

Motivation:

Face tracking cameras are valuable for increasing security in and around premises, aiding in accurate identification of individuals and detecting identity fraud. for example:

- Analyzing visitor at high security areas
- Customer behavior at market areas.

Objectives:

- Deployment on a human detection model on hardware
- Camera Level Control: Perform Focus Adjustment based on the position of human in camera frame.
- Real-time human movement tracking: Built a pan-tilt model using servo motor to track the detected human.

Dataset / Data collection:

- The model used is trained on WIDER FACE dataset, which is a large-scale face detection benchmark dataset.

Hardware and Software spec:

- Arduino IDE 2.3.2
- OpenMV IDE v4.1.5
- ESP32-CAM microcontroller
- Arduino NICLA VISION
- MG995 servo motor
- 7805 voltage regulator

Edge AI Model:

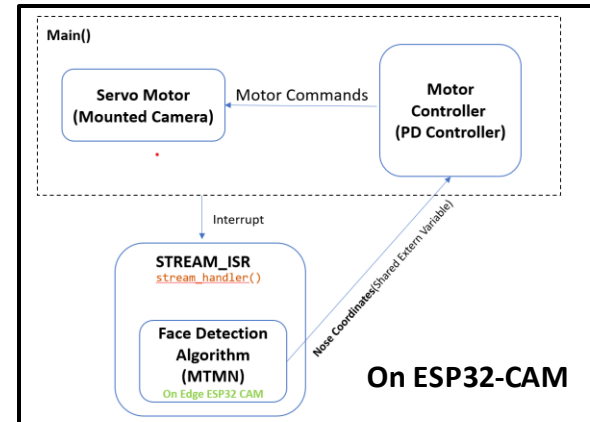
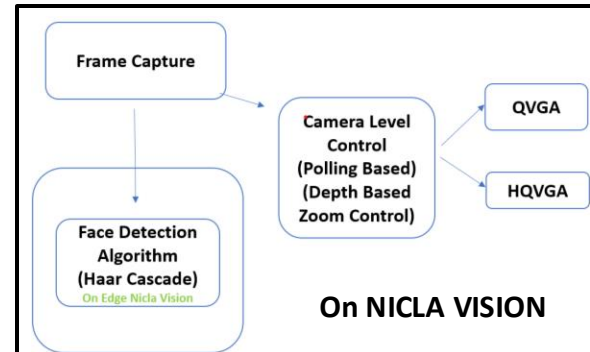
On NICLA VISION

- Face detection Model: Haar Cascade

On ESP32-CAM

- ML model: MTMN
 - MoblieNetV2 (MN)
 - Multi-task (MT) Cascaded Convolution Networks

Process Flow Diagram:

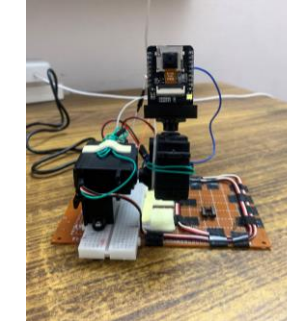


Prototype & demonstration:

On NICLA VISION



On ESP32-CAM



- ESP32-CAM
- Servo Motors MG995
- 3D Printed Casing to ESP 32 CAM on top of Servo Motor
- 7805 Voltage regulator to regulate voltage on the hardware.

Performance Evaluation:

On NICLA VISION

	HQVGA (fps)	QVGA (fps)
Without Detection	21	9
With Detection	19	7

Memory Utilization
Total \approx 182 KB
Used \approx 150 KB
Free \approx 20 KB

ON ESP32-CAM

	HQVGA (fps)
Without Detection	25
With Detection	8

Memory Utilization
Heap space \approx 145 KB
RAM \approx 520 KB

Outline:

- ❑ Hardware Setup and Component Selection
- ❑ Process Flow Diagram
- ❑ DL based Human Detection and gesture control Algorithm Implementation
- ❑ Integration of Human Detection with Servo Control using PID Controller.
- ❑ Performance Evaluation

Hardware Setup & Component Selection:

- ❑ We have used **Nicla Vision** to demonstrate **Camera Level Control** and **Face Detection** Capabilities.
- ❑ We have used the **ESP32 CAM**, to demonstrate AI based **Face Tracking Camera** and **Servo control**.
- ❑ We have used 2 Servo Motor : **MG 995** for the purpose of Camera Movement
- ❑ 3D Printed Casing for mounting ESP 32 CAM on top of Servo Motor.

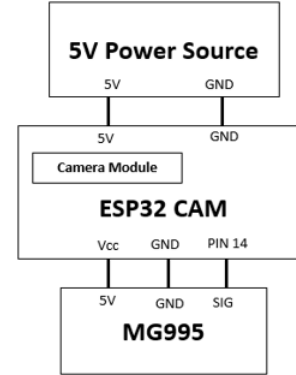


Fig 1: Pin Diagram
ESP32 CAM

Power Consumption:

1. Voltage: 5V
2. Current: 0.2 A
3. Power: 1 Watt



Fig 2: Nicla Vision Setup

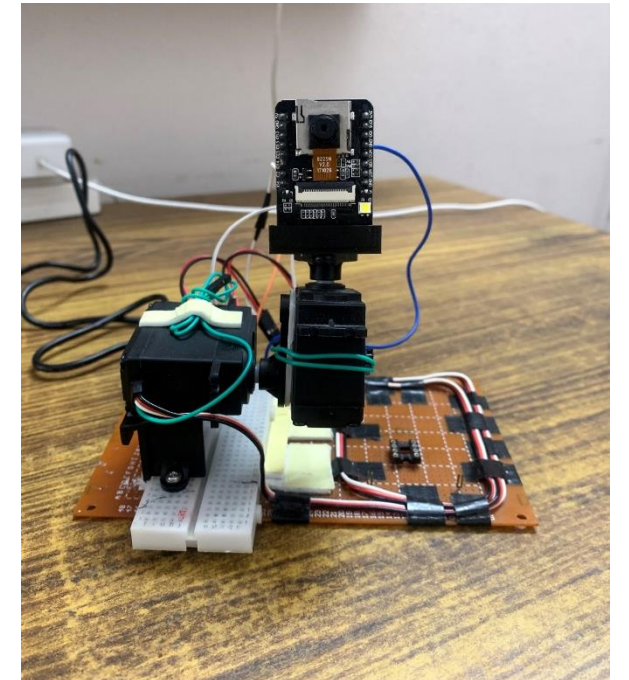


Fig 3: ESP32 CAM Setup

Process Flow Diagram (Nicla Vision):

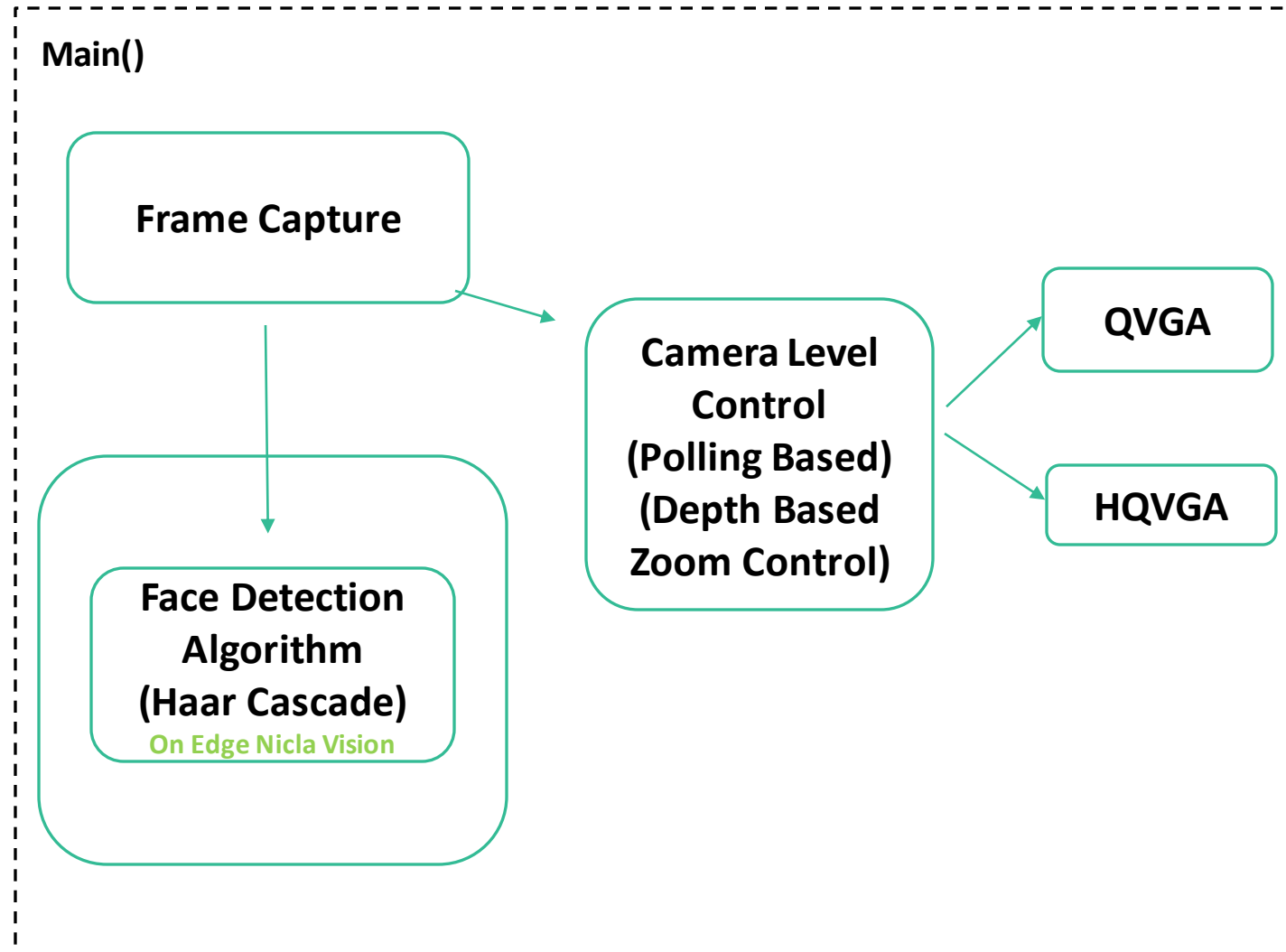


Fig : Process Flow Diagram for camera level control (depth zoom) on Nicla Vision

API Stack -Nicla Vision (Open MV)

Camera Level Zoom

```
if len(objects)!=0:
    center_x,center_y= str(objects[0][0] + objects[0][2]//2), str(objects[0][1] + objects[0][3]//2)
    area_face=objects[0][2]*objects[0][3]

    print("Face Center=", center_x+', '+center_y, "| Face area:",objects[0][2], "*", objects[0][3], "=", area_face)

    if (area_face) > referenceFaceArea:
        | sensor.set_framesize(sensor.QVGA)
    else:
        | sensor.set_framesize(sensor.HQVGA)
```

FACE DETECTION

```
face_cascade = image.HaarCascade("frontalface")
```

Performance Evaluation (Nicla Vision):

Time Metric:

	HQVGA (fps)	QVGA (fps)
Without Detection	21	9
With Detection	19	7

Memory Utilization:

QVGA Memory Utilization :

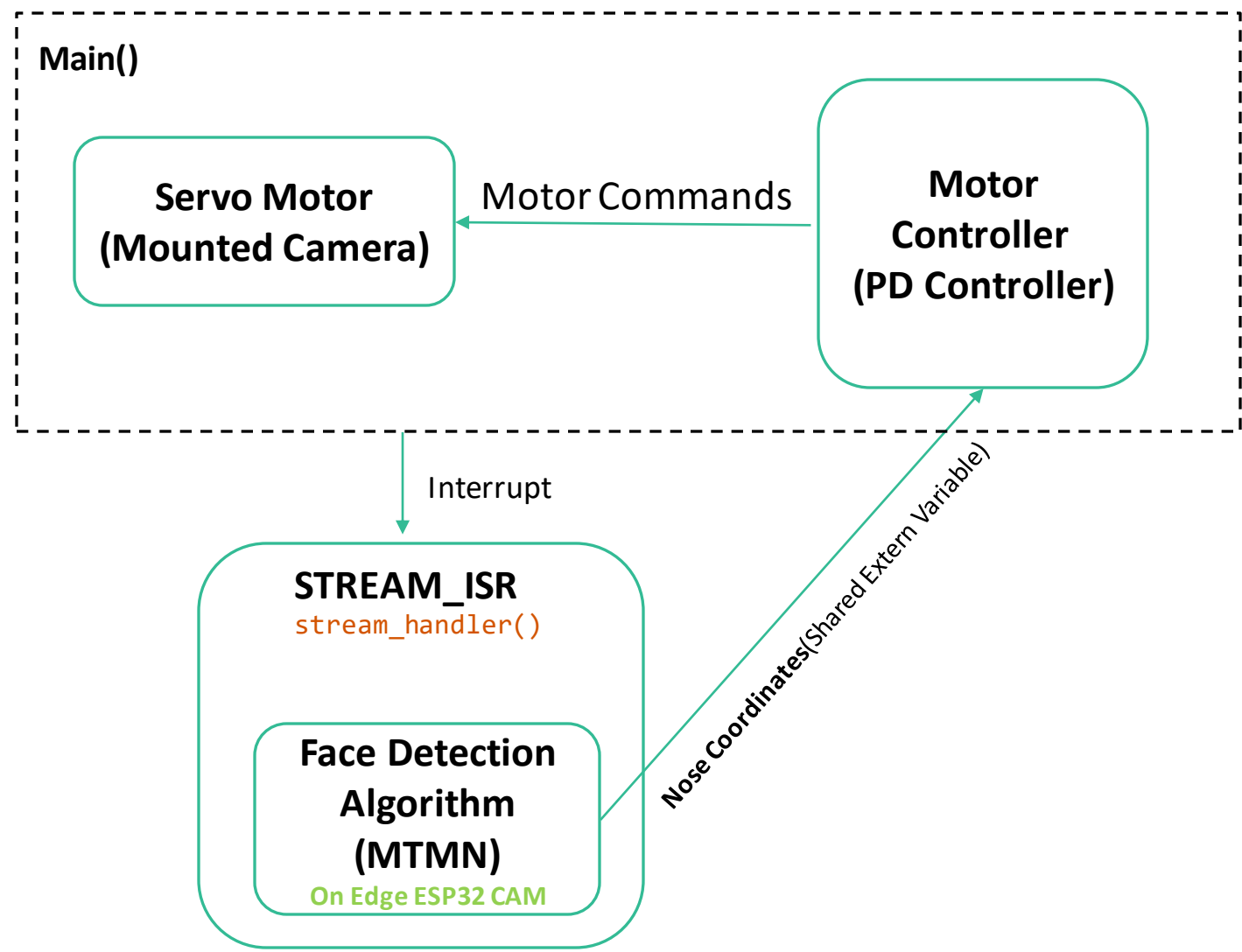
Used – 150 Kb

Free – 20 Kb

Total – 18.2Kb

Nicla Vision Total RAM - 1Mb

Process Flow Diagram (ESP32 CAM):



```
detected = false;
face_id = 0;
fb = esp_camera_fb_get();
if (!fb) {
```

FACE DETECTION

```
fr_ready = esp_timer_get_time();
box_array_t *net_boxes = NULL;
if(detection_enabled){
    net_boxes = face_detect(image_matrix, &mtmn_confidence);
}
```

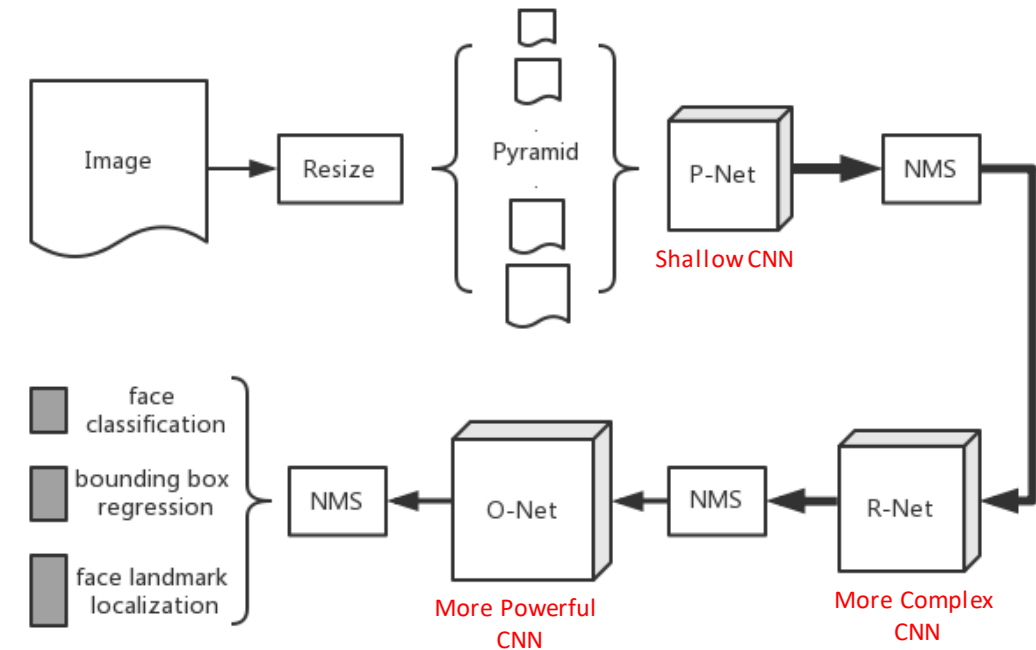
Face Detection Algorithm : MTMN

Overview

- MTMN is a lightweight **Human Face Detection Model**, which is built around [a new mobile architecture called MobileNetV2](#) and [Multi-task Cascaded Convolutional Networks](#), and is specially designed for embedded devices.

MTMN consists of three main parts:

1. Proposal Network (P-Net): Proposes candidate bounding boxes, and sends them to the R-Net;
2. Refine Network (R-Net): Screens the bounding boxes from P-Net;
3. Output Network (O-Net): Outputs the final results, i.e. the accurate bounding box, confidence coefficient and 5-point-landmark.



Face Detection Algorithm : MTMN

Training Dataset:

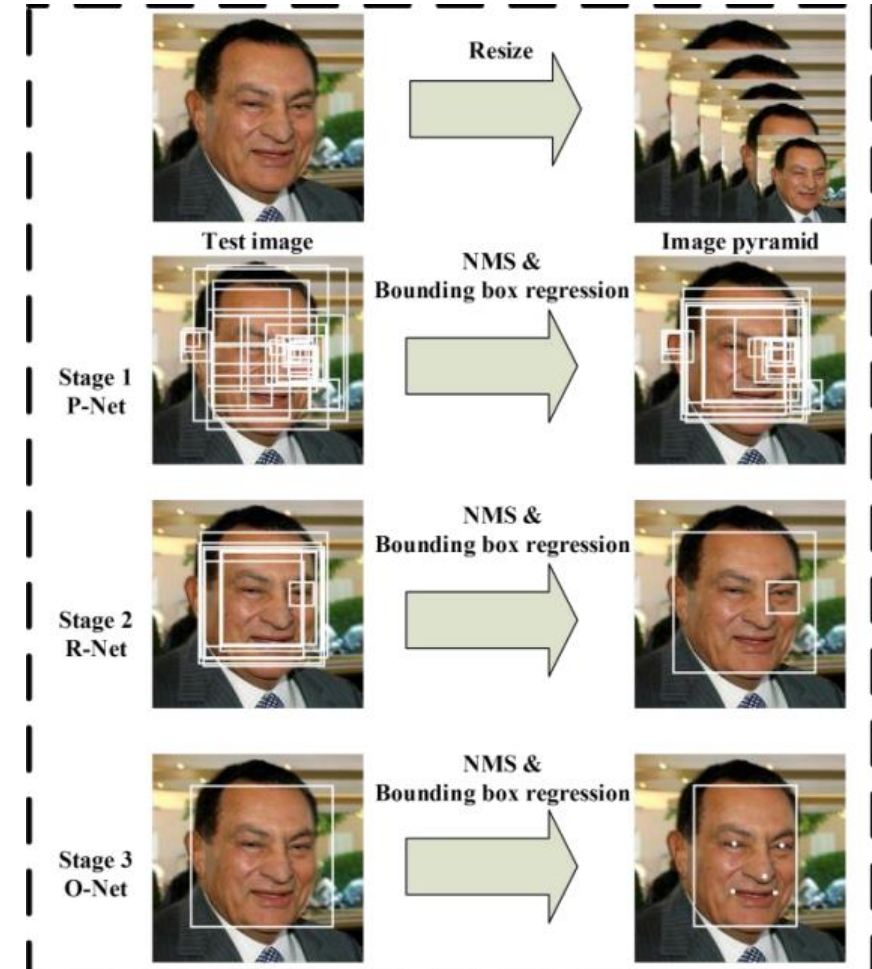
1. WIDER FACE Dataset: collect positives, negatives and part face(Bounding Boxes)
2. CelebA : Landmark faces are detected.

Training tasks to train our CNN detectors: face/non-face classification, bounding box regression, and facial landmark localization.

- 1) **Face classification:** The learning objective is formulated as a two-class classification problem.
- 2) **Bounding box regression:** For each candidate window, we predict the offset between it and the nearest ground truth (i.e., the bounding boxes' left top, height, and width).
- 3) **Facial landmark localization:** Similar to the bounding box regression task, facial landmark detection is formulated as a regression problem.

References :

1. S. Yang, P. Luo, C. C. Loy, and X. Tang, "WIDER FACE: A Face Detection Benchmark". arXiv preprint arXiv:1511.06523.
2. Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in IEEE International Conference on Computer Vision, 2015, pp. 3730-3738.



API Stack (ESP32 CAM)

```
box_array_t *face_detect(dl_matrix3du_t *image_matrix, mtmn_config_t *config);
```

This face_detect() handles the whole face-detection.

The inputs are:

image_matrix: an image in dl_matrix3du_t type

config: the configuration of MTMN.

The output is:

A box_array_t type value contains face boxes, as well as the score and landmark of each box.

This structure is defined as follows:

```
typedef struct tag_box_list
```

```
{
```

```
    fptp_t *score;
```

```
    box_t *box;
```

```
    landmark_t *landmark;
```

```
    int len;
```

```
} box_array_t;
```

The structure contains heads of arrays; each array has the same length,

which is the number of faces in the image.

Configuration Parameters

The minimum size of a detectable face

The scale of the gradient scaling for the input images.

The thresholds for P-Net.

The thresholds for R-Net.

The thresholds for O-Net.

Thresholds

1. **Score**: The threshold of confidence coefficient. The candidate bounding boxes with a confidence coefficient lower than the threshold will be filtered out.

2. **NMS**: The threshold of NMS. During the Non-Maximum Suppression, the candidate bounding boxes with an overlapping ratio higher than the threshold will be filtered out.

3. **Candidate Number**: The maximum number of allowed candidate bounding boxes. Only the first candidate_number of all the candidate bounding boxes will be kept.

Quantization Configurations

- Float Configuration (CONFIG_MTMN_LITE_FLOAT):
 - This is the configuration for the floating-point version of the neural networks.
 - Functions like pnet_lite_f, rnet_lite_f, and onet_lite_f are called when this configuration is active.
 - Floating-point weights and activations are used during the forward pass.
- Quantized Configuration (CONFIG_MTMN_LITE_QUANT):
 - Functions pnet_lite_q, rnet_lite_q, and onet_lite_q are called when quantization is active.
 - These functions involve quantized weights and activations. Quantized operations are more hardware-friendly.
- The DL_XTENSA_IMPL argument - quantized operations are implemented for the Xtensa architecture, which is used in ESPRESSIF chips.

Average Time Consumption (ms)

1. MTMN lite in quantization: 56.8
2. MTMN lite in float: 73.84

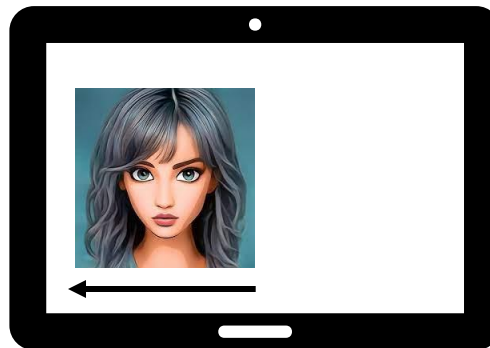
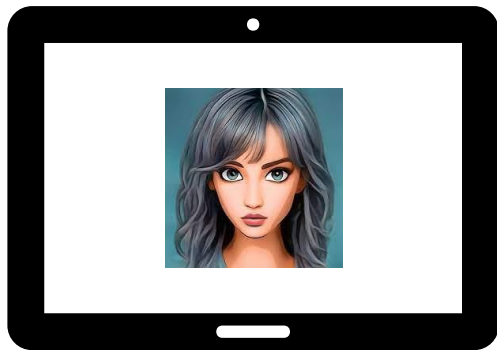
Speedup Achieved: x1.30

```
#if CONFIG_MTMN_LITE_FLOAT
    out = pnet_lite_f(in);
#endif

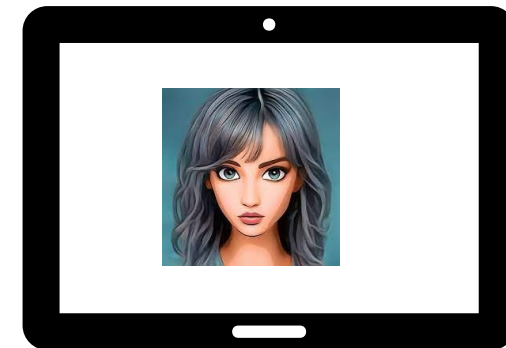
#if CONFIG_MTMN_LITE_QUANT
    ⚡ out = pnet_lite_q(in, DL_XTENSA_IMPL);
#endif
```

Servo Motor Control:

- ❑ Servo Motor Control will be achieved by writing angles directly to the motor commands.
- ❑ The motor needs to rotate in such a manner so as to track the movement of the human face within visibility area.



Face Movement



Servo Movement
To Align Face at Frame Centre

Integration of Face Detection with Servo Control using PID Controller:

- ❑ We used a controller to track the movement of the face, by calculating the error at each instant and using that error in our image resolution.
- ❑ $\text{Angle} = \text{Delta Angle} + \text{Angle}$
- ❑ $\text{error_x} = (\text{X res}/2) - \text{X Nose}$
- ❑ $\text{error_y} = (\text{Y res}/2) - \text{Y Nose}$: (Where X Res = 80, Yres=60 for a 160*120 frame size)
- ❑ $\text{delangle_y} = K_p * \text{error_y} + K_i * \text{integral_y} + K_d * (\text{error_y} - \text{prevError_y})$
- ❑ $\text{delangle_x} = K_p * \text{error_x} + K_i * \text{integral_x} + K_d * (\text{error_x} - \text{prevError_x})$
- ❑ Angle to be written to the servo is absolute angle, therefore the previous state angle has to be added to the delta angle obtained and then passed on to the servo command.

Integration of Face Detection with Servo Control using PID Controller:

Proportional Gain (K_p):

- Increases control signal proportionally to error.
- Quickens system response but may cause overshooting.

Derivative Gain (K_d):

- Adds anticipation by responding to the rate of error change.
- Dampens system, reducing overshoot.

Integral Gain (K_i):

- Helps reduce steady-state error by accumulating persistent error.
- May make the system sluggish and oscillatory.

Performance Evaluation:

Time Metric:

1. Without Detection

09:30:05.351	->	MJPG:	4000B	33ms	(30.3fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.398	->	MJPG:	3297B	48ms	(20.8fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.438	->	MJPG:	3299B	31ms	(32.3fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.476	->	MJPG:	3291B	39ms	(25.6fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.518	->	MJPG:	3277B	47ms	(21.3fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.555	->	MJPG:	4000B	32ms	(31.2fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.586	->	MJPG:	3262B	51ms	(19.6fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0
09:30:05.634	->	MJPG:	3256B	29ms	(34.5fps),	AVG:	39ms	(25.6fps),	0+0+0+0=0	0

avg_frame_time

avg_fps

2. With Detection

09:31:32.666	->	MJPG:	4899B	118ms	(8.5fps),	AVG:	119ms	(8.4fps),	22+57+0+34=114	DETECTED	0
09:31:32.745	->	MJPG:	4841B	120ms	(8.3fps),	AVG:	119ms	(8.4fps),	22+58+0+33=114	DETECTED	0
09:31:32.899	->	MJPG:	4828B	123ms	(8.1fps),	AVG:	119ms	(8.4fps),	23+53+0+39=116	DETECTED	0
09:31:33.025	->	MJPG:	4820B	117ms	(8.5fps),	AVG:	119ms	(8.4fps),	22+54+0+34=112	DETECTED	0
09:31:33.103	->	MJPG:	4836B	118ms	(8.5fps),	AVG:	119ms	(8.4fps),	23+55+0+34=113	DETECTED	0
09:31:33.260	->	MJPG:	4819B	118ms	(8.5fps),	AVG:	119ms	(8.4fps),	21+56+0+36=114	DETECTED	0
09:31:33.339	->	MJPG:	4837B	117ms	(8.5fps),	AVG:	119ms	(8.4fps),	22+55+0+34=112	DETECTED	0
09:31:33.464	->	MJPG:	4852B	122ms	(8.2fps),	AVG:	119ms	(8.4fps),	23+55+0+35=114	DETECTED	0

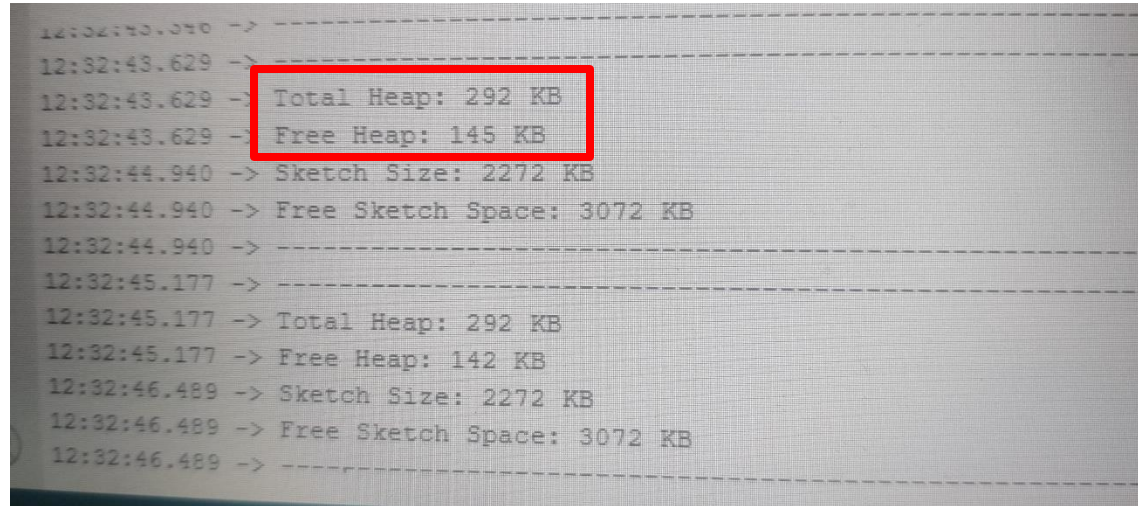
Time Metrics

1. `ready_time`: Time from the start to the point where the system is ready for face detection or recognition.
2. `face_time`: Time taken for face detection.
3. `recognize_time`: Time taken for face recognition.
4. `encode_time`: Time taken to encode the frame into a JPEG format.
5. `process_time`: Total time for the entire frame processing, including preparation, detection, recognition, and encoding.

ready_time, face_time, recognize_time, encode_time, process_time

Performance Evaluation:

Space and Memory Usage Metric:



```
12:32:43.040 -> -----
12:32:43.629 -> -----
12:32:43.629 -> Total Heap: 292 KB
12:32:43.629 -> Free Heap: 145 KB
12:32:44.940 -> Sketch Size: 2272 KB
12:32:44.940 -> Free Sketch Space: 3072 KB
12:32:44.940 -> -----
12:32:45.177 -> -----
12:32:45.177 -> Total Heap: 292 KB
12:32:45.177 -> Free Heap: 142 KB
12:32:46.489 -> Sketch Size: 2272 KB
12:32:46.489 -> Free Sketch Space: 3072 KB
12:32:46.489 -> -----
```

- With face detection model running the Heap Space used is: 145 KB (49.65% Utilization)
- ESP 32 Total RAM = 520Kb

Thank You