An Affordable Open-Source Stereo Camera for Real-Time Point Cloud Generation

ARIHANT JAIN, University of California San Diego, USA
KEYUSH ATTARDE, University of California San Diego, USA
SUMUKH MURTHY, University of California San Diego, USA
RISHI GUPTA, University of California San Diego, USA

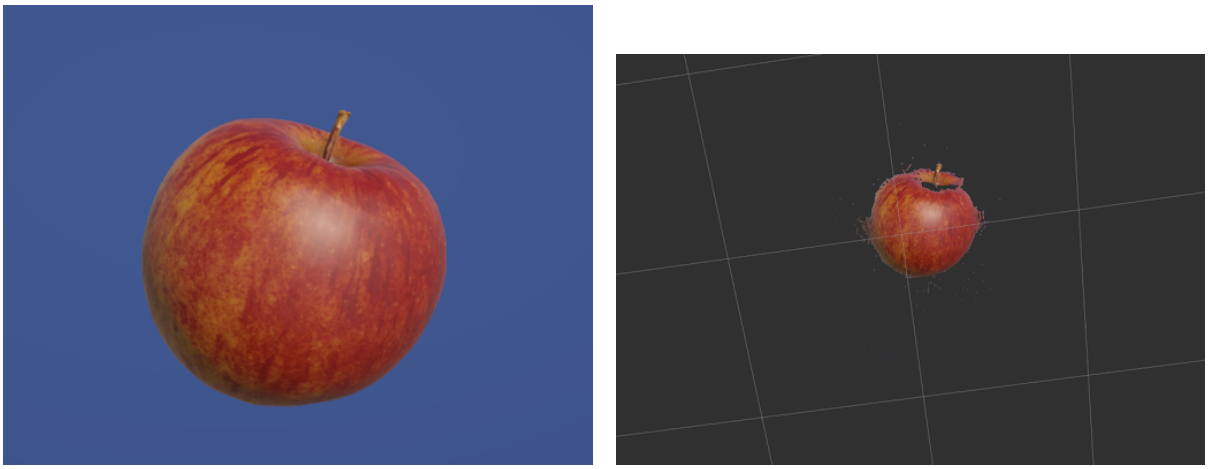

Fig. 1. Demonstration of the point cloud generation pipeline, a comparison of the raw left camera image and the corresponding generated point cloud.

1 Abstract

Commercial stereo cameras are expensive, with prices exceeding $500. In this study, we outline an original design for a stereo camera that costs $ 360 and is comparable to commercial products. The design is based around two hardware-synced iRayple cameras fixed rigidly to a 3D-printed box, and controlled by a Rubik Pi. The Pi receives input from the cameras and computes a colored 3D point cloud from the calculated disparity data. The camera fps of 100 at a resolution of 1280 x 1024 exceeds industry benchmarks measured against two commercial stereo cameras, though point cloud fidelity remains an issue. We anticipate that this low-cost design can spur further competition in the stereo camera market and improve public accessibility to Computer Vision.

2 Introduction

Depth perception is a foundational capability for a wide range of modern computing applications, including autonomous robotics, augmented and virtual reality, 3D scene reconstruction. A conventional monocular camera, however, captures only a flat two-dimensional projection of the world and cannot directly recover the distance to objects in a scene. Stereo vision addresses this limitation by mimicking human binocular vision: two cameras placed a fixed distance apart observe the same scene from slightly different viewpoints, and the disparity between

Authors' Contact Information: Arihant Jain, University of California San Diego, La Jolla, California, USA; Keyush Attarde, University of California San Diego, La Jolla, California, USA; Sumukh Murthy, University of California San Diego, La Jolla, California, USA; Rishi Gupta, University of California San Diego, La Jolla, California, USA.

corresponding pixels in the two images can be used to triangulate depth, the classical stereo correspondence problem [11]. The resulting depth information can be combined with color data to produce a colored 3D point cloud. A format in which each point encodes both a physical location in space and an RGB value, providing a rich volumetric representation of the environment.

Despite the maturity of stereo vision techniques, access to high-quality depth-sensing hardware remains a practical barrier. Commercial stereo cameras such as the Stereolabs ZED 2, and depth sensors such as solid-state LiDAR units like the Manifold Tech Odin1, routinely cost above $500. These price points place capable depth sensing out of reach for many students, researchers, hobbyists, and developers of low-cost robotic platforms. Moreover, commercial offerings are typically closed systems: their internal processing pipelines, synchronization mechanisms, and calibration procedures are proprietary, limiting both their educational value and their flexibility for customization.

In this project, we designed, built, and evaluated a complete open stereo camera platform with a total hardware cost of approximately $360. The system consists of two industrial iRayple cameras mounted rigidly in a custom 3D-printed enclosure and triggered simultaneously by a hardware synchronization circuit, ensuring that both image streams are captured within microseconds of one another. All computation is performed onboard a Rubik Pi, an embedded ARM platform. The software stack is organized as a set of ROS2 nodes and developed through a dockerized cross-compilation workflow, allowing development on standard x86 laptops with deployment to the ARM64 target. The processing pipeline rectifies the calibrated stereo image pair [13], computes a dense disparity map using Semi-Global Block Matching [5] with Weighted Least Squares smoothing, and reprojects matched pixels into 3D to form a colored point cloud in real time.

Beyond the core depth pipeline, the platform incorporates several extensions that broaden its utility. An onboard IMU enables visual-inertial odometry (VIO), allowing the camera's pose to be tracked as it moves so that point clouds from successive frames can be transformed into a common world frame and accumulated into larger scene reconstructions. A YOLO-based semantic segmentation module [9] provides an alternative point cloud coloring mode in which points are grouped and colored by detected object, supporting object-level understanding of the scene. Through careful optimization, the system captures and processes synchronized stereo imagery at approximately 100 frames per second at a resolution of 1280 × 1024, a frame rate that exceeds the commercial devices we benchmarked against at that resolution. Though, as we discuss later in this paper, the geometric fidelity of the resulting point clouds still leaves room for improvement relative to commercial alternatives.

The contributions of this work are as follows:

- **An open, low-cost stereo camera design.** We present a complete hardware design, including a custom synchronization trigger circuit, a rigid 3D-printed dual-camera enclosure, and a full intrinsic and extrinsic calibration procedure, built for approximately $360, a substantial reduction relative to comparable commercial devices.
- **A real-time onboard point cloud pipeline.** We implement a dense stereo depth pipeline (rectification, SGBM disparity estimation with WLS smoothing, and RGB reprojection into 3D) that runs entirely on an embedded Rubik Pi, structured as ROS2 nodes and supported by a reproducible Dockerized development and deployment workflow.
- **Motion tracking and scene accumulation.** We integrate an IMU and implement visual-inertial odometry, enabling the system to track its own motion and accumulate point clouds across frames into coherent reconstructions of larger environments.
- **Semantic point cloud augmentation.** We extend the pipeline with YOLO-based semantic segmentation, producing an alternative object-level coloring of the point cloud that can be toggled with a single parameter.

- **Benchmarking against commercial systems.** We evaluate our platform qualitatively and quantitatively against the Stereolabs ZED 2 stereo camera and the Manifold Tech Odin1 solid-state LiDAR, characterizing where our design exceeds commercial performance (frame rate, cost) and where it falls short (point cloud fidelity).

The remainder of this paper is organized as follows. Section 3 surveys related work in stereo vision, embedded depth sensing, and point cloud generation. Section 4 presents the technical material, covering the hardware design, calibration, the point cloud pipeline, VIO, semantic segmentation, and benchmarking methodology and results. Section 5 reviews our project milestones, including revisions made over the quarter and a discussion of objectives that were deferred. Section 6 concludes with a summary of our findings and directions for future work.

3 Related Works

We organize prior work into four areas: classical stereo correspondence, learning-based depth estimation, calibration and feature matching, and the platforms on which this research runs. Across all four, a consistent theme emerges: the algorithms are mature and open source, while the stereo camera hardware itself has been left to commercial vendors.

3.1 Classical Stereo Correspondence

The central problem in stereo vision is correspondence, matching pixels between the left and right images to estimate depth. Scharstein and Szeliski [11] formalized the field with a taxonomy decomposing stereo algorithms into matching cost, aggregation, optimization, and refinement stages, along with the Middlebury benchmark that made quantitative comparison standard practice. Among classical methods, Semi-Global Matching (SGM) [5] has proven the most durable: it approximates global smoothness optimization with pathwise aggregation along multiple scanlines, achieving near-global accuracy at cost linear disparity range. This efficiency is why SGM variants remain the default in stereo systems, including ours. Follow-up work [6] showed that matching costs such as Census and mutual information tolerate radiometric differences between cameras far better than raw intensity. As such, we took care to synchronize our cameras at the hardware level and use a matching function that is resilient to slight deviations in pixel intensity.

3.2 Learning-Based Stereo Matching

Recent work replaces hand-designed pipeline components with learned ones. Žbontar and LeCun [12] trained a convolutional network to compute matching costs, feeding them into an otherwise conventional SGM-style pipeline. Later architectures moved to end-to-end learning: GC-Net [7] regresses disparity directly from a 4D cost volume, and PSMNet [3] adds pyramid pooling for global context, ranking among the top KITTI methods at publication. These networks define the state of the art in accuracy but assume desktop GPU resources. We treat them as workloads a research platform should support rather than algorithms to reimplement, which is why our camera exposes raw, calibrated, synchronized image pairs suitable for both classical and learned pipelines with a SGM based pipeline for testing right out of the box.

3.3 Calibration and Feature Matching

Accurate depth depends on knowing the geometry of the camera pair. Zhang's planar technique [13] made calibration practical: observing a checkerboard in a few orientations recovers intrinsics and lens distortion without the precision-machined targets earlier methods required. Many systems since, including ours, calibrates this way. A related line of work addresses sparse correspondence through local features. SIFT [8] established the detect-describe-match paradigm, and SURF [1] accelerated it with integral images, but both were historically

patent-encumbered and too slow for embedded processors. ORB [10] solved both problems with a fast, license-free binary descriptor. Our pipeline uses it for feature matching across the stereo pair.

3.4 Platforms, Benchmarks, and the Open-Source Gap

Running these algorithms requires software infrastructure and a physical camera. The software side is well served: OpenCV [2] provides open implementations of Zhang calibration, SGM, and ORB, while real-time detectors such as YOLO [9] provide the infrastructure for perception based tasks such as the image segmentation we implemented. The hardware side is different. KITTI [4], the most influential stereo benchmark, was collected with a custom vehicle rig of industrial cameras that no other group can reproduce without comparable cost. Researchers needing live stereo data instead turn to commercial depth cameras, which are closed designs: fixed baselines, unmodifiable optics, and proprietary on-board processing. The result is an asymmetry. Every algorithmic component of stereo vision is open and modifiable, while the camera itself is a black box. Our project addresses this gap by developing an affordable, open-source stereo camera as a research platform, with open hardware, raw synchronized image access, and a software stack built on the components surveyed above.

4 Technical Material

This section describes the design and implementation of our stereo camera platform: the hardware and synchronization circuit, calibration and rectification, the ROS2 software architecture, the point cloud pipeline, visual-inertial odometry, and semantic segmentation, closing with our benchmarking results against commercial devices.

4.1 Hardware Design

The platform is built around two industrial iRayple cameras mounted rigidly inside a custom 3D-printed enclosure that also houses the Rubik Pi, an inertial measurement unit (IMU), and the synchronization circuit, fixing the cameras at a known, constant relative position. The rigid mount is essential; any drift in the relative pose of the two cameras would invalidate the extrinsic calibration and corrupt depth estimates.

Reliable stereo depth requires that both cameras expose at the same instant; otherwise, motion in the scene or of the camera introduces inconsistencies between the two views that manifest as depth error. We enforce this with a hardware synchronization circuit rather than software timing. A general-purpose I/O (GPIO) pin on the Rubik Pi drives a shared trigger line distributed to both cameras through an NPN transistor. When the pin is driven high, the transistor pulls its output low, completing the trigger circuit on the camera side and causing both cameras to capture a frame. Because the cameras share a single trigger edge, their exposures begin within microseconds of one another, satisfying the tight tolerance that stereo matching requires.

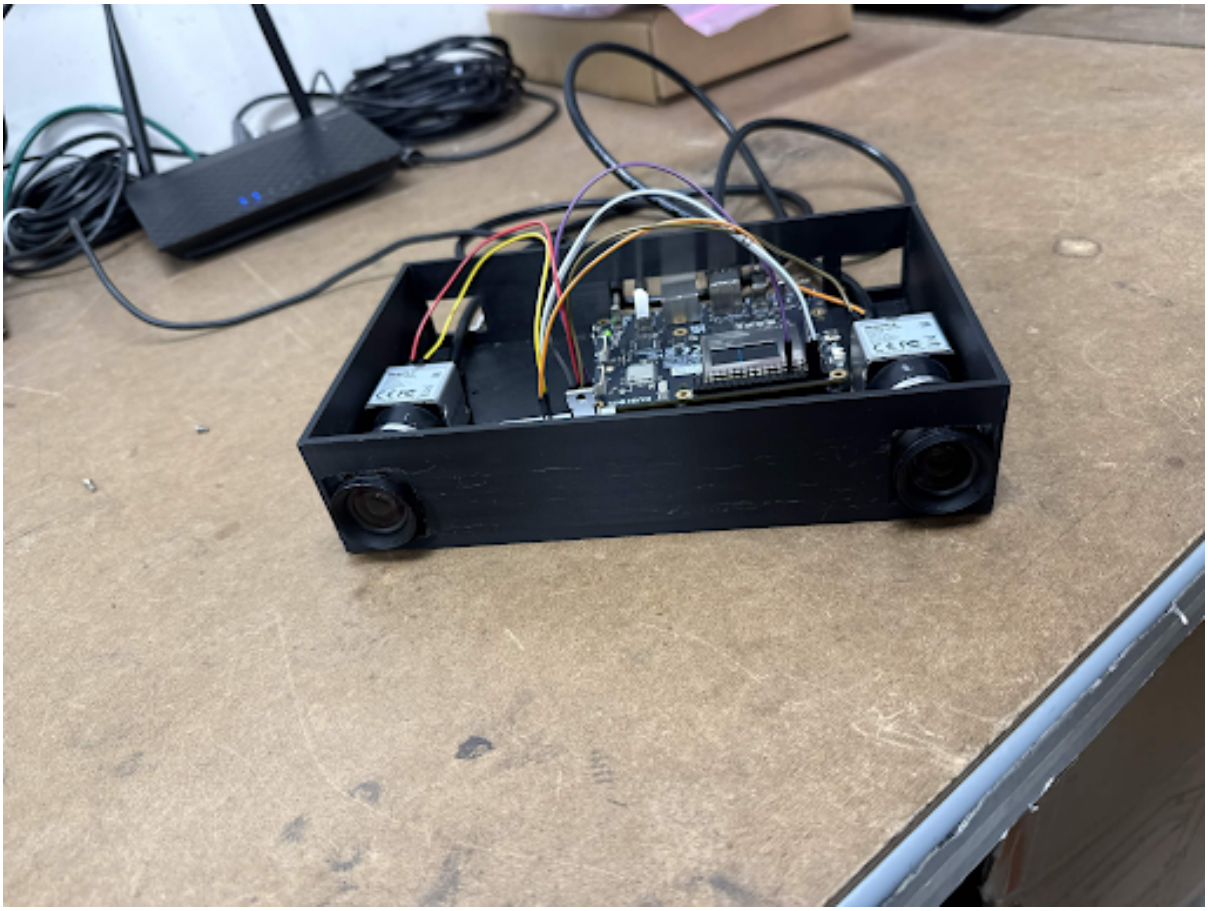

Fig. 2. Hardware setup.

4.2 Calibration and Rectification

Accurate depth recovery depends on knowing each camera's intrinsic parameters and the extrinsic geometry relating the two. We calibrate using Zhang's planar technique [13], observing a calibration target in several orientations to recover each camera's intrinsic matrix and lens distortion coefficients, along with the rotation and translation between the cameras. With these parameters, each incoming stereo pair is rectified so that corresponding points lie on the same horizontal scanline. Rectification reduces the correspondence search from two dimensions to a one-dimensional search along image rows, which is both faster and more robust, and is a prerequisite for the block-matching stage that follows.

4.3 Software and ROS2 Architecture

The software stack is organized as a set of ROS2 nodes communicating over well-defined topics. The camera node acquires synchronized frames and publishes the left and right images, while the IMU node publishes timestamped inertial measurements. These streams are consumed by the stereo depth node, which performs rectification, disparity estimation, and reprojection to produce the colored point cloud described in Section 4.4. A separate node

handles visual-inertial odometry, subscribing to both the image and IMU streams to estimate the camera's pose and publishing the resulting transforms through tf2 for visualization in tools such as RViz2. This decomposition keeps the point cloud module agnostic to its image source, the same node operates identically on frames from the physical cameras or from a synthetic source used during development.

Because the Rubik Pi is an embedded ARM64 platform, we build the software through a Dockerized cross-compilation workflow. Dependencies are compiled in a container on x86 development laptops and the artifacts are deployed to the ARM64 target. This allowed fast iteration without building on the resource-constrained Pi. This also makes the development environment reproducible across the team.

4.4 Point Cloud Pipeline

The point cloud pipeline is written in Python using OpenCV [2]. Each rectified stereo pair is processed into a colored 3D point cloud in which every point carries both a spatial position and an RGB value sampled from the source imagery.

We considered both sparse and dense reconstruction. We initially favored a sparse approach for efficiency, using the real-time ORB feature detector [10] to extract and match keypoints across the pair, which were then colored and projected into 3D using their disparities and the rectified projection matrices. In practice this did not yield enough keypoints to form a recognizable scene. We therefore moved to dense matching, computing a disparity map by matching blocks of pixels within the same row across the two rectified images. We use Semi-Global Block Matching (SGBM) [5], which optimizes an energy function with penalties for large disparity jumps to suppress noise and smooth the disparity map, followed by a Weighted Least Squares (WLS) filter for additional smoothing. Every matched pixel is reprojected into 3D and added to the point cloud with its original color. Although denser and more expensive than the sparse alternative, careful optimization allows the system to capture and process synchronized stereo imagery at approximately 100 frames per second at a resolution of 1280×1024 on the Rubik Pi.

To develop the pipeline before the physical camera was available, we rendered several synthetic stereo pairs in the Unity game engine, where ground-truth geometry and camera parameters are known exactly. Once the physical camera was assembled and calibrated, the camera node published rectified stereo data over ROS topics for the point cloud module to consume. Because the module operates only on rectified image pairs, the transition from synthetic to real input required no changes to the reconstruction code. We also wrote a small suite of unit tests to guard against regressions.

4.5 Visual-Inertial Odometry

To track the camera's motion and accumulate point clouds from successive frames into a coherent reconstruction of a larger scene, we implemented a visual-inertial odometry (VIO) pipeline fusing the camera and IMU streams. Rather than implementing the estimator from scratch, we built on the VINS node library, writing custom configuration files for our hardware. Fusing visual and inertial measurements requires an accurately characterized IMU, so we hand-tuned the IMU update frequency and noise parameters until the estimator produced stable, accurate pose estimates, and modified parts of the library to run efficiently within the Rubik Pi's compute budget. The system successfully tracks its own pose as it moves, publishing transforms through tf2 so that per-frame point clouds can be expressed in a common world frame and accumulated.

4.6 Semantic Segmentation

As an extension, we added semantic segmentation as an alternative point cloud coloring mode using YOLO [9], a real-time object detection model, to produce segmentation masks from the left image of each rectified pair. The per-object masks are combined into a general mask that labels each pixel with its associated object, or with no

object if no detection exceeds a confidence threshold of 0.8. After reprojection into 3D, the points of each detected object are colored uniformly with the average color of that object's pixels (or a random color if that average is black, to keep the object distinct), while unmasked pixels are colored black. This produces an object-level view of the point cloud that can be toggled with a single parameter.

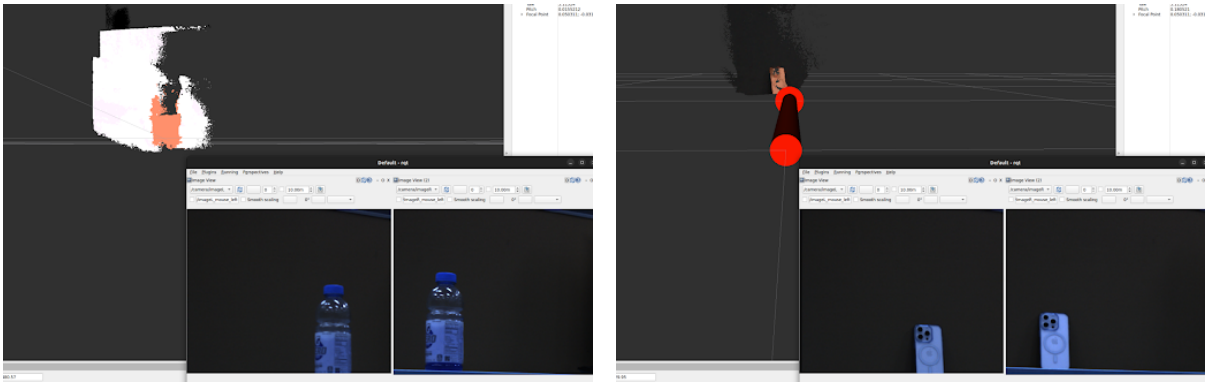

Fig. 3. Demonstration of yolo powered pointcloud segmentation

4.7 Benchmarking

We evaluated the platform qualitatively and quantitatively against two commercial devices: the Stereolabs ZED 2 stereo camera and the Manifold Tech Odin1 solid-state LiDAR, capturing comparable scenes and comparing the resulting point clouds on frame rate, resolution, and geometric fidelity. On capture performance our platform is competitive: it sustains a comparable or higher frame rate while operating at a higher resolution than the devices we benchmarked against. The trade-off is reconstruction quality. The fidelity of our point clouds is somewhat lacking relative to the commercial systems, whose proprietary depth pipelines and integrated optics produce cleaner, more geometrically accurate reconstructions. This is the expected cost of an open, low-cost design built from commodity components, and it points directly to the depth-estimation improvements discussed as future work.

5 Milestones

In our project specification, we laid out 5 deliverables and roughly 30 additional milestones. In this section we briefly review each one, noting whether it was achieved, any modifications we made, and pointing to the relevant technical section where applicable.

5.1 Deliverables

- **Project Specification.** Completed and submitted at the start of the project.
- **Minimum Viable Product (MVP).** Completed and presented at the mid-quarter oral update. We demonstrated a functional stereo camera running at 60 fps with a live RGB point cloud (Section 4.4).
- **Motion.** Completed. We wrote code to compensate for camera movement tracked by the IMU, shifting the existing point cloud by that amount so that new points assume accurate positions (Section 4.5).
- **Object Detection.** Completed, with a modification: the goal was changed from bounding-box detection to per-pixel segmentation, achieved by incorporating YOLO into the point-coloring code (Section 4.6).

- **Comparison Against Other Stereo Cameras.** Completed. We benchmarked against the Stereolabs ZED 2 stereo camera and the Manifold Tech Odin1 solid-state LiDAR, comparing our camera's performance against both (Section 4.7).

5.2 Other Milestones

Disparity Map and RGB Point Cloud. Completed early in the project. Using OpenCV in Python, we published an RGB point cloud from a computed disparity map, initially verified with synthetic images (Section 4.4).

RPI Cross-Compilation, Hardware Assembly. Completed early in the project. We designed and assembled the 3D printed box and setup the dockerized workflow. (Section 4.3).

Camera Calibration. Completed. We calibrated using an ArUco tag and a checkerboard to recover the intrinsics and extrinsics of both cameras (Section 4.2).

Hardware Sync. Completed for our MVP, ensuring both cameras expose within microseconds of one another (Section 4.1).

VIO Setup and VIO Working Under Motion. Completed. The system fuses camera and IMU data to track its own pose as it moves (Section 4.5).

ROS2 Architecture Setup and Point Cloud Viewable in RViz. Completed. The point cloud is published over ROS2 topics and visualized in RViz2 (Section 4.3).

Depth Estimation with Real-Time Input Streams, Reliability Testing, and Room Mapping. Completed. By publishing camera outputs to ROS2 topics and sourcing them as point cloud input, we ran the pipeline on live imagery in real time. The point cloud clearly displayed depth, remained stable unless moved, and was tested across several rooms whose contents were recognizable in the output.

Accumulating Point Cloud. Completed. After implementing VIO, we added a node that fuses the point cloud with the published transforms to accumulate a point cloud of a larger scene over time (Section 4.5).

Depth Estimation Verification. Completed. We placed known objects at fixed, measured distances and confirmed that the reported point distances matched the real-world camera-to-object distances using a tape measure.

ZED 2 Functional and Comparison Against ZED 2. Completed and discussed in our benchmarking results (Section 4.7).

Odin1 Function and Comparison Against Odin1. Completed and discussed in our benchmarking results (Section 4.7).

Basic Object Detection and Highlighting Objects in the Point Cloud. Completed late in the project with YOLO. A master mask combines per-object segmentation masks and maps each pixel to its most confident detection (or none). The point cloud is then colored by this mask to highlight recognized objects, validated on household items such as water bottles and chairs (Section 4.6).

Optimization Target of 100 fps. Completed through optimizations across the point cloud and VIO pipelines (Section 4.4).

Explore, Implement, and Relight Mesh Generation. Not attempted. These milestones aimed to add a module to convert the point cloud into a mesh and optionally relight it with a rasterizer or ray tracer. They were not

completed due to insufficient time at the end of the project. They were listed as stretch goals in the original specification and were therefore non-essential.

Explore and Implement Gaussian Splatting. Not attempted. These milestones aimed to train Gaussian splats from the point cloud, supervised with the original images, with the potential to outperform standard Gaussian splatting by also optimizing for disparity and depth. They were not completed due to insufficient time at the end of the project. As stretch goals in the original specification, they were non-essential.

6 Conclusion

Commercial stereo cameras remain expensive, with retail prices ranging from roughly $500 on the low end to several thousand dollars on the high end. These prices place capable depth sensing out of reach for many students, researchers, hobbyists, and developers of low-cost robotic platforms, effectively pricing many users out of real-time Computer Vision applications altogether.

In this project, we set out to develop an open, low-cost stereo camera capable of producing an accurate colored 3D point cloud of a live scene with performance comparable to commercial products, and in this we largely succeeded. The design presented here, comprising two hardware-synchronized iRayple cameras rigidly mounted in a 3D-printed enclosure and controlled by an embedded Rubik Pi, costs approximately $360 to produce, a substantial reduction relative to comparable commercial devices.

The full pipeline, from rectification and SGBM disparity estimation through reprojection into a colored point cloud, runs entirely onboard the Pi and produces clearly recognizable reconstructions of both static and dynamic scenes at approximately 100 frames per second at a resolution of 1280×1024, exceeding both benchmarked devices on frame rate and resolution. We further integrated visual-inertial odometry to track camera motion and accumulate point clouds across frames, and added YOLO-based semantic segmentation as an alternative object aware coloring mode, demonstrating that an open platform can support extensions well beyond raw depth capture.

The principal limitation of our design is point cloud fidelity, our reconstructions remain noisier and less dense than those of the commercial systems, an expected trade-off of an open design built from commodity components and a classical depth pipeline rather than proprietary optics and tuned processing. Several directions could close this gap. The most promising is replacing our classical SGBM pipeline with a learning-based stereo matching network, run on the Rubik Pi's onboard NPU to preserve real-time performance while improving depth accuracy. Additional gains may come from improved disparity post-processing, more rigorous calibration, and confidence-based outlier rejection. Finally, the scene-representation goals we deferred (mesh generation and Gaussian splatting) would convert our raw point clouds into denser, more continuous reconstructions. Taken together, these improvements suggest that a low-cost, open stereo camera can become a viable alternative to closed commercial systems, broadening access to depth sensing and real-time Computer Vision.

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. 2008. Speeded-Up Robust Features (SURF). *Computer Vision and Image Understanding* 110, 3 (2008), 346–359. doi:10.1016/j.cviu.2007.09.014

[2] Gary Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* 25, 11 (2000), 120–125.

[3] Jia-Ren Chang and Yong-Sheng Chen. 2018. Pyramid Stereo Matching Network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5410–5418. doi:10.1109/CVPR.2018.00567

[4] Andreas Geiger, Philip Lenz, and Raquel Urtasun. 2012. Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 3354–3361. doi:10.1109/CVPR.2012.6248074

[5] Heiko Hirschmüller. 2008. Stereo Processing by Semiglobal Matching and Mutual Information. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30, 2 (2008), 328–341. doi:10.1109/TPAMI.2007.1166

[6] Heiko Hirschmüller and Daniel Scharstein. 2009. Evaluation of Stereo Matching Costs on Images with Radiometric Differences. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31, 9 (2009), 1582–1599. doi:10.1109/TPAMI.2008.221

[7] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. 2017. End-to-End Learning of Geometry and Context for Deep Stereo Regression. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 66–75. doi:10.1109/ICCV.2017.17

[8] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110. doi:10.1023/B:VISI.0000029664.99615.94

[9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788. doi:10.1109/CVPR.2016.91

[10] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An Efficient Alternative to SIFT or SURF. In *2011 International Conference on Computer Vision (ICCV)*. Barcelona, Spain, 2564–2571. doi:10.1109/ICCV.2011.6126544

[11] Daniel Scharstein and Richard Szeliski. 2002. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision* 47, 1–3 (2002), 7–42. doi:10.1023/A:1014573219977

[12] Jure Žbontar and Yann LeCun. 2016. Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. *Journal of Machine Learning Research* 17, 65 (2016), 1–32.

[13] Zhengyou Zhang. 2000. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, 11 (2000), 1330–1334. doi:10.1109/34.888718