PransFly

Ponnekanti Pranathi[1,2*]

[1*]Computer Science and Engineering, IIT Goa, Farmagudi, Ponda, 403401, Goa, India.

Corresponding author(s). E-mail(s): ponnekanti.pranathi.21031@iitgoa.ac.in;

Abstract

This project explores the integration of advanced computer vision, natural language processing, and drone control technologies to enhance autonomous aerial systems. Leveraging the YOLO object detection model for real-time visual processing and OpenAI's GPT-based Large Language Models (LLMs) for contextual reasoning, this system enables drones to perform complex tasks guided by human-like comprehension of instructions.

Keywords: Computer Vision, Large Language Models, Drone, YOLO Object Detection

1 Introduction

Controlling drones through natural language brings a new level of intuitiveness to human-drone interaction, simplifying operations while opening doors for advanced AI systems to autonomously manage drone activities. This project leverages the capabilities of large language models (LLMs) to translate task instructions in natural language into executable commands for drones, revolutionizing how drones can be operated. By integrating a remote LLM (GPT-4o-mini) with cutting-edge object detection (YOLOv10x), facial recognition (face recognition library), and speech recognition (e.g., WebKit Speech Recognition), this project demonstrates an innovative and cohesive approach to drone control.

The system employs a multi-step process where user queries and real-time image frames from the drone's camera are sent to the remote LLM. The LLM processes these inputs—combining object detection results and user prompts—to generate actionable responses. Based on the response, the system commands the drone to perform specific tasks, such as detecting, identifying, or navigating around objects or individuals. The

incorporation of face recognition ensures user-specific tasks, such as identifying authorized personnel, while speech recognition provides an additional layer of accessibility, allowing for voice-based commands.

This integration results in a complex yet highly effective system capable of intelligent decision-making in real-world scenarios. The project's architecture not only bridges natural language understanding and robotics but also sets a foundation for the future of language-driven automation in drone technology.

2 Results

2.1 Object Detection and Classification

The system successfully uses the YOLOv10x model to detect and classify objects in the video feed in real-time. Detected objects are stored in a list (detected objects) and annotated on the video feed with bounding boxes and labels. The bounding boxes and annotations are dynamically updated on the video stream to reflect the current detections.

2.2 "Detect Only" Mode

Users can specify a target object for detection by including the phrase "detect only [target]" in their query. When "detect only" mode is active, the application focuses solely on detecting the specified target object. If the target object is detected, it is highlighted with a bounding box and label on the video stream.

2.3 Facial Recognition

For objects classified as "person," the system attempts to identify faces using the face recognition library. Recognized faces are matched against a database of known encodings (known face encodings), and the associated names (known face names) are displayed. For unknown faces, the user is prompted to provide a name for the newly detected face, which can then be added to the database.

2.4 Rotational Scanning

If an object relevant to the user's query is not immediately found, the system performs a virtual rotation (incrementing by 45 degrees up to 360 degrees) to scan for the target object. After a full rotation without finding the target object, the system concludes the task and notifies the user.

2.5 Integration with OpenAI GPT

The system generates a prompt for OpenAI's GPT model, combining the detected objects and the user's query, and includes a base64-encoded image of the current frame for contextual understanding. GPT provides a response indicating whether it can answer the query based on the detected objects, categorized as "yes" or "no." The application's behavior (e.g., stopping the task or continuing detection) is guided by GPT's response.

2.6 Real-Time Feedback via WebSockets

The results of detections, including annotated video frames and responses to user queries, are delivered in real time through WebSocket events. The client interface receives continuous updates of the detection progress and task status.

2.7 Multimodal Interaction

The system supports both textual queries and image data, enabling seamless interaction between video feed analysis and natural language understanding.

2.8 Voice command integration

Used Webspeech API, a speech recognition API which supports real time voice-to-text transcriptions.

3 Workflow of the application

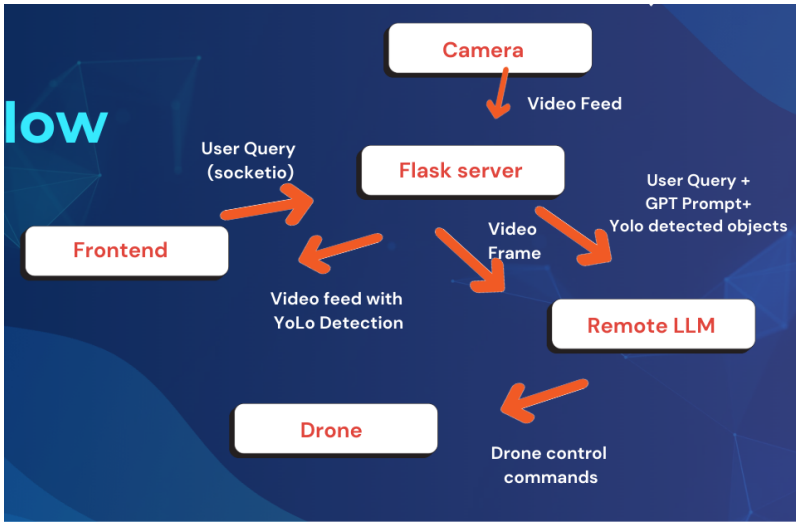

Fig. 1 Workflow of the application.

4 Figures

These are some of the screenshots of my application and its features. It has a user friendly interface to display the drone PoV(Drone Point of View) and a chat interface which is connected to the OpenAI's GPT model. It has a mic to enable speech recognition which does the same functionality as the text input.

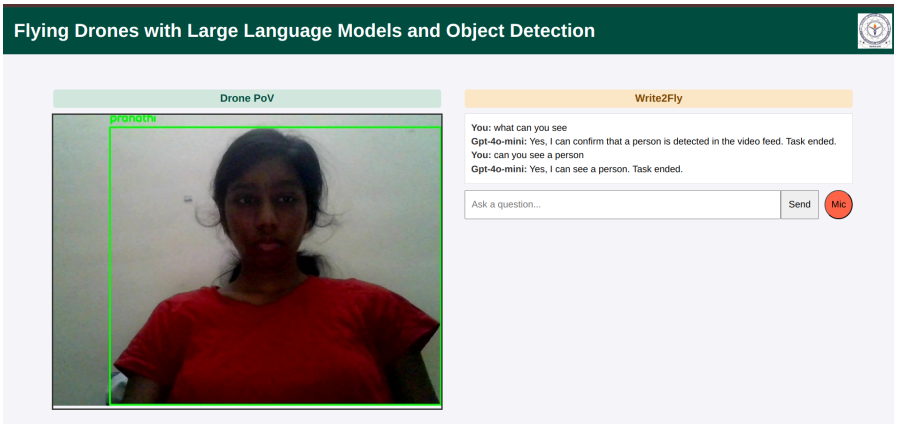

Fig. 2 UserInterface of PransFly

5 Algorithms, Program codes and Listings

5.1 Algorithms and Models used

Utilized yolov10x for object detection and manipulation of yolo focus by implementing detect only functionality. I have used 'face recognition' library and functionalities to identify and store the faces. Utilized socketio library for connection between the frontend and backend flask server(to send the queries and receive the response).

5.2 Program codes and Listings

5.2.1 Yolo Detection and Frame Annotation

The following listing demonstrates how YOLO is integrated for object detection and frame annotation:

Algorithm 1 YOLO Detection and Annotation

Results ← model.predict(source=frame, show=False, verbose=False)
Detected_objects ← [model.names[int(cls)] for cls in results[0].boxes.cls]
for all box, cls, conf in zip(results[0].boxes.xyxy, results[0].boxes.cls, results[0].boxes.conf) **do**
    obj_name ← model.names[int(cls)]
    $x_1, y_1, x_2, y_2 \leftarrow$ map(int, box)
    Draw rectangle with $(x_1, y_1), (x_2, y_2)$ and color $(0, 0, 255)$
    Add label obj_name at $(x_1, y_1 - 10)$
end for

Annotation Code:

```python
# Predict results using YOLO model
results = model.predict(source=frame, show=False, verbose=False)

# Extract detected object names
detected_objects = [model.names[int(cls)] for cls in results[0].boxes.cls]

# Loop through detected bounding boxes, classes, and confidences
for box, cls, conf in zip(results[0].boxes.xyxy, results[0].boxes.cls, results[0].
    boxes.conf):
    obj_name = model.names[int(cls)]
    x1, y1, x2, y2 = map(int, box)

    # Draw rectangle around the detected object
    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 2)

    # Add label above the bounding box
    cv2.putText(frame, obj_name, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0,
        0, 255), 2)
```

5.2.2 Face Recognition Integration

This listing explains how face recognition is used to match or register new faces:

Algorithm 2 Face Recognition Logic for YOLO Detected Person

if $obj\_name == \text{"}person\text{"}$ **then**
  $cropped\_face \leftarrow frame[y_1 : y_2, x_1 : x_2]$
  $rgb\_face \leftarrow \text{Convert}(cropped\_face, BGR \rightarrow RGB)$
  $face\_encodings \leftarrow \text{FaceEncodings}(rgb\_face)$
  if $face\_encodings \neq \emptyset$ **then**
    $matches \leftarrow \text{CompareFaces}(known\_face\_encodings, face\_encodings[0], \text{tolerance} = 0.6)$
    if $True \in matches$ **then**
      $match\_index \leftarrow \text{IndexOf}(True, matches)$
      $iname \leftarrow known\_face\_names[match\_index]$
    else
      $name \leftarrow \text{Input}(\text{"}Name : \text{"})$
      if $name \neq \emptyset$ and $name \notin known\_face\_names$ **then**
        $\text{Append}(known\_face\_encodings, face\_encodings[0])$
        $\text{Append}(known\_face\_names, name)$
      end if
    end if
  end if
end if

```python
if obj_name == "person":
    # Crop and preprocess the detected face
    cropped_face = frame[y1:y2, x1:x2]
    rgb_face = cv2.cvtColor(cropped_face, cv2.COLOR_BGR2RGB)

    # Generate face encodings
    face_encodings = face_recognition.face_encodings(rgb_face)
```

```
8
9       if face_encodings:
10          # Compare face encoding with known encodings
11          matches = face_recognition.compare_faces(known_face_encodings,
                face_encodings[0], tolerance=0.6)
12
13          if True in matches:
14              # Match found, retrieve the name
15              match_index = matches.index(True)
16              iname = known_face_names[match_index]
17          else:
18              # New face, prompt for a name
19              print("New face detected. Enter the person's name:")
20              name = input("Name: ").strip()
21
22              if name and name not in known_face_names:
23                  # Save new face encoding and name
24                  known_face_encodings.append(face_encodings[0])
25                  known_face_names.append(name)
```

5.2.3 Flask Web Application and WebSocket Integration

This listing shows the code structure for creating the video feed and handling client-server interactions:

Algorithm 3 Video Streaming and Question Handling via Flask and SocketIO

function VIDEO_STREAM
    return Response(generate_frames(), mimetype = 'multipart/x-mixed-replace; boundary=frame')
end function
function HANDLE_QUESTION(data)
    global *detected_objects*, *last_frame*
    question ← data['question']
    response_message ← query_openai(*gpt_prompt*, encode_image(*last_frame*))
    emit('*receive_answer*', {'*response*' : *response_message*})
end function

Flask and socket io integration Code:

```
1  from flask import Flask, Response, render_template
2  from flask_socketio import SocketIO
3  import cv2
4  import base64
5
6  app = Flask(__name__)
7  socketio = SocketIO(app)
8
9  # Video streaming route
10 @app.route('/video_feed')
11 def video_stream():
12     return Response(generate_frames(), mimetype='multipart/x-mixed-replace;
           boundary=frame')
13
14 # Route to render HTML page
15 @app.route('/video_page')
16 def video_page():
17     return render_template('index.html')  # Render the HTML template
```

```
18
19  # Function to encode the image frame
20  def encode_image(image):
21      _, buffer = cv2.imencode('.jpg', image)
22      return base64.b64encode(buffer).decode('utf-8')
23
24  # WebSocket event for processing questions from the client
25  @socketio.on('send_question')
26  def handle_question(data):
27      global detected_objects, task_ended, current_angle, last_frame
28
29      question = data['question']
30      response_message = ""
31
32      # Example logic for processing the question
33      if question == "Is the task done?":
34          response_message = "Yes, the task is complete."
35
36      # Emit the response back to the client
37      emit('receive_answer', {'response': f"{response_message} Task ended."})
38
39  if __name__ == '__main__':
40      socketio.run(app)
```

5.2.4 Integration with OpenAI GPT

I have used the model 'gpt-4o-mini' through openai api. This function takes an image along with the user's query and gpt prompt as input and sends the query to gpt model through the api. The response that it receives back is again emitted using socketio.

Algorithm 4 Query OpenAI GPT-4 with Base64 Image Input

function QUERY_OPENAI(*gpt_prompt*, *base64_image*)
  Set *openai.api_key* ← YOUR_API_KEY
  Create *response* with OpenAI API:
    Model: "gpt-4o-mini"
    Messages:
      {"role" : "user","content" : *gpt_prompt*}
      {"role" : "user","image_url" : ""+*base64_image*}
  return *response.choices*[0].*message.content*
end function

Code for Querying OpenAI GPT-4 with Base64 Image Input:

```
1   import openai
2
3   def query_openai(gpt_prompt, base64_image):
4       openai.api_key = "YOUR_API_KEY"  # Replace with your actual API key
5       response = openai.chat.completions.create(
6           model="gpt-4o-mini",
7           messages=[
8               {
9                   "role": "user",
10                  "content": gpt_prompt,
11              },
12              {
13                  "role": "user",
```

```
                "image_url": {
                    "url": f"{base64_image}"
                },
            },
        ]
    )
    return response.choices[0].message.content
```

5.2.5 Task completion and rotation

This code implements a task loop that uses object detection and OpenAI's API to determine the relevance of detected objects in a video feed to a user's question. The loop iteratively rotates a camera or view by 45 degrees, checking for relevant objects within each frame until either a relevant object is found or a full 360-degree rotation is completed. For each frame, if objects are detected and a previous frame is available, the last frame is encoded in Base64 format, and a prompt is generated for OpenAI. The prompt provides details about the detected objects and asks OpenAI to determine whether the objects are relevant to the user's query, responding with either "yes" or "no." Based on OpenAI's response, the task may end if relevance is detected ("yes"), or the camera rotation continues if not ("no"). If the camera completes a full rotation without finding relevant objects, the task ends with a message indicating that no relevant objects were detected. Throughout the process, feedback is printed to the console, and the final response is sent to a client using 'emit'. The below screenshot shows how the commands are being printed on the console to rotate or navigate whenever there is no positive relevance found.

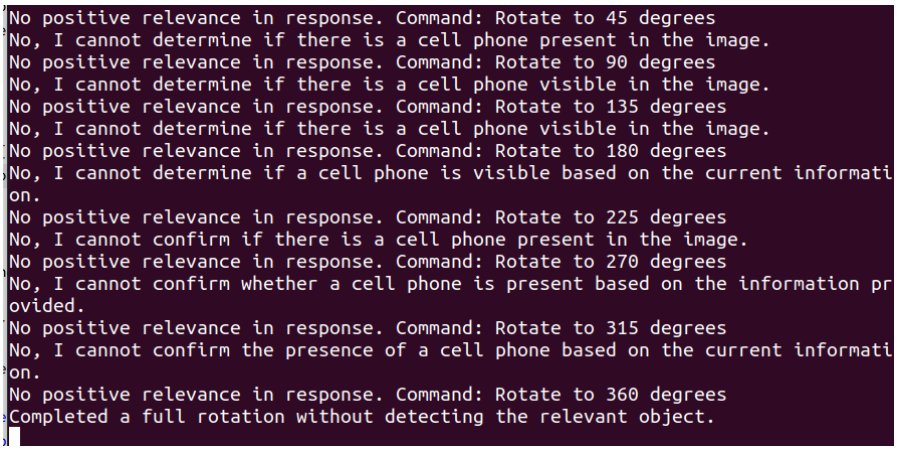

Fig. 3 Commands to navigate in console

Code for Task Execution with OpenAI and Object Detection:

```
import openai
import socketio
import cv2
import base64

```

Algorithm 5 Task Execution with OpenAI and Object Detection

Initialize `task_ended` as **False**, `current_angle` as 0
while not `task_ended` **and** `current_angle` < 360 **do**
  if `detected_objects` **and** `last_frame` $\neq$ **None then**
    `base64_image` $\leftarrow$ Encode(`last_frame`)
    `gpt_prompt` $\leftarrow$ "The user asked: '`question`'.
The following objects are currently detected in the video feed: `detected_objects`.
Based on this information, add 'yes' with a response if relevant, otherwise 'no'."
    `response_message` $\leftarrow$ QueryOpenAI(`gpt_prompt`, `base64_image`)
    Print(`response_message`)
    if "yes" **in** `response_message.lower()` **then**
      Print("Positive response detected. Ending task.")
      `task_ended` $\leftarrow$ **True**
      break
    else if "no" **in** `response_message.lower()` **then**
      Print("No positive relevance. Rotate to `current_angle` + 45 degrees.")
    end if
  end if
  `current_angle` $\leftarrow$ `current_angle` + 45
  `socketio.sleep(10)`
end while
if `current_angle` ≥ 360 **then**
  `task_ended` $\leftarrow$ **True**
  Print("Completed a full rotation without detecting the relevant object.")
  `response_message` $\leftarrow$ "No relevant object found after full rotation."
end if
Emit("receive_answer", {'response': `response_message` + " Task ended."})

```python
# Initialize global variables
task_ended = False
current_angle = 0

def encode_image(image):
    _, buffer = cv2.imencode('.jpg', image)
    return base64.b64encode(buffer).decode('utf-8')

def query_openai(gpt_prompt, base64_image):
    openai.api_key = "YOUR_API_KEY"  # Replace with your actual API key
    response = openai.chat.completions.create(
        model="gpt-4o-mini",
        messages=[
            {
                "role": "user",
                "content": gpt_prompt,
            },
            {
                "role": "user",
                "image_url": {
                    "url": f"{base64_image}"
                },
            },
        ]
    )
```

```
    return response.choices[0].message.content

while not task_ended and current_angle < 360:
    if detected_objects and last_frame is not None:
        base64_image = encode_image(last_frame)
        gpt_prompt = f"The user asked: '{question}'. \nThe following objects are
             currently detected in the video feed: {detected_objects}. Based on
             this information, add 'yes' with a response if relevant, otherwise 'no
             '."
        response_message = query_openai(gpt_prompt, base64_image)
        print(response_message)

        if "yes" in response_message.lower():
            print("Positive response detected. Ending task.")
            task_ended = True
            break
        elif "no" in response_message.lower():
            print(f"No positive relevance. Rotate to {current_angle + 45} degrees."
                )

    current_angle += 45
    socketio.sleep(10)

if current_angle >= 360:
    task_ended = True
    print("Completed a full rotation without detecting the relevant object.")
    response_message = "No relevant object found after full rotation."

socketio.emit("receive_answer", {'response': response_message + " Task ended."})
```

5.2.6 Voice command integration

Algorithm 6 Speech Recognition Integration

```
function startListening
    Initialize speech recognition system
    Set language to "en-US"
    Set recognition mode to "interimResults"
    Start recognition
    if Recognition started then
        Print "Voice recognition started"
    end if
    while Recognition continues do
        Capture interim results
        Update input field with transcript
    end while
    if Recognition ended then
        Print "Voice recognition ended"
        Send the transcribed text as a message
    end if
    if Error occurs then
        Print "Speech recognition error"
    end if
end function
```

Below is the JavaScript code used for implementing speech recognition:

Listing 1 Speech Recognition Integration Code

```
const recognition = new (window.SpeechRecognition)();
recognition.lang = 'en-US';
recognition.interimResults = true;

// Start listening on microphone button click
function startListening() {
    recognition.start();
    recognition.onstart = function() {
        console.log('Voice recognition started');
    };

    recognition.onresult = function(event) {
        let transcript = '';
        for (let i = event.resultIndex; i < event.results.length; i++) {
            transcript += event.results[i][0].transcript;
        }

        // Set the transcript in the input field
        document.getElementById("userInput").value = transcript;
    };

    recognition.onerror = function(event) {
        console.error('Speech recognition error', event);
    };

    recognition.onend = function() {
        console.log('Voice recognition ended');
        sendMessage();  // Send the transcribed text after recognition ends
    };
}
```

5.2.7 GPT-4's sentiment analysis

Making use of powerful gpt model's sentiment analysis functionality to check whether the relevance is found or not in the response. This step is useful in many other cases when we have to check relevance of the response to the visual representations and perform the navigation of the drone accordingly.

Sentiment Analysis

```
gpt_prompt = (f"The user asked: '{question}'. The following objects are currently detected in the video feed: "
              f"{', '.join(detected_objects)}. Based on this information, please add a 'yes' along with the gpt response if it is possible to provide an answer relevant to the user's question, "
              "or 'no' along with the gpt response if it is not possible.")

response_message = query_openai(gpt_prompt, base64_image)
print(response_message)  # Print the response to the console

if "yes" in response_message.lower():  # Positive sentiment suggests relevance
    print("Positive response detected. Ending task.")
    task_ended = True
    break
elif "no" in response_message.lower():
    print(f"No positive relevance in response. Command: Rotate to {current_angle + 45} degrees")
```

Algorithm 7 Sentiment Analysis

function QUERY_GPT_RESPONSE(*question*, *detected_objects*, *base*64_*image*)
    Generate *gpt_prompt* with user question and detected objects
    Set the prompt to ask if it is possible to provide an answer relevant to the question
    Send the *gpt_prompt* along with *base*64_*image* to OpenAI API
    Capture the response in *response_message*
    Print the *response_message*
    if "yes" in *response_message* **then**
        Print "Positive response detected. Ending task."
        Set *task_ended* ← True
        Exit the loop
    else if "no" in *response_message* **then**
        Print "No positive relevance in response."
        Print "Command: Rotate to current_angle + 45 degrees"
    end if
end function

6 Discussion

Integrating drones with large language models (LLMs) like GPT unlocks a wide range of advanced applications, enabling real-time analysis and contextual decision-making. For security, drones equipped with object detection can identify weapons, suspicious activities, or intruders, providing natural language reports about threats and understanding risks in scenarios like crowded areas or emergencies. In traffic monitoring, drones can recognize car models, brands, and modifications, cross-reference data with stolen vehicle databases, and answer natural language queries like, "How many red cars passed this street today?" In agriculture, drones can analyze crop types, recommend fertilizers or pest control solutions, and respond to farmer queries such as, "Which area needs more irrigation?" They can also explain environmental conditions impacting yield, like pest infestations. For healthcare, drones can recognize emergencies, explain medical kit contents, and offer step-by-step CPR instructions or locate the nearest hospital. These innovations demonstrate the transformative potential of combining drones with LLMs, enabling applications far beyond traditional systems and redefining efficiency and intelligence across multiple industries.

7 Conclusion

This project demonstrates a simulation of drone-like behavior, utilizing advanced functionalities implemented through YOLO-based object detection and the integration of large language models (LLMs). The code supports a "detect only" mode, enabling the system to prioritize and follow a specific target object in real-time, effectively mimicking a drone's ability to focus on a subject. Additionally, it incorporates a task rotation mechanism, whereby the system incrementally rotates by 45 degrees when no positive relevance is detected, systematically scanning the environment for the desired object.

The simulation concludes tasks upon either detecting the target object or completing a full 360-degree rotation. Given the unavailability of a physical drone, this implementation strives to simulate the core aspects of such a system, showcasing its potential to enhance drone operations through intelligent object detection and decision-making powered by LLMs.

References

[1] Redmon, Joseph, Divvala, Santosh, Girshick, Ross, and Farhadi, Ali. *You Only Look Once: Unified, Real-Time Object Detection.* arXiv preprint arXiv:1506.02640, 2015.

[2] Ultralytics. *Ultralytics YOLO.* Available at: https://github.com/ultralytics/ultralytics.

[3] OpenAI. *OpenAI API.* Available at: https://platform.openai.com/docs/.

[4] Ageitgey, Adam. *Face Recognition.* Available at: https://github.com/ageitgey/face_recognition.

[5] Pallets Projects. *Flask: The Python microframework for building web applications.* Available at: https://flask.palletsprojects.com/.

[6] Grinberg, Miguel. *Flask-SocketIO.* Available at: https://flask-socketio.readthedocs.io/.

[7] Bradski, Gary and Kaehler, Adrian. *OpenCV: Open Source Computer Vision Library.* Available at: https://opencv.org/.

[8] Python Software Foundation. *Base64 Encoding and Decoding in Python.* Available at: https://docs.python.org/3/library/base64.html.