ILLIXR-A: AN OPEN SOURCE EXTENDED REALITY SYSTEM FOR ANDROID PLATFORMS

BY

MADHUPARNA BHOWMIK

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2023

Urbana, Illinois

Adviser:

Professor Sarita V. Adve

ABSTRACT

There has been significant progress in virtual, augmented, and mixed reality (VR, AR, and MR) devices, collectively referred to as extended reality (XR). However, until recently, this domain was dominated by proprietary software, which slowed down research. This motivated the development of an open-source end-to-end XR system called ILLIXR (Illinois Extended Reality testbed). It was developed to democratize XR systems research, development, and benchmarking. ILLIXR has been used to provide quantitative characterizations of performance, power, and quality-of-experience metrics for current XR systems and to develop solutions to address the gap between current and desirable XR systems.

However, ILLIXR currently only supports the Linux platform, which limits its use in common consumer-grade devices that do not support Linux. The operating system used in most consumer devices, such as smartphones, smart TVs, tablets, and head-mounted displays (HMD), is Android. Android's open-source nature allowed it to be highly customizable and adaptable for different purposes, contributing to its widespread usage. Additionally, there is little detailed analysis of XR runtimes on the Android platform.

Hence, we present ILLIXR-A, a modular, open-source, and end-to-end XR system for the Android platform. ILLIXR-A leverages Monado to provide support for OpenXR-compatible applications, which is an emerging open interface between an XR application and runtime. It will enable researchers to utilize the full potential of ILLIXR on a wide variety of Android devices.

ILLIXR-A provides full head-tracking capabilities using Visual-Inertial Odometry (VIO) by utilizing the camera and built-in sensors in an Android device. It also includes an IMU Integrator and a pose predictor, which predicts the user's pose at a faster rate than VIO to compensate for the high latency of VIO. ILLIXR-A also performs Asynchronous Reprojection (Timewarp). Using ILLIXR-A, we evaluated an end-to-end XR system on an Android smartphone, and we present its performance characterization in this thesis. The experimental results highlight the need for efficient algorithms for low-power devices to meet the tight deadlines for a smooth XR experience. We also show how the quality of experience is affected by the complexity of an algorithm and provide future directions for research.

ACKNOWLEDGMENTS

I would like to start with thanking everyone in the ILLIXR group for constant support and help during the development phase of this project. This project couldn't have been possible without Professor Sarita Adve's help and guidance. I really appreciate her constant feedback and support throughout the process including both development and writing of this thesis. Thank you for being my mentor. I would also like to thank Huzaifa, who played a pivotal role in helping me start off the project by breaking the project into chunks and guiding me through the initial phase of development.

One major roadblock I hit during development was a Compiler related issue that I couldn't nail down. I really appreciate Jay Cha from ARM for coordinating with his team to help me with the support I needed. Thank you Paolo Jovon for looking into the issue and explaining it to me. Finally, I couldn't have solved it without Sam's help. I am very thankful for the help I received from them that helped me continue with the project. After ILLIXR was ported to Android, integrating it with Monado was the next major step. However, it was difficult to convert the current integration of Monado with ILLIXR to Android. Jefferey jumped in to help me and came up with a working solution in a week's time. I really appreciate his expertise in this matter and unblocking me at the right time.

Next I would like to thank Yihan for helping me figure out the tracking issue with OpenVINS. Thank you for putting in the time to resolve this issue. I would also like to thank Henry for the same. Qinjun has been my go-to person for any doubts I had about ILLIXR. Thank you for always being there to clarify my doubts.

I would also like to thank my family: Maa, Baba, Ritu and Didi for always being there for me by providing emotional support and helping me through all other facets of life. Finally I would like to thank Sahil, my partner, for always being there and encouraging me to work on this project. I also cannot thank him enough for all the help with writing this thesis.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 MOTIVATION

Extended Reality (XR), which encompasses augmented reality (AR), virtual reality (VR), and mixed reality (MR), has experienced significant advancements in recent years. However, the XR ecosystem has been predominantly dominated by proprietary devices and software, resulting in a fragmented landscape that impedes the progress of research and development in this field. This motivated the development of ILLIXR (Illinois Extended Reality testbed) [1] [2] [3], an open-source end-to-end XR system built to facilitate and democratize XR research.

With its modular and extensible design, ILLIXR allows researchers to explore the effects of different XR algorithms on the XR system by mixing and matching them. Furthermore, ILLIXR offers an OpenXR interface, utilizing Monado [4], enabling researchers to run OpenXR-compliant applications on the ILLIXR platform. The goal of ILLIXR is to bridge the existing gap between current XR capabilities (both hardware and software) and the requirements for an ideal XR experience by facilitating research in this domain.

ILLIXR has been successfully adopted by researchers in both academia and industry. However, one limitation of ILLIXR is its exclusive support for the Linux platform, which is not commonly used in consumer-grade devices. Consequently, ILLIXR has primarily been evaluated on Linux-based systems such as desktop computers and NVIDIA Jetson devices.

The evaluation of ILLIXR on Linux-based desktop systems, utilizing state-of-the-art algorithms and popular head-mounted devices like the Valve Index [5], has established a benchmark for XR systems. This evaluation not only demonstrates the potential achievable with current systems but also highlights the need for improved algorithms and hardware. Additionally, ILLIXR empowers researchers to collect metrics related to performance, quality of experience (QoE), and power usage, facilitating a deeper understanding of XR systems and driving future advancements.

Motivated by the goal of assessing ILLIXR's performance on various platforms and devices and examining the ability of AR/VR applications to deliver smooth user experiences, we embarked on the development of ILLIXR-A. Given Android's status as one of the most popular operating systems, widely used in a diverse range of consumer devices, we made the decision to port ILLIXR to the Android platform.

Therefore, in this work we present our experience porting ILLIXR to the Android platform and present ILLIXR-A, an Android port of ILLIXR that will enable researchers to use ILLIXR on a wide variety of Android devices. We will refer to ILLIXR as ILLIXR-L, representing the Linux

version, and ILLIXR-A, representing the Android version. ILLIXR-A leverages Monado [4], a software system to manage OpenXR state, to connect to OpenXR compatible Android applications similar to ILLIXR-L.

For testing ILLIXR-A, we primarily utilized Android smartphones. It was interesting to run ILLIXR-A on a smartphone because they are widely-adopted, general-purpose devices built for running lightweight apps, making phone calls, and browsing the internet. The Android port also enables researchers to use the full ability of the ILLIXR-L system on Android which includes testing different XR components, performance analysis and measuring QoE metrics. Therefore, ILLIXR-A will help expand the scope of ILLIXR-L and provide a full-fledged open-source XR system for the Android platform.

1.2 CONTRIBUTIONS

The following are the major contributions of this work:

1. **Expanded Compatibility:** ILLIXR-L was limited to Linux-based systems, but this work successfully extended its capabilities to the Android platform. This expansion greatly increases the number of supported devices, broadening the reach of ILLIXR.

2. **Modular and Extensible Design:** ILLIXR-A has a modular and extensible design similar to ILLIXR-L. This allows developers to experiment with different components of XR, such as testing new VIO algorithms or processing various Android sensor inputs (accelerometer, gyroscope, magnetometer), camera inputs, and more.

3. **OpenXR Compatibility:** ILLIXR-A leverages Monado to provide support for the OpenXR API. This means that OpenXR-compliant Android applications can now be used with ILLIXR-A. To showcase this compatibility, a sample OpenXR Android application has been developed using the Unity Game Engine.

4. **Enhanced Tracking Capability:** Although Monado provided support for pose tracking, it was limited to the use of rotation information only. With ILLIXR-A we made extensive changes to provide comprehensive tracking capabilities that can support both rotation and translation.

5. **Performance Characterization:** ILLIXR-A enables the performance characterization of a complete XR system on an Android device. Through testing and evaluation, the results obtained highlight the importance of efficient and robust XR algorithms in delivering a smooth and immersive XR experience on Android.

CHAPTER 2: BACKGROUND

2.1 EXTENDED REALITY (XR)

2.1.1 Introduction

Extended Reality (XR) is an umbrella term for AR (Augmented Reality), VR (Virtual Reality) and MR (Mixed Reality). AR involves interaction with the physical world enhanced by virtual objects. For example adding a virtual teapot on a physical table. VR is a fully immersive experience in which the user wears a headset and interacts with a virtual world. Popular games like Beat Saber, The Elder Scrolls V: Skyrim are examples of Virtual Reality. Mixed Reality is a mixture of AR and VR where physical and virtual objects interact with each other, Microsoft Holoportation [6] is an example of Mixed Reality.

There are various computer vision, robotic and rendering algorithms involved in building XR applications. An AR application creates a 3D Map of the environment using camera and depth sensor inputs. Edges, features and objects are detected from the images using computer vision algorithms. Once this is done the user's movements and orientation are tracked so that the virtual objects can be rendered correctly as the user moves. A VR application involves creating the assets for the virtual scene first. Frames are created from these assets that are displayed to the user and updated according to the user's movements. Adding audio along with the visuals enhances the experience. We can broadly classify XR algorithms into the following three categories:

1. Perception Pipeline

   The perception pipeline converts raw sensor information into usable data that can be used to understand the user's position and display the virtual environment accordingly. Examples of sensor information used in the perception pipeline include IMU (Inertial Measurement Unit) data and camera images. The perception pipeline of an XR system involves several algorithms. Here are some of the key algorithms used in the perception pipeline of an XR system.

   - SLAM (Simultaneous Localization And Mapping)
     SLAM is an algorithm that is used to create a map of an unknown environment at the same time determining where the user is in the map. It is used in robotics and XR to navigate in an unknown environment. Usually SLAM algorithms consist of a back-end and front-end. The front-end detects the key points from an image and converts

them to 3D landmarks whereas the back-end relates the new information with the previously stored information to create and update the map of the environment. The front-end also performs loop-closure, which is determining that the user has come back to a previously encountered position.

- VIO (Visual Inertial Odometry)
  VIO is used to estimate the user's pose (translation and rotation) based on information from Camera and IMU sensors. It tracks the movement of the user and estimates its translational and rotational motion. Like SLAM, VIO also tracks feature points in the image and combines that with the IMU information. However SLAM focuses on a global view whereas VIO focuses on a local setting and can be considered as a subset of SLAM.

- Hand Tracking
  Hand Tracking allows users to use hand gestures to control the inputs instead of using a controller. This provides a more natural and real-world like experience for users. Hand tracking involves tracking the movements of the different joints in the hands. Sensors like ultra-wide camera, depth sensors and infrared LEDs [7] are used for hand tracking.

- Eye Tracking
  Eye tracking involves taking images of the user's eyes and using it to find the gaze vector which indicates the direction the user is looking at. Knowing where the person is looking at can help the application provide a better and more realistic viewing experience. One example of this is Foveated rendering which is a technique that renders the center of the field of view at a higher resolution than the peripherals. Eye tracking can also be used as an input, for example selecting objects just by looking at them.

2. Visual Pipeline

   Visual pipeline incorporates the information about the user's position (from the Perception pipeline) with the application frame to generate the final display for the user. Following are some of the algorithms in this pipeline:

   - Asynchronous Reprojection

     It is used to reduce nausea or motion sickness in VR apps. In VR apps the scenes are generated as the user moves, but if a scene is not rendered on time then the user will have a degraded experience. To account for such delays in generating the next

frame in the virtual world, asynchronous reprojection can be used to approximate the next frame based on the user's current pose and previous frame. In ILLIXR-L, the Timewarp plugin performs asynchronous reprojection and we will be using these two terms interchangeably.

- Lens Distortion and Chromatic Aberration Correction

  It is used to minimize or eliminate color fringing in images caused by the dispersion of light through lenses. Lens distortion correction is used to correct the optical distortion that can occur when using wide-angle or zoom lenses. It aims to rectify the barrel or pincushion distortion, where straight lines appear curved or distorted in the image.

3. Audio

   The audio pipeline generates spatial audio in two phases which are audio encoding and playback. Audio encoding involves converting the audio signal into digital data and playback involves converting the digital data back into audio signal.

In our work, we only focus on the Perception and Visual pipelines, specifically VIO and Asynchronous Reprojection.

2.1.2 QoE Metrics for XR

Quality of experience (QoE) is a subjective measure of how satisfied the user is with the XR application. QoE metrics are helpful because they can guide the developers on which specific areas to improve the application on. Some factors that QoE encompasses are how well the application responds to user's movements and inputs, the quality of the visual elements, lag in seeing the effects of the inputs, motion-sickness etc. Since it is a subjective metric, a user study is the best way to measure QoE for an XR application. However a user study is not always feasible thus certain metrics like MTP and SSIM also exist to estimate the QoE. This is still an active area of research in the XR community and the metrics mentioned do not capture many aspects of the final video that is displayed to the user. Such as temporal coherence and smoothness or jitters.

1. MTP (Motion to Photon Latency)
   MTP is a measure of delay between user's movement in the physical world and the effects of it in the virtual world. MTP is affected by several factors such as latency between the physical event and the sensor reporting the event, time taken by an algorithm to process

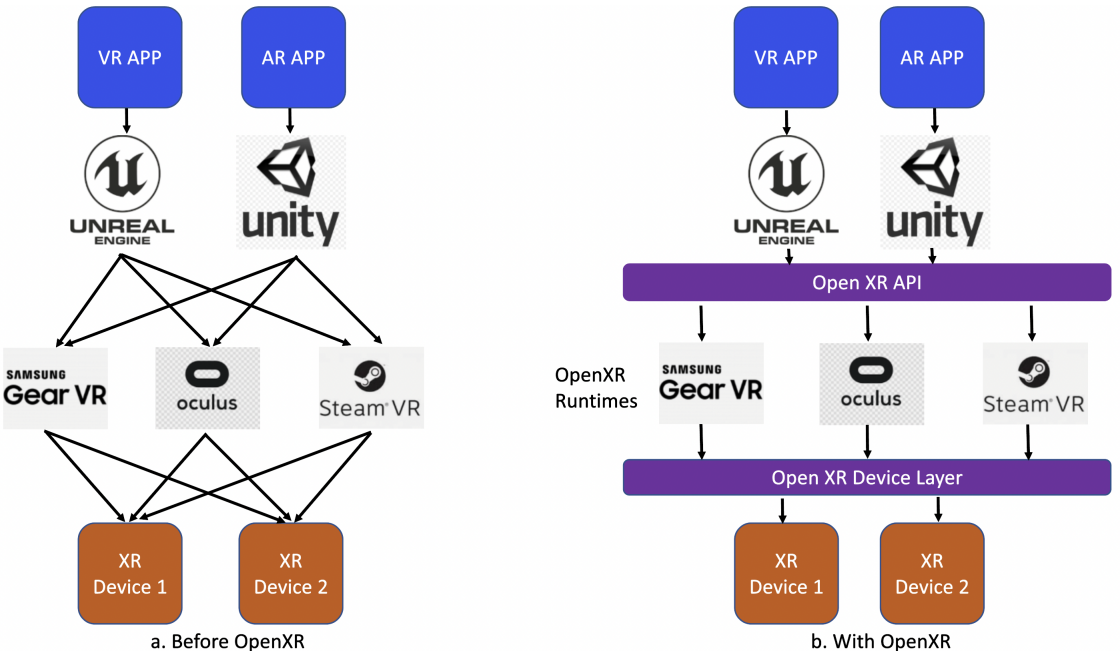

Figure 2.1: OpenXR

the sensor input and design the output response, and the time taken for the output to be displayed to the user. A Lower MTP indicates that the system is highly responsive to the user's movements.

2. SSIM (Structural Similarity Index Measure)
   SSIM measures the quality of the visuals in a video sequence by comparing two consecutive frames. It looks for structural similarity between the two frames to look for any visual artifacts and distortions. To increase the SSIM score we can use post-processing techniques such as anti-aliasing, motion blur and using a high resolution image.

2.1.3 OpenXR Specification

OpenXR is a standard for building XR applications that can run on multiple platforms. Before OpenXR, this space was highly fragmented. Vendors had proprietary software, and applications needed to be modified to work on different platforms. This caused an increase in development time and required developers to provide support for new devices. This is depicted in Figure 2.1.a. Thus, OpenXR aimed to provide a single API that all the applications can use irrespective of target platform. In this thesis, we mainly worked with the OpenXR API, Runtime and the Loader. 2.1.b shows the different components of OpenXR specifications.

1. OpenXR API

OpenXR API is a specification that defines a set of functions and data structures to create VR/AR applications that are compatible with a range of hardware devices and platforms. The API is the interface between the application and the runtime which implements this API.

2. OpenXR Runtime
   OpenXR Runtime provides an abstraction layer between the applications and the underlying hardware and Operating System. The runtime manages the hardware resources and provides the application with movement tracking, pose, eye tracking, and other inputs. This abstraction allows developers to develop applications that can run on multiple platforms. For example, SteamVR, HTC, and Meta are OpenXR runtimes.

3. OpenXR Device Layer
   The OpenXR Device Layer provides a standardized interface to interact with the input and output devices such as the headset, eye tracker, gesture recognition system etc. This allows applications to run on a variety of devices without worrying about the specifics of each device. It also makes OpenXR apps forward-compatible as any future device following OpenXR standards can also work with the application.

4. OpenXR Loader
   Multiple OpenXR runtimes could be installed in a device at the same time but only one of them can be active at a time. The OpenXR Loader is used to discover the active OpenXR runtime among the installed ones and use it. It can be packaged with the application or installed system-wide. The loader can also interact with OpenXR API Layers as shown in Figure 2.2. OpenXR API layers are optional components that can be used to validate, intercept any OpenXR calls, and debug the applications. The loader can inject any number of optional OpenXR API Layers between the application and the runtime.

5. OpenXR Input Mapping
   OpenXR allows one application to run on multiple different platforms; however, different devices may have different input types available. Thus it is important to map a logical action in the application with the physical inputs available in a device. This process is called Input Mapping in OpenXR. The application can map different actions with different device inputs, thus providing flexibility and portability to different hardware.

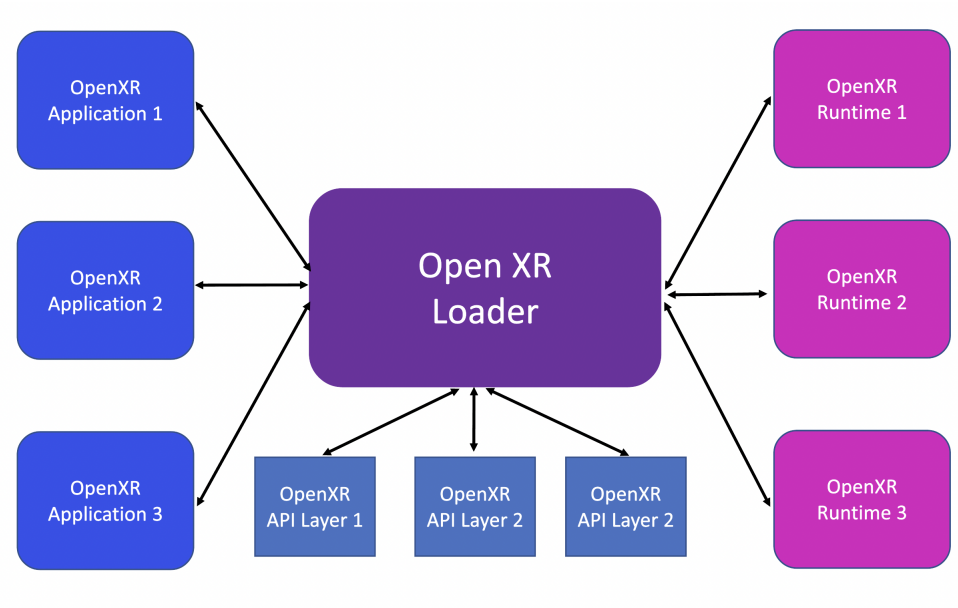

Figure 2.2: OpenXR Loader

2.2 ILLIXR

ILLIXR [1] is an open-source end-to-end XR system that promotes research in the XR domain (Figure 2.3). It enables researchers to study different parts of an XR system and carry out experiments to study performance, power and QoE metrics. We refer to it as ILLIXR-L in this thesis because it only supports the Linux Operating System. ILLIXR-L enables developers to build independent components of an XR system as plugins. For example, a plugin could be responsible for getting camera images and publishing them for other plugins to use. Plugins are distributed as shared object files and loaded dynamically by the ILLIXR-L runtime. This section offers an overview of the ILLIXR implementation. For more detailed information, we recommend referring to [3].

2.2.1 Core Implementation Concepts

The ILLIXR implementation consists of two key concepts to orchestrate the execution of the various plugins.

1. Phonebook
   In ILLIXR-L, plugins are loaded dynamically and they can provide functions called services that can be used by other plugins. So, a service provided by a plugin can only be used when it is loaded. Upon starting a plugin, it will register all its services in the Phonebook. Any plugin that wants to use a service provided by another plugin needs to lookup if

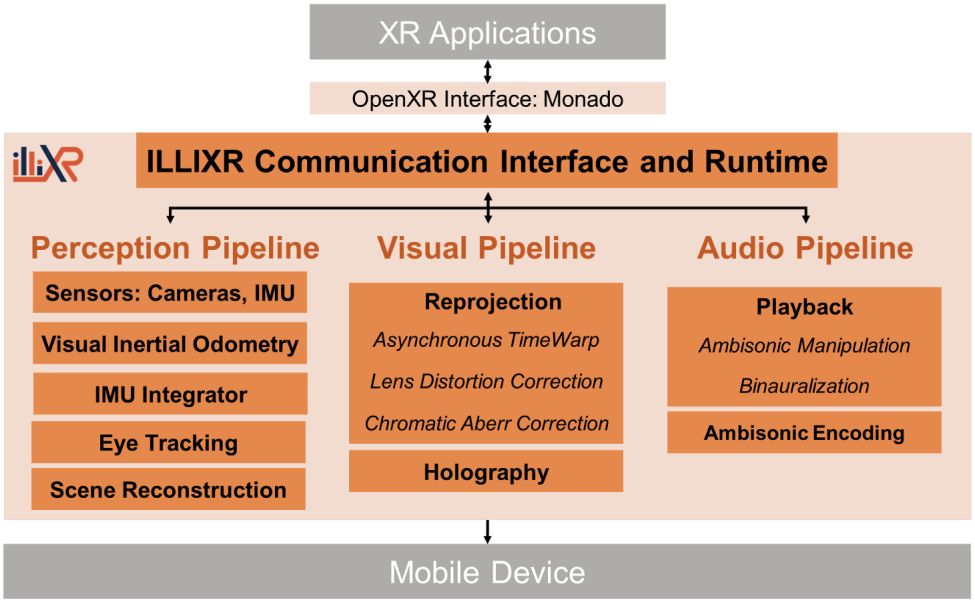

Figure 2.3: The ILLIXR (Illinois Extended Reality) testbed

the service is registered in the Phonebook. So, Phonebook essentially provides a look-up service for plugins.

2. Switchboard
   Switchboard is a manager for event streams called topics. Different plugins can subscribe to read from a topic or publish to a topic. Reading from a topic can be synchronous or asynchronous. To synchronously read from a topic, a plugin can schedule a callback which will be called each time an event is published on that topic. For asynchronous reads, a plugin can simply read the latest event from the topic.

2.2.2 Perception Pipeline

ILLIXR-L integrates different stereo and depth-sensing cameras like the ZED, Realsense, and Asus for obtaining the images and IMU information. ILLIXR-L has integrated two open-source systems, KimeraVIO [8] and OpenVINS [9], for VIO.

2.2.3 Visual Pipeline

ILLIXR-L's Timewarp plugin performs Asynchronous Reprojection. ILLIXR-L also has a plugin performing Computational Holography.

2.2.4 Audio Pipeline

ILLIXR-L uses libspatialaudio, an open-source library for spatial audio processing and playback.

2.2.5 ILLIXR-L Plugins

Table 2.1 contains a list of some of the ILLIXR-L plugins that are relevant to this work. A more comprehensive list of all plugins supported by ILLIXR can be found here [3].

2.2.6 ILLIXR-L Applications

ILLIXR-L supports a sample application called GlDemo which renders a static scene. This application can be directly used in ILLIXR as a plugin. Additionally, ILLIXR-L also supports OpenXR applications using Monado. Some OpenXR applications that have been used with ILLIXR are Platformer [10], Materials [11] and Sponza [12].

2.3 EXTENDED REALITY ON ANDROID

2.3.1 Google's AR Core

Google's ARCore [13] is a popular SDK for developing AR applications on Android. ARCore can be used to build applications that can place virtual objects on physical surfaces. ARCore can detect surfaces like tables in the real world and provides an interface to interact with them. It uses camera and IMU to detect the rotation and translation information of the device. ARCore is designed to adapt according to a device's capabilities and thus works on both high and mid-range Android devices. Since ARCore's release in 2018, it has been used in many popular mobile apps such as Pokemon Go, IKEA App, Snapchat etc. However, ARCore is not open-source so testing out new XR algorithms isn't feasible. Therefore we need a system like ILLIXR-A which provides the flexibility to switch algorithms.

2.3.2 Smartphone VR Headsets

Smartphone VR headsets are inexpensive devices that use a smartphone for display and processing. These devices usually consist of a pair of lenses and a place for the smartphone. Google Cardboard is a popular example of Smartphone VR Headset. These devices are cheaper because

the headset does not require any additional sensor like camera or IMU as the smartphone already has it.

Plugin	Function
Timewarp	Timewarp performs Asynchronous Reprojection.
Rk4 and GTSAM IMU Integrators	IMU integrator is used to provide pose at a higher frequency based on the IMU data and last pose returned by VIO algorithms.
Camera and IMU Plugins	Camera and IMU plugins provide the Camera frames and IMU inputs from the accelerometer and gyroscope sensors. ILLIXR-L supports Realsense, ZED, and Asus cameras.
OpenVINS and Kimera VIO	These are plugins implementing VIO algorithms that return the user's pose based on the Camera and IMU inputs.
GlDemo	It is a sample application that renders a static scene.

Table 2.1: Description of relevant ILLIXR-L plugins

2.3.3 Qualcomm AR Spaces

Snapdragon Spaces [14] now provide support for using a ThinkReality A3 HMD [15] with an Android phone (Motorola edge+). It is different from a Smartphone VR because it uses the smartphone for processing only and not for display. Thus it is very similar to connecting a standard HMD with a PC.

2.4 OPENXR ON ANDROID

Monado is an OpenXR state management software that supports the Android platform. Monado also provides an OpenXR Runtime broker app which can be used in an Android phone to select between OpenXR runtimes. It also provides the standard Teapot application from the Android NDK with OpenXR support. Monado's support for Android platform is still limited, as the pose returned by Monado only includes the gyroscope information and thus can only provide the rotation information of the device.

Popular game engines like Unity, Unreal, and Godot also provide OpenXR support for Android. However, it's worth mentioning that these game engines do not incorporate the necessary OpenXR loaders to build games specifically for Android smartphones. Nonetheless, they do offer support for other OpenXR loaders that cater to Android-based devices such as Oculus..

CHAPTER 3: IMPLEMENTATION AND METHODOLOGY

This chapter highlights the design and implementation details of ILLIXR-A. We also present the challenges faced during the development of ILLIXR-A and how we solved them. The code for ILLIXR-A will be available in the ILLIXR Github repository accessible from https://illixr.org.

3.1 CHALLENGES OF ANDROID DEVELOPMENT

Some of the main challenges of porting ILLIXR-L from Linux to Android were:

1. Build System: In ILLIXR-L, all dependencies are installed using a Python script, and the script internally calls cmake to build the plugins. For Android development, Gradle is the popular build tool that can use cmake to build C++ code. Thus porting ILLIXR-L to Android required configuring this new build system (Gradle + cmake) from scratch.

2. Programming Language: Java/Kotlin are the popular languages to develop Android Apps. Thus there are more resources and support for Android Development in Java/Kotlin than in C/C++. Also, many helper functions to manage UI elements are only present in Java/Kotlin. But since ILLIXR-L was written in C++, and it was not feasible to convert the entire ILLIXR-L code base to Java, we could not use any of these resources.

3. Dependencies: Android NDK (Native Development Kit) is used to build C/C++ code for Android. NDK supports only a few libraries and any other dependencies usually need to be built from source. Some of the external dependencies that required custom build were opencv and boost.

3.2 DIFFERENCES BETWEEN ILLIXR-L AND ILLIXR-A

3.2.1 Graphics Library Support

The current version of ILLIXR-L uses OpenGL (Open Graphics Library) for graphics. However, Android only supports a flavour of OpenGL called OpenGL ES (OpenGL for Embedded System). OpenGL ES is a subset of OpenGL that cuts down on the large OpenGL API and keeps only the bare minimum so that it can work on simpler, low-power devices. Thus OpenGL ES code can run on an OpenGL compatible device like a Desktop but an OpenGL code written for Desktop will not necessarily run on an embedded device like a smartphone. So, the first step was to port all the OpenGL code in ILLIXR-L to OpenGL ES. However, ILLIXR-L is now moving to Vulkan, a low-overhead, cross-platform API for graphics which is also supported on the

Android platform.

3.2.2 Windowing System

ILLIXR-L uses X11 (X Window System) on Linux which manages Graphical User Interfaces (GUIs). X11 consists of an X Server and X Client. ILLIXR-L uses GLX which is an OpenGL window-system API for X11. It is possible to use X11 with Android, but this requires using third-party applications. Thus the better choice was to go with EGL (developed by the Khronos Group) which is a newer windowing system designed for embedded devices. EGL is also a platform-independent OpenGL/GLES window-system API. It is lightweight compared to X11 and is included in the Android Native Development Kit. Therefore, porting the code from X11 to EGL was necessary to open and manage windows on Android.

3.2.3 Compiler

ILLIXR-L provides an extensible interface that lets users write individual plugins that are compiled separately, have its own dependencies, and can be exchanged with another plugin of similar type, without making changes to the rest of the system. These plugins are shared object files that are linked dynamically. Thus if a dynamically loaded plugin needs to use a service from another plugin which is also dynamically loaded, the service cannot be statically constructed because the plugin is dynamically loaded. Thus, ILLIXR-L provides the phonebook service which allows plugins to register and lookup a service dynamically. For example, when plugin A is loaded it can register a service S using register_impl() which can then be looked up by another plugin (also dynamically loaded) using lookup_impl(). The typeId of the class/struct is used as a key in a hash map to identify the service in lookup_impl().

However, since the classes/structs corresponding to the service are defined in two different compilation units, dynamic_pointer_cast() between those two objects, may not necessarily work. The C++ standard does not mandate that the typeids of identically-defined structs should be identical. It is compiler dependent and the compiler on Ubuntu assigns the same typeids to the same struct defined in two different plugins. Thus dynamic_pointer_cast() between these structs work fine in the Desktop version of ILLIXR-L. However, the compiler implementation of typeid() in the Android NDK assigns different typeids to struct definitions that are identical but defined in different plugins.

This resulted in runtime issues in the Android port, where plugins were unable to look up existing services registered by other plugins. The issue was fixed by using typenames as the key

to the hash table and using a reinterpret_pointer_cast() to force the cast. This was the one of the issues encountered during Android port caused due to compiler differences.

3.2.4 Synchronization between Timewarp and GlDemo

GlDemo is an application in ILLIXR that renders a static scene, while Timewarp performs asynchronous reprojection on the frames generated by GlDemo.

In the OpenGL framework, a GL context can only be active in one thread at a time. This means that sharing a GL context requires explicit synchronization between threads. To avoid the need for synchronization, ILLIXR-L's Timewarp and GlDemo use two different GL contexts, each linked to two separate GL windows. The Timewarp's window is used for the final display, while GlDemo's window is a dummy window with a size of 1x1.

However, in ILLIXR-A, we encountered a limitation where we couldn't create separate windows for Timewarp and GlDemo due to the fact that an Android activity can only manage one window. Consequently, GlDemo and Timewarp share the same window and the same GL context in ILLIXR-A. This means that synchronization is required between GlDemo and Timewarp because they share the same GL context.

Since Timewarp and GlDemo are loaded as dynamic plugins, we are unable to define a global lock that is accessible to both plugins. Instead, we have implemented a separate plugin called 'common_lock' that provides methods for acquiring and releasing locks. Timewarp and GlDemo can call these functions from the 'common_lock' plugin to acquire or release the necessary locks.

3.3 OPENXR INTERFACE USING MONADO

ILLIXR-L uses Monado [4] to support OpenXR applications. We began this work in 2022, and based our implementation on ILLIXR-L's version 3.1.2 [16]. This version did not utilize Monado's compositor. It only relied on Monado only for managing the OpenXR application state. It did not use Monado's compositor and did not use Monado to display the application frames. In this version, ILLIXR-L would receive application frames from Monado and display them using its own window within Timewarp. However, Monado underwent significant changes in the past year, introducing important enhancements to its compositor and display management specifically for Android support.

To take advantage of these improvements in Monado's compositor, we made the decision to modify the way ILLIXR-L interacts with Monado. Rather than solely relying on Monado for

OpenXR state management, we decided to incorporate its compositor and display functionalities while performing Asynchronous Reprojection (Timewarp) in ILLIXR-L. This change introduced the need for synchronization between Monado and ILLIXR-L to signal each other when frame processing is completed. [1]

As a result, during the implementation of ILLIXR-A, we opted to reuse the common_lock plugin mentioned in section 3.2.4 to establish semaphores for synchronization between Monado's compositor and ILLIXR-A's Timewarp.

3.4 POSE ESTIMATION

ILLIXR-A provides the ability to estimate pose using various methods like ILLIXR-L. In the most common scenario it uses VIO with Camera images and IMU sensor inputs. It also supports using datasets for providing inputs for VIO.

3.4.1 Sensors Needed for Pose Estimation

To enable Visual-Inertial Odometry (VIO), we required inputs from both an IMU sensor and a camera. To achieve this, we introduced new plugins to the system that are capable of extracting data from Android's Camera2 [17] and Sensor APIs [18].

The Samsung Galaxy S22 Plus is equipped with three back cameras. However, processing images captured by multiple cameras simultaneously would impose a significant computational burden. Therefore, we made the decision to capture images solely from the ultra-wide 12 MP rear camera, which offers a Field of View (FoV) of 120 degrees. These images were captured at a rate of 20 frames per second (20 fps). We utilized these images specifically for Monocular pose prediction using OpenVINS. In scenarios where we wanted to test the stereo mode, we utilized the same camera image twice.

For the IMU readings, we utilized the accelerometer and gyroscope sensors. The respective callbacks were scheduled at a rate of 200 frames per second (200 FPS). As a result, IMU readings were written to the "imu_cam" topic 200 times per second, and the VIO plugin subscribed to this topic to access the data. Additionally, camera images were added to the data at a rate of 20 times per second.

In summary, the VIO system obtained data from the Samsung Galaxy S22 Plus's ultra-wide rear camera at 20 fps, while the IMU readings from the accelerometer and gyroscope were acquired at a higher rate of 200 fps.

[1] ILLIXR-L's new version is planned to be released soon with a more efficient interface between Monado and ILLIXR-L

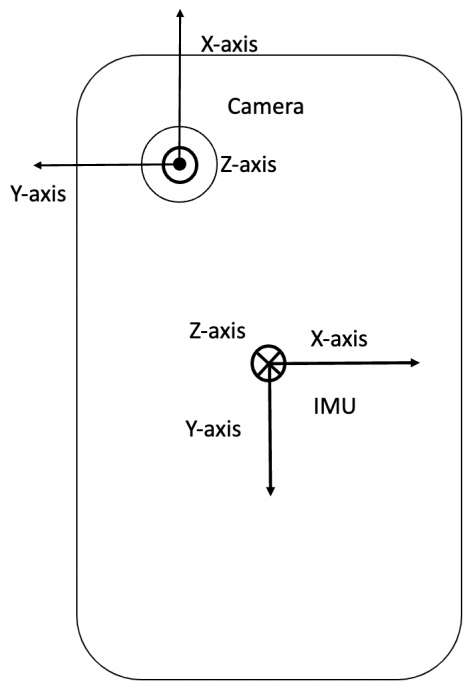

Figure 3.1: Android Camera-IMU Axes

3.4.2 Calibration of Sensors

VIO relies on having accurate intrinsic parameters of the camera and the camera-to-IMU transformation matrix. The camera intrinsics encompass essential parameters such as the focal lengths (fx, fy), principal points (cx, cy), radial distortion (k1, k2), and tangential distortion (p1, p2).

The Camera-IMU transformation matrix describes the spatial relationship, including rotation and translation, between the camera and the IMU on the Android smartphone. Figure 3.1 provides a visual representation of the camera and IMU coordinate systems.

For calibration purposes, we employed standard methods. To obtain the intrinsic parameters of the camera, we utilized Kalibr [19], a widely used open-source tool for camera and IMU calibration. This involved recording several minutes of camera feed with the presence of April Tags or Checkerboard patterns. The captured frames were then converted into a ros [20] bag file, which was subsequently utilized by Kalibr to calculate the camera's intrinsic values.

However, calculating the camera-to-IMU transformation matrix with Kalibr requires the presence of white noise and random walk characteristics of the gyroscope and accelerometer. To obtain these values, we utilized an open-source tool called allan_variance_ros [21]. This method necessitated measuring the IMU readings of the device while at rest for a minimum of three hours. With the obtained noise and random walk values, we could accurately determine the

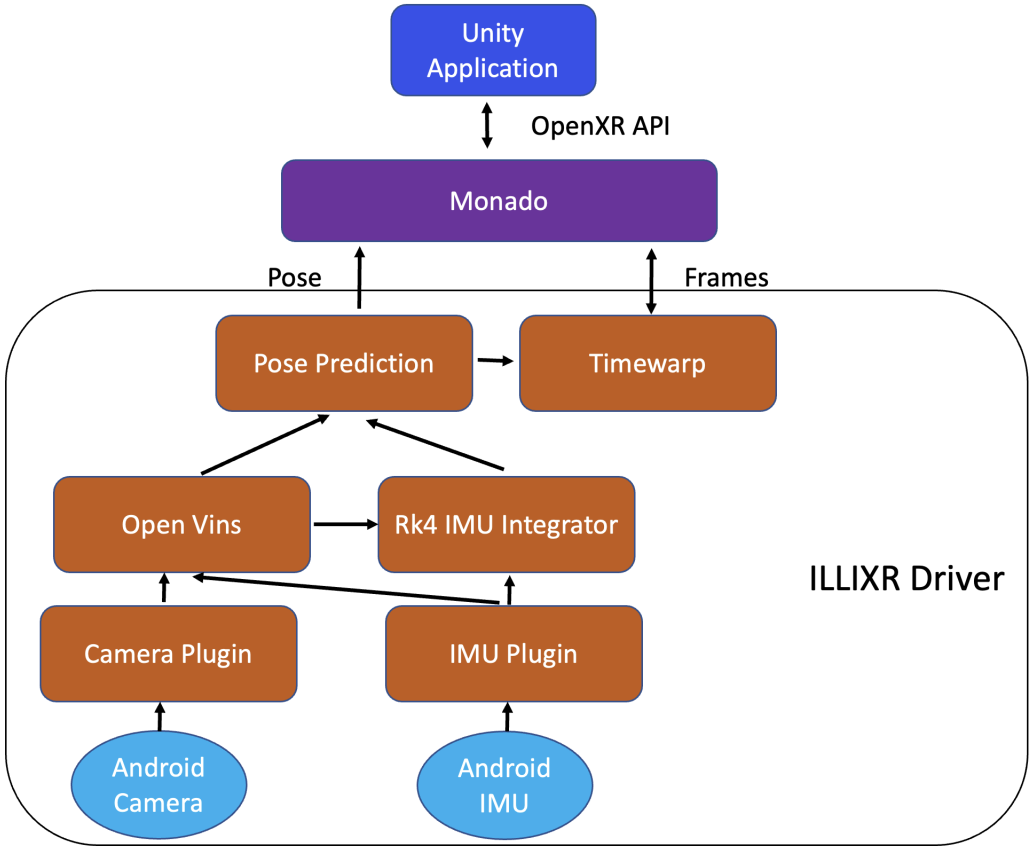

Figure 3.2: ILLIXR-A conceptual data-flow for a Unity application running with OpenVINS for head-tracking and Rk4 IMU Integrator.

camera-to-IMU transformation matrix using Kalibr. This process involved capturing images of April Tags or Checkerboard patterns while exciting all three axes of translation and the three rotational degrees of freedom.

3.4.3 Tuning VIO Parameters

Running VIO with camera and IMU sensor on an android phone is computationally expensive. Especially using stereo images can make a VIO algorithm take really long to calculate the pose. Thus tuning VIO parameters is important to get the poses on time. This includes reducing the total number of features tracked, using monocular mode instead of stereo and disabling online calibration.

3.4.4 Pose Estimation using the EuRoC Dataset

ILLIXR-A also provides support for using datasets for sensor and camera inputs. For experi-

ments, we used the EuRoC [22] dataset. The EuRoC dataset recorded on a Micro Aerial Vehicle (MAV) consisting of an IMU, Cameras, Leica MS560 and Vicon 6D consists of synchronized IMU measurements, stereo images and ground-truth. Using the dataset was particularly useful for testing trajectory accuracy because of known ground-truth trajectory.

3.5 EXAMPLE INSTANTIATION OF THE ILLIXR-A SYSTEM

Figure 3.2 illustrates the data flow in a particular instance of ILLIXR-A with Rk4 IMU integrator, OpenVINS and Unity application. It shows a conceptual producer-consumer data movement. The implementation uses switchboard for modularity and managing the plugins.

CHAPTER 4: EXPERIMENTS AND RESULTS

This chapter provides an initial performance and Quality of Experience (QoE) analysis of ILLIXR-A. We begin by examining the CPU and memory utilization of ILLIXR-A and investigate how these metrics vary with different Visual Inertial Odometry (VIO) parameters. Through these experiments, we gain insights into the areas of ILLIXR-A that require optimization and improvements. Notably, these experiments led to the discovery of a performance bug in Monado, which, upon resolution, resulted in a significant reduction in CPU usage. Finally, this chapter also presents the QoE metrics for ILLIXR-A which includes Motion to Photon Latency (MTP) and trajectory prediction accuracy.

4.1 EXPERIMENTAL SETUP

All the experiments conducted were carried out on a Samsung Galaxy S22 Plus smartphone. Table 4.1 presents the specifications of the phone.

Specification	Description
Display	6.6" flat FHD+ Dynamic AMOLED 2X, 60 Hz
Rear Camera 1	12MP Ultra Wide Camera, FOV: 120 degrees
Battery Capacity	4500 mAh
Memory and Storage	8 GB RAM (LPDDR5) with 128 GB internal storage
Application Processor	4nm 64-bit Octa-Core Processor (1 x 2.8GHz (Maximum Clock Speed) + 3 x 2.5GHz + 4 x 1.8GHz)

Table 4.1: Samsung S22 Plus Specifications. [23]

For the performance evaluation of the system, we selected three distinct applications, namely GlDemo, Teapot, and Unity.

Firstly, we chose GlDemo, an ILLIXR-A plugin, as our initial application. Unlike the other two, GlDemo does not require Monado and can be directly integrated with ILLIXR-A. By selecting GlDemo, we aimed to assess the individual contributions of ILLIXR-A's XR components without any involvement of Monado.

Next is the Teapot [24] application, a lightweight OpenXR application that has been used as a sample application with Monado. This particular application solely requires rotation information and was specifically chosen to showcase a minimalistic OpenXR application.

Figure 4.1: GlDemo : Native OpenGL application

Lastly, we incorporated Medieval Room, developed using the Unity Game Engine, which requires both translational and rotational information of the device. Our decision to include this application was motivated by the desire to exhibit the system's ability to support general applications created using a popular game engine adhering to the OpenXR standard. Additional details about each of the applications are provided below.

1. GlDemo : Native OpenGL Application

   This is a sample application in ILLIXR-A which can be loaded as a plugin. Thus, this application does not require using Monado for OpenXR interactions. GlDemo uses both the translation and rotation information to navigate in a virtual room. Figure 4.1 shows the GlDemo application.

2. Teapot : Native OpenXR application

   Teapot is a native OpenXR application from Monado. Teapot application renders a teapot on the scene based on the device's rotation. This application does not require the translation information of the user as it only rotates the teapot. Figure 4.2 shows the teapot application.

3. Medieval Room: OpenXR application made using Unity This application consists of a room in a medieval century setting [25]. Figure 4.3 shows an image of the room. This application allows the user to navigate inside the virtual room based on the user's physical

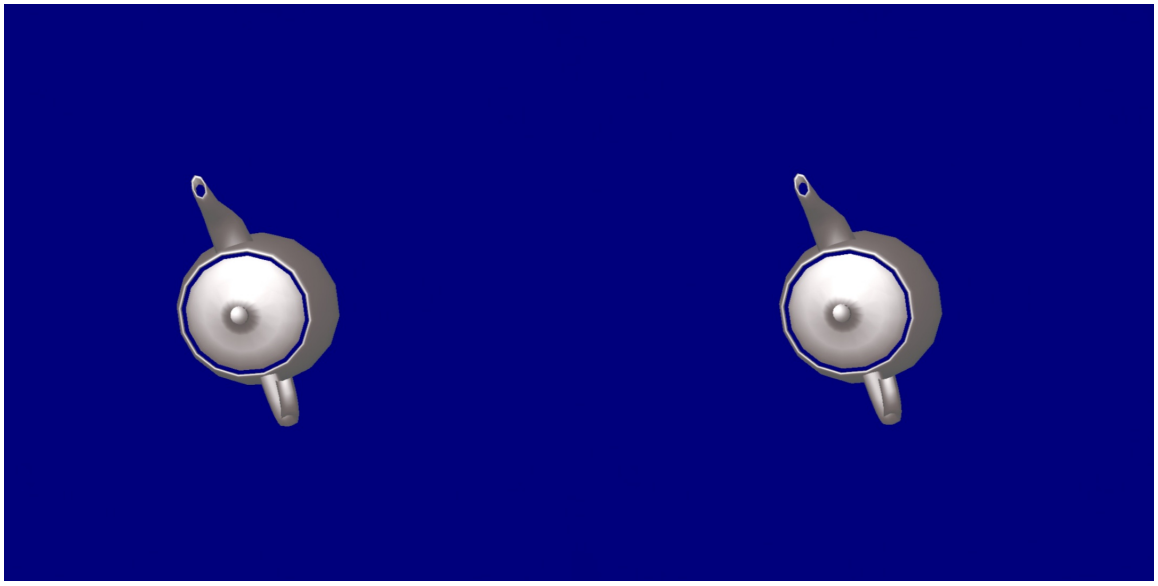

Figure 4.2: Teapot : Native OpenXR application

movement using both translation and rotation motion of the person. It is built using the Unity Game Engine with a custom OpenXR loader. The Medieval Room application is also referenced as Unity application.

To gather performance data from the Android applications, we utilized a combination of tools: the Android Studio profiler, Unity profiler, and SimplePerf. The Android Studio profiler allowed us to monitor the CPU and memory usage of the applications comprehensively. Meanwhile, the Unity profiler specifically provided us with the frame rate analysis of the Medieval Room Unity application. Additionally, we employed SimplePerf to gain insights into the specific contributions of individual threads to the overall CPU usage of the application. This was helpful in identifying CPU-intensive threads and optimizing their performance.

4.2 PERFORMANCE ANALYSIS

4.2.1 CPU Utilization of Different XR Components

This section focuses on comparing the CPU utilization of different XR components within ILLIXR-A. For the purpose of this analysis, we exclude the CPU usage of the application itself and instead focus solely on the contributions of the core XR components. The results presented herein pertain to the GlDemo and Medieval Room applications. As the Medieval Room and Teapot applications utilize the same XR components, we do not provide separate results for the Teapot application.

Figure 4.3: Medieval Room: OpenXR application made using the Unity Game Engine

Figure 4.4 shows the contribution of the different XR components (plugins) to the overall CPU usage when the system is run with the Medieval Room application. The figure also shows some of the Monado components like IPC (used for communication with the application) because Medieval Room runs on top of Monado. The Figure shows the relative percentage of CPU used by the different XR components. The total CPU usage is about 10 percent. From the Figure we can see that VIO takes the maximum percentage of CPU. The next highest contributor is the Compositor[2], followed by Rk4 IMU Integrator, Timewarp, Monado IPC, Camera and IMU components.

We also performed the same experiment with GlDemo. Figure 4.5 shows the contribution of ILLIXR-A components with the GlDemo application. The total CPU usage of ILLIXR-A with GlDemo is about 15% whereas with Medieval Room it is about 21%. We can see a similar trend here as well where VIO has the maximum CPU usage. Thus, optimizing VIO can improve the performance the most.

4.2.2 Memory Usage of XR Components

Figure 4.6 provides a comparison of the memory utilization between ILLIXR-A with GlDemo and Medieval Room. It is important to note that in the case of Medieval Room, the graph only represents the memory used by ILLIXR-A itself and not the memory utilized by the application. Specifically, the Medieval Room application alone consumes approximately 800 MB of memory.

[2]The large contribution of the compositor in CPU utilization could be due to the synchronization mentioned in section 3.3

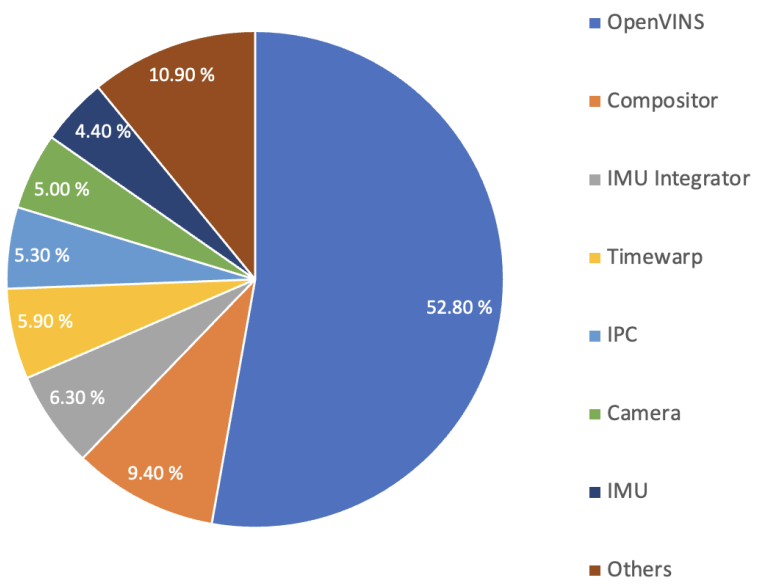

Figure 4.4: CPU usage of XR components in ILLIXR-A with Medieval Room application.

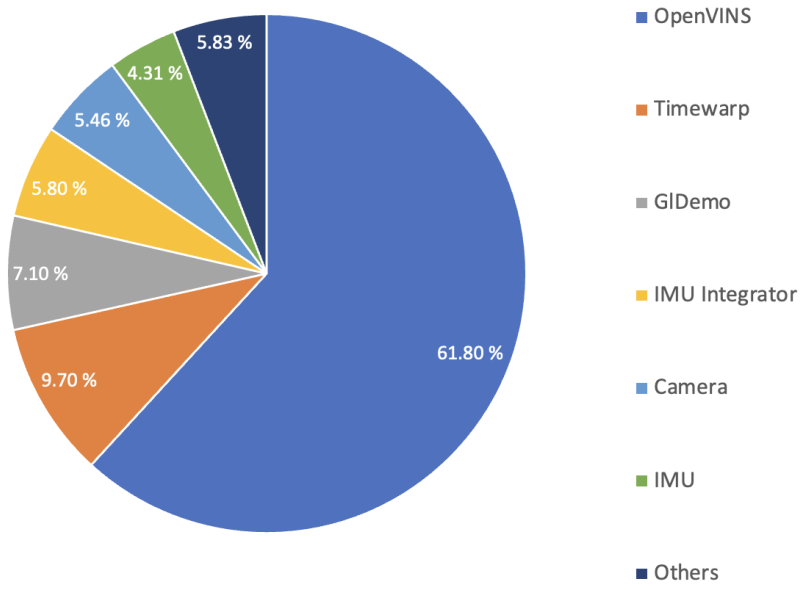

Figure 4.5: CPU usage of XR components in ILLIXR-A with GlDemo application.

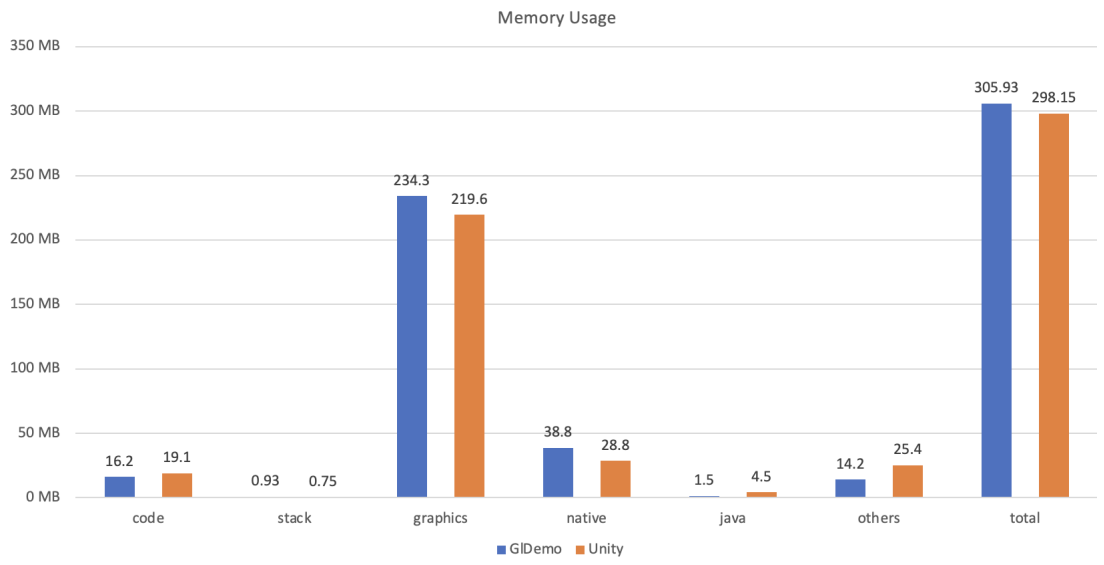

Figure 4.6: Comparison of memory used by ILLIXR-A with GlDemo and Medieval Room. Note that GlDemo includes application memory whereas Medieval room does not.

From the figure, we can observe that the memory occupied by code is higher for Medieval Room compared to GlDemo. This is primarily due to the fact that running the Unity application requires the usage of Monado, whereas GlDemo does not.

Additionally, the graphics memory usage is higher for GlDemo because the memory utilized by the GlDemo application itself is included in the graph, whereas it is not accounted for in the case of Medieval Room. In the figure, the "Native" category represents memory allocated by native C/C++ code, and it is higher for GlDemo due to the aforementioned reasons.

It is worth mentioning that ILLIXR-A is written purely in native code, with the exception of the wrapper for the Native Activity class [26], which involves Java code. On the other hand, Monado includes Java code for the Monado application. Consequently, the Java allocations are higher for Medieval Room.

4.2.3 CPU and Memory Usage of Applications

Figure 4.7 compares the CPU and Memory usage of the three applications. CPU utilization for the runtime with Medieval Room and Teapot applications are similar because both are using Monado with ILLIXR-A. Even though Teapot does not need position information, OpenVINS still calculates it thus there is no difference in CPU usage. However we can see that there is a huge difference between the CPU used by the Teapot and Medieval Room applications. This is because the Teapot application uses simple graphics whereas Medieval Room is more complex.

In the case of GlDemo CPU and Memory usage for the runtime and the application are shown

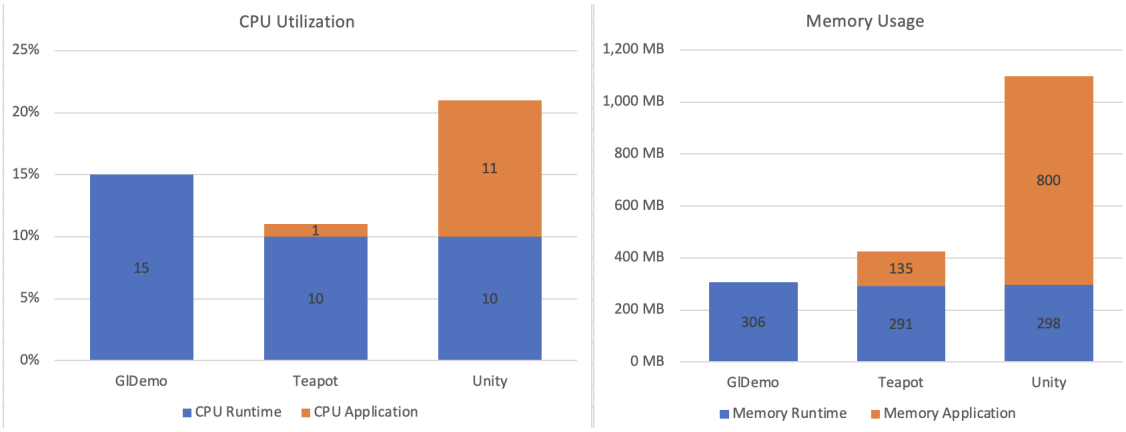

Figure 4.7: CPU utilization and Memory usage comparison of GlDemo, Teapot and Medieval Room applications. Note that application and runtime memory and CPU utilization is merged in the case of GlDemo.

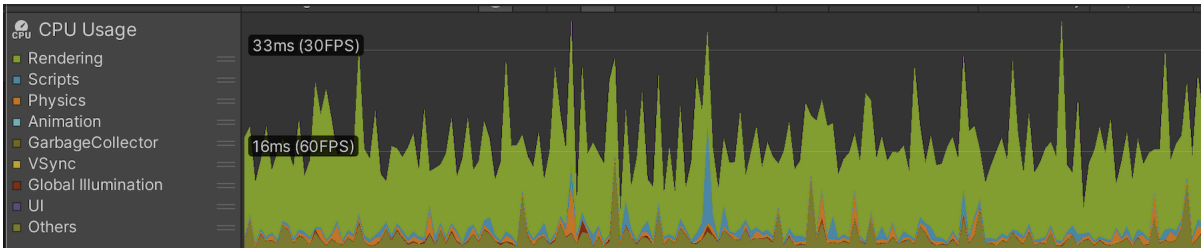

Figure 4.8: Medieval Room application rendering FPS on Samsung S22 smartphone

together because GlDemo runs in the same process as ILLIXR-A. GlDemo has higher overall CPU utilization than Teapot but less than Medieval Room. This is because Teapot application is relatively simpler than GlDemo and Medieval Room application is more complex compared to GlDemo.

Memory usage of the runtime for all the applications is similar. Medieval Room alone uses about 800 MB of memory which is significantly higher than the other two.

4.2.4 Application Rendering Frames Per Second (FPS)

Figure 4.8 shows the frame rate for the Medieval Room Application. The frame rate is about 60FPS mostly, however it can vary due to factors like heating of the smartphone, occasionally dropping the rendering rate to around 30 FPS. This shows the variability in application rendering. We can also see the contribution of components other than rendering, such as Physics, Garbage Collector etc. in the figure. We have only shown the FPS for Medieval Room because it is an Unity application and the Unity profiler shows this information. We cannot get this information for GlDemo and Teapot because we cannot profile these applications with the Unity

profiler. Future work involves building such profiling support in ILLIXR-A similar to ILLIXR-L.

4.3 PERFORMANCE COMPARISON OF DIFFERENT VIO PARAMETERS

4.3.1 VIO Parameters

In this section we compare four different configurations used to calculate user's current pose with varying complexity of inputs and algorithms. We used the Medieval Room application for all of these experiments.

1. Monado
   Monado's rotation prediction uses only Gyroscope inputs from the IMU. This configuration is the original Monado code and does not use ILLIXR-A.

2. IMU Integrator
   This configuration does not use OpenVINS and only uses the RK4 IMU Integrator for pose prediction. So, it only uses IMU inputs (Accelerometer + Gyroscope) and no Camera images. It uses ILLIXR-A with all the core ILLIXR-A components other than OpenVINS.

3. VIO Android
   This configuration uses ILLIXR-A with OpenVINS plugin. The OpenVINS parameters used for this configuration are the ones tuned for Android. It uses a single image for VIO and thus uses Monocular setting.

4. VIO Desktop
   This configuration is same as the VIO Android configuration but uses OpenVINS parameters used for evaluation of ILLIXR-L. It uses stereo images as input.

Table 4.2 summarizes the OpenVINS parameters used for VIO Android and VIO Desktop configurations:

4.3.2 CPU Utilization

Figure 4.9 shows the average CPU utilization of the four-different settings for pose prediction. We can see that CPU utilization of Monado is the highest. This is because of a performance bug in Monado. The IMU polling code in Monado is inefficient because of a busy while loop. Whereas in the IMU plugin in ILLIXR-A this bug is not present and thus we can see relatively lower CPU utilization.

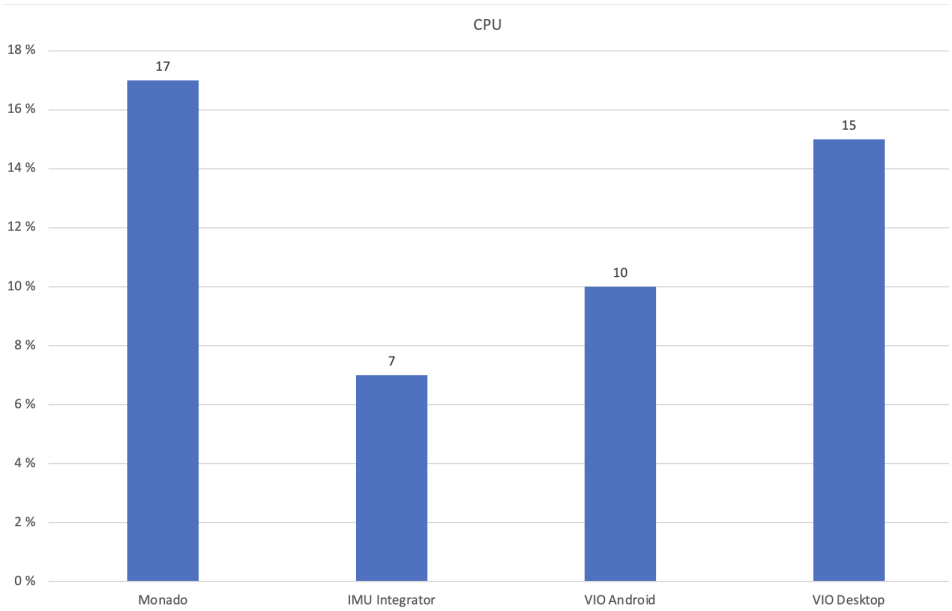

Figure 4.9: CPU utilization of the runtime for Monado, IMU Integrator, and VIO with Android and Desktop Parameters.

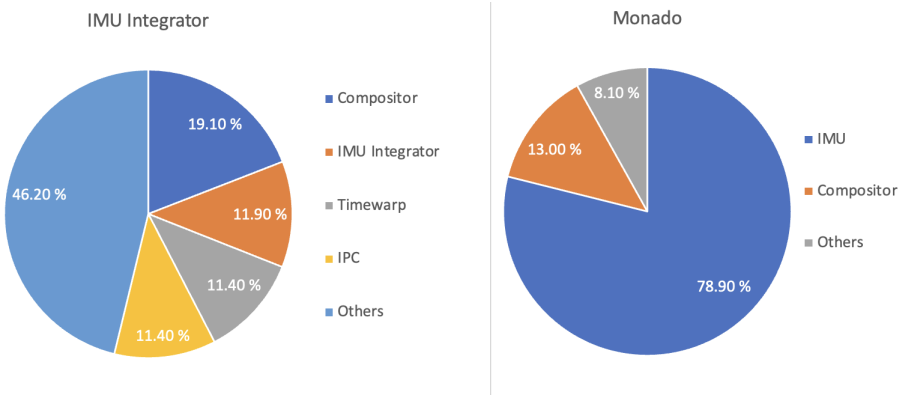

Figure 4.10: IMU Integrator vs Monado CPU Usage

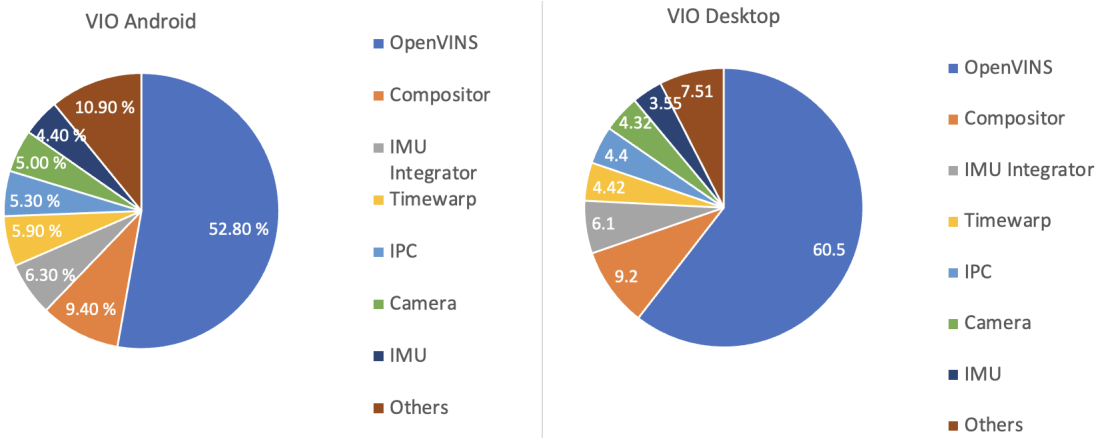

Figure 4.11: VIO Android vs VIO Desktop Parameters CPU Usage

Open Vins Parameter	ILLIXR-L Parameters	Android Parameters
Camera Input	Stereo	Monocular
Maximum SLAM Features	50	20
Maximum SLAM Features in Update	25	10
Maximum MSCKF in Update	999	100
Camera Intrinsic Calibration	True	False

Table 4.2: VIO Parameters

Figure 4.10 shows the breakup of CPU utilization by different components for IMU Integrator and Monado settings. We can see that IMU polling takes the most CPU time in the Monado setting. The compositor is the highest contributor in IMU Integrator and second highest in Monado.

From Figure 4.9 we can see that the configurations that use OpenVINS use significantly more computational resources than the IMU Integrator because it also uses Camera Images for tracking. VIO Android uses less computational resources than VIO Desktop due to the reduction in features tracked and use of Monocular tracking instead of Stereo. Figure 4.11 further shows the contribution of different components to CPU utilization in both the configurations. Note that VIO contributes about 60% of the total CPU utilization with the Desktop parameters and about 52% with the Android parameters. However, average CPU usage of VIO Desktop is about 50% higher than VIO Android and thus it takes significantly more overall CPU time.

4.3.3 Memory Usage

Figure 4.12 shows the memory usage of the four different configurations. We can see that the configurations that use ILLIXR-A (IMU Integrator, VIO Android and VIO Desktop) use much more memory than Monado. This is because these configurations allocate memory for all the ILLIXR-A plugins in addition to Monado.

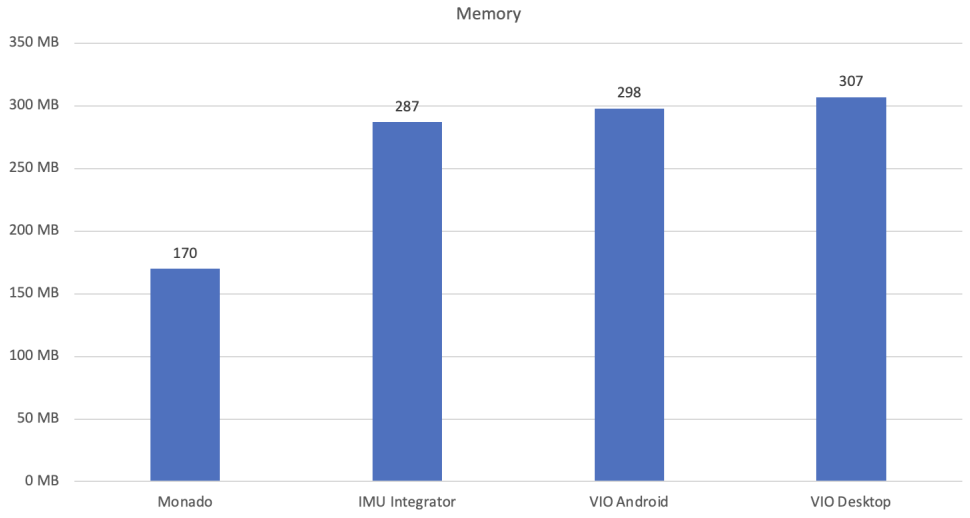

Figure 4.12: Memory Usage of Monado, IMU Integrator, and VIO with Android and Desktop Parameters. IMU Integrator uses similar amount of memory as VIO settings because both of these configurations use the same plugins in ILLIXR-A. Only difference is that OpenVINS does not perform any expensive computer vision tasks in the IMU Integrator setting.

Platform	Sponza	Materials	Platformer	AR Demo
Desktop	3.1±1.1	3.1±1.0	3.0±0.9	3.0±0.9
Jetson-HP	13.5±10.7	7.7±2.7	6.0±1.9	5.6±1.4
Jetson-LP	19.3±14.5	16.4±4.9	11.3±4.7	12.0±3.4

Figure 4.13: MTP for different Applications [1]

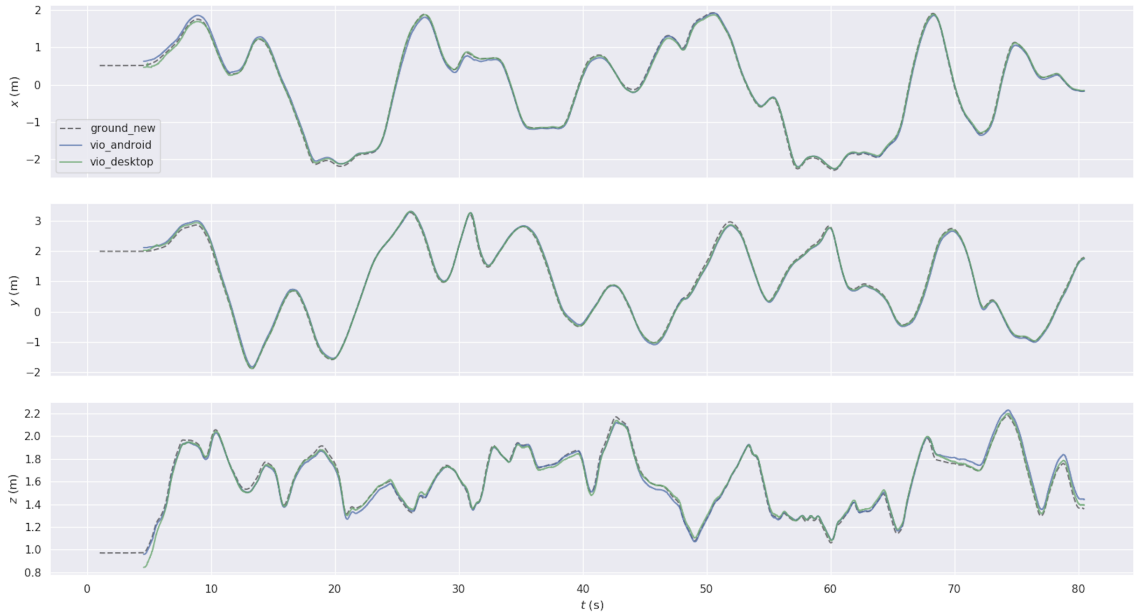

Figure 4.14: VIO Android and VIO Desktop Trajectories with EuRoC Dataset

4.4 QOE METRICS

4.4.1 Motion to Photon Latency (MTP)

The target MTP for AR is 5ms and VR is 20ms [1]. Figure 4.13 shows the MTP values for different applications (Sponza, Materials, Platformer and AR Demo) run on Desktop and Jetson (High-Power and Low-Power settings) from the ILLIXR paper [1]. In our measurement of MTP for ILLIXR-A using the GlDemo application with VIO Android setting, the average MTP was determined to be 12ms, with a standard deviation of 1.7ms (using a methodology similar to that in [1]). Consequently, the measured MTP value in ILLIXR-A is notably higher than that achieved on Desktop but comparable to the results obtained on Jetson (albeit for a much simpler application).

4.4.2 Trajectory

To compare the quality of pose prediction between VIO Android and VIO Desktop parameters we plotted the pose generated by these configurations against the ground truth pose of the EuRoC dataset. Figure 4.14 shows the trajectories. We can see the trajectories are very close to the ground truth trajectory for both VIO Android and VIO Desktop parameters.

We can see that the trajectories with VIO Android and VIO Desktop are very similar. This shows that we do not lose much tracking accuracy due to reduction in OpenVINS parameters. Table 4.3 shows the absolute and relative difference in the pose generated w.r.t ground truth pose. We can see that the absolute pose error is slightly lower in the case of VIO Desktop than VIO Android. However, relative pose error is almost similar for the two configurations.

4.4.3 Frame and Tracking Quality

Even though pose returned from VIO Android and Desktop are similar, there is a difference in the quality of the frames between the two configurations. From the following video links, we can see that the experience is more jittery with the VIO Desktop setting than with the VIO Android setting. The reason is VIO with Desktop parameters takes too long to estimate the pose due to which an older pose is used for a longer time. We observe that IMU Integrator is not able to track and the scene drifts away. However if we only enable rotation then the scene drifts slowly. The video link added here shows the slow drift case. Further, Monado can only provide the rotation information thus we only see rotation changes in the Monado video.

1. Monado: Video Link.

	Absolute Pose Error		**Relative Pose Error**	
	VIO Desktop	**VIO Android**	**VIO Desktop**	**VIO Android**
max	1.758999	1.786787	0.286528	0.286528
mean	0.914218	0.924620	0.014045	0.014030
median	0.863482	0.871581	0.012730	0.012327
min	0.067720	0.095470	0.000000	0.000000
rmse	0. 992371	1.009453	0.017545	0.017314
std	0.386011	0. 405060	0.010515	0.010145

Table 4.3: Pose Error with respect to EuRoC's Ground Truth Pose

2. IMU Integrator: Video Link

3. VIO Android: Medieval Room and GlDemo

4. VIO Desktop: Medieval Room and GlDemo

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

Our experience in porting an end-to-end XR system to the Android platform was accompanied by several challenges. Camera calibration issues, software dependencies, compiler differences, and hardware resource constraints were among the obstacles encountered. Moreover, finding suitable applications for evaluation posed a challenge due to the limited use of OpenXR on Android smartphones. To overcome this, we developed a Unity application that supports OpenXR, works seamlessly with Monado, and can be extended to other runtimes with the appropriate OpenXR Loader. The porting of ILLIXR-L to Android unlocks opportunities to evaluate and optimize various XR components on this platform. The modular design of ILLIXR-A facilitates this process, as different components can be modified and interchanged independently.

The evaluation section of our work emphasizes the significance of optimizing algorithms like Visual Inertial Odometry (VIO) to deliver a smooth user experience. If VIO calculations take a substantial amount of time, tracking can fail, particularly when sudden movements occur. Therefore, it is crucial to fine-tune VIO according to the hardware capabilities of the device. Presently, VIO has been tailored for the Samsung S22 Plus smartphone, but in the future, we anticipate having different configurations based on the specifications of each device.

We also presented Quality of Experience (QoE) metrics by calculating Motion-to-Photon (MTP) latency and compared it with ILLIXR-L running on a high-end Desktop PC and NVIDIA Jetson. We also evaluated the quality of trajectory prediction with different VIO parameters and showed that we can achieve similar accuracy while reducing computation time. Finally, we created videos showing a subjective comparison of quality. While the experiments performed thus far are preliminary, we hope that our work provides a foundation for researchers to carry out XR research on the Android platform.

One of the future directions for this work involves exploring efficient VIO algorithms. Currently, we observe tracking loss when the phone is moved suddenly due to a reduced number of points available for tracking. Retaining feature points for longer durations could enhance smooth tracking and accommodate sudden movements. An optimized VIO algorithm has the potential to make tracking more robust. Additionally, the Samsung phone used for evaluation possesses three cameras, but we are currently utilizing only one due to the slower performance of OpenVINS with stereo. Exploring how to effectively leverage the capabilities of all the cameras is a future undertaking. Furthermore, porting ILLIXR-A to Qualcomm Spaces represents an intriguing future direction, as it would enable users to utilize ILLIXR-A with an HMD and a smartphone.

Improvements can also be made to the current ILLIXR-A application to enhance usability for developers. For instance, adding a user interface to facilitate plugin selection and report the device's computational and sensor capabilities would streamline the development process. We anticipate that ILLIXR-A will serve as a valuable tool for XR researchers, enabling them to enhance algorithms and build efficient hardware tailored for the Android platform.

REFERENCES

[1] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair, B. Tian, H. Yuan, J. Zhang, and S. V. Adve, "Illixr: Enabling end-to-end extended reality research," in *2021 IEEE International Symposium on Workload Characterization (IISWC)*, 2021, pp. 24–38.

[2] M. Huzaifa, R. Desai, S. Grayson, X. Jiang, Y. Jing, J. Lee, F. Lu, Y. Pang, J. Ravichandran, F. Sinclair et al., "Illixr: An open testbed to enable extended reality systems research," *IEEE Micro*, vol. 42, no. 4, pp. 97–106, 2022.

[3] "Illixr website," https://illixr.org/.

[4] "Monado - open source xr platform," https://monado.dev/, Accessed 2020.

[5] "Valve index," https://store.steampowered.com/valveindex.

[6] "Microsoft holoportation," https://www.microsoft.com/en-us/research/project/holoportation-3/.

[7] "Ultraleep - hand tracking," https://www.ultraleap.com/company/news/blog/how-hand-tracking-works/.

[8] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, "Kimera: an open-source library for real-time metric-semantic localization and mapping," in *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2020. [Online]. Available: https://github.com/MIT-SPARK/Kimera

[9] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4666–4672.

[10] "Platformer 3d," https://github.com/GDQuest/godot-beginner-2d-platformer.

[11] "Godot material testers," https://godotengine.org/asset-library/asset/123.

[12] "Godot sponza," https://gitlab.freedesktop.org/monado/demos/godot-sponza-openxr.

[13] "Google arcore," https://developers.google.com/ar.

[14] "Snapdragon spaces," https://www.qualcomm.com/products/features/snapdragon-spaces-xr-platform.

[15] "Thinkreality a3 hmd," https://www.lenovo.com/us/en/thinkrealitya3/?orgRef=https%253A%252F%252Fwww.google.com%252F.

[16] "Illixr version 3.1.2," https://github.com/ILLIXR/ILLIXR/releases.

[17] "Android camera api," https://developer.android.com/ndk/reference/group/camera.

[18] "Android sensor api," https://developer.android.com/guide/topics/sensors/sensors_overview.

[19] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1280–1286.

[20] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[21] "Allan variance ros," https://github.com/ori-drs/allan_variance_ros, Accessed 2020.

[22] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, 2016. [Online]. Available: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract

[23] "Samsung s22 plus specifications," https://www.samsung.com/global/galaxy/galaxy-s22/specs/.

[24] "Monado android teapots," https://gitlab.freedesktop.org/monado/demos/androidteapots.

[25] "Medieval room asset from unity asset store by omni studio," https://assetstore.unity.com/packages/3d/environments/free-medieval-room-131004.

[26] "Android native activity," https://developer.android.com/reference/android/app/NativeActivity.