Manipulation using LLM and Modular CV Model Integration

ANONYMOUS AUTHOR(S)

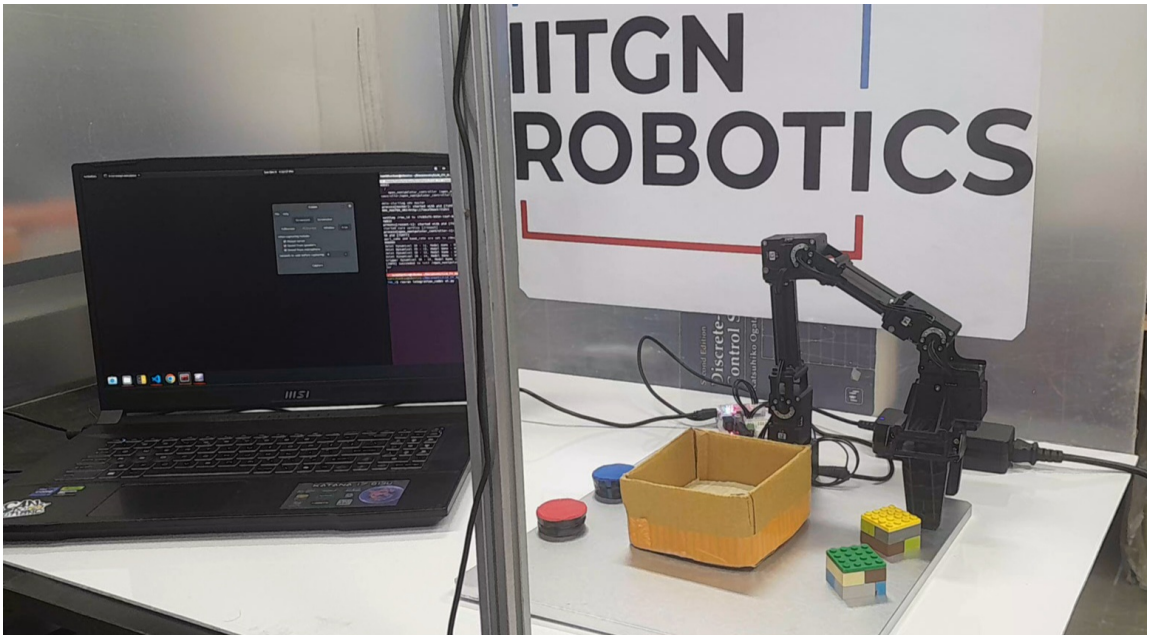


Fig. 1. OpenMANIPULATOR-X's workspace where the experiments are performed.

Achieving seamless robotic manipulation through voice commands is made possible by advancements in voice recognition, machine learning, and computer vision. A key strength lies in the modularity of the computer vision models, allowing for the integration of various zero-shot object recognition models tailored to specific tasks. This adaptability ensures the system can be easily updated or expanded as new models emerge. The process involves converting voice commands into text, extracting relevant objects using a Large Language Model, and determining the manipulator's joint angles via an Inverse Kinematics algorithm. This approach enhances scalability, flexibility, and future-proofing for advanced robotic manipulation.

CCS Concepts: • **Do Not Use This Code → Generate the Correct Terms for Your Paper**; *Generate the Correct Terms for Your Paper*; Generate the Correct Terms for Your Paper; Generate the Correct Terms for Your Paper.

Additional Key Words and Phrases: Do, Not, Us, This, Code, Put, the, Correct, Terms, for, Your, Paper

Manuscript submitted to ACM

ACM Reference Format:
Anonymous Author(s). 2018. Manipulation using LLM and Modular CV Model Integration. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 18 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The quest for seamless human-machine interaction has made significant strides in recent years, with advancements in voice recognition technology, machine learning, and computer vision playing pivotal roles. In the realm of robotic manipulation, achieving a task through a mere voice command epitomizes the synergy of these technologies.

A key strength of this project or pipeline lies in the modularity of the computer vision (CV) models used. Modularity allows for the integration of various zero-shot object recognition models, each tailored to specific tasks or object types, thus enhancing the system's adaptability and scalability.

The applications of this current research work is highly versatile, and for our experimentation and to show the physical working of our algorithms and integration of multiple models (LLM, CV, NLP, etc...) we take a very useful application: a robotic electrical lab and machine workshop assistant, which can perform manipulation tasks on tools and components in its configuration space (C-space or vicinity) by mere voice commands from the user. The manipulations are made possible through the OpenMANIPULATOR-X, which is a 4-DOF robotic manipulator. To test the working, we tend to ask the manipulator to place objects, tools and components from one place to another (point A –> point B).

2 Methodology

We can easily curate the sequence of steps we must take to ensure that the manipulation task is accomplished by a mere voice command from the user. So, that directly leads us to the voice recognition part of the pipeline. Once, we recognize the voice from the user and convert it into "`speech_text`", we feed the same to the LLM model. From which we would get only specific parts of the sentence which is required for the Zero-shot object recognition CV models (in our case objects present in the sentence). The CV models process the LLM's input and gives us the required coordinates to the manipulator (to perform pick-and-place). We would need an IK algorithm to solve the given coordinates and find the joint angles of the manipulator (here 4 joint angles), so that the manipulation can finally be performed successfully. The above sequence of steps can be simply depicted through the below flowchart.

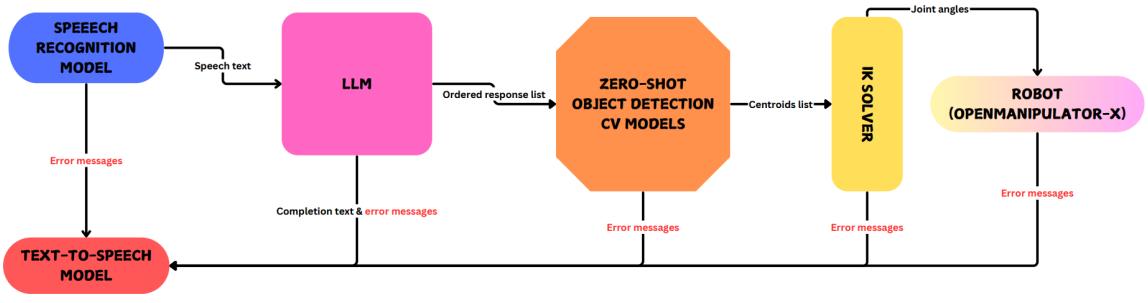


Fig. 2. Flowchart

2.1 Speech Recognition Model

Speech recognition models convert audio input, such as sounds from a microphone or pre-recorded files, into text by processing audio signals into meaningful features. They use neural networks to match sounds to words (acoustic models), ensure grammatical correctness with language models, and output the final transcribed text for further use. Some of Speech Recognition Models are DeepSpeech by Mozilla, Whisper by OpenAI and Google Speech-to-Text API.

For our task, we have used Google Speech-To-Text API, which is capable of capturing live audio and processing it in real-time. It supports over 100 languages, including accents like English(India). While performing the task, the language was chosen as "en-IN" to account for the Indian English accent.

```
import speech_recognition as sr
recognizer = sr.Recognizer()

with sr.Microphone() as source:
    # Recognize speech with Google Web Speech API using English (India)
    audio = recognizer.listen(source, timeout=5, phrase_time_limit=10)
    speech_text = recognizer.recognize_google(audio, language="en-IN")
    print(f"Recognized Speech: {speech_text}")
```

2.2 Large Language Model - "Qwen/Qwen2.5-72B-Instruct"

Large Language Models (LLMs) are a type of artificial intelligence that can understand, generate, and interact with human language. These models, such as OpenAI's GPT-3 and Google's BERT, are trained on vast amounts of text data, allowing them to comprehend context, answer questions, and produce human-like text. LLMs are versatile, supporting tasks like translation, summarization, answering queries, and content creation.

So, for our task we use an LLM called "Qwen2.5-72B-Instruct" developed by an organization called Qwen, which refers to the LLM family built by Alibab Cloud. The model consists of 72.7 billion parameters, which includes 2.7 billion embedding layers and 80 layers, which are stacked to form the deep neural network. For our use case, we deploy this specific model in two scenarios:

(1) To generate the answer for a basic query: "`speech_text`. *Give me only the two objects and its properties mentioned in the previous sentence in correct order of appearance, separated by commas.*"

(2) To generate a grammatically correct statement for a basic query: "`speech_text`. *Now say that you have placed the object asked to place.*"

The deployment of the model requires us to import OpenAI's "`openai`" module. Once imported we need to create an API key from our hugging face profile. The we implement it using the following code:

```
from openai import OpenAI

# Initialize the OpenAI client
client = OpenAI(
    base_url="https://api-inference.huggingface.co/v1/",
    api_key="hf_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"  # Replace with your actual Hugging Face API key
)

def analyze_instruction(speech_text):
    messages = [
        {
            "role": "user",
            "content": f"\"{speech_text}\". Give me only the two objects and its properties mentioned in the previous sentence in correct
                order of appearance, separated by commas."
        }
    ]
    try:
        completion = client.chat.completions.create(
            model="Qwen/Qwen2.5-72B-Instruct",
```

```
            messages=messages,
            max_tokens=500
        )
        output_content = completion.choices[0].message.content
        return output_content
    except Exception as e:
        print(f"Error during LLM inference: {e}")
        return "Error processing instruction."
```

The LLM's output would be fed to the Zero-shot object detection CV models in list format −> (1), for example if "speech_text" is *"Place the screwdriver on top of the circular plate"* then the "ordered_response_list" would be ["screwdriver", "circular plate"]. Once, the task manipulation is completed the LLM's output sentence would be sent to the text-to-speech module in order to give the user an interactive feel −> (2).

2.3 Zero-shot Object Detection CV Models

Zero-shot object detection (ZSD) is a computer vision task where models detect objects in images without prior training on those specific object classes. Given an image and a list of candidate classes, ZSD models output bounding boxes and labels for detected objects. This approach eliminates the need for extensive data annotation and model fine-tuning, making it efficient for applications like image search, object counting, and tracking. In our pipeline we use four different ZSD models, which obtain "ordered_response_list" from the LLM and they process the objects present in the list to draw the bounding boxes with certain confidence level or score to eventually calculate the centroid of the specific object asked for.

All of these models follow the same sequence of steps to return us the centroids of the objects and all of them recognize objects better when only a single object is given to it to detect. Hence, define a function which takes "model_id", "object_text", "frame_end" and "camera_index" as input parameters and finally returns "centroid", "confidence", "processing_time" and "processed_frame".

```
def detect_objects(model_id, object_text, frame_end=30, camera_index=2):
    device = "cuda" if torch.cuda.is_available() else "cpu"
    """
    Code which computes centroid, confidence, processing_time and returns the processed_frame which contains the bounding boxes and the
        centroids of the highest confident detection among the other detections.
    """
    return centroid, confidence, processing_time, processed_frame
```

2.3.1 Grounding DINO model (tiny variant). The Grounding DINO model combines the DINO architecture with grounded pre-training for open-set object detection. Developed by IDEA-Research, it extends a closed-set object detection model with a text encoder, enabling it to detect objects without prior training on specific classes. The tiny variant achieves impressive results, such as 52.5 AP (average precision) on COCO (Common Objects in Context dataset) zero-shot, making it highly effective for tasks like image search and object counting. Its ability to generalize across domains without additional training data makes it a powerful tool for zero-shot object detection.

Deploying the model using python requires us to import "AutoProcessor" and "AutoModelForZeroShotObjectDetection" from hugging face's "transformers library. Once imported we use the below code to deploy the same.

```
model_id = "IDEA-Research/grounding-dino-tiny"
processor = AutoProcessor.from_pretrained(model_id)
model =
AutoModelForZeroShotObjectDetection.from_pretrained(model_id).to(device)
    def process_frame_dino(frame, text):
        frame_copy = frame.copy()
        original_height, original_width = frame_copy.shape[:2]
```

```
        frame_rgb = cv2.cvtColor(frame_copy, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(frame_rgb)
        inputs = processor(images=image, text=text, return_tensors="pt")
        inputs = {key: value.to(device) for key, value in inputs.items()}
        start_time = time.time()
        with torch.no_grad():
            outputs = model(**inputs)
        processing_time = time.time() - start_time
        results = processor.post_process_grounded_object_detection(
            outputs,
            inputs["input_ids"],
            box_threshold=0.5,
            text_threshold=0.4,
            target_sizes=[(original_height, original_width)]
        )
        """
        Code to compute the highest_confidence_centroid, highest_confidence_score
        """
        else:
            print(f"No boxes detected for text: {text}")
        return highest_confidence_centroid, highest_confidence_score, processing_time, frame_copy

    centroid, confidence, processing_time, processed_frame = process_frame_dino(selected_frame, object_text)
```

From the code you can see that we are converting the frame we got from open-cv to RGB format, which is because open-cv module defines every pixel of a frame in BGR format and most of the CV models uses RGB format for the processing, hence the conversion. Once, it is converted we input the "`object_text`" along with the image to the processor, which gives us results. The results consists of bounding boxes and their respective confidence levels. The code to obtain the highest confidence centroids, bounding box and score is a simple iteration over all the boxes and we color the highest confidence bounding box in blue color to differentiate it from others.

The computation of centroid is just finding the coordinates of the bounding box and simply finding the centre point of the respective rectangle (bounding box). The code which computes the centroid is common for all the models.

```
if highest_confidence_box:
    x_min, y_min, x_max, y_max = highest_confidence_box
    centroid_x, centroid_y = highest_confidence_centroid
    cv2.rectangle(frame_copy, (x_min, y_min), (x_max, y_max), (255, 0, 0), 2)  # Blue color for highest confidence
    cv2.circle(frame_copy, (centroid_x, centroid_y), 5, (255, 0, 0), -1)  # Blue color for highest confidence
    cv2.putText(
        frame_copy,
        f"Highest Conf: ({centroid_x}, {centroid_y})",
        (x_min, y_max + 20),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.5,
        (255, 0, 0),
        1
    )
```

2.3.2 OmDet model. The OmDet model, proposed in "Real-time Transformer-based Open-Vocabulary Detection with Efficient Fusion Head" by Tiancheng Zhao et al., is designed for zero-shot (open-vocabulary) object detection. Utilizing the Swin-Tiny-HF architecture, which employs a hierarchical structure and shifted windows, OmDet significantly reduces computational complexity. This allows for efficient and scalable performance across various vision tasks. The model excels in detecting objects without predefined labels, ensuring adaptability for diverse applications. Its efficient design with the Swin-Tiny-HF architecture ensures high performance and fast inference, making it suitable for real-time scenarios where speed and accuracy are crucial.

The deployement of this model is very similar to the DINO model except here we use "`OmDetTurboForObjectDetection`" to initiate the model. Once we deploy it, obtaining the results, highest confidence score, centroids, bounding box and frame copy are exactly the same as before.

```
model_id = "omlab/omdet-turbo-swin-tiny-hf"
processor = AutoProcessor.from_pretrained(model_id)
model = OmDetTurboForObjectDetection.from_pretrained(model_id).to(device)
```

2.3.3 OWL-ViT. The OWL-ViT (Vision Transformer for Open-World Localization) is a zero-shot text-conditioned object detection model proposed by Matthias Minderer et al. It leverages CLIP (Contrastive Language-Image Pre-Training) as its multi-modal backbone, combining a ViT (Vision Transformer) for visual features and a causal language model for text features. The model removes the final token pooling layer and attaches lightweight classification and box heads to each transformer output token.

Text-conditioned refers to the model's ability to use text queries to influence the detection process. Contrastive loss is a technique used to train the model by maximizing the similarity between image-text pairs. OWL-ViT's architecture enables open-vocabulary classification, making it adaptable for diverse applications by allowing it to detect objects based on text descriptions. The model is trained from scratch and fine-tuned on detection datasets to enhance its performance. The deployment of this model is also similar to the last two models, except we import "`OwlViTProcessor` and "`OwlViTForObjectDetection` from "`transformers`" to initiate the model.

```
model_id = "google/owlvit-base-patch32"
processor = OwlViTProcessor.from_pretrained(model_id)
model = OwlViTForObjectDetection.from_pretrained(model_id).to(device)
```

2.3.4 OWLv2. The OWLv2 model, short for Open-World Localization version 2, is a zero-shot text-conditioned object detection model proposed in "Scaling Open-Vocabulary Object Detection" by Matthias Minderer, Alexey Gritsenko, and Neil Houlsby. Like its predecessor OWL-ViT, OWLv2 uses CLIP as its multi-modal backbone, combining a ViT to extract visual features and a causal language model for text features. OWLv2 employs a ViT-B/16 Transformer architecture, which offers finer detail capture compared to the ViT-B/32 used in OWL-ViT.

OWLv2 removes the final token pooling layer and attaches lightweight classification and box heads, enabling open-vocabulary classification by replacing fixed classification layer weights with class-name embeddings from the text model. It provides improved scalability, real-time performance, and better generalization to unseen objects. The enhanced training process and advancements in text processing pipelines contribute to OWLv2's superior efficiency and adaptability, making it highly effective for diverse applications requiring zero-shot object detection based on text queries.

These improvements mark OWLv2 as a significant advancement over OWL-ViT in open-vocabulary detection tasks. However, the deployment is exactly the same except we use "`Owlv2Processor`" and "`Owlv2ForObjectDetection` instead of the previous modules from the "`transformers`" library.

```
model_id = "google/owlv2-base-patch16-ensemble"
processor = Owlv2Processor.from_pretrained(model_id)
model = Owlv2ForObjectDetection.from_pretrained(model_id).to(device)
```

2.4 Inverse Kinematics

Inverse kinematics (IK) is a crucial concept in the field of robotics, where it is used to determine the joint configurations needed for a robot manipulator to reach a desired end-effector position and orientation. Unlike forward kinematics, which calculates the end position from given joint angles, IK works in reverse by specifying the target location and computing the necessary joint angles to achieve it. This process involves solving complex mathematical equations, often

requiring iterative methods such as Jacobian-based algorithms. IK is essential for tasks like robotic arm manipulation, enabling precise movement and positioning for applications in manufacturing, medical robotics, and autonomous systems.

The IK can be performed using "`MoveIt!`" packages for robots by feeding the URDF (Unified Robot Description Format) file to the "`MoveIt!`" setup assistant. Once the package is built it would contain controller launch files which could perform position control on the respective robot using the default IK solver (`KDLKinematicsPlugin`). Now, in our case the "`MoveIt!`" package's IK solver was very slow and it moved to singularity even when a position is easily attainable by the manipulator (OpenMANIPULATOR-X). So, we decided to create our own IK algorithm for the 4-DOF manipulator (excluding gripper, 1-DOF). To solve for the joint angles the OpenMANIPULATOR-X can be viewed as a combination of a 3R planar manipulator (3-DOF) fixed on top of a revolute joint (1-DOF).

2.4.1 IK of a 3R Planar Manipulator. The IK equations can be obtained easily for a 3R Planar Manipulator and are readily available in most of the reknown robotics textbooks. We can however attain the equations by applying geometry and trigonometry on Figure 4.

By apply triangular law of vector addition and law of sines, we can get the below equations:

$$\alpha = \cos^{-1}\left(\frac{x_3^2 + y_3^2 - L_{12}^2 - L_{23}^2}{2L_{12}L_{23}}\right) \tag{1}$$

$$\beta = \sin^{-1}\left(\frac{L_{23}\sin\alpha}{\sqrt{x_3^2 + y_3^2}}\right) \tag{2}$$

From the above figure we can attain the below equation through trigonometry:

$$P_3 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} x_e - L_{34}\cos(\gamma) \\ y_e - L_{34}\cos(\gamma) \end{bmatrix} \tag{3}$$

Now, through simple geometry we can get the joint angles of all these joints. Almost every position in the plane can have two orientations - elbow up (a) and elbow down (b). For our purpose, which are mostly pick-and-place of objects we have the liberty to choose only the elbow down (b) positions all the time. Therefore:

$$\theta_1^b = \left(\tan^{-1}\frac{y_3}{x_3} + \beta\right) \tag{4}$$

$$\theta_2^b = -(180 - \alpha) \tag{5}$$

$$\theta_3^b = \gamma - \theta_1^b - \theta_2^b \tag{6}$$

2.4.2 IK of only the base joint. The base angle can be calculated through simple tan inverse of y/x, where y and x are the end-effector coordinates. And the equation would be:

$$\theta_{\text{base}} = \tan^{-1}\left(\frac{y_e}{x_e}\right) \tag{7}$$

Equations (4), (5), (6) and (7) are the ones which we will be integrating together to compute all the joint angles of OpenMANIPULATOR-X.

```
def ikinematics_3R(positionx, positiony, positionz, gamma, link1=0.13, link2=0.125, link3=0.14):
    L12 = link1
    L23 = link2
    L34 = link3
```

Fig. 3. OpenMANIPULATOR-X, a 4-DOF (R) manipulator

```
    xe = (positionx**2 + positiony**2)**0.5 - 0.025
    ye = positionz - 0.04
    g = np.radians(gamma)
    x3 = xe - L34 * np.cos(g)
    y3 = ye - L34 * np.sin(g)
    C = np.sqrt(x3**2 + y3**2)
    if (L12 + L23) > C:
        a = np.degrees(np.arccos((L12**2 + L23**2 - C**2) / (2 * L12 * L23)))
```

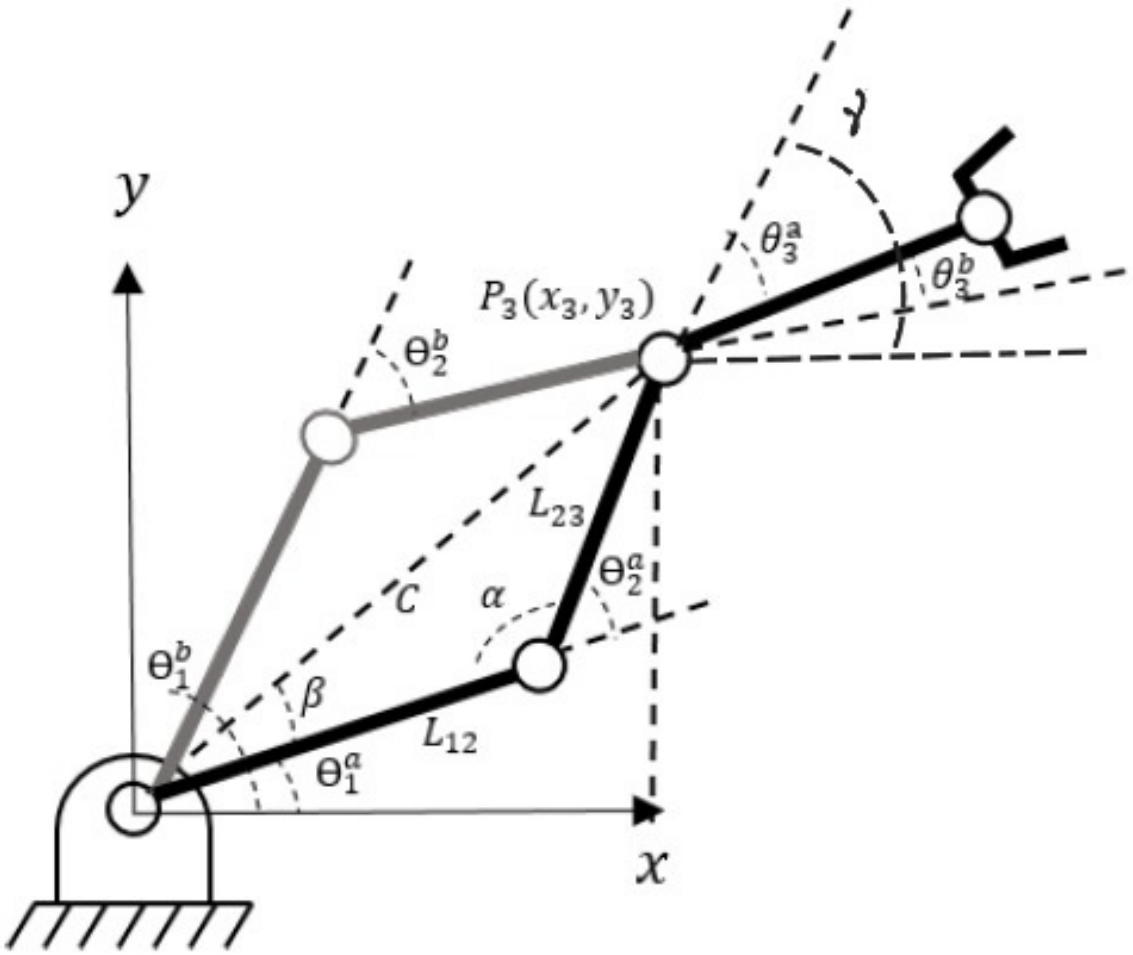


Fig. 4. 3R manipulator diagram with joint angles and link lengths

```
        B = np.degrees(np.arccos((L12**2 + C**2 - L23**2) / (2 * L12 * C)))
        J1b = 90 - (np.degrees(np.arctan2(y3, x3)) + B)
        J2b = -(-(180 - a) + 90)
        J3b = -(gamma - (np.degrees(np.arctan2(y3, x3)) + B) + (180 - a))
        return J1b, J2b, J3b
    else:
        print("Dimension error! End-effector is outside the workspace.")
        return None
def IK_4R(x, y, z, orientation):
    J1 = np.degrees(np.arctan2(y, x))
    J2, J3, J4 = ikinematics_3R(x, y, z, orientation)
    return J1, J2, J3, J4
```

Still we have to transform end-effector coordinates from the camera's base frame to the manipulator's base frame. We also need to convert the coordinates from pixels to meters. The transformation can be done by manually locating a

known point and finding the pixel coordinates of the same from the camera frame. We write a separate function called "`transform_pixels(x,y)`" to perform the above transformations and conversions.

```
def transform_pixels(x,y):
    ym = (x - 435)*0.000625
    xm = (y - 15)*0.000625
    return xm, ym
```

Finally, with the joint angles ready for any possible location in the C-space of the manipulator we perform normal joint control which is much faster and accurate than the "`MoveIt!`" package's position control.

2.5 Text-to-Speech Model

In order to provide a response to the user, we have used Google Text-To-Speech service, that converts text input into speech. It supports over 100 languages and accents. It provides options to customize the speed of the speech by the slow and speed_factor parameters.

We have used the text to speech model to provide a feedback like "Speech Recognized", and to provide feedback for some errors in recognition like, "Could not understand the audio. Please try again.".

```
from gtts import gTTS
from playsound import playsound
def text_to_speech(text, lang='en', speed_factor=1.1):
    # Generate speech from text
    tts = gTTS(text=text, lang=lang, slow=False)
    audio_file = "src/integration_codes/src/voices/output.mp3"
    tts.save(audio_file)

    # Adjust the speed of the speech
    if speed_factor != 1.0:
        change_speed(audio_file, speed_factor)

    # Play the audio
    playsound(audio_file)
```

3 Results and Metrics

We test the working of our project or pipeline by placing the manipulator in three different scenarios:

- colored blocks of different shapes
- electrical lab environment
- mechanical lab environment

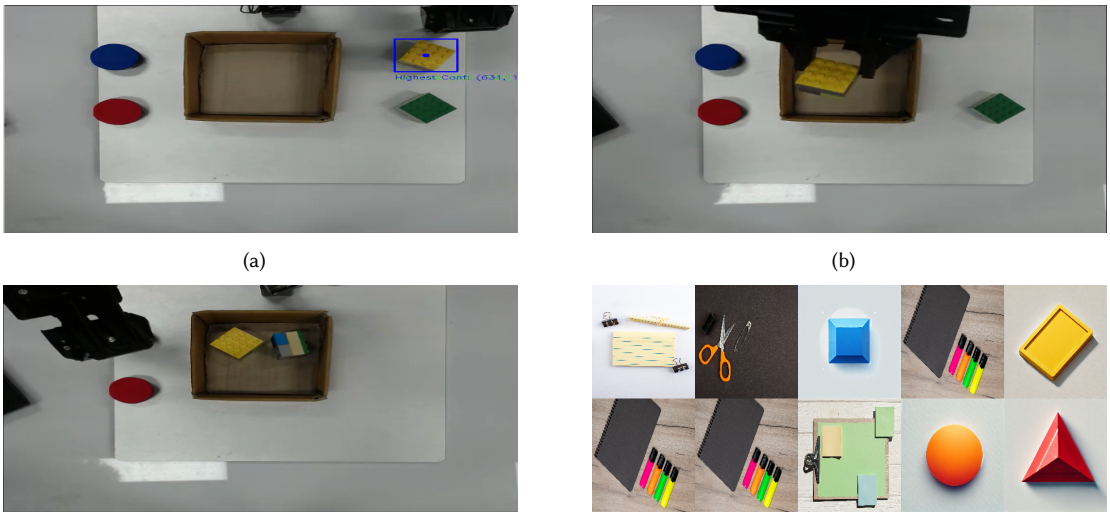
As mentioned earlier we give the manipulator natural voice commands and it assists you in the lab by completing the assigned task through manipulation (pick-and-place).

This video link (watch in 1.75x) shows the performance of the manipulator under different scenarios and different CV models. It is also well recommended to perform benchmark tests on these CV models to get a clear idea on how well or worse it performs under these different scenarios. So, we create a sample dataset of 50 images, which consists of 20 images of electrical or electronic components and tools, 20 images of mechanical tools and 10 images of simple shapes of different colors and popular stationary. We feed the sample dataset to all of these Zero-shot object detection CV models.

The data accumulation of these CV models can be seen in this Excel worksheet.

3.1 Colored Blocks of Different Shapes

In the Video (watch in 1.75x) we ask the robot to place the yellow square object inside the cardboard box followed by asking it to place the green square object, the blue circular object and the red circular object inside the cardboard box. We clearly see it perform the manipulations seamlessly without any issues.



(a) (b) (c) (d)

Fig. 5. Testing on different shapes, colours, and stationery items.

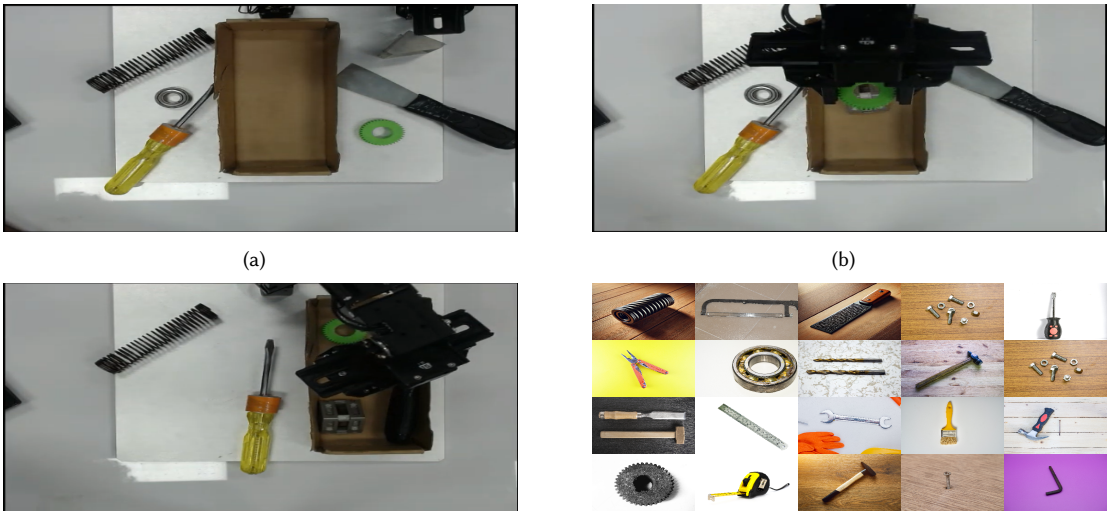
Table 1. Testing Models on Shapes, Colours, and Stationery Items.

Models	Accuracy	Average Processing Time (s)	Mean Confidence (Correct Predictions)
Grounding DINO	70.00%	10.489	80.81%
Omdet Turbo	80.00%	2.347	68.15%
OwlV2	90.00%	15.247	60.43%
OwlVit	80.00%	2.157	34.18%

After testing the four models on different shapes and colors, *OwlV2* was the most accurate. *OwlVit* emerged to be the fastest model, and *Grounding DINO* proved to be the most confident model in predictions.

3.2 Mechanical Tools

In the Video (watch in 1.75x) we ask the robot to place all the mechanical tools in clockwise order as shown in the video, inside the cardboard box. We clearly see it perform the manipulations seamlessly without any issues.



(a) (b) (c) (d)

Fig. 6. Testing models on mechanical tools and components

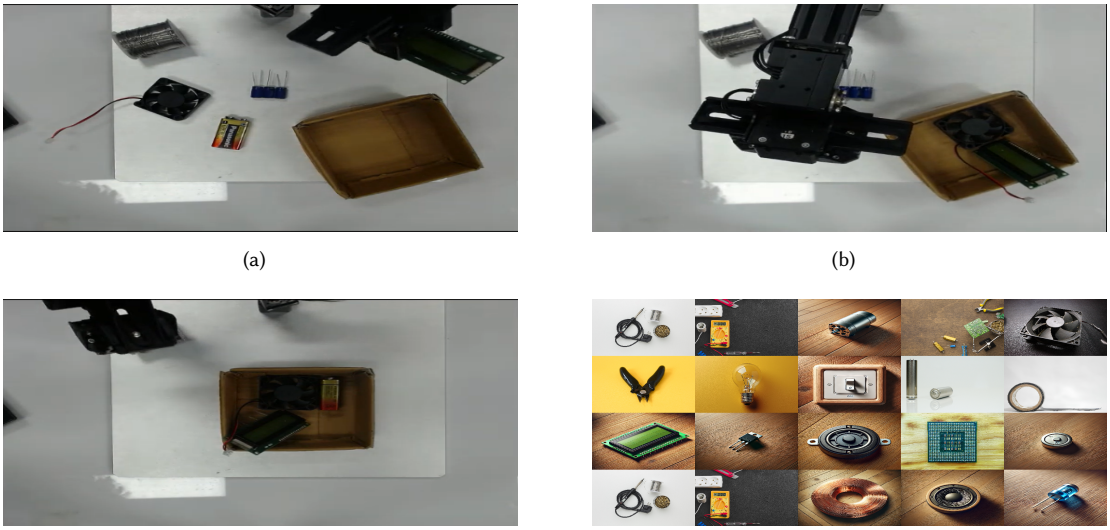
Table 2. Comparison of models to detect Mechanical Tools.

Models	Accuracy	Average Processing Time (s)	Mean Confidence (Correct Predictions)
Grounding DINO	80.00%	14.076	81.25%
Omdet Turbo	75.00%	2.372	60.34%
OwlV2	80.00%	14.883	47.04%
OwlVit	65.00%	1.175	28.44%

While testing over Mechanical Tools, *Grounding DINO* and *OwlV2* exhibited the maximum accuracy. *OwlV2* was the fastest model and *Grounding DINO* was the most confident model for predictions.

3.3 Electrical Tools

In the Video (watch in 1.75x) we ask the robot to place all the electrical or electronic tools and components in a specific order as shown in the video, inside the cardboard box. We clearly see it perform the manipulations seamlessly without any issues.



(a) (b) (c) (d)

Fig. 7. Testing models on electrical and electronic tools and components

Table 3. Comparison of models for detecting Electrical Tools.

Models	Accuracy	Average Processing Time (s)	Mean Confidence (Correct Predictions)
Grounding DINO	70.00%	13.524	80.72%
Omdet Turbo	95.00%	2.158	68.10%
OwlV2	95.00%	17.008	49.70%
OwlVit	75.00%	1.145	28.73%

While testing on Electrical Tools, *Omdet Turbo* and *OwlV2* showed the highest accuracy. *OwlVit* was the most accurate model, and *Grounding DINO* was the most confident model for predictions.

3.4 Overall Performance

Table 4. Overall comparison of models.

Models	Accuracy	Average Processing Time (s)	Mean Confidence (Correct Predictions)	Confidence-Weighted Accuracy
Grounding DINO	74.00%	13.138	80.97%	59.92%
Omdet Turbo	84.00%	2.282	65.34%	54.88%
OwlV2	88.00%	15.806	50.93%	44.81%
OwlVit	72.00%	1.360	29.84%	21.48%

Overall, *OwlV2* was the most accurate model. *OwlVit* was the fastest model, and *Grounding Dino* was the most confident model.

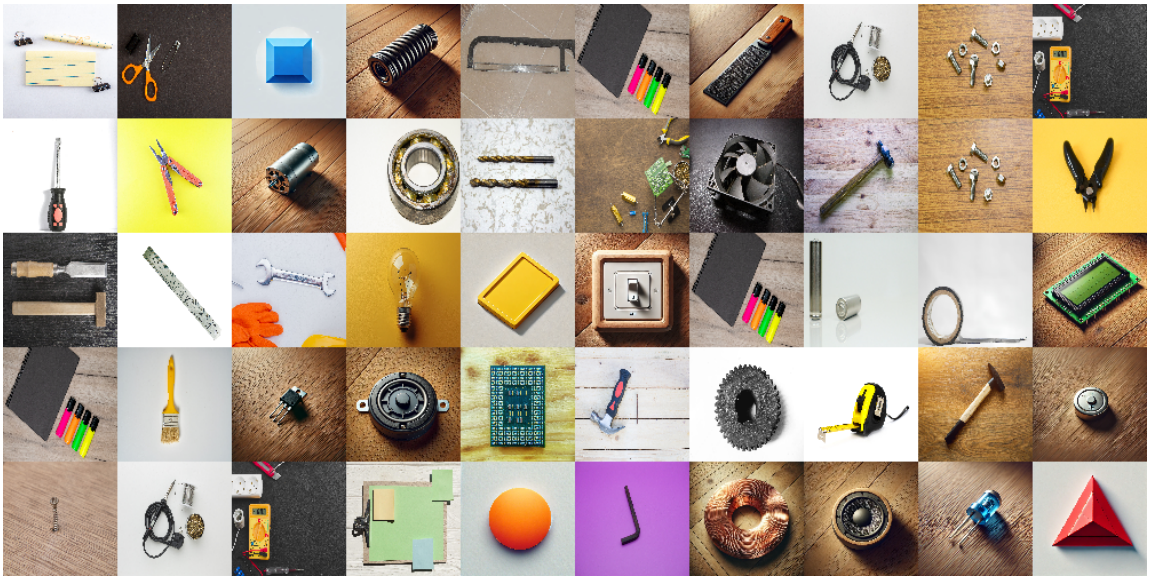


Fig. 8. Complete sample dataset of 50 images.

For the highest accuracy, *OwlV2* is preferred, albeit at the cost of speed and confidence. This is because it uses a deeper and more complex architecture, which increases accuracy but reduces speed. For good confidence and accuracy, *Grounding DINO* excels, making it reliable but slightly slower. Its use of additional attention mechanisms or object-centric processing explains its high confidence but also incurs a speed penalty. The fastest model was *OwlVit*, but it consistently performed poorly in accuracy and confidence, making it ideal only for high-speed applications. This is due to its relatively lighter architecture. *Omdet Turbo* was a well-balanced model, as it uses a moderately heavy architecture, making it suitable for tasks that require a balance between speed and accuracy.

4 Why Modularity of CV Models is Important?

The modularity of computer vision (CV) models is crucial for advancing the development and application of Robot-LLM systems that integrate language models with visual perception. Modular CV models offer flexibility, scalability, and adaptability, which are essential in addressing the diverse challenges faced in robotics and multi-modal AI systems.

(1) Modularity facilitates customization and reuse. By designing CV systems as interchangeable components, developers can easily tailor specific modules for tasks like object detection, scene segmentation, or gesture recognition without rebuilding the entire pipeline. This approach enables researchers to reuse pre-trained modules in different contexts, reducing development time and computational costs.

(2) Modularity enhances scalability. Complex robotics applications often require incremental updates or additions to their vision capabilities. A modular architecture allows seamless integration of new functionalities. This scalability is critical in dynamic environments where robotic systems need to adapt to evolving requirements.

(3) Modularity supports fault isolation and debugging. In large-scale systems, identifying the source of errors can be challenging. Modular CV architectures enable developers to isolate and diagnose specific components, ensuring more efficient troubleshooting and maintenance.

If in future, we need to add a new, better zero-shot object detection CV model to our pipeline we just need to add or change only the "`process_frame`" function. Once, that is done we are good to go. This makes our implementation modular, which is crucial for all the reasons given above. Finally, modularity promotes collaboration across research domains. Teams can independently develop and improve modules, fostering innovation and rapid prototyping.

5 Challenges Faced

- Finding the right LLM which is simple to deploy and works everytime for the same repetitive instruction it has been given.
- Writing modules for every block in the pipeline (efficient IK, Modular CV, pick-and-place, speech recognition, user interface, text-to-speech) which would be finally imported in the UI code.
- Overlay of live webcam feed and the display of "`processed_frame`" together in the same window of the user interface.
- Devising a single common function to detect objects and return centroids for all of the zero-shot object detection models.
- Figuring out to how to divide the inverse kinematics problem among the parts of the OpenMANIPULATOR-X manipulator, to solve it efficiently and easily.

6 Future Scope and Discussion

The integration of robotics and Large Language Models (LLMs) presents significant potential, given that both fields are among the fastest-evolving domains in modern technology. The rapid adoption of systems like OpenAI's ChatGPT into everyday life underscores the transformative impact of LLMs. Extending this capability to interact effectively with the physical environment through robotic systems offers unparalleled opportunities for societal impact. A technology pipeline, such as the one proposed in this study, has the potential to bridge the gap between digital intelligence and real-world application, surpassing the current limitations of LLMs in generating textual content.

One of the most impactful applications of such a pipeline is in assistive technologies, particularly for individuals with disabilities. For instance, the world's blind population could benefit significantly from a system capable of interpreting visual inputs and executing contextual physical tasks. Beyond the blind community, individuals with varying disabilities could use this cross-domain innovation to perform everyday activities more efficiently. Our experimentation, which employed a standard webcam and a 4-DOF manipulator, highlights the versatility of such systems. The addition of a depth-sensing camera, for example, could allow manipulators to perform precise tasks like placing tools directly into a user's hands.

The modular design of the pipeline ensures that it remains future-proof. Faster, more accurate CV or LLM modules can be seamlessly integrated without requiring a complete redesign. Another exciting avenue lies in industrial automation, where modular CV-LLM systems could optimize assembly lines by integrating complex decision-making capabilities with robotic dexterity. These advancements underscore the transformative potential of modular, cross-domain pipelines in addressing real-world challenges across industries.

7 Conclusion

The integration of robotics and Large Language Models (LLMs) with modular computer vision (CV) architectures represents a transformative step in advancing multi-modal AI systems. Our research highlights the importance of modularity in ensuring adaptability, scalability, and interoperability, allowing the pipeline to evolve with advancements

in both CV and LLM technologies. Through experimentation with a simple setup, we demonstrated the pipeline's potential in real-world applications, from assistive technologies for individuals with disabilities to broader industrial automation tasks. The modular design ensures future readiness, enabling seamless integration of more accurate and efficient models to meet emerging challenges.

This innovative cross-domain approach bridges the gap between digital intelligence and physical interaction, paving the way for a wide range of use cases. Ultimately, this research underscores the immense potential of combining LLMs with robotics, offering a pathway toward smarter, more inclusive, and highly adaptable systems for diverse real-world environments.

8 Citations and Bibliographies

The use of BibTEX for the preparation and formatting of one's references is strongly recommended. Authors' names should be complete — use full first names ("Donald E. Knuth") not initials ("D. E. Knuth") — and the salient identifying features of a reference should be included: title, year, volume, number, pages, article DOI, etc.

The bibliography is included in your source document with these two commands, placed just before the `\end{document}` command:

```
\bibliographystyle{ACM-Reference-Format}
\bibliography{bibfile}
```

where "`bibfile`" is the name, without the "`.bib`" suffix, of the BibTEX file.

Citations and references are numbered by default. A small number of ACM publications have citations and references formatted in the "author year" style; for these exceptions, please include this command in the **preamble** (before the command "`\begin{document}`") of your LATEX source:

```
\citestyle{acmauthoryear}
```

Some examples. A paginated journal article [2], an enumerated journal article [10], a reference to an entire issue [9], a monograph (whole book) [23], a monograph/whole book in a series (see 2a in spec. document) [17], a divisible-book such as an anthology or compilation [12] followed by the same example, however we only output the series if the volume number is given [13] (so Editor00a's series should NOT be present since it has no vol. no.), a chapter in a divisible book [34], a chapter in a divisible book in a series [11], a multi-volume work as book [22], a couple of articles in a proceedings (of a conference, symposium, workshop for example) (paginated proceedings article) [3, 15], a proceedings article with all possible elements [33], an example of an enumerated proceedings article [14], an informally published work [16], a couple of preprints [6, 7], a doctoral dissertation [8], a master's thesis: [4], an online document / world wide web resource [1, 27, 35], a video game (Case 1) [26] and (Case 2) [25] and [24] and (Case 3) a patent [32], work accepted for publication [29], 'YYYYb'-test for prolific author [30] and [31]. Other cites might contain 'duplicate' DOI and URLs (some SIAM articles) [21]. Boris / Barbara Beeton: multi-volume works as books [19] and [18]. A couple of citations with DOIs: [20, 21]. Online citations: [35–37]. Artifacts: [28] and [5].

References

[1] Rafal Ablamowicz and Bertfried Fauser. 2007. *CLIFFORD: a Maple 11 Package for Clifford Algebra Computations, version 11.* Retrieved February 28, 2008 from http://math.tntech.edu/rafal/cliff11/index.html

[2] Patricia S. Abril and Robert Plant. 2007. The patent holder's dilemma: Buy, sell, or troll? *Commun. ACM* 50, 1 (Jan. 2007), 36–44. https://doi.org/10.1145/1188913.1188915

[3] Sten Andler. 1979. Predicate Path expressions. In *Proceedings of the 6th. ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL '79)*. ACM Press, New York, NY, 226–236. https://doi.org/10.1145/567752.567774

[4] David A. Anisi. 2003. *Optimal Motion Control of a Ground Vehicle.* Master's thesis. Royal Institute of Technology (KTH), Stockholm, Sweden.

[5] Sam Anzaroot and Andrew McCallum. 2013. *UMass Citation Field Extraction Dataset.* Retrieved May 27, 2019 from http://www.iesl.cs.umass.edu/data/data-umasscitationfield

[6] Sam Anzaroot, Alexandre Passos, David Belanger, and Andrew McCallum. 2014. Learning Soft Linear Constraints with Application to Citation Field Extraction. arXiv:1403.1349

[7] Lutz Bornmann, K. Brad Wray, and Robin Haunschild. 2019. Citation concept analysis (CCA)—A new form of citation analysis revealing the usefulness of concepts for other researchers illustrated by two exemplary case studies including classic books by Thomas S. Kuhn and Karl R. Popper. arXiv:1905.12410 [cs.DL]

[8] Kenneth L. Clarkson. 1985. *Algorithms for Closest-Point Problems (Computational Geometry).* Ph. D. Dissertation. Stanford University, Palo Alto, CA. UMI Order Number: AAT 8506171.

[9] Jacques Cohen (Ed.). 1996. Special issue: Digital Libraries. *Commun. ACM* 39, 11 (Nov. 1996).

[10] Sarah Cohen, Werner Nutt, and Yehoshua Sagic. 2007. Deciding equivalances among conjunctive aggregate queries. *J. ACM* 54, 2, Article 5 (April 2007), 50 pages. https://doi.org/10.1145/1219092.1219093

[11] Bruce P. Douglass, David Harel, and Mark B. Trakhtenbrot. 1998. Statecarts in use: structured analysis and object-orientation. In *Lectures on Embedded Systems*, Grzegorz Rozenberg and Frits W. Vaandrager (Eds.). Lecture Notes in Computer Science, Vol. 1494. Springer-Verlag, London, 368–394. https://doi.org/10.1007/3-540-65193-4_29

[12] Ian Editor (Ed.). 2007. *The title of book one* (1st. ed.). The name of the series one, Vol. 9. University of Chicago Press, Chicago. https://doi.org/10.1007/3-540-09237-4

[13] Ian Editor (Ed.). 2008. *The title of book two* (2nd. ed.). University of Chicago Press, Chicago, Chapter 100. https://doi.org/10.1007/3-540-09237-4

[14] Matthew Van Gundy, Davide Balzarotti, and Giovanni Vigna. 2007. Catch me, if you can: Evading network signatures with web-based polymorphic worms. In *Proceedings of the first USENIX workshop on Offensive Technologies (WOOT '07)*. USENIX Association, Berkley, CA, Article 7, 9 pages.

[15] Torben Hagerup, Kurt Mehlhorn, and J. Ian Munro. 1993. Maintaining Discrete Probability Distributions Optimally. In *Proceedings of the 20th International Colloquium on Automata, Languages and Programming (Lecture Notes in Computer Science, Vol. 700)*. Springer-Verlag, Berlin, 253–264.

[16] David Harel. 1978. *LOGICS of Programs: AXIOMATICS and DESCRIPTIVE POWER.* MIT Research Lab Technical Report TR-200. Massachusetts Institute of Technology, Cambridge, MA.

[17] David Harel. 1979. *First-Order Dynamic Logic.* Lecture Notes in Computer Science, Vol. 68. Springer-Verlag, New York, NY. https://doi.org/10.1007/3-540-09237-4

[18] Lars Hörmander. 1985. *The analysis of linear partial differential operators. III.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. viii+525 pages. Pseudodifferential operators.

[19] Lars Hörmander. 1985. *The analysis of linear partial differential operators. IV.* Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], Vol. 275. Springer-Verlag, Berlin, Germany. vii+352 pages. Fourier integral operators.

[20] IEEE 2004. IEEE TCSC Executive Committee. In *Proceedings of the IEEE International Conference on Web Services (ICWS '04)*. IEEE Computer Society, Washington, DC, USA, 21–22. https://doi.org/10.1109/ICWS.2004.64

[21] Markus Kirschmer and John Voight. 2010. Algorithmic Enumeration of Ideal Classes for Quaternion Orders. *SIAM J. Comput.* 39, 5 (Jan. 2010), 1714–1747. https://doi.org/10.1137/080734467

[22] Donald E. Knuth. 1997. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms (3rd. ed.).* Addison Wesley Longman Publishing Co., Inc.

[23] David Kosiur. 2001. *Understanding Policy-Based Networking* (2nd. ed.). Wiley, New York, NY.

[24] Newton Lee. 2005. Interview with Bill Kinder: January 13, 2005. Video. *Comput. Entertain.* 3, 1, Article 4 (Jan.-March 2005). https://doi.org/10.1145/1057270.1057278

[25] Dave Novak. 2003. Solder man. Video. In *ACM SIGGRAPH 2003 Video Review on Animation theater Program: Part I - Vol. 145 (July 27–27, 2003)*. ACM Press, New York, NY, 4. https://doi.org/99.9999/woot07-S422 http://video.google.com/videoplay?docid=6528042696351994555

[26] Barack Obama. 2008. A more perfect union. Video. Retrieved March 21, 2008 from http://video.google.com/videoplay?docid=6528042696351994555

[27] Poker-Edge.Com. 2006. Stats and Analysis. Retrieved June 7, 2006 from http://www.poker-edge.com/stats.php

[28] R Core Team. 2019. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

[29] Bernard Rous. 2008. The Enabling of Digital Libraries. *Digital Libraries* 12, 3, Article 5 (July 2008). To appear.

[30] Mehdi Saeedi, Morteza Saheb Zamani, and Mehdi Sedighi. 2010. A library-based synthesis methodology for reversible logic. *Microelectron. J.* 41, 4 (April 2010), 185–194.

[31] Mehdi Saeedi, Morteza Saheb Zamani, Mehdi Sedighi, and Zahra Sasanian. 2010. Synthesis of Reversible Circuit Using Cycle-Based Approach. *J. Emerg. Technol. Comput. Syst.* 6, 4 (Dec. 2010).

[32] Joseph Scientist. 2009. The fountain of youth. Patent No. 12345, Filed July 1st., 2008, Issued Aug. 9th., 2009.

[33] Stan W. Smith. 2010. An experiment in bibliographic mark-up: Parsing metadata for XML export. In *Proceedings of the 3rd. annual workshop on Librarians and Computers (LAC '10, Vol. 3)*, Reginald N. Smythe and Alexander Noble (Eds.). Paparazzi Press, Milan Italy, 422–431. https://doi.org/99.9999/woot07-S422

[34] Asad Z. Spector. 1990. Achieving application requirements. In *Distributed Systems* (2nd. ed.), Sape Mullender (Ed.). ACM Press, New York, NY, 19–33. https://doi.org/10.1145/90417.90738
[35] Harry Thornburg. 2001. *Introduction to Bayesian Statistics.* Retrieved March 2, 2005 from http://ccrma.stanford.edu/~jos/bayes/bayes.html
[36] TUG 2017. *Institutional members of the TEX Users Group.* Retrieved May 27, 2017 from http://wwtug.org/instmem.html
[37] Boris Veytsman. 2017. *acmart—Class for typesetting publications of ACM.* Retrieved May 27, 2017 from http://www.ctan.org/pkg/acmart

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009