\infty)$ into N segments, and define two parameters called *lower* and *increment*. Lets continue with the example to compute the value of $\beta_j(1)$. If $ATE_{Gain}(1,j) < lower$ then $\beta_j(1) = 0$; if $lower < ATE_{Gain}(1,j) \leq lower + increment$ then $\beta_j(1) = 1$; if $lower + increment < ATE_{Gain}(1,j) \leq lower + 2\times increment$ then $\beta_j(1) = 2$; and so on, until $lower + increment \times N < ATE_{Gain}(1,j)$ then $\beta_j(1) = N$. After a series of tests, we set $N = 18$, $lower = -45\%$ and $increment = 10\%$. Thus, each element in β_j will take on values from 0 to $N = 18$.

Fitness Function. With NS, fitness is given using a density estimation measure. For instance, in the original NS algorithm [16], fitness is given by the average distance in behavioral space between

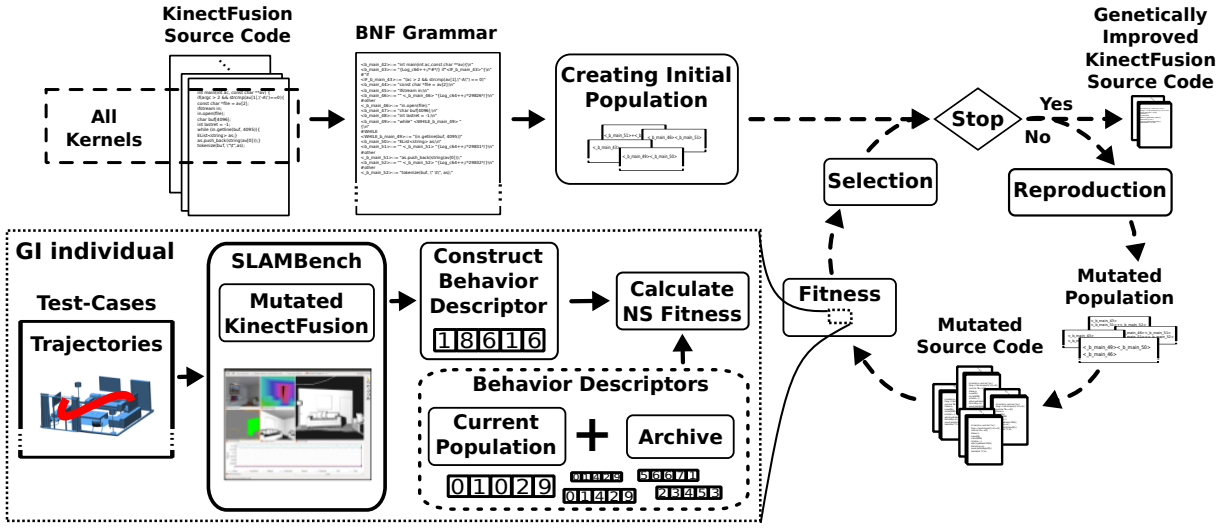

Figure 1: Illustration of the GISMOE process used to improve the KinectFusion SLAM system through SLAMBench with Novelty Search.

a descriptor of an individual solution and the descriptors of it's k nearest neighbors. The neighbors are obtained from the current population and from a population archive, where the descriptors of previous solutions can be stored. Another approach is to model novelty in a probabilistic manner, proposed in [24], which eliminates the need to manage an archive but works under the assumption that the dimensions of the behavior descriptor are independent, which might not be the case. In this work, fitness is given by the average Manhattan distance between an individual's behavior descriptor $\beta(j)$ and an archive of all previously generated descriptors $\alpha(o)$ that also includes the behaviors in the current population, expressed as

$$F_{NS}(j) = \frac{1}{h}\sum_{o=1}^{h} d(\beta(j), \alpha(o)) \tag{3}$$

where h is the size of the archive. Moreover, fitness is only computed for those individuals that satisfy the following: (1) the number of untracked frames must be lower than 40%; (2) succeed in compilation and execution of SLAMBench on the test cases. In the case that these constraints are not met, we assign a fitness value that does not permit this individual to be a parent, and the individual does not generate a behavior descriptor.

To summarize, Figure 1 shows the complete GI search with GISMOE. The dashed arrows indicate the basic GISMOE loop, as presented in Section 3, and the solid arrows inside the solid block (inner-left corner) indicate the evaluation process using SLAMBench. The GISMOE process starts by defining the BNF grammar from the KinectFusion source code. Then, the initial population is created, with 20 individuals selecting from the BNF grammar randomly. If compilation is successful, for each of the test cases SLAMBench measures $EXT_{i,gain}(c, j)$ and $ATE_{gain}(c, j)$, and the behavior descriptor is constructed. Next fitness is assigned based on behavioral novelty, as defined in Equation 3. Selection chooses the best 50% of individuals in the population, on which the search operators are applied to generate a new population that replaces the previous one. However, if some of the chosen individuals have been assigned the fail fitness, they are discarded and replaced by random ones. The evolutionary cycle continues until the stop condition is reached, which is a maximum of 200 generations.

5 EXPERIMENTS

The experiments were carried out on a virtual machine with 32 Cores and 32 GB RAM with Ubuntu 16.04, on top of a KVM host running Ubuntu 16.04 server on a DELL R430 with 2 Intel Xeon E5-2650 v4 2.2 GHz (12C/24T). Fitness evaluation was parallelized at the individual level with GNU Parallel [29], reducing the experiment time from a month of computation to two days. As already mentioned, we use the C++ implementation of KinectFusion and use 50% of the *test-cases* from the ICL-NUM benchmarks for training and testing (see Table 1). In particular, videos 2 and 4 are used to train the search, and videos 1 and 3 are used as unseen tests for the improved versions of Kinect Fusion. The SLAMBench parameters are set to their default values, as well most of GISMOE.

5.1 Results

The results of the NS-GISMOE search over the training set is presented graphically in Figure 3 and Figure 2. Figure 3 presents a convergence plot for the best values in the population for the average gain in total execution time over all fitness cases $\overline{EXT}_{Gain}$ $\overline{EXT}_{Gain}$ and the average gain in the absolute trajectory error $\overline{ATE}_{Gain}$. Figure 2 presents a Pareto front of all the solutions found, considering $\overline{EXT}_{Gain}$ and $\overline{ATE}_{Gain}$ of all the evolved solutions. The approximated Pareto front of the experiment shows that the performance of most individuals is in the range of −60% to 60% for $\overline{ATE}_{Gain}$ and −80% to 40% for $\overline{EXT}_{Gain}$. The plot also includes the reference individual (red dot) located at position $(0, 0)$ with no gain, and two individuals (green and purple) that were selected for further analysis to illustrate the type of solutions produced by the search. In this case we are not interested in the average performance

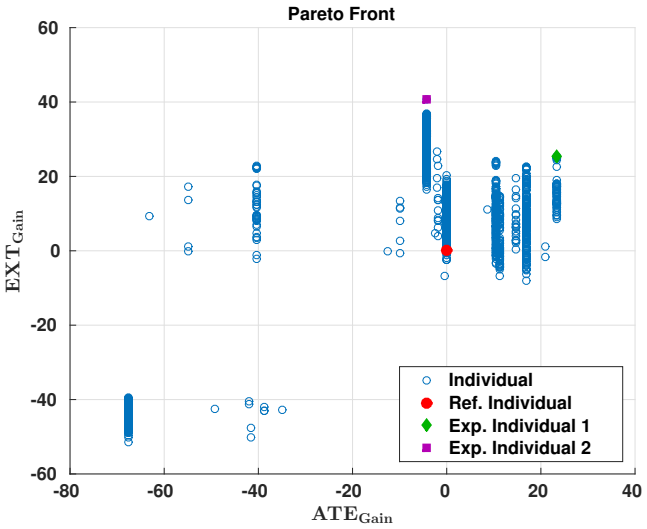

Figure 2: Approximated Pareto front found by NS-GISMOE.

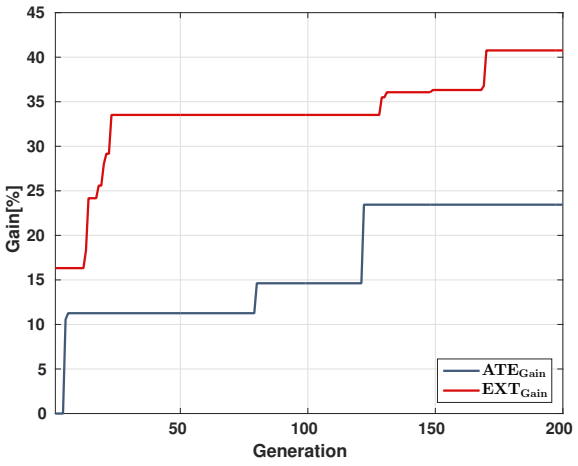

Figure 3: Convergence plot for the best $\overline{ATE}_{Gain}$ and $\overline{EXT}_{Gain}$ in the population at each generation.

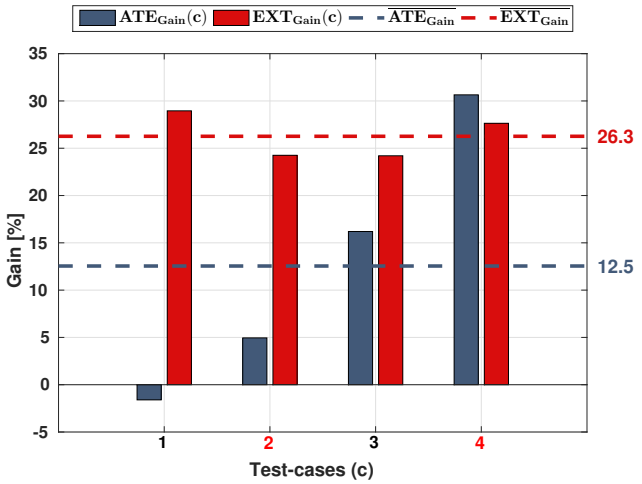

Figure 4: Analysis of the first individual, showing the gain achieved on each test case and the average gain.

of all individuals, but to chose individual with characteristics that were deemed important for further analysis. Each is described in the following subsections, with a justification of why they were chosen.

5.2 First Individual

The first individual, indicated by a green diamond marker in Figure 2, is chosen based on the fact that it improves both performance criteria, ATE and EXT, but has the best overall performance based on ATE. It includes a total of 48 edits across all of the KinectFUsion Kernels. The performance of this individual is shown in Figure 4, in the form of bar graphs that show the percentage of gain (or improvement) in the vertical axis for each test case (horizontal axis). Two bars are shown for each test case, that represent ATE_{Gain} (blue) and EXT_{Gain} (red) (higher values are better). Also, the mean gain, $\overline{EXT}_{Gain}$ and $\overline{ATE}_{Gain}$, over all test cases are shown as dashed lines (blue and red, respectively). The figure also identifies the test cases used for training using red bold numbers. $\overline{EXT}_{Gain} = 26\%$ and $\overline{ATE}_{Gain} = 12.5\%$, with performance being quite stable over all test cases. In particular, the gains in EXT do not vary much relative to the test cases, while the improvements in ATE do seem to depend on the test cases. In the first test video we have a slight performance loss based on ATE, but it is less than 2%. On the other hand, on the the second test video (test case 3), the gain in ATE is above average, which is a very positive result.

Regarding EXT, a more detailed analysis is provided in Figure 5, showing the average execution time required to compute a single video frame from each test case. Execution time is given for each pipeline in KinectFusion, with dashed lines representing the reference performance (unmodified code) and solid lines representing the performance of the chosen individual. Notice that the individual eliminates all computations due to Rendering, which is a simple modification to achieve a speedup since the Rendering pipeline is only required for visualization. Nonetheless, the reduction in EXT is not only due to this modification, because it only represents 9.7% of the average gain in total execution time, which is 26.3%.

What is promising in this solution is that both ATE and EXT are improved, with previous works assuming that this was not plausible [2, 19, 31]. Figure 7 presents a 3D rendering of a video frame from test sequence 1, comparing the default KinectFusion system with this genetically modified version. The figure clearly shows that both 3D reconstructions are quite similar, confirming that the functional properties are preserved in the modified version, and in fact improved when ATE is considered.

5.3 Second Individual

This individual, indicated by a purple square marker in Figure 2, includes a total of 53 edits across all of the KinectFusion Kernels. While the first individual exhibits a quite stable performance, across

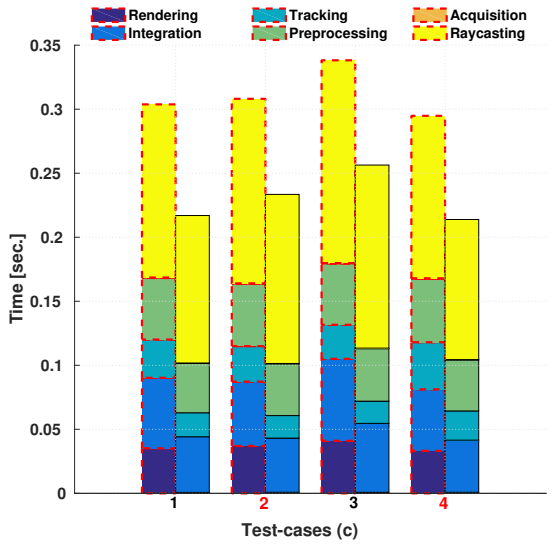

Figure 5: Analysis of the average execution time required to process a single video frame for the first chosen individual. Plots show the time required by each pipeline of KinectFusion for each test case of the ICL-NUM dataset. Results are shown for the reference system (dashed line), and the chosen individual (solid lines).

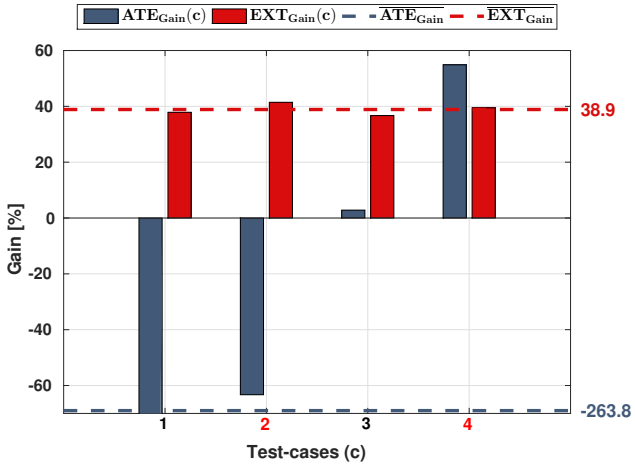

Figure 6: Analysis of the second individual, showing the gain achieved on each test case and the average gain.

both training and testing instances, this is not the case for this individual. Consider that the second and fourth test cases are used for training, while the first and third are used for testing. Notice that in terms of EXT, this individual clearly achieves a substantial improvement across all test cases, with an average of 38.9%. However, in terms of ATE the behavior is quite erratic. For the first two cases, performance is very poor, in particular the first test case pushes the average gain to -263% (the bar plot goes out of range from the plot limits). On the other hand, for the other two test cases performance is equivalent (test case 3) or substantially more accurate (test case 4, with an improvement close to 60%). Therefore, it seems that this is a much more specialized individual, with localization accuracy apparently depending on the characteristics of the scene or video sequence. If one inspects each test case manually, it is not at all evident why such large performance differences would arise.

6 CONCLUSION AND FUTURE WORK

This paper presents a GI approach to improve a complex computer vision system that solves the SLAM problem. In particular, GISMOE is applied to the KinectFusion SLAM system, to improve both functional (ATE) and non-functional (EXT) properties. To evaluate each solution candidate generated with GISMOE the SLAMBench tool is used, which provides a detailed comparison of the efficiency-accuracy trade-off of SLAM algorithms. Besides this novel application of GI to a known and prominent problem from the computer vision and robotics communities, the paper makes a second important contribution to GI and GP. This work presents the first integration of novelty search into a GI system. NS allows the GI process to explore behavior space more freely, promoting the generation of unique solution candidates, which is important given the highly neutral nature of a GI fitness landscape. To achieve this, a behavior descriptor is proposed, that considers the performance of each individual in a fine grained manner. For instance, EXT is considered independently for each pipeline in KinectFusion, to allow the search to detect individuals that provide only partial improvements to the system.

The GI search produced at least two notable solutions, described in detail in this paper. First, a solution that generalizes quite well over a benchmark set of test videos that is a standard in the SLAM community, with improvements on both ATE and EXT, of 26.3% and 12.5% respectively. Second, a more specialized solution where the improvements in EXT are of nearly 40%, while ATE behavior depends on the video sequence, with two cases showing substantial performance degradation, while the other two cases showing performance improvements of up to 60%.

Future work will try to better understand the modifications proposed to KinectFusion by GISMOE and NS. It will be important to reverse engineer what changes are most crucial to improve efficiency and accuracy in a SLAM system. Moreover, a formal comparison between standard objective-based GI and NS should be carried out, to determine what are the main differences and similarities between both search processes.

ACKNOWLEDGMENTS

This research was funded by CONACYT (Mexico) Fronteras de la Ciencia 2015-2 Project No. FC-2015-2:944, and first authors was supported by CONACYT graduate scholarship No. 302532.

REFERENCES

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

[2] B. Bodin, L. Nardi, M. Z. Zia, H. Wagstaff, G. S. Shenoy, M. Emani, J. Mawer, C. Kotselidis, A. Nisbet, M. Lujan, B. Franke, P. H. J. Kelly, and M. O'Boyle. 2016. Integrating algorithmic parameters into benchmarking and design space exploration in 3D scene understanding. In *2016 International Conference on Parallel Architecture and Compilation Techniques (PACT)*. 57–69.

Figure 7: Comparison of the 3D surface reconstruction from the first test video from ICL-NUM: (left) reference KinectFusion and (right) GI-improved version (first individual).

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. 2016. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *Trans. Rob.* 32, 6 (Dec. 2016), 24.

[4] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. 2007. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Trans. Pattern Anal. Mach. Intell.* 29, 6 (June 2007), 1052–1067.

[5] Jakob Engel, Thomas Schöps, and Daniel Cremers. 2014. LSD-SLAM: Large-Scale Direct Monocular SLAM. In *Computer Vision – ECCV 2014*, David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars (Eds.). Springer International Publishing, Cham, 834–849.

[6] Mark Gabel and Zhendong Su. 2010. A Study of the Uniqueness of Source Code. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '10)*. ACM, New York, NY, USA, 147–156.

[7] A. Handa, T. Whelan, J. McDonald, and A. J. Davison. 2014. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 1524–1531.

[8] Saemundur O. Haraldsson, John R. Woodward, Alexander E. I. Brownlee, and Kristin Siggeirsdottir. 2017. Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO '17)*. ACM, New York, NY, USA, 1513–1520.

[9] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. 2012. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Comput. Surv.* 45, 1, Article 11 (Dec. 2012), 61 pages.

[10] W.B. Langdon and M. Harman. 2015. Optimizing Existing Software With Genetic Programming. *Evolutionary Computation, IEEE Transactions on* 19, 1 (Feb 2015), 118–135.

[11] W. B. Langdon and S. M. Gustafson. 2010. Genetic Programming and Evolvable Machines: ten years of reviews. *Genetic Programming and Evolvable Machines* 11, 3/4 (Sept. 2010), 321–338. https://doi.org/doi:10.1007/s10710-010-9111-4

[12] William B. Langdon and Mark Harman. 2014. Genetically Improved CUDA C++ Software. In *Revised Selected Papers of the 17th European Conference on Genetic Programming - Volume 8599 (EuroGP 2014)*. Springer-Verlag New York, Inc., New York, NY, USA, 87–99.

[13] William B. Langdon, Brian Yee Hong Lam, Justyna Petke, and Mark Harman. 2015. Improving CUDA DNA Analysis Software with Genetic Programming. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO '15)*. ACM, New York, NY, USA, 1063–1070.

[14] William B. Langdon, Marc Modat, Justyna Petke, and Mark Harman. 2014. Improving 3D Medical Image Registration CUDA Software with Genetic Programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO '14)*. ACM, New York, NY, USA, 951–958.

[15] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. 2012. GenProg: A Generic Method for Automatic Software Repair. *IEEE Trans. Softw. Eng.* 38, 1 (Jan. 2012), 54–72.

[16] Joel Lehman and Kenneth O. Stanley. 2008. Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. In *Proc. of the Eleventh Intl. Conf. on Artificial Life (ALIFE XI)*. MIT Press, Cambridge, MA.

[17] Ke Mao, Mark Harman, and Yue Jia. 2016. Sapienz: Multi-objective Automated Testing for Android Applications. In *Proceedings of the 25th International Symposium on Software Testing and Analysis (ISSTA 2016)*. ACM, New York, NY, USA, 94–105.

[18] Yuliana Martínez, Enrique Naredo, Leonardo Trujillo, and Edgar Galván López. 2013. Searching for novel regression functions. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*. 16–23.

[19] Luigi Nardi, Bruno Bodin, Sajad Saeedi, Emanuele Vespa, Andrew J. Davison, and Paul H. J. Kelly. 2017. Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper. *CoRR* abs/1702.00505 (2017).

[20] Luigi Nardi, Bruno Bodin, M. Zeeshan Zia, John Mawer, Andy Nisbet, Paul H. J. Kelly, Andrew J. Davison, Mikel Luján, Michael F. P. O'Boyle, Graham Riley, Nigel Topham, and Steve Furber. 2015. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[21] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. LujÃąn, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber. 2015. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 5783–5790.

[22] Enrique Naredo, Miguel Aurelio Duarte-Villaseñor, Manuel de Jesús García Ortega, Carlos E. Vázquez López, Leonardo Trujillo, and Oscar S. Siordia. 2016. Novelty Search for the Synthesis of Current Followers. *Computación y Sistemas* 20, 4 (2016).

[23] Enrique Naredo and Leonardo Trujillo. 2013. Searching for Novel Clustering Programs. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO '13)*. ACM, New York, NY, USA, 1093–1100. https://doi.org/10.1145/2463372.2463505

[24] Enrique Naredo, Leonardo Trujillo, Pierrick Legrand, Sara Silva, and Luis Muñoz. 2016. Evolving genetic programming classifiers with novelty search. *Inf. Sci.* 369 (2016), 347–367.

[25] Enrique Naredo, Paulo Urbano, and Leonardo Trujillo. 2017. The training set and generalization in grammatical evolution for autonomous agent navigation. *Soft Comput.* 21, 15 (2017), 4399–4416.

[26] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time Dense Surface Mapping and Tracking. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR '11)*. IEEE Computer Society, Washington, DC, USA, 127–136.

[27] J. Petke, S. Haraldsson, M. Harman, w. langdon, D. White, and J. Woodward. 2017. Genetic Improvement of Software: a Comprehensive Survey. *IEEE Transactions on Evolutionary Computation* PP, 99 (2017), 1–1.

[28] Justyna Petke, Mark Harman, William B. Langdon, and Westley Weimer. 2014. Using Genetic Improvement and Code Transplants to Specialise a C++ Program to a Problem Class. In *Revised Selected Papers of the 17th European Conference on Genetic Programming - Volume 8599 (EuroGP 2014)*. Springer-Verlag New York, Inc., New York, NY, USA, 137–149.

[29] O. Tange. 2011. GNU Parallel - The Command-Line Power Tool. *;login: The USENIX Magazine* 36, 1 (Feb 2011), 42–47.

[30] Paulo Urbano, Enrique Naredo, and Leonardo Trujillo. 2014. Generalization in Maze Navigation Using Grammatical Evolution and Novelty Search. In *Theory and Practice of Natural Computing*, Adrian-Horia Dediu, Manuel Lozano, and Carlos Martín-Vide (Eds.). Springer International Publishing, Cham, 35–46.

[31] M. Z. Zia, L. Nardi, A. Jack, E. Vespa, B. Bodin, P. H. J. Kelly, and A. J. Davison. 2016. Comparative design space exploration of dense and semi-dense SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 1292–1299.