

**RTES FINAL PROJECT REPORT**

FACE TRACKER

*Induja Narayanan*

*Aditya Ganesh*

*ECEN 5623, Real-Time Embedded Systems, Spring 2025*

*May 02, 2025*

Table of Contents

[1.Introduction 3](#_Toc197179848)

[GitHub Link 3](#_Toc197179849)

[2.Design Overview 3](#_Toc197179850)

[a.Hardware Overview 3](#_Toc197179851)

[b.Software Overview 3](#_Toc197179852)

[c.High Level Software Design & Description 4](#_Toc197179853)

[Flow Explanation 4](#_Toc197179854)

[3.System Requirements 5](#_Toc197179855)

[Must Have Minimum System Requirements 5](#_Toc197179856)

[Good to Have Target Requirements 5](#_Toc197179857)

[Stretch Goals 5](#_Toc197179858)

[4.Detailed Design 5](#_Toc197179859)

[a. Service Description 5](#_Toc197179860)

[Design Descriptions 5](#_Toc197179861)

[Image Capture Service: 6](#_Toc197179862)

[Face Detection Service 7](#_Toc197179863)

[Cursor Translation Service 8](#_Toc197179864)

[Image Compression Service: 9](#_Toc197179865)

[Logging Service 10](#_Toc197179866)

[b. Message Sequence Diagram 11](#_Toc197179867)

[c. RM analysis 12](#_Toc197179868)

[d. System Testing – Plan and Results 12](#_Toc197179869)

# 1.Introduction

Face tracker is a real-time embedded application designed to translate facial movements into cursor control on a computer interface. Using a camera as the input device, the system captures video frames, detects the user's face, calculates its center coordinates, and moves the cursor accordingly. The system is optimized for low-latency performance and includes features for logging and image compression to support analysis and storage.

## GitHub Link

<https://github.com/Induja21/RTES_Final_Project/tree/master>

# 2.Design Overview

## a.Hardware Overview

The Face Movement to Cursor Tracker System relies on minimal hardware components to achieve real-time performance

**Webcam:** A standard USB webcam from Logitech C2470 with resolution of atleast 640x480 and supporting YUYV pixel format. This shall be used for face and eye detection

**Host System:** Raspberry Pi 4

## b.Software Overview

The system is implemented in C++ and leverages several libraries for real-time processing:

**Core Libraries**:

* **OpenCV (v4.x)**: For face detection (CascadeClassifier), image conversion (cvtColor), and JPEG compression (imencode).
* **ZeroMQ (libzmq)**: For non-blocking inter-service communication (PUB/SUB, PUSH/PULL sockets).
* **V4L2 API**: For video capture and buffer management.
* **uinput**: For cursor control via Linux kernel input subsystem.
* **C++ Standard Library**: For file I/O (std::ofstream), mutexes (std::mutex), and string handling (std::stringstream).

**Development Environment**:

* Compiler: GCC
* Build System: Makefile for dependency management and compilation.

**Operating System**:

* Linux

## c.High Level Software Design & Description

A diagram of a diagram

AI-generated content may be incorrect.

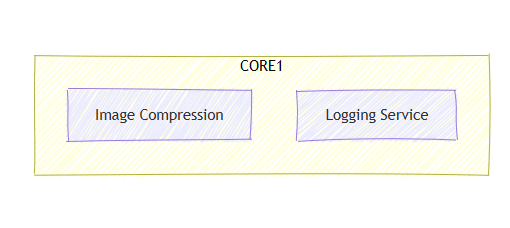
### Flow Explanation

* **Image Capture** captures frames from the webcam and publishes them via ZMQ.
* **Face Detection** and **Image Compression** subscribe to frames.
* **Face Detection** calculates and send face center coordinates to **Cursor Translation**.
* **Cursor Translation** moves the cursor and sends logs of face center along with new cursor position to **Logging** service
* **Image Compression** and **Logging** receive the captured image and sent meta data saves along with timestamp. The logging service writes to disk only during shutdown.

### Core allotment

A screenshot of a computer

AI-generated content may be incorrect.



# 3.System Requirements

## Must Have Minimum System Requirements

1. The system must detect face orientation.
2. The system should estimate head direction from camera input in real time.
3. The system must map head orientation to real-time cursor movement with low latency.
4. The system must support rate monotonic scheduling of at least two periodic real-time services with measurable jitter and WCET profiling.

## Good to Have Target Requirements

1. The system should log head orientation and movement with accurate timestamps to a file for later review or debugging.
2. The system should compress captured images periodically and store them in memory for optional user inspection or training.

## Stretch Goals

1. Use **eye tracking** (pupil position) to control the cursor for movement control.
2. Add standalone calibration services for eye and face detection

# 4.Detailed Design

## a. Service Description

### Design Descriptions

The system comprises five services, each designed with real-time constraints in mind:

#### Image Capture Service:

A diagram of a workflow

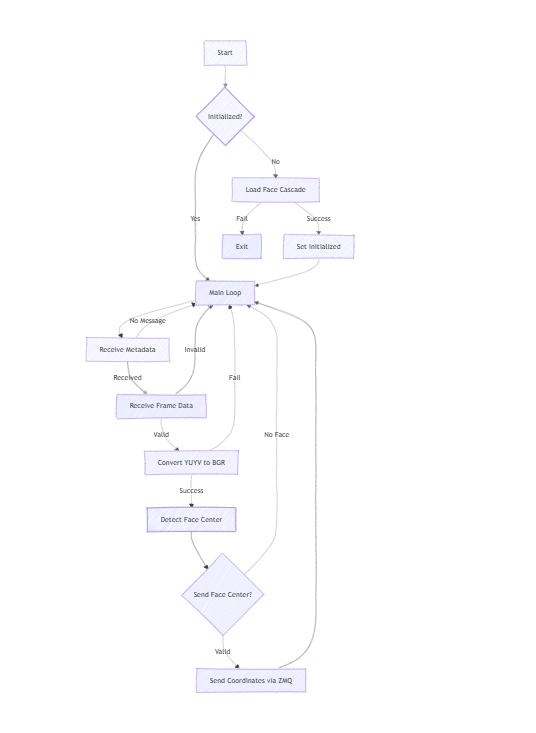
AI-generated content may be incorrect.

This service captures 640x480 YUYV frames from the webcam using V4L2.Uses memory-mapped buffers (V4L2\_MEMORY\_MMAP) with 4 pre-allocated buffers to avoid dynamic allocation.Sends frames as pointers via ZMQ zero-copy mechanism (zmq::message\_t with free\_buffer callback).

**Real-Time Aspects**

* Non-blocking V4L2 operations (O\_NONBLOCK, EAGAIN handling) ensure no delays in frame capture.
* Streaming mode (VIDIOC\_STREAMON) provides continuous frame delivery at 20 FPS (50 ms per frame).
* Deadline: 60 ms, achieved through zero-copy and non-blocking ZMQ sends.

#### Face Detection Service



This service detects faces in frames and computes the face center using OpenCV.Converts YUYV to BGR for OpenCV compatibility, then to grayscale for detection.Uses CascadeClassifier with optimized parameters (scaleFactor=1.1, minimumNeighbour=2).

Real-Time Aspects:

* Non-blocking ZMQ receive/send ensures no delays in frame processing.
* Optimized face detection parameters and minimal post-processing (processes only the first face) keep execution within 100 ms.

#### Eye Detection Service

The **Eye Center Detection Service** identifies the center of the left eye in video frames using OpenCV. It converts YUYV frames to BGR for OpenCV compatibility, then to grayscale for processing. Using CascadeClassifier, it detects faces and eyes with parameters (scaleFactor=1.1, minimumNeighbour=2), identifies the left eye, and locates the eyeball center via HoughCircles (minDist=cols/8, minRadius=rows/6, maxRadius=rows/2), stabilizing the result by averaging the last 5 centers.

**Real-Time Aspects**:

* Non-blocking ZMQ receive/send prevents delays in frame handling.
* Optimized detection parameters and focused processing (detects left eye only, stabilizes with fixed window) ensure execution within 100 ms.

A diagram of a flowchart

AI-generated content may be incorrect.

#### Cursor Translation Service

A diagram of a software process

AI-generated content may be incorrect.

This servies Translates face center coordinates into cursor movements using uinput.It applies smoothing with a fixed window (5 frames) to reduce jitter. And it Maps coordinates to screen space and sends EV\_ABS events via uinput.

Real-Time Aspects

* Fixed smoothing window ensures deterministic timing
* Direct uinput writes provide low-latency cursor updates
* Non-blocking ZMQ receive/send ensures responsiveness.
* Deadline: 50 ms, achieved through lightweight processing and direct hardware access.

#### Image Compression Service:

A diagram of a process

AI-generated content may be incorrect.

It compresses the frames to JPEG and saves them to disk.Converts YUYV to BGR, compresses using cv::imencode with fixed quality (80).Writes to disk with buffered I/O (std::ofstream in binary mode).

Real-Time Aspects:

* Non-blocking ZMQ receive ensures no delays in frame acquisition.
* Deadline: 250 ms

#### Logging Service

A diagram of a software company

AI-generated content may be incorrect.

Logs cursor movements and timestamps to a CSV file. Uses mutex-protected I/O for thread safety. Generates timestamps with CLOCK\_MONOTONIC for high-resolution logging.

Real-Time Aspects:

* Non-blocking ZMQ receive ensures no delays in message processing.
* Buffered writes and one-time file initialization minimize I/O latency.
* Deadline: 70 ms, achieved through lightweight logging operations.

## b. Message Sequence Diagram

A screenshot of a computer screen

AI-generated content may be incorrect.

The sequence diagram for the **Face Movement to Cursor Tracker System** illustrates the end-to-end flow of data through its five services—**Image Capture**, **Face Detection**, **Cursor Translation**, **Image Compression**, and **Logging**—along with interactions with external entities like the **Webcam**, **Display (uinput)**, and **Disk**. The process begins with the **Image Capture Service**, which dequeues a YUYV frame (640x480) from the webcam using non-blocking V4L2 (VIDIOC\_DQBUF) within its 60 ms deadline. It then publishes the frame metadata and data via ZeroMQ (ZMQ) PUB/SUB sockets to both the **Face Detection** and **Image Compression** services, using a zero-copy mechanism to minimize overhead. The non-blocking nature of ZMQ (indicated by dashed lines) ensures that **Image Capture** can re-queue the buffer (VIDIOC\_QBUF) and proceed without waiting, maintaining real-time performance.

The **Face or Eye Detection Service**, operating within a 100 ms deadline, subscribes to the frame, converts it from YUYV to BGR, detects the face using OpenCV’s CascadeClassifier, and computes the face center coordinates. It then pushes these coordinates to the **Cursor Translation Service** via ZMQ PUSH/PULL, again non-blocking to avoid delays. **Cursor Translation**, with a stringent 50 ms deadline, applies smoothing, maps the coordinates to screen space, and moves the cursor via uinput, while also sending log messages to the **Logging Service** (250 ms deadline) for timestamped CSV storage. Concurrently, the **Image Compression Service** (70 ms deadline) processes the same frame, converting it to BGR, compressing it to JPEG, and saving it to disk. The diagram highlights real-time constraints with deadlines noted above each service, and asynchronous ZMQ communication ensures that services operate independently without blocking, meeting their respective timing requirements.

## c. RM analysis

## d. System Testing – Plan and Results