國立臺灣大學電機資訊學院電信工程學研究所

碩士論文

Graduate Institute of Communication Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master's Thesis

在雲端往返延遲與運算資源限制下之 AR 眼鏡移動人臉標記

Tagging Moving Faces for AR Glasses under Round-Trip Latency and Computing Resource Constraints

薛仁豪

Ren-Hau Shiue

指導教授：陳宏銘 博士

Advisor: Homer H. Chen, Ph.D.

中華民國 115 年 2 月

February, 2026

doi:10.6342/NTU202600774

誌謝

首先，我要感謝我的指導教授陳宏銘教授，教授不僅給予了密切而寶貴的指導，其嚴謹的研究態度也在各方面都啟發了我，幫助我在研究的道路上走得更扎實，也讓我在處理各項事務時能更加穩重。而教授對於表達與書寫能力的重視，以及隨之而來的訓練，更讓我能更加順暢地表達自己的意見，提升與他人合作的能力。

接著，我要感謝我的口試委員：陳宏銘教授、鍾國亮教授、林澤教授、李佩君教授、與施光祖博士，委員們在口試期間提供了許多珍貴的意見，從這些意見當中，我學習到了很多。此外，我也要感謝所有 MPAC 實驗室的同學們，特別是悉心傳授我經驗的陳泳源學長、幫助我進行實驗的沈驁毅同學、時常與我討論研究的彭瑎同學及呂則諺同學、給予我許多論文撰寫建議的李天敏同學。他們的協助讓我的研究能順利進行，與他們的討論過程更讓我學到許多寶貴的知識。

我也要謝謝常常幫助我的朋友們。感謝黃品皓與姚梃睿同學時常與我討論研究時遇到的問題，給出他們獨特的見解，也謝謝張嘉恩同學給了我許多報告上的建議。最後，我也要感謝我的家人，感謝他們不遺餘力的支持，讓我能毫無後顧之憂地進行研究。

doi:10.6342/NTU202600774

中文摘要

具備人臉辨識能力的擴增實境眼鏡，能在視野中的人物上疊加虛擬姓名標籤，使使用者在不打斷眼神接觸的情況下辨識他人。然而，在輕量化擴增實境眼鏡上實現可靠的標籤擺放仍具挑戰，主因包括雲端往返延遲、眼鏡端運算資源受限，以及缺乏深度感測器。延遲會使得標籤落後於移動的目標，而當標籤擺放的深度與真實目標的深度不一致時，則在使用者注視目標時可能產生複視的現象。本論文提出一套眼鏡端與伺服器端協作的人臉標記系統，能在眼鏡端提供即時的人臉定位，以及在不依賴深度感測器的前提下，估測移動目標的絕對尺度深度。在人臉定位方面，我們設計了一個基於孿生網路的拆分式定位流程，將計算分流至眼鏡端與伺服器端。伺服器端週期性更新目標模板，而眼鏡端僅執行輕量的搜尋分支，以達成即時定位。在深度估計方面，我們結合視覺慣性同步定位與建圖系統 SLAM 與單目深度估計網路，利用 SLAM 提供的稀疏絕對深度樣本，將網路輸出的相對深度圖進行尺度對齊，得到具有絕對尺度的深度圖，並由此估計目標深度。在三個公開資料集、延遲感知的評估協定下，本系統的人臉定位在 93.11% 的影格中可達到 IoU 大於 0.5，優於眼鏡端人臉偵測的 78.08% 與伺服器端人臉偵測的 53.90%。此外，在四個具代表性的社交情境中，我們的深度估計在 96.15%的影格中，能將深度誤差維持在複視閾值以下，優於人臉寬度先驗基線的 85.40% 與 Depth Anything V2 的 78.11%。

關鍵字：擴增實境、人臉標記、雲端與邊緣運算、孿生網路、深度估計。

doi:10.6342/NTU202600774

ABSTRACT

Augmented reality (AR) glasses with face recognition can overlay virtual name labels on people in view, allowing users to identify others without breaking eye contact. However, reliable label placement on lightweight AR glasses remains challenging due to round-trip latency, limited on-glasses computing resources, and the lack of a depth sensor. Latency causes the rendered label to lag behind the moving target, while depth mismatch can induce double vision when the user fixates on the target. We present a glasses–server collaborative face tagging system that provides real-time on-glasses face localization and metric depth estimation for moving targets without requiring a depth sensor. For face localization, we design a split Siamese pipeline that divides computation between the glasses and the server. The server periodically updates the target face template, while the glasses execute only the lightweight search branch. For depth estimation, we combine visual–inertial SLAM with a monocular depth estimation network. We use sparse metric depth samples from SLAM to affinely align the network's relative depth map to metric scale, producing a metric depth map from which we estimate the target face depth. Evaluated under a latency-aware protocol on three public datasets, our face localization achieves an IoU above 0.5 in 93.25% of frames on average, outperforming on-glasses face detection at 77.38% and server-side face detection at 53.99%. In addition, averaged across four representative social scenarios, our depth estimation keeps the depth error below the diplopia threshold in 96.15% of frames on average, surpassing the Face-Width Prior baseline at 85.40% and Depth Anything V2 at 78.11%.

Keywords: Augmented reality, face tagging, cloud and edge computing, Siamese network, depth estimation.

doi:10.6342/NTU202600774

CONTENTS

誌謝 i

中文摘要 ii

ABSTRACT iii

CONTENTS iv

LIST OF FIGURES vi

LIST OF TABLES viii

Chapter 1 Introduction 1

Chapter 2 Related Work 7

2.1 Face Recognition on Smart Glasses and AR Glasses 7

2.2 Monocular Depth Estimation 8

2.3 Siamese Networks for Similarity Learning 10

Chapter 3 Proposed System 12

3.1 Problem Formulation 12

3.2 System Architecture 13

3.3 Face Localization Pipeline 15

3.3.1 Server-Assisted Template Initialization and Update 15

3.3.2 Search Region Selection 17

3.3.3 Similarity Map from Split Siamese Matching 17

3.3.4 Deriving the Label Anchor from Facial Landmarks 18

3.4 Depth Estimation Pipeline 20

3.4.1 Sparse Metric Depth Samples from Visual-Inertial SLAM 20

3.4.2 Relative Depth Map from Monocular Depth Estimation 22

3.4.3 Affine Alignment to Metric Depth Samples 22

doi:10.6342/NTU202600774

3.5 Delay Compensation....23
Chapter 4 Experiments....25
4.1 Evaluation of Face Localization....25
4.1.1 Datasets....25
4.1.2 Metrics and Latency-Aware Evaluation Protocol....26
4.1.3 Experimental Setup and Hardware Platforms....27
4.1.4 Localization Accuracy across Various Compute Power Levels....28
4.1.5 Localization Accuracy under Various Network Latency Settings....31
4.2 Evaluation of Depth Estimation....34
4.2.1 Data Collection....34
4.2.2 Perceptually Motivated Metrics and Evaluation Protocol....34
4.2.3 Compared Methods and Implementation Details....36
4.2.4 Depth Estimation Results....38
4.3 System Implementation and Runtime Evaluation....40
4.3.1 Implementation Details....40
4.3.2 Power, Battery, and Thermal Characteristics....42
Chapter 5 Discussion and Future Work....45
Chapter 6 Conclusion....46
REFERENCES....47

doi:10.6342/NTU202600774

LIST OF FIGURES

Fig. 1.1 Examples of misalignment between augmented labels and the target faces. (a) Image-plane misalignment results in incorrect label association. (b) Depth misalignment causes double vision of the augmented label. 2

Fig. 3.1 System architecture. 14

Fig. 3.2 Face localization pipeline. 16

Fig. 3.3 Depth Estimation pipeline. 21

Fig. 4.1 Latency-aware evaluation protocol for localization. We align the prediction with the ground truth by linearly interpolating the ground-truth boxes between frames ℓ and $\ell + 1$. 26

Fig. 4.2 Localization performance under various compute power levels. 30

Fig. 4.3 Localization performance versus network round-trip latency. 33

Fig. 4.4 Example first-person views from the four scenes in our evaluation dataset, captured by the camera on the RayNeo X2 AR glasses. (a) Lobby. (b) Classroom. (c) Laboratory. (d) Corridor. 35

Fig. 4.5 Double vision condition. The real face is at the ground-truth depth z_{gt}, while the label is rendered at the estimated depth z_{est}. This depth mismatch produces an viewing-angle difference θ. Double vision occurs when θ exceeds the diplopia threshold. 36

Fig. 4.6 Distribution of absolute depth estimation error across scenes. 39

Fig. 4.7 Distribution of relative depth estimation error across scenes. 39

Fig. 4.8 Hardware setup of our implementation. (a) RayNeo X2 AR glasses. (b) A server running Linux equipped with an NVIDIA RTX 4090 GPU. 41

doi:10.6342/NTU202600774

Fig. 4.9 Measured average power consumption under four incremental system configurations. 43

Fig. 4.10 Measured battery life under continuous operation for four incremental system configurations. 43

Fig. 4.11 CPU and GPU temperatures of the RayNeo X2 during continuous operation. The curves show the measured temperature over time under the full pipeline configuration at an ambient temperature of 24 °C. 44

doi:10.6342/NTU202600774

LIST OF TABLES

Table 4.1 Functional partitioning of the evaluated systems. 27

Table 4.2 Evaluation platforms spanning various compute power levels. 28

Table 4.3 Edge processing time of evaluated systems under various computing power levels. 29

Table 4.4 Localization performance of the evaluated systems on the three datasets under 100ms server round-trip latency 32

Table 4.5 Double-vision-free ratio across scenes. 38

doi:10.6342/NTU202600774

Chapter 1 Introduction

Augmented Reality (AR) glasses enhance user perception by overlaying digital information onto real-world scenes, enabling applications such as navigation, context-aware assistance, and real-time information retrieval. Among these, face recognition is a compelling use case, as it can assist in social settings such as classrooms and conferences, and help professionals in workplaces such as hospitals and factories, where rapid identification is essential. With AR glasses, the name label can be placed directly on the target person, allowing the user to identify the person without breaking eye contact. This placement not only accelerates recognition but also allows the recognition behavior to integrate more naturally into social interactions.

Unlike conventional face recognition, which mainly focuses on recognition accuracy, face tagging on AR glasses introduces an additional requirement that the name label remain reliably aligned with the target in the user's view. Misalignment can be visually uncomfortable and can undermine the reliability of label placement, thereby reducing users' willingness to use the system. To better understand and address this requirement, we analyze it along two dimensions, image-plane alignment and depth alignment, with different tolerance requirements. Image-plane alignment concerns whether the label is placed on the intended person. In multi-person scenes, misalignment can cause the label to drift onto another person, as shown in Fig. 1.1(a). Depth alignment concerns whether the label is placed at a depth consistent with the target. A small depth mismatch may be tolerable, but a large mismatch can make the label appear in front of or behind the target and lead to double vision when the user fixates on the target, as shown in Fig. 1.1(b). To make these requirements concrete, we define two acceptance thresholds. For image-plane

doi:10.6342/NTU202600774

(a) (b)

Fig. 1.1 Examples of misalignment between augmented labels and the target faces. (a) Image-plane misalignment results in incorrect label association. (b) Depth misalignment causes double vision of the augmented label.

alignment, a result is considered acceptable if the intersection-over-union (IoU) between the localized face bounding box and the ground-truth bounding box exceeds 0.5. For depth alignment, a result is considered acceptable if the angular disparity induced by depth mismatch remains below 2.92mrad [1] to avoid double vision. Therefore, reliable face tagging requires satisfying both criteria simultaneously over time. In this thesis, we focus on common indoor and semi-open social settings (e.g., corridors, classrooms, and halls), where face tagging on AR glasses is particularly beneficial. However, meeting these requirements for socially wearable, lightweight AR glasses intended for such settings is challenging for two reasons. First, limited on-glasses computing resources increase end-to-end latency, causing the name label to lag behind the target on the image plane. Second, these glasses often lack a depth sensor, making accurate depth estimation difficult. Accordingly, this thesis presents a face tagging system that satisfies these criteria under such constraints to enable reliable label placement. In this thesis, we focus on

doi:10.6342/NTU202600774

indoor and semi-open environments, such as corridors, classrooms, and halls, which are common social settings where face tagging on AR glasses is particularly beneficial.

We first focus on face localization, which estimates the target face location on each frame. In this context, image-plane alignment depends not only on localization accuracy but also on localization speed. When localization is slow, the face may have already moved by the time the label is rendered, resulting in a larger misalignment between the label and the target. Therefore, reliable face tagging requires face localization to be both accurate and fast, yet prior systems cannot achieve both simultaneously. For example, Rahardjo and Chen stream images captured by the camera on Jorjin's J7EF Plus AR glasses over MQTT to a server and return the identity and face location to the glasses [2]. Similarly, Liao et al. send an image captured by the Microsoft HoloLens 2 headset to Microsoft Azure Cognitive Services for face recognition and display the returned identity on the headset [3]. In this server-based design, face localization and recognition are computed remotely, so the results arrive at the glasses only after the network round-trip latency. By then, the target may have already moved, causing the name label to lag behind. In contrast to server-based design, Łysakowski et al. run a YOLOv8 detector directly on the Microsoft HoloLens 2 headset and report an end-to-end latency of approximately 100ms from image capture to rendering the label [4]. This result suggests that, even on such a bulky headset, running face detection can still be too slow for real-time localization. To reduce the computational burden, MediaPipe performs face detection periodically and uses optical-flow-based tracking to propagate the bounding boxes in the intervening frames, thereby updating the face location at a higher rate [5]. Farasin et al. adopt a similar strategy in which face detection is performed remotely, while the glasses run a Mean Shift tracker to maintain temporal continuity across successive bounding boxes returned by the server [6]. Despite their low computational cost, such classical tracking methods can

doi:10.6342/NTU202600774

become unreliable under rapid head motion or partial occlusions, causing drift and unstable face location estimates. Taken together, prior systems fail to fully utilize the combined computational resources of the glasses and the server for face localization. We argue that a more effective face localization framework should treat the glasses and the server as a single distributed system and explicitly design the localization model and inference pipeline so that both sides actively contribute. This perspective makes face localization fast enough for real-time label placement and eliminates the need to rely on fragile tracking to maintain continuity. Building on this principle, we design a split Siamese pipeline for face localization that assigns complementary roles to the glasses and the server. Because the target's facial appearance and scale do not change rapidly, the server performs detection and updates the template features at a low rate, while the glasses execute only the lightweight per-frame search branch to localize the face and render the label in real time. The server computes template features asynchronously, and the glasses apply the most recent available template without blocking the rendering loop. This design not only leverages the server's computational resources for heavy processing, but also minimizes the impact of network latency.

We next focus on depth estimation, which estimates the target's metric depth to render the label at the correct depth for comfortable label placement. In our setting, because the glasses lack a depth sensor, depth information must be inferred from RGB images together with inertial measurement unit (IMU) measurements. Prior work has explored depth estimation under similar sensing constraints, yet each family of methods has limitations, making it difficult to meet the requirement in our setting. Classical structure-from-motion and simultaneous localization and mapping (SLAM) methods rely on triangulation under a static-scene assumption and therefore cannot reliably estimate the depth of moving targets. To relax this assumption, prior work extends triangulation

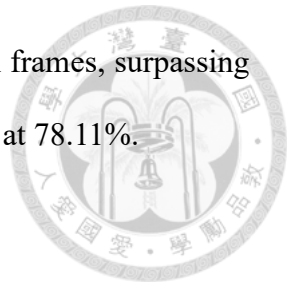
doi:10.6342/NTU202600774

with explicit motion priors for the target, such as linear-trajectory models [7] or B-spline formulations [8]. These methods can accommodate target motion, but they require sustained camera translation to accumulate sufficient parallax, and their performance degrades when the actual motion deviates from the assumed model. To avoid this reliance on parallax, recent learning-based approaches estimate depth directly from a single image. Monocular depth estimation networks provide per-pixel depth estimates from a single image, but recovering metric depth from a single view is intrinsically ill posed. As a result, most networks predict relative depth, leaving an unknown global scale and shift with respect to metric depth. To mitigate these limitations, we combine visual–inertial SLAM with a monocular depth estimation network to exploit their complementary strengths. Specifically, SLAM provides sparse metric depth samples from reconstructed map points, which we use to estimate a global scale and shift that convert the network's relative depth map into a per-pixel metric depth map. As a result, we can estimate the target face depth in metric scale from this depth map, even when the target is moving.

Building on the integration of the above two core components, we present a face tagging system for lightweight AR glasses that enables reliable label placement under round-trip latency, limited on-glasses computing resources, and the absence of a depth sensor. Through glasses–server collaboration via a split Siamese pipeline, the proposed system performs real-time face localization on the glasses. It further integrates visual–inertial SLAM with a monocular depth estimation network to estimate the moving target's metric depth, thereby reducing double vision caused by depth mismatch between the label and the target face. Evaluated under a latency-aware protocol on three public datasets, our face localization achieves an IoU above 0.5 in 93.25% of frames, outperforming on-glasses face detection [4] at 77.38% and server-side face detection [2] at 53.99%. In addition, averaged across four representative social scenarios, our depth estimation keeps

doi:10.6342/NTU202600774

the depth error below the diplopia threshold in 96.15% of evaluation frames, surpassing the Face-Width Prior baseline at 85.40% and Depth Anything V2 [9] at 78.11%.



doi:10.6342/NTU202600774

Chapter 2 Related Work

This chapter reviews prior work that informs the design of face tagging on AR glasses. We organize the discussion around three threads. We first review how face recognition has been deployed on smart glasses and AR glasses, and how system designs are shaped by the constraints of lightweight AR glasses. We then survey monocular depth estimation, highlighting the transition from geometric reconstruction to learning-based depth estimation, and emphasizing the gap between relative depth and metric depth. Finally, we introduce the Siamese network architecture and how it enables similarity learning. We also review its development for localization tasks and discuss how this architecture inspires our design.

2.1 Face Recognition on Smart Glasses and AR Glasses

Early smart glasses extended conventional glasses by integrating additional components for assistive use. Google Glass, for example, integrates a front-facing camera, a microphone and a speaker, wireless connectivity, and a small display that can present information in the user's view. These capabilities enabled functions such as capturing photos, playing audio, and translation. Face recognition was also explored as one such application. Mandal et al. presented a wearable face recognition system on Google Glass that performs recognition using conventional eigenfeature regularization and extraction. In their system, the glasses capture photos, transmit them to a paired smartphone for recognition, and then return the recognition results to the glasses for display [10]. Daescu et al. later developed a face recognition system based on convolutional neural networks. Because neural networks require greater computational resources, their system transmits

doi:10.6342/NTU202600774

captured photos to a remote server for inference [11].

As AR glasses matured, face recognition was increasingly integrated into AR applications, where name labels were overlaid on the target rather than shown separately. For example, Liao et al. present an augmented classroom teaching system built on HoloLens 2. Their system uses face detection to localize students, invokes a cloud-based face recognition service to identify them, and uses the built-in time-of-flight sensor to measure depth for AR display [3]. However, HoloLens is too bulky for everyday social use, which motivates efforts to bring face recognition to lightweight AR glasses. McKelvey et al. explored face recognition on Vuzix Blade, a pair of lightweight AR glasses featuring a see-through, right-eye monocular display. Due to limited on-glasses compute, they used lightweight components such as a Haar cascade classifier for face detection and Eigenfaces for recognition, and only when paired with a tablet did they run a more accurate detector such as MTCNN on the external device [12]. In contrast, Rahardjo and Chen propose a cloud-based face recognition system for AR glasses, where frames are streamed to the cloud for recognition and the results are returned for display [2]. While a purely cloud-based design alleviates the compute constraints on the glasses, it introduces network latency, which can leave the end-to-end delay too high for reliable tagging. Overall, existing systems still fall short of enabling reliable face tagging on lightweight AR glasses, which motivates our work.

2.2 Monocular Depth Estimation

Monocular depth estimation estimates depth from images captured by a single camera and has been studied for decades. Early geometric approaches recover depth by tracking 2D feature points across frames, estimating camera motion from those tracks,

doi:10.6342/NTU202600774

and triangulating the corresponding rays. Tomasi and Kanade proposed a factorization approach that stacks tracked image coordinates over time into a measurement matrix and applies a rank three decomposition to recover camera motion and 3D shape under an orthographic projection model [13]. In parallel, incremental SLAM methods framed mapping and localization as recursive state estimation, updating camera pose and landmark positions whenever new measurements arrive, often using an extended Kalman filter [14]. These methods can estimate depth reliably when the target being triangulated remains static in the world, but they can fail when the target moves, because the same 2D track no longer corresponds to a single fixed 3D point, breaking the triangulation assumption.

To relax the static target assumption, later work incorporated parametric motion models so depth can still be inferred when the target moves. Avidan and Shashua proposed trajectory triangulation and assumed the target follows either linear motion or conic motion, which makes the 3D trajectory identifiable from a monocular image sequence [7]. Park et al. extended this idea by representing the target trajectory with a B spline and reconstructing it from a series of 2D projections [8]. These methods show that moving target depth is still recoverable when the motion can be approximated well by the chosen model. Nevertheless, their performance depends on whether the model matches real motion, and reliable depth estimation still requires sufficient parallax over time, which typically requires continuous camera motion.

In recent years, learning-based methods have shifted the focus to per-pixel depth estimation from a single image. Monodepth2 is a representative example that shows how a neural network can be trained without per-pixel ground-truth depth by enforcing self-supervised photometric consistency across adjacent views [15]. Such training objectives can learn depth ordering and overall scene structure, but the global metric scale remains

doi:10.6342/NTU202600774

ambiguous. As a result, the output is defined only up to an unknown global scale and shift, commonly referred to as relative depth. Recent models such as Depth Anything V2 further improve accuracy and robustness, and they perform reliably across diverse scenes as relative depth estimators [9]. However, estimating depth in physical units remains challenging because the metric scale must remain consistent across cameras with distinct intrinsic parameters and as the camera pose changes over time. Recent work therefore calibrates a relative depth map at inference time by fitting scale and shift parameters using a small set of samples with known metric depth, for example a few LiDAR points [16]. While lightweight AR glasses typically do not include LiDAR, the underlying idea is directly relevant. Inspired by this, our approach uses SLAM to obtain sparse metric depth samples from the static background, and then fits alignment parameters that convert the network's relative depth map into a metric depth map.

2.3 Siamese Networks for Similarity Learning

Siamese networks learn a similarity function by processing two inputs with twin branches that share the same weights, then comparing the resulting feature embeddings. This idea was introduced by Bromley et al. for signature verification, where the network is trained on pairs of samples and learns to output nearby embeddings for matching pairs and separated embeddings for nonmatching pairs [17]. A typical Siamese design therefore consists of a backbone that maps each input patch into an embedding, followed by a distance or correlation operation that produces a similarity score, often trained with a pairwise objective such as contrastive loss.

This similarity learning view later became a natural fit for tracking and localization, where the task can be formulated as matching a target template against a larger search

doi:10.6342/NTU202600774

region in the next frame. Bertinetto et al. proposed a fully convolutional Siamese tracker that computes feature for the template and the search region, then performs cross correlation to produce a similarity map whose peak indicates the target location. This design enables real time tracking with a compact model and a lightweight matching operation [18]. Leveraging the twin-branch structure of Siamese networks and the fact that a face template typically changes slowly, we split the localization pipeline between the server and the glasses. The server performs template extraction and occasional updates, while the glasses execute only the lightweight search branch with correlation-based localization. This split design substantially reduces the computational load on the glasses.

doi:10.6342/NTU202600774

Chapter 3 Proposed System

In this chapter, we present the proposed system in detail. Section 3.1 describes the overall objective of the system, its inputs and outputs, and introduces the notation used in the remainder of the chapter. Section 3.2 then follows the data flow to explain the system architecture and the roles of the individual modules. Sections 3.3 and 3.4 delve into the two main processing pipelines, namely face localization and depth estimation. Finally, Section 3.5 presents the delay compensation performed before rendering.

3.1 Problem Formulation

We consider a pair of AR glasses equipped with a monocular RGB camera and an IMU, connected via a wireless network to a remote server with higher computational resources. Let I_k denote the RGB image captured by the camera at discrete time index k, and let m_k denote the IMU measurements associated with the same index, obtained by aggregating the higher-rate accelerometer and gyroscope readings between the capture instants of I_{k-1} and I_k. When $k = 0$, there is no preceding interval to aggregate, so we define m_0 to be empty. Let $p_k \in \mathbb{R}^3$ denote the 3D position of a target face at time index k. We seek a real-time and causal method that outputs an estimate $\hat{p}_k$ as soon as I_k is captured, using only measurements up to time index k:

$$\hat{p}_k = f(I_{0:k}, m_{0:k}), \quad k = 0,1,2,\ldots \tag{3.1}$$

The estimated 3D positions $\{\hat{p}_k\}$ serve as anchors for rendering labels on the AR display.

doi:10.6342/NTU202600774

3.2 System Architecture

To realize real-time, causal estimation on compute-constrained AR glasses, we adopt a cloud–edge collaborative architecture as shown in Fig. 3.1. The core idea is to offload compute-intensive modules to the server while keeping latency-critical modules on the glasses. Accordingly, the glasses handle sensor acquisition, the search branch of face localization, delay compensation, and label rendering. A separate MQTT client thread handles communication with the server, and the server runs the template branch of face localization, face recognition, visual–inertial SLAM, and depth estimation. The colors in Fig. 3.1 indicate typical update rates, with the IMU running at 200 Hz, the camera and search branch operating at 30 fps, and the server-side pipelines and MQTT message exchanges updating at 5 fps.

On the glasses, the rendering thread acquires images from a monocular RGB camera and inertial measurements from the IMU. For each new frame, this thread invokes the search branch of the face localization pipeline, taking the current image and the latest template received from the server as input. The template includes the target face bounding box and its template features. Conditioned on this template, the search branch localizes the target face on the image plane, thereby defining a 2D anchor for label placement in the current frame. Next, the delay compensation module estimates the camera motion over the processing delay using IMU measurements together with the SLAM state returned by the server (including the camera pose, velocity and IMU biases). It then uses the estimated motion and the target face depth to back-project the delayed 2D anchor to 3D and propagate it to the current time, yielding an estimate of the current 3D anchor. Finally, the display module renders the corresponding name label at this anchor.

Also on the glasses, the MQTT client thread handles all communication with the

doi:10.6342/NTU202600774

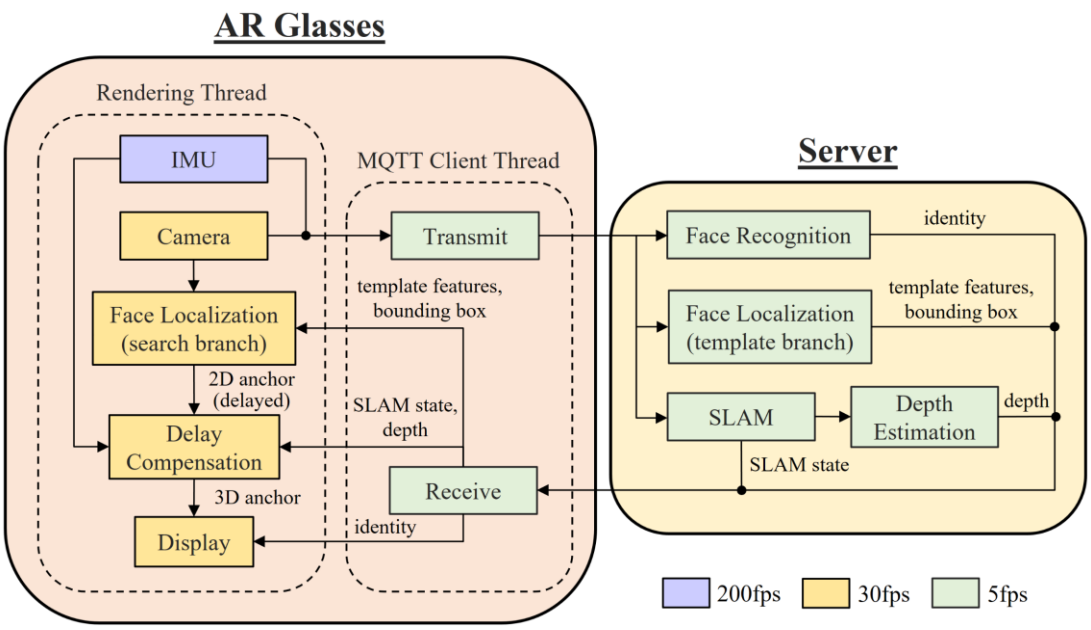


Fig. 3.1 System architecture.

server. It periodically transmits captured images and the corresponding IMU measurements to the server. On the downlink, it receives the latest SLAM state, the identity, the target face bounding box and template features, and the depth estimate. These results are cached in a buffer accessible to the rendering thread, so that template updates do not block rendering.

On the server, three pipelines operate in parallel on the received images and IMU stream. The first pipeline is the template branch of face localization, which detects the target face in each received frame and extracts the template features and face bounding box used by the search branch on the glasses. The second pipeline performs face recognition on the detected face and assigns an identity label to the target. The third pipeline estimates the target face depth by combining visual–inertial SLAM with a monocular depth estimation model. Together, these pipelines output the target identity, the updated template features and bounding box, the depth estimate, and the SLAM state estimates that are sent back to the glasses. Although these outputs arrive at the glasses

doi:10.6342/NTU202600774

with substantial round-trip latency, they typically evolve slowly and therefore remain useful for on-glasses processing. In particular, the template features, face scale, depth estimate, and IMU bias estimates vary gradually over time. When the bias estimates are sufficiently accurate, the camera velocity can be reliably propagated from the last server estimate using IMU measurements. Moreover, the identity label changes only when the system switches to an another target.

we next describe the key pipelines in detail, including face localization, depth estimation, and delay compensation. We begin with face localization.

3.3 Face Localization Pipeline

This section describes the face localization pipeline that outputs a 2D anchor for label placement. To minimize on-glasses computation, we split the pipeline into a server branch and an edge branch using a Siamese-network-based design, as shown in Fig. 3.2. We begin by describing how the server branch initializes and periodically updates the target template, which is sent back to the glasses as the latest available template. We then describe the edge branch, which localizes the target face using this template. It starts with search region selection, followed by split Siamese matching to produce a similarity map, and finally localizes the target face from the similarity map and derives the 2D anchor using facial landmarks.

3.3.1 Server-Assisted Template Initialization and Update

At the beginning of a localization task, the template buffer on the glasses is empty. The glasses therefore transmit the first frame I_0 to the server, where the server computes the template and sends it back to the glasses for buffering. For notational simplicity, we

doi:10.6342/NTU202600774

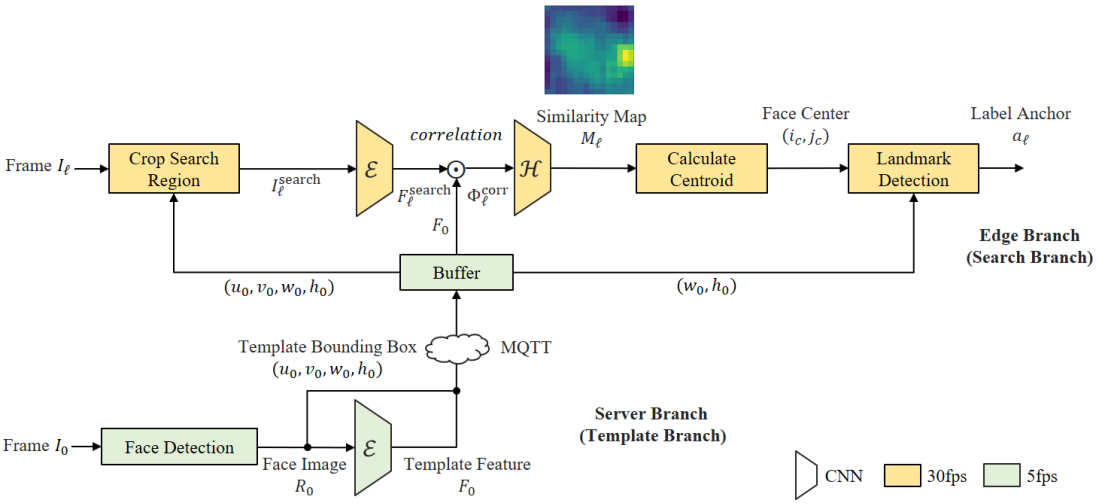


Fig. 3.2 Face localization pipeline.

use the subscript 0 to denote template-related symbols. This notation does not imply that only the first frame is used to compute the template; the same operations are applied whenever a new frame is transmitted to the server to update the template.

Upon receiving I_0, the server applies a face detector to localize the target face and obtain its bounding box. We crop the face image using this bounding box and pass it through a lightweight CNN to extract the template feature.

Formally, let $\mathcal{D}$ denote the face detector and $\mathcal{E}$ the CNN-based template feature extractor. Given the first image I_0 transmitted to the server, the server branch computes

$$b_0 = \mathcal{D}(I_0), \qquad R_0 = \text{Crop}(I_0, b_0), \tag{3.2}$$

$$F_0 = \mathcal{E}(R_0) \in \mathbb{R}^{C \times h_T \times w_T}, \tag{3.3}$$

where b_0 is the bounding box of the target face, R_0 is the corresponding cropped face image, and F_0 is the template feature. We refer to the pair (b_0, F_0) as the template of the target face. Once computed on the server, the template is transmitted to the glasses and cached in the template buffer for subsequent use by the edge branch.

doi:10.6342/NTU202600774

3.3.2 Search Region Selection

Due to server processing time and network latency, the template (b_0, F_0) computed from I_0 becomes available on the glasses after an ℓ-frame delay, where ℓ denotes the network round-trip latency measured in frames. By the time the template arrives, the camera has captured frame I_ℓ, and the search branch starts operating. Let $b_0 = (u_0, v_0, w_0, h_0)^\top$ denote the template bounding box. From frame I_ℓ, we crop a search patch I_ℓ^{search} using a rectangular window centered at (u_0, v_0), whose width W_ℓ and height H_ℓ are defined as

$$W_\ell = (1 + \alpha\ell)w_0, \qquad H_\ell = (1 + \alpha\ell)h_0. \tag{3.4}$$

Here, α is a positive parameter. In this way, the search region scales with both the face size and the latency, so that it remains large enough to cover the target face despite its motion during the latency.

3.3.3 Similarity Map from Split Siamese Matching

Given the search patch I_ℓ^{search}, the next step is to localize the target face within this patch. We reuse the same feature extractor $\mathcal{E}$ as in the template branch, that is, the weights of $\mathcal{E}$ are shared between the template and search branches. Applying $\mathcal{E}$ to the search patch yields the search feature

$$F_\ell^{\text{search}} = \mathcal{E}\left(I_\ell^{\text{search}}\right) \in \mathbb{R}^{C \times h_S \times w_S}. \tag{3.5}$$

To compute similarities between the template and each location in the search region, we adopt a point-wise correlation formulation [19]. We flatten the spatial dimensions of the template and search feature maps using a vectorization operator $\text{vec}(\cdot)$:

doi:10.6342/NTU202600774

$$\Phi_0 = \mathrm{vec}(F_0) \in \mathbb{R}^{C\times(h_\mathrm{T}w_\mathrm{T})}, \tag{3.6}$$

$$\Phi_\ell^{\mathrm{search}} = \mathrm{vec}\left(F_\ell^{\mathrm{search}}\right) \in \mathbb{R}^{C\times(h_\mathrm{S}w_\mathrm{S})}, \tag{3.7}$$

where each column of Φ_0 or $\Phi_\ell^{\mathrm{search}}$ corresponds to the C-dimensional feature vector at a single spatial location of the template or search feature map, respectively. The point-wise correlation matrix between the template and search features is then given by

$$\Phi_\ell^{\mathrm{corr}} = \Phi_0^\top \Phi_\ell^{\mathrm{search}} \in \mathbb{R}^{(h_\mathrm{T}w_\mathrm{T})\times(h_\mathrm{S}w_\mathrm{S})}. \tag{3.8}$$

Each row of $\Phi_\ell^{\mathrm{corr}}$ corresponds to a spatial location in the template feature map, and contains its correlation scores with every spatial location in the search feature map. We then reshape $\Phi_\ell^{\mathrm{corr}}$ into an $h_\mathrm{S} \times w_\mathrm{S}$ grid with $h_\mathrm{T}w_\mathrm{T}$ channels, just like a multi-channel image. It is then fed into a lightweight convolutional head $\mathcal{H}$, which aggregates the channels and produces a single-channel similarity map $M_\ell \in \mathbb{R}^{h_\mathrm{S}\times w_\mathrm{S}}$. High responses in M_ℓ indicate likely locations of the target face. In the next step, we use M_ℓ to estimate the face center and derive the 2D anchor for label placement on frame I_ℓ.

3.3.4 Deriving the Label Anchor from Facial Landmarks

Given the similarity map $M_\ell \in \mathbb{R}^{h_S\times w_S}$ over the search patch of frame I_ℓ, we derive a 2D anchor for label placement in two steps. We first estimate the face center from M_ℓ, and then refine it using a lightweight facial landmark detector. Facial landmarks provide semantically meaningful reference points, making the anchor less sensitive to noise in the similarity map.

To estimate the face center, we normalize M_ℓ into a discrete probability distribution

doi:10.6342/NTU202600774

$$\widetilde{M}_\ell(i,j) = \frac{M_\ell(i,j)}{\sum_{i'=1}^{w_S}\sum_{j'=1}^{h_S} M_\ell(i',j')} \tag{3.9}$$

and compute the centroid on the similarity map as

$$i_c = \sum_{i,j} i\,\widetilde{M}_\ell(i,j), \qquad j_c = \sum_{i,j} j\,\widetilde{M}_\ell(i,j). \tag{3.10}$$

The resulting (i_c, j_c) is the estimated face center on the similarity map, and we map it to the image coordinate system of frame I_ℓ using the known origin and scale of the search patch.

After obtaining this center, we estimate a face bounding box on frame I_ℓ and then detect facial landmarks to derive a more stable anchor. Using this center as the box center and the width and height of the template bounding box b_0 as its size, we define a face bounding box $\hat{b}_\ell$ on frame I_ℓ and crop the face image as,

$$I_\ell^{\text{face}} = \text{Crop}\big(I_\ell, \hat{b}_\ell\big). \tag{3.11}$$

We then determine the final anchor using facial landmarks. Let f_{lm} denote a lightweight five-point facial landmark detector applied to I_ℓ^{face},

$$L_\ell = f_{\text{lm}}\big(I_\ell^{\text{face}}\big) \in \mathbb{R}^{5\times 2}. \tag{3.12}$$

Each row of L_ℓ stores the 2D image coordinates of one facial landmark. Among these five landmarks, we select the two eye landmarks, compute their midpoint, and place the label anchor slightly above this midpoint. The resulting 2D anchor on frame I_ℓ is denoted by $(u_\ell^{\text{face}}, v_\ell^{\text{face}})$. At this point, one run of the face localization pipeline is complete. The same pipeline is then repeated for every captured frame to produce a per-frame 2D anchor for label placement. In the next section, we estimate the target face depth. This depth is later used in Section 3.5 to back-project the 2D anchor to 3D.

doi:10.6342/NTU202600774

3.4 Depth Estimation Pipeline

This section describes our depth estimation pipeline for obtaining a metric depth estimate of the target face, as shown in Fig. 3.3. Our approach combines two complementary sources of depth information. First, a visual–inertial SLAM module reconstructs a sparse 3D map and provides metric depths for map points visible in the current frame. Second, a monocular depth estimation model predicts a per-pixel relative depth map over the image, which covers both static background and moving objects but is defined only up to a global scale and shift. We use the sparse metric depth samples from SLAM as reference to fit the global scale and shift parameters, thereby converting the relative depth map into a per-pixel metric depth map.

3.4.1 Sparse Metric Depth Samples from Visual-Inertial SLAM

In this section, we describe how the visual–inertial SLAM module provides sparse metric depth samples in each frame. The procedure consists of two steps. First, we run the SLAM module to update the current set of 3D map points and the SLAM state, including the camera pose, velocity, and IMU bias estimates. Second, we project the map points into the image plane of the current frame to obtain a sparse set of 2D image locations, each associated with a metric depth value. We refer to these as metric depth samples. These samples serve as references for calibrating the relative depth map in Section 3.4.3.

We next formalize this procedure. Let I_0 denote the first frame transmitted to the server, and let m_0 denote its associated IMU measurements. the SLAM module estimates the current SLAM state s_0, and the current set of 3D map points $\mathcal{X}_0^{\mathrm{W}}$ as

doi:10.6342/NTU202600774

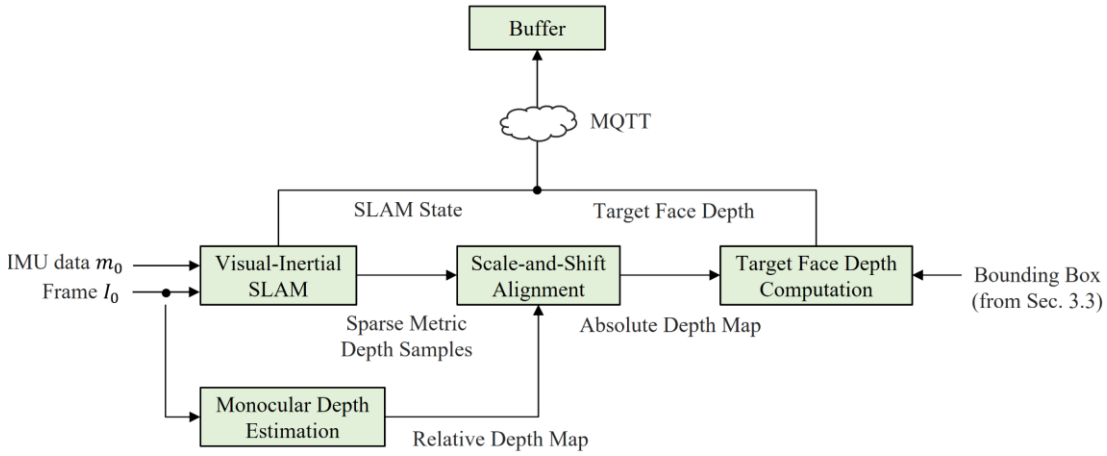


Fig. 3.3 Depth Estimation pipeline.

$$\left(s_0, \mathcal{X}_0^{\mathrm{W}}\right) = \mathrm{SLAM}(I_0, m_0), \tag{3.13}$$

where $\mathrm{SLAM}(\cdot)$ denotes a single update step of a visual–inertial SLAM estimator. We abstract away its internal operations and specify only its inputs and outputs, so that the formulation applies to a range of SLAM implementations. The SLAM state is defined as

$$s_0 \triangleq \left(R_{\mathrm{WC},0}, t_{\mathrm{WC},0}, v_0^{\mathrm{W}}, \beta_0\right), \tag{3.14}$$

where $R_{\mathrm{WC},0} \in SO(3)$ is the rotation matrix from the camera coordinate system to the world coordinate system, $t_{\mathrm{WC},0} \in \mathbb{R}^3$ is the camera position expressed in world coordinates, $v_0^{\mathrm{W}} \in \mathbb{R}^3$ is the camera velocity expressed in world coordinates, and $\beta_0 \in \mathbb{R}^2$ is the IMU bias.

We next transform the 3D map points $\mathcal{X}_0^{\mathrm{W}}$ into the camera coordinate system. For each map point $X_{0,i}^{\mathrm{W}} \in \mathcal{X}_0^{\mathrm{W}}$, its coordinates in the camera coordinate system are

$$X_{0,i}^{\mathrm{C}} = R_{\mathrm{WC},0}^{\top}\left(X_{0,i}^{\mathrm{W}} - t_{\mathrm{WC},0}\right) = \left(x_{0,i}^{\mathrm{C}}, y_{0,i}^{\mathrm{C}}, z_{0,i}^{\mathrm{C}}\right)^{\top}. \tag{3.15}$$

We then project it onto the image plane of the image I_0 and get

doi:10.6342/NTU202600774

$$\left(u_{0,i}, v_{0,i}\right)^{\top} = \pi\left(\mathrm{K}X_{0,i}^{\mathrm{C}}\right), \tag{3.16}$$

where K denotes the camera intrinsic matrix and $\pi(\cdot)$ the projection operator. We then collect all visible map points and yield a sparse set of metric depth samples

$$\mathcal{S}_0^{\mathrm{abs}} = \left\{\left(u_{0,i}, v_{0,i}, z_{0,i}^{\mathrm{C}}\right) \mid z_{0,i}^{\mathrm{C}} > 0, \left(u_{0,i}, v_{0,i}\right) \in [0, W) \times [0, H)\right\}, \tag{3.17}$$

where W and H denote the image width and height, respectively. Importantly, the same procedure is applied to any later image transmitted to the server to obtain the corresponding metric depth samples.

3.4.2 Relative Depth Map from Monocular Depth Estimation

In parallel with the SLAM pipeline, we apply a monocular depth estimation model to the same image I_0 to obtain a per-pixel relative depth map. We denote this model by $\mathrm{MDE}(\cdot)$ and write its output as

$$D_0^{\mathrm{rel}} = \mathrm{MDE}(I_0) \in \mathbb{R}^{H \times W}, \tag{3.18}$$

where D_0^{rel} is a relative depth map that assigns a relative depth value to every pixel in I_0, but these values are defined only up to an unknown global scale and shift. Next, we will align this relative depth map D_0^{rel} with the sparse metric depth samples $\mathcal{S}_0^{\mathrm{abs}}$ from the previous section to estimate this scale and shift and obtain a per-pixel metric depth map for I_0.

3.4.3 Affine Alignment to Metric Depth Samples

Ideally, the metric depth map can be obtained from the relative depth map by a global

doi:10.6342/NTU202600774

affine transformation. Specifically, we model the metric depth map as

$$D_0^{\mathrm{abs}} = aD_0^{\mathrm{rel}} + b, \tag{3.19}$$

where $D_0^{\mathrm{abs}} \in \mathbb{R}^{H\times W}$ is a metric depth map defined on the $H \times W$ pixel grid of frame I_0, and $a, b \in \mathbb{R}$ are global scale and shift parameters shared across all pixels.

In practice, we estimate the global parameters a and b by fitting the affine transformation to the sparse metric depth samples $\mathcal{S}_0^{\mathrm{abs}}$ using least squares:

$$(a^*, b^*) = \underset{a',b'}{arg\,min} \sum_{(u,v,z)\in\mathcal{S}_0^{\mathrm{abs}}} \left(a'\mathcal{I}(D_0^{\mathrm{rel}}; u, v) + b' - z\right)^2, \tag{3.20}$$

where $\mathcal{I}(D_0^{\mathrm{rel}}; u, v)$ denotes bilinear interpolation of the discrete map D_0^{rel} evaluated at the generally non-integer sample location (u, v). To reduce the influence of outlier depth samples, we adopt a two-pass fitting strategy. We first obtain an initial solution, identify inliers based on the residuals, and then refit the affine model using only these inliers. Using (a^*, b^*), we obtain the per-pixel metric depth map

$$D_0^{\mathrm{abs}} = a^* D_0^{\mathrm{rel}} + b^*. \tag{3.21}$$

Finally, we estimate the target face depth by averaging $D_0^{\mathrm{abs}}(u, v)$ over the pixels inside the target bounding box b_0:

$$z_0^{\mathrm{face}} = \frac{1}{|\mathcal{P}(b_0)|} \sum_{(u,v)\in\mathcal{P}(b_0)} D_0^{\mathrm{abs}}(u, v) \tag{3.22}$$

The resulting target face depth, together with the SLAM state s_0, is transmitted back to the glasses and stored in a buffer for subsequent delay compensation and label placement.

3.5 Delay Compensation

At this point, the glasses have determined where the name label should be placed at

doi:10.6342/NTU202600774

time ℓ. Section 3.3 provides the 2D face location $(u_\ell^{\text{face}}, v_\ell^{\text{face}})$ at time ℓ, and Section 3.4 provides the face depth estimate z_0^{face} at time 0. We approximate $z_\ell^{\text{face}} \approx z_0^{\text{face}}$, as the target face depth changes only marginally over the ℓ-frame interval relative to the tolerance implied by the diplopia threshold. Combining these results, we form an anchor estimate at time ℓ as $\hat{\mathrm{p}}_\ell = \left(u_\ell^{\text{face}}, v_\ell^{\text{face}}, z_\ell^{\text{face}}\right)$. Before rendering the label, we further mitigate the effect of the processing delay introduced by the computation on the glasses.

Due to the processing delay ϵ on the glasses, the label is rendered at time $\ell + \epsilon$ rather than at ℓ. We therefore estimate the camera motion during this delay and propagate the anchor to rendering time. Specifically, we first use the SLAM state s_0 returned by the server and the IMU measurements $\mathrm{m}_{0:\ell+\epsilon}$ to propagate the state to s_ℓ and $\mathrm{s}_{\ell+\epsilon}$:

$$\mathrm{s}_\ell = g(\mathrm{s}_0, \mathrm{m}_{0:\ell}), \qquad \mathrm{s}_{\ell+\epsilon} = \mathrm{g}(\mathrm{s}_\ell, \mathrm{m}_{\ell:\ell+\epsilon}). \tag{3.23}$$

Here, $g(\cdot)$ denotes the standard IMU-based state propagation used in visual–inertial odometry, and we omit its internal equations because this propagation is a well-known and deterministic operation. We next warp the anchor to the rendering time as

$$\hat{p}_{\ell+\epsilon} = R_{\mathrm{WC},\ell+\epsilon}^{\top}(R_{\mathrm{WC},\ell}(\mathrm{z}_\ell^{\text{face}}\mathrm{K}^{-1}\begin{bmatrix} u_\ell^{\text{face}} \\ v_\ell^{\text{face}} \\ 1 \end{bmatrix}) + t_{\mathrm{WC},\ell} - t_{\mathrm{WC},\ell+\epsilon}). \tag{3.24}$$

Finally, we use $\hat{p}_{\ell+\epsilon}$ as the final anchor and render the name label at this position using the AR stereoscopic rendering pipeline.

doi:10.6342/NTU202600774

Chapter 4 Experiments

In the first two sections of this chapter, we evaluate the performance of our system and baseline systems on benchmark datasets for face localization and depth estimation. The remaining sections describe the implementation of our system and evaluate its runtime performance in terms of power consumption, battery life, and thermal characteristics.

4.1 Evaluation of Face Localization

4.1.1 Datasets

We evaluate face localization on three public datasets: Long-Term Face Tracking (LTFT) [20], ChokePoint [21], and MobiFace [22]. LTFT consists of four outdoor and six indoor videos. The scenes are highly crowded, with between 24 and 148 subjects per video, and both the subjects and the camera move throughout each video. This combination of crowding and motion makes LTFT the most challenging dataset in our evaluation. ChokePoint consists of 48 indoor surveillance videos recorded by fixed cameras overlooking a corridor. The camera remains static while a small number of subjects walk through the field of view, producing frequent but relatively simple motion patterns. MobiFace consists of 80 mobile videos recorded by smartphone users. Most sequences are selfie-style, so faces typically appear close to the camera, and both the camera and the subjects exhibit only limited motion. Overall, ChokePoint and MobiFace are less challenging for face localization than LTFT, yet they cover complementary configurations of subject and camera motion that are representative of practical use cases.

doi:10.6342/NTU202600774

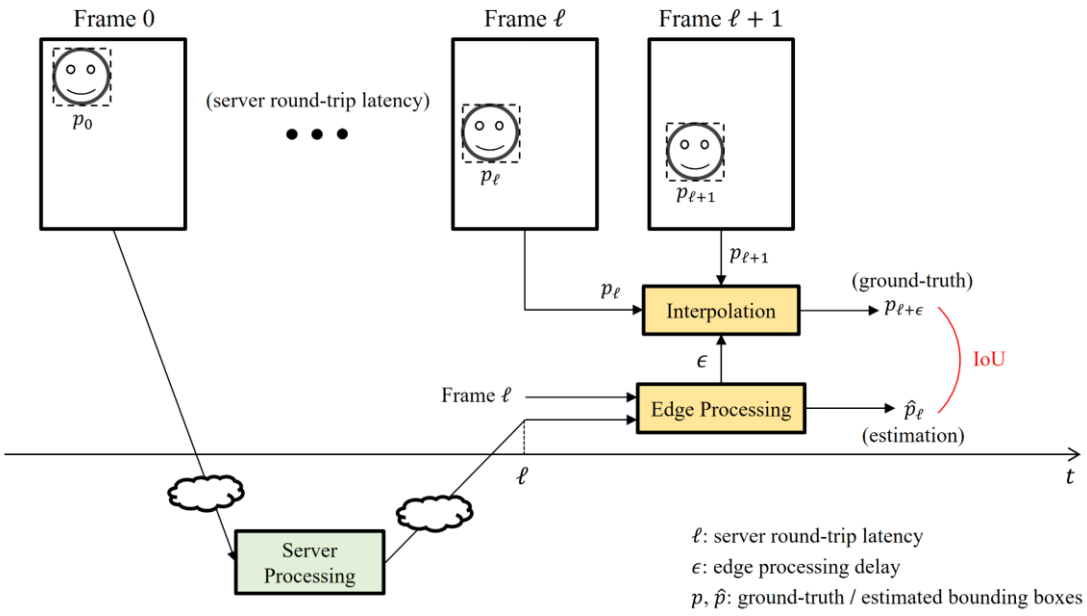


Fig. 4.1 Latency-aware evaluation protocol for localization. We align the prediction with the ground truth by linearly interpolating the ground-truth boxes between frames ℓ and $\ell+1$.

4.1.2 Metrics and Latency-Aware Evaluation Protocol

We use two metrics, intersection-over-union (IoU) and Accuracy. For each evaluated frame, we first compute the IoU between the estimated face bounding box and the corresponding ground-truth bounding box. Accuracy is then defined as the proportion of frames whose IoU exceeds 0.5.

To mimic the behavior of the deployed system, we explicitly account for both the server round-trip latency and the edge processing delay during evaluation. All delays are expressed in frames. We denote the server round-trip latency by ℓ and the edge processing delay by ϵ. During evaluation, we simulate the behavior of the deployed system using the protocol in Fig. 4.1. For each frame, the server output is delayed by ℓ frames and combined with edge processing that incurs an additional delay ϵ on the glasses. The resulting bounding box is treated as the label that would be displayed after both delays, i.e., at the display time of that frame. To obtain the ground-truth bounding box at this

doi:10.6342/NTU202600774

display time, we linearly interpolate between the ground-truth boxes in the two adjacent frames, and then compute IoU and Accuracy over all frames.

4.1.3 Experimental Setup and Hardware Platforms

We evaluate five system configurations for face localization: Cloud Detection [2], MediaPipe [5], Edge Mean Shift [6], Edge Detection [4], and the proposed system. We adapt these methods from prior work to our task, and their cloud–edge functional partitioning is summarized in Table 4.1. To enable a fair comparison, we standardize the face detection model across all configurations. Whenever a configuration relies on an object detector, we replace its detector with the same single-shot face detector [23].

System	**Edge**	**Cloud**
Cloud Detection [2]	-	Face Detection
MediaPipe [5]	Kanade-Lucas-Tomasi Feature Tracker	Face Detection
Edge MeanShift [6]	Mean Shift	Face Detection
Edge Detection [4]	Face Detection	-
Proposed	Siamese Network (search branch)	Siamese Network (template branch)

Table 4.1 Functional partitioning of the evaluated systems.

In Section 4.1.4, we evaluate these systems on devices spanning various compute power levels. Specifically, we consider four representative device types in increasing order of compute power: AR glasses, smartphone, laptop, and server, as listed in Table 4.2. For evaluation, we adopt the protocol in Fig. 4.1. For each system configuration on each hardware platform, we first measure its average processing time per frame and use this value to set the corresponding edge processing delay ϵ. We then evaluate localization

doi:10.6342/NTU202600774

accuracy offline by applying the latency-aware protocol with a fixed server round-trip latency ℓ of 100ms for all configurations and hardware platforms.

Device Type	Product	Compute Unit	Compute Power (GFLOPs)
AR Glasses	RayNeo X2	Qualcomm Snapdragon XR2	1,250
Smartphone	Samsung Galaxy S25	Qualcomm Snapdragon 8 Elite	1,950
Laptop	Asus TUF Gaming FX504	NVIDIA GeForce GTX 1050 Ti Mobile	2,488
Server	-	NVIDIA GeForce RTX 4090	82,580

Table 4.2 Evaluation platforms spanning various compute power levels.

In Section 4.1.5, we evaluate these systems under various network latency settings. In this set of experiments, all configurations share the same hardware setup, where the edge components run on the RayNeo X2 glasses and the cloud components run on a server equipped with an NVIDIA GeForce RTX 4090 GPU. We use the edge processing delay ϵ measured on the RayNeo X2 and evaluate each system configuration offline under a set of pre-defined server round-trip latency values ℓ, while keeping the protocol in Fig. 4.1 unchanged.

4.1.4 Localization Accuracy across Various Compute Power Levels

In this section, we first measure the average edge-side processing time of each configuration on each hardware platform to set the corresponding edge processing delay ϵ, and then evaluate localization performance under a fixed server round-trip latency ℓ of 100ms.

Table 4.3 reports the measured edge processing time of all system configurations under various compute power levels. For each configuration and each edge device, we

doi:10.6342/NTU202600774

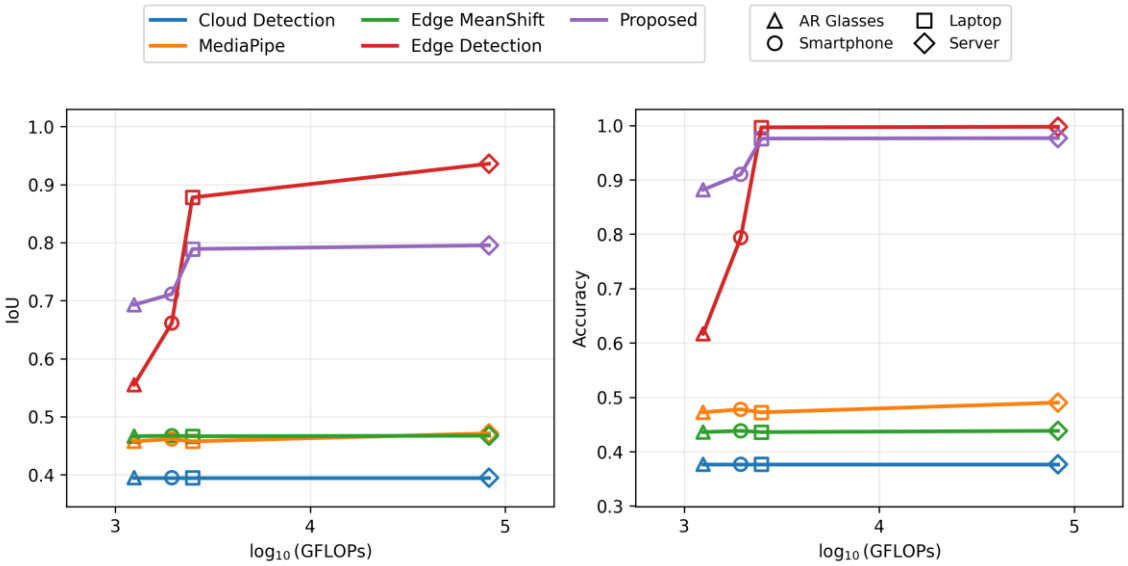
measure the edge-side processing time per frame and report the average over all evaluated frames. As expected, configurations with lightweight tracking on the glasses, such as Edge MeanShift and MediaPipe, incur relatively small edge processing time across all platforms. In contrast, Edge Detection is substantially more compute-intensive on the edge device and therefore exhibits the largest edge processing time on low-power platforms. The proposed system incurs a moderate edge processing time because the glasses execute only the Siamese search branch, while the heavier template branch is offloaded to the server. Overall, Table 4.3 shows that increasing edge compute power reduces the edge processing time for all configurations that perform non-trivial computation on the edge.

	AR Glasses	Smartphone	Laptop	Server
Cloud Detection [2]	0ms	0ms	0ms	0ms
MediaPipe [5]	9ms	7ms	8ms	1ms
Edge MeanShift [6]	2ms	1ms	1ms	1ms
Edge Detection [4]	78ms	48ms	12ms	5ms
Proposed	37ms	27ms	7ms	3ms

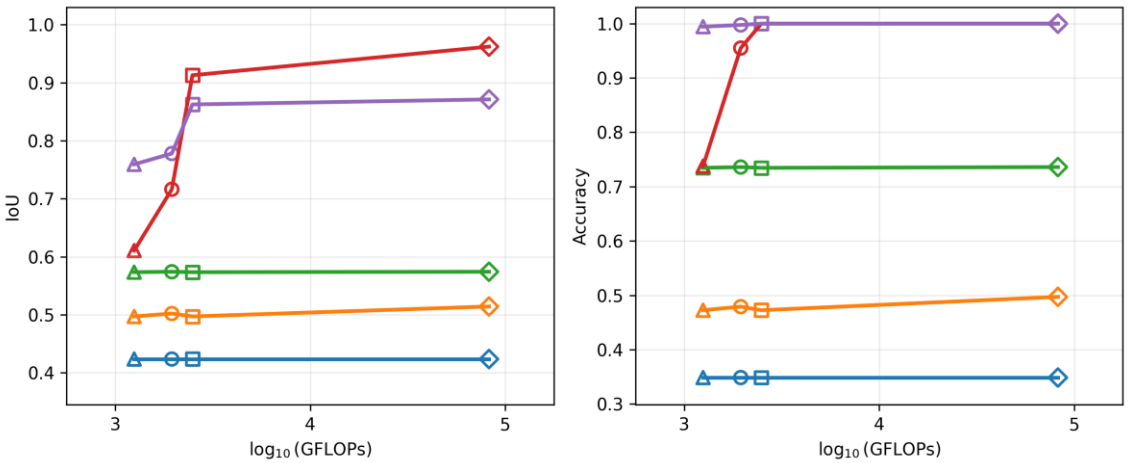
Table 4.3 Edge processing time of evaluated systems under various computing power levels.

We next examine localization performance as a function of compute power. Fig. 4.2 plots the resulting IoU and Accuracy on the three datasets under various compute power levels. On LTFT and ChokePoint, the proposed system achieves better localization performance under low compute power levels. As compute power increases, the performance of Edge Detection gradually catches up with that of the proposed system and

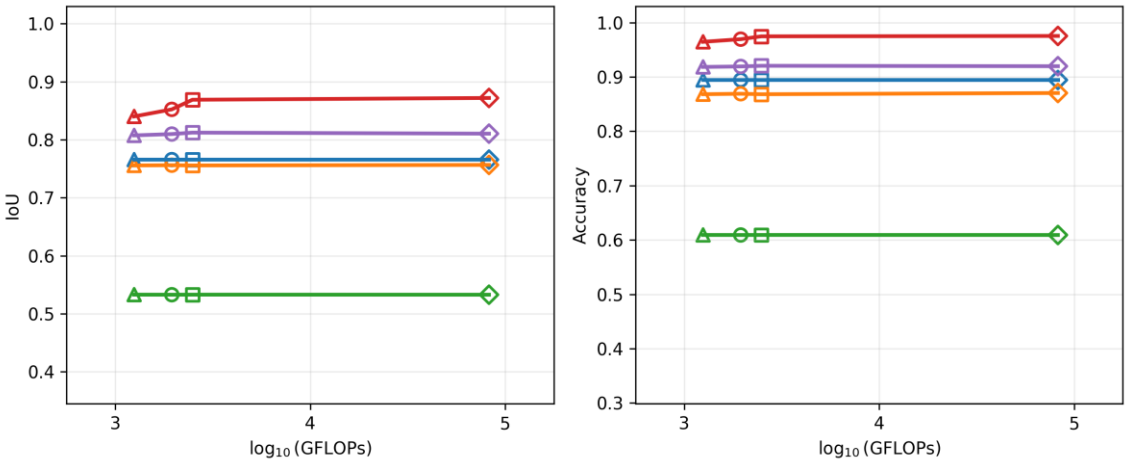
doi:10.6342/NTU202600774



(a) LTFT



(b) ChokePoint



(c) MobiFace

Fig. 4.2 Localization performance under various compute power levels.

doi:10.6342/NTU202600774

slightly surpasses it when compute power reaches the laptop level and above. This trend suggests that under limited compute resources, the detection-based approach is more affected by its higher computational cost, whereas this effect becomes less pronounced when sufficient compute resources are available. In contrast, the proposed system maintains relatively strong performance even under low compute power levels. On MobiFace, where the relative motion is small and latency-induced misalignment is less pronounced, localization performance remains stable across compute power levels, and Edge Detection achieves the best overall results.

4.1.5 Localization Accuracy under Various Network Latency Settings

In this section, we evaluate localization accuracy under various server round-trip latency settings on the RayNeo X2 glasses. We test all system configurations on the LTFT, ChokePoint, and MobiFace datasets using the latency-aware protocol in Fig. 4.1. The edge processing delay ϵ for each configuration is set according to its measured average processing time per frame on the RayNeo X2 reported in Section 4.1.4. We then evaluate each configuration offline under a set of pre-defined server round-trip latency values ℓ from 0 to 300 ms. The resulting IoU and Accuracy are shown in Fig. 4.3.

Among all configurations, Cloud Detection is the most sensitive because its IoU and Accuracy drop sharply as ℓ increases, as the glasses directly use the delayed bounding boxes returned from the server. The tracking-based configurations, MediaPipe and Edge MeanShift, also show a clear degradation in performance as ℓ increases. Their IoU and Accuracy decrease steadily with larger server round-trip latency, since a larger ℓ implies that the tracker must follow the face across more frames, which increases the chance of occlusions and the accumulation of tracking error. Edge Detection relies purely on face detection on the glasses and is therefore largely insensitive to server round-trip latency,

doi:10.6342/NTU202600774

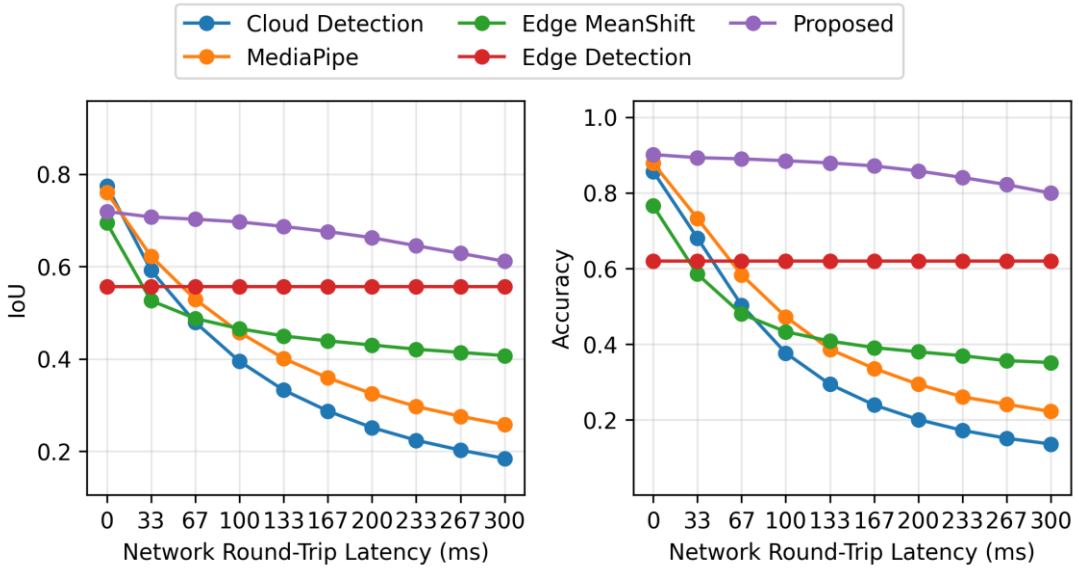
resulting in relatively stable performance as ℓ increases. However, its larger edge processing delay ϵ can make it less favorable on LTFT and ChokePoint, where the targets exhibit substantial motion. On MobiFace, where the relative motion is small, Edge Detection is slightly better than the proposed system. Finally, the proposed system remains relatively insensitive to server round-trip latency. Its performance degrades more gradually than the tracking-based configurations and Cloud Detection as ℓ increases. In particular, while the average IoU decreases with larger latency, the IoU on most frames remains above 0.5, so the Accuracy drops at a slower rate.

To provide a representative snapshot for quantitative comparison, Table 4.3 reports the localization performance of all configurations under a fixed server round-trip latency of 100ms. Consistent with the trends in Fig. 4.3, the proposed system achieves the best performance on LTFT and ChokePoint, while Edge Detection performs best on MobiFace, where the relative motion is small, making the larger edge processing delay less harmful.

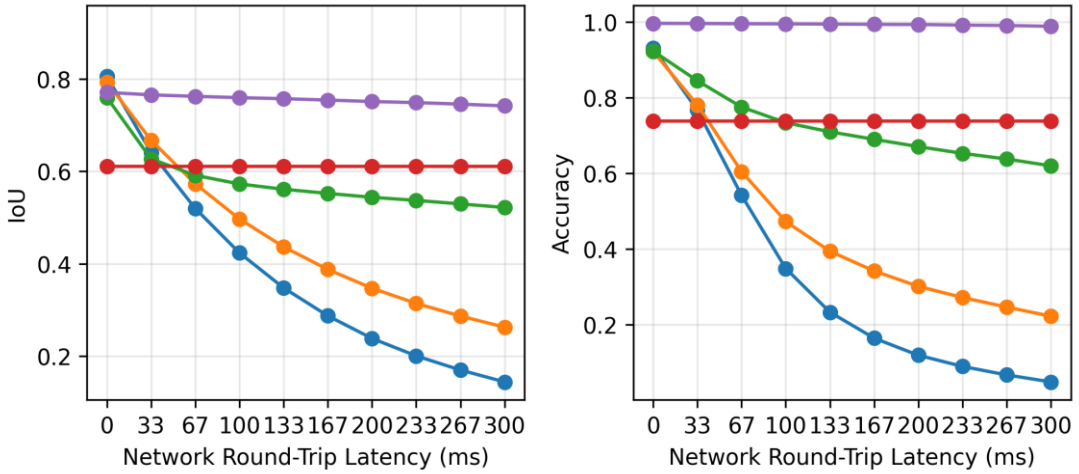
System	LTFT		ChokePoint		MobiFace	
	IoU	Accuracy	IoU	Accuracy	IoU	Accuracy
Cloud Detection [2]	0.39483	0.37673	0.42360	0.34868	0.76573	0.89443
MediaPipe [4]	0.45754	0.47237	0.49704	0.47263	0.75570	0.86824
Edge MeanShift [5]	0.46544	0.43295	0.57261	0.73366	0.53309	0.60943
Edge Detection [6]	<u>0.55639</u>	<u>0.61906</u>	<u>0.61069</u>	<u>0.73785</u>	**0.83975**	**0.96453**
Proposed	**0.69670**	**0.88461**	**0.75926**	**0.99437**	<u>0.80746</u>	<u>0.91846</u>

Table 4.4 Localization performance of the evaluated systems on the three datasets under 100ms server round-trip latency

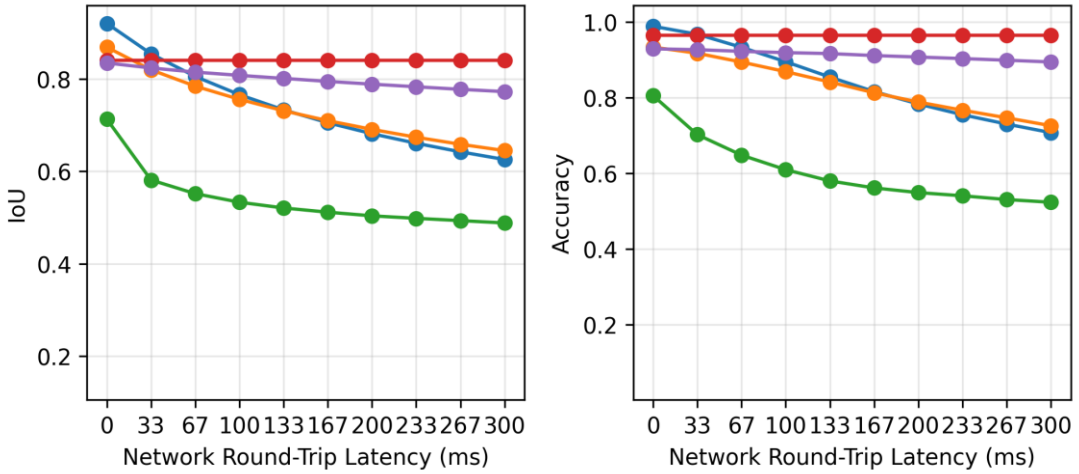
doi:10.6342/NTU202600774



(a) LTFT



(b) ChokePoint



(c) MobiFace

Fig. 4.3 Localization performance versus network round-trip latency.

doi:10.6342/NTU202600774

4.2 Evaluation of Depth Estimation

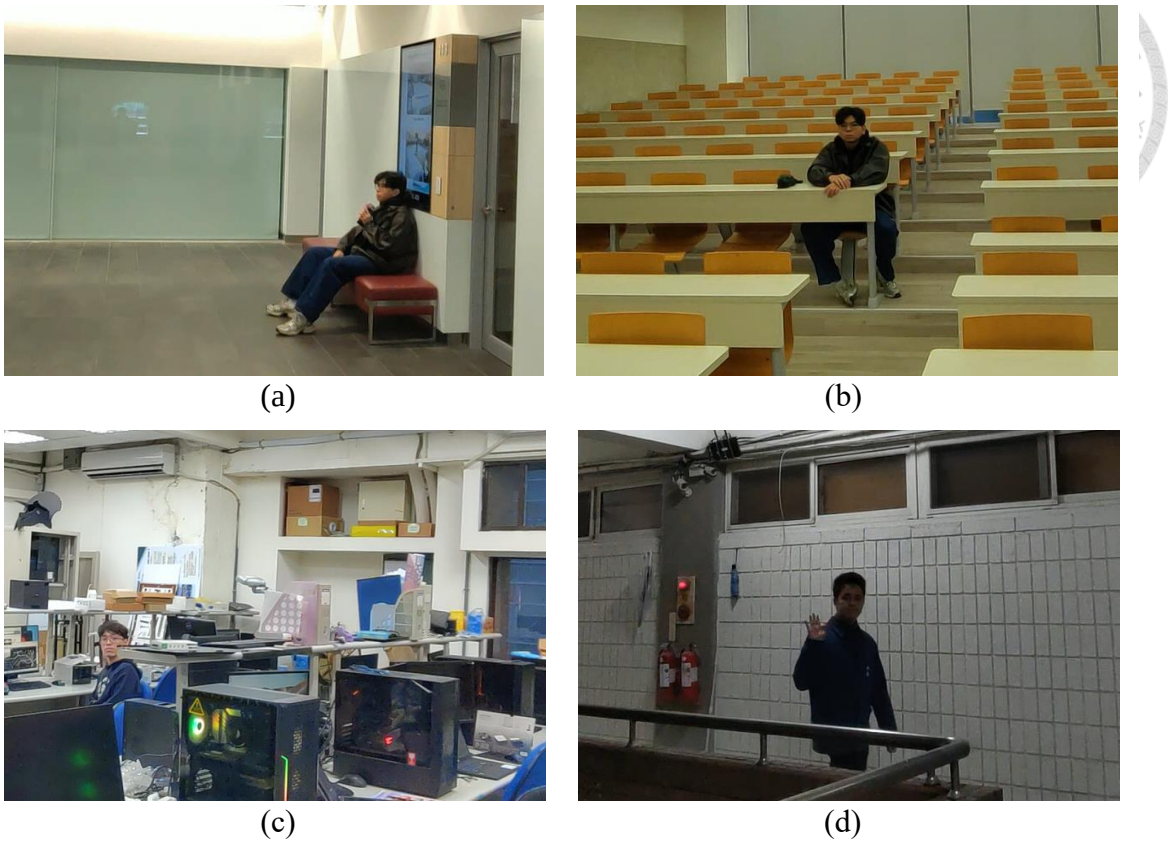
4.2.1 Data Collection

We evaluate the proposed depth estimation pipeline using data captured with RayNeo X2 AR glasses equipped with a monocular RGB camera and an IMU. The recordings are collected from common social settings and representative motion patterns, and include four scenes, Lobby, Classroom, Laboratory, and Corridor. Together, these scenes cover various combinations of camera motion and target motion, including both static and moving cases. Example first-person views of the four scenes are shown in Fig. 4.4, captured by the camera on the AR glasses. are recorded at 30 fps with an image resolution of 1920×1080. IMU measurements are logged at 200 Hz and timestamped together with the camera frames. Across the four scenes, the user–target distance spans approximately 1–5 m. For the remaining experiments, we report results by analyzing each scene separately.

4.2.2 Perceptually Motivated Metrics and Evaluation Protocol

We evaluate face depth on a per-frame basis for the videos in Section 4.2.1. For each frame, we construct a ground truth from facial landmarks. Specifically, we first measure the physical distance between the landmarks on the target face. Using this reference, we then detect the landmarks of the target face in each frame, and estimate the face depth from the distance between these landmarks in pixels. Landmarks are detected using a CNN-based facial landmark detector [24]. To improve the reliability of the ground truth, we restrict evaluation to frames where the target face is near-frontal. Those frames where the target face is non-frontal are excluded because out-of-plane head rotation introduces foreshortening, which biases the inferred depth.

doi:10.6342/NTU202600774



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

Fig. 4.4 Example first-person views from the four scenes in our evaluation dataset, captured by the camera on the RayNeo X2 AR glasses. (a) Lobby. (b) Classroom. (c) Laboratory. (d) Corridor.

We adopt a perceptually motivated metric, called the double-vision-free ratio, defined as the fraction of frames in which the depth estimation error is small enough to avoid double vision. To decide when double vision occurs, we use the diplopia threshold defined in [24], and we set this threshold to 2.92mrad by averaging the diplopia thresholds across subjects reported in that study. This threshold is an angular quantity, and its relationship with the depth estimation error is illustrated in Fig. 4.5. For each frame, we convert the depth estimation error into viewing-angle difference

$$\theta = \frac{\mathrm{IPD}}{2}\left|\frac{1}{z_{\mathrm{est}}} - \frac{1}{z_{\mathrm{gt}}}\right|, \tag{4.1}$$

doi:10.6342/NTU202600774

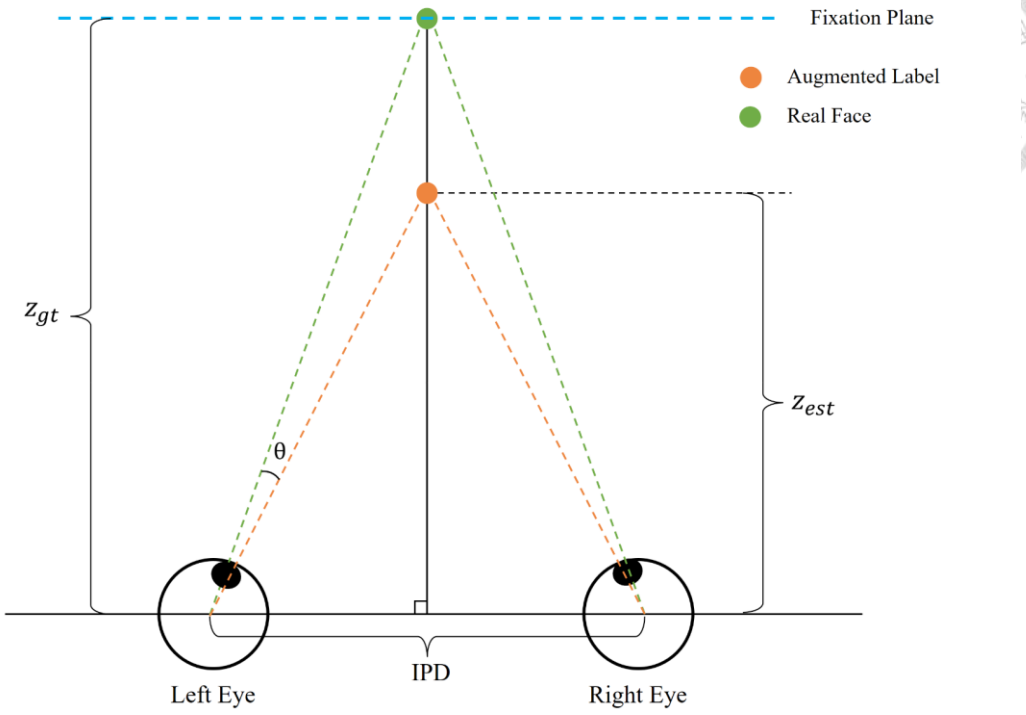


Fig. 4.5 Double vision condition. The real face is at the ground-truth depth z_{gt}, while the label is rendered at the estimated depth z_{est}. This depth mismatch produces an viewing-angle difference θ. Double vision occurs when θ exceeds the diplopia threshold.

where z_{gt} is the ground-truth face depth, z_{est} is the estimated face depth, and IPD denotes the interpupillary distance. We say that double vision occurs when $\theta > 2.92\text{mrad}$. The double-vision-free ratio is then computed as the fraction of frames with $\theta \leq 2.92\text{mrad}$, so higher values indicate better depth estimation. We also compute the absolute error and relative error as auxiliary metrics. All metrics are computed on the same set of valid frames, after excluding frames with non-frontal face.

4.2.3 Compared Methods and Implementation Details

We compare the proposed depth estimation pipeline with two baselines. For fairness,

doi:10.6342/NTU202600774

both baselines use the same face bounding boxes produced by the face detector in Section 4.1 [23], and all methods are evaluated on the same set of valid frames defined in Section 4.2.2. The first baseline is Face-Width Prior. It estimates the depth from the width of the detected face bounding box. In principle, this approach would require knowing the physical face width of the target, which is not available in general. We therefore approximate this quantity by a fixed average face width and set the reference value to 13.5 cm, obtained as the average of the reported male and female mean face widths in an anthropometric survey [25]. Typical adult face widths vary by roughly one standard deviation of about 0.6–0.7 cm, so for a randomly chosen adult our assumed width can be expected to differ from the true width by a similar proportion. This inter-subject variation, together with inaccuracies in the face width estimates from the face detector, induces an expected source of error. We therefore treat the Face-Width Prior as a simple, naive baseline for converting face size to depth in practice.

The second baseline is Depth Anything V2 [9], a recent state of the art model for monocular depth estimation. It is trained on large scale synthetic datasets with ground truth depth and on real images with pseudo labeled depth, and its variants for metric depth estimation are further finetuned on indoor datasets with metric supervision such as Hypersim so that the model predicts depth in meters. This baseline reflects the performance of a modern method that relies only on RGB input and allows us to check whether such a method is sufficient for the accuracy requirements of our label placement task. In our evaluation, we apply the model to each frame independently and estimate the face depth by averaging the predicted depth values within the face bounding box on the depth map.

doi:10.6342/NTU202600774

4.2.4 Depth Estimation Results

This section reports the depth estimation results for the three compared methods.

Method	Lobby	Classroom	Laboratory	Corridor
Face-Width Prior	0.9867	0.9871	0.7937	0.6484
Depth Anything V2	**0.9973**	0.9796	0.7411	0.4062
Proposed	0.9925	**0.9903**	**0.9981**	**0.9298**

Table 4.5 Double-vision-free ratio across scenes.

The double-vision-free ratio is summarized in Table 4.3, and the distributions of relative and absolute depth estimation errors are shown in Fig. 4.6 and Fig. 4.7. We analyze the results by scene. The Face-Width Prior, however, is less affected by scene characteristics. Because it uses a fixed nominal face width as a prior, its performance variation is mainly driven by inter-subject differences in face width.

In Lobby and Classroom, the environments are indoor and provide relatively stable visual structure. All methods achieve a high double-vision-free ratio, indicating that their depth estimates are generally accurate enough to satisfy the diplopia threshold in these conditions. The proposed method is not always the most accurate in terms of the absolute error or relative error, but its error distribution remains within a range that rarely triggers double vision.

In Laboratory, the scene is more challenging due to windows, which introduce long-range background content and reflections. Depth Anything V2 degrades substantially, exhibiting a wider error distribution and more outliers in both absolute and relative errors, which leads to a clear drop in the double-vision-free ratio, whereas the proposed method remains stable and maintains a high double-vision-free ratio.

doi:10.6342/NTU202600774

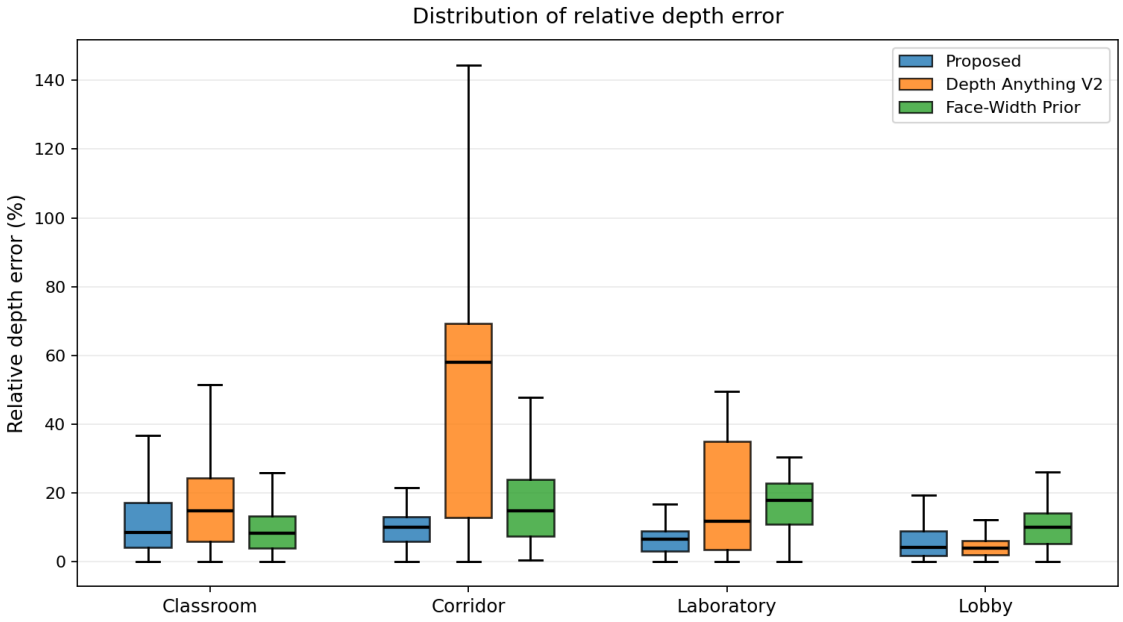


Fig. 4.7 Distribution of relative depth estimation error across scenes.

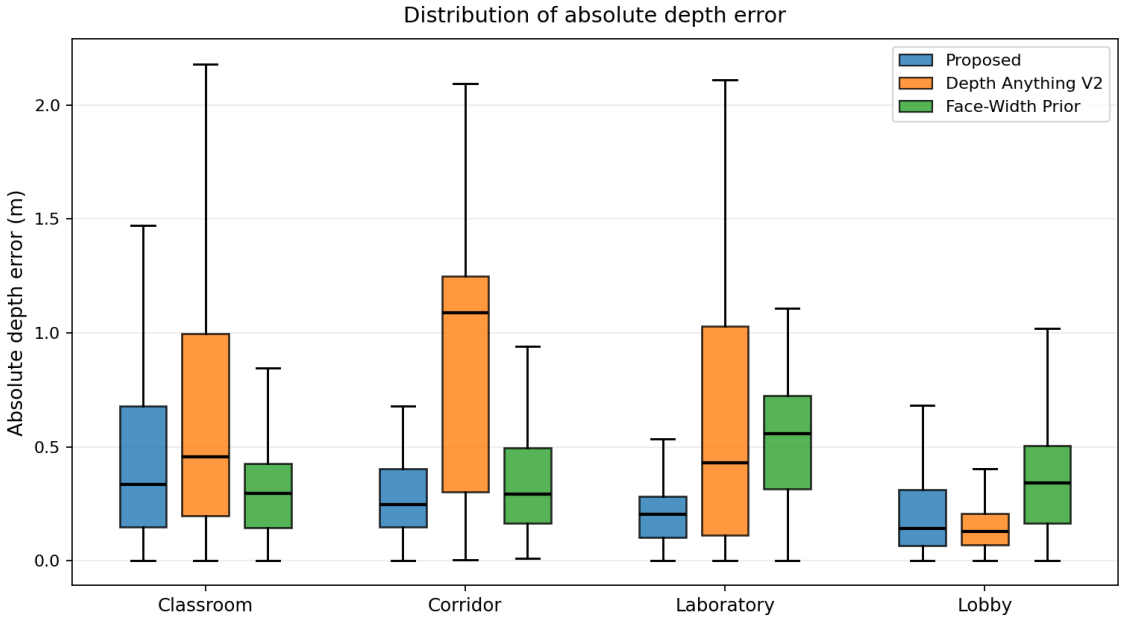


Fig. 4.6 Distribution of absolute depth estimation error across scenes.

In Corridor, the scene is the most difficult among the four due to long-range views, uneven brightness, and fewer trackable feature points. Depth Anything V2 is strongly affected by the long-range views in this scene and frequently produces large overestimation, resulting in a severe drop in the double-vision-free ratio.

doi:10.6342/NTU202600774

4.3 System Implementation and Runtime Evaluation

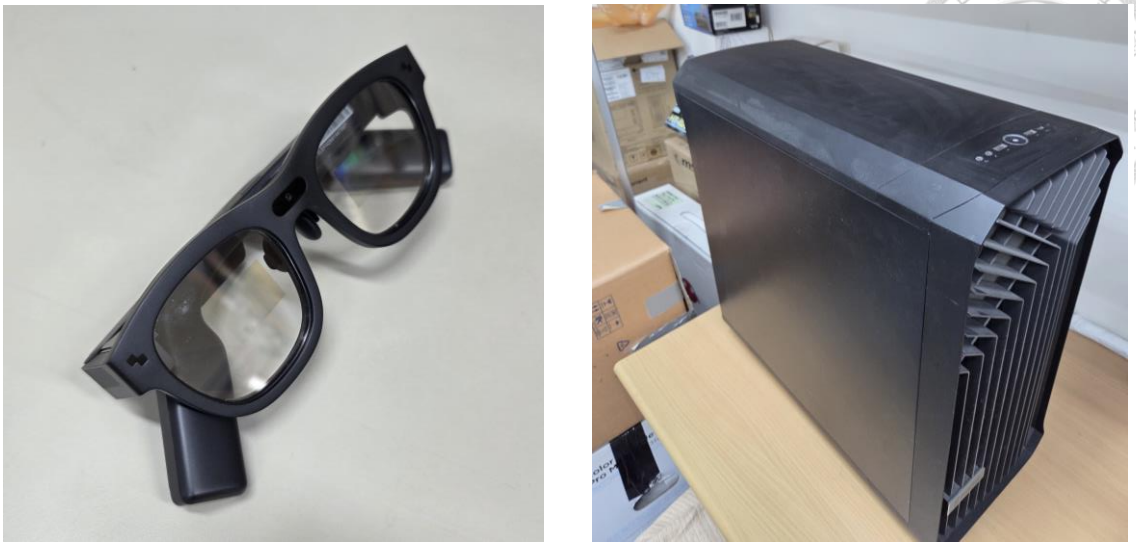
4.3.1 Implementation Details

To provide the necessary context for reproducibility, we summarize the software stack and execution environments of the implementation of our system. Fig. 4.8 shows the hardware setup, and Table 4.3 summarizes the implementation details of the main operations, including the programming languages, third-party libraries, and model formats used on the glasses and on the server.

On the glasses side, the system runs on RayNeo X2 AR glasses with Android as the operating system. The application is developed in Unity, so most glasses-side operations are implemented in C#. For data acquisition, video frames and IMU measurements are obtained via a Java-based plugin on Android that exposes the native camera API and IMU data to Unity. Facial landmark detection is performed by SeetaFace2, compiled as a native C++ plugin and invoked from C#. The face localization branch on the glasses executes an ONNX model through Unity Barracuda. MQTT communication is implemented using the uPLibrary package.

On the server side, the system runs on a Linux PC equipped with an NVIDIA RTX 4090 GPU. As summarized in Table 4.3, the server-side operations are primarily implemented in Python. MQTT communication uses the paho-mqtt library. Face recognition employs AdaFace exported to TorchScript, and the server-side face localization network is likewise deployed in TorchScript. Depth estimation relies on Video Depth Anything, also executed via TorchScript. For SLAM, we run the official VINS-Mono distribution inside a Docker container that provides a ROS environment, and we interface the ROS pipeline with the Python processes through rosbridge.

doi:10.6342/NTU202600774



(a) (b)

Fig. 4.8 Hardware setup of our implementation. (a) RayNeo X2 AR glasses. (b) A server running Linux equipped with an NVIDIA RTX 4090 GPU.

Operation	**Programming Language**	**Third-Party Library**	**Model Format**
Data Acquisition	Java	-	-
Face Localization (glasses)	C#	-	ONNX
Landmark Detection	C++	SeetaFace2 [26]	-
MQTT (glasses)	C#	uPLibrary	-
MQTT (server)	Python	Paho MQTT	-
Face Recognition	Python	AdaFace [27]	TorchScript
Face Localization (server)	Python	-	TorchScript
SLAM	C++	VINS-Mono [28]	-
Depth Estimation	Python	Video Depth Anything [29]	TorchScript

Table 4.3 Implementation details of the main operations on the glasses and the server.

doi:10.6342/NTU202600774

4.3.2 Power, Battery, and Thermal Characteristics

In this section, we characterize the power consumption, battery life, and temperature of the RayNeo X2 AR glasses when running the proposed system using the implementation described in Section 4.3.1. All measurements were conducted at an ambient temperature of 24 °C and are initialized from a fully charged battery.

We first measure power consumption. To isolate the power contribution of each operation, we evaluate a set of incremental configurations. The initial configuration keeps the display continuously enabled. We then enable the camera. Next, we enable Network transmission to send captured frames to the server and receive the returned results. Finally, we enable the full pipeline including face localization. As shown in Fig. 4.9, power consumption is dominated by the display and the camera, which consume about 0.97 W and 0.96 W, respectively. Network communication adds about 0.15 W, which is substantially smaller. Face localization contributes only about 0.04 W, indicating that its power consumption is negligible compared with the display and camera.

We then measure battery life under the same four incremental configurations. As shown in Fig. 4.10, the results closely follow the power measurements. With the display enabled only, the glasses operate for about 2 hours. With all components enabled, the system operates for about 1 hour.

Finally, we measure temperature. As shown in Fig. 4.11, the CPU and GPU temperatures increase at the beginning of operation and then converge to a steady state. This convergence indicates that the average heat generation is balanced by the device heat dissipation capability. As the temperature does not increase without bound, the proposed system can operate continuously without overheating.

doi:10.6342/NTU202600774

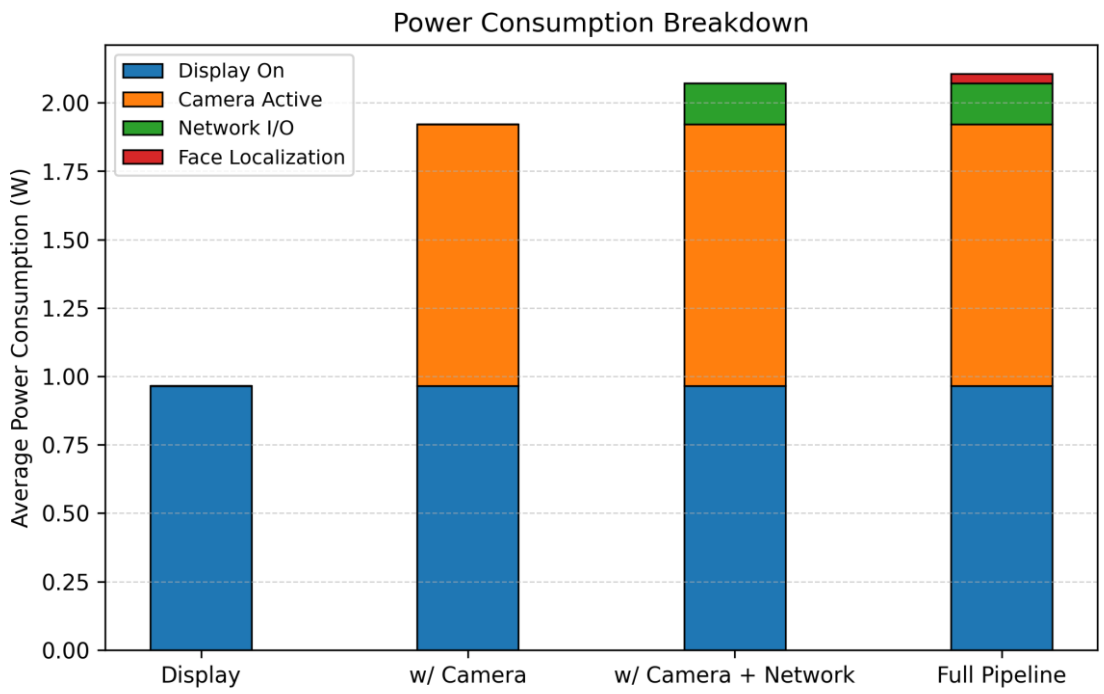


Fig. 4.9 Measured average power consumption under four incremental system configurations.

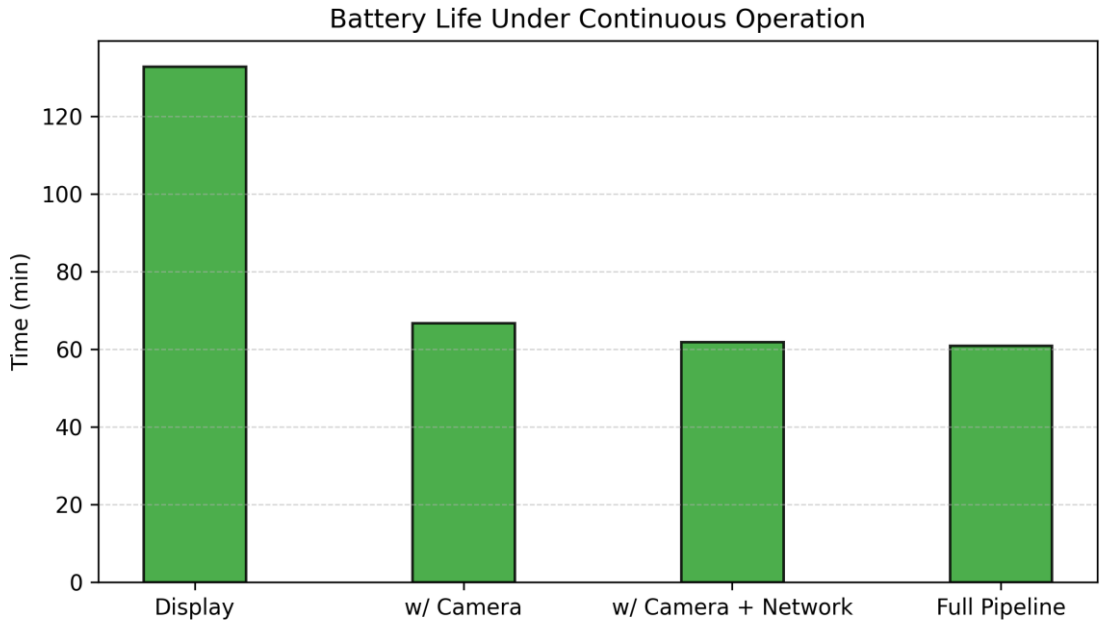


Fig. 4.10 Measured battery life under continuous operation for four incremental system configurations.

doi:10.6342/NTU202600774

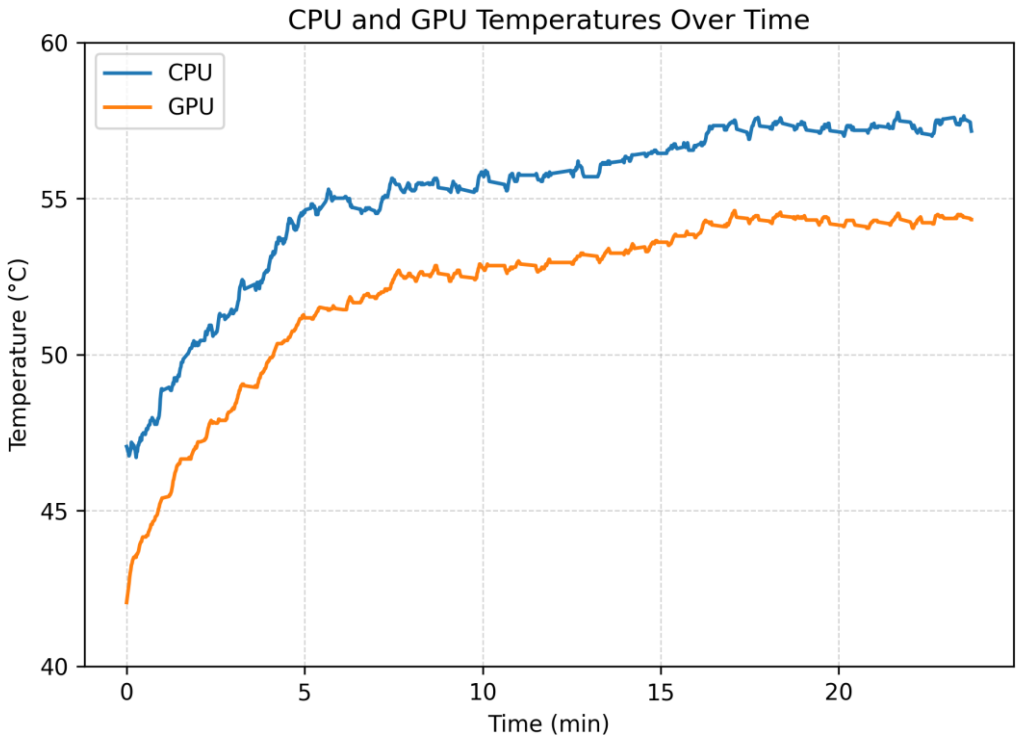


Fig. 4.11 CPU and GPU temperatures of the RayNeo X2 during continuous operation. The curves show the measured temperature over time under the full pipeline configuration at an ambient temperature of 24 °C.

doi:10.6342/NTU202600774

Chapter 5 Discussion and Future Work

Although the proposed system is designed under stringent practical constraints, it still has limitations to be addressed and clear room for improvement. We first discuss the requirements and limitations of the current design and then outlines promising directions for future work.

First, the proposed system still imposes a non-negligible computational load on the AR glasses. Its performance may therefore degrade on more severely resource-constrained devices, which his motivates further effort toward reducing the on-glasses computational load. In this thesis, we showed that a division-of-labor strategy can already yield substantial benefits. A natural next step is to explicitly design and train the model for split execution. For example, future work could leverage neural architecture search to jointly determine the network architecture and the partition point under constraints, using the glasses-side computational cost as a regularization term in the training objective to balance localization accuracy against on-glasses computation. This would encourage a lightweight glasses-side component while preserving localization accuracy, enabling the system to generalize to a broader range of AR glasses.

Second, the proposed depth estimation pipeline depends on SLAM to obtain metric scale. This dependency introduces a key limitation in outdoor and wide-open environments, where SLAM can be unstable due to insufficient reliable features. When SLAM degrades, the subsequent scale calibration becomes unreliable and may even underperform a monocular depth estimation model used alone. A promising approach is to develop a single multimodal model that takes images and IMU measurements as input and directly predicts metric depth. This approach removes the hard dependency on SLAM and avoids SLAM-induced failure in challenging environments.

doi:10.6342/NTU202600774

Chapter 6 Conclusion

In this thesis, we present a face tagging system for lightweight AR glasses that enables reliable label placement under round-trip latency, limited on-glasses computing resources, and the absence of a depth sensor. Through glasses–server collaboration, the proposed system provides real-time on-glasses face localization and metric depth estimation for moving targets. Evaluated under a latency-aware protocol on three public datasets, our face localization achieves an IoU above 0.5 in 93.25% of frames, outperforming on-glasses face detection at 77.38% and server-side face detection at 53.99%. In addition, averaged across four representative social scenarios, our depth estimation keeps the depth error below the diplopia threshold in 96.15% of evaluation frames, surpassing the Face-Width Prior baseline at 85.40% and Depth Anything V2 at 78.11%. Overall, these results demonstrate that tagging moving faces on lightweight AR glasses is feasible under practical constraints. We believe that this glasses–server collaborative architecture can be extended to other AR tagging applications that require reliable label placement.

doi:10.6342/NTU202600774

REFERENCES

[1] A. L. Duwaer and G. Van Den Brink, "What is the diplopia threshold?" *Perception & Psychophysics*, vol. 29, no. 4, pp. 295–309, 1981.

[2] D. A. B. Rahardjo and H. H. Chen, "Cloud-Based Face Recognition for Augmented Reality Glasses," *IEEE International Symposium on Mixed and Augmented Reality Adjunct*, pp. 685–688, 2023.

[3] J. Liao, Y. Xu, Y. Guan, and G. Liu, "An Augmented Classroom Teaching System based on AR and Facial Recognition," *The Journal of Applied Instructional Design*, vol. 13, no. 2, pp. 11–16, 2024.

[4] M. Łysakowski, K. Żywanowski, A. Banaszczyk, M. R. Nowicki, P. Skrzypczyński, and S. K. Tadeja, "Real-time onboard object detection for augmented reality: Enhancing head-mounted display with YOLOv8," *IEEE International Conference on Edge Computing and Communications*, pp. 364–371, 2023.

[5] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "MediaPipe: A Framework for Building Perception Pipelines," *Proceedings of the Third Workshop on Computer Vision for AR/VR at IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[6] A. Farasin, F. Peciarolo, M. Grangetto, E. Gianaria, and P. Garza, "Real-time Object Detection and Tracking in Mixed Reality using Microsoft HoloLens," in *Proceedings of the 15th International Joint Conference on Computer Vision*, Imaging and Computer Graphics Theory and Applications, vol. 4, pp. 165–172, 2020.

doi:10.6342/NTU202600774

[7] S. Avidan and A. Shashua, “Trajectory Triangulation: 3D Reconstruction of Moving Points from a Monocular Image Sequence,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 4, pp. 348–357, 2000.

[8] H. S. Park, T. Shiratori, I. Matthews, and Y. Sheikh, "3D Reconstruction of a Moving Point from a Series of 2D Projections," *European Conference on Computer Vision*, vol. 6313, pp. 158–171, 2010.

[9] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, "Depth Anything V2," *Advances in Neural Information Processing Systems*, vol. 37, pp. 21875–21911, 2024.

[10] B. Mandal, S.-C. Chia, L. Li, V. Chandrasekhar, C. Tan, and J.-H. Lim, "A Wearable Face Recognition System on Google Glass for Assisting Social Interactions," *Asian Conference on Computer Vision Workshops*, vol. 9010, pp. 419–433, 2015.

[11] O. Daescu, H. Huang, and M. Weinzierl, "Deep learning based face recognition system with smart glasses," in *Proceedings of the 12th ACM International Conference on Pervasive Technologies Related to Assistive Environments*, pp. 218–226, 2019.

[12] C. McKelvey, R. Dreyer, D. Zhu, W. Wang, and J. Quarles, "Energy-Oriented Designs of an Augmented-Reality Application on a Vuzix Blade Smart Glass," in *Proceedings of the IEEE Tenth International Green and Sustainable Computing Conference*, pp. 1–8, 2019.

[13] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography," *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.

[14] R. Smith, M. Self, and P. Cheeseman, "A Stochastic Map for Uncertain Spatial Relationships," in *Proceedings of the Fourth International Symposium on Robotics*

doi:10.6342/NTU202600774

Research, pp. 467–474, 1988.

[15] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into Self-Supervised Monocular Depth Estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3828–3838, 2019.

[16] Y. Long, H. Yu, and B. Liu, "Depth completion towards different sensor configurations via relative depth map estimation and scale recovery," *Journal of Visual Communication and Image Representation*, vol. 80, art. no. 103272, 2021.

[17] J. Bromley, J. W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Säckinger, and R. Shah, "Signature verification using a Siamese time delay neural network," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 4, pp. 669–688, 1993.

[18] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-convolutional Siamese networks for object tracking," *European Conference on Computer Vision Workshops*, vol. 9914, pp. 850–865, 2016.

[19] V. Borsuk, R. Vei, O. Kupyn, T. Martyniuk, I. Krashenyi, and J. Matas, "FEAR: Fast, Efficient, Accurate and Robust Visual Tracker," *European Conference on Computer Vision, Lecture Notes in Computer Science*, vol. 13682, pp. 644–663, 2022.

[20] G. Barquero, C. Fernández Tena, and I. Hupont, "Long-Term Face Tracking for Crowded Video-Surveillance Scenarios," *IEEE International Joint Conference on Biometrics*, pp. 42–49, 2020.

[21] Y. Wong, S. Chen, S. Mau, C. Sanderson, and B. C. Lovell, "Patch-based Probabilistic Image Quality Assessment for Face Selection and Improved Video-based Face Recognition," *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 81–88, 2011.

doi:10.6342/NTU202600774

[22] Y. Lin, S. Cheng, J. Shen, and M. Pantic, "MobiFace: A Novel Dataset for Mobile Face Tracking in the Wild," *IEEE International Conference on Automatic Face and Gesture Recognition*, pp. 1–8, 2019.

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," *European Conference on Computer Vision, Lecture Notes in Computer Science*, vol. 9905, pp. 21–37, 2016.

[24] Z. He, J. Zhang, M. Kan, S. Shan, and X. Chen, "Robust FEC-CNN: A High Accuracy Facial Landmark Detection System," *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 2044–2050, 2017.

[25] E. O. Ewunonu and C. I. P. Anibeze, "Anthropometric Study of the Facial Morphology in a South-Eastern Nigerian Population," *Human Biology Review*, vol. 2, no. 4, pp. 314–323, 2013.

[26] S. Wu, M. Kan, Z. He, S. Shan, and X. Chen, "Funnel-Structured Cascade for Multi-View Face Detection with Alignment-Awareness," *Neurocomputing*, vol. 221, pp. 138–145, 2017.

[27] M. Kim, A. K. Jain, and X. Liu, "AdaFace: Quality Adaptive Margin for Face Recognition," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18750–18759, 2022.

[28] T. Qin, P. Li, and S. Shen, "VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018.

[29] S. Chen, H. Guo, S. Zhu, F. Zhang, Z. Huang, J. Feng, and B. Kang, "Video Depth Anything: Consistent Depth Estimation for Super-Long Videos," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22831–22840, 2025.

doi:10.6342/NTU202600774