HyperSight: Boosting Distant 3D Vision on a Single Dual-camera Smartphone

Zifan Liu
Shanghai Jiao Tong University
Shanghai, China
lzfmzj@sjtu.edu.cn

Hongzi Zhu*
Shanghai Jiao Tong University
Shanghai, China
hongzi@sjtu.edu.cn

Junchi Chen
Shanghai Jiao Tong University
Shanghai, China
calm_furious@sjtu.edu.cn

Shan Chang
Donghua University
Shanghai, China
changshan@dhu.edu.cn

Lili Qiu
University of Texas at Austin
Austin, USA
lili@cs.utexas.edu

ABSTRACT

Smartphones with dual cameras are increasingly popular due to the need of supporting 3D vision. The depth information is critical for 3D vision. However, the two cameras on a smartphone are too close to accurately estimate the depth information especially for objects beyond two meters. In this paper, we propose an innovative system, called HyperSight, to estimate the depth information of objects using a dual camera smartphone. HyperSight realizes a virtual long-baseline stereo vision rig by having a user to move the phone in the air. The phone movement is continuously tracked and estimated using the short-baseline dual camera seeing nearby objects. We implement HyperSight as software on a Commercial-Off-The-Shelf (COTS) smartphone and conduct real-world experiments. The results show that when measuring feature-rich objects at a distance of five meters, HyperSight achieves a mean depth error of 6cm, which is up to 10× and 18× improvement in the accuracy compared with the stereo vision system using the native dual cameras and the Measure app based on ARKit 1 on mobile devices, respectively.

CCS CONCEPTS

• **Human-centered computing → Ubiquitous and mobile computing systems and tools**; **Mobile devices**.

KEYWORDS

dual cameras, smartphone, depth estimation, near-far diversity

ACM Reference Format:
Zifan Liu, Hongzi Zhu, Junchi Chen, Shan Chang, and Lili Qiu. 2019. HyperSight: Boosting Distant 3D Vision on a Single Dual-camera Smartphone. In *The 17th ACM Conference on Embedded Networked Sensor Systems (SenSys '19), November 10–13, 2019, New York, NY, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3356250.3360029

*Corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SenSys '19, November 10–13, 2019, New York, NY, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6950-3/19/11…$15.00
https://doi.org/10.1145/3356250.3360029

1 INTRODUCTION

Motivation. Smartphones with dual cameras on the back are gaining great popularity in recent years. Though this trend is not completely ubiquitous yet, it is well on its way. For example, with a wide-angle lens and a telephoto lens, iPhone since 7 Plus is able to offer optical zoom and background blur; Huawei P9 and higher version smartphones equip one RGB camera and another monochrome camera, and combine the images together with algorithms for a better picture. As the original purpose of the extra camera is to bring astounding mobile photography, this requires a short separation between the cameras. With dual cameras, however, it has great potential to let a phone have an understanding of its surroundings by getting the depth and 3D coordinate information about what it sees or where it locates, breeding new mobile applications, such as indoor navigation [2], object recognition and tracking [10], and Augmented Reality (AR) services [1] [7]. For instance, a user can measure the size or establish a 3D model of an object of interest far from her with her phone.

However, getting depth and 3D coordinate information of surroundings on a smartphone is challenging due to the following requirements. First, it requires high accuracy in a large range. Due to the short separation, these native dual cameras can not accurately estimate the depth of distant objects. The 3D vision region with such cameras is quite short as illustrated in Figure 1(a). Second, it should be implementable on COTS smartphones and do not rely on other dedicated sensors. Third, it should be computationally tractable. Real-time response and power consumption are critical for resource-constrained mobile devices.

In the literature and industry, there have been a number of schemes in obtaining depth or 3D coordinate information on mobile devices. The Simultaneous Localization and Mapping (SLAM) technique [16] has been implemented on both Android and iOS platforms, such as Wikitude [25] and Kudan [15]. More advanced SLAM techniques like Google ARCore [7] and Apple ARKit [1] can identify walls and objects in the environment. These schemes combine inertial measurement unit (IMU) sensors and one single camera to track the phone movement and derive the 3D information of an environment. The derived information is normally not accurate due to large IMU sensor reading errors. With dual cameras available on a phone, classic stereo version algorithms [23] [11] can be used. However, such schemes need a large distance between

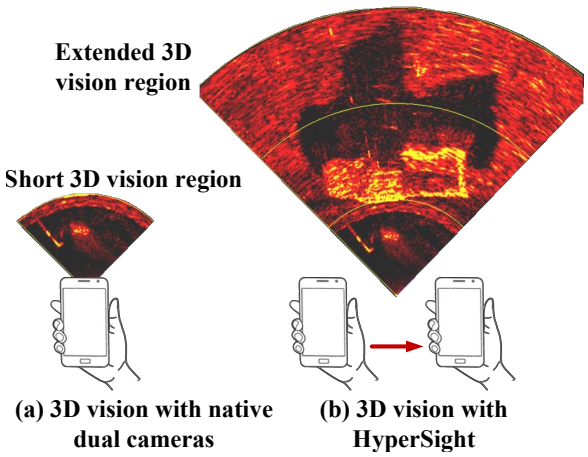


Figure 1: Illustration of effective 3D vision regions with native dual cameras and with HyperSight, respectively.

the two cameras, which cannot be easily deployed on smartphones. Another category of schemes in 3D reconstruction [4] [22] leverage dedicated sensors such as RGB-D cameras and time-of-flight laser sensors, which are not available on COTS smartphones. Some research studies attempt to get 3D maps using a single camera [19] [20]. However, it is difficult to get 3D information in absolute scale with only one camera.

Our approach. In this paper, we propose HyperSight, a system to improve the depth estimation accuracy for far away objects using off-the-shelf dual-camera smartphones. Due to the extremely short baseline between the dual cameras, it is very difficult to accurately measure the depth of objects far from the phone using classic stereo vision algorithms. The core idea of HyperSight is to constitute a virtual long-baseline stereo vision system by having a user to casually slide a phone with hand in the air. During the slide operation, images are continuously taken by the dual cameras and processed to track the motion of the phone. After getting the moving distance and orientation of the phone, the effective 3D vision region of the phone is largely extended through a virtual long-baseline stereo vision system, as illustrated in Figure 1(b).

Challenges and contributions. The first challenge is how to get the precise transformation (i.e., the translation and rotation) of the phone when it is moving. In traditional stereo vision systems, a *native stereo vision* is formed by two images simultaneously taken by two cameras, and the relationship between the two cameras is fixed and can be estimated via camera calibration. However, in HyperSight, a *virtual stereo vision* is formed by the two images taken by the same camera (e.g., one of the dual cameras, referred to as *the reference camera*) before and after a hand movement. Hand movement is unpredictable and cannot be estimated in advance, which makes it challenging to apply the existing work.

To address this challenge, we develop a novel approach inspired by the *near-far diversity phenomenon* of stereo vision (i.e., the depth resolution is low for distant keypoints but very high for nearby keypoints). We use a native stereo vision captured by the dual cameras to calculate the 3D coordinates of nearby keypoints. Then, with a sufficient number of such keypoints appearing in a virtual stereo vision, the corresponding phone transformation constituting this virtual stereo vision can be accurately derived.

The second challenge is the contradiction between the long-baseline requirement for ranging distant objects and that only short translations of the phone can be accurately estimated. As mentioned above, to have a sufficient number of nearby keypoints appearing in a virtual stereo vision, the corresponding transformation has to be short to have enough overlapping view fields.

To tackle this challenge, we automatically divide a long sliding operation into multiple short segments. The transformation of each segment, referred to as a *sub-transformation*, is accurately estimated and aggregated to derive the overall phone transformation after the slide.

We implement a prototype system based on Google Pixel 3 XL, a COTS Android smartphone. In our implementation, HyperSight is built on the `OpenCV3` library and leverages a 8-core CPU to speed up image processing. HyperSight is easy to use and can be used as a building block for other AR applications. We evaluate the performance of HyperSight through intensive real-world experiments in different environments. The results show that with the help of feature-rich nearby objects, HyperSight can achieves a mean depth estimation error of about 6cm for feature-rich distant object at a distance of five meters, which is up to 10× and 18× more accurate than the stereo vision system using the native dual cameras and the *Measure* app based on ARKit 1 [1], respectively.

In summary, our major contributions consist of (i) a novel algorithm to accurately estimate the depth of an object using dual cameras on a smartphone, (ii) a prototype implementation, and (iii) a systematic evaluation that shows the high accuracy of HyperSight.

2 PRELIMINARIES

2.1 Camera Model

A general camera model can be characterized by a simple pinhole camera model plus with lens distortions. In the pinhole camera model, as illustrated in Figure 2, a point Q in the physical world is projected through the pinhole, referred to as the *projection center*, to a point q on the image plane. For the ease of representation, we depict an equivalent virtual image plane in front of the pinhole plane. Given the coordinates of point Q in the world coordinate system, (X, Y, Z), if the camera coordinate system coincides with the world coordinate system, the pixel location of the projected point q on the virtual image plane, (x, y), satisfies the following equations:

$$x = f_x \cdot \frac{X}{Z} + c_x, \quad y = f_y \cdot \frac{Y}{Z} + c_y, \tag{1}$$

where f_x and f_y are the focal length represented in the units of pixels along the $x-$ and $y-$axis of the image plane and c_x and c_y are the possible displacement between the image center and the foot point. These are called the *intrinsic* parameters of the camera.

In case where the camera coordinate system does not coincide with the world coordinate system, an cartesian 3D point in world coordinates Q_w can be transformed in camera coordinates Q_c via a rotation and translation:

$$Q_c = R(Q_w - C_w) \quad \text{or} \quad Q_c = RQ_w + t, \tag{2}$$

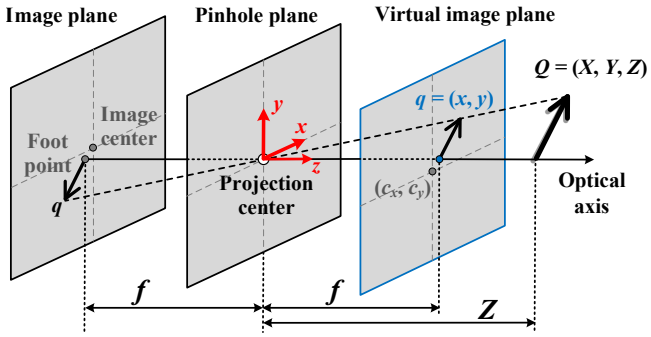


Figure 2: The fundamental pinhole model of a camera.

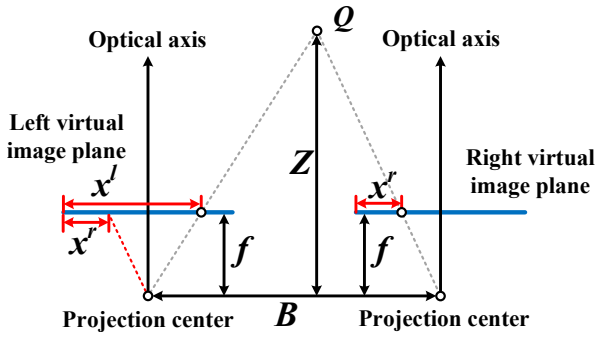


Figure 3: Derivation of the depth Z with a simplified binocular stereo rig of equal focal lengths.

where C_w is the projection center in world coordinates; R is a 3×3 rotation matrix describing the orientation of the camera in space and $t = -RC_w$ is a translation vector denoting the translation between two coordinate systems. The R and t are called the *extrinsic* parameters of the camera.

In practice, a lens instead of a small pinhole is used to focus a large amount of light on a point, but it comes at the cost of introducing radial and tangential distortions. The radial distortion occurs because rays farther from the center of the lens are bent more than those closer from the center with a spherical lens. Such distortion is zero at the optical center of the imager and increases as it moves toward the periphery, which can be characterized and corrected by the first few terms of a Taylor series expansion according to the following equations:

$$\begin{aligned} x_{\text{corrected}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{\text{corrected}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{3}$$

where (x, y) is the original location of the distorted point and $(x_{\text{corrected}}, y_{\text{corrected}})$ is the new location as a result of the correction; r is the distance from the point to the optical center; k_1, k_2 and k_3 are three distortion coefficients.

The tangential distortion is due to manufacturing defects resulting from the lens not being exactly parallel to the image plane, which can be characterized and corrected by the following equations:

$$\begin{aligned} x_{\text{corrected}} &= x + [2p_1 y + p_2(r^2 + 2x^2)] \\ y_{\text{corrected}} &= y + [p_1(r^2 + 2y^2) + 2p_2 x] \end{aligned} \tag{4}$$

where p_1,p_2 are two distortion coefficients.

Both the camera's intrinsic and extrinsic parameters and the distortion parameters can be estimated through a process of camera calibration.

2.2 Near-far Diversity with Stereo Vision

In binocular stereo vision, keypoints in two images taken at the same time from two separate cameras are matched and the differences are analyzed to yield depth information.

For example as illustrated in Figure 3, we assume that the image planes of two identical and undistorted cameras are exactly coplanar with each other and the optical axes of those cameras are parallel and a known distance apart. We further assume the images are row-aligned and that every pixel row of one camera aligns exactly with the corresponding row in the other camera. In this simplified case, if a point Q in the physical world is viewed in the left and the right images at (x^l, y^l) and (x^r, y^r), respectively, the depth Z can be derived by using similar triangles. In specific, we have

$$Z = \frac{B \cdot f_x}{x^l - x^r}, \tag{5}$$

where B, referred to as the *baseline*, is the distance between both cameras; f_x is the focal length as defined in (1); $d = x^l - x^r$ is referred to as the *disparity* between both views. Note that if the dual cameras have distinct focal lengths, which is often the case for the dual cameras on smartphones, the depth Z can also be easily derived.

From the above equation, it can be seen that the depth is inversely proportional to the disparity between views, i.e., the further the object is the smaller the disparity would be, which makes such stereo vision systems have high depth resolution only for objects relatively near the cameras but very low depth resolution for objects far away from the cameras, due to the rounding errors when measuring the disparity in pixels. We refer to this phenomenon as the *near-far diversity* of depth estimation in binocular stereo vision. To verify this, we measure the depth of objects of different ranges with our experimental binocular stereo module (see §5.1) with a short baseline of about 2.5cm, and plot the cumulative distribution functions (CDFs) in Figure 4. It can be seen that the error of measurement dramatically increases with the depth of objects. For example, with a 90% precision, the depth error is about 5.8% for objects of 1m away while that drops to about 29.4% for objects of 7m away.

If we increase the baseline of such a binocular stereo rig with all other settings unchanged, the the disparity between views would also increase, which leads to better depth estimation accuracy. Unfortunately, the dual cameras on smartphones are designed mainly for special photo effects but have more severe near-far diversity problem for depth estimation with a very short baseline (e.g., normally less than 1.5cm), resulting in very inaccurate results for ranging distant objects.

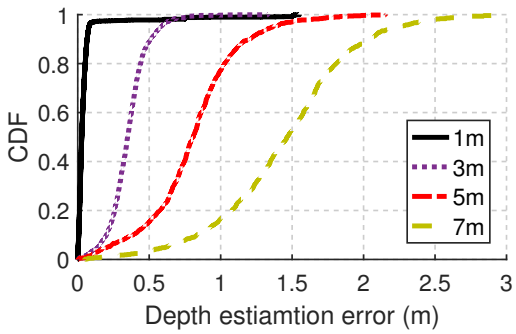


Figure 4: CDF of depth estimation errors with short baseline stereo vision.

3 SYSTEM DESIGN OF HYPERSIGHT

3.1 Overview

HyperSight is a depth estimation system built on smartphones with dual cameras, which overcomes the short baseline issue for ranging far objects. The core idea is to constitute a *virtual long-baseline* stereo vision rig by moving such a phone in the air. To this end, the main challenge is to estimate the extended baseline of this virtual stereo vision rig in very high accuracy. In HyperSight, we solely rely on the dual cameras and the distance diversity of seen objects to tackle this challenge.

Figure 5 illustrates the operation flow of HyperSight. To use HyperSight, a user needs to hold a dual-camera phone, selects a target object on the device screen (e.g., the five-pointed star in Figure 5), and *freely* slides the device in the air along the indicated direction displayed on the screen while both cameras conduct video recording. As a result, a consecutive series of frames (denoted as F_i, $i = 1, \ldots, k$ is the index of a frame) are generated. Each frame contains two images, denoted as F_i^l and F_i^r, taken by the two cameras, respectively. Being aware of the near-far diversity problem in stereo vision, HyperSight leverages nearby objects seen by the dual cameras to continuously tracks the translation (e.g., t_j) and rotation (e.g., R_j) of the phone between two frames (e.g., from $F_{\widehat{j}}$ to $F_{\widehat{j+1}}$). Then the overall geometric transformation of the phone

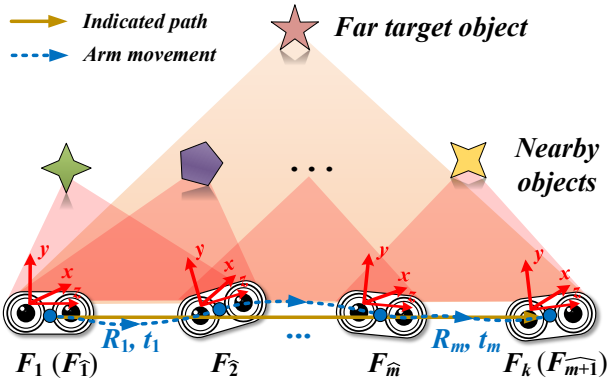


Figure 5: Illustration of HyperSight's operation workflow.

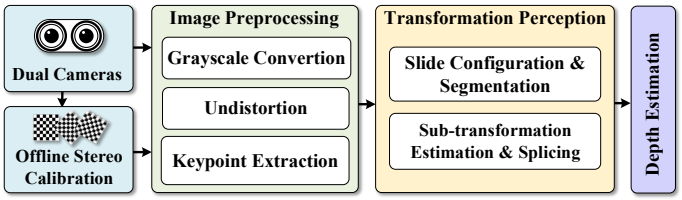


Figure 6: The system architecture of HyperSight.

can be estimated by accumulating all intermediate translations and rotations. Finally, images taken before the and after the overall transformation (e.g., images taken at F_1 and at F_k) and the overall transformation information are used to estimate the depth of target object.

The architecture of HyperSight, illustrated in Figure 6, consists of four main components as follows:

Offline Stereo Calibration. Stereo calibration estimates the intrinsic and distortion parameters of each camera and finds the rotation matrix and translation vector between the two cameras. Such parameters are fixed once the dual cameras are manufactured and therefore stereo calibration only needs to be conducted once for future use.

Image Preprocessing. During video recording, each image taken by both cameras is converted to gray scale and undistorted according to the distortion parameters derived from the camera calibration. Then distinctive keypoints that are invariant to location, scale and rotation, and robust to affine transformations and changes in illumination are identified. All subsequent operations are then based on extracted keypoints.

Phone Transformation Perception. This component estimates the transformation of the phone in 3D space caused by the slide operation of the user. To derive accurate transformation information, the slide is automatically broke into smaller segments and the sub-transformation of each segment is respectively estimated, according to those common nearby keypoints in images taken at both ends of a segment. Finally, the overall transformation in terms of phone translation and rotation is perceived by splicing each sub-transformation.

Depth Estimation: Given the transformation information about the slide operation, keypoint matching is conducted to identify common far keypoints on the two images taken before and after the slide, and the depth of such keypoints can be calculated according to the stereo vision geometry.

3.2 Offline Stereo Calibration

It is fundamental to know the intrinsic and distortion parameters of each camera as introduced in the camera model. In addition, cameras will almost never be exactly aligned in the frontal parallel configuration depicted in Figure 3. Therefore, it is also necessary to acquire the relative rotation and translation between the twocameras. To calibrate such parameters, a user (or the phone manufacturer) needs to take a few images of a "chessboard" (i.e., a planar object with a pattern of alternating black and white squares) at various orientations with the dual cameras.

We first adopt the classic single-camera calibration approach [27] to separately estimate the intrinsic and distortion parameters of each camera. In principle, three chessboard interior corners yielding six constraints (because each point on the image has both an x and a y coordinate) are all that is needed to solve the five distortion parameters. Therefore, one image of a chessboard is all that we need to compute the distortion parameters. As the chessboard moves in each image, we need to solve four invariant intrinsic parameters and six extrinsic parameters for each image. With the planar chessboard, because four points are sufficient to express a planar perspective view, we get eight constraints from each image. Thus, two images of a 3×3 chessboard are the minimum that could solve the single-camera calibration problem. In practice, for high-quality results, we take twenty images of a 8×9 chessboard. Finally, all the parameters are refined by minimizing the re-projection error of all chessboard corners using a robust Levenberg-Marquardt iterative algorithm.

To get the relative rotation R^* and translation t^* between the two cameras, given the intrinsic and extrinsic parameters of both cameras for each image, a chessboard corner point P can be put in the camera coordinates $P^l = R^l P + t^l$ and $P^r = R^r P + t^r$ for the left and right cameras, respectively. Because $P^l = (R^*)^T (P^r - t^*)$, we have $R^* = R^r (R^l)^T$ and $t^* = t^r - R^* t^l$. We then take the mean values of R^* and t^* over all chessboard corner pairs to mitigate image noise and rounding errors.

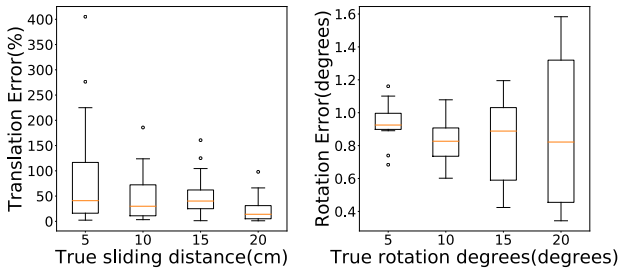
3.3 Image Preprocessing

To compare different images taken by the dual cameras at the same time or at different time, we need to preprocess each image to extract keypoints of interest. More specifically, each image is converted to gray scale for the ease of calculating the image gradients and undistorted according to (3)(4) and the calibrated distortion parameters.

In HyperSight, we adopt the SIFT algorithm [18] to identify distinctive keypoints that are invariant to location, scale and rotation, and robust to affine transformations and changes in illumination for each image. As a result, keypoints are localized as maxima and minima of the result of difference of Gaussians function applied in scale space to a series of smoothed and resampled images. For each keypoint, the gradient magnitude and direction is calculated around the point, and the dominant orientation is assigned to this keypoint. Moreover, a keypoint descriptor is calculated and associated with each keypoint, which is a vector of 128 values, describing the gradient orientation distributions of 16 surrounding blocks of 4×4 size. With keypoint descriptors, keypoints between different images can be matched by identifying the smallest Euclidean distance of their descriptors. The RANSAC algorithm [6] can be employed to remove false matching.

3.4 Phone Transformation Perception

The key component in HyperSight is to perceive the transformation caused by the user's slide operation. One straightforward scheme is to estimate the phone motion and rotation using inertial sensors, which are power efficient and readily available on smartphones. For example, an intuitive way to estimate the moving speed of the phone along one axis is to calculate the integral of linear acceleration along that axis over time. In addition, according to our prior work [8] [26],



(a) Translation estimation error using 3D accelerometer **(b) Rotation estimation error using gyroscope**

Figure 7: Large estimation errors are produced when using inertial sensors to estimate phone transformation.

there is accumulative error of integral, which is approximately a linear function of time. Given the fact that the true velocity at both ends of a slide is zero, the linear model of errors can be derived and utilized to infer accurate moving speed. As a result, the displacement between any two time instants during a slide can be derived by taking the integral of corrected moving speed over time. Similarly, the rotation of the phone can be estimated by calculate the integral of angular speed readings from the gyroscope over time. Figure 7(a) and (b) depict the translation and rotation estimation errors using above scheme, respectively. The ground truth of sliding distance is obtained by sliding an experiment phone on a slider and the ground truth of rotation degrees is obtained by rotating the phone on a turnplate. It can be seen that mean translation estimation errors are over 10% and the mean rotation errors are over 0.8 degree in all cases. Consequently, due to sensor reading errors, it is very hard for the above scheme to meet the high accuracy requirement for phone transformation estimation.

In contrast, HyperSight solely relies on the following two computer vision techniques to get accurate phone transformation estimation.

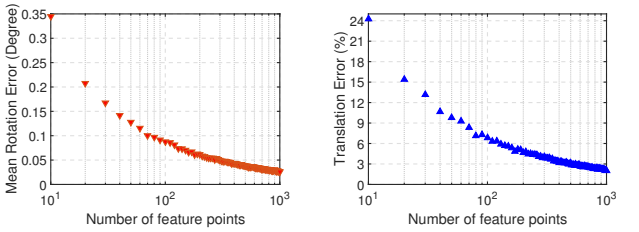
3.4.1 Slide Configuration and Segmentation. One major concern is *how far a user should slide a phone so as to achieve the desired depth estimation accuracy?* We have the following observation.

Observation 1. *The depth estimation error can be reduced to any desired ratio as long as the baseline of the dual cameras can be sufficiently prolonged.*

Recall that the disparity of a keypoint between images is measured as $d = x^l - x^r$ (see Figure 3), where x^l and x^r are in units of pixels. Due to image noise and digitization error, the disparity is accurate only to within one pixel. Assume that Z' is the estimated depth when the measured disparity d' is one pixel deviated from the ground truth d, i.e., $d' = d \pm 1$, according to (5), we have

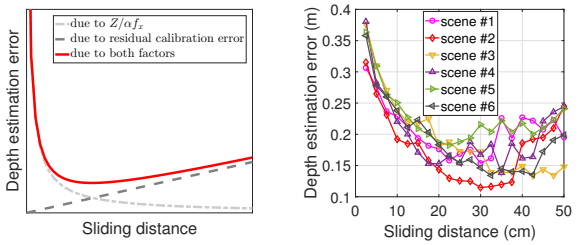
$$\frac{Z}{Z'} = 1 \pm \frac{1}{d} = 1 \pm \frac{Z}{Bf_x}. \tag{6}$$

This means that, given f_x (note that cameras on smartphones usually have an invariant lens and focal length due to the restricted size) and a desired depth estimation error ratio α, i.e., $1/d \leqslant \alpha$, B should be no less than $Z/\alpha f_x$.



(a) Mean rotation estimation error (b) Mean translation estimation error

Figure 8: Phone transformation estimation accuracy versus the number of keypoints used in solving the PnP problem.



(a) Simulated depth estimate error (b) Measured depth estimate error

Figure 9: Relation between the sliding distance and the overall depth estimation accuracy.

Implication: From Observation 1, it seems that the longer the user slides the phone, the better the depth estimation results would be. The lower bound of the sliding distance is $Z/\alpha f_x$.

However, the above observation is only true when B and the rotation of the phone can be accurately estimated after the slide operation. We investigate *how to accurately estimate the transformation of the phone* and have the following observation.

Observation 2. *The phone transformation can be accurately estimated when sufficient common nearby keypoints exist before and after a slide operation.*

As illustrated in Figure 5, given a well calibrated dual cameras, the 3D coordinates of a common keypoint on the two images of one frame can be established with the native stereo vision. If a set of such n keypoints are also seen on one of the images of another frame, the transformation of the phone between the two frames can be estimated by solving a Perspective-n-Point (PnP) problem (detailed in § 3.4.2). Due to the near-far diversity problem in stereo vision, only nearby keypoints can be used and their 3D coordinates can be accurately estimated. In addition, as the number of keypoints increases in solving the PnP problem, the accuracy of the transformation estimation also increases. For example, we examine the transformation estimation accuracy through simulations, where our experimental binocular stereo module sees 1,500 randomly generated points (<2m away from the module) before and after a group of random transformations. Figure 8 plots the mean rotation and mean translation estimation errors as functions of the number of keypoints used in solving the PnP problem.

Implication: To see sufficient number of common nearby keypoints, it is evident that the dual cameras cannot be slid too far, which contradicts Observation 1. In HyperSight, this contradiction is resolved by breaking a long-distance slide into small segments, estimating the transformation of each segment (called sub-transformation), and splicing all sub-transformations to derive the transformation of the slide.

Observation 3. *The transformation estimation error gradually increases as the number of sub-transformations increases due to residual camera calibration error.*

As camera calibration error exists, systematic errors are accumulated each time a sub-transformation is estimated and aggregated. In summary, as illustrated in Figure 9(a), the depth estimation error dramatically decreases and then gradually increases as the sliding distance keeps increasing. We measure distinct target objects of 5m away from our prototype system in different scenarios. Figure 9(b) shows that the experimental measurement is highly consistent with the simulation result. It can be seen that, given a target scenario, the optimal sliding distance and slide segmentation scheme exist. However, it is hard to find the optimal slide configuration and segmentation scheme because the target scenario information, the residual camera calibration error, and how a user would move the phone are unknown.

In practice, HyperSight adopts a threshold-based scheme. Specifically, to determine the sliding distance, the sets of keypoints on image F_1^l and F_1^r, denoted as $\mathcal{F}_{F_1^l}$ and $\mathcal{F}_{F_1^r}$, respectively, are matched and the depth information of matched keypoints is estimated with the native stereo vision according to (5). According to Observation 1, given the average depth of keypoints on the target object, denoted as $\overline{Z'}$, the sliding distance can be determined as $\overline{Z'}/\alpha f_x$, where α is the desired error threshold. To segment the slide, the set of nearby keypoints on the beginning frame of a new segment are first identified with the native stereo vision. We denote the beginning frame of the i^{th} segment as $F_{\widehat{i}}$ (e.g., $F_{\widehat{1}} = F_1$ as illustrated in Figure 5,) and the set of nearby keypoints found on $F_{\widehat{i}}$ as $\mathcal{N}_{F_{\widehat{i}}}$. Then, the ending frame of the ith segment, which is also the beginning frame of the $i+1^{\text{th}}$ segment $F_{\widehat{i+1}}$, is the frame that has the maximum frame index such that at least δ nearby keypoints identified in $F_{\widehat{i}}$ can be found on the left (or the right) image of such a frame through keypoint matching, i.e.,

$$\widehat{i+1} = max\{x : \mathcal{F}_{F_x^l} \cap \mathcal{N}_{F_{\widehat{i}}} > \delta\}. \tag{7}$$

In this way, a slide can be continuously divided into segments and estimated by aggregating all sub-transformations of each segment (see §3.4.2) until the required distance $\overline{Z'}/\alpha f_x$ is reached.

It should be noted that the slide is automatically segmented and do not need a user to pause during the slide. In addition, HyperSight do not have a rigid requirement on how the slide operation should be conducted. In general, as images are taken during the slide operation, it is satisfactory as long as the user controls the sliding speed so that taped images are not blur until the required sliding distance is reached. For instance, the user can choose an arbitrary direction to start a slide operation in 3D space.

3.4.2 Sub-transformation Estimation and Splicing. As stated above, given the beginning and ending frames of the i^{th} segment, i.e., $F_{\widehat{i}}$

and $F_{\widehat{i+1}}$, sub-transformation of the i^{th} segment is estimated in the following three steps:

1) Resolving the 3D coordinates of nearby keypoints in one frame: Keypoints between the two images $F^l_{\widehat{i}}$ and $F^r_{\widehat{i}}$ of $F_{\widehat{i}}$ is matched, and their 3D coordinates in the coordinate system of the left camera at frame $F_{\widehat{i}}$ can be resolved according to the camera model (2). The set of nearby keypoints, i.e., $\mathcal{N}_{F_{\widehat{i}}}$, are then identified, if the depth of a keypoint is less than a distance threshold σ. To choose a σ is a tradeoff between system usability and performance. Specifically, a short distance threshold finds less near objects to use, i.e., poor usability while a large distance threshold leads to coarse 3D coordinate estimations, i.e., poor performance. In practice, we select a distance threshold so that the errors is less than 2% when using the native dual cameras for depth estimation.

2) Identifying common nearby keypoints in the beginning and ending frames: Then, the common nearby keypoints are found between image $F^l_{\widehat{i}}$ and $F^l_{\widehat{i+1}}$ through keypoint matching.

3) Estimating sub-transformation: After that, given the 3D coordinates of common nearby keypoints and their 2D projections on image $F^l_{\widehat{i+1}}$, the rotation and translation of the dual cameras from $F_{\widehat{i}}$ to $F_{\widehat{i+1}}$ can be estimated by solving a PnP problem. In HyperSight, we adopt the EPnP algorithm [17] to solve the PnP problem because of its high accuracy and low computational cost. The key idea of EPnP is using four control points to represent all input points, dramatically reducing the computation complexity. We briefly introduce the EPnP algorithm in the Appendix. In general, as the number of keypoints increases, the performance of EPnP improves. Finally, we adopt bundle adjustment (BA) [24] to adjust the 3D coordinates of all input points and the estimated sub-transformation (i.e., the estimated rotation R_i and the translation t_i) to minimize the re-projection error.

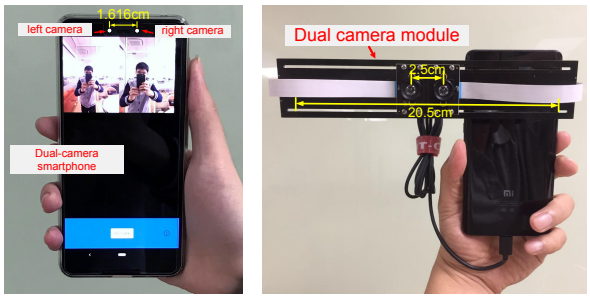
With known sub-transformations, R_i and t_i, $1 \leqslant i \leqslant m$, the transformation of the slide R and t can be derived as

$$\begin{aligned} R &= \prod_{i=1}^{m} R_{m+1-i} \\ t &= t_m + \sum_{i=1}^{m-1} \Big(\prod_{j=i+1}^{m} R_{m-j+i+1} t_i \Big). \end{aligned} \tag{8}$$

With R and t, the depth of the target object can be estimated by matching the keypoints of the target object on image F^l_1 and F^l_k and calculating the depth of all matched keypoints with stereo vision geometry.

4 IMPLEMENTATION

Hardware: Though there are a rich set of COTS smartphone models that are equipped with dual cameras, we implement a prototype of HyperSight on Google Pixel 3 XL, running Android 9 Pie which provides the Multi-camera API to obtain simultaneous images from the front dual cameras. The smartphone has a 2.8GHz 8-core CPU, 4GB memory and an Adreno 630 GPU. The dual cameras have a short baseline and the left and the right cameras have a fixed optical focal length of 2.03mm and 2.97mm, respectively. Figure 10(a) depicts our prototype implementation and the operation screen.



(a) HyperSight implemented on an Android phone (b) Native stereo vision system using a dual camera module with adjustable baselines

Figure 10: (a) Our prototype implementation; (b) Native stereo vision system developed for comparison.

Software: We implement HyperSight as an app using the Multi-camera API introduced with Android Pie to access the dual camera. More specifically, we turn off the auto focus feature and set the focal distance of both cameras to infinity. Images can be taken by both cameras with a wide set of resolutions, ranging from 3264×2448 pixels to 176×144 pixels. For the ease of performance comparison with a native stereo vision system, images are taken with a resolution of 1280×720 pixels at 5fps. We first run the offline camera calibration module for the dual cameras. After calibration, the intrinsic parameters (f_x, f_y, c_x, c_y) of the left and the right cameras are (724.0586, 725.2506, 655.3243, 361.0285) and (1059.9694, 1062.4389, 650.7577, 363.0094) in units of pixels, respectively. The baseline of the dual cameras is 1.6163cm. The relative rotation and translation between the two cameras are also calibrated. With this resolution, we conduct depth estimation using the native dual cameras and learn an appropriate distance threshold $\sigma = 2.5$m with a 95% confidence interval. We build the `OpenCV3` library for basic algorithms including camera calibration, keypoint extraction and matching, and the EPnP algorithm.

It should be noted that it is natural to operate HyperSight with real dual cameras. It would be a little bit awkward when operating HyperSight with front dual cameras as a user cannot see the screen when he/she slides the phone.

5 PERFORMANCE EVALUATION

5.1 Methodology

We first investigate how system parameters affect the performance of HyperSight and then examine whether the type and distance of nearby and far objects also change the result of depth estimation in our laboratory. With parameters properly configured, we then conduct real-world experiments in two common indoor environments, i.e., a campus coffee shop and a supermarket. We recruit 10 volunteers, i.e., three females and seven males, aged from 18 to 43, including five undergraduate students, four graduate students, and one faculty member, as HyperSight operators in all experiments. Each volunteer operates the sliding maneuvers based on his/her own preference.

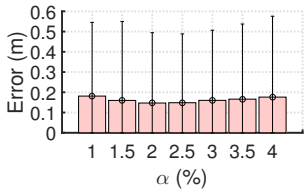


Figure 11: Effectiveness of α.

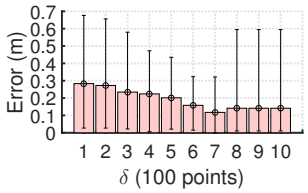


Figure 12: Effectiveness of δ.

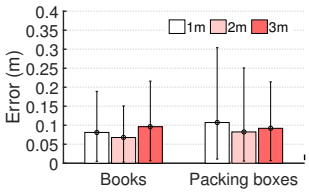


Figure 13: Impact of nearby objects.

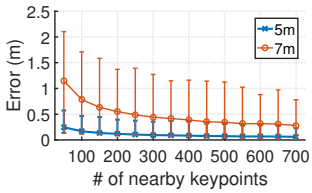


Figure 14: Impact of available nearby keypoints.

For comparison, we also implement a native stereo vision system using a commercial dual camera module as shown in Figure 10(b), which supports the UVC protocol to obtain simultaneous images from both cameras and has a variable baseline ranging from 2.5cm to 20.5cm. Images can be taken by both cameras with a resolution of 1280×720 pixels. After calibration, the intrinsic parameters (f_x, f_y, c_x, c_y) of the left and the right cameras are (1121.3967, 1121.5880, 777.1217, 294.9875) and (1131.5911, 1131.2504, 547.7323, 330.4111) in units of pixels, respectively. The relative rotation and translation between the two cameras are recalibrated each time we change the baseline between both cameras. We compare HyperSight with the following schemes:

- **Short-baseline stereo vision (SSV):** In this scheme, we take a frame of two images with the dual camera module calibrated with a baseline of 2.5cm. Images are preprocessed and matched keypoints are directly applied to naive stereo vision for depth estimation. BA is also used to further improve depth estimation accuracy.
- **Long-baseline stereo vision (LSV):** In this scheme, we conduct similar operations as in the short-baseline stereo vision, except that the baseline of the dual camera module is set to the largest distance of 20.5cm and recalibrated.
- **ARKit Measure app:** ARKit [1] is a set of software development tools based on SLAM technology to enable developers to build augmented-reality apps for iOS. *Measure* is an ARKit application allowing users to measure objects in the real world using a single camera on iOS device. We use the *Measure* app based on ARKit 1 to measure a tape on the ground in the corresponding experiment environment with an iPhone 7.
- **ORB-SLAM2:** ORB-SLAM2 [21] is a real-time SLAM system that computes the camera trajectory as well as a sparse 3D reconstruction map with true scale. Its stereo version takes a series of stereo rectified images as input. The system extracts keypoints in stereo input with ORB, an efficient alternative to SIFT, and provides loop closure and re-localizing of the camera in real time.

We consider the metric of accuracy, defined as the difference between the estimated depth of a target object and the ground truth. It is not straightforward, however, to get the ground truth for two reasons. First, as HyperSight needs a user to slide the phone and estimates the depth of a far object relative to the image plane of F_1^l and F_k^l, the actual location of the baseline B is uncertain. Second, for an arbitrary object, there would be many keypoints of different depth associated with the object on images, i.e., the depth or size of the object. To better obtain the ground truth, we manually select those keypoints identified on F_1^l and F_k^l that locate *on the front surface* of the object and calculate the depth of the object as the mean depth estimation of these front keypoints. The ground truth is measured from a laser range finder to a target object at the estimated center location of a slide operation.

5.2 Parameter Configuration

In this experiment, we study the effectiveness of system parameters, i.e., the desired error threshold α and the number of common nearby keypoints between frames δ. We consider two target objects five-meter away from an operator and use several packing boxes one-meter away from the operator as nearby objects. First, we set δ = 1000 points and vary α from 1% to 4% with an increment of 0.5%. For each pair of α and δ, we let each operator conduct HyperSight to measure the depth of each target for twenty times.

Figure 11 plots the average estimation errors as a function of α. It can be seen that in general, HyperSight can achieve stable and low depth estimation error, i.e., < 4%, under different α values. In addition, as α increases (i.e., the corresponding required sliding distance decreases according to $B \propto Z/\alpha f_x$), the estimation error first declines because the accumulative transformation estimation error gets negligible, and then rises because the baseline extension is also reduced. As a result, the estimation error reaches the minimum at α = 2%.

Then, we fix α = 2% and vary δ from 100 points to 1000 points with an increment of 100 points. We repeat the experiment. Figure 12 plots the mean estimation error as a function of δ. Similar results are observed as δ increases and the minimal average (i.e., 2.2%) is achieved when δ = 700. The reason is that, as δ increases, sub-transformations can be better estimated but the number of sub-transformations also increases. The result is highly consistent with the analysis of Observation 2 and 3.

In practice, the optimal configuration of α and δ is device dependent and can be easily learned by measuring a known object through cross search.

5.3 Impact of Nearby Objects

In this experiment, we study how the types and distances of nearby objects affect the performance of HyperSight. We compare two groups of nearby objects, i.e., a pile of books and a sets of bottles and packing boxes. We then measure the depth of a printer as the reference object in our laboratory, where the printer is set five meters away from the phone operator and we vary the distance of each set of nearby objects from one meter to three meters. Then we set α = 2% and δ = 700 and let each operator run HyperSight

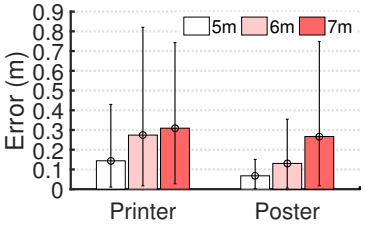


Figure 15: Impact of target objects.

Figure 16: Real-world test venues: a coffee shop (left) and a supermarket (right).

and the laser range finder to measure the depth of the printer for twenty times, respectively. The average required sliding distance is 22.3cm.

Figure 13 plots the average depth estimation error as a function of nearby object types and distances. It can be seen that, given the same distance, the depth estimation error is slightly smaller when using the set of books as the nearby objects because more keypoints can be found with books. Moreover, in this setting, it is clear that the estimation errors reaches the minima when the nearby objects are located at a distance of two meters. The reason is two-fold. On one hand, when nearby objects are very close, HyperSight breaks a slide into more segments in order to satisfy the requirement of δ, which leads to larger accumulative transformation estimation error due to the residual camera calibration error. On the other, when nearby objects are relative further, though the accumulative transformation estimation error is reduced, the estimation accuracy of each sub-transformation also drops due to inaccurate 3D coordinate estimation with the native dual cameras (see §3.4.2).

We further examine the impact of availability of nearby objects (or nearby keypoints) to the system performance. Specifically, we use the same configuration of α and δ. We use the pile of books set at a distance of 2 meters from operators as nearby objects, and use the printer as target object set at five and seven meters away from operators, respectively. Similarly, we let operators to measure the depth of the printer except that we randomly select the number of available nearby keypoints used in the algorithm, ranging from 50 to 700 with an increment of 50.

Figure 14 plots the the mean depth estimation errors and the corresponding 90% confidence intervals as a function of the number of available nearby keypoints. It is clear to see that obtaining sufficient nearby keypoints are key to the performance of HyperSight. When there are insufficient or even no nearby keypoints, HyperSight cannot perceive the phone transformation accurately in terms of the mean and the variance of depth estimation errors and therefore could fail in distant 3D vision in such circumstances. For example, when there are only about 50 common nearby keypoints available for sub-transformation estimation, the mean depth estimation error reaches 4.8% and 16.8% for objects at 5m and 7m away, respectively.

5.4 Impact of Far Objects

In this experiment, we study how different target objects affect the performance of HyperSight. We consider two target objects, i.e., a planar poster and a cubic printer. We put a pile of books 2m away from the operator as nearby objects and vary the distance of each target object from 5m to 7m with an interval of 1m. Then we set $\alpha = 2\%$ and $\delta = 700$ and let each operator run HyperSight and the laser range finder to measure the depth of both objects for twenty times, respectively.

Figure 15 plots the average depth estimation error as a function of target types and distances. It can be seen that the estimation error rises as the distance of target objects increases. This is because a far target object needs a long sliding distance. Given the same nearby objects and δ, the number of sub-transformations to be aggregated increases, leading to larger phone transformation and depth estimation errors. We can observe that the performance of HyperSight is similar for different target objects. In principle, HyperSight only depends on the image quality and the number of keypoints in the view fields of the dual cameras and has nothing to do with other factors of environments.

5.5 Real-world Experiments

We compared HyperSight with candidate schemes in two real-world test venues, i.e., a campus coffee shop and a supermarket as illustrated in Figure 10(b), where nearby objects can be easily found. To be fair, we lower the image resolution of our prototype to be the same as that of the native stereo vision system (i.e., 1280×720 pixels). In each environment, we randomly select a target object and, for each operator, we change the facing direction from the operator to the object for ten times. In each direction, we vary the distance between the operator and the target object from 5m to 7m and let the operator run HyperSight and SSV, respectively, to measure the depth of the target for five times. We repeat the process to measure ten targets. For LSV, we change the baseline of the dual camera module and recalibrate the camera before we conduct similar measurements in each venue. For *Measure* app, we measure a tape of different lengths (i.e., 3m, 4m and 5m) on the ground with rich wood texture for fifty times in each venue. For ORB-SLAM2, we use the same sequences of dual images as input as HyperSight does except that images are pre-rectified as required for ORB-SLAM2. We derive the camera trajectory using both schemes and calculate the depth of the same target keypoints for comparison.

Figure 17 depicts the CDFs of depth estimation errors using HyperSight and the SSV for target objects in different ranges. It is evident that HyperSight dramatically improves the ability of short-baseline dual camera device to range distant objects, achieving very low errors of 2.4%, 4.2%, and 7.7% with a precision of 90% at the distance of 5m, 6m, and 7m, respectively, which is up to 10×, 7×, and

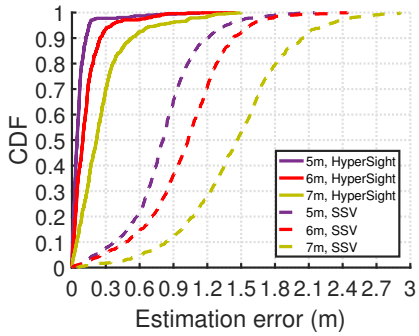


Figure 17: CDF of ranging errors using HyperSight and SSV.

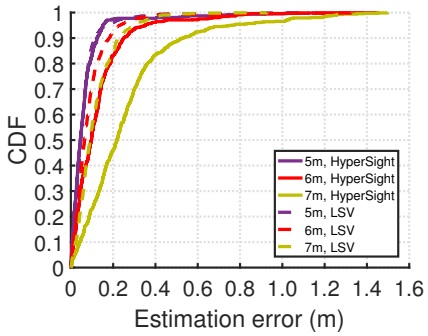


Figure 18: CDF of ranging errors using HyperSight and LSV.

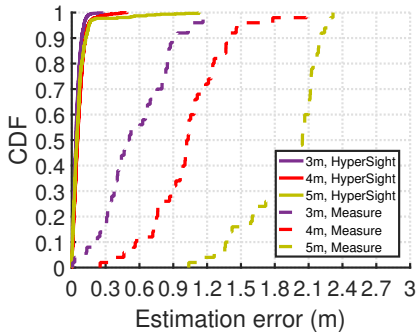


Figure 19: CDF of ranging errors using HyperSight and Measure app.

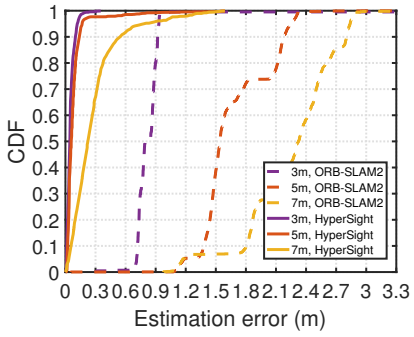


Figure 20: CDF of ranging errors using HyperSight and ORB-SLAM2.

5× better than SSV, respectively. Figure 18 plots the CDFs of depth estimation errors using HyperSight and the LSV. It can be seen that, compared with the LSV which has a similar baseline of most COTS stereo cameras, HyperSight can achieve similar accuracy. It should also be noted that the performance of HyperSight drops faster than that of the LSV as the depth of targets increases. This demonstrates the limitation of HyperSight (see §6). Figure 19 depicts the CDF of measuring errors using HyperSight and the ARKit Measure app. It can be seen that Measure, which uses one single camera and motion sensors, cannot achieve accurate depth measurement. In contrast, HyperSight utilizes a binocular system and demonstrates its efficacy in ranging far objects and achieves a maximum 18× performance gain at a distance of five meters, compared with Measure. Figure 20 depicts the CDF of measuring errors using HyperSight and ORB-SLAM2. It can be seen that HyperSight outperforms ORB-SLAM2 which also uses stereo images. It can also be seen that ORB-SLAM2 also takes stereo images as input and can achieve better depth estimation accuracy than *Measure*. The results demonstrate the efficacy of using nearby objects to help distant 3D vision on mobile devices with a dual camera.

5.6 Computational Overhead

The main computational overhead of HyperSight steps from identifying keypoints using the SIFT algorithm. In our implementation, an image is processed on a single CPU core using the `openCV` library which provides heavily optimized image processing routines. We process eight images simultaneously on the 8-core CPU and therefore the runtime for each image is equivalently divided by eight. We measure the total execution time for extracting keypoints and their descriptors over 500 images of 1280×720 pixels, taken from all three environments. The mean number of identified keypoints on an image is over 980. The mean runtime for identifying keypoints on an image using one core is 2.826 seconds and the mean runtime for one phone transformation estimation is 0.488 second. In the prototype, the dual cameras take video recording at 5fps (i.e., ten images with at most six of them required) and a user should slide the phone slowly to guarantee the requirement of δ. In the future, we will consider to implement feature detection algorithms on mobile GPUs [12] to speed up HyperSight and to reduce the power consumption. It is also preferable to move such computation intensive tasks to edge servers if available.

6 LIMITATIONS

In this work, we introduce how to utilize HyperSight to obtain the depth information of objects. Indeed, HyperSight can resolve the 3D coordinates of far keypoints in the reference camera coordinate system, which can be a building block for many appealing applications such as creating 3D object and room models, and phone posture estimation.

Nevertheless, HyperSight also have the following four main limitations. First, HyperSight needs to see nearby objects in order to track the movement of the phone. This limits the usage scenarios of HyperSight, e.g., in the center of an empty hall. Second, HyperSight can be considered as an asynchronous camera array, and therefore it fits best in static scenarios where the target and most objects are not moving. Third, due to the residual camera calibration error, the transformation of a slide operation cannot be accurately estimated when the sliding distance is too long, which restricts the maximal operational distance of HyperSight. Last but not least, HyperSight requires both nearby objects and far objects to have sufficient identified keypoints, which further limits the usage of HyperSight in the scenarios with plain colors and textures. These limitations also direct our future work. As for the first limitation, in

case there is no enough nearby objects, helping objects like people can be temporally used.

7 RELATED WORK

7.1 Development of stereo vision system

In [3], a classical stereo vision system is introduced. In that system, extrinsic and intrinsic parameters of the two cameras are estimated through calibration offline. Then after rectification and stereo correspondence, depth information can be extracted from a pair of images taken by the two cameras. The classical stereo vision system is not applicable on smartphones because it requires a relatively long baseline to guarantee accuracy. Okutomi et al [23] study the effect of the length of baseline, and find a tradeoff between the precision of distance estimation and the accuracy of stereo matching. Based on their observation, they propose an algorithm that combines multiple baseline stereo pairs to overcome the tradeoff between accuracy and precision. However, the stereo pairs used in their algorithm are taken by a camera moving laterally, and a smartphone in hand is not able to get such pairs. Kawanishi et al [11] develop a panoramic stereo system using six cameras and a hexagonal pyramidal mirror. The key idea of their system is to specially arrange the cameras and the mirror so that the cameras have a fixed viewpoint virtually. Their system is hard to deploy on COTS devices.

7.2 Methods of 3D reconstruction

A category of schemes use RDB-D cameras to get 3D structures of objects or environments directly and reconstruct 3D scenes. Cui et al [4] propose a method for object scanning using depth maps taken from around an object with a time-of-flight camera. The key part of their work is to combine different 3D super-resolution techniques and take noise characteristics of the sensor into account. For an antique head, the model reconstructed by their method achieves a mean error of less than 1cm. Izadi et al [9] reconstruct 3D scenes of indoor environments using Kinect, a structure-light based camera. Their system utilizes depth information obtained from Kinect to estimate the pose of the sensor and build 3D models of the surrounding scene. Kerl et al [13] utilize dense depth information to estimate the pose of the sensor and achieve a mean absolute trajectory error of 3.4cm. These systems all require RGB-D sensors, which haven't been deployed on most of the COTS smartphones.

Another category of schemes attempt to reconstruct 3D scenes with a single camera. MonoSLAM [5] builds probabilistic 3D map with keypoints and utilizes Extended Kalman Filter (EKF) to update system states including camera motion parameters and location of keypoints. Their system uses only one single freely moving camera as the input source and achieves real-time SLAM. Klein et al [14] presents a method of camera pose tracking and mapping in small AR workspaces. Compared to other SLAM systems, their system uses denser but low-quality keypoints. ORB-SLAM [19] uses Oriented FAST and Rotated BRIEF (ORB) detector to detect keypoints and eliminates cumulative error by loop closing and relocalization. Maps built with only one single camera are in relative scale. Actually, one camera cannot get 3D information in absolute scale without the help of other sensors.

8 CONCLUSION

In this paper, we have proposed HyperSight, an depth estimation system on dual-camera smartphones. HyperSight overcomes the low accuracy issue of using the native short-baseline dual cameras to range a distant object, and hits an order of magnitude performance gain comparing with the state-of-the-art single camera system. HyperSight is implemented as software on COTS dual-camera devices and requires no other specialized sensors. We believe HyperSight's salient advantages will enable a wide range of 3D modeling and AR applications on mobile devices.

ACKNOWLEDGEMENTS

This work was partially supported by the National Key R&D Program of China (2018YFC1900700), National Natural Science Foundation of China (Grant No. 61772340, 61672151, 61420106010), Shanghai Rising-Star Program (Grant No.17QA1400100), and DHU Distinguished Young Professor Program.

APPENDICES

A BRIEF INTRODUCTION TO EPNP ALGORITHM

Given a set of n input points and their 3D coordinates in the world coordinate system $\{P_i^w\}_{i=1,\ldots,n}$, choose four control points with world coordinates $\{C_j^w\}_{j=1,\ldots,4}$ to express them

$$P_i^w = \sum_{j=1}^{4} \alpha_{ij} C_j^w \tag{9}$$

where $\sum_{j=1}^{4} \alpha_{ij} = 1$.

The relation also holds in the camera coordinate system, so the camera coordinates of these points can be expressed as

$$P_i^c = \sum_{j=1}^{4} \alpha_{ij} C_j^c \tag{10}$$

where superscript c represents the camera coordinate system.

According to the camera model, we have

$$w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_c \\ 0 & f_v & v_c \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha ij \begin{bmatrix} x_j^c \\ y_j^c \\ z_j^c \end{bmatrix} \tag{11}$$

where w_i is a scalar parameter, $[x_j^c, y_j^c, z_j^c]^T$ are the 3D coordinates of control points in the camera coordinate system and $[u_i, v_i]^T$ are 2D projections of the input points. f_u, f_v are focal length coefficients and (u_c, v_c) is the principal point of the camera, and they can be obtained by camera calibration.

From the last row of (11) we have

$$w_i = \sum_{j=1}^{4} \alpha_{ij} z_j^c \tag{12}$$

Then from the first two row of (11) we have

$$\sum_{j=1}^{4} \alpha_{ij} f_u x_j^c + \alpha_{ij}(u_c - u_i) z_j^c = 0 \tag{13}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_v y_j^c + \alpha_{ij}(v_c - v_i) z_j^c = 0 \tag{14}$$

Let $x = [C_1^{cT}, C_2^{cT}, C_3^{cT}, C_4^{cT}]^T$, and then we have

$$Mx = 0 \tag{15}$$

where the elements of M come from the coefficients of (13) and (14).

The solution therefore belongs to the null space of matrix M and can be expressed as

$$x = \sum_{i=1}^{N} \beta_i v_i \tag{16}$$

where v_i are the colums of the right-singular vectors of M corresponding to the N null singular values.

The next step is to choose the right linear combination of the vectors in the null space, i.e., to find the right β_i. Depending on the amount of noise, N can be 1, 2, 3 or 4. EPnP doesn't pick a value among 1,2,3 and 4. Instead, EPnP computes the solutions in all the four cases and then choose the one that leads to the smallest reprojection error,

$$res = \sum_i dist^2(K[R|t] \begin{bmatrix} P_i^w \\ 1 \end{bmatrix}, u_i) \tag{17}$$

where $dist(m, n)$ denotes the 2D distance between m and n, and K is the intrinsic matrix of the camera.

(1) Case $N = 1$: In this case, $x = \beta v$. Let $v^{[i]}$ be the sub-vector that corresponds to the coordinates of the control point C_i^c, then for all i and j we have the distance constraints

$$\|\beta v^{[i]} - \beta v^{[j]}\|^2 = \|C_i^w - C_j^w\|^2 \tag{18}$$

Thus β can be computed by

$$\beta = \frac{\sum_{\{i,j\} \in [1,4]} \|v^{[i]} - v^{[j]}\| \cdot \|C_i^w - C_j^w\|}{\sum_{\{i,j\} \in [1,4]} \|v^{[i]} - v^{[j]}\|^2} \tag{19}$$

(2) Case $N = 2$: In this case, $x = \beta_1 v_1 + \beta_2 v_2$, and for all i and j

$$\|(\beta_1 v_1^{[i]} + \beta_2 v_2^{[i]}) - (\beta_1 v_1^{[j]} + \beta_2 v_2^{[j]})\|^2 = \|C_i^w - C_j^w\|^2 \tag{20}$$

$\beta_1 and \beta_2$ can be computed by solving a linear system $L\beta = \rho$, where L is a 6×3 matrix formed with the elements of v_1 and v_2, ρ is a 6-vector with $\|C_i^w - C_j^w\|^2$. $\beta = [\beta_{11}, \beta_{12}, \beta_{22}]^T$ is the unknown vector, where $\beta_{11} = \beta_1^2, \beta_{12} = \beta_1\beta_2, \beta_{22} = \beta_2^2$.

(3) Case $N = 3$: Similar to the case $N = 2$, x can be computed by solving $L\beta = \rho$, where

$$\beta = [\beta_{11}, \beta_{12}, \beta_{13}, \beta_{21}, \beta_{22}, \beta_{33}]^T. \tag{21}$$

(4) Case $N = 4$: In this case, there are 6 constraints but 10 unknowns in the linear system $L\beta = \rho$, so we have to add extra constraints

$$\beta_{ab}\beta_{cd} = \beta_{a'b'}\beta_{c'd'} \tag{22}$$

where $\{a', b', c', d'\}$ represents any permutation of $\{a, b, c, d\}$.

With C_i^w and their coordinates in the camera coordinate system C_i^c, the translation t and rotation R of the camera coordinate system with respect to the world coordinate system can be solved.

REFERENCES

[1] Apple Inc. [n. d.]. ARKit. https://developer.apple.com/documentation/arkit. [Online; accessed July-2018].

[2] Andrew J Barry and Russ Tedrake. 2015. Pushbroom Stereo for High-speed Navigation in Cluttered Environments. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. 3046–3052.

[3] Gary Bradski and Adrian Kaehler. 2008. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media.

[4] Yan Cui, Sebastian Schuon, Derek Chan, Sebastian Thrun, and Christian Theobalt. 2010. 3D Shape Scanning with a Time-of-Flight Camera. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1173–1180.

[5] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. 2007. MonoSLAM: Real-time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29, 6 (2007), 1052–1067.

[6] Martin A Fischler and Robert C Bolles. 1987. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In *Readings in computer vision*. Elsevier, 726–740.

[7] Google Inc. [n. d.]. ARCore. https://developers.google.com/ar/. [Online; accessed July-2018].

[8] Haofu Han, Jiadi Yu, Hongzi Zhu, Yingying Chen, Jie Yang, Yanmin Zhu, Guangtao Xue, and Minglu Li. 2014. SenSpeed: Sensing driving conditions to estimate vehicle speed in urban environments. In *Proceedings of IEEE INFOCOM*.

[9] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. 2011. KinectFusion: Real-time 3D Reconstruction and Interaction using a Moving Depth Camera. In *Proceedings of ACM Symposium on User Interface Software and Technology*. 559–568.

[10] Andrew E Johnson, Steven B Goldberg, Yang Cheng, and Larry H Matthies. 2008. Robust and Efficient Stereo Feature Tracking for Visual Odometry. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. 39–46.

[11] Takahito Kawanishi, Kazumasa Yamazawa, Hidehiko Iwasa, Haruo Takemura, and Naokazu Yokoya. 1998. Generation of High-resolution Stereo Panoramic Images by Omnidirectional Imaging Sensor using Hexagonal Pyramidal Mirrors. In *Proceedings of IEEE International Conference on Pattern Recognition*, Vol. 1. 485–489.

[12] Guy-Richard Kayombya. 2010. *SIFT Feature Extraction on a Smartphone GPU Using OpenGLES2.0*. Master's thesis. Massachusetts Institute of Technology.

[13] Christian Kerl, Jurgen Sturm, and Daniel Cremers. 2013. Dense Visual SLAM for RGB-D Cameras. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2100–2106.

[14] Georg Klein and David Murray. 2007. Parallel Tracking and Mapping for Small AR Workspaces. In *Proceedings of IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR)*. 225–234.

[15] Kudan Inc. [n. d.]. Kudan. https://www.kudan.eu/. [Online; accessed July-2018].

[16] Thomas Lemaire, Cyrille Berger, Il-Kyun Jung, and Simon Lacroix. 2007. Vision-based SLAM: Stereo and Monocular Approaches. *International Journal of Computer Vision* 74, 3 (2007), 343–364.

[17] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. 2009. EPnP: An Accurate O(n) Solution to the PnP Problem. *International Journal of Computer Vision* 81, 2 (2009), 155.

[18] David G Lowe. 2004. Distinctive Image Features from Scale-invariant Keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110.

[19] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardós. 2015. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics* 31, 5 (2015), 1147–1163.

[20] Raul Mur-Artal and Juan D Tardós. 2017. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262.

[21] Raúl Mur-Artal and Juan D. Tardós. 2017. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *IEEE Transactions on Robotics* 33, 5 (2017), 1255–1262.

[22] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. 2011. KinectFusion: Real-time Dense Surface Mapping and Tracking. In *Proceedings of IEEE international symposium on Mixed and Augmented Reality (ISMAR)*. 127–136.

[23] Masatoshi Okutomi and Takeo Kanade. 1993. A Multiple-baseline Stereo. *IEEE Transactions on pattern analysis and machine intelligence* 15, 4 (1993), 353–363.

[24] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. 1999. Bundle Adjustment - A Modern Synthesis. In *Proceedings of International Workshop on Vision Algorithms*. 298–372.

[25] Wikitude GmbH Inc. [n. d.]. Wikitude. https://www.wikitude.com/. [Online; accessed July-2018].

[26] Jiadi Yu, Hongzi Zhu, Haofu Han, Yingying Chen, Jie Yang, Yanmin Zhu, Zhongyang Chen, Guangtao Xue, and Minglu Li. 2016. SenSpeed: Sensing Driving Conditions to Estimate Vehicle Speed in Urban Environments. *IEEE Transactions on Mobile Computing* 15, 1 (2016), 202–216.

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