ДОДАТОК А

Код програми

Лістинг А.1 – PARE, demo.py

```
import os
os.environ['PYOPENGL_PLATFORM'] = 'egl'

import sys
import cv2
import time
import joblib
import argparse
from loguru import logger

sys.path.append('.')
from pare.core.tester import PARETester
from pare.utils.demo_utils import (
    download_youtube_clip,
    video_to_images,
    images_to_video,
)

CFG = 'data/pare/checkpoints/pare_w_3dpw_config.yaml'
CKPT = 'data/pare/checkpoints/pare_w_3dpw_checkpoint.ckpt'
MIN_NUM_FRAMES = 0


def main(args):

    demo_mode = args.mode

    if demo_mode == 'video':
        video_file = args.vid_file

        # ========= [Optional] download the youtube video ========= #
        if video_file.startswith('https://www.youtube.com'):
            logger.info(f'Downloading YouTube video
\"{video_file}\"')
            video_file = download_youtube_clip(video_file, '/tmp')

            if video_file is None:
                exit('Youtube url is not valid!')
```

```
            logger.info(f'YouTube Video has been downloaded to
{video_file}...')

        if not os.path.isfile(video_file):
            exit(f'Input video \"{video_file}\" does not exist!')

        output_path = os.path.join(args.output_folder,
os.path.basename(video_file).replace('.mp4', '_' + args.exp))
        os.makedirs(output_path, exist_ok=True)

        if os.path.isdir(os.path.join(output_path, 'tmp_images')):
            input_image_folder = os.path.join(output_path,
'tmp_images')
            logger.info(f'Frames are already extracted in
\"{input_image_folder}\"')
            num_frames = len(os.listdir(input_image_folder))
            img_shape = cv2.imread(os.path.join(input_image_folder,
'000001.png')).shape
        else:
            input_image_folder, num_frames, img_shape =
video_to_images(
                video_file,
                img_folder=os.path.join(output_path, 'tmp_images'),
                return_info=True
            )
        output_img_folder = f'{input_image_folder}_output'
        os.makedirs(output_img_folder, exist_ok=True)
    elif demo_mode == 'folder':
        args.tracker_batch_size = 1
        input_image_folder = args.image_folder
        output_path = os.path.join(args.output_folder,
input_image_folder.rstrip('/').split('/')[-1] + '_' + args.exp)
        os.makedirs(output_path, exist_ok=True)

        output_img_folder = os.path.join(output_path, 'pare_results')
        os.makedirs(output_img_folder, exist_ok=True)

        num_frames = len(os.listdir(input_image_folder))
    elif demo_mode == 'webcam':
        logger.error('Webcam demo is not implemented!..')
        raise NotImplementedError
    else:
        raise ValueError(f'{demo_mode} is not a valid demo mode.')

    logger.add(
        os.path.join(output_path, 'demo.log'),
        level='INFO',
        colorize=False,
    )
    logger.info(f'Demo options: \n {args}')
```

```
    tester = PARETester(args)

    total_time = time.time()
    if args.mode == 'video':
        logger.info(f'Input video number of frames {num_frames}')
        orig_height, orig_width = img_shape[:2]
        total_time = time.time()
        tracking_results = tester.run_tracking(video_file,
input_image_folder)
        pare_time = time.time()
        pare_results = tester.run_on_video(tracking_results,
input_image_folder, orig_width, orig_height)
        end = time.time()

        fps = num_frames / (end - pare_time)

        del tester.model

        logger.info(f'PARE FPS: {fps:.2f}')
        total_time = time.time() - total_time
        logger.info(f'Total time spent: {total_time:.2f} seconds
(including model loading time).')
        logger.info(f'Total FPS (including model loading time):
{num_frames / total_time:.2f}.')

        if not args.no_save:
            logger.info(f'Saving output results to
\"{os.path.join(output_path, "pare_output.pkl")}\".')
            joblib.dump(pare_results, os.path.join(output_path,
"pare_output.pkl"))

        if not args.no_render:
            tester.render_results(pare_results, input_image_folder,
output_img_folder, output_path,
                                  orig_width, orig_height,
num_frames)

            # ========= Save rendered video ========= #
            vid_name = os.path.basename(video_file)
            save_name = f'{vid_name.replace(".mp4",
"")}_{args.exp}_result.mp4'
            save_name = os.path.join(output_path, save_name)
            logger.info(f'Saving result video to {save_name}')
            images_to_video(img_folder=output_img_folder,
output_vid_file=save_name)

            # Save the input video as well
            images_to_video(img_folder=input_image_folder,
output_vid_file=os.path.join(output_path, vid_name))
```

```
            # shutil.rmtree(output_img_folder)

        # shutil.rmtree(image_folder)
    elif args.mode == 'folder':
        logger.info(f'Number of input frames {num_frames}')

        total_time = time.time()
        detections = tester.run_detector(input_image_folder)
        pare_time = time.time()
        tester.run_on_image_folder(input_image_folder, detections,
output_path, output_img_folder,
                                   run_smplify=args.smplify)
        end = time.time()

        fps = num_frames / (end - pare_time)


        del tester.model

        logger.info(f'PARE FPS: {fps:.2f}')
        total_time = time.time() - total_time
        logger.info(f'Total time spent: {total_time:.2f} seconds
(including model loading time).')
        logger.info(f'Total FPS (including model loading time):
{num_frames / total_time:.2f}.')


    logger.info('================ END ================')


if __name__ == '__main__':
    parser = argparse.ArgumentParser()

    parser.add_argument('--cfg', type=str, default=CFG,
                        help='config file that defines model
hyperparams')

    parser.add_argument('--ckpt', type=str, default=CKPT,
                        help='checkpoint path')

    parser.add_argument('--exp', type=str, default='',
                        help='short description of the experiment')

    parser.add_argument('--mode', default='video', choices=['video',
'folder', 'webcam'],
                        help='Demo type')

    parser.add_argument('--vid_file', type=str,
                        help='input video path or youtube link')
```

```
    parser.add_argument('--image_folder', type=str,
                        help='input image folder')


    parser.add_argument('--output_folder', type=str,
default='logs/demo/demo_results',
                        help='output folder to write results')

    parser.add_argument('--tracking_method', type=str,
default='bbox', choices=['bbox', 'pose'],
                        help='tracking method to calculate the
tracklet of a subject from the input video')

    parser.add_argument('--detector', type=str, default='yolo',
choices=['yolo', 'maskrcnn'],
                        help='object detector to be used for bbox
tracking')

    parser.add_argument('--yolo_img_size', type=int, default=416,
                        help='input image size for yolo detector')

    parser.add_argument('--tracker_batch_size', type=int, default=12,
                        help='batch size of object detector used for
bbox tracking')

    parser.add_argument('--staf_dir', type=str,
default='/home/mkocabas/developments/openposetrack',
                        help='path to directory STAF pose tracking
method installed.')

    parser.add_argument('--batch_size', type=int, default=16,
                        help='batch size of PARE')

    parser.add_argument('--display', action='store_true',
                        help='visualize the results of each step
during demo')

    parser.add_argument('--smooth', action='store_true',
                        help='smooth the results to prevent jitter')

    parser.add_argument('--min_cutoff', type=float, default=0.004,
                        help='one euro filter min cutoff. '
                             'Decreasing the minimum cutoff frequency
decreases slow speed jitter')

    parser.add_argument('--beta', type=float, default=1.0,
                        help='one euro filter beta. '
                             'Increasing the speed coefficient(beta)
decreases speed lag.')
```

```
    parser.add_argument('--no_render', action='store_true',
                        help='disable final rendering of output
video.')

    parser.add_argument('--no_save', action='store_true',
                        help='disable final save of output results.')

    parser.add_argument('--wireframe', action='store_true',
                        help='render all meshes as wireframes.')

    parser.add_argument('--sideview', action='store_true',
                        help='render meshes from alternate
viewpoint.')

    parser.add_argument('--draw_keypoints', action='store_true',
                        help='draw 2d keypoints on rendered image.')

    parser.add_argument('--save_obj', action='store_true',
                        help='save results as .obj files.')

    parser.add_argument('--smplify', action='store_true',
                        help='run MMPose and smplify to refine poses
further')

    args = parser.parse_args()

    main(args)
```

Лістинг А.2 – MediaPipe, main.js

```
import DeviceDetector from "https://cdn.skypack.dev/device-detector-
js@2.2.10";

// Usage: testSupport({client?: string, os?: string}[])
// Client and os are regular expressions.
// See: https://cdn.jsdelivr.net/npm/device-detector-
js@2.2.10/README.md for
// legal values for client and os
testSupport([
  {client: 'Chrome'},
]);

function testSupport(supportedDevices:{client?: string; os?:
string;}[]) {
  const deviceDetector = new DeviceDetector();
  const detectedDevice = deviceDetector.parse(navigator.userAgent);
```

```
  let isSupported = false;
  for (const device of supportedDevices) {
    if (device.client !== undefined) {
      const re = new RegExp(`^${device.client}$`);
      if (!re.test(detectedDevice.client.name)) {
        continue;
      }
    }
    if (device.os !== undefined) {
      const re = new RegExp(`^${device.os}$`);
      if (!re.test(detectedDevice.os.name)) {
        continue;
      }
    }
    isSupported = true;
    break;
  }
  if (!isSupported) {
    alert(`This demo, running on
${detectedDevice.client.name}/${detectedDevice.os.name}, ` +
          `is not well supported at this time, continue at your own
risk.`);
  }
}

const controls = window;
const drawingUtils = window;
const mpObjectron = window;

const config = {locateFile: (file) => {
  return
`https://cdn.jsdelivr.net/npm/@mediapipe/objectron@0.3.1627447724/${f
ile}`;
}};

const examples = {
  images: [
    {name: 'Camera image', src:
'https://assets.codepen.io/5409376/camera.jpg'},
    {name: 'Chair image', src:
'https://assets.codepen.io/5409376/chair.jpg'},
    {name: 'Cup image', src:
'https://assets.codepen.io/5409376/cup.jpg'},
    {name: 'Shoe image', src:
'https://assets.codepen.io/5409376/shoe.jpg'},
  ],
  videos: [
    {
      name: 'Camera video',
      src: 'https://assets.codepen.io/5409376/camera_3_8.mp4'
```

```
    },
    {
      name: 'Chair video',
      src: 'https://assets.codepen.io/5409376/chair_10_1.mp4'
    },
    {name: 'Cup video', src:
'https://assets.codepen.io/5409376/cup_43_42.mp4'},
    {
      name: 'Shoe video',
      src: 'https://assets.codepen.io/5409376/shoe_1063642984-sd-
trim-short.mov'
    }
  ],
};

// Our input frames will come from here.
const videoElement =
    document.getElementsByClassName('input_video')[0] as
HTMLVideoElement;
const canvasElement =
    document.getElementsByClassName('output_canvas')[0] as
HTMLCanvasElement;
const controlsElement =
    document.getElementsByClassName('control-panel')[0] as
HTMLDivElement;
const canvasCtx = canvasElement.getContext('2d')!;

// We'll add this to our control panel later, but we'll save it here
so we can
// call tick() each time the graph runs.
const fpsControl = new controls.FPS();

// Optimization: Turn off animated spinner after its hiding animation
is done.
const spinner = document.querySelector('.loading')! as
HTMLDivElement;
spinner.ontransitionend = () => {
  spinner.style.display = 'none';
};

function onResults(results: mpObjectron.Results): void {
  // Hide the spinner.
  document.body.classList.add('loaded');

  // Update the frame rate.
  fpsControl.tick();

  // Draw the overlays.
  canvasCtx.save();
  canvasCtx.drawImage(
```

```
      results.image, 0, 0, canvasElement.width,
canvasElement.height);
  if (!!results.objectDetections) {
    for (const detectedObject of results.objectDetections) {
      // Reformat keypoint information as landmarks, for easy
drawing.
      const landmarks: mpObjectron.Point2D[] =
          detectedObject.keypoints.map(x => x.point2d);
      // Draw bounding box.
      drawingUtils.drawConnectors(canvasCtx, landmarks,
          mpObjectron.BOX_CONNECTIONS, {color: '#FF0000'});

      // Draw Axes
      drawAxes(canvasCtx, landmarks, {
        x: '#00FF00',
        y: '#FF0000',
        z: '#0000FF',
      });
      // Draw centroid.
      drawingUtils.drawLandmarks(canvasCtx, [landmarks[0]], {color:
'#FFFFFF'});
    }
  }
  canvasCtx.restore();
}

const objectron = new mpObjectron.Objectron(config);
objectron.onResults(onResults);

// Present a control panel through which the user can manipulate the
solution
// options.
new controls
    .ControlPanel(controlsElement, {
      selfieMode: false,
      modelName: 'Shoe',
      maxNumObjects: 5,
      minDetectionConfidence: 0.5,
      minTrackingConfidence: 0.99,
    })
    .add([
      new controls.StaticText({title: 'MediaPipe Objectron'}),
      fpsControl,
      new controls.SourcePicker({
        onSourceChanged: (name, type) => {
          objectron.setOptions({staticImageMode: type !== 'image'});
          objectron.reset();
        },
        onFrame:
            async (input: controls.InputImage, size:
```

```
controls.Rectangle) => {
              const aspect = size.height / size.width;
              let width: number, height: number;
              if (window.innerWidth > window.innerHeight) {
                height = window.innerHeight;
                width = height / aspect;
              } else {
                width = window.innerWidth;
                height = width * aspect;
              }
              canvasElement.width = width;
              canvasElement.height = height;
              await objectron.send({image: input});
            },
        examples,
      }),
      new controls.Toggle({title: 'Selfie Mode', field:
'selfieMode'}),
      new controls.DropDownControl({
        title: 'Model',
        field: 'modelName',
        options: [
          {
            name: 'Shoe',
            value: 'Shoe',
          },
          {
            name: 'Camera',
            value: 'Camera',
          },
          {
            name: 'Chair',
            value: 'Chair',
          },
          {
            name: 'Cup',
            value: 'Cup',
          },
        ]
      }),
      new controls.Slider({
        title: 'Max Num Objects',
        field: 'maxNumObjects',
        range: [1, 10],
        step: 1,
      }),
      new controls.Slider({
        title: 'Min Detection Confidence',
        field: 'minDetectionConfidence',
        range: [0, 1],
```

```
        step: 0.01
      }),
      new controls.Slider({
        title: 'Min Tracking Confidence',
        field: 'minTrackingConfidence',
        range: [0, 1],
        step: 0.01
      }),
    ])
    .on(x => {
      const options = x as mpObjectron.Options;
      videoElement.classList.toggle('selfie', options.selfieMode);
      objectron.setOptions(options);
    });

function drawAxes(
    canvasCtx: CanvasRenderingContext2D, landmarks:
mpObjectron.Point2D[],
    color: {x: string; y: string; z: string;}) {
  const {
    BACK_BOTTOM_RIGHT,
    BACK_TOP_LEFT,
    BACK_TOP_RIGHT,
    FRONT_BOTTOM_LEFT,
    FRONT_BOTTOM_RIGHT,
    FRONT_TOP_RIGHT,
    FRONT_TOP_LEFT,
    CENTER
  } = mpObjectron.BOX_KEYPOINTS;

  const xMidPoint = lineIntersection(
      [landmarks[BACK_BOTTOM_RIGHT], landmarks[FRONT_TOP_RIGHT]],
      [landmarks[BACK_TOP_RIGHT], landmarks[FRONT_BOTTOM_RIGHT]]);
  const yMidPoint = lineIntersection(
      [landmarks[BACK_TOP_LEFT], landmarks[FRONT_TOP_RIGHT]],
      [landmarks[FRONT_TOP_LEFT], landmarks[BACK_TOP_RIGHT]]);
  const zMidPoint = lineIntersection(
      [landmarks[FRONT_TOP_RIGHT], landmarks[FRONT_BOTTOM_LEFT]],
      [landmarks[FRONT_TOP_LEFT], landmarks[FRONT_BOTTOM_RIGHT]]);

  const LINE_WIDTH = 8;
  const TRIANGLE_BASE = 2 * LINE_WIDTH;

  drawingUtils.drawConnectors(
      canvasCtx, [landmarks[CENTER], xMidPoint], [[0, 1]],
      {color: color.x, lineWidth: LINE_WIDTH});
  drawingUtils.drawConnectors(
      canvasCtx, [landmarks[CENTER], yMidPoint], [[0, 1]],
      {color: color.y, lineWidth: LINE_WIDTH});
  drawingUtils.drawConnectors(
```

```
      canvasCtx, [landmarks[CENTER], zMidPoint], [[0, 1]],
      {color: color.z, lineWidth: LINE_WIDTH});

  drawTriangle(
      canvasCtx, xMidPoint, TRIANGLE_BASE, TRIANGLE_BASE, color.x,
      arctan360(
          xMidPoint.x - landmarks[CENTER].x,
          xMidPoint.y - landmarks[CENTER].y) +
          Math.PI / 2);
  drawTriangle(
      canvasCtx, yMidPoint, TRIANGLE_BASE, TRIANGLE_BASE, color.y,
      arctan360(
          yMidPoint.x - landmarks[CENTER].x,
          yMidPoint.y - landmarks[CENTER].y) +
          Math.PI / 2);
  drawTriangle(
      canvasCtx, zMidPoint, TRIANGLE_BASE, TRIANGLE_BASE, color.z,
      arctan360(
          zMidPoint.x - landmarks[CENTER].x,
          zMidPoint.y - landmarks[CENTER].y) +
          Math.PI / 2);
}

function lineIntersection(
    a: mpObjectron.Point2D[], b: mpObjectron.Point2D[]):
mpObjectron.Point2D {
  const yDiffB = b[0].y - b[1].y;
  const xDiffB = b[0].x - b[1].x;

  const top = (a[0].x - b[0].x) * yDiffB - (a[0].y - b[0].y) *
xDiffB;
  const bot = (a[0].x - a[1].x) * yDiffB - (a[0].y - a[1].y) *
xDiffB;
  const t = top / bot;

  return {
    x: a[0].x + t * (a[1].x - a[0].x),
    y: a[0].y + t * (a[1].y - a[0].y),
    depth: 0,
  };
}

function drawTriangle(
    ctx: CanvasRenderingContext2D, point: mpObjectron.Point2D,
height: number,
    base: number, color: string, rotation: number = 0) {
  const canvas = ctx.canvas;
  const realX = canvas.width * point.x;
  const realY = canvas.height * point.y;
  ctx.save();
```

```
  ctx.beginPath();
  ctx.fillStyle = color;
  ctx.strokeStyle = color;
  ctx.lineWidth = 1;
  ctx.translate(realX, realY);
  ctx.rotate(rotation);
  ctx.moveTo(base / 2, 0);
  ctx.lineTo(0, -height);
  ctx.lineTo(-base / 2, 0);
  ctx.lineTo(base / 2, 0);
  ctx.translate(-realX, -realY);
  ctx.fill();
  ctx.stroke();
  ctx.restore();
}

function arctan360(x: number, y: number) {
  if (x === 0) {
    return y >= 0 ? Math.PI / 2 : -Math.PI / 2;
  }

  const angle = Math.atan(y / x);

  if (x > 0) {
    return angle;
  }

  return y >= 0 ? (angle + Math.PI) : angle - Math.PI;
}
```