KinectFusion on Steroids

Swarnendu Biswas
University of Texas at Austin, USA
sbiswas@ices.utexas.edu

Yan Pei
University of Texas at Austin, USA
ypei@cs.utexas.edu

Donald S. Fussell
University of Texas at Austin, USA
fussell@cs.utexas.edu

Keshav Pingali
University of Texas at Austin, USA
pingali@cs.utexas.edu

Abstract

We study opportunities for introducing controlled approximation in SLAM algorithms such as KinectFusion. We show that our approach produces substantial savings in run time, with acceptable tradeoff in accuracy.

Keywords approximate computing, SLAM, KinectFusion, control

1 Introduction

Simultaneous Localization and Mapping (SLAM) algorithms are used to control the motion of robots, drones, and UAVs. The key steps are (i) capture details of the surrounding environment, (ii) construct a map of the environment, and (iii) estimate the position and orientation (i.e., localization) of the reference object in the environment [3, 5].

Problem. SLAM programs use computationally intensive kernels like stencil computations and filters, which involve a lot of floating-point computations [2, 6]. Some SLAM algorithms use non-linear models for motion capture, which are also computationally expensive. These computational costs are an impediment to the widespread adoption of SLAM.

Approximating SLAM. In this work, we explore the use of approximation in SLAM algorithms to trade off accuracy for running time. SLAM algorithms expose algorithmic *knobs* that can be tuned for this purpose [4, 6], but policies for controlling these knobs are not well understood.

In our study, we explore online controllers for the *KinectFusion* algorithm [3, 5], as implemented in the SLAMBench infrastructure [2, 4, 6], which is a convenient platform for studying tradeoffs between accuracy and running time for this algorithm. KinectFusion takes depth values from a *Kinect* sensor and processes them using parallel algorithms to perform mapping and localization in room-size indoor environments [3, 5]. The quality of the output is usually estimated using the *average trajectory error (ATE)* metric, which is a measure of the average deviation of the trajectory in the approximate computation from the actual trajectory, which is the ground truth. SLAMBench outputs an estimate of the running time of the algorithm [4].

The space of knob settings for this application has roughly two million points [2], so exhaustive profiling of the knob space is infeasible. Therefore, we ranked the knobs by their importance in influencing quality and running time, as is done in the Pupil system [7]. The three most important knobs are compute size ratio (csr), iterative closest point threshold (icp), and integration rate (ir). In this work, we control only these three knobs for approximating KinectFusion (see Section 2). Our results, described in Section 3, show that it is possible to get significant performance improvements with tolerable loss in accuracy.

Algorithm 1 FRAME_DIFF heuristic applied on each frame.

```
curr_knobs ← current KinectFusion config
if curr_frame ≤ NUM_BOOTSTRAP_FRAMES then
    continue
end if
curr_frame_diff ← curr_frame - prev_frame
frame_hist ← frame_hist ∪ curr_frame_diff
avg_diff ← (∑frame_hist) / HIST_LEN
max_diff ← max(frame_hist)
threshold ← avg_diff + (max_diff - avg_diff) / 2
increasing ← sorted_increasing(frame_hist)
sample_diff ← computeSampleDiff()
if curr_frame_diff > threshold or increasing then
    curr_knobs ← curr_knobs - 1                    ▷ Increase precision
else if sample_diff ≤ SAMPLE_DIFF_THRES then
                          ▷ Set knobs to the highest precision settings
    curr_knobs ← curr_knobs[0]
else                                               ▷ Decrease precision
    curr_knobs ← curr_knobs + 1
end if
```

2 Controlling KinectFusion

Since SLAM is not useful if it cannot map the 3D environment or localize the camera with reasonable accuracy, we consider two separate constraints on permissible approximations. The first one is more strict.

1. **Constraint 1**: Approximation should not induce loss of tracking of frames earlier than the default setting of KinectFusion.
2. **Constraint 2**: The ATE after an approximate KinectFusion computation should be less than n cm.

2.1 Insights

Insight 1. *More precise computation may be needed if the camera is moving rapidly.* Our control system estimates the rate of movement using the difference between depth values

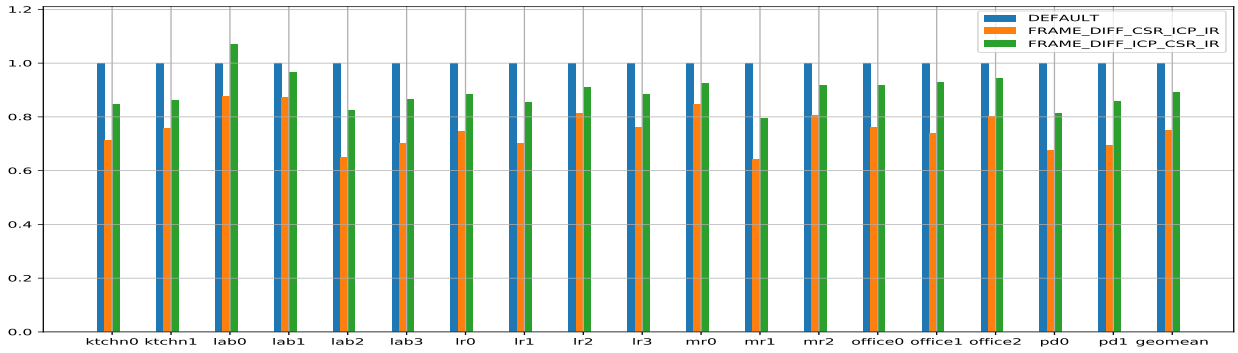

Figure 1. Normalized run time of KinectFusion with heuristics that do not lose tracking before the default algorithm.

across successive frames, ramping up the precision as this estimate gets larger.

Insight 2. *Tracking of frames is easier if the scene contains features like objects or furniture, and is more difficult if the scene is relatively smooth such as a wall or a door.* Our control system samples random portions of the input depth frame, and computes the inter-sample deviation of the depth values. A low inter-sample deviation potentially indicates a potential smooth surface, so the precision of the computation is increased (if needed, advanced computer vision algorithms can be used to detect smooth surfaces).

2.2 Heuristics

Based on these insights, we have devised a heuristic controller that we call FRAME_DIFF. Algorithm 1 shows the pseudo-code. A few initial frames are used for bootstrapping, and no online control is applied during this time. The controller maintains a history of the inter-frame difference (i.e., each incoming depth frame), using a sliding window mechanism to compute a running *threshold* information, which is used to model the variations of the trajectory. A simpler alternative is to use a constant threshold, but our experience shows that a constant threshold does not work well for different types of input trajectories.

By changing the order in which knobs are adjusted, we get several variations of FRAME_DIFF. We studied two variations: FRAME_DIFF_CSR_ICP_IR and FRAME_DIFF_ICP_CSR_IR. The two policies differ in the order in which the knobs are varied.

3 Experimental Results

For our evaluation, we extended the C++ implementation of KinectFusion in the open-source SLAMBench [4] infrastructure[1] with our online control algorithm. We ran our experiments on a quad-core Dell PowerEdge 1950 Xeon X5355 system, with 2.66GHz clock frequency and 32GB primary memory. We use four living room trajectories from the ICL-NUIM dataset [1] as benchmarks. In addition, we used a first generation Kinect to collect fourteen trajectories in an indoor environment. For the trajectories we collected, we use the trajectory computed by the most accurate setting of KinectFusion to simulate the ground truth.

Given each constraint, we evaluate two variants of the FRAME_DIFF heuristic on the input trajectories, and measure the run time and the ATE. Figure 1 shows the results of approximating KinectFusion with Constraint 1 that does not allow loss of tracking before the default setting of KinectFusion, DEFAULT. The bars in Figure 1 are the average of three trials and are normalized to DEFAULT. For Constraint 1, FRAME_DIFF_CSR_ICP_IR reduces the average run time by 26%, while FRAME_DIFF_ICP_CSR_IR reduces the overhead by 13%. Approximating csr first performs better since it has the most impact on run time. Both the configurations meet Constraint 1 and do not induce any loss of tracking before DEFAULT, and the ATE is less than or equal to 5 cm for all but two trajectories, lr3[2] and mr1. Using the weaker Constraint 2, the two heuristics, FRAME_DIFF_CSR_ICP_IR and FRAME_DIFF_ICP_CSR_IR reduce the run time by 35% and 30% respectively (figure omitted for space). As before, controlled approximation of KinectFusion does not induce an ATE of more than 5 cm for all but two trajectories. In this context, we observe that the overhead of the heuristics at every execution step is insignificant (~1ms compared to ~1-2s for processing one frame in KinectFusion).

4 Conclusion

In this study, we show simple heuristics that are effective in reducing the run time of KinectFusion, with acceptable tradeoffs on the localization accuracy. For future, we plan to extend the control system to OpenCL/CUDA platforms, implement PAPI-based energy estimations, and evaluate the benefits of approximation on an ODROID XU4 platform.

[1] https://github.com/pamela-project/slambench

[2] For lr3, DEFAULT itself incurs an ATE of 11.5 cm.

References

[1] [n. d.]. ICL-NUIM RGB-D Benchmark Dataset. http://www.doc.ic.ac.uk/~ahanda/VaFRIC/iclnuim.html. ([n. d.]). Accessed: 2017-12-09.

[2] Bruno Bodin, Luigi Nardi, M. Zeeshan Zia, Harry Wagstaff, Govind Sreekar Shenoy, Murali Emani, John Mawer, Christos Kotselidis, Andy Nisbet, Mikel Lujan, Björn Franke, Paul H.J. Kelly, and Michael O'Boyle. 2016. Integrating Algorithmic Parameters into Benchmarking and Design Space Exploration in 3D Scene Understanding. In *Proceedings of the 2016 International Conference on Parallel Architectures and Compilation (PACT '16)*. ACM, New York, NY, USA, 57–69. https://doi.org/10.1145/2967938.2967963

[3] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. 559–568.

[4] 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 International Conference on Robotics and Automation (ICRA)*. arXiv:1410.2167.

[5] 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 *IEEE ISMAR*. IEEE.

[6] Oscar Palomar, Andy Nisbet, John Mawer, Graham Riley, and Mikel Lujan. 2017. Reduced precision applicability and trade-offs for SLAM algorithms. In *Third Workshop on Approximate Computing (WACAS)*.

[7] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '16)*. ACM, New York, NY, USA, 545–559. https://doi.org/10.1145/2872362.2872375