Final Project Report: 3D Marker Tracking Cameras on Low-Cost Embedded Platforms

Vishal Chandra (chandrav@umich.edu), Darren Fitzgerald (darrenwf@umich.edu),
Sonia Kim (kminseo@umich.edu), Aida Ruan (aidaruan@umich.edu),
Andrew Schallwig (arschall@umich.edu)

EECS 452 - Winter 2024, 24 April 2024

I. Background/Motivation

The core goal of our project is to implement a low-cost embedded motion capture system that can accurately localize and track points in space while running efficiently on easily-accessible hardware. This will allow small game studios, independent filmmakers, and researchers to make use of this vital spatial technology at fractions of typical costs. We find this to be a particularly interesting project due to its emphasis on accessibility in that we seek to create a solution for a common engineering task that is both affordable and robust. We also find it interesting due to the variety of signal processing and engineering challenges that the project will present. Among the techniques we will leverage are image processing, wireless communication, filtering, and embedded system engineering.

II. Existing Solutions

In movie production, video game development, and robotics, being able to effectively localize points on objects, on-screen actors, or obstacles is a necessity. Current systems used in research and industry tend to be expensive or require complicated setups. Other systems are highly engineered for challenging environments but not accessible for students, small labs, or quick prototyping. For instance, companies like OptiTrack and Qualisys provide high-precision motion capture systems, but they are typically expensive, with prices ranging from tens of thousands to hundreds of thousands of dollars, which can be prohibitive for smaller studios and independent researchers. Our approach offers a competitive advantage in terms of affordability and accessibility. By leveraging low-cost hardware like ESP32-Cam modules and open-source software, we can significantly reduce the upfront investment required for motion capture technology to less than $50, making it more accessible to a wider range of users. Additionally, our focus on simplicity and ease of setup allows for quick prototyping and experimentation, catering to the needs of small studios and researchers.

III. System Architecture

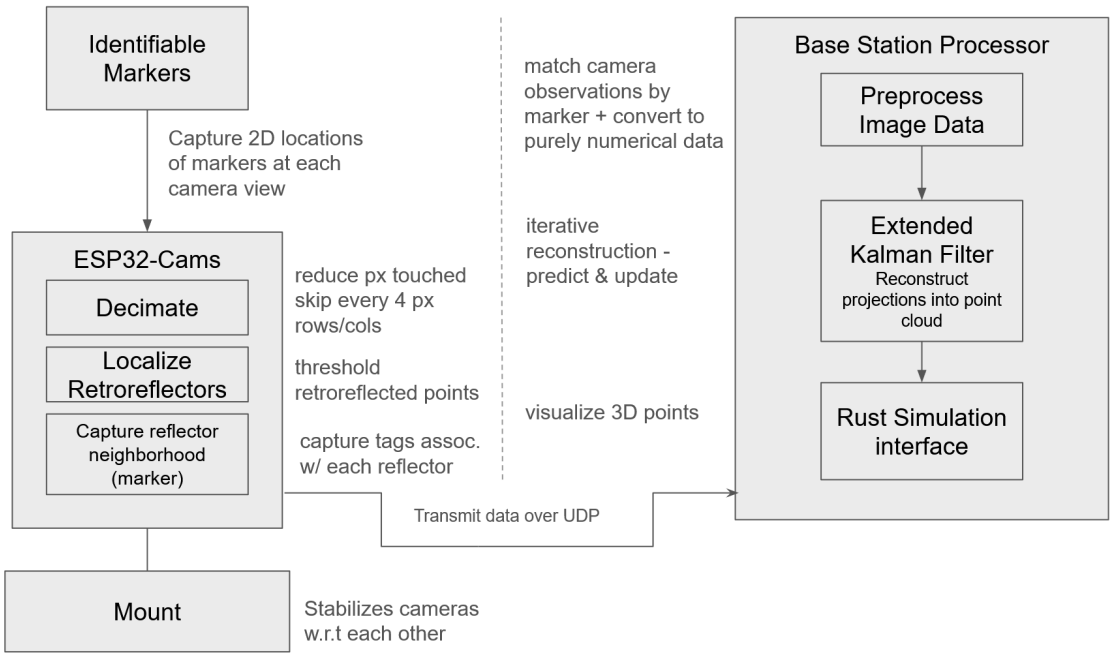

Figure 1: *Top-level design block diagram showing major components and interfaces*

As described in Figure 1, there are three main modules in this design: ESP32-Cam to collect live images and capture necessary information, UDP communication for transmitting information, and a base station to perform reconstruction and visualization.

ESP32-Cams are mounted on a solid support to stabilize their relative locations, in other words, to ensure that the positions of the cameras are consistent throughout all experiments. The primary objective is to deploy retroreflective markers within the camera's field of view and accurately process the captured images in real time. Markers have unique identification through April tags that can facilitate the ease of matching the same object across all camera views. We specifically used a 48h12 custom tag family with center 2x2 pixels undetected so that when the marker is placed in the center of the tag, it doesn't disrupt the April tag detection.

After capturing the 2D images at each camera view, we first perform some preprocessing of the images on the ESP32-Cam before transmitting marker information to a base station for further processing. This includes decimating to reduce the number of pixels touched when locating the bright retroreflective marker position. This will help accelerate the computational speed. We also capture the neighborhood of the marker for the tags associated with each thresholded marker. Then, we transmit the preprocessed data

(windowed image of the marker and the surrounding tag) from multiple ESP32-Cam units to the base station using UDP.

Within the UDP transmission step, we packetize all the necessary data to transmit to the base station for running the Extended Kalman Filter (EKF). Using the windowed image we processed from the ESP32-Cam, we run the open-source April tag detector to identify the tag IDs and center (x,y) positions of the markers. We also packetize the camera IDs associated with each center position.

On the base station, a pool of threads dedicated to processing and tag detection takes in the incoming streams of packetized windows. Each thread is dedicated to one camera's window stream and marshals packets to extract a window's sensor position and timestamp of capture. The AprilTag library is then used to check for the presence of a detectable tag and, if a tag is detected, its center position on the camera's image sensor, timestamp, and capturing camera's ID are sent to a point estimation thread pool.

Point reconstruction is performed by applying an implementation of the Iterated EKF to the data received from the image detection thread pool. To eliminate error from camera extrinsics, precomputed calibration data consisting of a rotation matrix and a translation vector concatenated together are used to identify the capturing camera's position and orientation with respect to the origin. This matrix, along with the captured tag's data from the image detection step, is routed to a thread dedicated to estimating its position. The estimate is then visualized using the Kiss3D library.

IV. Important Technical Issues

There were two main challenges we encountered throughout our project. First, when capturing raw images from the ESP32-Cam, memory allocation posed challenges in processing speed. The use of PSRAM for accessing the image buffer proved to be slower compared to SRAM. Without JPEG encoding, the ESP32-CAM only supports up to QVGA (320x240). This limited the image resolution and therefore the range of detection. If we were not constrained to doing image processing on the ESP32, first transmitting images to the base station through JPEG, then decoding, and finally running tag detection there would yield smaller-sized, higher resolution images and thus be faster and more accurate.

Secondly, Rust offered better performance compared to Python for UDP servers, especially for real-time processing where high throughput and low latency are critical. This was especially important for parallelizing the input given to the EKF and reading in

windowed images from the multiple cameras. While we experimented with a serial transmission of AprilTags, receiving, running detection, and then storing the data before running EKF was unusably slow. Therefore, it is recommended that you use Rust to implement the UDP and the EKF (since it receives the UDP packets in Rust).

V. Experiment

In the experimental setup (see Figure 2 and 3), we mount the ESP32-Cam on a solid support with ring lights attached to each of them from behind. We move the markers in the camera views and once the base station receives a series of windowed images, we visualize the motion capture (see Figure 4).

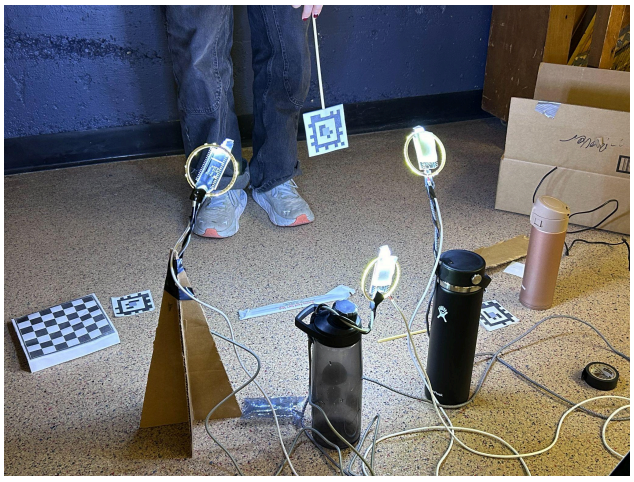

Figure 2: *Experimental setup (three cameras and one marker)*

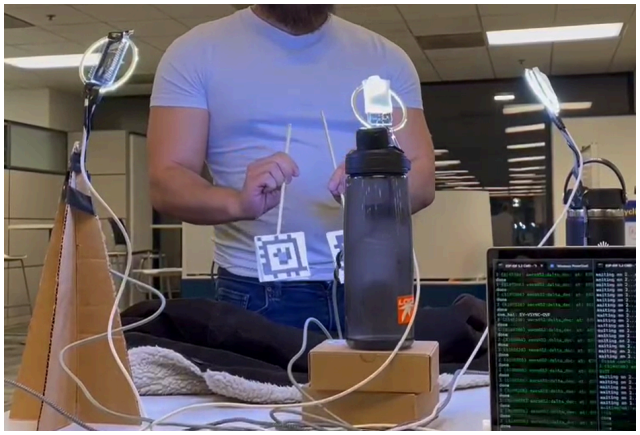

Figure 3: *Another experimental setup (three cameras and two markers)*

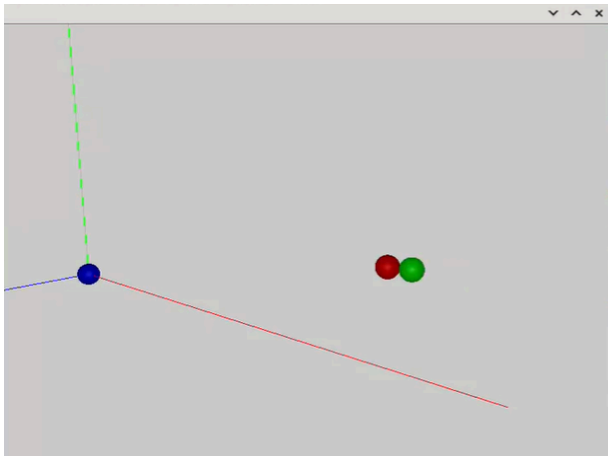

Figure 4: *Screen capture Rust visualization with two markers (blue indicates the origin of the reference axes)*

VI. Results

We created a successful 3D motion tracking system for normal movements and one tag, with the system setup shown in Figure 2 and 3. This is within a cubic area measuring approximately 1.5 feet in length, width, and height, and averages at around a 3 fps updating speed of the 3D location. For two or more tags and faster motion, the limiting factor is the speed of the windowing of detections from the ESP32-CAM. This hinders the "smoothness" of motion capture and can cause the reconstruction to sometimes miss key movements that may be more extreme.

VII. Future Work

While the current project has achieved substantial milestones and addressed key objectives, there are several directions for future exploration and improvement. The following are potential directions for future work:

1. Camera tuning:
    a. Several configurable options on the ESP32-CAM have been adjusted for optimal detection of retro-reflectors, including turning off auto-exposure and lowering brightness. However, other options such as contrast or exposure could be tuned to minimize noise, which includes incorrectly cropped images or windows not including an AprilTag being sent over.
2. User interface:
    a. Camera calibration routines can be automated to facilitate user accessibility. Currently, we obtain the images of the calibration object (chessboard) from each camera and generate camera matrices into a .npz file which is sent to the base station for the EKF. We can reduce the

number of manual data transmissions by automating this process using UDP packets.

3. Algorithmic refinement:
    a. Due to the time constraint, it was challenging to debug and fine-tune the UKF model we initially developed. For future work, it would be worthwhile to further investigate the architecture of UKF, test it on our data collection, and compare the performance against the current EKF.
    b. We can also leverage cross-validation and optimization techniques to find the optimal values of the measurement and process noise covariance matrices that minimize a predefined cost function. Currently, we are fine-tuning these parameters empirically. Objective functions may include metrics such as mean squared error, residual variance, or prediction error to quantify the performance of the EKF.

VIII. References

Choosing an AprilTag Family. (n.d.). https://www.ssontech.com/docs/SynthEyesUM_files/Choosing_an_AprilTag.html

OV2640 camera module 2 million pixels (2MP). iFuture Technology. (2024, April 2). https://ifuturetech.org/product/ov2640-camera-module-2-million-pixel-2mp/#:~:text=200%20w%20pixel%20OV2640%20camera,grade%20megapixels%20camera%20is%20optional.

Ribeiro, Maria & Ribeiro, Isabel. (2004). Kalman and Extended Kalman Filters: Concept, Derivation and Properties.