Continental Demo for Frankfurt Auto Show.

This application demonstrates the ability to run multiple network on the Inference Engine side-by-side. Runs pose estimation, face identification and voice identification to emulate a secret pose + voice based locking system.

The application behaves in the following way:

1. User walks in, application uses pose estimation network to find people and detect faces. Users are initially assigned a green bounding box, 'anonymous' id and are in a locked state for both voice and pose.
2. User waves at the camera using their right forearm (*a wave is currently programmed to be only recognized if the elbow is above the shoulder level*), application starts data recording process for the waved user. If multiple users wave at the same time, they will be processed one by one. User being recorded is assigned a red bounding box, an id and continue to be in a locked state. Users waiting to be recorded are assigned a blue bounding box instead and continue to stay anonymous till they start being recorded.
3. At the beginning of the recording process, after the first face feature is extracted, the user is checked against the database to find potential matches.
    1. If the user already exists in the database, they are given 5 seconds before recording audio for 5 seconds. If the audio matches the entry in database for that user, their audio lock is set to an unlocked state. The pose is continuously matched every frame with the pose stored in database till the pose matches, after which the pose lock is set to an unlocked state.
    2. If the user wasn't found in the database, application first records and extracts features for three angles of their face. Application then proceeds to wait for 5 seconds before recording audio for 5 seconds. Finally, the application waits for another 5 seconds before recording the pose. After the recording process is over the user is reverted to a green bounding box, has an id and is in the unlocked state.
4. If no user has waved at the camera, application tries to identify a random individual on the frame by checking their face against the database.

Setup

1. Setup compiler and PICO framework as described in https://github.com/FWDNXT/sdk
2. `pip3 install -r requirements.txt`
3. `cd tracker; python3 iou.py`

Running the Demo

```
usage: main.py [-h] [--vid VID] [--width WIDTH] [--height HEIGHT] [--simple]
               [--max-people MAX_PEOPLE] [--face-thresh FACE_THRESH]
```

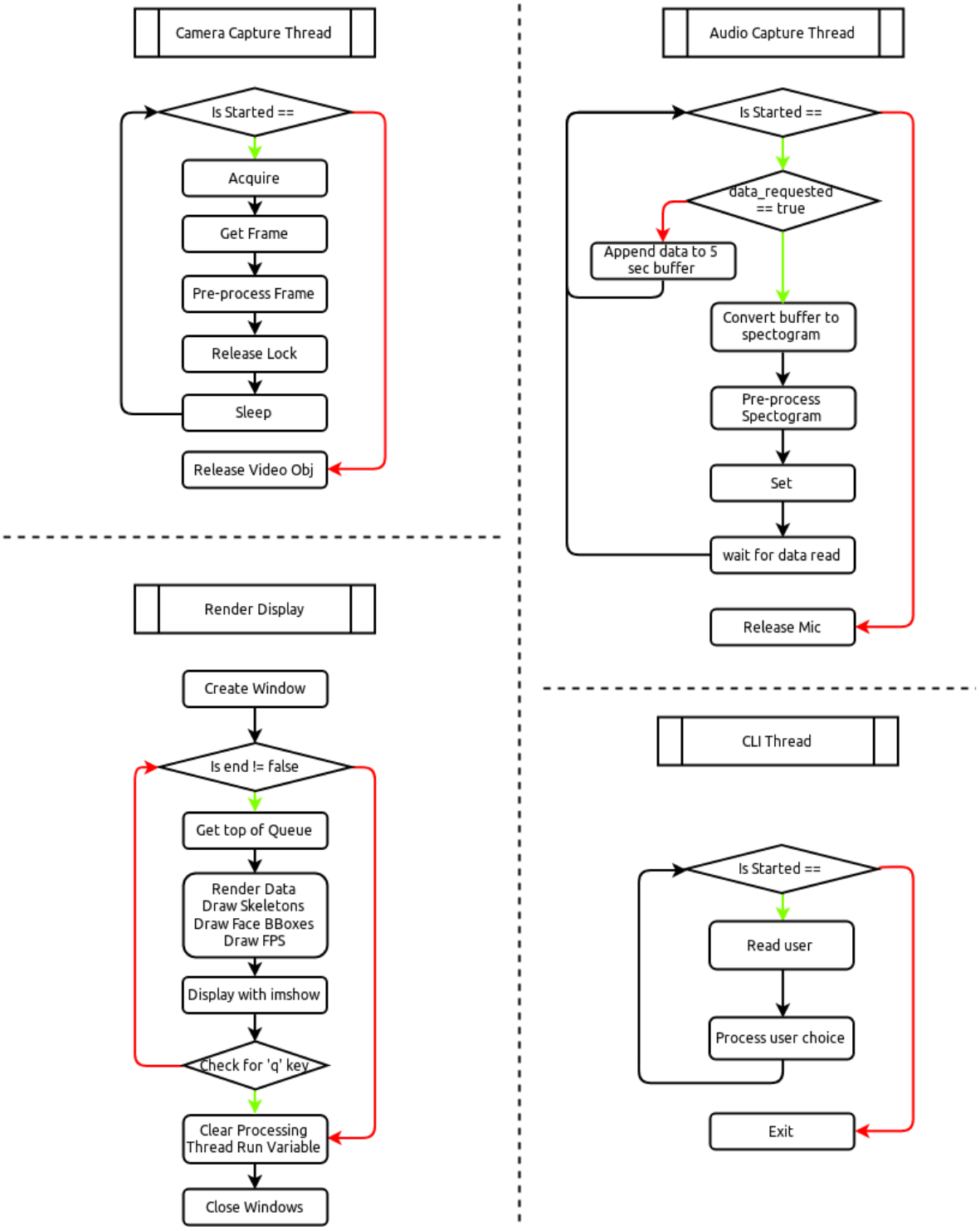
```
               [--hm-thresh HM_THRESH] [--jm-thresh JM_THRESH]
               [--device {cpu,gpu,fie}] [--db-dir DB_DIR]
               [--onnx-dir ONNX_DIR] [--bin-dir BIN_DIR]
               [--pose-weights POSE_WEIGHTS] [--face-weights FACE_WEIGHTS]
               [--voice-weights VOICE_WEIGHTS] [--bitfile BITFILE] [--load]

optional arguments:
  -h, --help            show this help message and exit
  --vid VID             Camera ID or Video File
  --width WIDTH         Video Capture Width
  --height HEIGHT       Video Capture Height
  --simple              Run in simple mode. No faceid or voice
  --max-people MAX_PEOPLE
                        Max number of entries in DB
  --face-thresh FACE_THRESH
                        Threshold for face matching in DB
  --hm-thresh HM_THRESH
                        Threshold to control keypoint extraction from heatmap
  --jm-thresh JM_THRESH
                        Threshold to control keypoint extraction from jointmap
  --device {cpu,gpu,fie}
                        Where to run models
  --db-dir DB_DIR       directory to store database files
  --onnx-dir ONNX_DIR   directory to store generated onnx files
  --bin-dir BIN_DIR     directory to store generated bin files
  --pose-weights POSE_WEIGHTS
                        Weights for Pose Estimation Model
  --face-weights FACE_WEIGHTS
                        Weights for Face Identification Model
  --voice-weights VOICE_WEIGHTS
                        Weights for Voice Identification Model
  --bitfile BITFILE
  --load                Load bitfile
```

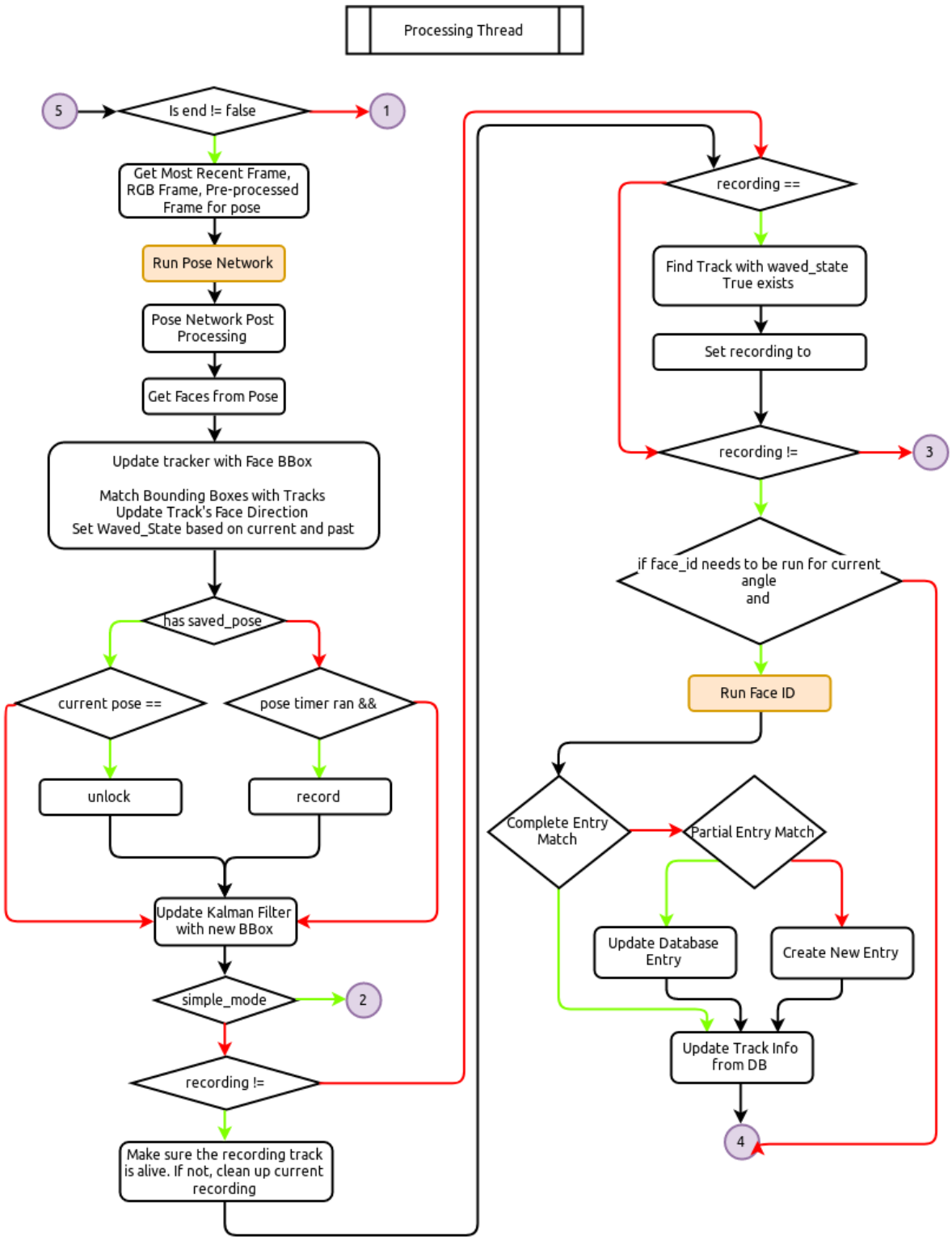
Design

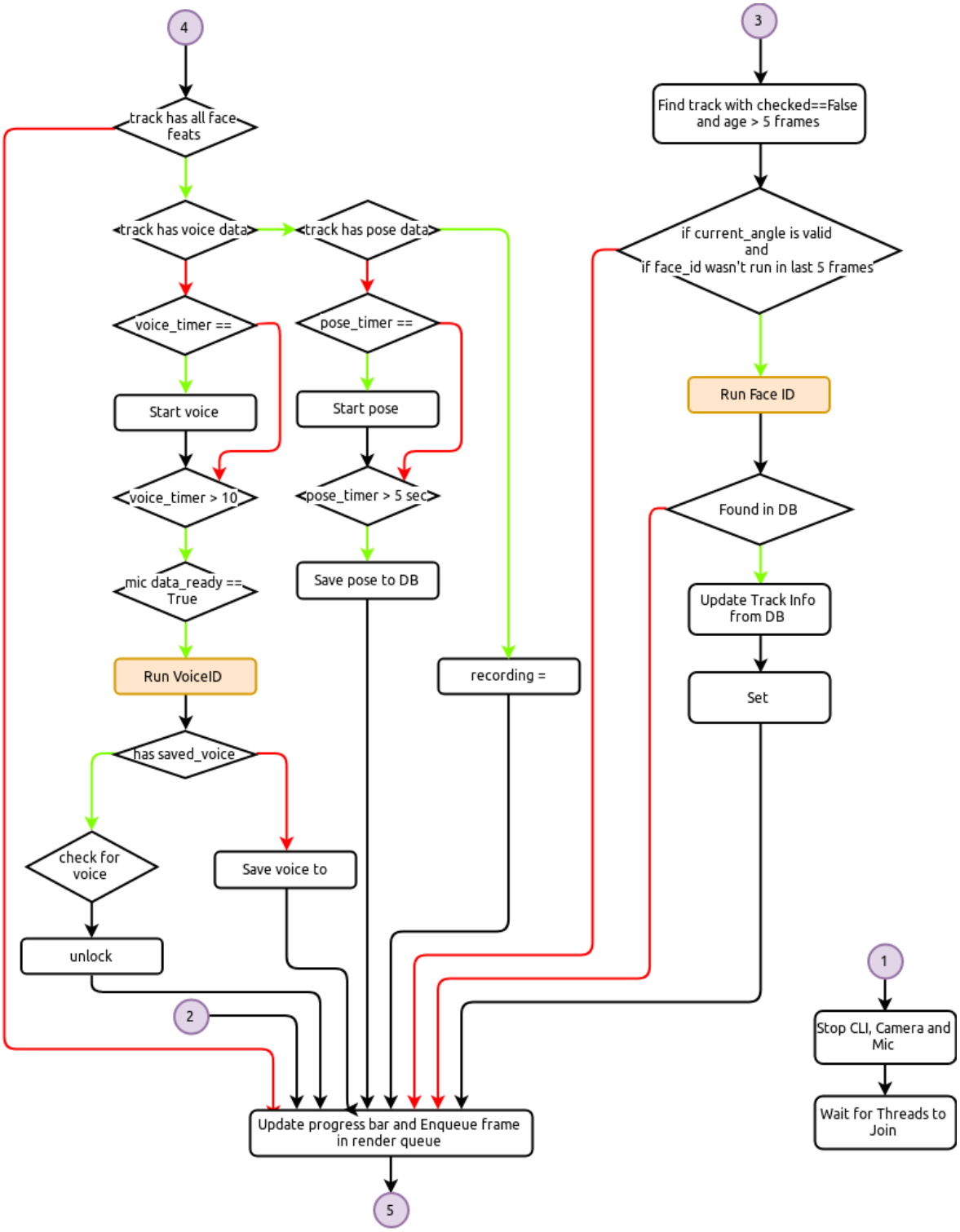
1. The overall application is made of 5 threads : Camera, Audio, Render, Command Line and Application
2. The application uses a set of numpy memmap files as DB. This DB is backed up every time the application is run (in case of any catastrophic failure).
3. Camera thread runs in a loop, grabs frame from the camera device, and runs the preprocessing steps on the frame. Application thread can acquire lock and grab latest frame whenever ready. This minimizes some of the camera read time and preprocessing time.
4. Audio thread runs in a loop, maintains a buffer of 5 sec audio data. When data is requested, stops recording audio, generates spectrogram and waits till the spectrogram is read by the application thread and the resumes populating the audio buffer.
5. Command Line Interface (CLI) thread waits for user to interact to with the terminal, processes the request (name user, list user or delete user) in a loop.
6. Render thread waits for the application thread to push frames and data, renders and displays them. It is assumed the render thread can finish one cycle faster than the application thread.

7. Application thread runs in a loop, reads frame from the camera, runs pose estimation, extracts bounding box from the pose estimation output and updates tracker state. Then proceeds to check if any of the tracks in the frame want to be recorded and if not, tries to identify people in the frame and assign them an id. If someone wants to be recorded, the application thread runs the recording process till either the user is done recording or leaves the frame.

Software Flow




Processing Thread
5
Is end != false
1
Get Most Recent Frame, RGB Frame, Pre-processed Frame for pose
Run Pose Network
Pose Network Post Processing
Get Faces from Pose
Update tracker with Face BBox
Match Bounding Boxes with Tracks
Update Track's Face Direction
Set Waved_State based on current and past
has saved_pose
current pose ==
pose timer ran &&
unlock
record
Update Kalman Filter with new BBox
simple_mode
2
recording !=
Make sure the recording track is alive. If not, clean up current recording
recording ==
Find Track with waved_state True exists
Set recording to
recording !=
3
if face_id needs to be run for current angle
and
Run Face ID
Complete Entry Match
Partial Entry Match
Update Database Entry
Create New Entry
Update Track Info from DB
4


4
track has all face feats
track has voice data
track has pose data
voice_timer ==
pose_timer ==
Start voice
Start pose
voice_timer > 10
pose_timer > 5 sec
mic data_ready == True
Save pose to DB
Run VoiceID
recording =
has saved_voice
check for voice
Save voice to
unlock
2
Update progress bar and Enqueue frame in render queue
5
3
Find track with checked==False and age > 5 frames
if current_angle is valid and if face_id wasn't run in last 5 frames
Run Face ID
Found in DB
Update Track Info from DB
Set
1
Stop CLI, Camera and Mic
Wait for Threads to Join