360ViewPET: View Based Pose EsTimation for Ultra-Sparse 360-Degree Cameras

Qian Zhou, Bo Chen, Zhe Yang, Hongpeng Guo and Klara Nahrstedt
Department of Computer Science, University of Illinois Urbana-Champaign
Email: {qianz, boc2, zheyang3, hg5, klara}@illinois.edu

***Abstract*—Immersive virtual tours based on 360-degree cameras, showing famous outdoor scenery, are becoming more and more desirable due to travel costs, pandemics and other constraints. To feel immersive, a user must receive the view accurately corresponding to her position and orientation in the virtual space when she moves inside, and this requires cameras' orientations to be known. Outdoor tour contexts have numerous, ultra-sparse cameras deployed across a wide area, making camera pose estimation challenging. As a result, pose estimation techniques like SLAM, which require mobile or dense cameras, are not applicable. In this paper we present a novel strategy called 360ViewPET, which automatically estimates the relative poses of two stationary, ultra-sparse (15 meters apart) 360-degree cameras using one equirectangular image taken by each camera. Our experiments show that it achieves accurate pose estimation, with a mean error as low as 0.9 degree.**

I. Introduction

Watching a video produced by a camera assigns us the view at the camera's position, making us virtually travel there. Multiple 360° cameras provide the views at multiple positions and in arbitrary directions, and can be used for immersive virtual tourism. Virtual tours are preferred to physical ones for those constrained by time and travel costs or mobility impairments; during pandemics, they become further appreciated due to travel restrictions and social distancing.

Specifically, we envision a graph comprised of multiple 360° cameras which keep streaming 360° videos to the cloud: each camera is a vertex which resembles a tour station; an edge exists between two adjacent cameras with a line of sight and it resembles a tour road. A user travels in the virtual space along a tour route (i.e. a sequence of connected edges) which is defined statically (e.g., travel companies plan professional routes beforehand) or dynamically (e.g., tourists reach one station and decide next stop). Ideally, a virtual tourist provides the cloud with her virtual position (which spot of which road) and orientation relative to the road, and receives the corresponding view *in real-time*. Thus, our envisioned system aims at new ***spatially-temporally immersive experience*** far superior to that of a single 360° camera based system which fixes user views at one position, and that of a prerecorded image/video based system which fixes user views at one moment (e.g., Google Street View [1] uses vehicles or people called Trekkers to prerecord views, so user views have no update or, at best, occasional update over time).

So back to our envisioned system. First, given a user's virtual position, it is easy for the cloud to find which camera should be selected for user view generation. Second, the cloud generates user view by cropping the camera's 360° view to a subview according to the user's virtual orientation and field of view ($\sim$120°). E.g., if she is virtually at Station $\mathcal{A}$ of Road $\mathcal{A}$-$\mathcal{B}$ with orientation 0°, then Camera $\mathcal{A}$'s 120° subview toward the road forward will be her view.

However, the cropping step cannot be completed if the cloud only has the user's virtual position and orientation; cameras' orientations relative to roads are indispensable (elaborated in Section II-B). Camera poses can be manually measured at installation by professional teams with instruments (e.g., in urban planning), but it is cumbersome. Besides, camera pose estimation is well studied in computer vision, particularly using SLAM [2], [3], [4], but it requires mobile cameras which record tens of frames per second while moving. Unfortunately, in our outdoor tour contexts, cameras should be deployed on a large scale, across a wide area to offer users rich scenery. Thus, they are stationary, numerous and ultra-sparse (tens of meters apart, elaborated in Section II-C). Since they are numerous, manual measurement becomes a huge workload; and because they are ultra-sparse, a SLAM strategy can get only 0.1 frame per second in average, and will fail. An automatic pose estimation approach that works for ultra-sparse 360° cameras is requisite, challenging and unexplored.

Realizing that the envisioned immersive virtual tourism system requires a series of problems to be solved, in this paper we take one step toward it by presenting **360ViewPET**, a strategy for automatic **View** based **P**ose **EsT**imation of ultra-sparse **360**° cameras. We only require each camera to upload an equirectangular view to 360ViewPET running on the cloud, which will find the relative poses of two cameras by searching a hallmark pattern in their views; such pose information is computed once (unless deployment changes), stored and used over and over for user view generation. 360ViewPET eliminates the need of manual measurement, thus essentially simplifies the deployment of virtual tourism systems; also, it achieves accurate pose estimation for 360° cameras which are up to **15 m apart**. To the best of our knowledge, no existing visual pose estimation approach applies to such sparse cameras. We claim our contributions as follows:

1) We propose 360ViewPET for pose estimation of two 360° cameras using one image taken by each. It uses a novel vision based strategy which finds poses via searching a hallmark pattern in cameras' views, and works for ultra-sparse cameras up to **15 m apart**.

978-1-6654-3734-9/21/$31.00 ©2021 IEEE
DOI 10.1109/ISM52913.2021.00008

2) We additionally use multiple ways to dramatically reduce the hallmark pattern search space and improve efficiency. It shortens the time of one estimation operation from tens of hours to **5 sec** when running on a laptop.
3) We collect real data from 6 outdoor tour routes with 3 to 4 360° cameras, up to 15 m apart, deployed in each. Our extensive experiments show that solutions based on SLAM or image registration totally fail for such sparse cameras, while 360ViewPET not only works but also achieves a mean error as low as **0.9**°.

II. Assumptions & Problem Description

A. Tour Mode

We assume a graph comprised of numerous ultra-sparse (tens of meters apart) 360° cameras (intrinsics are not needed) streaming 360° videos to the cloud. Each camera is a vertex resembling a tour station; an edge exists between two adjacent cameras with a line of sight, and it resembles a tour road. A user travels in the virtual space along a tour route (i.e. a sequence of connected edges) made beforehand or on demand.

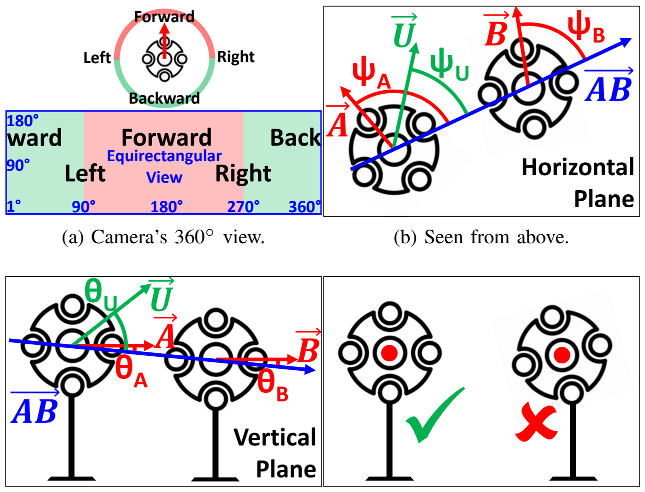
Our long-term goal is an immersive virtual tourism system which detects a user's position and orientation in the virtual space, and presents her with the corresponding user view in real-time. It aims at much higher immersion than existing systems like Google Street View (more detail in the second paragraph of Section I). In this paper, we assume that the user's virtual position (which spot of which road) is provided to the cloud, so the cloud knows which camera should be selected for user view generation. Her virtual orientation relative to the road is also provided, and the cloud crops the camera's 360° view to a subview based on her orientation and field of view (FoV), getting her view. We show in Section II-B that the cropping step cannot be completed without cameras' orientations relative to roads. It justifies that our work—camera pose estimation—is one of the many necessary steps toward the immersive virtual tourism system.

B. Relation Between Camera's 360° View and User View

Camera's View. As shown in Fig. 1a, though a 360° camera looks spherical, it also has a physical front and thus a forward direction like a traditional camera. In this paper, a 360° camera's orientation refers to its forward direction, and is marked with a red arrow in figures.

Each 360° video frame is an equirectangular view whose x-coordinate spans from 1° to 360° and y-coordinate from 1° to 180°. An object's coordinates in a camera's view depend on its position relative to the camera. E.g., those in the camera's forward direction appear in the middle of the view, with x-coordinate 180°; those in the backward, left, right direction have x-coordinate 360°, 90°, 270°. We show the universal mapping relation as below, and as an example, the user is assumed to be virtually at Camera $\mathcal{A}$'s position:

1) Yaw. Fig. 1b shows the look of a camera pair $\mathcal{A}$ and $\mathcal{B}$ seen from above, and the projections of four vectors on a horizontal plane, including $\mathcal{A}$'s orientation $\vec{A}$, $\mathcal{B}$'s orientation $\vec{B}$, the user's orientation $\vec{U}$, and the road direction $\overrightarrow{AB}$.



(a) Camera's 360° view. (b) Seen from above.

(c) Seen from right. (d) Seen from front.

Fig. 1: 360° camera's view and yaw, pitch, roll.

Yaw ψ_A, ψ_B, ψ_U is the angle made by $\vec{A}$, $\vec{B}$, $\vec{U}$ with $\overrightarrow{AB}$ respectively. ψ_U is known (an assumption in Section II-A) while the other two are unknown.

Map User Orientation to View (x-coordinates). Because $\vec{A}$ and $\vec{U}$ have an angle difference ψ_A-ψ_U, the objects in the user's forward direction have x-coordinate 180°+ψ_A-ψ_U in $\mathcal{A}$'s view; if a person's horizontal FoV is denoted as FoV_x ($\sim$120°), the user view is the subview of $\mathcal{A}$'s view with x-coordinate 180°+ψ_A-$\psi_U \pm \frac{FoV_x}{2}$. Thus, to get the user view, ψ_A is required besides ψ_U (known). If the user is at Camera $\mathcal{B}$'s position, we replace ψ_A with ψ_B to find her view.

So ψ_A and ψ_B are needed. If the installer simply installs cameras at planned locations without paying attention to their orientations, ψ_A and ψ_B are independent and arbitrary values in [1°, 360°]. We define $\Delta\psi$=ψ_A-ψ_B. Note that estimating ($\Delta\psi$, ψ_B) is equivalent to estimating (ψ_A, ψ_B).

2) Pitch. Fig. 1c shows the look seen from right, and the projections on the vertical plane where $\mathcal{A}$ and $\mathcal{B}$ coexist. Pitch θ_A, θ_B, θ_U is the angle made by $\vec{A}$, $\vec{B}$, $\vec{U}$ with $\overrightarrow{AB}$ respectively. θ_U is known while the other two not.

Map User Orientation to View (y-coordinates). Because $\vec{A}$ and $\vec{U}$ have an angle difference θ_U-θ_A, the objects in the user's forward direction have y-coordinate 90°+θ_U-θ_A in $\mathcal{A}$'s view; if a person's vertical FoV is denoted as FoV_y ($\sim$90°), the user view is the subview of $\mathcal{A}$'s view with y-coordinate 90°+θ_U-$\theta_A \pm \frac{FoV_y}{2}$. If the user is at Camera $\mathcal{B}$'s position, we replace θ_A with θ_B to find her view.

So θ_A and θ_B are needed too. We define $\Delta\theta$=θ_A-θ_B. Note that estimating ($\Delta\theta$, θ_B) is equivalent to estimating (θ_A, θ_B). Theoretically, θ_A, θ_B can be arbitrary in [-90°, 90°].

Double Alignment. If $\vec{A}$, $\overrightarrow{AB}$ have the same direction, and $\vec{B}$, $\overrightarrow{AB}$ have the same direction, i.e., $\Delta\psi$=ψ_B=$\Delta\theta$=θ_B=0°, we say the cameras are double aligned.

3) Roll. Fig. 1d shows the look seen from front, and a

camera's orientation vector is perpendicular to the reader, appearing as a red point. In this paper we assume that a camera has no significant rotation around its orientation vector, which is a common requirement and easily guaranteed in practice.

So far, we have shown that four pose angles ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B) must be estimated for user view generation.

C. Problems and Challenges

In outdoor tour contexts, cameras should be deployed on a large scale, across a wide area to offer users rich scenery. It means that the cameras are:

1) numerous. Manually measuring numerous camera poses is burdensome, so an automatic estimation approach is crucial.

2) ultra-sparse. Only ultra-sparse cameras (tens of meters apart, like streetlights) can be widely deployed. SLAM does not work in this case: it requires a mobile camera to record 10 to 30 frames per second while moving [2], [3], [4]; let's assume that the camera speed is 1.5 m/s (near humans' typical walking speed), then SLAM requires a frame every 5 to 15 cm; our camera distance is 15 m, at least 100 times sparser.

An automatic pose estimation approach that works for such ultra-sparse 360° cameras is requisite but has not been studied.

III. 360ViewPET Overview

360ViewPET takes V_A and V_B—the 360° equirectangular views of a camera pair $\mathcal{A}$ and $\mathcal{B}$ (tens of meters apart with a line of sight)—as input, and outputs the pose angles ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B) of $\mathcal{A}$ and $\mathcal{B}$.

It estimates ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B) based on the fact that a hallmark pattern of feature correspondences will appear in the overlay of V_A and V_B *iff* $\mathcal{A}$ and $\mathcal{B}$ are double aligned, i.e. ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B) = (0°, 0°, 0°, 0°). Also, note that by rotating V_A and V_B using ($\Delta\psi'$, ψ_B', $\Delta\theta'$, θ_B'), we can get the views of a new camera pair $\mathcal{A}'$ and $\mathcal{B}'$ whose pose angles are ($\Delta\psi$-$\Delta\psi'$, ψ_B-ψ_B', $\Delta\theta$-$\Delta\theta'$, θ_B-θ_B'). Specifically,

1) 360ViewPET rotates V_A and V_B using N different ($\Delta\psi_i'$, ψ_{B_i}', $\Delta\theta_i'$, θ_{B_i}') combinations, $i = 1, 2, \ldots, N$, getting the views of N different $\mathcal{A}_i'$ and $\mathcal{B}_i'$ pairs whose pose angles are ($\Delta\psi$-$\Delta\psi_i'$, ψ_B-ψ_{B_i}', $\Delta\theta$-$\Delta\theta_i'$, θ_B-θ_{B_i}').
2) For each $\mathcal{A}_i'$ and $\mathcal{B}_i'$, 360ViewPET detects if the hallmark pattern appears in their view overlay. Assume that $\mathcal{A}_k'$ and $\mathcal{B}_k'$ have the pattern, then we know they are double aligned, i.e. ($\Delta\psi$-$\Delta\psi_k'$, ψ_B-ψ_{B_k}', $\Delta\theta$-$\Delta\theta_k'$, θ_B-θ_{B_k}') = (0°, 0°, 0°, 0°). Then ($\Delta\psi_k'$, ψ_{B_k}', $\Delta\theta_k'$, θ_{B_k}') are output as the estimated values of ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B).
3) 360ViewPET also involves strategies to reduce N, shortening the time of pose estimation without accuracy loss.

In the following sections, we introduce the hallmark pattern and its detection, and then elaborate 360ViewPET's workflow.

IV. Hallmark Pattern of Double Alignment

A hallmark pattern of feature correspondences will appear in the overlay of V_A and V_B *iff* Camera $\mathcal{A}$ and $\mathcal{B}$ are double aligned. Thus, if the pattern is detected, we know the four pose angles of $\mathcal{A}$ and $\mathcal{B}$ are all 0°.

A. Cameras' Views When Double Aligned

First we introduce a special visual phenomenon when cameras are double aligned. Imagine that a person physically moves from Camera $\mathcal{A}$'s position to $\mathcal{B}$'s position along $\overrightarrow{AB}$ and faces the direction of $\overrightarrow{AB}$, then:

1) the objects in front of her will expand in her eyes;
2) the objects behind her will contract to her;
3) those to her exact left or right will move parallel to her.

If Camera $\mathcal{A}$ and $\mathcal{B}$ face the direction of $\overrightarrow{AB}$ (i.e. double aligned), then $\mathcal{A}$ resembles the person before moving, and $\mathcal{B}$ resembles her after moving. Thus the visual phenomenon will appear in V_A and V_B (Fig. 2): object expansion in the middle (x-coordinate: 180°); contraction on the left and right margins (x: 0° and 360°); parallel movement between the middle and margins (x: 90° and 270°).

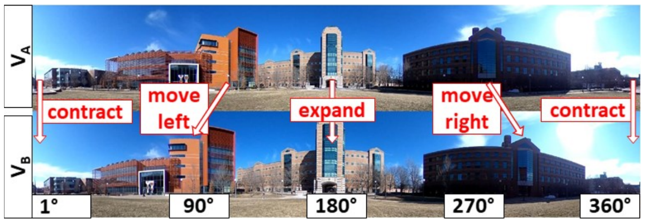


Fig. 2: Cameras' views when double aligned.

B. Feature Correspondences When Double Aligned

Feature Arrow (FA). Feature detection (e.g., SURF [5], ORB [6]) and matching techniques automatically detect features in two images and pair the corresponding ones. A feature arrow (FA) is defined as an arrow (denoted as $\overrightarrow{p_Ap_B}$) drawn in the overlay of V_A and V_B, with its start point p_A (marked with ○) at a feature in V_A and end point p_B (marked with +) at the corresponding feature in V_B (Fig. 3). Like a vector, an FA's angle is the one it makes with the positive x-axis.

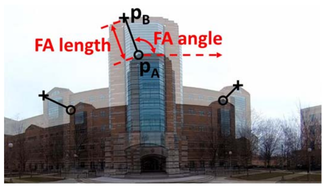


Fig. 3: Feature arrow in view overlay.

Hallmark FA. The visual phenomenon corresponds to the FA pattern in Fig. 4, called the *hallmark pattern of double alignment*. An FA matching this pattern is a *hallmark FA*:

1) View expansion. FA $\overrightarrow{p_Ap_B}$ radiates outward from the middle center p_C (i.e., $\angle\overrightarrow{p_Ap_B} \approx \angle\overrightarrow{p_Cp_A}$), if x_A (p_A's x-coordinate) is near 180°.
2) View contraction. $\overrightarrow{p_Ap_B}$ radiates inward to the left margin center p_L (i.e., $\angle\overrightarrow{p_Ap_B} \approx \angle\overrightarrow{p_Ap_L}$), if x_A is near 0°; it radiates inward to the right margin center p_R (i.e., $\angle\overrightarrow{p_Ap_B} \approx \angle\overrightarrow{p_Ap_R}$), if x_A is near 360°.

3) Parallel movement. $\overrightarrow{p_A p_B}$ is left (angle: 180°) if x_A is near 90°, and right (angle: 0°) if x_A is near 270°.

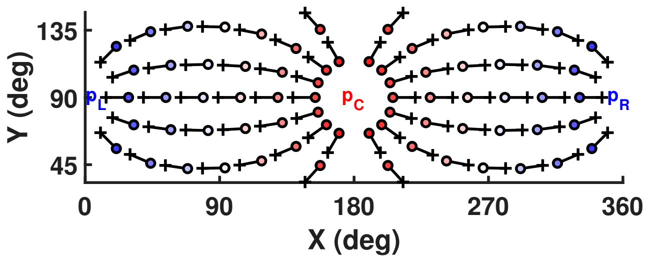


Fig. 4: Hallmark pattern of double alignment. ○: p_A, +: p_B.

As for an FA with $x_A \in$ (180°, 270°), its angle should be a weighted mean of $\angle\overrightarrow{p_C p_A}$ and 0°. And the closer it is to the center p_C, the more weight is given to $\angle\overrightarrow{p_C p_A}$. A similar rule applies to FAs with $x_A \in$ (0°, 90°), (90°, 180°), (270°, 360°). We find that a simple linear interpolation works well enough. The general formula of a hallmark FA's angle α is:

$$\alpha = \begin{cases} (1-\lambda)\cdot\angle\overrightarrow{p_A p_L} + \lambda\cdot 180^\circ & \text{if } 0^\circ \le x_A < 90^\circ \\ (1-\lambda)\cdot 180^\circ + \lambda\cdot\angle\overrightarrow{p_C p_A} & \text{if } 90^\circ \le x_A < 180^\circ \\ (1-\lambda)\cdot\angle\overrightarrow{p_C p_A} + \lambda\cdot 0^\circ & \text{if } 180^\circ \le x_A < 270^\circ \\ (1-\lambda)\cdot 0^\circ + \lambda\cdot\angle\overrightarrow{p_A p_R} & \text{if } 270^\circ \le x_A < 360^\circ \end{cases} \quad (1)$$

$$\lambda = \frac{x_A \bmod 90^\circ}{90^\circ} \quad (2)$$

λ is a weight increasing from 0 to 1 as x_A increases from the left endpoint to the right endpoint, in interval [0°, 90°), [90°, 180°), [180°, 270°) or [270°, 360°).

V. 360ViewPET Workflow

As introduced in Section III, Camera $\mathcal{A}$ and $\mathcal{B}$ have unknown pose angles ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B), and 360ViewPET aims to find those angles based on their view V_A and V_B.

Assume that there are N different ($\Delta\psi'_i$, $\psi_B{}'_i$, $\Delta\theta'_i$, $\theta_B{}'_i$) combinations (see combination selection in Section V-C), $i = 1, 2, \ldots, N$. For each combination, 360ViewPET:

1) rotates $\mathcal{A}$ and $\mathcal{B}$ using the combination, getting the views and FAs of a new camera pair $\mathcal{A}'_i$ and $\mathcal{B}'_i$ whose pose angles are ($\Delta\psi$-$\Delta\psi'_i$, ψ_B-$\psi_B{}'_i$, $\Delta\theta$-$\Delta\theta'_i$, θ_B-$\theta_B{}'_i$);
2) counts the number of hallmark FAs.

The camera pair with the most hallmark FAs is regarded as double aligned. Assume that the pair is $\mathcal{A}'_k$ and $\mathcal{B}'_k$, then we know ($\Delta\psi$-$\Delta\psi'_k$, ψ_B-$\psi_B{}'_k$, $\Delta\theta$-$\Delta\theta'_k$, θ_B-$\theta_B{}'_k$) = (0°, 0°, 0°, 0°). Finally, ($\Delta\psi'_k$, $\psi_B{}'_k$, $\Delta\theta'_k$, $\theta_B{}'_k$) are output as the estimated values of ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B).

A. Obtain Feature Arrows

Search Space. We first focus on introducing functionality, and temporarily use a huge 4D search space: because $\Delta\psi$, $\psi_B \in$ [1°, 360°] and $\Delta\theta$, $\theta_B \in$ [-90°, 90°] (Section II-B), every ($\Delta\psi'$, $\psi_B{}'$, $\Delta\theta'$, $\theta_B{}'$) combination in [1°, 360°]^2×[-90°, 90°]^2 is tried. Our search granularity is 1°, so all the four are integers. Under this condition, $N \approx 4$ billion.

Given a ($\Delta\psi'$, $\psi_B{}'$, $\Delta\theta'$, $\theta_B{}'$) combination, we need to get the views of Camera $\mathcal{A}'$ and $\mathcal{B}'$ whose pose angles are ($\Delta\psi$-$\Delta\psi'$, ψ_B-$\psi_B{}'$, $\Delta\theta$-$\Delta\theta'$, θ_B-$\theta_B{}'$). But instead of physically rotating $\mathcal{A}$ and $\mathcal{B}$ by $\Delta\psi'$+$\psi_B{}'$, $\psi_B{}'$ horizontally and $\Delta\theta'$+$\theta_B{}'$, $\theta_B{}'$ vertically, which is unfeasible or burdensome in practice, we just need to rotate the cameras' views—V_A and V_B.

Horizontal Rotation. To get the view of a camera physically rotated right by ψ', we just need to rotate the original view left by ψ'. Specifically, the original view's subview with x-coordinate [1°, ψ'] is cut and appended to the right margin. Since 1° and 360° are adjacent, the new view is seamless.

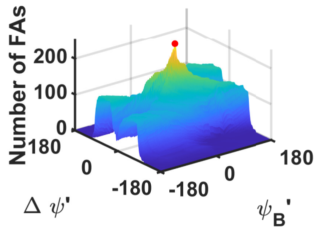
Vertical Rotation. If a camera physically rotates down by θ', the objects in front of it will move up by θ' in its view while those behind it will move down by θ'.

Feature Detection & Matching. Recall that it is FAs that are used for finding double alignment. There are two ways to get the FAs of a camera pair's views. 1) rotate and detect: for each ($\Delta\psi'$, $\psi_B{}'$, $\Delta\theta'$, $\theta_B{}'$) combination, we rotate V_A, V_B accordingly to get new views, then call a feature detection and matching algorithm to get FAs. 2) detect and rotate: we call a feature detection and matching algorithm to get the FAs of the original V_A, V_B for only once before the search starts, and for each combination, we just rotate the feature points accordingly to get new FAs. The first strategy needs feature detection and matching to be called for every test; the second strategy constantly needs one, so we use this way.

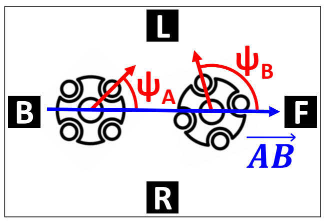
B. Count Hallmark Feature Arrows

Given a set of FAs, we need to count how many of them match the hallmark pattern of double alignment in Fig. 4. An FA is regarded as a hallmark one if the difference between its angle and the hallmark FA angle α (Formula (1)) is less than a threshold TH_α (e.g., 20°, see evaluation in Section VI-C).

Once the two steps above are performed for all the combinations, the combination leading to the most hallmark FAs becomes known. Since it makes the hallmark pattern most obvious, it is regarded to have made a double aligned camera pair. Fig. 5a shows an example of the number of hallmark FAs when traversing $\Delta\psi'$ and $\psi_B{}'$ (with fixed $\Delta\theta'$ and $\theta_B{}'$). We see a remarkable peak: the double alignment state results in many more hallmark FAs than unaligned states, thus the detection of it is easy and accurate.



(a) Count hallmark FAs.



(b) Image registration.

Fig. 5: Count hallmark feature arrows & reduce search space.

C. Reduce Search Space

There are $N \approx 4$ billion ($\Delta\psi'$, ψ_B', $\Delta\theta'$, θ_B') combinations in [1°, 360°]^2×[-90°, 90°]^2, and a brute-force 4D search is unacceptably slow. To speed it up, we tried to use only a subset of randomly selected combinations, or probabilistic algorithms like simulated annealing, but they cannot always find the optimal combination that leads to the most hallmark FAs. Here we introduce how to reduce N by ~20k times *without missing the optimal combination*, then a brute-force search can be used to find the combination quickly.

Approach for $\Delta\psi$ and $\Delta\theta$ Scope Reduction. $\Delta\psi$=ψ_A-ψ_B is the yaw difference of two cameras, and can be any value in [1°, 360°]. To narrow down its scope, one may think of paying some attention during camera deployment, to make $\mathcal{A}$ and $\mathcal{B}$ roughly face the same direction (e.g., north) so later we only need to search $\Delta\psi$ around 0°. This requires extra equipment (e.g., magnetometer) and non-negligible manual work. Instead, our strategy is to use *image registration* to find the rough values of $\Delta\psi$ and $\Delta\theta$ first, denoted as $\Delta\tilde{\psi}$, $\Delta\tilde{\theta}$. If $\Delta\tilde{\psi}$, $\Delta\tilde{\theta}$ have a max error of ϵ, then 360ViewPET only needs to search $\Delta\psi$, $\Delta\theta$ in [$\Delta\tilde{\psi}$-ϵ, $\Delta\tilde{\psi}$+ϵ], [$\Delta\tilde{\theta}$-ϵ, $\Delta\tilde{\theta}$+ϵ]. Our experiments in Section VI-B show that $\epsilon < 5°$.

Image registration [7] automatically discovers the correspondences (e.g., using feature matching) of images which capture the same scene from different viewports, and properly stitches them. E.g., based on feature matching, if an object is found to appear in both V_A and V_B, and with x-coordinate x_A and x_B respectively, it implies that V_A should be shifted left by an offset Δx=x_A-x_B and stitched to V_B.

Δx is useful to us because it roughly equals $\Delta\psi$. As illustrated in Fig. 5b, an object F in front of $\overrightarrow{AB}$ will appear in V_A with x_A=180°+ψ_A and in V_B with x_B=180°+ψ_B, thus x_A-x_B=ψ_A-ψ_B, i.e. Δx=$\Delta\psi$. It is also true if a backward object B is used. However, it can be easily proved (details omitted due to space limitation) that a left object L has $\Delta x > \Delta\psi$ and a right object R has $\Delta x < \Delta\psi$. So they are not always equal and $\Delta\psi$ cannot be accurately obtained.

Facing this issue, we evenly cut V_A and V_B to multiple patches horizontally; for each patch we compute a Δx, and use their mean as a guess of $\Delta\psi$. The intuition is that a left object's overestimated $\Delta\psi$ will cancel out a right object's underestimated $\Delta\psi$ to some degree, reducing the error.

Approach 2 for $\Delta\theta$ Scope Reduction. Here is another way to narrow down $\Delta\theta$ scope besides image registration: if each camera's orientation vector is made roughly horizontal to the ground during deployment (actually people spontaneously do it in practice), then $\Delta\theta$ will be close to 0°, and searching it in [-2°, 2°] is mostly sufficient. Note that making cameras roughly horizontal is significantly easier and more natural than making them all face north, so scope reduction via conscious camera deployment is suitable for $\Delta\theta$ but not $\Delta\psi$.

Approach for θ_B Scope Reduction. If $\mathcal{A}$ and $\mathcal{B}$ both have horizontal orientations, then θ_B=$\arctan(\frac{h}{d})$ where d is their location distance and h is height difference. In our context, two cameras are at most 15 m apart, and drastic ground height changes are not likely to happen within such a distance. When d = 15 m and $\theta_B \in$ [-5°, 5°], we have $h \in$ [-1.3m, 1.3m]. It means that searching $\theta_B \in$ [-5°, 5°] can make $\mathcal{A}$ find and get double aligned with a $\mathcal{B}$ which is 1.3 m higher or lower than it. This should suffice most cases in practice.

Summary. With these techniques and heuristics, we can reduce the overall search number from $N = 360^2 \times 180^2$ (above 4 billion) to $11 \times 360 \times 5 \times 11$ (~200k, with ϵ = 5°, $\Delta\theta \in$ [-2°, 2°], $\theta_B \in$ [-5°, 5°]), by ~20k times.

VI. Evaluation

We conduct extensive experiments to evaluate our 360ViewPET and two existing approaches of camera pose estimation, using real data collected from 360° cameras which are 15 m apart. As shown in Table I, we find that: SLAM totally fails for such sparse cameras; image registration only works out two pose angles; 360ViewPET solves all the four angles, with an overall mean error as low as 0.9°.

TABLE I: Pose estimation performance comparison.

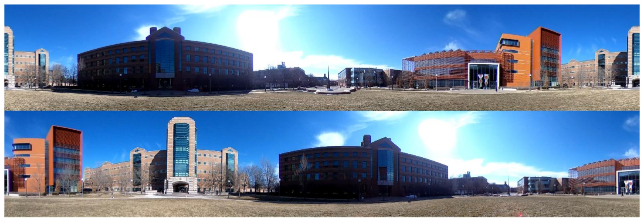
	SLAM	**ImageRegistration**	**360ViewPET**
Error of $\Delta\psi$, $\Delta\theta$	N/A	1.7°, 0.8°	0°, 0.8°
Error of ψ_B, θ_B	N/A	N/A	0.9°, 1.9°

Settings. We collect data from RICOH THETA 360° cameras in 6 outdoor tour routes. The first 4 tours have 4 cameras each, deployed along a straight line (some are north-south and others are east-west), hence 3 adjacent camera pairs (C_1-C_2, C_2-C_3, C_3-C_4, all are 15 m apart). The last 2 tours have 3 cameras each, located on the vertices of a triangle, also 3 adjacent camera pairs (C_4-C_2, C_2-C_3, C_3-C_4). Therefore, we have 18 diverse test cases with different objects in different directions. Additionally, we manipulate the data by removing some FAs from them, to generate more and harsher test cases.

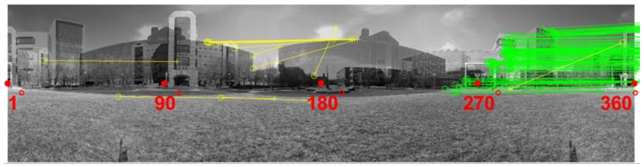
For each case, we denote the first camera as $\mathcal{A}$ and the second as $\mathcal{B}$, and two synchronized frames V_A, V_B are extracted from the videos of $\mathcal{A}$ and $\mathcal{B}$ respectively. Each frame has 3840×1920 pixels, for horizontal [1°, 360°] and vertical [1°, 180°]. We only use the vertical pixels in [45°, 135°] for pose estimation because others are near polar and distorted, so a cropped frame spans 360° horizontally and 90° vertically.

A. Qualitative Evaluation of 360ViewPET

We use the example below to demonstrate the effect and result of 360ViewPET. Fig. 6a shows the initial views of $\mathcal{A}$ and $\mathcal{B}$; they are unaligned and only 55 hallmark FAs in their view overlay are found (Fig. 6b). 360ViewPET tries different ($\Delta\psi'$, ψ_B', $\Delta\theta'$, θ_B') combinations, and when using (263°, 290°, 0°, 0°), the views are better aligned (Fig. 7a) and the number of hallmark FAs rises to 170 (Fig. 7b). The best aligned views (Fig. 8a) and the most hallmark FAs (Fig. 8b) appear when (258°, 285°, 0°, 0°) are used, so they are output as the estimated pose angles. The ground truth pose angles are (258°, 284°, 2°, -1°), very close to the output.

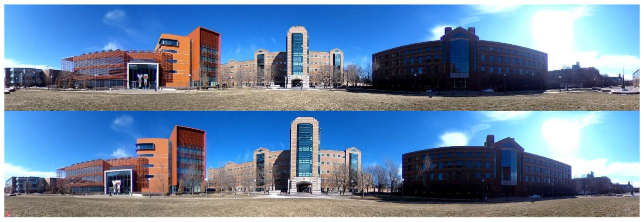


(a) Initial view V_A and V_B.

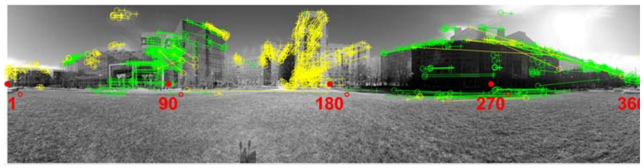


(b) 55 hallmark (green) FAs in view overlay; yellow FAs are non-hallmark.

Fig. 6: Initially unaligned cameras.

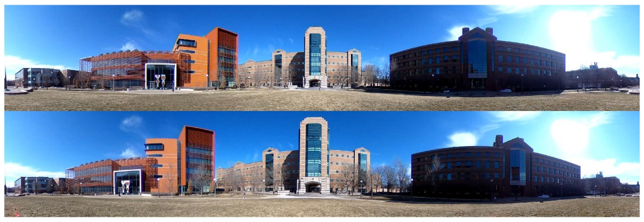


(a) Views when ($\Delta\psi'$, ψ_B', $\Delta\theta'$, θ_B') = (263°, 290°, 0°, 0°).

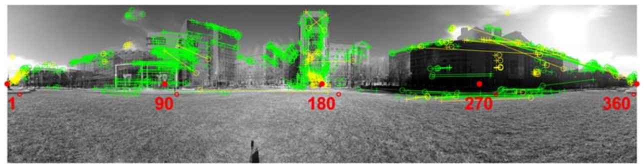


(b) 170 hallmark (green) FAs in view overlay; yellow FAs are non-hallmark.

Fig. 7: Near double aligned cameras.

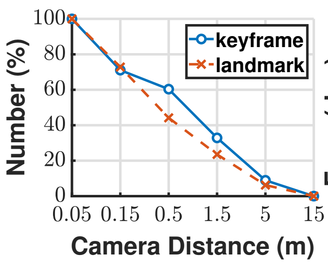


(a) Views when ($\Delta\psi'$, ψ_B', $\Delta\theta'$, θ_B') = (258°, 285°, 0°, 0°).

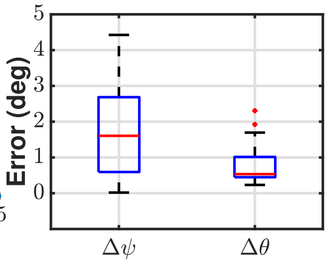


(b) 252 hallmark (green) FAs in view overlay.

Fig. 8: Double aligned cameras.



(a) SLAM.



(b) Image registration.

Fig. 9: Existing approaches fail for ultra-sparse cameras.

B. Quantitative Evaluation of Existing Approaches

SLAM. We use our 360° dataset to test OpenVSLAM [4], which is claimed to be the first open-source visual SLAM framework that can accept equirectangular imagery. The distance between two cameras is 0.05, 0.15, 0.5, 1.5, 5, 15 m, corresponding to frame rate 30, 10, 3, 1, 0.3, 0.1 FPS respectively. Fig. 9a shows the number of landmarks and keyframes that are recognized for map construction. We normalize the data and make the numbers be 100% when the frame rate is 30 FPS (5 cm). We see that the numbers rapidly deteriorate as the camera distance increases. SLAM strategies mostly use 10 to 30 FPS (camera distance 5 to 15 cm) [2], [3], [4]. When the distance is 15 m, no landmark or keyframe can be recognized at all, leading to an empty map, so the strategy totally fails to work for ultra-sparse cameras.

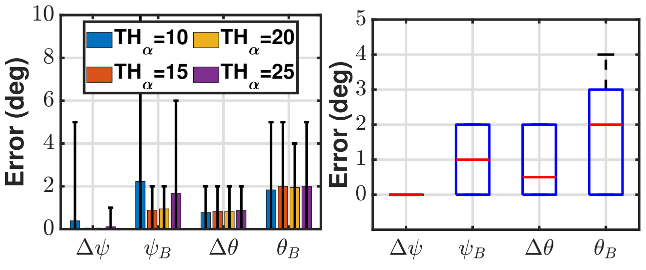
Image Registration. As explained in Section V-C, V_A and V_B are cut into multiple patches (here we use 12), a Δx is computed for each patch, and their mean is used as a guess of $\Delta\psi$. Besides, RANSAC is used for outlier FA elimination. It works out $\Delta\psi$ (mean error 1.7°; max error 4.4°) and $\Delta\theta$ (mean error 0.8°; max error 2.3°), but not ψ_B or θ_B (Fig. 9b).

C. Quantitative Evaluation of 360ViewPET

Since the image registration technique's output $\Delta\tilde{\psi}$ has max error 4.4°, 360ViewPET only needs to search ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B) in [$\Delta\tilde{\psi}$-5°, $\Delta\tilde{\psi}$+5°]×[1°, 360°]×[-2°, 2°]×[-5°, 5°] (Section V-C). It has mean error (0°, 0.9°, 0.8°, 1.9°) and max error (0°, 2°, 2°, 4°) for the four angles, with an overall mean error and max error as low as 0.9° and 2° respectively.

Impact of Threshold TH_α. Fig. 10a shows the impact of TH_α, the threshold of detecting hallmark FAs (Section V-B). A too small threshold (e.g., 10°) makes many hallmark FAs detected as non-hallmark; a too large threshold (e.g., 25°) makes many non-hallmark FAs detected as hallmark. Both lead to pose estimation with low accuracy. We find that 15° and 20° result in small errors in all the four angles.

Fig. 10b shows the detailed results for TH_α= 20°: mean error (0°, 0.9°, 0.8°, 1.9°) and max error (0°, 2°, 2°, 4°). We notice that $\Delta\psi$ is more accurate than $\Delta\theta$, so is ψ_B than θ_B. This is because our method performs estimation based on FAs in cropped V_A and V_B, which span 360° horizontally but only 90° vertically. Thus estimating yaw related angles (ψ_A, ψ_B,

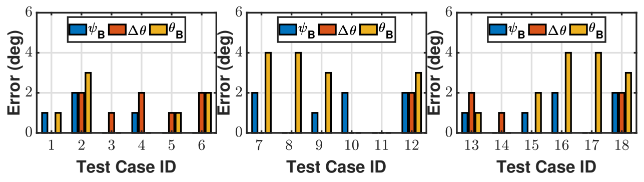


(a) Impact of TH_α. (b) Detailed error distribution.

Fig. 10: Estimation error of 360ViewPET.

$\Delta\psi$) has more FAs to use than estimating pitch related angles (θ_A, θ_B, $\Delta\theta$), and is more accurate.

Impact of Tour Route. Fig. 11 shows the errors for each of the 18 test cases (note that $\Delta\psi$ is always 0°, thus not shown). Cameras are deployed along a north-south straight line in Case 1–6 and east-west in Case 7–12 and on triangle vertices in Case 13–18. We do not find a remarkable impact of tour routes on pose estimation accuracy, which implies that 360ViewPET is fairly stable in different situations.



(a) Linear tour. (b) Linear tour. (c) Triangular tour.

Fig. 11: Different tours have similar estimation accuracy.

More and Harsher Test Cases. The 18 tour cases we have tested have three buildings of interest in the scenes and they provide FAs for pose estimation. A critical question is how well 360ViewPET can work in scenes with fewer FAs available (e.g., occlusion, fewer objects around, objects with poor features). To study this, we add different FA masks to our current V_A and V_B, removing some FAs from them and making pose estimation harder. In this way we generate more and harsher test cases than the 18 ones. Specifically, a mask applying to the forward, backward, left, right direction will remove those FAs with x-coordinates between 180°±45°, 360°±45°, 90°±45°, 270°±45°, respectively. We also test the cases where two directions have no FAs.

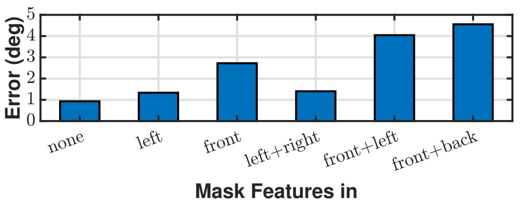
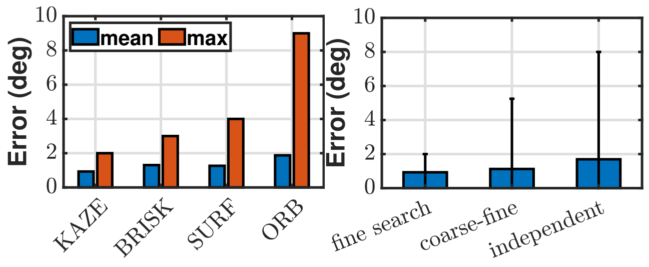


Fig. 12: More and harsher test cases.

Fig. 12 shows the overall error of cases where the FAs in zero, one or two directions are missing. As is seen, the error increases from 0.9° to 1.3° when the FAs in the left are removed, and to 2.7° when those in the forward direction are removed. The influence of no right FA is similar to that of no left FA, and no backward FA is similar to no forward FA, so the two results are omitted here. We can see that the absence of left/right FAs has less impact on the accuracy than that of forward/backward FAs. Since left/right FAs make less contrition than forward/backward ones, we expect that if masks are used on two directions, the error will be: masking left+right < masking front+left < masking front+back. This is congruent with Fig. 12: the three situations have an error of 1.4°, 4°and 4.5°, respectively. It is good to see that even in a case with no FA in two directions, the error is still small.

Impact of Feature Detection Algorithm. Fig. 13a shows the performances of four famous feature detectors: KAZE [8], BRISK [9], SURF [5], ORB [6]. They have similar overall mean errors: 0.9°, 1.3°, 1.2°, 1.8°. But the differences in max errors are remarkable: 2°, 3°, 4°, 9°. KAZE is recommended from the aspect of estimation accuracy.



(a) Feature detection algorithm. (b) Search strategy.

Fig. 13: Impact of other factors.

Impact of Search Strategy. Fig. 13b shows the performances of three search strategies. Fine search means exhausting every ($\Delta\psi$, ψ_B, $\Delta\theta$, θ_B) combination in [$\Delta\tilde{\psi}$-5°, $\Delta\tilde{\psi}$+5°]×[1°, 360°]×[-2°, 2°]×[-5°, 5°]; coarse-fine search performs fine search for $\Delta\psi$, $\Delta\theta$, θ_B, but searches ψ_B in [1°, 360°] with a granularity of 5° first to find its rough value, then fine searches it in ±4° around the rough value; independent search first fixes $\Delta\theta$, θ_B at 0° and searches $\Delta\psi$, ψ_B in [$\Delta\tilde{\psi}$-5°, $\Delta\tilde{\psi}$+5°]×[1°, 360°], then fixes $\Delta\psi$, ψ_B at the found values and searches $\Delta\theta$, θ_B in [-2°, 2°]×[-5°, 5°]. Though coarse-fine and independent searches are faster than fine search, they have much larger max errors (5.2° and 8°) than fine search's 2°. We use fine search.

Time Cost. Search space reduction shortens the pose estimation time for a camera pair from tens of hours to 5 sec in average (by ~20k times), when running with MATLAB on a laptop (2.7GHz CPU, 32GB RAM) using no GPU. If implemented using C/C++ or run on a server, it is reasonable to expect the time to be within 1 sec. Also, note that pose information is computed offline once (unless camera deployment changes), then stored in the cloud, and used repeatedly for user view generation. So taking seconds long is fine.

VII. Related Work

6DoF Video System. A single 360° camera system only has 3 rotational DoF; 6DoF (plus 3 translational DoF) requires multiple cameras. One type of 6DoF systems [10], [11], [12], [13] consists of sparsely deployed cameras which keep uploading live views, and a user can continuously change her view from one camera to another to virtually travel. 360ViewPET is related to this type, but studies camera pose estimation, a topic which has not been studied in these systems. Google Street View [1] uses prerecorded views, which is less relevant to immersive virtual tours and our work.

Another type of 6DoF systems [14], [15], [16] is based on depth image based rendering. They use a dense camera constellation (e.g., in [15] 16 cameras form a 1 m diameter rig) and target contexts where user motion is head-scale (e.g., in virtual conferences). They are inapplicable to our contexts: virtual outdoor tours across a wide area.

Image Registration. Image registration [7] discovers the correspondences of images capturing the same scene from different viewports or at different times and properly stitches them. Template matching is preferred for image registration when the images are rich of distinctive colors; feature detection (e.g., ORB [6]) is recommended when the images have distinctive shapes. Unlike 360ViewPET, image registration can estimate $\Delta\psi$ and $\Delta\theta$, but not ψ_B or θ_B.

Visual Based Localization (VBL). 360ViewPET belongs to VBL which recovers the poses of cameras based on the photos they took. Some VBL work [17], [18], [19] requires coarse camera intrinsics or extrinsics (e.g., from photo EXIF tags, GPS or magnetometer) as a prior, but 360ViewPET does not. New solutions [20] based on deep learning are proposed; unlike them, 360ViewPET needs no training.

Simultaneous Localization and Mapping (SLAM). SLAM also belongs to VBL, and has been significantly studied. Classical work (e.g., PTAM [2], ORB-SLAM [3]) applies to planar images, and new approaches [4], [21], [22] extend to spherical images, but they have the same drawback—require mobile cameras which record tens of frames per second while moving. However, we have ultra-sparse cameras and the frame rate is only 0.1 FPS in average, making SLAM totally fail. 360ViewPET works well despite such sparse cameras.

VIII. Conclusion

Camera pose estimation is a necessary step toward video-based immersive virtual tourism. In this paper, we present the design and evaluation of 360ViewPET, a strategy which automatically finds the relative poses of two 360° cameras (up to 15 m apart) using one equirectangular image taken by each camera, with a mean error as low as 0.9°. It is the only approach we know so far that works for such sparse cameras and applies to virtual outdoor tours across a wide area.

Acknowledgment

This research was funded by the National Science Foundation CNS 1900875, by the Postdoctoral Fellowship Program at CS UIUC and by the Grainger College of Engineering funding. The presented views in the article are of the authors and do not represent the views of the funding organizations.

References

[1] D. Anguelov, C. Dulong, D. Filip, C. Frueh, S. Lafon, R. Lyon, A. Ogale, L. Vincent, and J. Weaver, "Google street view: Capturing the world at street level," *Computer*, vol. 43, no. 6, pp. 32–38, 2010.

[2] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE, 2007, pp. 225–234.

[3] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "Orb-slam: a versatile and accurate monocular slam system," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[4] S. Sumikura, M. Shibuya, and K. Sakurada, "Openvslam: a versatile visual slam framework," in *Proceedings of the 27th ACM International Conference on Multimedia*, 2019, pp. 2292–2295.

[5] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[6] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[7] B. Zitova and J. Flusser, "Image registration methods: a survey," *Image and vision computing*, vol. 21, no. 11, pp. 977–1000, 2003.

[8] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, "Kaze features," in *European Conference on Computer Vision*. Springer, 2012, pp. 214–227.

[9] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2548–2555.

[10] X. Corbillon, F. De Simone, G. Simon, and P. Frossard, "Dynamic adaptive streaming for multi-viewpoint omnidirectional videos," in *Proceedings of the 9th ACM Multimedia Systems Conference*, 2018, pp. 237–249.

[11] H. Pang, C. Zhang, F. Wang, J. Liu, and L. Sun, "Towards low latency multi-viewpoint 360 interactive video: A multimodal deep reinforcement learning approach," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 991–999.

[12] T. Maugey, L. Guillo, and C. L. Cam, "Ftv360: a multiview 360° video dataset with calibration parameters," in *Proceedings of the 10th ACM Multimedia Systems Conference*, 2019, pp. 291–295.

[13] K. K. Sreedhar, I. D. Curcio, A. Hourunranta, and M. Lepistö, "Immersive media experience with mpeg omaf multi-viewpoints and overlays," in *Proceedings of the 11th ACM Multimedia Systems Conference*, 2020, pp. 333–336.

[14] J. Huang, Z. Chen, D. Ceylan, and H. Jin, "6-dof vr videos with a single 360-camera," in *2017 IEEE Virtual Reality (VR)*. IEEE, 2017, pp. 37–44.

[15] A. P. Pozo, M. Toksvig, T. F. Schrager, J. Hsu, U. Mathur, A. Sorkine-Hornung, R. Szeliski, and B. Cabral, "An integrated 6dof video camera and system design," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 6, pp. 1–16, 2019.

[16] M. Broxton, J. Flynn, R. Overbeck, D. Erickson, P. Hedman, M. Duvall, J. Dourgarian, J. Busch, M. Whalen, and P. Debevec, "Immersive light field video with a layered mesh representation," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, pp. 86–1, 2020.

[17] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: exploring photo collections in 3d," in *ACM siggraph 2006 papers*, 2006, pp. 835–846.

[18] C. Arth, C. Pirchheim, J. Ventura, D. Schmalstieg, and V. Lepetit, "Instant outdoor localization and slam initialization from 2.5 d maps," *IEEE Computer Architecture Letters*, vol. 21, no. 11, pp. 1309–1318, 2015.

[19] B. Zeisl, T. Sattler, and M. Pollefeys, "Camera pose voting for large-scale image-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2704–2712.

[20] Y. Shavit and R. Ferens, "Introduction to camera pose estimation with deep learning," *arXiv preprint arXiv:1907.05272*, 2019.

[21] X. X. Zhu, Y. Yu, P. F. Wang, M. J. Lin, H. R. Zhang, and Q. X. Cao, "A visual slam system based on the panoramic camera," in *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. IEEE, 2019, pp. 53–58.

[22] Y. Zhang and F. Huang, "Panoramic visual slam technology for spherical images," *Sensors*, vol. 21, no. 3, p. 705, 2021.