Sensor Modeling and Benchmarking — A Platform for Sensor and Computer Vision Algorithm Co-Optimization

Andrew Berkovich, Chiao Liu
Facebook Reality Labs
andrew.berkovich@oculus.com

Abstract

We predict that applications in AR/VR devices [1] and intelligence devices will lead to the emergence of a new class of image sensors — machine perception CIS (MPCIS). This new class of sensors will produce images and videos optimized primarily for machine vision applications, not human consumption. Unlike human perception CIS, where the ultimate criterion is visual image quality [2], there is no existing criterion to judge MPCIS sensor performance. In this paper, we present a full stack sensor modeling and benchmarking pipeline (from sensors to algorithms) that could serve as the platform for performance evaluation. We illustrate how sensor modeling and benchmarking help us understand complex system trade-offs and dependencies between sensor and algorithm performance, specifically for simultaneous localization and mapping (SLAM).

I. Introduction

Unlike typical benchmarking datasets (e.g. KITTI [3]) that support software benchmarking and optimization, sensor modeling and benchmarking encompasses the full system including data generation, sensor operation, optics, computer vision algorithms, and statistical analysis (illustrated in Fig. 1). It enables system-level optimization from silicon and sensors to higher-level, application-specific performance. This system-level analysis is particularly critical in the case of on-sensor compute and enables a new form of hardware/software co-design focused specifically on MPCIS.

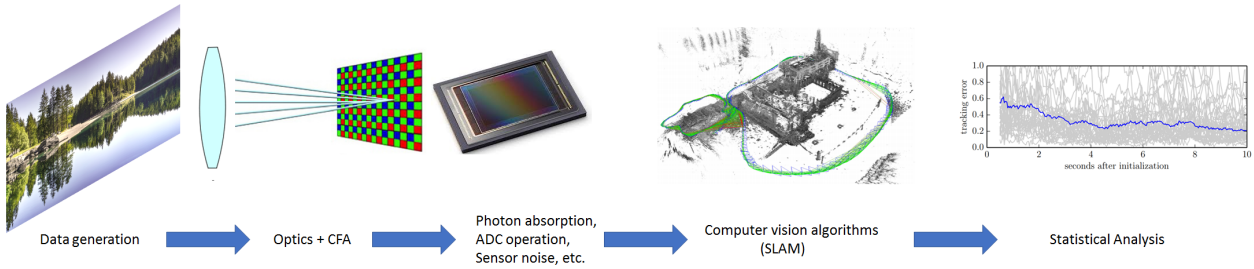


Fig. 1: Cartoon illustration of sensor benchmarking pipeline

II. SLAM - A Case Study on Sensor/Algorithm Co-Optimization

Applications in AR/VR require the ability that camera systems can map their environment and estimate their location within that map. Recent works on Simultaneous Localization and Mapping (SLAM) have demonstrated real-time solutions to this problem [4], [5]. At a high-level these algorithms compute and track image features (e.g. corners) across frames. Camera pose and scene geometry are estimated from feature correspondences across frames.

The robustness and performance of a SLAM system is highly dependent on the number and quality of features located within each frame. The image sensors used in a SLAM system can dramatically impact tracking performance. In real-world, day/night, indoor/outdoor use cases, the uncontrolled lighting conditions make feature tracking more challenging and necessitates image sensors to have both excellent low light sensitivity and high dynamic range (HDR). Furthermore, SLAM algorithms are highly sensitive to geometric noise and image distortion including both motion blur and rolling shutter distortion. HDR schemes such as multi-exposure and interleaved exposureintroduce similar types of image distortion. Thus single exposure, global shutter operation is strongly preferred to multi-capture and rolling shutter operation.

Nearly all SLAM algorithms are built upon probabilistic models that aim to predict camera motion and world model from noisy sensor measurements. Due to stochastic processes (e.g. sensor noise) a SLAM system will output different results even when traversing identical trajectories. Furthermore, tracking quality of a given SLAM algorithm is highly data dependent. Among several other factors, the ability to establish correspondences across multiple frames depends

heavily on the environment, camera motion, and lighting conditions. Thus, traditional image quality metrics [2] (e.g. signal-to-noise ratio, photo-response non-uniformity, dynamic range) are not sufficient to quantify the performance of computer vision applications such as SLAM as they do not capture the probabilistic and data-dependent nature of these algorithms.

We need an end-to-end system model to enable data-driven optimization across the entire hardware-software stack. This optimization includes many parameters related to image sensors such as field-of-view, spatial resolution, noise, dynamic range, spectral range, etc. Furthermore, existing AR/VR systems and autonomous vehicles integrate multiple cameras and inertial measurement units (IMUs). Thus, this system-level optimization also includes sensor placement and interactions. This system model requires large input datasets that cover both combinations of system design choices along with variations in environment, camera motion, and lighting conditions.

III. End-to-End System Modeling

A. Input Dataset Generation

Synthetic datasets offer two major advantages over datasets captured with physical rigs and off-the-shelf (OTS) sensors. First, synthetic data is not limited by performance of OTS sensors (e.g. cannot simulate ultra high dynamic range sensors). Second, generation of synthetic data is cheaper, faster, and more flexible than data collection using physical hardware. Synthetic datasets are available from academic groups (e.g. ICL-NUIM dataset [6]) but are generally limited in size (<30 minutes) and scene diversity (e.g. indoor only). Recently, NVIDIA has begun offering tools [7] for developing and testing algorithms geared towards automotive applications. We generate data using synthetic data collection rigs, containing the number, position, and orientation of all sensors. User-defined trajectories control how data collection rigs move through the environment, including variations in camera motion (e.g. running vs walking). These trajectories provide groundtruth (GT) data for SLAM and can be used to model IMU sensors which is beyond the scope of this work. Similarly, graphics engines can provide GT depth and photometric information for every pixel in an image.

B. Sensor Modeling

Input to the sensor model is number of photons (photons/exposure) incident on a given pixel and its output is either an analog voltage or a digital number (DN). For simplicity we begin by considering the simple case of a single 3T pixel connected directly to an ADC. The core functions of the pixel include light detection (photon-to-electron conversion), charge sensing (charge-to-voltage conversion), and quantization (voltage-to-DN conversion).

Detection is modeled as a stochastic process that maps input N_{ph} (number of photons incident on pixel $P_{i,j}$) to output N_e (number of electrons generated in the photodiode). N_e is generated by a Poisson random number generator (RNG) with rate parameter $\lambda = N_{ph} \cdot QE \cdot FF$ where QE is quantum efficiency and FF is pixel fill-factor. QE can be modeled differently for each color band rendered by the graphics engine and can include transmission properties of elements in a color filter array (CFA). The use of a Poisson RNG adds shot noise to the sensor model.

The charge sensing block models conversion gain (g) of the pixel (including source follower) and simulates temporal noise sources such as read noise and kT/C noise. The input to this block is N_e (output from detection block) and its output is pixel voltage $v_{pix} = v_{reset} - N_e \cdot g + v_{noise}$. Reset voltage of the FD node (v_{reset}) as seen at the output of the SF is non-stochastic while v_{noise} is generated by a Gaussian RNG with zero mean and standard deviation $\sigma = \sqrt{v_{read,rms}^2 + v_{kTC,rms}^2}$ to simulate read noise and kT/C noise.

The quantization block takes as input v_{pix} from the sensing block and outputs DN. For simplicity we assume a single-slope ADC and generate two control signals: analog ramp (v_{ramp}) and digital ramp (DN_{ramp}). For each value of DN_{ramp}, we compare v_{ramp} against v_{pix} and output the value of DN_{ramp} when the v_{ramp} and v_{pix} signals cross. This process introduces quantization noise to the sensor model. We can simulate non-linear quantization by changing v_{ramp} and DN_{ramp} to non-linear functions.

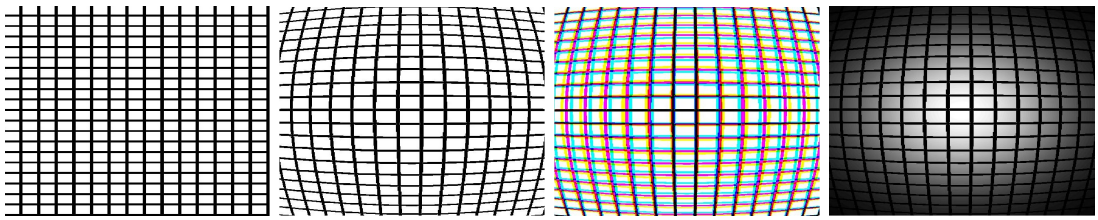
As we move from a single pixel to an array we can incorporate spatial noise ranging from pixel-level variations (e.g. non-uniform conversion gain, SF offsets) to column-level variations (e.g. non-uniform column-level ADCs) and even die-to-die variations. For example, QE can be a global parameter to simulate a monochrome sensor. Alternatively, it can be a pixel-level parameter ($QE_{i,j}$) to simulate a CFA where all pixels have a unique spectral filter with varying transmission properties. We can also treat v_{reset} as a pixel-level parameter that varies across the array to simulate

fixed pattern noise (FPN). Similarly, we can simulate input-offset for parallel-ADC architectures (e.g. column-level ADCs) by modifying v_{pix} to include an ADC offset term v_{off} which can vary for each simulated ADC.

Using this framework, we can support a wide range of sensor behaviors and architectures. For example, we can include photoresponse non-uniformity (PRNU) by applying a compressive non-linearity to v_{pix}. HDR schemes, such as multi-exposure, dual conversion gain, can also be modeled by either combining subsequent frames or by incorporating multiple detection and sensing blocks within a single pixel. Overall, this parametrized sensor model allows sensor parameters, critical to sensor optimization and design, such as conversion gain, bit-depth, and full-well capacity to be included in system-level benchmarking and optimization.

C. Optical Stack Modeling

Design of image sensors and their optics are tightly linked. To fully understand tradeoffs between system performance and sensor design parameters, such as pixel size, it is critical that the optical stack be modeled alongside the sensor. The primary goal of lens modeling is to mimic the mapping functions of lenses that produce image distortion. We separate the lens model into two main components: image generation and sampling. First, distortion-free images (Fig. 2a) are generated by the graphics engine—images can have arbitrary FoV and resolution. Second, images are down-sampled based on the mapping function of the lens as illustrated in Fig. 2b. The output of this sampling function is an image with FoV and resolution matched to the simulated lens model and image sensor. The same approach can be applied separately to each color channel rendered by the graphics engine to approximate chromatic aberration (Fig. 2c). Effects like lens roll-off (Fig. 2d), vignetting, and optical cross-talk can be modeled by applying simple convolutional filters to the distorted images. Distortion in Fig. 2 is exaggerated to illustrate lens model effects.



(a) (b) (c) (d)

Fig. 2: (a) Distortion-free input image to lens model (b) Simulated lens distortion (c) Simulated chromatic abberation, separate mapping functions are used for R,G, and B color channels (d) Simulated lens roll-off

The data produced by such modeling provide unique opportunities. For example, it offers a tool to develop and test de-mosaicing algorithms [8]. Such work could also jointly optimize for novel CFA configurations. Another example is optimization of FoV. To correctly understand trade-offs between FoV and tracking performance, it is essential that effects like lens distortion and roll-off be correctly modeled and fed into a SLAM algorithm.

D. Full Image Rendering Pipeline

The full image rendering pipeline has been described in terms of three distinct functional block. The first is synthetic image generation. In this paper we use Unreal Engine 4 (UE4) as our graphics engine. The output from this block represents photons generated in the physical world prior to impinging on an optical sensor. The output from the graphics engine feeds into the second functional block: optical stack model. This block maps photons from the real-world to the focal plane of an image sensor and introduces optical effects like cross-talk. The output from this block represents photon flux as it is observed at the pixel-level of an image sensor. The final block is the sensor model. It is critical that the optical stack be modeled before the lens as sensors can display non-linearities in photon response. This is critical when simulating effects like lens roll-off which are far simpler to implement on linear images than on non-linear images. The output of the rendering pipeline is images that represent images captured by the simulated sensor. Sample image are shown in Fig. 3 for varying ADC resolution (or pixel bit-depth). Effects like motion blur can be simulated by rendering and merging sub-frames.

IV. Full-Stack Benchmarking

We demonstrate examples of end-to-end system benchmarking using a VGA-resolution sensor with a simple 6T pixel. We model a monochrome sensor by merging (weighted sum of) the separate RGB color channels rendered in UE4 based on average QE of the sensor for each band and simulate a frame rate of 30fps.

One optimization we explore is pixel bit-depth. While OTS offer >12-bit resolution, it's unclear whether such sensors are over-engineered for applications like SLAM. We generate synthetic images with varying ADC resolution (Fig. 3) and benchmark SLAM performance. As we decrease bit-depth, position jitter (high frequency noise in position estimation) increases from 0.8mm for 8-bit resolution to 0.9mm and 5.8mm for 4-bit and 2-bit resolution respectively. Similarly, we see an increase in prediction error (as described in [4]) from roughly 22mm to 3mm for 4-bit and 2-bit resolution. Across both performance metrics we see an improvement in tracking performance metrics of roughly 84% moving from 2-bit resolution to 4-bit and 13% from 4-bit to 8-bit.

(a) 2-bit (b) 8-bit (c) SLAM trajectory

Fig. 3: (a,b) Output from image rendering pipeline for a sensor with 2-bit and 8-bit ADC resolution. (c) Visualizes the outputs of SLAM including the estimated trajectory (green spline) and tracked features (blue and green dots).

We present a similar evaluation of read noise on SLAM performance. Read noise is particularly important in low-light scenarios and impacts pixel size and power consumption. We generate synthetic images with varying levels or read noise (ranging from $v_{read,rms} = 100\mu V$ to $300\mu V$) and benchmark SLAM performance with a simulated 6T pixel architecture and conversion gain of $g = 10\mu V/e^{-}$. Increasing read noise from 100μV to 300μV reduces the median number of features tracked across all frames by roughly 8%. However, it has neglible effect on pose estimation error. We note that the synthetic data used for this benchmarking simulated a well-lit, day-time environment.

V. Discussion and Limitations

We argue that performance metrics such as pose error and jitter are far more power tools in system-level optimization than traditional sensor performance metrics as they directly measure system performance. Conclusions of benchmarking data are heavily dependent on applications and use cases. Once system specifications are defined (e.g. pose error <10mm for 99% of trajectories), benchmarking provides a low-cost, flexible tool to rapidly iterate on system designs and estimate whether or not a particular solution meets system requirements. Ultimately, sensor modeling and benchmarking is only as good as the data fed into it. Thus, photorealistic rendering and accurate representations of real-world scenes are critical to achieving accurate benchmarking results. Sensor modeling relies on simulated sensor performance, unless similar sensors have already been developed. This makes accurate simulations and modeling of sensor behavior and performance a key aspects of this work. As such, sensor modeling and benchmarking truly spans the entire system stack, from pixel engineers and sensor designers to computer vision experts and system engineers.

References

[1] C. Liu *et al.*, "Sensors for Future VR Applications," in *Proc. of Intl. Image Sensor Workshop (IISW)*, 2017.

[2] J. Alakarhu, "Image sensors and Image Quality in Mobile Phones," in *Proc. of Intl. Image Sensor Workshop (IISW)*, 2007.

[3] A. Geiger *et al.*, "Vision meets Robotics: The KITTI Dataset," in *Intl. Journal of Robotics Research (IJRR)*, 2013.

[4] J. Engel, "Large-Scale Direct SLAM and 3D Reconstruction in Real-Time," PhD Thesis, 2017, Technical University Munich.

[5] R. A. Newcombe *et al.*, "DTAM: Dense tracking and mapping in real-time," in *Intl. Conf. on Computer Vision*, Nov 2011, pp. 2320–2327.

[6] A. Handa *et al.*, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *IEEE ICRA*, May 2014.

[7] "NVIDIA DRIVE Platform," https://www.nvidia.com/en-us/self-driving-cars/drive-platform/, accessed: 2019-04-15.

[8] N. Syu *et al.*, "Learning deep convolutional networks for demosaicing," *CoRR*, vol. abs/1802.03769, 2018. http://arxiv.org/abs/1802.03769