<u>20-CREDIT PROJECT T1-23-34 REPORT</u>

TASKTAP VISION:

SEAMLESS OBJECT INTERACTION AND PRECISION TASK DELEGATION

ADRIJ SHARMA

IMT2019004

INSTRUCTOR: PROF. DINESH B JAYAGOPI

MULTIMODAL PERCEPTIONS LAB

TERM 1 2023-23

PROBLEM STATEMENT

Mains Goal

Autism Spectrum Disorder Detection using Computer Vision.

What is Autism?

1. Lack of eye contact during social interactions
2. Limited use of gestures
   a. Waving goodbye
   b. Pointing to show something interesting
3. Repetitive movements
   a. hand flapping
   b. body rocking
   c. spinning
4. Unusual reactions to sensory stimuli including sound, smell, taste, look, or feel.
5. Difficulty with interactive play.

Sub-problem

1. Correct/Incorrect Pointing to objects.
2. Depth-based mesh estimation of hand in 3D video.
3. Exact hand localization.
4. Hand gesture recognition.
5. 3D point cloud segmentation of hand.
6. The bounding box of the toy is entered by a finger.

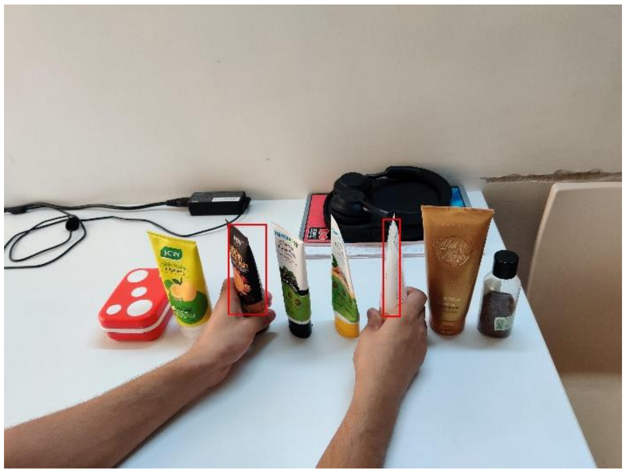
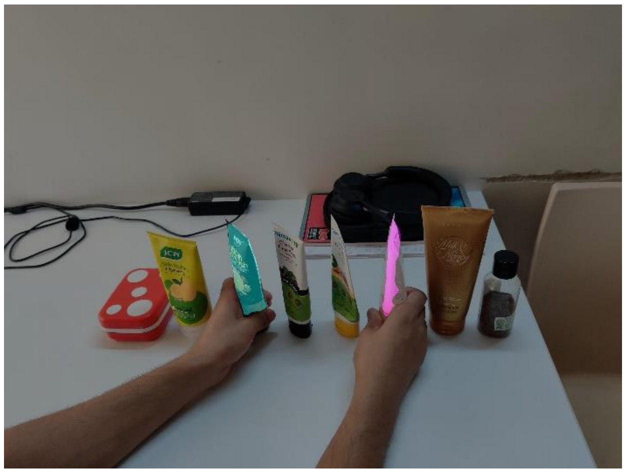
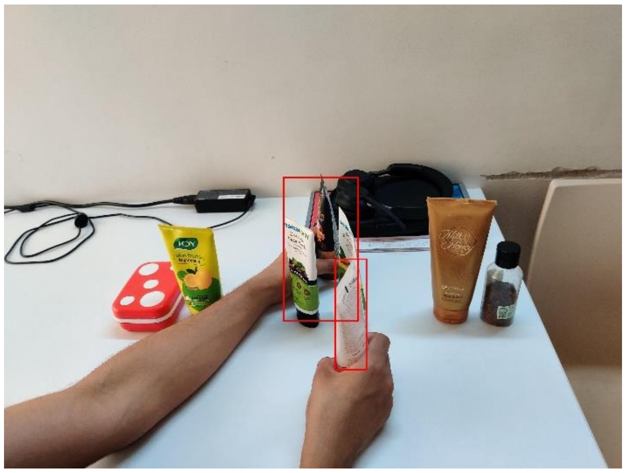
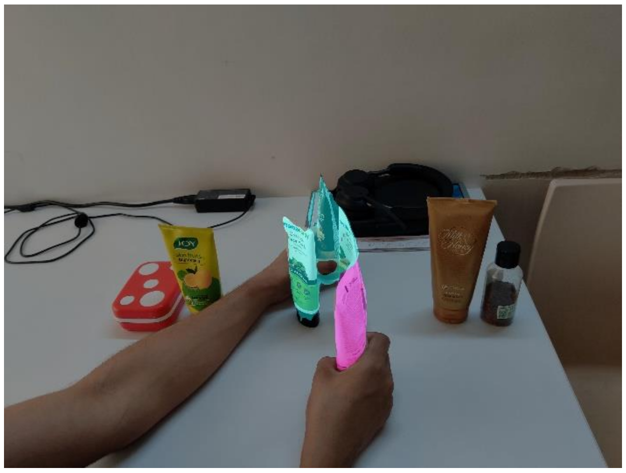
The ultimate goal is to develop a cutting-edge Android/iOS application that empowers user P1 to seamlessly capture an object using their device's camera by simply tapping the screen. Subsequently, P1 can provide instructions to subject S (another person) regarding a specific action to perform with the identified object. The application is designed to deliver a conclusive verdict at the end, assessing the accuracy of the assigned task and the chosen object. This innovative solution aims to streamline the process of object interaction and task execution, creating a user-friendly and efficient experience.

SOLUTION APPROACHES

1. Hand-object segmentation using existing published works.
    a. Fine-Grained Egocentric Hand-Object Segmentation: Dataset, Model, and Applications [1].
    b. YOLOv5 with Mixed Backbone for Efficient Spatio-Temporal Hand Gesture Localization and Recognition.[2]
    c. Detecting Human-Object Contact in Images.[3]
2. Hand-object real-time video segmentation using YOLO-v8[4], SWEM, SwiftNet.
    a. SWEM: Towards Real-Time Video Object Segmentation with Sequential Weighted Expectation-Maximization [5]
    b. SwiftNet: Real-time Video Object Segmentation.[6]
3. Object and hand depth calculation in an image using Monocular Depth Estimation. [7,8,9,10,11]
4. Mobile Application.

SOLUTION 1: HAND-OBJECT SEGMENTATION USING EXISTING PUBLISHED WORKS.

Glimpse:

Code: https://github.com/owenzlz/EgoHOS?tab=readme-ov-file

Dataset: https://www.seas.upenn.edu/~shzhou2/projects/eos_dataset/

Algorithm (the important part):

In this research on hand-object segmentation, the authors introduce a crucial concept called "dense contact boundary" to address the challenge of understanding the relationship between a hand and an interacting object. Traditional segmentation approaches based on appearance alone are insufficient because the same object may or may not require segmentation depending on whether the hand is in contact with it. The proposed dense contact boundary is defined as the contact region between the hand and the interacting object.

To create this dense contact boundary, the authors dilate the labelled hand and object masks in an image, identify the overlapped region between the two dilated masks, and binarize it to serve as pseudo-ground truth for the contact boundary. This boundary helps in training a network to predict it, enhancing the model's ability to segment interacting objects accurately. The dense contact boundary not only improves segmentation but also provides valuable information for downstream tasks like activity recognition and 3D mesh modelling of hand and object. Additionally, the authors introduce a context-aware compositional data augmentation pipeline. This pipeline selects "clean" background scenes, devoid of hands or interacting objects, and combines them with hand-object instances to create new semantically consistent images. Two methods for obtaining "clean" backgrounds are presented: using a binary classifier to find frames with no hands and employing image inpainting to remove hands from existing scenes. The proposed data augmentation technique is shown to be effective in experiments.

Result of images and videos recorded by us:

https://drive.google.com/drive/folders/1Q4QP9qVD3HgqXlVvTrOh76DdWgCDPdm4?usp=sharing

Visualization of object-hand contact boundaries (Visualize.py):

1. visualize_twohands Function:
    a. Input Parameters:
        i. img: Original image.
        ii. seg_result: Segmentation result for two hands.
        iii. alpha: Controls the transparency of the segmentation overlay.
    b. Functionality:
        i. Creates an array seg_color for the segmented regions with specified colors for each class (background, left hand, right hand).
        ii. Combines the original image and the segmented colors using the specified alpha value.
        iii. Returns the visualized image.

2. visualize_cb Function:
    a. Input Parameters:
        i. img: Original image.
        ii. seg_result: Segmentation result for contact points.
        iii. alpha: Controls the transparency of the segmentation overlay.
    b. Functionality:
        i. Creates an array seg_color for the segmented regions with specified colors for each class (background, contact points).

   ii. Combines the original image and the segmented colours using the specified alpha value.
   iii. Returns the visualized image.

3. visualize_twohands_obj1 and visualize_twohands_obj2 Functions:
   a. Input Parameters:
      i. img: Original image.
      ii. seg_result: Segmentation result for two hands and one or two additional objects.
      iii. alpha: Controls the transparency of the segmentation overlay.
   b. Functionality:
      i. Creates an array seg_color for the segmented regions with specified colors for each class (background, hands, objects).
      ii. Combines the original image and the segmented colors using the specified alpha value.
      iii. Returns the visualized image.

Object Recognition:

1. Pretrained YOLOv8 models have 80 classes. They are trained on the popular COCO (Common Objects in Context) dataset for both segmentation as well as detection.
2. ResNet: Load a pre-trained ResNet model. Preprocess the image with the segmented object. Pass the preprocessed image through the ResNet model to obtain predictions. Extract features from a relevant layer based on the bounding box information. Utilize the extracted features for object recognition.

SOLUTION 2: HAND-OBJECT REAL-TIME VIDEO SEGMENTATION USING YOLO-V8

To assess the optimal tactile interaction between a human hand and a given object, we have implemented four distinct approaches:

1. Evaluating the intersection between the bounding box of the object and the bounding box of the human hand.
2. Analyzing the overlap between the bounding box of the object and the segmentation result of the human hand (yielding the most accurate outcomes).
3. Comparing the segmentation result of the object with the segmentation result of the human hand.
4. Assessing the intersection of the segmentation result of the object with the bounding box of the human hand.

These methodologies allow us to comprehensively gauge the appropriateness of the tactile connection between the human hand and the object in question.

Code:
https://colab.research.google.com/drive/14vAdjyILuhyi8GO8og6wmPBD6Bp3Yuy0?usp=sharing

Code Summary:

The code processes each frame of the video, detecting objects and segmenting the scene. It specifically focuses on identifying overlapping regions between a person and other objects. The script calculates the intersection over union (IoU) for each frame, measuring the overlap between the person's segmentation mask and other object-bounding boxes. Detected objects are classified, and the script outputs the count of each object per frame. Furthermore, it identifies instances of potential overlap by comparing the calculated IoU with a predefined threshold. The results include a detailed breakdown of object occurrences and highlight objects that potentially overlap with a person, based on a specified criterion. This script provides insights into the spatial relationships between objects in a video, particularly focusing on interactions involving a person.

Intersection Over Union Computation Algorithm:

The script employs YOLO models for object detection and segmentation in a video, concentrating on identifying overlaps between a person and other objects. It calculates the Intersection over Union (IoU) metric to assess the degree of overlap. The IoU is computed using several functions:

1. compute_overlapping_area(overlapping_points): This function calculates the signed area of a polygon formed by overlapping points using the shoelace formula.

2. find_overlapping_region(box_points, boundary_points): Extracts corner points of the bounding box and checks if each boundary point lies within the bounding box. Returns the overlapping points.
3. get_intersection(box_points, boundary_points): Calculates the IoU by finding the area of overlap between the bounding box and segmentation mask using `find_overlapping_region()` and `compute_overlapping_area()`.
4. compute_boundary_area(boundary_points): Computes the area of a polygon formed by boundary points using the shoelace formula.
5. get_union(box_points, boundary_points): Calculates the union area of the bounding box and segmentation mask by summing their areas and subtracting the intersection area.

The script iterates through each frame, extracts information on detected objects and segmentation masks, and assesses the IoU between the person's segmentation mask and other object bounding boxes. Detected objects are classified, and the script outputs the count of each object per frame. Potential overlap is identified based on a user-defined threshold, providing a detailed analysis of spatial relationships between objects, particularly focusing on interactions involving a person. This approach enables a nuanced understanding of object interactions within the video content.

Real-time segmentation

Real-time code:

https://drive.google.com/file/d/1fdFZx3ltYQ_TX5XswKpMCL2hLm-7LTGx/view?usp=sharing

To make the system real-time compatible, you'll need to modify the code to continuously process incoming frames from a live video stream rather than processing frames from a pre-recorded video. Real-time processing typically involves capturing frames from a camera feed and performing object detection and analysis on each frame as it arrives.

1. Use a Live Video Stream:
    a. Instead of processing frames from a pre-recorded video file, the code is modified to capture frames from a live video stream.
    b. The `cv2.VideoCapture` class is used to open a connection to the default camera (`0` is usually the default camera index). This allows the code to read frames in real-time from the camera feed.
2. Replace Video File Loop with Real-Time Loop:
    a. The code replaces the loop that iterates through frames from a video file with a loop that continuously captures frames from the live video stream.
    b. The `cap.read()` function is used to read each frame from the live video stream.

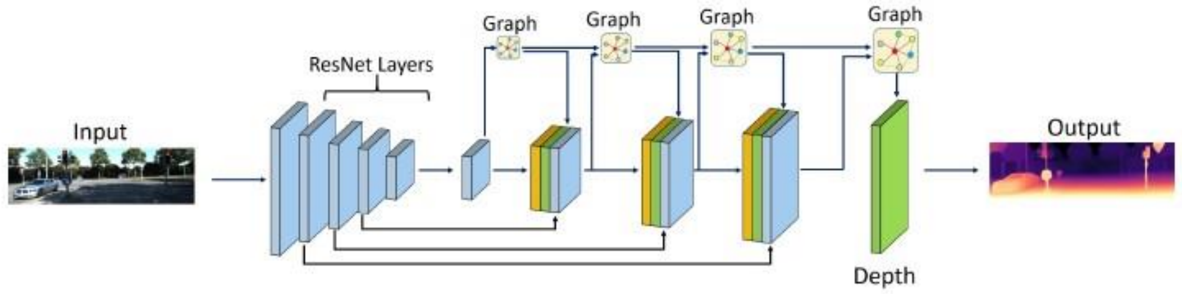
    c. The loop continues until the user decides to exit (e.g., by pressing the 'q' key). This loop structure is common in real-time applications to keep processing frames as long as the video stream is active.

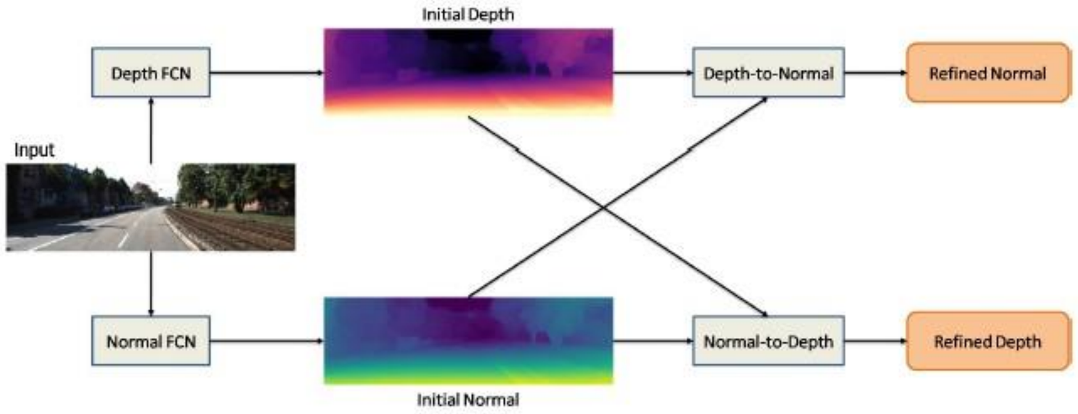
3. Adjust Processing Speed:
    a. Real-time systems often require adjustments to the processing speed to match the frame rate of the camera.
    b. The code processes each frame as it arrives, making it suitable for real-time applications. However, the actual speed at which frames are processed may depend on the complexity of the object detection and segmentation models.
4. Limitations:
    a. Real-time performance depends on factors such as the complexity of the model used for object detection and segmentation, as well as the capabilities of the hardware.
    b. If the current model or computation is too slow for real-time processing, potential solutions include using a more lightweight model, optimizing the existing model for speed, or leveraging hardware acceleration (e.g., GPU).
5. Asynchronous Processing (Optional):
    a. For more advanced real-time applications, asynchronous processing techniques can be considered to improve concurrency.
    b. Libraries like `asyncio` can be used to implement asynchronous processing, enabling the system to perform multiple tasks concurrently.

<u>SOLUTION 3: DEPTH ESTIMATION</u>

Monocular depth estimation (MDE) is a process in which depth information of a scene is estimated using a single image. It is a challenging task as depth cues are not readily available in a single image. However, with the advancements in deep learning, significant progress has been made in this field. Working of MDE:

1. Network Architecture: A deep learning model, typically based on convolutional neural networks (CNNs), is designed specifically for MDE. This model takes an RGB image as input and predicts the depth map as output.
2. Training: The network is trained using the collected dataset. The RGB images are fed into the network, and the predicted depth maps are compared to the ground truth depth maps. Through iterative adjustments of the model's parameters, the network aims to minimize the difference between the predicted and ground truth depth maps.
3. Loss Function: A loss function is employed to quantify the disparity between the predicted depth maps and ground truth depth maps. Commonly used loss functions include mean squared error (MSE) or Huber loss.
4. Optimization: The network parameters are optimized using techniques like gradient descent. The gradients of the loss function with respect to the network parameters are computed, and the parameters are updated accordingly to minimize the loss.
5. Inference: Once the network is trained, it can be used for inferring depth from new, unseen images. A test image is inputted into the trained network, and the network predicts the depth map for the image. This predicted depth map provides an estimation of the scene's depth information.





<u>Observations after using MDE and future path:</u>

1. It should be noted that successful MDE using deep learning relies on the availability of labelled training data with RGB-depth map pairs. By learning from this data, the network becomes capable of capturing depth-related features and patterns for accurate depth estimation during inference.
2. Utilizing the annotated EgoHOS dataset proves instrumental for our current task. To enhance its applicability, we aim to meticulously annotate it for depth, ensuring a comprehensive RGB format for seamless integration and optimal readiness. By training the network using labelled data, it learns to establish the relationship between RGB images and depth information, enabling accurate depth estimation for new images.
3. This model is too heavy for real-time usage. To optimize real-time performance in applications, there is a demand for lightweight Monocular Depth Estimation (MDE) networks. Nevertheless, we must be mindful that adopting lightweight networks may occasionally entail compromises in accuracy and resolution when predicting depth maps.

CODE, RESULTS AND VIDEOS LINK

https://drive.google.com/drive/folders/1jKc0yNeRfuAhzyoEhZQrOQDKb5f6UeM8?usp=sharing

REFERENCES

1. Zhang, Lingzhi, Shenghao Zhou, Simon Stent, and Jianbo Shi. "Fine-grained egocentric hand-object segmentation: Dataset, model, and applications." In European Conference on Computer Vision, pp. 127-145. Cham: Springer Nature Switzerland, 2022.
2. Luis, Acevedo-Bringas, Benitez-Garcia Gibran, Olivares-Mercado Jesus, and Takahashi Hiroki. "YOLOv5 with Mixed Backbone for Efficient Spatio-Temporal Hand Gesture Localization and Recognition." IEICE Proceedings Series 78, no. P2-05 (2023).
3. Chen, Yixin, Sai Kumar Dwivedi, Michael J. Black, and Dimitrios Tzionas. "Detecting Human-Object Contact in Images." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 17100-17110. 2023.
4. https://github.com/ultralytics/ultralytics
5. Wang, Haochen, Xiaolong Jiang, Haibing Ren, Yao Hu, and Song Bai. "Swiftnet: Real-time video object segmentation." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1296-1305. 2021.
6. Lin, Zhihui, Tianyu Yang, Maomao Li, Ziyu Wang, Chun Yuan, Wenhao Jiang, and Wei Liu. "Swem: Towards real-time video object segmentation with sequential weighted expectation-maximization." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 1362-1372. 2022.
7. Masoumian, Armin, Hatem A. Rashwan, Julián Cristiano, M. Salman Asif, and Domenec Puig. "Monocular depth estimation using deep learning: A review." Sensors 22, no. 14 (2022): 5353.
8. Gasperini, Stefano, Nils Morbitzer, HyunJun Jung, Nassir Navab, and Federico Tombari. "Robust monocular depth estimation under challenging conditions." In Proceedings of the IEEE/CVF international conference on computer vision, pp. 8177-8186. 2023.
9. https://github.com/zhyever/Monocular-Depth-Estimation-Toolbox
10. https://github.com/nianticlabs/monodepth2
11. https://github.com/ialhashim/DenseDepth