European Media and Immersion Lab

D3.3 – Hardware integration & Adaptation of foundation libraries of AR Magic Lanterns

Work Package 3 – Lighthouse Projects at Laboratory Nodes

Authors:

Paul Hine (UPF)
Renato Muñoz (UPF)
Narcis Pares (UPF)

Grant Agreement number **101070533**
Action Acronym **EMIL**
Action Title **European Media and Immersion Lab**
Call **HORIZON-CL4-2021-HUMAN-01**

Co-funded by the European Union

UK Research and Innovation

EMIL project is funded by the European Union and UK Research and innovation.

Version date of the Annex I against which the assessment will be made	*Start date of the project*	*Due date of the deliverable*	*Actual date of submission*	*Lead BEN / AP for the deliverable*	*Dissemination level of the deliverable*
18.3.2022	1.9.2022	31.8.2023	31.8.2023	UPF	Public

Document reviewer(s)	
Name	*Beneficiary*
Juhani Tenhunen	AALTO

Abstract

This deliverable describes the advances in hardware and system integration of the devices known as Augmented Reality Magic Lanterns, designed and developed by the UPF partner. The document describes the progress in moving from the initial TRL4 demonstrator that had been tested under controlled conditions at the historical site of Barcino (the ancient Roman Barcelona), up to the desired TRL8 device that will be achieved by the end of EMIL. In this process we describe the reasons for leaving behind the off-the-shelf-hardware version of the TRL4 demonstrator prototype and justify the need for a different HW integration. Hence, this document provides an account of all the investigations, hardware analysis, computer vision utilities, platform compatibilities, etc., that are needed for reaching a final integration and how the UPF team has undertaken these tasks.

Contents

1 Introduction

The AR Magic Lantern is an augmented reality flashlight that enables users to see and interact with virtual content projected onto physical surfaces around them. It is designed especially to support *group-oriented*, *co-located*, and *situated* AR experiences. This device is the implementation of a novel AR paradigm that UPF has defined, known as the “World-as-Support" (WaS) paradigm. This paradigm provides huge advantages over the existing usual AR paradigms when analysing group-based user experiences in public spaces such as heritage sites. The most common AR paradigm is that known as “Window-on-the World” (WoW) which is implemented by mobile devices such as smartphones and tablets. These devices force users to put all their attention on the screen and therefore the users pay little attention to the physical world (“digital dividers”) and are difficult to share with other users. The other existing AR paradigm is that of “Extended Vision” which is implemented by AR headsets. These devices provide narrow fields of view, ghostly virtual images and, in their current technological state they are not very robust for using in public spaces and are quite costly. Both these two types of paradigms and their respective devices force every user to have to use/wear one such device and users are never too sure of what other users are seeing. This does not facilitate users with motor or cognitive special needs to enjoy the augmented experiences and also breaks the group dynamics of groups of visitors to these sites. The AR Magic Lanterns, as implementers of the WaS paradigm, project the augmented stimuli on top of the real-world surfaces, adapting the projection such that they look adequately blended in the real world. This makes it accessible to a group of visitors (typically 2 to 8 individuals) without forcing each one to use/wear one such device and allowing for face-to-face communication between them enjoying the visualization, discussion, and interaction of the experience as a group. For further details on the paradigm and the device, EMIL’s website [1] provides a comprehensive overview.

The prototype or demonstrator that UPF had achieved before the start of EMIL was based on off-the-shelf components that were placed within a 3D printed external casing that looked like a flashlight. The components were:

- a smartphone that acted as the computing device for performing the computer-vision SLAM computations to determine the pose of the device and as the virtual world controller and renderer (figure 1b),
- a pico-projector to project the augmented stimuli onto the physical world around the user (figure 1a.1),
- a loudspeaker to provide sound feedback to the user (figure 1a.3),

[1] https://emil-xr.eu/lighthouse-projects/upf-ar-magic-lantern/

- a battery to feed the devices during a sufficient time to make the experience rich for the user (figure 1a.2).

This device showed a number of weaknesses as well as some black boxes that we could not control. This is why this lighthouse project from EMIL is moving forward in designing and developing a new AR Magic Lantern based on electronic devices and components that are better controlled by us and provide solutions to the weaknesses that our demonstrator showed.

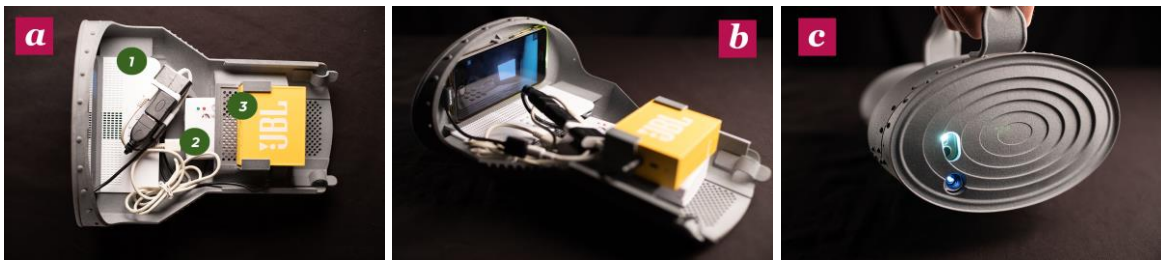

Figure 1 Detail views of the TRL4 prototype of the AR Magic Lantern: (a) top view, case open, (b) side view, case open, and (c) front view, case closed.

This report: (a) reviews motivations and challenges for integration, (b) shows progress on integration, (c) provides detailed specifications of hardware and software development, and (d) discusses work to come.

2 Motivations and challenges for integration

Augmented reality requires precise real-time tracking and localization of the user's visual perspective (6-DOF head tracking for HMDs and device tracking for WoW) in order to convincingly blend virtual content with the physical world. The AR subsystems that perform these functions utilize sophisticated computer vision and machine learning processes that are resource intensive and must therefore be highly optimized to run on mobile hardware. Early prototype iterations of the ARML chose to use a pre-built AR subsystem that was already optimized to run on a smartphone. The options available at the time were ARKit from Apple and ARCore from Google. ARKit was chosen because of its superior tracking performance and ability to store and reload a scene consisting of virtual content positioned accurately with respect to a physical space (persistence). The prototypes using ARKit used an iPhone 7 and were later updated to use the iPhone SE 2020.

Although the pre-built and optimized AR subsystem provided by ARKit allowed us to rapidly prototype a hardware and software platform for the ARML, it presented limitations that blocked us from improvement in six key areas: form factor, deployment, tracking stability, tracking performance, localization reliability, and system performance.

2.1 Form factor

In the previous prototype, the iPhone's front-facing camera was required for tracking, so the front part of the lantern had to accommodate a smartphone mounted with the camera facing forward. Even with a small smartphone (iPhone 7), the width of the lantern front was an awkward 21cm. Many high-end smartphones have large screens, meaning the design would have to accommodate up to 16x8cm, increasing the lantern width even more. To support the design metaphor of the torch (flashlight), our goal is to produce a body for the lantern that is cylindrical with a diameter no larger than 12 cm. Single-board computers (SBCs) are a good substitute for a smartphone in terms of size and compatibility, but the computing power of SBCs lags behind the latest smartphone. A big challenge will be optimizing both systems to run on the single computer (see *System Performance)*. In the previous prototype, a great deal of space inside the lantern housing was occupied by cables and adapters. The space required by cables and adapters remains a problem even as we move to SBCs, so our objective is to source or build custom cabling to minimize the space they occupy.

2.2 Deployment

Loading the software onto the previous prototype required that we rebuild and re-install the application on the smartphone. Because the phone runs the Apple mobile OS, iOS, the build and install process is hindered by a complex licensing protocol enforced by Apple. To load new software and content onto the lantern, the developer had to open the lantern body, remove the phone and attach it to a computer. This put a lot of physical stress on the phone and lantern body and thus negatively impacted the durability of the whole system. The deployment process for loading updates to software or content to the lantern should be simple and require no special licensing. The developer should be able to load updates wirelessly to the lantern using only Unity. We must integrate all of the application components (gameplay, tracking) into one Android application. Many of the open-source programs and tools for tracking are tailored to Linux and must be ported to Android. Therefore, we must write a layer that binds the Unity application code to the C++ code for tracking.

2.3 Tracking Stability

While ARKit provides a very robust tracking and localization system, it is designed to run specifically for the limited visual hardware of the smartphone (e.g., single camera) and for specific environmental conditions (e.g., good lighting). For the conditions required by the lantern, tracking became very unstable. In the lower light conditions needed to see the projections, it had difficulty tracking because smartphone cameras are not sensitive to low light. Also, since the camera captures not only the physical world, but also the projected augmented stimuli, the camera does not perceive a stable physical world. Hence, the projections themselves often confused the tracking system because they were seen as features of the environment. The lantern requires a custom tracking system that works

with multiple cameras and infrared illuminators (active stereo) so depth measurements can enhance tracking accuracy and stability in low lighting. The tracking software layer must be customized to ignore the area of the image containing the projector output. There are several mature open-source projects available for tracking that use the stereo hardware and depth data, but they must be tuned for use in the environmental conditions we expect and adjusted to ignore the "noise" introduced by the projected content.

2.4 Tracking Performance

The tracking system in the previous prototype, ARKit, is optimized to run on mobile hardware. Even on an iPhone 7 from 2016, the frame rate and latency were low enough that a simple screen-based AR experience ran smoothly with no perceived lag. However, screen-based AR can get away with more lag because it can artificially “slow down” (or delay) the video feed to match the latency and framerate of the tracking updates. When projecting AR content onto physical surfaces, the latency and frame rate demands are higher because the artificial sync used for screen-based AR is not available. Even on an Apple iPhone 12 Pro from 2020, there is enough lag in the lantern prototype to ruin the perception that AR content is really attached to physical locations. The lantern’s motion-to-photon latency (the time required to update the projected image after a movement) must be sufficiently low that AR objects appear to stay firmly attached to their physical location while the user moves the lantern across them. We must be able to optimize all parts of the motion-to-photon pipeline to reach this goal. But even fully optimized, we expect that the processing required by the SLAM tracking system (long-range) incurs latency that exceeds the motion-to-photon target. We must therefore use direct sensor data to do low-latency (short-range) "dead-reckoning" updates to the orientation. The low-latency/short-range and high-latency/long-range systems must operate in parallel and cooperate.

2.5 Localization Reliability

On the previous prototype, ARKit provided methods to store and re-load maps it made of a physical space, which helped us create persistent content for our prototype museum experiences. There is a step after re-loading a map when the system must re-localize the camera within the saved space. As there is no dependable way to help the system re-localize, the technician would have to move the device slowly and carefully around the play area, sometimes for a few minutes, before the re-localization completed. This is not a practical solution for end-users. An end-user (e.g., museum visitor) starting an experience with the lantern should be able to point the lantern at a QR code or poster on the wall and dependably begin the experience seconds later with correct re-localization. Incorporating markers into a tracking session is not a basic function of SLAM tracking systems. On the other hand, the one we are using, RTAB-Map, does provide some support for it. We must prepare the map in a specific way to support markers.

2.6 System Performance

In the previous prototype, the integration between Unity and iOS meant that apps made with Unity ran with native performance on the iPhone. AR Foundation is the Unity package that allows it to work with ARKit on the device, also at native speeds. However, AR processing is very compute intensive, which left fewer resources available to the rendering and gameplay parts of the app. There were noticeable performance degradations like dropped frames and tracking loss as the AR and graphics subsystems competed for resources. Most components of ARKit are a "black box" that do not allow tuning or customization. The tracking system of the lantern must adapt to the resource demands on the system. For example, if there is AR content in view with big resource needs, the tracking system should fall back to a lower-fidelity but higher-performance mode. To minimize latency and overall device size, the goal is to run all lantern functions from a single computer. This will require fine-tuned optimization, excellent resource management, and, as mentioned above in *Deployment*, a translation of some Linux software to the Android platform.

3 Progress and specifications

3.1 Computer vision and tracking

To build and optimize a Computer Vision (CV) tracking system prototype, we need to try many combinations of inputs, processing stages, and outputs. We chose to develop the computer vision system on top of Robot Operating System (ROS) [2], a set of tools and libraries for building quick and robust robotics projects on Linux systems. Because the vision challenges of an autonomous robot are similar to those of our augmented reality device, we found ROS well-suited to our requirements.
For our SLAM framework, we chose RTAB-Map (Real-Time Appearance-Based Mapping), a state-of-the-art, open-source implementation that is designed to take advantage of image inputs that include depth data (see *Hardware Platform*). Like ROS, RTAB-Map also has a modular architecture that allows us to try different combinations of processing components (e.g., feature extractors) throughout the vision pipeline.

The CV tracking system solution is a set of configuration files and software components that tune the ROS and RTAB-Map frameworks to run optimally with our hardware and tracking requirements, including:

[2] https://www.ros.org/

Open-source modules developed to solve common CV problems:

- We use many routines from OpenCV, a library that provides hundreds of utility functions for manipulating and processing image-type data.
- We also use open-source modules in our system called "feature extractors" for finding and tracking the position of recognizable regions (features) of images.

Software components we developed to solve challenges unique to our tracking requirements:

- Our system must provide an *anchoring* function that allows users to attach virtual content to specific points (anchors) in a physical space. We wrote a software component in Python that optimizes RTAB-Map behavior around anchors to provide the best performance and experience possible when the user is interacting with anchored content.
- Additionally, our system must ignore the image area occupied by the projector content, which would confuse the tracking algorithm, so we modified the RTAB-Map source to allow us to define which areas of the image to use for feature finding. For starting the tracking session, we configured RTAB-Map so users can just point at a visual marker (poster or fiducial marker) on the wall.

Figure 2 The hardware configuration of the computer vision benchmarking system using the HTC Vive Tracker (a.3) for ground truth in a large test environment (b). This shows an earlier configuration where we evaluated the RealSense D435 (a.1), which has no built-in IMU in combination with the IMU readings from the RealSense T265 (a.2).

Testing the performance of a computer vision tracking system requires that we compare our solution to a "ground truth" benchmark, i.e., test it against the performance of another solution which has a known accuracy. For the benchmark, we used the Steam VR Tracking System with 4 base stations

and an HTC Vive Tracker unit (figure 2b). The accuracy of this system has been tested in several studies and found to be in a large but bounded range of ≤1mm to 7.5cm [3].

To compare both systems, we utilize two key performance indicators: Absolute Trajectory Error (ATE) and Relative Pose Error (RPE). ATE provides a numerical value indicating the extent to which the estimated trajectory deviates from the ground truth. It accomplishes this by measuring the Euclidean distance between corresponding positions or points along the paths. On the other hand, RPE measures the disparity between the estimated relative poses and the ground truth poses. Typically, these poses are represented as transformations or rotations between frames. The computation of RPE involves comparing the relative poses in terms of translation and rotation errors. In both cases, lower values signify higher accuracy and a closer alignment with the ground truth.

We ran the benchmark test across a set of state-of-the-art feature extractors and compiled the results into a dataset and report. We will use these results to choose an optimal set of feature extractors for the final hardware resources available to the device.

To perform the benchmark trial, we attached the Vive Tracker (figure 2a.3) to our CV tracking system (figure 2a), which used an Intel RealSense D435 (figure 2a.2) for depth imaging and a RealSense T265 (figure 2a.2) for the IMU. Note that on the final prototype we use the RealSense D455 (figure 5a), which combines the stereo imaging with an onboard IMU on a single device. We then ~~and~~ moved around a room 8m by 8m (figure 2b) and recorded the data from both systems. After the walk, we transformed the data received by each system to a common coordinate system and compared the tracking results (figures 3 and 4).

[3] Holzwarth, V., Gisler, J., Hirt, C., & Kunz, A. (2021). Comparing the Accuracy and Precision of SteamVR Tracking 2.0 and Oculus Quest 2 in a Room Scale Setup. *2021 the 5th International Conference on Virtual and Augmented Reality Simulations*, 42–46. https://doi.org/10.1145/3463914.3463921

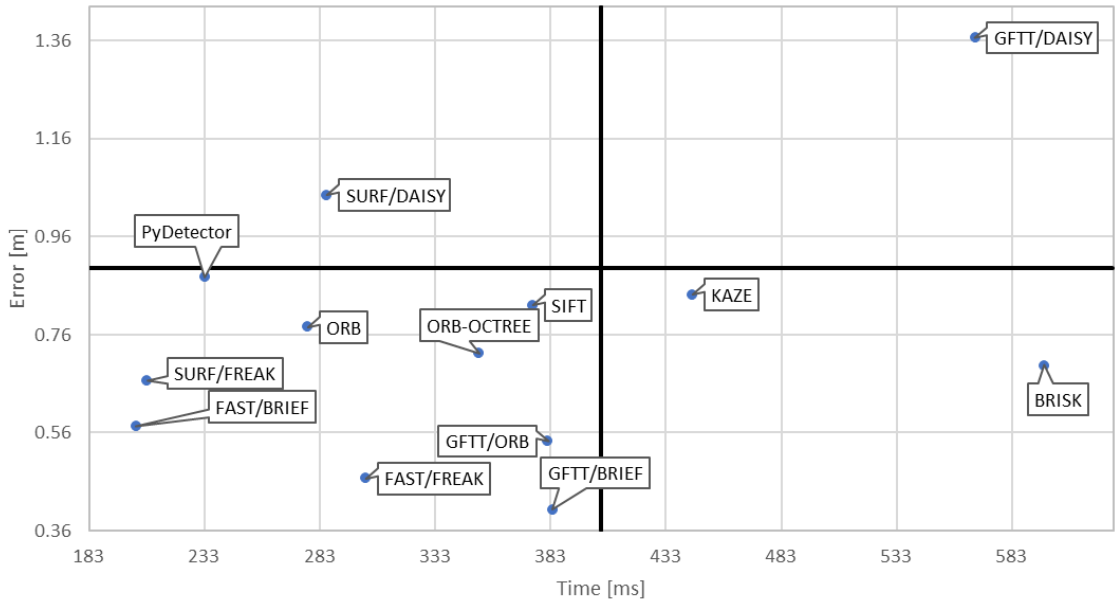

Figure 3 Performance comparison of feature extractors with medium resolution (640x480) images: processing time vs maximum absolute trajectory error.

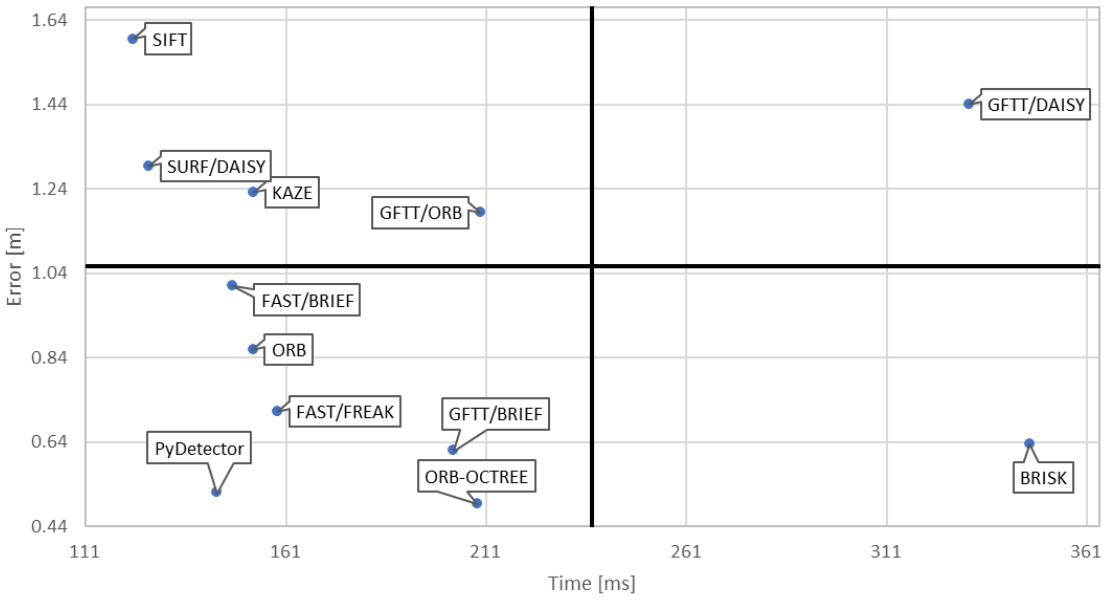

Figure 4 Performance comparison of feature extractors with low resolution images (320 x 240): processing time vs maximum absolute trajectory error.

The accuracy of this system has been thoroughly examined in multiple studies, yielding consistent results within a specific range. To find a desirable balance between translational error and processing time, we plotted the results of Absolute Trajectory Error (mATE) on the Y-axis against the maximum processing time (mPT) on the X-axis (figures 3, 4). Across all the feature extractors tested, the system

demonstrated an mATE range of 35 to 135 cm and mPT range of 190 to 590 ms when using medium resolution images (figure 3), and an mATE range of 40 to 160 cm and mPT range of 120 to 350 ms when using low resolution images (figure 4). To optimize the system's performance for our use case, we utilized the PyDetector feature extractor using low quality images. With this feature extractor, we achieved an mATE of 51 cm and mPT of 144 ms.

3.2 Unity integration

Unity is one of the world's most popular game engines and software for game creation. By developing our game platform for the ARML on top of Unity, we get the benefit of a rich editor application and extensive developer community. We modelled the integration between Unity and the CV tracking system on AR Foundation, an SDK maintained by Unity that allows developers to write code against a common API and run it on a variety of AR subsystems, including ARKit (Apple), ARCore (Google), and OpenXR. We do not plan to make our integration an official plugin for AR Foundation because that would require additional developer resources and time. However, we plan to follow its patterns for a subset of functions so that its API is familiar to developers who have already worked with AR Foundation.

The ARML Unity integration is a series of Unity components and utility functions. Currently, the integration connects the Unity game runtime to the ROS runtime that controls our CV tracking system. This connection is built on top of existing libraries from the ROS community that allow messages to be passed between ROS and Unity over any network connection. Our Unity components have associated C# code that *subscribes* to events that fire whenever ROS sends a message. For example, when ROS sends an updated camera position/orientation message, our Unity component handles the message event by updating the camera position/orientation in the game. In the other direction, Unity sends ROS the anchor positions from the scene when the game begins.

Because the connection between Unity and ROS runs over a network, we are able to easily test and iterate our integration. One strength of engine/editors like Unity is the ability to rapidly prototype and test "live" within the editor interface. To facilitate this, we first start the ROS runtime on the Linux system connected to the camera and sensors (not the one running Unity). The ROS runtime begins publishing messages over the network. We then start the Unity editor "play" mode, which runs our Unity application live in the editor. The Unity application receives the messages from ROS and updates the game components in Unity. We can very easily stop the "play" mode, make changes to our Unity code, run it again and see the changes, all without restarting the ROS runtime. When we want to test it on our prototype device, we build the Unity code to our onboard graphics/gameplay computer (see Hardware Platform), which is connected by local network (direct ethernet cable) to the ROS computer.

3.3 Hardware platform

3.3.1 Computing and graphics

The move away from "off-the-shelf" mobile hardware (i.e., smartphone and pico-projector) has necessitated extensive investigation and trials to determine the appropriate hardware platform for the TRL8 prototype(s) (see *Motivations and challenges for integration*). Because we want the ARML to support large, intricate games populated by rich 3D graphics, the primary requirement of our compute/graphics module was that it provided a powerful GPU and an OS that allows Unity to run natively and take advantage of that GPU (hardware acceleration). Unity has build utilities that allow it to run natively with hardware acceleration on iOS devices, Windows devices, Linux devices running on Intel CPUs, and Android devices running on ARM. For reasons already stated, iOS devices were not an option. We evaluated mobile Windows and Linux systems such as the Latte Panda 3 Delta but found their GPU hardware acceleration to be low-performance compared with the best Android device running an ARM CPU/GPU.

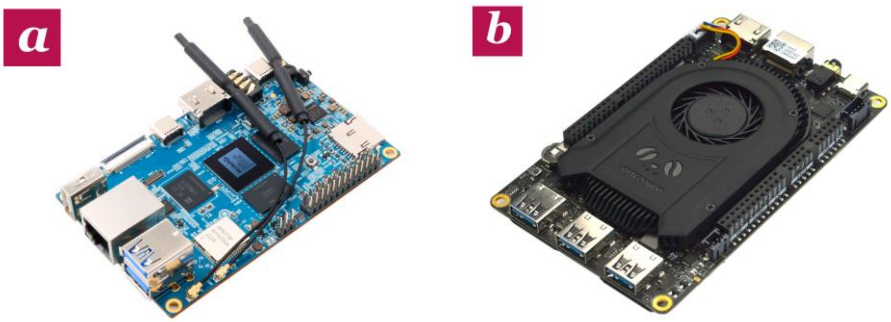

Figure 5 The single-board computers (SBCs) used in the prototype of the ARML: (a) Orange Pi 5B and (b) LattePanda 3 Delta 864

We chose the Orange Pi 5B single-board computer (SBC) as the compute unit for the ARML prototype (figure 5a). The Orange Pi 5B uses the RockChip RK3588S system-on-a-chip (SOC), which provides a high performance graphics coprocessor (GPU), the ARM Mali G610, and a powerful 8 core ARM processor (CPU) running at up to 2.4 GHz. The SBC also provides crucial onboard features such as support for up to 32 GB RAM (we are using 4 GB currently), up to 256 GB onboard storage (eMMC), dual-band Wi-Fi, Bluetooth, audio input and output, USB-C connectivity, and HDMI video output. Orange Pi supplies an Android build that is tuned to work on this specific SBC, so it is well suited for running games built with Unity.

Because ROS will not run natively on Android, we currently run it on a second SBC, the aforementioned LattePanda 3 Delta 864 (figure 5b). This SBC features a powerful mobile Intel CPU, the Celeron N5105, which can run at clock speeds of up to 2.9 GHz. It has out-of-the-box support for

all recent versions of the Ubuntu Linux operating system, which means it easily runs ROS and RTAB-Map.

The system running ROS must be connected via network to the system running Unity (see *Unity Integration*), so on our ARML prototype we connect the Orange Pi 5B to the LattePanda 3 Delta with an ethernet cable. This forms an ad-hoc network between the two computers, so no additional networking hardware (e.g., routers) are necessary.

3.3.2 Cameras and sensors

There were many options and combinations available for camera and sensor hardware. We based our selection on components with good long-term availability that were already well-supported by RTAB-Map, the framework underlying our CV tracking system. In addition, we used environmental criteria in our selection, such as the requirement for a system that works well in low light and that can block/ignore interference the projector. We evaluated two depth cameras with integrated motion sensors (IMUs), the Intel RealSense D455 (figure 6a) and the OAK-D Pro Wide (figure 6b).

Figure 6 Stereo cameras and IMUs evaluated for the prototype: (a) Intel RealSense D455, (b) OAK-D Pro W, (c) Intel RealSense 435 and Intel RealSense T265 used as a pair.

We decided to use the Intel RealSense D455 camera because it had better compatibility with RTAB-Map. It has a sufficiently wide field of view that the projector content only fills 20% of the image area. It features two wide-angle monochromatic cameras sensitive to infrared light, a wide-angle, high-resolution RGB camera, and an infrared pattern projector. The software onboard the camera fuses the data from the two stereo cameras, enhanced by the projected infrared dot pattern, with the RGB image to create an RGB-D (RGB plus depth) image for each recorded frame. This system is known as "active stereo," which gives vastly improved depth quality compared to "passive stereo," in which the two stereo images are processed without the infrared dot pattern.

3.3.3 Projector

When it came to choosing a projector for the ARML prototype, we looked for a solution that had a sufficiently small form factor, ample brightness, support for HDMI video input, and a very large depth-

of-focus (DOF) so that it would project crisp images onto surfaces at a large range of distances from the user (without requiring mechanical focus adjustment). While there are some powerful off-the-shelf projectors on the market that fill these criteria, we looked instead to "development modules," which are hardware kits that connect several low-level components into a product that is not for commercial sale but instead designed to be a functional unit within a pre-commercialization prototype. They have open, well-documented specifications that allow a manufacturing partner to further integrate their components when designing the commercialized product.

For the projector, we chose the "EKB E4500MKII Focus" (figure 7a), a variant of an official evaluation module produced by Texas Instruments, the world's largest manufacturer of DLP (digital light processing) imaging devices (e.g., screens, televisions, projectors). The variant features a special lens that gives it improved DOF over other evaluation modules, allowing us to set the focal length such that images appear sharp in a range of 1.4 to 5 meters. It provides an effective brightness of 600 ANSI Lumens, which is sufficient for use indoors in low to medium lighting conditions, and a sufficiently high image resolution (1280 x 800) for our needs. We will add a post-processing stage in Unity that compresses the dynamic range of the projected images and improves overall contrast and clarity.

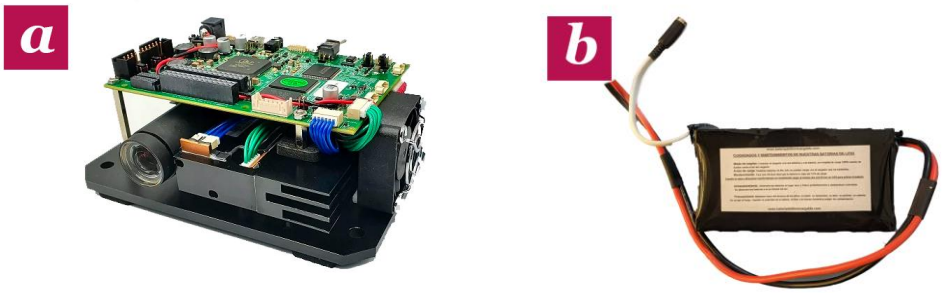

Figure 7 (a) The EKB E4500 MKII Focus is a DLP projector development module for use in embedded systems. (b) Custom rechargeable lithium-ion battery pack (12 VDC).

3.3.4 Batteries and Power

Our primary requirement for the power system of the ARML prototype is that it use a single battery to power all the devices. This avoids the downtime necessary to charge the battery in-device and instead allows the end user to exchange the battery when it is depleted. Therefore, we set our power capacity requirement such that it allows the system to run uninterrupted for at least two hours, which gives game designers the freedom to build long, complex games and experiences for the ARML.

We worked with a battery manufacturer to build a custom rechargeable battery pack that contains a set of lithium-ion cells (type 21700) and a circuit board to safely manage charging and discharging (figure 7b). It outputs 12 VDC and has a capacity of 25 Ah. One lead from the battery goes directly to

the projector, which takes a 12 VDC input, and another goes to the LattePanda 3 Delta, which also takes a 12 VDC input. To supply power to the Orange Pi 5B, which requires 5 VDC input, we use a circuit board that converts 12 VCD input to 5 VDC output.

3.4 Industrial Design

To test different hardware component configurations, we needed a physical platform that supported interchangeable modules. The industrial design objectives for this version of our prototype were therefore focused on modularity and versatility. We already had identified a set of components that needed to fit into the support (see *Hardware Platform*), so we based the design around them and left room for adjustment (figure 8a). For example, the mounting rails on the base and upper supports have the same width and support component orientations either parallel or orthogonal to the length of the body. In the current configuration, we chose to mount all of the boards orthogonally because it left their ports accessible on the open sides of the body (figure 8b). The physical support is printed using low-cost 3D printers and assembled with standard hardware (e.g., screws, bolts, nuts, washers, etc).

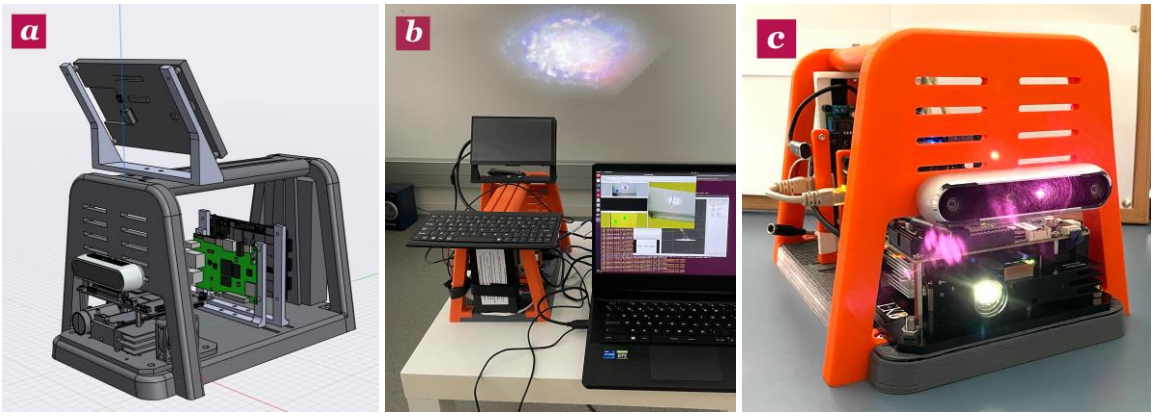

Figure 8 The physical support for new lab prototype: (a) CAD design, (b) open design allows easy connection of parts to development computer, (c) front view without development attachments.

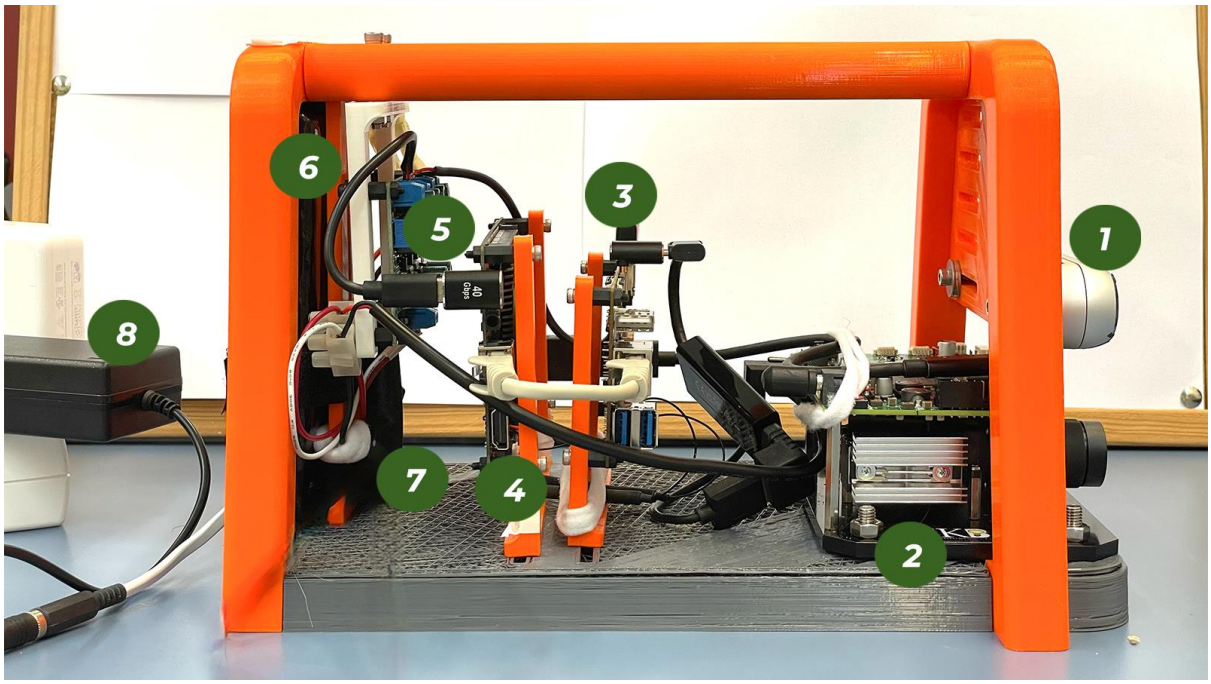

Figure 9 Detail of ARML physical support, showing: (1) Intel RealSense D455 depth camera, (2) E4500MKII projector, (3) Orange Pi 5B compute module with ethernet connection to (4) LattePanda 3 Delta tracking computer, (5) voltage step-down converter module, (6) battery pack, (7) power switch, (8) battery charger.

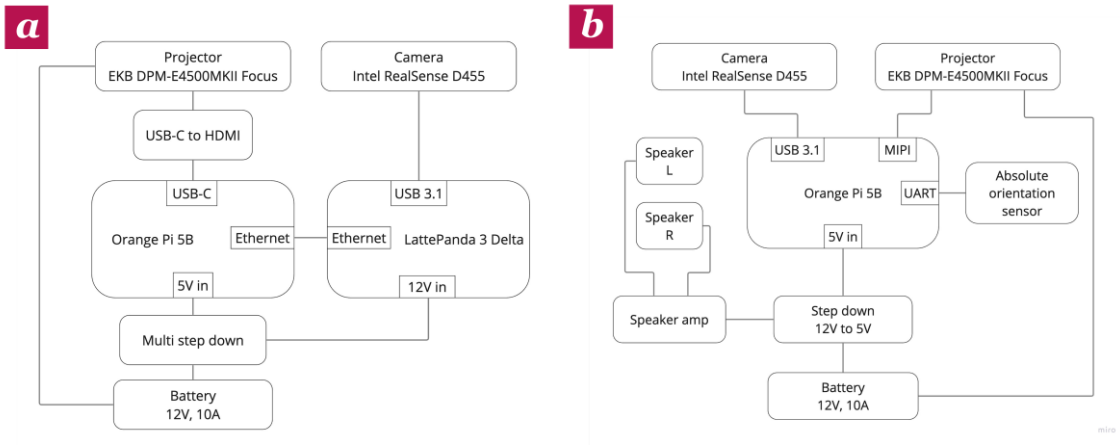

Figure 10 Block diagrams showing (a) current ARML prototype [pictured in figure 8] and (b) further integrated prototype with single compute board and audio subsystem.

4 Conclusion and Next steps

Building our CV tracking system on ROS has allowed us to quickly build, test, and improve a working tracking system for the ARML. However, it must run on Linux, while our requirements for the final

prototype dictate that the Unity application be deployed to a system running Android (see *Motivations and challenges*). Therefore, we are now running two separate computers and linking them over a network connection (see *Hardware Platform*, *Unity Integration, figures 9 and10a*). This introduces complexity, weight, higher power consumption, and, more critically, an unacceptable motion-to-photon latency. Therefore, the next step is to write a tightly-coupled C++ program for Android that connects RTAB-Map to our Unity integration without requiring ROS or Linux. This will allow us to run the prototype on a single compute board (figure 10b).

To further reduce the motion-to-photon latency we plan to add a high-frequency absolute orientation sensor module connected directly to the compute board via a high-speed bus (UART). The module is powered by the Bosch BNO055 smart sensor, which integrates an accelerometer, gyroscope, geomagnetic sensor and an ARM microcontroller running sensor fusion software onboard. The sensor fusion software combines all of the sensor data and reports and absolute orientation. We will also add an audio subsystem that includes a 2-channel amplifier board and two speakers.

In parallel, we are working with an industrial designer to construct a compact and ergonomic enclosure for the components. We are also working with an electrical engineer to build custom cabling to connect all of the components, which will allow us to further reduce the final space requirements of the lantern prototype.

Appendix 1: List of Terms

Term	Definition
Android	Operating system maintained by Google that runs on mobile devices and embedded systems (like smart TVs, appliances, etc)
AR Subsystem	The SDK that packages low-level systems like the tracking system to allow developers to mix virtual and physical content into a cohesive application.
ARCore	An AR subsystem for Google Android devices
ARKit	An AR subsystem for Apple iOS devices
Augmented Reality (AR)	Broadly used to describe any technology that overlays digital content onto specific locations in a physical space.
Fiducial marker	A 2D image placed in a physical space to serve as a calibration reference point for a visual tracking system.
Inside-out Tracking	A tracking system that uses cameras and sensors on the device to be tracked. The tracking range is unbounded.
iOS	Operating system that runs only on Apple mobile devices (smartphones and tablets)
LattePanda 3 Delta	Single-Board Computer with good Linux performance
Linux	General purpose open-source operating system that supports a large range of devices
Mixed Reality (MR)	Broadly used to describe a subset of augmented reality that allows embodied user interaction with 3D digital objects in the world.
Orange Pi 5	Single-Board Computer with powerful graphics capabilities and good Android performance
Outside-in Tracking	A tracking system that requires camera and sensor units (stations) to be placed in fixed positions in the physical space. The tracking range of the system is determined by the number and range of individual stations.
RTABMAP	An open-source implementation of SLAM
Simultaneous Localization and Mapping (SLAM)	A tracking system that builds and optimizes a database of visual features to perform inside-out tracking over large areas.
Single-Board Computer (SBC)	A microcomputer that has all hardware components needed (e.g., wifi, audio, graphics, CPU, etc) onboard and often has similar power requirements as a mobile device.
Tracking system	The hardware and software elements that process camera and sensor inputs to update the position and orientation of a virtual camera.
Window-on-the-World (WoW)	A paradigm that describes the experience of using screen-based AR, as on a smartphone or tablet.
World-as-Support (WaS)	An AR paradigm that requires digital content to be projected onto physical structures of the world.