Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper

Algorithmic Performance-Accuracy Trade-off in 3D Vision Applications Using HyperMapper

Luigi Nardi*, Bruno Bodin†, Sajad Saeedi*,

Emanuele Vespa*, Andrew J. Davison*, Paul H. J. Kelly*

*Department of Computing, Imperial College London
London, UK
{l.nardi, s.saeedi, e.vespa14, ajd, p.kelly}@imperial.ac.uk
†Institute for Computing Systems Architecture, The University of Edinburgh
Edinburgh, Scotland
bbodin@inf.ed.ac.uk

***Abstract*—In this paper we investigate an emerging application, 3D scene understanding, likely to be significant in the mobile space in the near future. The goal of this exploration is to reduce execution time while meeting our quality of result objectives. In previous work, we showed for the first time that it is possible to map this application to power constrained embedded systems, highlighting that decision choices made at the algorithmic design-level have the most significant impact.**

As the algorithmic design space is too large to be exhaustively evaluated, we use a previously introduced multi-objective random forest active learning prediction framework dubbed HyperMapper, to find good algorithmic designs. We show that HyperMapper generalizes on a recent cutting edge 3D scene understanding algorithm and on a modern GPU-based computer architecture. HyperMapper is able to beat an expert human hand-tuning the algorithmic parameters of the class of computer vision applications taken under consideration in this paper automatically. In addition, we use crowd-sourcing using a 3D scene understanding Android app to show that the Pareto front obtained on an embedded system can be used to accelerate the same application on all the 83 smart-phones and tablets with speedups ranging from 2x to over 12x.

***Index Terms*—design space exploration; machine learning; computer vision; SLAM; embedded systems; GPU; crowd-sourcing;**

I. Introduction

Motivated by the increasing complexity of hardware and software components, automatic performance tuning techniques have flourished in the past few years. While closed-form mathematical performance models have been successfully applied to compiler optimizations, they often lack of accuracy or expressive power, which could undermine the ability to capture the complex interactions that occur between tool-chain and hardware parameters. This intricacy is exacerbated when including algorithmic parameters in the tuning practice, where deep domain knowledge may be required to meet multiple conflicting design goals. This paper extends the work done in [40] where the HyperMapper framework was introduced. The authors showed how going beyond conventional benchmarking in computer systems research is possible by exposing the algorithmic-level design space. They used a vertical approach to design and program heterogeneous MPSoCs exploring all levels of the stack, from compilers to the micro-architecture, to optimally map the executed code onto such diverse hardware resources. The authors found that the algorithmic space enables important approximate computing research exploring trade-off in accuracy and performance. The rationale behind including the algorithmic parameters in the design space exploration is that although these algorithms are tuned for desktop systems, the same configurations are not optimal in a mobile MPSoC setting.

In this paper we focus on one set of emerging applications that is becoming significant in the mobile space: real-time 3D scene understanding in computer vision. In particular, we investigate dense simultaneous localization and mapping (dense SLAM) algorithms which are extremely computationally demanding. One such dense SLAM algorithm is KinectFusion [30] (KFusion) which estimates the pose of a depth camera whilst constructing a highly detailed 3D model of the environment. Another well-known dense SLAM algorithm is the ElasticFusion algorithm [42]. While in KinectFusion the map is shown by dense voxels, in ElasticFusion, the map is shown by small disc-shaped objects called surfels. Unlike KinectFusion, ElasticFusion has loop closure functionality built in and uses both the depth and the RGB cameras. In this paper, KinectFusion and ElasticFusion are the benchmarks on which the experiments are performed. Since these algorithms are typically tuned for high-end desktops with high power budget, executing them on power-constrained embedded devices is very challenging and, therefore, represents a realistic future application use case. We use the SLAMBench benchmarking framework [28], which contains KFusion [30] and ElasticFusion [42] implementations, as it allows us to capture the performance metrics used to drive our design space exploration.

We define the performance in terms of accuracy of estimated trajectory (in centimeters, lower is better) and runtime (measured as wall clock time per frame in seconds, lower is better). The runtime is sometimes also quantified by the number of

frames processed in one second, i.e. frames per second (FPS), higher is better; the current Microsoft Kinect (or equivalent ASUS Xtion Pro) RGB-D sensor runs at 30 FPS, so 30 FPS is needed for real-time processing. These two metrics interact and are considered simultaneously for a through evaluation of the system.

Since the algorithmic design space can be extremely large, it is not feasible to try all possible configurations. Instead, we sample the domain space and automatically build a model that predicts the two performance metrics for a given configuration. Using this model, and a methodology from machine learning known as active learning, we predict a two dimensional performance Pareto-optimal configurations curve that can be then stored on the machine to support dynamic adaptation, automatically selecting the best combination of algorithmic parameters for a given scene and accuracy-performance objective. While a human might pick one good design, we generate a whole Pareto front, with hundreds of design alternatives each optimal for a given situation. This enable us to dynamically adapt while being sure to be close to optimal whatever our context. This requires far more work than a human could do. The human gets to focus on setting the objectives.

In previous work [40], by exploring the resulting Pareto curve on the KFusion application we obtain a mapping to an embedded platform that results in a 6.6-fold speedup over the original mobile implementation. More precisely, this new configuration runs at nearly 40 FPS while maintaining an acceptable accuracy (under 5 cm localization error) and keeping power consumption under 2 Watts. The Pareto front contains many more configurations, allowing us to trade between runtime, power consumption, and accuracy, depending on our desired goals. For example, we can also find points which minimize power consumption (e.g., a configuration providing 11.92 FPS at 0.65W) or optimized for execution time without exceeding a given power budget (29.09 FPS at less than 1W).

Additionally, in a recent work [41], we have demonstrated that the design space exploration can be extended to include physical parameters such as the motion of the camera and the structure of the environment. By this extension, not only the performance of the SLAM algorithm is improved, but also the robustness is increased.

This paper demonstrates that HyperMapper generalizes on a recent cutting edge 3D scene understanding algorithm, i.e. ElasticFusion [42]. By exploring the resulting Pareto curve we obtain a mapping to a modern discrete GPU-based system which is a very similar machine to the one used by the ElasticFusion developers. That results in a 1.52-fold speedup over the original design defined by the default configuration while also improving accuracy. The default configuration was defined by the the original developers of ElasticFusion. Another configuration shows a 2-fold improvement in accuracy (2.69 cm) compared to the default configuration (5.58 cm) with a speedup of 1.25. HyperMapper is able to beat an expert human hand-tuning the algorithmic parameters of the class of Computer Vision applications taken under consideration in this paper automatically.

In addition, we use crowd-sourcing using the Android SLAMBench KFusion app to show that the Pareto front obtained on an embedded system can be used to speed up the same application on all the 83 smart-phones and tablets crowd-sourced with speedups ranging from 2 to over 12.

The main contributions of the paper are:

- We demonstrate how HyperMapper's algorithmic design-space exploration generalizes across two very different applications, i.e. KFusion and ElasticFusion, and on multiple devices.
- In order to explore the potential for this approach we evaluate our methodology on an emerging SLAM benchmarking framework, i.e. SLAMBench, which supports quantitative evaluation of solution accuracy and execution time. On the new application considered, ElasticFusion, we obtain a a 1.52x best improvement in execution time and 2-fold improvement in accuracy over an hand-tuned implementation by a SLAM domain expert.
- We show how the algorithmic Pareto front learned on one device speeds up a variety of smart-phones and tablets evaluated using a crowd-sourcing experiment.

II. Background

Simultaneous localization and mapping (SLAM) systems aim to perform real-time localization and mapping "simultaneously" from a sensor moving through an unknown environment. Localization typically estimates the location and pose of the sensor with respect to a map which is extended as the sensor explores the environment. Dense SLAM systems in particular map entire 3D surfaces, as opposed to non-dense (feature-based) systems where maps are represented at the level of sparse point landmarks. Dense SLAM systems enable a mobile robot to perform path planning and collision avoidance, or an augmented reality (AR) system to render physically plausible animations at appropriate locations in the scene. Recent advances in computer vision have led to the development of real-time algorithms for dense SLAM such as KFusion [30] and ElasticFusion [42]. Such algorithms estimate the pose of a depth camera while building a highly detailed 3D model of the environment (see [4]).

Such real-time 3D scene understanding capabilities can radically change the way robots interact with the world. While classical feature-based SLAM techniques are now crossing into mainstream products via embedded implementations, such as Project Tango [3] and Dyson 360 Eye [2], *dense* SLAM algorithms with their high computational requirements are largely at the prototype stage on GPU-based PC or laptop platforms [30], [42]. However, when running in an embedded context, it is not feasible to include a large GPU with high power and cooling requirements. In addition, even when running on a desktop system it is important to run on an optimal design configuration because this would save machine resources that would allow to reduce the overall system latency. While offloading to a remote machine is possible in some circumstances, this can introduce additional latency

which makes it unsuitable for real-time situations such as AR or UAV navigation applications.

KFusion registers and fuses the stream of measured noisy depth frames from a depth camera (such as Microsoft Kinect), as the scene is viewed from different viewpoints, into a clean 3D geometric map. It is beyond the scope of this paper to go into the details of the KFusion algorithm, we briefly outline the key computational steps involved, for more information the reader should refer to [40]. SLAMBench provides multiple implementations of the KFusion algorithm. We use the OpenCL implementation, and execute each OpenCL kernel on the GPU of our target platforms.

KFusion normalizes each incoming depth frame and applies a bilateral filter (*Preprocessing*) to reduce noise. In the *Tracking* step, it computes a point cloud (with normals) for each pixel in the camera frame of reference and estimates the new 3D pose of the moving camera by registering this point cloud with the current global map using iterative closest point (ICP) [10]. Once the new camera pose has been estimated, the corresponding depth map is fused into the current 3D reconstruction (*Integration*). KFusion utilizes a voxel grid as the data structure to represent the map, employing a truncated signed distance function (TSDF) to represent 3D surfaces. The 3D surfaces are present at the zero crossings of the TSDF and can be recovered by a *Raycasting* step, which is also useful for visualizing the reconstruction.

ElasticFusion is an incremental, dense SLAM algorithm, which supports local and global loop closure without employing an explicit pose graph. The algorithm creates a surfel-based model of the environment. Each frame attempts to perform local loop closures - to stay close to the mode of the map distribution, and also global loop closures - to avoid drift and maintain global consistency.

In this work we use the SLAMBench benchmarking framework [28] which enables evaluation of runtime and accuracy for KFusion and ElasticFusion. SLAMBench is provided with an absolute trajectory error (ATE) metric. The ATE is domain-specific accuracy metric calculated as the mean difference between the real trajectory and the estimated trajectory of a camera produced by a SLAM implementation. Thus, smaller ATE implies less deviation from the real trajectory.

Hand optimization of algorithmic parameters in SLAM applications is in general not feasible. Fig. 1 shows the KFusion runtime response surface when varying two algorithmic parameters, keeping the rest of the parameters as for the default configuration. The picture depicts a non-convex, multi-modal and non-smooth runtime response surface which is in general very difficult to hand-tune by trial and error.

III. Design Spaces and Methodology

In this section we describe our approach, including a detailed explanation of the design space parameters and the objectives which we are targeting. In Section III-E, we go on to describe the search techniques we use to guide our exploration through the design space. The methodology used is the same as the one in [40], the reader can refer to that work for more information.

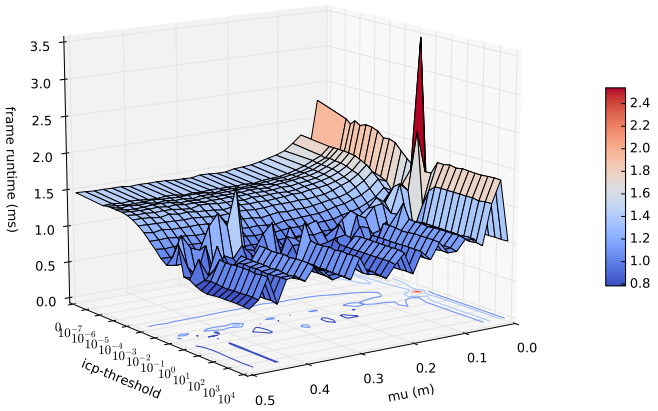


Fig. 1: KFusion runtime response surface when varying just two parameters mu and $icp - threshold$, keeping the rest of the parameters as for the default configuration. This shows the non-convexity, multi-modality and non-smoothness between configurations.

A. Experimental Setting

In order to evaluate our design space exploration (DSE) we use the SLAMBench framework with the ICL-NUIM [19], [18] dataset, specifically the first 400 frames of living room trajectory 2. We halved the original sequence in order to reduce the overall execution time of the benchmark; this was done after careful consideration that the accuracy metric is still representative of the whole sequence.

Usual approaches in performance optimization consider benchmark suites that are, in general, a set of small kernels extracted from real applications. A criticism to what can be learnt from a benchmark suite is that they may not well represent and capture the complex interaction of kernels in a real-world application. Our applications are composed of more than 10 GPU-accelerated kernels. They present the opportunity to tackle exploration of parameters at the algorithmic level that is not possible with conventional benchmark suites.

During execution, the following two performance metrics are collected: 1) computation time and, 2) absolute trajectory error (ATE) of the frame sequence.

B. KFusion Design Space

We summarize the algorithmic parameters that mostly affect our performance metrics. In the case of the SLAMBench implementation of the KFusion algorithm, we have access to the listed parameters. An extensive explanation of these can be found in [30], [28].

- **Volume resolution**: The resolution of the scene being reconstructed. As an example, a 64x64x64 voxel grid captures less detail than a 256x256x256 voxel grid.
- μ **distance**: The output volume of KFusion is defined as a truncated signed distance function (TSDF) [30]. Every

volume element (voxel) of the volume contains the best likelihood distance to the nearest visible surface, up to a truncation distance denoted by the parameter μ, also referred as mu in the text.

- **Pyramid level iterations**: The number of block averaging iterations to perform while building each level of the image pyramid.
- **Compute size ratio**: The fractional depth image resolution used as input. As an example, a value of 8 means that the raw frame is resized to one-eighth resolution.
- **Tracking rate**: The rate at which the KFusion algorithm attempts to perform localization. A new localization is performed after every tracking rate number of frames.
- **ICP threshold**: The threshold for the iterative closest point (ICP) algorithm [10] used during the tracking phase.
- **Integration rate**: As the output of KFusion is a volumetric representation of the recorded scene, it needs to be repeatedly expanded using new frames. A new frame is integrated after every integration rate number of frames.

We observe that the KFusion algorithmic design space consists of roughly 1,800,000 points.

C. ElasticFusion Design Space

We summarize the algorithmic parameters considered in the ElasticFusion design exploration via the SLAMBench framework. An extensive explanation of these can be found in [42].

In general, there are two categories of parameters in ElasticFusion: algorithmic parameters, thresholds and flags. The algorithmic parameters and thresholds considered in our exploration are:

- **ICP/RGB weight**: Relative ICP/RGB tracking weight. Incremental pose estimation is done both in the photometric RGB space and the geometric depth space. Then the results are merged by applying this weight parameter.
- **Depth cut off**: Cutoff distance for depth processing. The algorithm ignores the raw depth input larger than this threshold.
- **Confidence threshold**: Surfel confidence threshold. As a surfel is observed more, the confidence of the surfel increases. Once the confidence of the surfel is larger than a threshold, it is included in the processing pipeline. Lowering this threshold will create a noisy map.

The flags are:

- **Disable SO3 pre-alignment**: While tracking, setting this flag disables pre-alignment in 3D rotation group, known as SO(3).
- **Open loop**: Setting flag disables the local loop closure code in ElasticFusion,
- **Relocalisation**: By setting this flag, ElasticFusion attempts to relocate its pose, i.e.'get back on track' if it is lost,
- **Fast odometry**: By setting this flag, the RGB odometry uses only a single level pyramid, hence faster processing.
- **Frame to frame RGB**: Setting this flag enables frame-to-frame RGB tracking.

We observe that the ElasticFusion algorithmic design space consists of roughly 450,000 possible configurations. Exploration on ElasticFusion is a work in progress, additional parameters that significantly affect performance will be considered in future explorations, e.g. compute size ratio, ICP error threshold, ICP count threshold, covariance threshold, photometric threshold, and fern threshold.

D. Multi-Objective Optimization Goal

In a multi-objective optimization setting, a single solution that minimizes all performance metrics simultaneously does not exist in general. Therefore, attention is paid to Pareto-optimal solutions — that is, solutions that cannot be improved in any of the objectives without degrading at least one of the other objectives.

E. Design Space Exploration Tool

The algorithmic parameter space we are investigating is too large to be exhaustively evaluated on the hardware platform. We use HyperMapper introduced in [40] to a cheaper route of training a predictive machine learning model over a handful of examples (points in the parameter space) evaluated on hardware. HyperMapper accurately create a surrogate model and predicts the performance over the entire parameter space, while being many orders of magnitude faster as compared to running the application on hardware over a video sequence for big parameter settings. Unfortunately since we do not know the performance over the parameter space, we are also unaware of the points for which running a physical experiment will be most informative, in the sense of yielding the greatest increase in the prediction accuracy of our model - a classic chicken and egg problem. Thus, we resort to bootstrapping predictive models (two separate randomized decision forests [11] for accuracy and runtime prediction) from a small number of randomly drawn samples in the parameter space. These models are then refined in subsequent iterations by drawing more samples from the parameter space (and retraining over the collective set); the new samples are now drawn to implicitly maximize the prediction accuracy near the respective Pareto optimal fronts. This strategy of letting the predictive model decide which samples will be most beneficial in increasing predictive accuracy over unseen regions of the parameter space is called active learning [14].

The combination of many weak regressors (binary decisions) allows approximating highly non-linear and multi-modal functions with great accuracy. HyperMapper trains separate regressors to learn the mapping from our input (parameter) space to each output variable, i.e. the two performance metrics. This methodology is depicted in Figure 2 from the original work [40] and explained in the next sections. Refer to the original paper for more information.

Active learning is a paradigm in supervised machine learning which uses fewer training examples to achieve better prediction accuracy - by iteratively training a predictor, and using the predictor in each iteration to choose the training

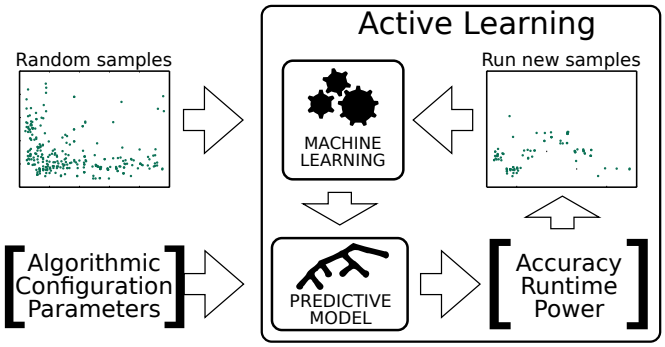


Fig. 2: The learning step is based on a tiny subset of the overall algorithmic space; these are the samples that are actually run. Subsequently, the predictive model can predict accuracy and performance of an unseen configuration depending on its parameters.

examples which will improve its performance over a predefined objective. Thus the accuracy of the predictive model is incrementally improved by interleaving exploration and exploitation steps, as shown by the feedback loop in Figure 2. Since our objective is to accurately estimate the points near the Pareto front, we use the current predictor to provide performance values over the entire parameter space and thus estimate the Pareto fronts for accuracy and runtime (separately). For the next iteration, only parameter points near the predicted Pareto front are sampled (and evaluated on hardware), and subsequently used to train new predictors using the entire collection of training points from current and all previous iterations. This process is repeated over a number of iterations.

Algorithm 1 shows the pseudo-code of the model-based search algorithm used in HyperMapper.

Data: config. pool X, random sampling batch size rs
Result: Pareto front P
$X_{out} \leftarrow$ sample rs distinct configurations from X;
$Y_{ATE}, Y_{run} \leftarrow Evaluate(X_{out})$;
$M_{ATE} \leftarrow Fit\_Random\_Forest(X_{out}, Y_{ATE})$;
$M_{run} \leftarrow Fit\_Random\_Forest(X_{out}, Y_{run})$;
$P \leftarrow Predict\_Pareto(M_{ATE}, M_{run}, X)$;
while $P - X_{out} \neq \emptyset$ **do**
  $x \leftarrow P - X_{out}$;
  $y_{ATE}, y_{run} \leftarrow Evaluate(x)$;
  $X_{out} \leftarrow X_{out} \cup x$;
  $Y_{ATE} \leftarrow Y_{ATE} \cup y_{ATE}$;
  $Y_{run} \leftarrow Y_{run} \cup y_{run}$;
  $M_{ATE} \leftarrow Fit\_Random\_Forest(X_{out}, Y_{ATE})$;
  $M_{run} \leftarrow Fit\_Random\_Forest(X_{out}, Y_{run})$;
  $P \leftarrow Predict\_Pareto(M_{ATE}, M_{run}, X)$;
end
return P;

Algorithm 1: Pseudo-code for HyperMapper. $-$ denotes set difference and $\cup$ denotes set union.

IV. Experiments

In this section we describe how we evaluated our design space exploration. We begin by providing a more detailed description of the target platforms (Section IV-A). We then briefly summarize our key results (Section IV-B), before providing more detail on the results of the generalization of HyperMapper in Section IV-C. For completeness and ease of comparison we partially report the results from our previous work on HyperMapper and design space exploration for the KFusion benchmark [40].

A. Platforms

For our experiments we use a Hardkernel ODROID-XU3 platform based on the Samsung Exynos 5422, an ASUS T200TA with an Intel Atom Z3795, and a desktop computer with an NVIDIA GTX 780 Ti GPU.

The Exynos 5422 includes a Mali-T628-MP6 GPU alongside ARM's big.LITTLE heterogeneous multiprocessing solution, consisting of four Cortex-A15 "big" performance tuned out-of-order processors, and four Cortex-A7 "LITTLE" energy tuned in-order processors. The Mali-T628-MP6 GPU consists of two separate OpenCL devices: one with four cores and another with two. In our experiments we only use the 4-core OpenCL which excludes partitioning tasks across multiple GPU devices. This is a potential avenue to explore in order to deliver even higher performance within a power budget. The ODROID-XU3 supports OpenCL 1.1 and the GNU gcc compiler version 4.8.2 is used.

The ASUS Transformer T200 tablet contains an Intel Atom Z3795 SoC, which includes a quad-core Intel Atom CPU running at up to 2.4 GHz. An Intel HD Graphics GPU is also present, containing 6 execution units and running at up to 778 MHz. We use the open source Beignet [1] OpenCL runtime which supports version 1.2 of the OpenCL standard and was produced by Intel's Open Technology Center. The GNU gcc compiler used is 5.3.1.

The desktop machine we used is a 8-core Intel Ivy Bridge E5-1620 v2 CPU augmented with a high-end discrete NVIDIA GPU GTX 780 Ti. The CUDA toolkit version is 7.5.18 and OpenGL version is 1.4. The machine runs Ubuntu OS kernel 14.04.4 and the GNU gcc compiler version 4.8.4 is used.

We run KFusion on the ODROID-XU3 and ASUS mobile platforms using OpenCL, and ElasticFusion on the NVIDIA desktop using CUDA.

B. Outcome in a glance

We observe that the default KFusion configuration provides a frame-rate of 6 FPS on the ODROID-XU3 embedded system. Our design space exploration results show significantly better frame-rates with comparable accuracy. For example, a configuration exists in the real-time range (29.09 FPS) and with a similar accuracy ATE compared to the default configuration (4.47 cm). These results are consistent also on the ASUS machine. In addition, the selected best configurations perform well across a wide range of 83 mobile platforms crowd-sourced. These platforms are running the Android version of

SLAMBench configured using the Pareto front, with speedups ranging from 2 to 12 over default.

We observe that the default ElasticFusion configuration provides a frame-rate of 45 FPS on the NVIDIA desktop machine. Our design space exploration results show significantly better frame-rates and better accuracy. For example, a configuration exists that speeds-up the runtime by 1.52 compared to default while also improving accuracy. Another configuration shows a 2-fold improvement in accuracy (2.69 cm) compared to the default configuration (5.58 cm) with a speedup of 1.25.

Active learning effectively and consistently pushes the Pareto front toward better solutions. Taking into account the domain layer of the stack unleashes unprecedented performance trade-offs compared to the more usual compiler optimizations.

C. Exploration

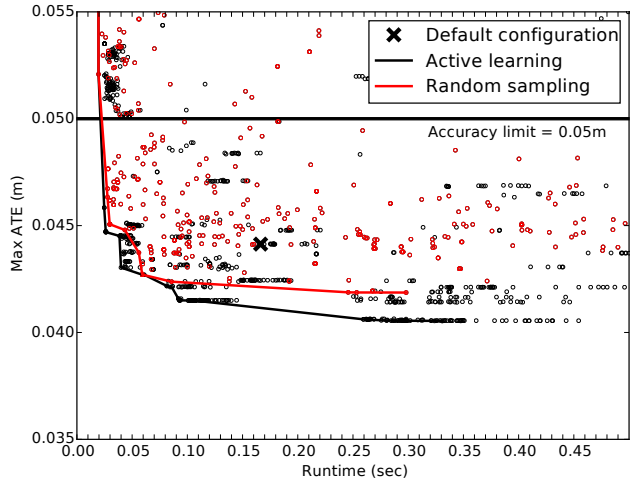
The algorithmic space consists of application parameters described in Section III-B and III-C. As described in Section III-E, we first sample this space at random, and then use active learning in order to push the Pareto front toward better solutions (refer to Figure 3).

a) Sampling: We draw 3,000 uniformly distributed random samples from the parameter space and evaluate the KFusion pipeline on the video stream; for both platforms the cumulated runtimes take roughly 5 days. By using random sampling, we observe that the Pareto front cannot be improved beyond 2,000 samples. Thus, there is an inflection point beyond which random sampling is unproductive.

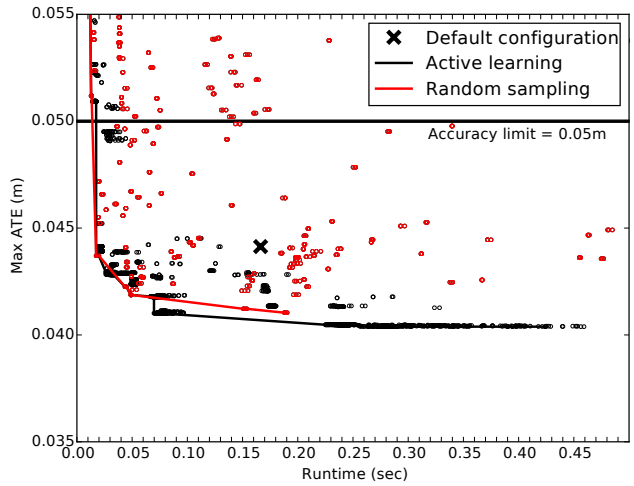
A similar number of uniformly distributed samples (2,400) is used on ElasticFusion running on the NVIDIA machine.

b) Active learning: In order to further explore optimal points in the design space, we employ active learning in conjunction with random decision forest (see III-E). For the KFusion benchmark on ODROID-XU3 this produces 1,142 new samples after 6 iterations, thus increasing the total number of samples to 4,142. Note that the number of samples produced per iteration is not constant as it depends on the predicted points' proximity to the Pareto front. We observe that the number of samples per iteration varies between 100 and 300. The runtime of these new configurations was faster, close to a day, as most of these configurations were good configurations (accurate and fast). The training of the random forest model was fast as well, less than two minutes for every iteration. With the ASUS T200TA platform, 1392 new points has been produced by active learning. On ElasticFusion on the NVIDIA platform 999 active learning points are collected.

c) Active learning effectiveness: Figure 3 shows the overall improvement of the Pareto front obtained with active learning (in black) compared to the Pareto obtained with random sampling (in red). For the ODROID-XU3 we observe that random sampling provides a set of 333 valid configurations, i.e. 333 configurations with a max ATE smaller than 5 cm. For the ASUS T200TA, we found 291 valid configurations during the sampling. Furthermore, by using the active learning technique, we observe 642 new possible configurations with



(a) ODROID-XU3

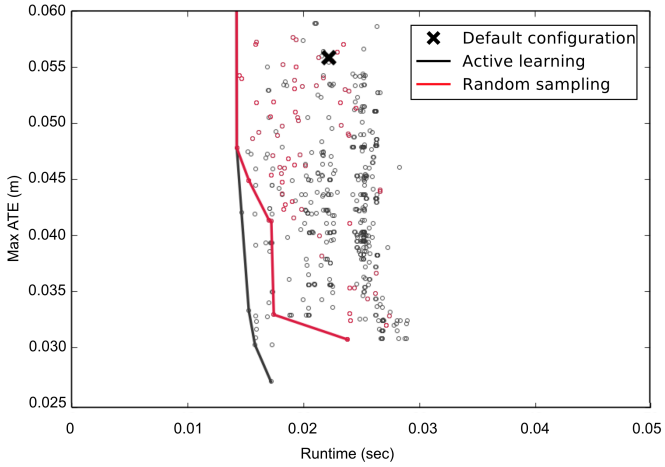


(b) ASUS T200TA

Fig. 3: Algorithmic design space exploration on the KFusion benchmark like shown in the original paper [40]. Random sampling (red) and active learning (black).

an ATE of less than 5 cm on the ODROID-XU3, and 665 on the ASUS T200TA. This means we have produced twice as many valid points as random sampling, for roughly a third of the number of samples. These ratios are an indicator of the effectiveness of our active learning-based prediction model. There is a discrepancy between predicted and measured performance. This is shown by the active learning points in Figure 3 that do not lie on the Pareto front. Note that there are 36 points on the Pareto front for the ODROID-XU3 and 167 points for the ASUS T200TA.

Figure 4 shows the overall improvement on the ElasticFusion benchmark. This plot shows that HyperMapper is able to generalize the results obtained on KFusion on a fundamentally different and complex benchmark. Here again the active learning is consistently achieving an improvement in execution time and accuracy with respect to random sampling.



NVIDIA 780 Ti

Fig. 4: Algorithmic design space exploration on the ElasticFusion benchmark. Random sampling (red) and active learning (black).

See Table I for details on the Pareto front and its algorithmic configuration values.

By using the described techniques to explore the algorithmic spaces, we have obtained a 6.35x improvement in execution time (best speed), compared to the default configuration on the ODROID-XU3 board. This important speedup can be explained by the fact that the application was tuned on a fundamentally different machine by the original developer, i.e. a NVIDIA Quadro GPU-based desktop. It is then not surprising that this default configuration performs poorly on a new target, the ODROID-XU3 or the ASUS in this case.

The best speed up on the NVIDIA GPU is 1.52 while at the same time improving accuracy by 1.33, see Table I. With respect to KFusion on ODROID-XU3 and ASUS, ElasticFusion on the NVIDIA GPU is a different test case. The ElasticFusion developers used a similar NVIDIA GTX machine to develop the application and, in addition, they used a brute force grid search to tune the parameters. HyperMapper is able to beat the human when compared to a similar setting than the hand-tuning one. Additionally HyperMapper is also able to find a configuration that performs 2x better in terms of accuracy.

In [40] we showed the correlation of the feature space with the runtime and the error metrics. We invite the reader to refer to that paper for correlation analysis.

D. Crowd-sourcing

An Android app has been developed for the KFusion benchmark of SLAMBench [5]. People can freely download the SLAMBench app and automatically run the best performing (best runtime) algorithmic configuration from the Pareto front computed on the ODROID-XU3 board together with the default configuration in order to benchmark their devices. For practical reasons, only 100 frames are run. The app automatically collects the results and send it over the network to a centralized database for analysis purposes. In total 83 platforms ran the app. Figure 5 shows this crowd-sourcing experiment. The plot shows the speedup over the default configuration run on the same device. The speedup ranges between 2 and more than 12. This result confirms the hypothesis that best performing configurations found on one machine usually perform well also on different but similar machines. Specifically, most of the mobile devices in the market are ARM-based devices and these are the devices that populate our crowd-sourcing experiment. In [43] the authors show that there is a strong Pearson and Spearman correlation between configurations that perform well on one machine and configurations that perform well on another machine. And so that results on one machine can be often used to speed up a second machine. This is a form of zero-shot transfer learning that does not guarantee optimality but is showing to be effective. In [43] the authors also show that the zero-shot learning approach does not seem to work in general when the machines are fundamentally different, like for example from an Intel SandyBridge to an AppliedMicro X-Gene ARM 64-bit.

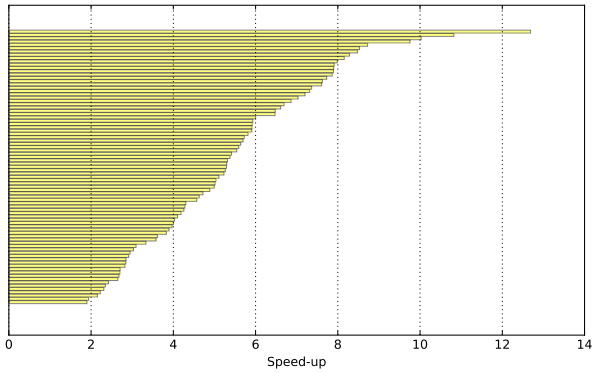


Fig. 5: The OpenCL KinectFusion has been run on 83 smartphones and tablets from the market. For each device, we computed the speedup of the best configuration we found for the ODROID-XU3 and the original default configuration.

V. Related work

The computer vision community primarily focuses on developing accurate algorithms [19], [37], almost always running on high-performance and power hungry systems. As computer vision technology becomes mature, a few benchmarks [38], [13], [35] have attempted to refocus research on runtime constrained contexts. Similarly, new challenges such as the Low-Power Image Recognition Challenge (LPIRC 2016) are emphasizing the importance of low-power embedded implementations of computer vision applications. In this context, recently SLAMBench [28] enabled quantitative, comparable, and validatable experimental research in the form of a benchmark framework for dense 3D scene understanding on a wide range of devices. Adding energy consumption as a metric when evaluating

	Error (m)	Runtime (s)	ICP	Depth	Confidence	SO3	Close-Loops	Reloc	Fast-Odom	FTF RGB
Default	0.0558	22.2	10	3	10	1	0	1	0	0
Best speed	0.0420	**14.6**	5	6	9	0	0	1	1	0
	0.0332	15.2	4	6	9	0	0	1	1	0
	0.0302	15.8	2	10	4	0	0	1	1	0
Best accuracy	**0.0269**	17.2	1	10	4	0	0	1	1	0

TABLE I: The Pareto efficiency points as a result of the design space exploration on the ICL NUIM Living Room 2 Dataset. On the top row are reported the results for the default configuration, highlighted in boldface are the fastest and the most accurate configuration.

computer vision applications, has enabled energy constrained systems such as battery-powered robots and embedded devices to become evaluation platforms. Zeeshan et al. [39] is a first attempt at exploring SLAM configuration parameters trading off performance for accuracy on embedded systems. In [44] the authors exploit the SLAMBench framework to explore optimization of multi-kernel application using a dataflow model.

During the last two decades, several design space exploration techniques and frameworks have been used in a variety of different contexts ranging from embedded devices, to compiler research, and system integration. Ansel et al. [7] introduced an extensible and portable framework for empirical performance tuning. It runs an ensemble of search techniques systematically allocating larger budgets to those who perform well, using a multi-armed bandit optimal budget allocation strategy. Norbert et al. tackle the software configurability problem for binary [34] and for both binary and numeric options [33] using a performance-influence model which is based on linear regression. They optimize for execution time on several examples exploring algorithmic and compiler spaces in isolation.

In particular, machine learning (ML) techniques have been recently employed in both architectural and compiler research. Khan et al. [24] employed predictive modeling for cross-program design space exploration in multi-core systems. The techniques developed managed to explore a large design space of chip-multiprocessors running parallel applications with low prediction error. In [45] Balaprakash et al. introduce AutoMOMML, an end-to-end, ML-based framework to build predictive models for objectives such as performance, and power. [46] presents the ab-dynaTree active learning parallel algorithm that builds surrogate performance models for scientific kernels and workloads on single-core, multicore and multinode architectures. In [48] the authors propose the Pareto Active Learning (PAL) algorithm which intelligently samples the design space to predict the Pareto-optimal set.

VI. Conclusions and Future Work

3D scene understanding algorithms are generally complex and depend on a number of parameters that subtly interact with each other. Further configuration choices at the compiler and hardware level increase the mapping complexity, effectively making the manual tuning practice very difficult and often simply unfeasible. In this paper we demonstrated how the HyperMapper tool introduced in [40] is effective across different SLAM algorithm implementations and different hardware platforms. Crucially, ElasticFusion's computational profile is very different from KFusion, confirming the robustness of our approach. The crowd-sourced data allowed us to access a variety of devices and simulation settings, e.g. HW platforms, compiler and operating system versions, and showed consistent and important speedups on today market mobile platforms.

In future work, we aim to add more SLAM input data-sets in order to encompass a larger number of scenarios, providing more breadth in terms of trajectories and real-world use cases. Regarding HyperMapper, we will not only investigate new techniques to reduce the dimension of the design space, but also more advanced transfer learning and resampling techniques. A powerful application of such techniques would be to treat multiple algorithms, compilers and platforms on the same tuning session, effectively enacting an algorithm selection tailored to the specific operative scenario. In this case domain-specific languages (DSLs) [36], [47] would be the perfect vehicle to harness the algorithmic exploration automatically.

VII. Acknowledgments

We acknowledge funding by the EPSRC grant PAMELA EP/K008730/1. We thank the PAMELA Steering Group for the useful discussions. We also thank the various students that contributed to this project, in particular Denise Carroll and Alfonso White.

References

[1] Beignet. https://www.freedesktop.org/wiki/Software/Beignet/.
[2] Dyson 360 Eye web site. http://www.dyson.co.uk/vacuums/robot/Dyson-360-Eye/.
[3] Project Tango web site. https://www.google.com/atap/projecttango.
[4] SLAMBench web site. http://apt.cs.manchester.ac.uk/projects/PAMELA/tools/SLAMBench.
[5] SLAMBench Google store app. https://play.google.com/store/apps/details?id=project.pamela.slambench&hl=en_GB.
[6] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. P. O'Boyle, J. Thomson, M. Toussaint, and C. K. I. Williams. Using machine learning to focus iterative optimization. In *CGO 2006*, pages 295–305, Washington, DC, USA, 2006. IEEE Computer Society.
[7] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe. Opentuner: an extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316. ACM, 2014.
[8] ARM Ltd. big.little technology.
[9] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. In *IEEE Computer*, volume 36, pages 45–52, April 2003.

[10] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. In *IEEE Trans. Pattern Anal. Mach. Intell.* volume 14, issue 2, 1992, 239-256.

[11] L. Breiman. *Classification And Regression Trees*. Chapman and Hall, London, UK, 1984.

[12] J. Cavazos, C. Dubach, F. Agakov, E. Bonilla, M. F. P. O'Boyle, G. Fursin, and O. Temam. Automatic performance model construction for the fast software exploration of new hardware designs. In *CASES 2006*, pages 24–34, New York, NY, USA, 2006. ACM.

[13] J. Clemons, H. Zhu, S. Savarese, and T. Austin. MEVBench: A mobile computer vision benchmarking suite. In *IISWC*, 2011.

[14] C. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[15] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

[16] M. Fallon, P. Marion, R. Deits, T. Whelan, M. Antone, J. McDonald, and R. Tedrake. Continuous humanoid locomotion over uneven terrain using stereo fusion. In *ICHR*, 2015.

[17] G. Fursin, Y. Kashnikov, A. Memon, Z. Chamski, O. Temam, M. Namolaru, E. Yom-Tov, B. Mendelson, A. Zaks, E. Courtois, F. Bodin, P. Barnard, E. Ashton, E. Bonilla, J. Thomson, C. K. Williams, and M. O'Boyle. Milepost gcc: Machine learning enabled self-tuning compiler. *International Journal of Parallel Programming*, 39(3):296–327, 2011.

[18] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. SceneNet: Understanding Real World Indoor Scenes With Synthetic Data. *ArXiv e-prints 1511.07041*, 2015.

[19] A. Handa, T. Whelan, J. McDonald, and A. Davison. A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In *ICRA*, 2014.

[20] J. L. Henning. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 2006.

[21] X. Hu, G. Greenwood, S. Ravichandran, and G. Quan. A framework for user assisted design space exploration. In *DAC, 1999*, pages 414–419.

[22] D. Hulens, T. Goedemé, and J. Verbeke. How to choose the best embedded processing platform for on-board UAV image processing? *Proceedings VISAPP 2015*, pages 1–10, 2015.

[23] E. Kang, E. Jackson, and W. Schulte. An approach for effective design space exploration. In *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, volume 6662 of *Lecture Notes in Computer Science*, pages 33–54. Springer Berlin Heidelberg, 2011.

[24] S. Khan, P. Xekalakis, J. Cavazos, and M. Cintra. Using Predictive Modeling for Cross-Program Design Space Exploration in Multicore Systems. In *PACT, 2007*, pages 327–338.

[25] B. C. Lee and D. M. Brooks. Accurate and efficient regression modeling for microarchitectural performance and power prediction. *SIGARCH Comput. Archit. News*, 34(5):185–194, Oct. 2006.

[26] A. Magni, C. Dubach, and M. F. O'Boyle. A large-scale cross-architecture evaluation of thread-coarsening. In *Proc. of SC13: Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. ACM, 2013.

[27] S. Moll. Decompilation of LLVM IR. Master's thesis, 2011.

[28] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *ICRA*, 2015.

[29] S. Neema, J. Sztipanovits, G. Karsai, and K. Butts. Constraint-based design-space exploration and model synthesis. In *Embedded Software*, volume 2855 of *Lecture Notes in Computer Science*, pages 290–305. Springer Berlin Heidelberg, 2003.

[30] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, 2011.

[31] C. Papachristos, D. Tzoumanikas, and A. Tzes. Aerial robotic tracking of a generalized mobile target employing visual and spatio-temporal dynamic subject perception. In *IROS, 2015*, pages 4319–4324.

[32] R. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison. Dense planar SLAM. In *ISMAR*, 2014.

[33] N. Siegmund, A. Grebhahn, S. Apel, and C. Kästner. Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294. ACM, 2015.

[34] N. Siegmund, S. S. Kolesnikov, C. Kästner, S. Apel, D. Batory, M. Rosenmüller, and G. Saake. Predicting performance via automated feature-interaction detection. In *Proceedings of the 34th International Conference on Software Engineering*, pages 167–177. IEEE Press, 2012.

[35] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 567–576, June 2015.

[36] L. Nardi, C. Sorror, F. Badran, S. Thiria YAO: A Software for Variational Data Assimilation Using Numerical Models. In *Computational Science and Its Applications (ICCSA)*, 2009.

[37] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *IROS*, 2012.

[38] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. SD-VBS: The San Diego vision benchmark suite. In *IISWC*, 2009.

[39] M. Z. Zia, L. Nardi, A. Jack, E. Vespa, B. Bodin, P. H. J. Kelly, and A. J. Davison. Comparative Design Space Exploration of Dense and Semi-Dense SLAM. In *ICRA*, 2016.

[40] B. Bodin, L. Nardi, M. Z. Zia, A. Jack, H. Wagstaff, G. S. Shenoy, M.Emani, J. Mawer, C. Kotselidis, A. Nisbet, M. Lujan, B. Franke, P. H. J. Kelly, and M. F. P. O'Boyle. Integrating Algorithmic Parameters into Benchmarking and Design Space Exploration in 3D Scene Understanding. In *PACT* 2016.

[41] S. Saeedi, L. Nardi, E. Johns, B. Bodin, P. H. J. Kelly, and A. J. Davison. Application-oriented Design Space Exploration for SLAM Algorithms. In *ICRA*, 2017.

[42] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison ElasticFusion: Dense SLAM Without A Pose Graph. In *RSS*, 2015.

[43] A. Roy, P. Balaprakash, P. D. Hovland, S. M. Wild Exploiting performance portability in search algorithms for autotuning. In *Parallel and Distributed Processing Symposium Workshops, 2016 IEEE International*, 2016.

[44] B. Bodin, L. Nardi, P. H. J. Kelly, M. F. P. O'Boyle Diplomat: Mapping of Multi-kernel Applications Using a Static Dataflow Abstraction. In *IEEE International Symposium on Modelling, Analyisis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2016.

[45] P. Balaprakash, A. Tiwari, S.M. Wild, L. Carrington, P.D. Hovland Auto-MOMML: Automatic Multi-objective Modeling with Machine Learning. In *International Conference on High Performance Computing, 2016*, 2016.

[46] P. Balaprakash, R. Gramacy, S.M. Wild Active-learning-based surrogate models for empirical performance tuning. In *CLUSTER*, 2013.

[47] L. Nardi, F. Badran, P. Fortin, S. Thiria YAO: A Software for Variational Data Assimilation Using Numerical Models. In *IEEE International Conference on High Performance Computing and Communications (HPCC)*, 2012.

[48] M. Zuluaga,G. Sergent, A. Krause and M. Püschel Active Learning for Multi-Objective Optimization. In *ICML*, 2013.