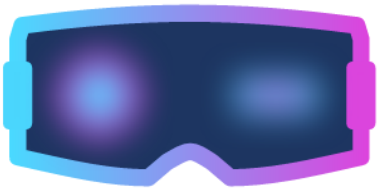


CognitiveXR

User Documentation

Authors
Thomas Rausch
CognitiveAR Contributors

https://netidee.at/cognitivear
Version: 2021-01-24

Table of Content

Introduction

The vision of CognitiveXR is a platform to seamlessly interface wearable AR devices with smart city environments that are pervaded by edge computing infrastructure. The platform is tailored to enable multi-user cognitive assistance applications that require (1) real-time sensor data from the environment, such as approaching cars or pedestrians, and (2) computing resources for low-latency video processing using AI.

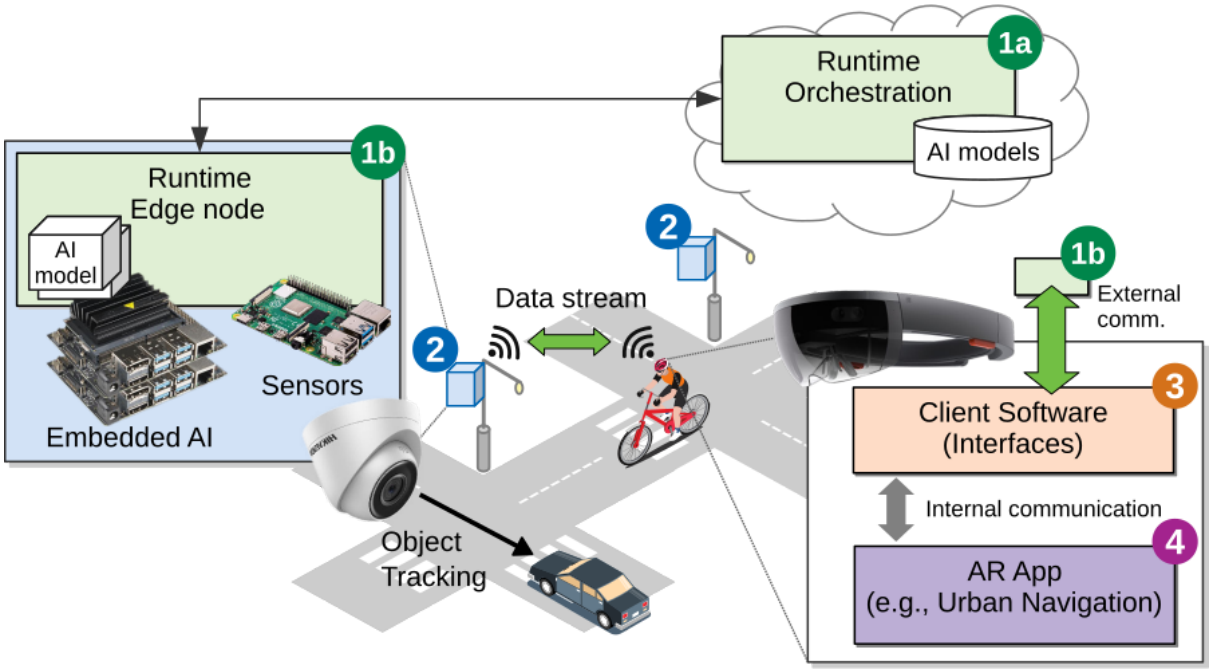
The **users of CognitiveXR** are primarily **developers of Augmented Reality applications**, as well as **operators of edge computing infrastructure** that hosts services for such applications. Most relevant to users are therefore the APIs provided by CognitiveXR, as well as the components involved in running the entire platform on edge infrastructure.

This document summarises the architecture and main components of CognitiveXR, and collects links to documentation of individual components for more details.

Architecture

The following figure shows the main architectural components of our system

(1) Runtime platform that provisions AI services from a repository in the cloud (1) to edge nodes (1b)
(2) Edge nodes such as a smart lamp post, providing compute resources and access to environmental sensor data;
(3) device-specific client software on an AR or mobile device, that facilitates transparent access to the system
(4) actual AR application running on the specific device.



Components

The central software components of CognitiveXR are:

- **CogStream**: a platform for distributed real-time AI-based video analytics:
  - **CogStream Engine:** a video analytics engine processes a video feed received over the network and annotates it with metadata. Engines are instantiated dynamically by the Mediator.
  - **CogStream Mediator**: allows AR apps to request CogStream Engines available in the system, and negotiates a video transformation pipeline
- **CPOP**: The Cyber-Physical Object Positioning (CPOP) System
- **cognitivexr-clients**: Client libraries to connect to CogStream and CPOP from AR devices

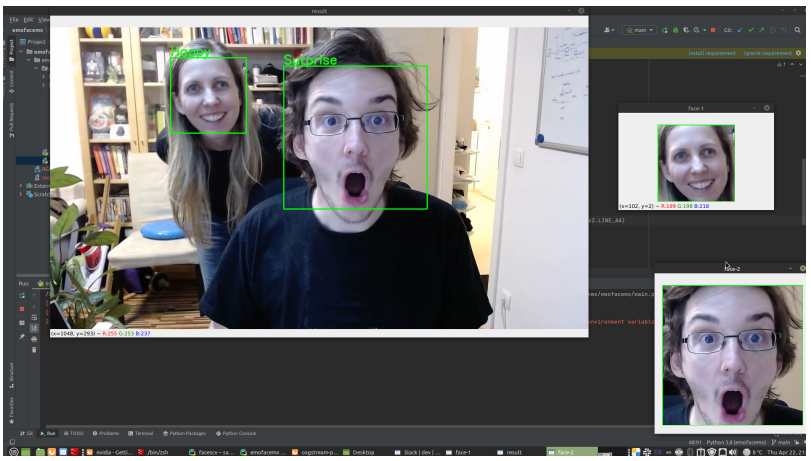
You can choose to run CognitiveXR on any container-based deployment platform, such as Docker-Swarm or Kubernetes. For communication between devices and edge nodes, we use standards like websockets and MQTT.

CogStream

Suppose you want to build an assistive AR app that helps the wearer of AR glasses to interpret the emotional response of conversational partners. The AR glasses would record the person you are conversing with, offload the video to a video analytics engine that responds with the bounding boxes of the face and the label of the emotional response. Such operations are computationally expensive and generally cannot be performed on the device, but have to be offloaded to nearby infrastructure with AI-accelerators. Moreover, the sender (AR glasses) and receiver (video analytics engine) need to negotiate a video streaming format.

CogStream facilitates this end-to-end. AR devices request an engine (e.g., for facial expression recognition) via the Mediator, which instantiates the engine (if available), and negotiates the streaming format. The AR device then connects directly to the newly created engine, and begins receiving the metadata processed by the engine. The AR app can then further process or visualise the metadata as required by the application.

For development purposes, you can also instantiate and connect to engines directly. The images on the right shows the debug view of a particular engine that uses OpenCV to detect faces and a pre-trained MXNet AI model (FERPlus) to interpret facial expressions.



Documentation

- **Engine** streaming protocol
  https://github.com/cognitivexr/CogStream
- **Mediator**
  https://github.com/cognitivexr/CogStream/tree/main/mediator

Available engines

The following engines are available out-of-the-box. However, CogStream is pluggable, such that custom engines can be easily implemented. Engines are meant to be started by the Mediator, but can also be started directly and debugged with their respective engine clients.

- YOLOv5 object detector:
  https://github.com/cognitivexr/CogStream/tree/main/engines/engines-py/yolov5
- Facial expression recognition (FER) using OpenCV and MXNet
  https://github.com/cognitivexr/CogStream/tree/main/engines/engines-py/fermx
- Simple face detection with OpenCV
  https://github.com/cognitivexr/CogStream/tree/main/engines/engines-py/facescv
- Recorder to record the received video feed (useful for debugging)
  https://github.com/cognitivexr/CogStream/tree/main/engines/engines-go/plugin/record

Cyber-Physical Object Positioning (CPOP)

Tracking objects in the immediate vicinity, even if they are not visible to the wearer of AR glasses, is a fundamental aspect of assistive AR apps for urban spaces. Suppose a cyclist is wearing AR glasses that can display approaching cars that are hidden behind the corner of a house. To correctly display virtual representations of physical objects in AR apps, CognitiveXR tracks objects and devices in a global coordinate space. To track physical objects like cars, CPOP uses regular RGB cameras and deep learning-based object detection to translate the detected objects into 3D positions. The CPOP server sends positional updates to an MQTT broker or network, for example Mosquitto[1] or EMMA[2], respectively, that AR devices receive via subscription to the same brokers.

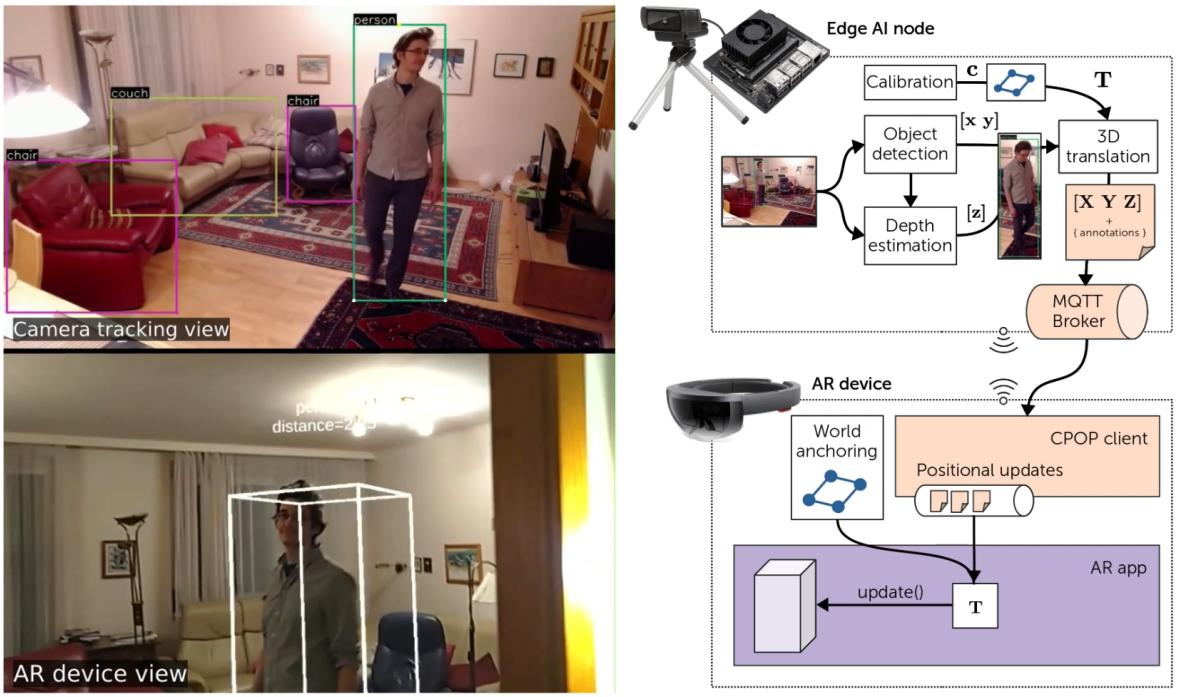
To set up CPOP, the following steps are required (details in the linked documentation)

- Connect a webcam to a device that can run Pytorch
- To anchor the camera to a common origin, run the CPOP calibration procedure using an AruCo marker
- Start an MQTT broker on a server
- Start CPOP server on a device that has a camera attached to it
- To calibrate the AR app, we do not yet provide an out-of-the-box mechanism. We have

[1] https://mosquitto.org/download/
[2] http://dsg.tuwien.ac.at/staff/trausch/pub/PID5190461.pdf

Example

Here is an example of how we used CPOP to create a simple application that can track pedestrians through walls. A detailed explanation can be found in this presentation: **IEEE VR DISCE'21: Towards a Platform for Smart City-Scale Cognitive Assistance Ap…**



Documentation

- CPOP https://github.com/cognitivexr/cpop
- Calibrating the camera: https://github.com/cognitivexr/cpop#calibrate-the-camera
- Run the service: https://github.com/cognitivexr/cpop#run-the-service

Complete deployment example

A complete example of how to deploy the entire system on to edge computing infrastructure can be found here: https://github.com/cognitivexr/cognitivear-demo