Testing Approximations for Augmented and Virtual Reality

Samuel Grayson
grayson5@illinois.edu
University of Illinois at Urbana-Champaign

Navi Ning
xning5@illinois.edu
University of Illinois at Urbana-Champaign

ABSTRACT

AR/VR systems have stringent realtime performance, power, and area constraints, which are difficult to simulatenously satisfy.

One way to bring this difficulty down is through the use of *approximation techniques*; In many cases, the output only has to fool a human, so the algorithms can be simplified or reduced. However, one does not know *a priori* which approximations will 'fool the human.' This project seeks automataically determine the acceptability of approximations, so approximation-configurations can be rapidly searched.

1 INTRODUCTION

- A virtual reality system presents the user with an opaque visual display that consumes their entire field-of-vision. Within this field, what the user sees is determined by estimating the pose (position and orientation) of the users head in the real-world and rendering a virtual world from that pose. This gives the user the illusion of being immersed in the virtual world.
- An augmented reality system uses a transparent display, so virtual elements can be overlayed on the physical world. THe virtual objects should move synchronously with the physical objects they overlay, since they are rendered from the user's head-pose.

There are some commercially available AR and VR systems, such as HTC Vive Pro (example of VR) and Microsoft HoloLense 2 (example of AR). However, the quality-of-experience in these systems can stand to be improved. In order to fully immerse the user, more resolution and a faster latency are needed. Furthermore, AR/VR systems need to be mobile/tetherless to support the full breadth of AR/VR applications, and tetherless systems imply a strict power constraint on top of the existing performance constraints. There are multiple orders of magnitude between state-of-the-art AR/VR systems and ideal futuristic systems in performance, power, and area. save space here

AR/VR systems typically use an IMU sensor and a pair of stereo cameras to capture the environment. The system detects visual features in the physical environment, builds up a map of them over time, and uses this map to localize itself (simultaneous localization and mapping or SLAM). Prior work shows that XR systems without audio spend between 10 and 30% in the SLAM computation, depending on the application, platform, physical environment, and user[4]. Existing SLAM algorithms *already have* approximation knobs. In fact, they have 'too many;' nobody knows which ones to turn. Therefore, automatically searching for feasible SLAM approximations could greatly improve system performance.

We wanted to build out a system *in practice* not just in theory. Therefore, we built off of Illinois eXtended Reality System (ILLIXR): an open-source runtime for AR/VR applications cite ILLIXR. Screenshot of ILLIXR

This project seeks to answer or partially answer three questions:

R.Q. 1 How can we test SLAM approximations in ILIXR?

Figure 1: Compute times for various approximation configs (lower is better).

Insert graph

R.Q. 2 How can we do so *automatically*?

R.Q. 3 How can we do so *automatically* and *quickly*?

- Due to time constriants, We only provide a partial implementation of this research question.

2 IMPLEMENTATION

Our implementation is located at .

2.1 R.Q. 1 Approximation Testing

With the following infrastructure, we can test out different approximations and manaully inspect the output for acceptance for **R.Q. 1**. We also need to know how much time the approximation saves, so we know if it is 'worth it.' The data we collected on compute-times are shown in section 2.1.

2.1.1 Approximation Knobs. ILLIXR uses state-of-the-art components such as OpenVINS [3] for SLAM. We located approximation knobs in OpenVINS and consulted domain experts on which ones should be modified and tuned. These are given in href, see comment in LaTeX source.

The exact set of knobs is not so significant, as long as I have some useful ones, some useless ones, and they are orthogonal.

2.1.2 Timing Infrastructure. In order to test the approximations, I need to know how much time they are saving. Timing ILLIXR is not straightforward.

- Whole-program CPU time won't work because, because ILLIXR will find another way to spend the spare cycles.
- `perf` won't work because it does not have dynamic information (the arguments) of the function. It is thus impossible to disambiguate which component to charge a function-call to.
- Language-level tools won't work because they are oblivious to threads launched and joined inside a function. Several functions in OpenVINS launch a short-running thread to parallelize the computation.

Since none of these off-the-shelf solutions will work, we created a lightweight framework for CPU time logging. At each function one wants to instrument, one can call a macro with a static label (such as a function name) and dynamic label (such as arguments that disambiguate the function-call). We use `clock_gettime` to measure the actual time the thread spends scheduled. We use resource-acquisition-in-initialization RAII to automatically time the enclosing scope. This pushes the current call onto a thread-local stack. These stack-frame times get pushed onto a global list and serialized only

Table 1: OpenVINS approximation knobs. See OpenVINS documentation for more details[3].

Knob	Range (approx. first)	Meaning
num_pts	50–300	fill these in
use_rk4_integration	[false, true]	
use_stereo	[false, true]	
use_klt	[false, true]	
downsample_camera	[true, false]	
enable_async_reproject	[false, true]	(outside of OpenVINS, elsewhere in ILLIXR) queries a current pose and reprojects the frame just before display, making up for some latency in SLAM

Figure 2: System-level error for various approximation configs (lower is better).

Insert graph

at the very end of the program, so as to not perturb performance during execution. link to common/cpu timer3.hpp

2.2 R.Q. 2 Automatic Approximation Testing

In order to automatically accept or reject approximations, we collect system-level error metrics. The data we collect is shown in section 2.2. In a real system, we would determine a threshold of acceptability from user-studies, but that is outside the scope of this project; We will continue under the assumption that *there is some* threshold.

2.2.1 Approximation Configuration Infrastructure. We created a script that runs ILLIXR over an arbitrary set of approximation configurations. The script configures ILLIXR by generating an ILLIXR config file and environment variables. link to experiment.py run illixr

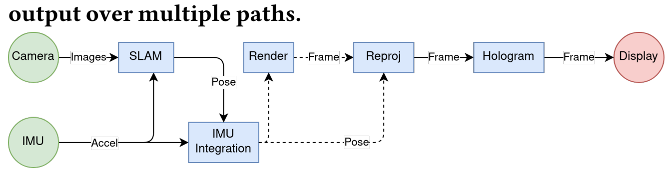
2.2.2 System-Level Metrics. In the interest of repeatability, we switched to ILLIXR's offline-mode which uses a dataset on disk instead live sensor data. We used the EuRoC dataset, which has high-quality ground-truth pose data [1].

The output of an XR system with pre-recorded sensor trace is essentially a video stream. Capturing the video-stream is non-trivial because dumping frames eagerly would perturb the computation, while deferring until the end of the computation involves buffering which has proven too memory intensive. Since ILLIXR uses OpenGL commands to draw frames, we can just capture every OpenGL command in a binary format and replay those commands offline. This is excatly what the `apitrace` tool does [2]. link to experiment.py line that computes results.frames

We generate that video-stream from the ground-truth data, and then compare the two frame-by-frame. First, we tried using Structural Similarity Index Metric (SSIM), which has been used to compare XR video streams before citation. However, this may not be the best metric to compare video streams which have variation in their pose.

With SSIM performing poorly, we also tried a hand-made metric called homography distance: For each frame of the both videos, we use SIFT to identify comon features in both frames cite SIFT and take the median difference in position (we also tried mean and max difference). link to video dist.py feature dist Like a user, it emphasizes rotations (in which every pixel moves) over translations (in which only pixels on foreground objects move). Most of the errors in SIFT correspond to valid relaxations of user acceptance. For example, SIFT will be unable to detect many features if the background is uniform (say your looking at a blue sky), but a human user, unable to lock onto any feature, would also be less able to detect small movements and would be more forgiving.

Figure 3: AR/VR DAG, showing how SLAM flows to the output over multiple paths.



2.3 R.Q. 3 Fast Automatic Approx Testing

The prior method involves running the whole system for one minute (enough time to collect a representative trace), about two minutes to replay the OpenGL trace, and ten minutes to compute the video distances. Ideally, we could use approximation auto-tuning such as ApproxHPVM [5], but this goal is still quite far away. First, we need to evaluate system-level effects of approximations at a much faster rate.

2.3.1 Estimating System-Level Error by Component-Level Proxies. The component-level metrics are much easier to get than the system-level ones. Only a small set of components have to be run, and they can be run without realtime scheduling (no need to `sleep` to get the timing right). Determining the system-level error from the component-level error analytically is too difficult. AR/VR systems have multiple paths from SLAM to the output, which makes the error *non-compositional*. For example, errors SLAM at time t in render can be corrected by SLAM at $t+\delta$ through the timewarp (in fig. 3).

Instead of an analytical model, we might consider an empirical one, using machine learning techniques. We have both binary-valued knobs and continuous knobs. It is possible that the effect of each knob is also non-compositional; turning on two approximations simultaneously could have a different impact than turning them both on separately. Therefore, a two- or three-layer fully-connected neural network will probably be able to capture finish writing

2.3.2 Component-Level Metrics. For SLAM-level metrics, we compare the poses SLAM returned to the ground-truth. However, since the ground-truth data was collected by different sensors, it is stored in a different frame-of-reference. We used the Umeyama's Method to register a correspondence from SLAM's poses to the ground-truth dataset for one for a non-approximate SLAM[6]. This provided the coordinate transformation we could use for the rest of the approximation-trials. We capture the error in position and orientation *separately*, because they have a different impact on users. Rotations are the most important movement, because even if the user only moves a few degrees, everything in the world moves by an appreciable pixel-distance; However, with translation, distant objects stay in roughly the same place. Only foreground items respond to head-translations.

3 DISCUSSION

Many of the challenges were due to nitty-gritty engineering details. For example, actually getting the transformation right took a lot of time and debugging. Using thread-local and global variables for the CPU timers was also especially challenging. There were a few bugs that we had to work-around rather than fix, such as a race condition in the graphics code. This is the regular pain of working on a large research project.

This also implies that many of the improvements we made were engineering details that we can upstream. Trying to actually use ILLIXR to do a course project has given myself (Sam speaking) more ideas for improving it than when I was thinking about improving it full-time!

4 CONCLUSION

In just one semester, we designed and implemented a system that answers **R.Q. 1** and **R.Q. 2**. We have set groundwork for **R.Q. 3**. We also propose and evaluate a novel video distance metric. Along the way, we made software-engineering improvements on ILLIXR will be incorporated into the mainline in the near future.

REFERENCES

[1] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. 2016. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research* (2016). https://doi.org/10.1177/0278364915620033 arXiv:http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html
[2] José Fonseca. [n.d.]. Apitrace. https://github.com/apitrace/apitrace.
[3] Patrick Geneva, Kevin Eckenhoff, Woosik Lee, Yulin Yang, and Guoquan Huang. 2020. OpenVINS: A Research Platform for Visual-Inertial Estimation. In *Proc. of the IEEE International Conference on Robotics and Automation*. Paris, France. https://github.com/rpng/open_vins
[4] Muhammad Huzaifa, Rishi Desai, Xutao Jiang, Joseph Ravichandran, Finn Sinclair, and Sarita V. Adve. 2020. Exploring Extended Reality with ILLIXR: A New Playground for Architecture Research. arXiv:cs.DC/2004.04643
[5] Hashim Sharif, Prakalp Srivastava, Muhammad Huzaifa, Maria Kotsifakou, Keyur Joshi, Yasmin Sarita, Nathan Zhao, Vikram S. Adve, Sasa Misailovic, and Sarita Adve. 2019. ApproxHPVM: A Portable Compiler IR for Accuracy-Aware Optimizations. *Proc. ACM Program. Lang.* 3, OOPSLA, Article 186 (Oct. 2019), 30 pages. https://doi.org/10.1145/3360612
[6] S. Umeyama. 1991. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13, 4 (1991), 376–380. https://doi.org/10.1109/34.88573

A PRE-EXISTING AND CONCURRENT WORK

This work was *not* done for this project:

- OpenVINS already had tunable parameters.
- ILLIXR already had timing infrastructure, but it was insufficient for our purposes, because it could not be adapted to time *inside* OpenVINS.
- The ILLIXR research group had already considered using SSIM, but their implementation was not ready at the time we needed one, so we implemented it ourselves. We could not upstream our implementation because theirs is able to handle a wider class of applications.
- The ILLIXR research group helped me find and use Umayama's Method for aligning the pose.

This work was for this project:

- We designed and implemented a the infrastructure for configuring approximations across all of ILLIXR (not just OpenVINS).
- We rewrote the timing infrastructure (from scratch).
- We implemented component-level metrics.
- We implemented the coordinate transformation code.
- We implemented our own version of system-level metric (apitrace, SSIM, homography-distance).
- We designed and implemented our own video metric (homography-distance).
- We explored the idea of a estimating system-error by component-level proxy specifically for this projet.