Keypoint Model for PFL - Inference and Similarity Matching

This system introduces a keypoint-based matching approach to streamline pose initialization by leveraging a precomputed pool of calibration images.

At system startup, we run a keypoint detection model over a set of calibration images to extract structured 2D keypoints associated with known 3D locations in the arena. When a live video frame is received, the same model is applied to detect its keypoints. These are then matched against the calibration pool to find the most geometrically similar calibration frame.

By using cosine distance (or a similar metric) for matching and ranking candidates, we can identify the best-fit calibration frame for pose estimation. This approach enables fast and accurate pose initialization, minimizing delays and instability caused by poor early-frame estimates.

📘 Instructions

1. Setting up the sources:
    a. Clone the repository and setup the environment. Conda is recommended

    ```
    git clone https://github.com/bastinj98/PFL-Keypoint-Model.git
    ```

    ```
    cd PFL-Keypoint-Model
    ```

    ```
    conda create -n ai-cv python=3.10
    conda activate ai-cv
    pip install -r requirements.txt
    ```

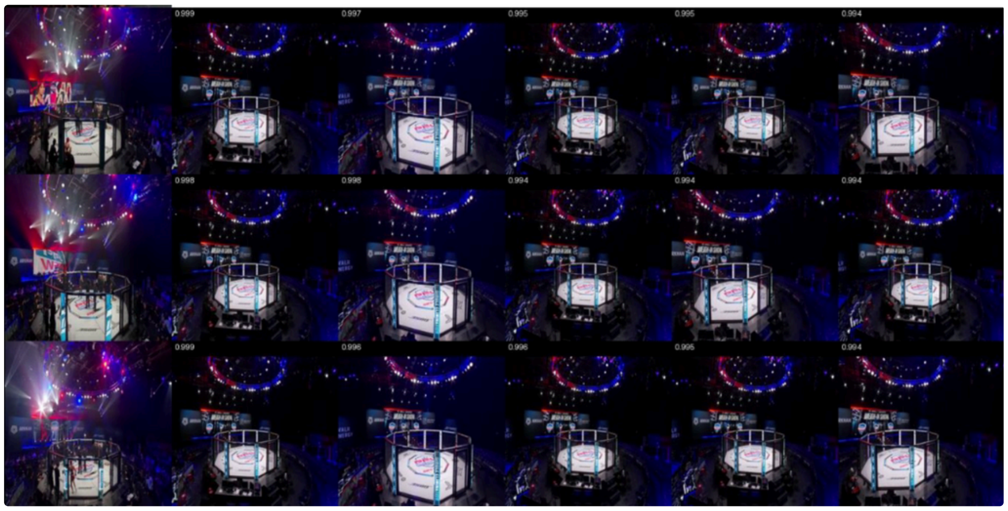
2. Assemble the input data: Download both the input video and calibration image folder and place them inside “PFL-Keypoint-Model/data” folder
    a. Input Video
    b. Calibration Image Set
3. Download the trained keypoint model weights: Both the onnx and pytorch models are available. in this tutorial, lets stick with the pytorch (.pt) model for python inference. Download the model and place it in “PFL-Keypoint-Model/models”
    a. Keypoint Model
    b. Onnx Model for (pipeline - not needed for this step)
4. Running the inference:

```
python src/pfl_match_keypoints.py \
  --calib_folder /home/bb/Dev/PFL-Keypoint-Model/data/calib_frames \
  --video_path /home/bb/Dev/PFL-Keypoint-Model/data/sample_input_video.mp4 \
  --num_frames 20 \
  --top_k 5
```

Argument	Description
`--calib_folder`	**(Required)** Path to the folder containing calibration images. These images are used to build the FAISS index for keypoint matching.
`--video_path`	**(Required)** Path to the input video from which frames will be sampled and matched against calibration images.

`--index_path`	Path to save or load the FAISS index. If the file doesn't exist, a new index will be created from the calibration folder. Default: `faiss_index/calib_index.faiss`
`--metadata_path`	Path to save or load associated metadata (e.g., frame filenames, descriptors). Default: `faiss_index/calib_meta.pkl`
`--output_sheet`	Path where the generated contact sheet (match visualization) will be saved. Default: `keypoint_contact_sheet.jpg`
`--num_frames`	Number of frames to extract from the input video for matching. Default: `20`
`--top_k`	Number of top matching calibration images to retrieve per video frame. Default: `5`

3. Result: A matchsheet that assembles the query frame and the top k maches will be generated and saved in “PFL-Keypoint-Model/outputs” folder



1.
2.

Highlight important information in a panel like this one. To edit this panel's color or style, select one of the options in the menu.

📋 C++ Pipeline (TRTer) *(under updation)*

The snapshot of Test.cpp shows the configuration of sources and models. All the links provided above works here as well.

```
int main()
{


    std::string inputVideoPath = "C:\\Users\\BB\\Videos\\PFL\\keypoint_test_footage_PFL.mp4";
    std::string calibImagesDir = "C:\\Users\\BB\\Videos\\PFL\\calib_frames";
    std::string onnxModelPath = "C:\\ProgramData\\Injecto\\Models\\ONNX\\KeyPoint_PFL_YOLOv8m_Pose_small.onnx";//argv[2];

```

```cpp
    std::vector<std::vector<float>> calibDescriptors;
    std::vector<std::string> calibFilenames;


    ProcessCalibFrames(calibImagesDir, onnxModelPath, calibDescriptors, calibFilenames);

    faiss::IndexFlatIP* faissIndex = nullptr;
    BuildFaissIndex(calibDescriptors, faissIndex);


    RunKeypointModel(inputVideoPath, onnxModelPath, faissIndex, calibFilenames);

    delete faissIndex;

    cOUT << "Processing complete (TensorRT only)";
    TGITech::Stacy::Get().YallaBye();
    return 0;
}
```

Filter by label

There are no items with the selected labels at this time.