THU SUMMER SCHOOL 2025

CHANTAL DEUSCH | JEREMY DIEM | JAN GASCHLER | SERHAT GÜREL | PAULINA PYCZOT | VALENTIN TALMON-L'ARMÉE

Image Detection with YOLO AI on a Raspberry Pi

Agenda

1. Setup Raspberry Pi
2. Licence Plate Recognition
3. Face Recognition

Visual Process Structure

Raspberry Pi

Taking Pictures

Detect Plate

Reading Plate

Live Stream

Detect Person and Face

Learn Face

Set Up Raspberry Pi

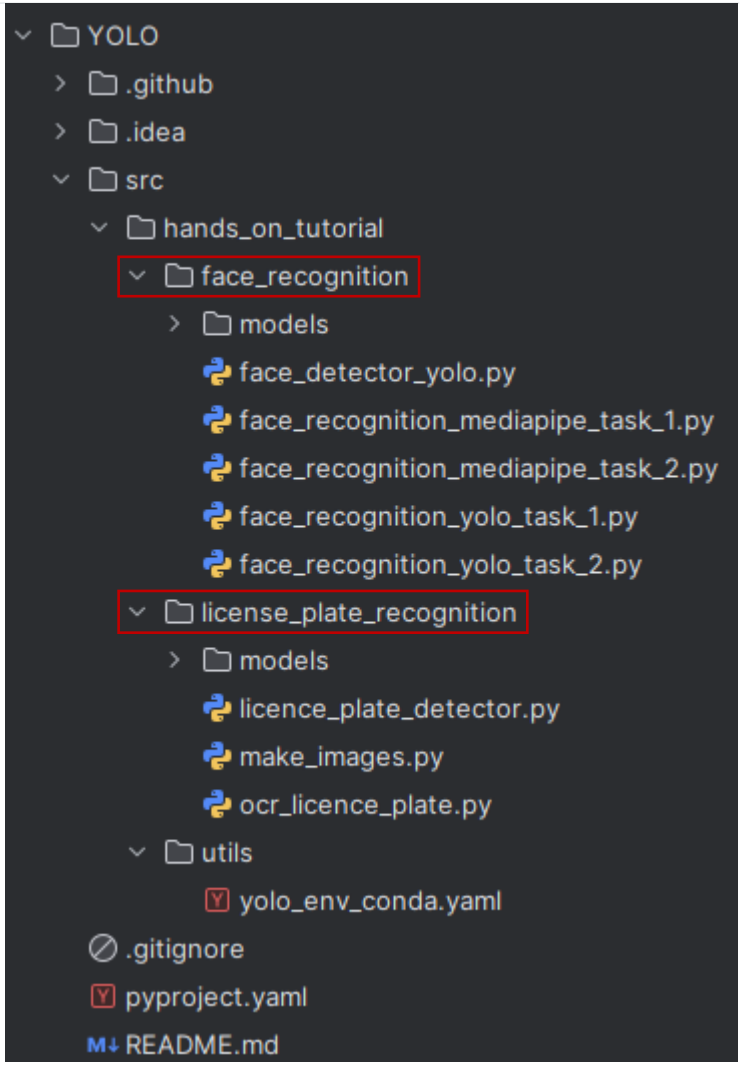
To set up the Raspberry Pi, you will need the following physical components:

- Pre-configured SD card
- Mouse
- Keyboard
- Monitor with HDMI support
- USB-compatible camera
- Power supply & Micro-HDMI to HDMI cable

Set Up Raspberry Pi

Setting Up the Project

1. Clone Git repository
   - *git clone https://github.com/TOPR-yoloteam/YOLO.git*
2. Navigate into the project folder
   - *cd ./YOLO*



Set Up Raspberry Pi

Setting Up the Conda Environment

Why do we need a virtual environment?

- Keeps dependencies organized
- Avoids version conflicts
- Isolates the project from global Python
- Makes setup easier and cleaner

📎 **Nice to know:**
To activate your conda environment, use
$ conda activate <env>
To deactivate an active environment, use
$ conda deactivate

Installation of Conda

1. Download the Miniconda installer and make it executable
   - *wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-aarch64.sh*
   - *chmod +x Miniconda3-latest-Linux-aarch64.sh*
2. Run the installer
   - *./Miniconda3-latest-Linux-aarch64.sh*
   - Press ENTER, scroll to the end of the EULA (= End User License Agreement)
   - Press *q* and type '*yes*' to accept the terms
   - Press ENTER to install Miniconda3 into */home/admin/miniconda3*
   - Type '*yes*' to activate conda on startup
3. Test it: Activate conda
   - *source ~/miniconda3/bin/activate*
   - You're now on your base environment!
   - Deactivate the environment with *conda deactivate*

Set Up Raspberry Pi

Installation of Libraries with .yml

Why .yml?

- Ensures consistent installation of required packages
- Makes it easy to recreate and share the exact environment setup
- All needed dependencies (general, license plate recognition & face recognition) in one place

Setting Up the virtual Conda Environment with .yml

1. Optional: Create a project folder and navigate into it
   - *mkdir <folder_name>*
   - *cd <folder_name>*
2. Create a new conda environment
   - *conda env create –f ./YOLO.yml*
3. Activate the environment
   - *conda activate yolo_env_conda*
   - The location of your environment is */home/admin/miniconda3/envs/yolo_env_conda*

hands_on_tutorial\...\yolo_env_conda.yaml

```yaml
name: yolo_env_conda
channels:
  - conda-forge
dependencies:
  # General libraries
  - python=3.9            # Python version
  - pip                   # Python package manager
  - numpy=1.26.4          # Numerical operations and array handling
  - matplotlib            # Visualization and plotting
  - opencv                # Computer vision library (image and video processing)
  - pygame                # Multimedia handling (e.g., simple UI, window management)

  # Special libraries for text, characters, and license plate recognition
  - pytesseract           # Python binding for Tesseract OCR
  - tesseract             # Tesseract OCR engine (backend for pytesseract)

  # Special libraries for face/person recognition
  - ultralytics           # YOLO models for object detection (including person detection)
  - dlib                  # Machine learning toolkit (especially for facial landmarks)
  - face_recognition      # High-level face detection and recognition built on dlib

  # pip-only packages
  - pip:
      - mediapipe         # Google framework for face/hand/pose detection
      - torch==1.10.0     # PyTorch framework (deep learning backend)
      - tensorflow==2.8.0 # TensorFlow framework (for models that require it)

# Create environment with: conda env create -f yolo_env_conda.yml
# Activate environment with: conda activate yolo_env_conda
```

If there should be a problem using the .yaml file you can setup your conda environment manually by using the installation file YOLO/src/hands_on_tutorial/utils/YOLO_man.txt

Setup Raspberry Pi

Setting Up the Python Development Environment

Thonny

1. A simple and beginner-friendly Python development environment
2. Pre-installed on Raspberry Pi OS
3. Built-in debugger

How to Start and Use Thonny

1. Type *thonny* in the Terminal
2. Use the file explorer to navigate to the Python file you want to edit by using the *Load* button
3. To run the script, click the green *Run* button at the top (or press F5 on your Keyboard)

License Plate Recognition

License Plate Recognition

Helpful tips

- To see changes better, use the following code after changing an image

```
cv2.imshow( winname: "image", image)
cv2.waitKey(0)
```

- Code to be written is marked with #TODO X in the source code

```
#TODO 1
```

- Google loves you, ChatGPT doesn't <3

License Plate Recognition

Taking Pictures

- To recognize license plates, we need images. These images must be captured first.
  We do this using the *Image* class in the *make_images.py* file.
  - Run the script to start capturing images from the webcam

License Plate Recognition

Detect License Plate

- In order to be able to read the text from license plates later, we must first recognize the license plate as such

- ***Task 1:*** **Modify the Python code to detect a license plate and draw its corresponding bounding box**

- *Hint:* The Ultralytics docs homepage provides information on how to use a YOLO model :')

License Plate Recognition

Detect License Plate

- To achieve the best OCR results, we now want to save the license plate as a single image

- ***Task 2:*** **Save the license plate as a single image in a seperate folder named "license_plates"**

- *Hint:* Use the plates bounding box coordinates!

License Plate Recognition

Read License Plate

- Colored images can cause problems in character recognition, therefore we need to preprocess the input image

- ***Task 1:*** **Preprocess Image on color, threshold, dilation and contours**

- *Hint:* Use the OpenCV library for image preprocessing. For contours use the given method "find_and_sort_contours"

License Plate Recognition

Read License Plate

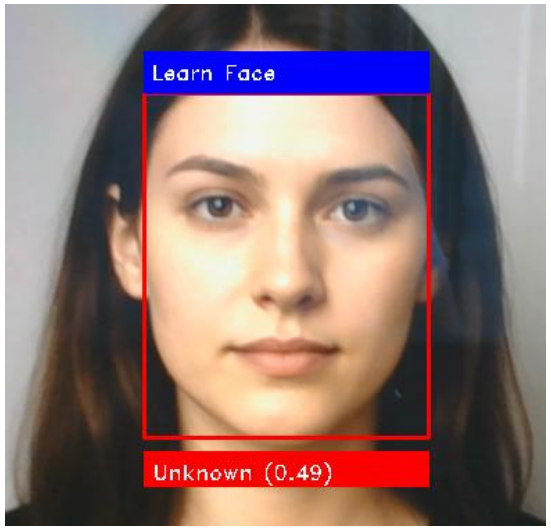
- If all steps are done correctly, start with the OCR!

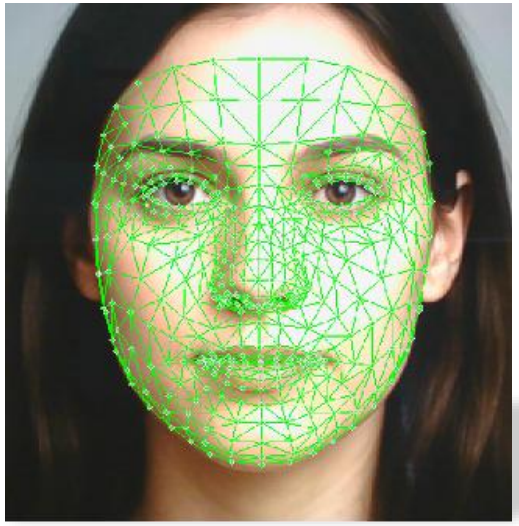
- ***Task 2:*** **Extract text from the contours and print the results**


```
Image: image_0_0.png
Text: ULDT805 | Probabilities: 86.33
```

- *Hint:* What does "extract_text_from_contours" return?

Face Recognition





Face Recognition

Please choose between:

- YOLO

› In this tutorial, we explore how the YOLOv8n-face model can be used to detect human faces in real time

› The YOLOv8n-face architecture is fine-tuned specifically for faces, combining high detection accuracy with a relatively small model size (only 1 object class)

› Although **YOLO** is highly optimized for GPU acceleration and real-time performance, it tends to lag more than **MediaPipe**, especially on resource-constrained devices like a Raspberry Pi

- MediaPipe

› A framework by Google providing lightweight models, particularly FaceMesh for detailed facial landmark detection.

› Often highly performant, even on resource-constrained devices like the Raspberry Pi.

› Focuses on extracting facial features (landmarks) which can then be used for recognition by comparing their relative positions.

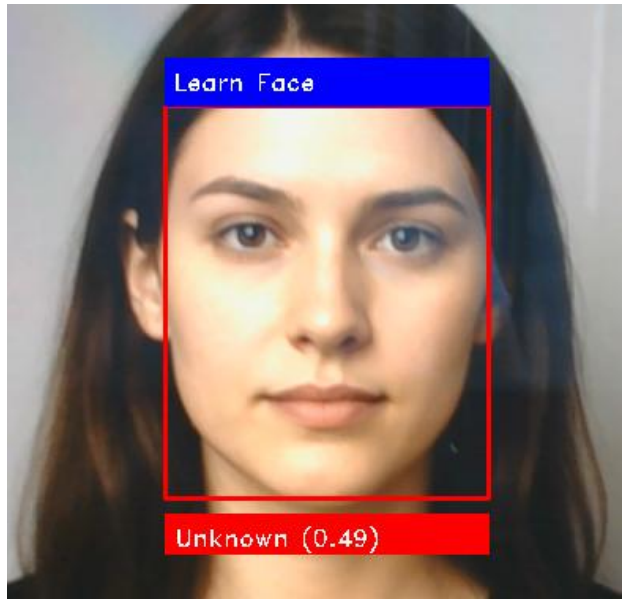
Face Recognition

If you want to solve the MediaPipe exercises, please skip to page 33

YOLO

Task 1:

- Implement **face detection** based on YOLO.
- Detect **faces** and draw bounding boxes around them.



Face Recognition

YOLO Task 1 tips

Task 1.1:

- Ensure the self.model_path correctly points to the "yolov8n-face.pt" file within the project structure.

Task 1.2:

- You run detection by calling the model object directly: results = self.model(...).
- Pass the video frame as the main argument to the model.
- **Crucial:** Use the classes=[0] argument to tell the model to *only* detect faces (which is class 0 in this specific face model).

Task 1.3:

- The results object contains detections. You often need to loop through results, then result.boxes, then each box.
- The coordinates are typically stored within a box object, often under the attribute .xyxy[0].
- The coordinates might be floating-point numbers. Use map(int, ...) to convert them to integers before drawing.
  - *Example:* x1, y1, x2, y2 = map(int, box.xyxy[0])

Face Recognition

YOLO Task 1 tips

Task 1.4:

- You can use OpenCV's drawing function: cv2.rectangle(...).
- Remember the required arguments: cv2.rectangle(image, pt1, pt2, color, thickness).
  - *pt1* is the top-left corner (x1, y1).
  - *pt2* is the bottom-right corner (x2, y2).
  - *Color* is a BGR tuple, e.g., (0, 255, 0) for green.
  - *Thickness* is an integer, e.g., 2.

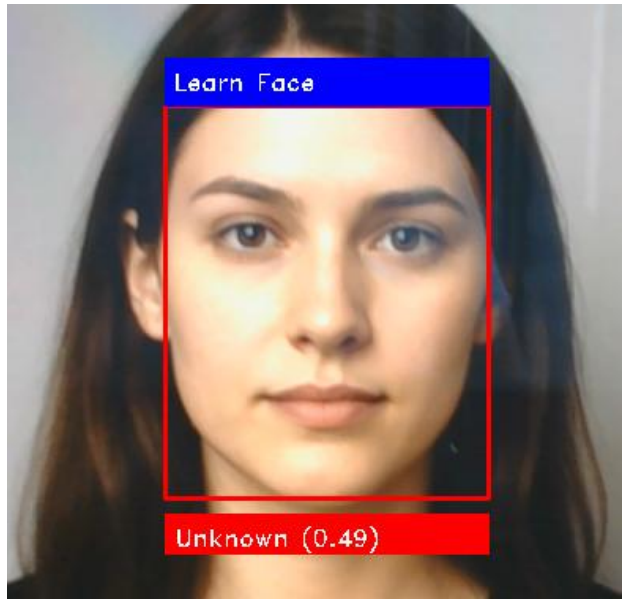
Face Recognition

If you want to solve the MediaPipe exercises, please skip to page 33

YOLO

Task 2:

- Extend the system by adding face recognition.
  Person detection is **already provided**.
  Implement recognizing known faces and mark them in the image.



Face Recognition

YOLO Task 2 tips

Task 2.1:

- Make sure you're reading image files from the correct directory (self.faces_dir).
- Use face_recognition.load_image_file() to load each image.
- Get the face encoding using face_recognition.face_encodings(loaded_image)[0].
  - Note the [0] assumes one face per known image file.
- Store the encoding in self.known_face_encodings and the name (from filename) in self.known_face_names. Keep these lists synchronized!

Task 2.2:

- Get the face_location passed to the save_face function.
- Crop the face from the stored self.current_frame using these coordinates: face_image = self.current_frame[top:bottom, left:right].
- Consider expanding the coordinates slightly before cropping to get more context, but be careful not to go outside the frame boundaries (max(0, ...), min(width, ...)).
- Save the face_image using cv2.imwrite() into self.faces_dir. Give it a filename based on the entered name.
- **Very Important:** After saving, immediately call self.load_known_faces() again so the system recognizes the newly added person right away.

Face Recognition

YOLO Task 2 tips

Task 2.3, 2.5., 2.6, 2.7:

- Use the self.state variable ("normal" or "entering_name") to control behavior.
- In mouse_callback: If state is "normal", check if the click (x, y) is inside any stored button area. If yes, change self.state to "entering_name" and store which face was clicked (self.selected_face_loc).
- In the main loop (run): If self.state is "entering_name", handle keyboard input (cv2.waitKey(1)):key == 13 (Enter): Save the face if self.current_text is not empty, then switch back to "normal" state.
  - key == 27 (Esc): Switch back to "normal" state.
  - key == 8 (Backspace): Remove last character from self.current_text.
  - Printable characters: Append chr(key) to self.current_text.
- Draw the text input UI (draw_text_input) and highlight the selected face only when in the "entering_name" state.

Face Recognition

YOLO Task 2 tips

Task 2.4:

- You need face locations and encodings for the *current* video frame first.
- Use face_recognition.face_locations() and face_recognition.face_encodings().
- **Remember:** Convert the OpenCV frame (BGR) to RGB before passing it to face_recognition functions: rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB).
- Use face_recognition.compare_faces(self.known_face_encodings, encoding_to_check) to get potential matches.
- Use face_recognition.face_distance(self.known_face_encodings, encoding_to_check) to find the distance for each comparison. Lower distance means a closer match.
- Find the best match using np.argmin(face_distances) on the distances array.
- Verify the best match: Check if matches[best_match_index] is True.

Face Recognition

YOLO Task 2 tips

Task 2.4:

- Draw the name label using cv2.putText().
- Draw the "Learn Face" button (using cv2.rectangle and cv2.putText) **only** if the face is "Unknown" and the system is in the "normal" state.
- For the button click to work later, store the button's screen coordinates *and* the corresponding face location (top, right, bottom, left) together (e.g., in self.button_area).

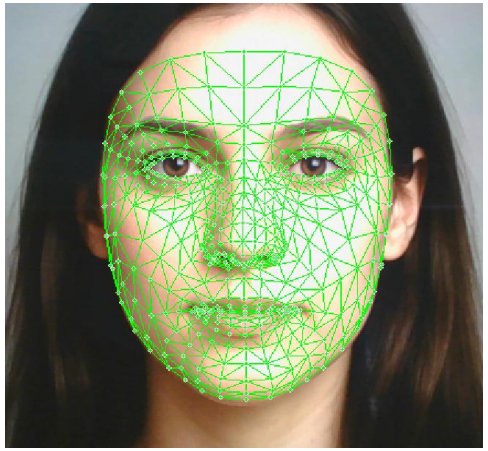
Face Recognition

If you want to solve the YOLO exercises, please go back to page 25

MediaPipe

Task 1:

- Implement **face detection** based on MediaPipe.
- Face recognition is not required in this task.
- Detect **faces** and draw bounding boxes around them.



Face Recognition

MediaPipe Task 1 tips

Task 1.1:

- Create the FaceMesh instance using mp.solutions.face_mesh.FaceMesh(...).
- You can set parameters like max_num_faces=... and min_detection_confidence=... during initialization (check MediaPipe documentation for details).

Task 1.2:

- **Important:** MediaPipe expects images in RGB format, but OpenCV captures in BGR. Convert the frame: image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB).
- Call the process() method on your self.face_mesh object, passing the image_rgb.

Task 1.3:

- Use OpenCV's cv2.rectangle(image, pt1, pt2, color, thickness).pt1 should be the top-left corner: (left, top).pt2 should be the bottom-right corner: (right, bottom).
- Choose a color (BGR tuple like (0, 255, 0) for green) and thickness.

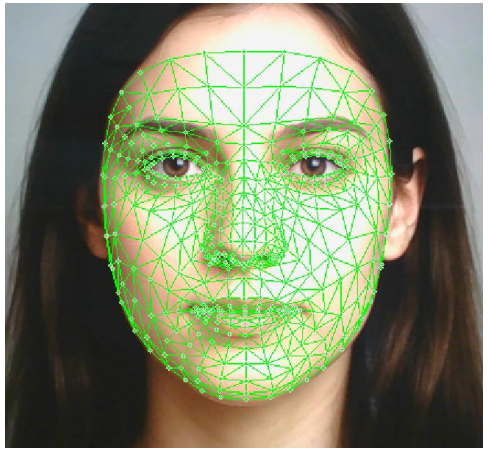
Face Recognition

If you want to solve the YOLO exercises, please go back to page 25

MediaPipe

Task 2:

- Extend the system by adding face recognition.
- Person detection is **already provided**.
- Implement recognizing known faces and mark them in the image.



Face Recognition

MediaPipe Task 2 tips

Task 2.1:

- You don't need all 400+ landmarks from FaceMesh. Use the provided key_landmarks_indices list to select a smaller, specific set (e.g., 50 landmarks).
- Loop through key_landmarks_indices. For each index idx, get the landmark data: landmark = face_landmarks.landmark[idx].
- Store the **normalized** coordinates landmark.x and landmark.y (values between 0.0 and 1.0).
- Create a flat NumPy array from these coordinates (e.g., [x1, y1, x2, y2, ..., x50, y50]). The size must be consistent!

Task 2.2:

- This happens in the compare_landmarks function.
- You need to compare the detected landmarks array to *every* stored landmark sample for *every* known person (for known_landmarks in person_landmarks_list:).
- A good way to measure similarity is the Euclidean distance: distance = np.linalg.norm(landmarks - known_landmarks).
- Keep track of the min_distance found and the best_match_index (which person it corresponds to).

Face Recognition

MediaPipe Task 2 tips

Task 2.3:

- After checking all known landmarks, take the overall min_distance.
- Compare it to self.recognition_threshold.
- If min_distance < self.recognition_threshold, it's a match! Set is_known_face = True and get the name using best_match_index.
- If the distance is larger, it's "Unknown" (is_known_face = False).
- Consider calculating a confidence score based on how far below the threshold the distance is.