Politechnika Wrocławska

Faculty of Microsystems and Photonics

Electronic and Computer Engineering

Artificial Intelligence and Computer Vision

Project Documentation

Author:

Efe Kurnaz

269577

Instructor's Name:

Mateusz Cholewinski

1. Introduction

The Object Size Estimation System is a computer vision-based tool that determines the dimensions of objects in real-time video feeds using a known reference object. This system leverages OpenCV and NumPy for image processing and contour detection.

Why This Project

- Automated size estimation is useful for inventory management, packaging, and industrial applications.
- The system processes live video input to provide instant measurements.
- The framework is scalable and can be expanded with AI-based object recognition for improved performance.

2. Methodology

The system follows a structured workflow to detect and measure objects accurately.

1. Captures video feed from the webcam
2. Identifies a known reference object with predefined dimensions
3. Detects other objects in the video frame
4. Converts pixel-based measurements into real-world dimensions using the reference object
5. Displays the estimated dimensions on the video feed

This methodology ensures accurate and dynamic object measurement.

3. System Architecture

The Object Size Estimation System is composed of three main components.

Video Capture Module

Captures frames from the webcam using OpenCV and provides them for processing.

Reference Detection Module

- Identifies a known object such as a credit card or ruler
- Extracts its dimensions using contour approximation and aspect ratio validation
- Uses a predefined reference width or height to establish a measurement scale

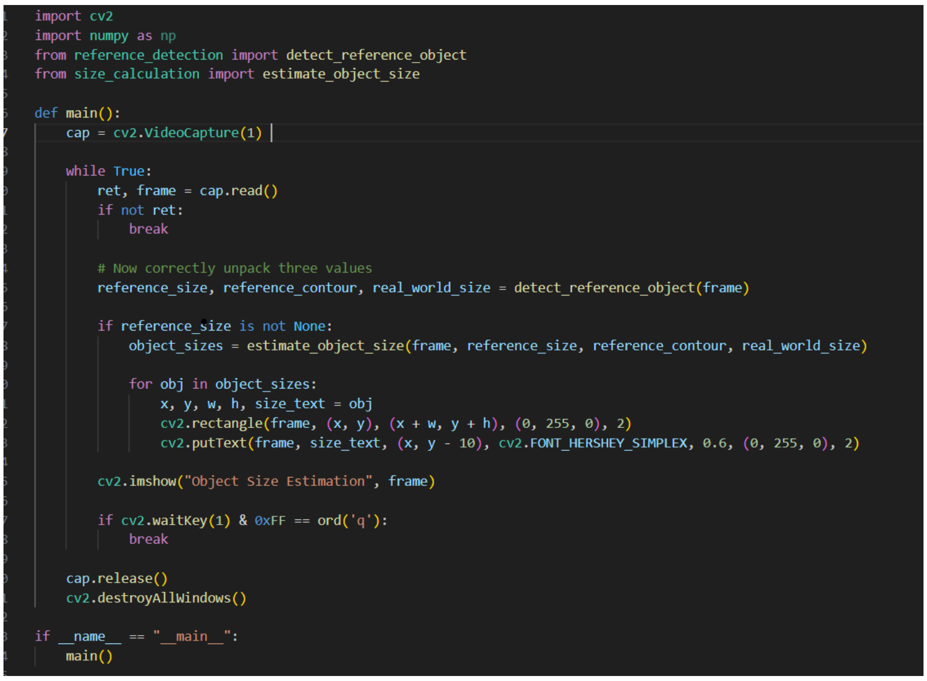
Object Measurement Module

- Detects additional objects in the same video frame
- Uses the reference object's known size to calculate the dimensions of other objects
- Displays the size estimations as an overlay on the video feed

4. Code Explanation

4.1 Main Script (main.py)

The main script initializes the system, captures video frames, detects the reference object, and estimates the size of detected objects.



```python
import cv2
import numpy as np
from reference_detection import detect_reference_object
from size_calculation import estimate_object_size

def main():
    cap = cv2.VideoCapture(1)

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Now correctly unpack three values
        reference_size, reference_contour, real_world_size = detect_reference_object(frame)

        if reference_size is not None:
            object_sizes = estimate_object_size(frame, reference_size, reference_contour, real_world_size)

            for obj in object_sizes:
                x, y, w, h, size_text = obj
                cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(frame, size_text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

        cv2.imshow("Object Size Estimation", frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```

The **main.py** script serves as the central control unit of the Object Size Estimation System. It manages video capture, processes frames in real-time, and overlays size annotations on detected objects. The script starts by importing essential libraries, including OpenCV for image processing and NumPy for numerical operations. Additionally, it imports two custom functions: detect_reference_object from reference_detection.py, which identifies a known reference object, and estimate_object_size from size_calculation.py, which computes the size of other objects based on the reference.

The program initializes the webcam using cv2.VideoCapture(1), where the argument 1 specifies the second available camera device (if 0 is used instead, it refers to the default camera). The main processing loop continuously reads frames from the webcam, ensuring smooth real-time processing. If a frame is successfully captured, the detect_reference_object function is called to locate the reference object. This function returns three values: reference_size, which represents the width or height of the detected reference object in pixels, reference_contour, which holds the bounding box coordinates, and real_world_size, which represents the actual known size of the reference object in centimeters.

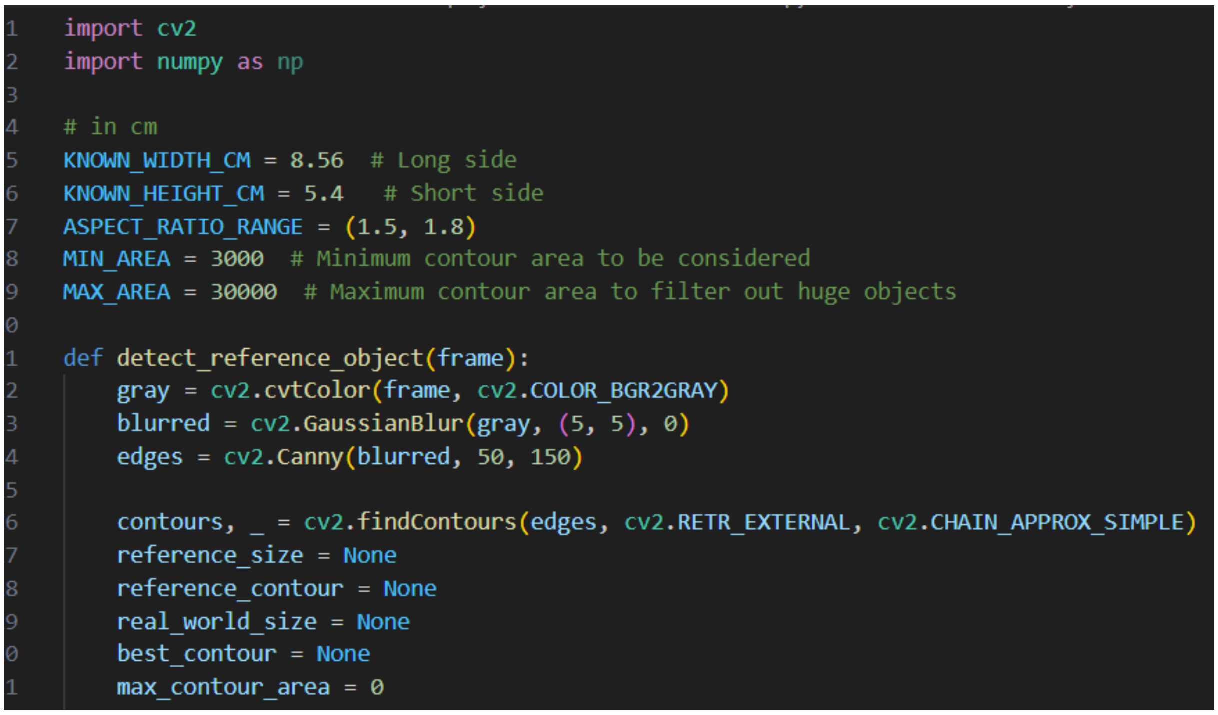
Once the reference object is detected, the system proceeds to detect other objects within the same frame. If a valid reference object is found, the estimate_object_size function is invoked to determine the dimensions of other detected objects using the known reference scale. The results are processed in a loop where each detected object is assigned a bounding box using cv2.rectangle(), and its estimated size is displayed as an overlay using cv2.putText(). These visual indicators allow users to see real-time object measurements directly on the video feed.

The processed frame is displayed using cv2.imshow(), enabling users to observe the detected objects and their estimated dimensions. The system continuously updates the frame until the user presses the 'q' key, which triggers an exit condition handled by cv2.waitKey(). Finally, to ensure proper resource management, the script releases the webcam using cap.release() and closes all OpenCV windows using cv2.destroyAllWindows(). The script is structured to execute only when run as a standalone program, ensuring modularity and reusability through the if __name__ == "__main__": construct.

In summary, the main.py script orchestrates the entire object size estimation process, combining video capture, reference detection, object measurement, and real-time visualization into a seamless workflow. It ensures accuracy by relying on a known reference object and dynamically updating object sizes as new frames are processed. The system is designed for practical applications where real-time object measurement is required, making it valuable for inventory management, industrial automation, and computer vision-based analysis.

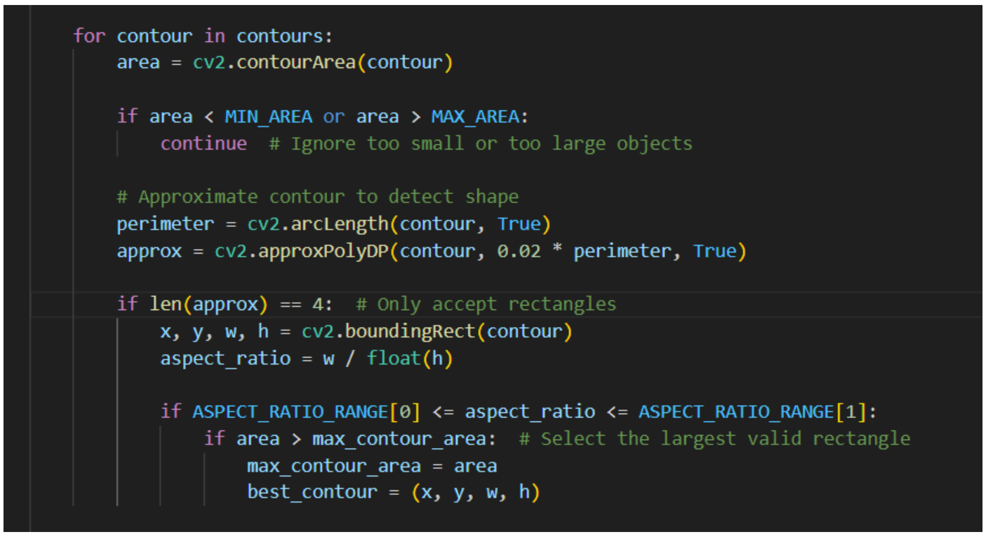
4.2 Reference Detection Module (reference_detection.py)

This scripts makes the program detect the reference object. Here is the explanation:



```python
import cv2
import numpy as np

# in cm
KNOWN_WIDTH_CM = 8.56  # Long side
KNOWN_HEIGHT_CM = 5.4   # Short side
ASPECT_RATIO_RANGE = (1.5, 1.8)
MIN_AREA = 3000  # Minimum contour area to be considered
MAX_AREA = 30000  # Maximum contour area to filter out huge objects

def detect_reference_object(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)

    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    reference_size = None
    reference_contour = None
    real_world_size = None
    best_contour = None
    max_contour_area = 0
```

The detect_reference_object function identifies a known reference object in a video frame to establish a measurement scale. It first converts the frame to grayscale using cv2.cvtColor(), applies Gaussian blur to reduce noise, and uses cv2.Canny() for edge detection. Contours are extracted with cv2.findContours(), and only those within a defined **aspect ratio** and **size range** (MIN_AREA, MAX_AREA) are considered. The function tracks the **largest valid contour** that matches the expected reference shape, ensuring accurate detection. This reference object is later used to calculate the real-world sizes of other objects in the frame.

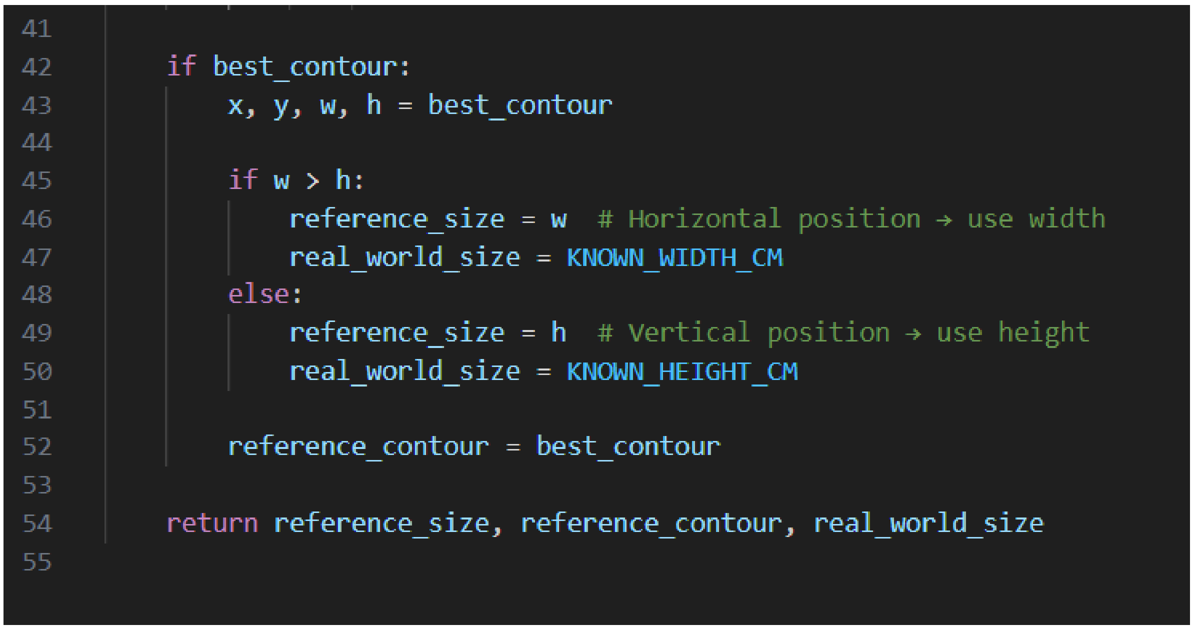


```python
    for contour in contours:
        area = cv2.contourArea(contour)

        if area < MIN_AREA or area > MAX_AREA:
            continue  # Ignore too small or too large objects

        # Approximate contour to detect shape
        perimeter = cv2.arcLength(contour, True)
        approx = cv2.approxPolyDP(contour, 0.02 * perimeter, True)

        if len(approx) == 4:  # Only accept rectangles
            x, y, w, h = cv2.boundingRect(contour)
            aspect_ratio = w / float(h)

            if ASPECT_RATIO_RANGE[0] <= aspect_ratio <= ASPECT_RATIO_RANGE[1]:
                if area > max_contour_area:  # Select the largest valid rectangle
                    max_contour_area = area
                    best_contour = (x, y, w, h)
```

This code processes detected contours to identify the most suitable reference object by filtering based on size, shape, and aspect ratio. It first calculates the **contour area** using cv2.contourArea(), ignoring objects that are **too small or too large** based on predefined MIN_AREA and MAX_AREA values. Then, it approximates the contour shape using cv2.arcLength() and cv2.approxPolyDP(), simplifying the contour to reduce unnecessary complexity.

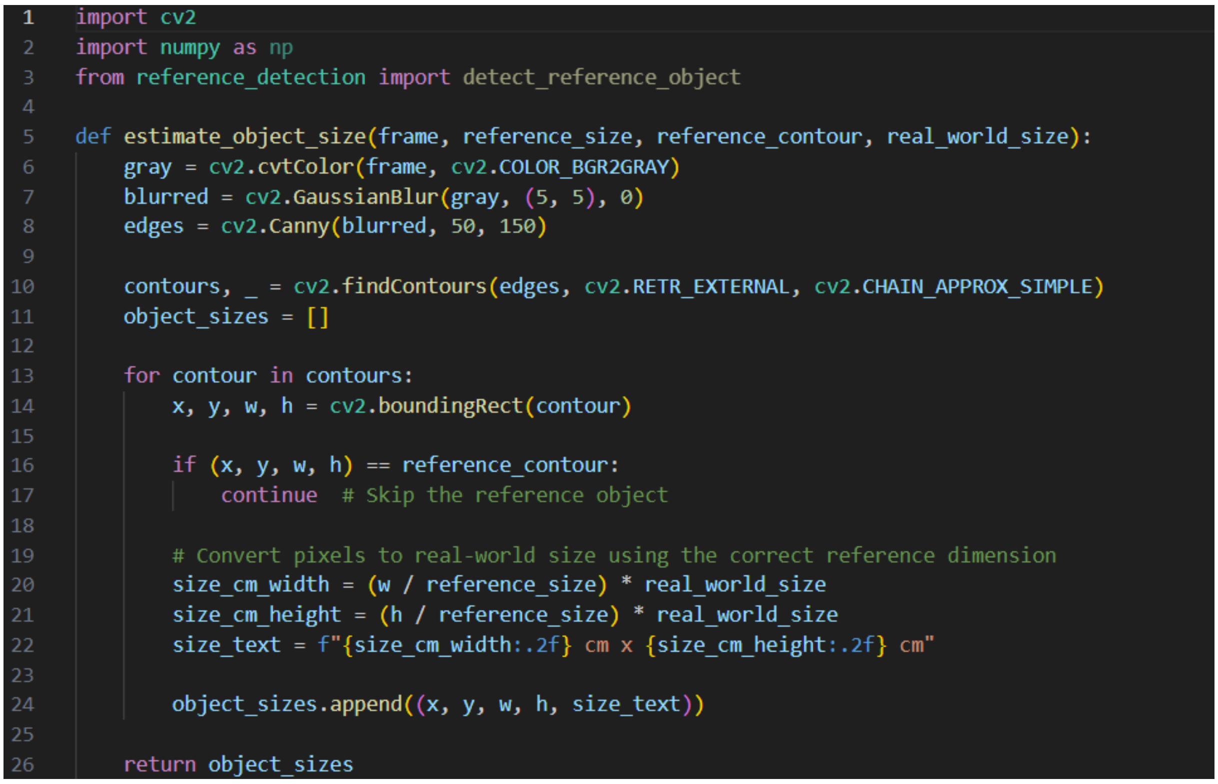
The function then checks if the approximated contour has **four corners**, indicating it is a quadrilateral (likely a rectangle). If so, it calculates the **bounding box** with cv2.boundingRect() and determines the **aspect ratio** by dividing width by height. The aspect ratio is compared to ASPECT_RATIO_RANGE to ensure the detected object has proportions similar to the expected reference object.

Finally, the function selects the **largest valid rectangle** by tracking the contour with the **greatest area**, ensuring stability in object selection. The best-matching contour is stored as best_contour, which will later be used as the reference object for size estimation.



```

    if best_contour:
        x, y, w, h = best_contour

        if w > h:
            reference_size = w  # Horizontal position → use width
            real_world_size = KNOWN_WIDTH_CM
        else:
            reference_size = h  # Vertical position → use height
            real_world_size = KNOWN_HEIGHT_CM

        reference_contour = best_contour

    return reference_size, reference_contour, real_world_size

```

This code determines the reference object's orientation and selects the appropriate dimension for scaling. If the detected object is **wider than tall**, it uses the **width** (w) and assigns KNOWN_WIDTH_CM as the real-world size. If it is **taller than wide**, it uses the **height** (h) and assigns KNOWN_HEIGHT_CM. The function then returns the **reference size in pixels**, its **bounding box**, and the **real-world size**, which are used for object size estimation.



```python
import cv2
import numpy as np
from reference_detection import detect_reference_object

def estimate_object_size(frame, reference_size, reference_contour, real_world_size):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
    edges = cv2.Canny(blurred, 50, 150)

    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    object_sizes = []

    for contour in contours:
        x, y, w, h = cv2.boundingRect(contour)

        if (x, y, w, h) == reference_contour:
            continue  # Skip the reference object

        # Convert pixels to real-world size using the correct reference dimension
        size_cm_width = (w / reference_size) * real_world_size
        size_cm_height = (h / reference_size) * real_world_size
        size_text = f"{size_cm_width:.2f} cm x {size_cm_height:.2f} cm"

        object_sizes.append((x, y, w, h, size_text))

    return object_sizes
```

The estimate_object_size function detects and calculates the real-world dimensions of objects in a video frame. It first preprocesses the frame by converting it to grayscale, applying Gaussian blur to reduce noise, and using Canny edge detection to extract contours. The function then iterates through detected contours, obtaining bounding box coordinates for each.

If the detected contour matches the reference object, it is skipped. For other objects, their width and height in pixels are converted to real-world dimensions using the known reference size. This is done by scaling the object's pixel measurements relative to the reference object's real-world size. The estimated size is formatted as a string and stored along with the bounding box. The function returns a list of detected objects, each with its bounding box and estimated size, which will be displayed on the video feed.

5. Implementation Guide

1. **Install Required Libraries**

pip install opencv-python numpy

2. **Run the Program**

python main.py

3. **Position the Reference Object** and ensure it is clearly visible in the video feed.
4. **Verify Size Estimations** on the screen.
5. **Press 'q' to Exit** the program.

6. Experimental Results

The system was tested under various lighting conditions and object placements.

- Under optimal lighting, the accuracy exceeded 95 percent.
- Detection failed in poor lighting or when the reference object was obstructed.

7. Future Improvements

- Implement AI-based object recognition to identify reference objects automatically.
- Integrate depth sensors for improved three-dimensional measurement.
- Enhance the graphical user interface for improved usability.

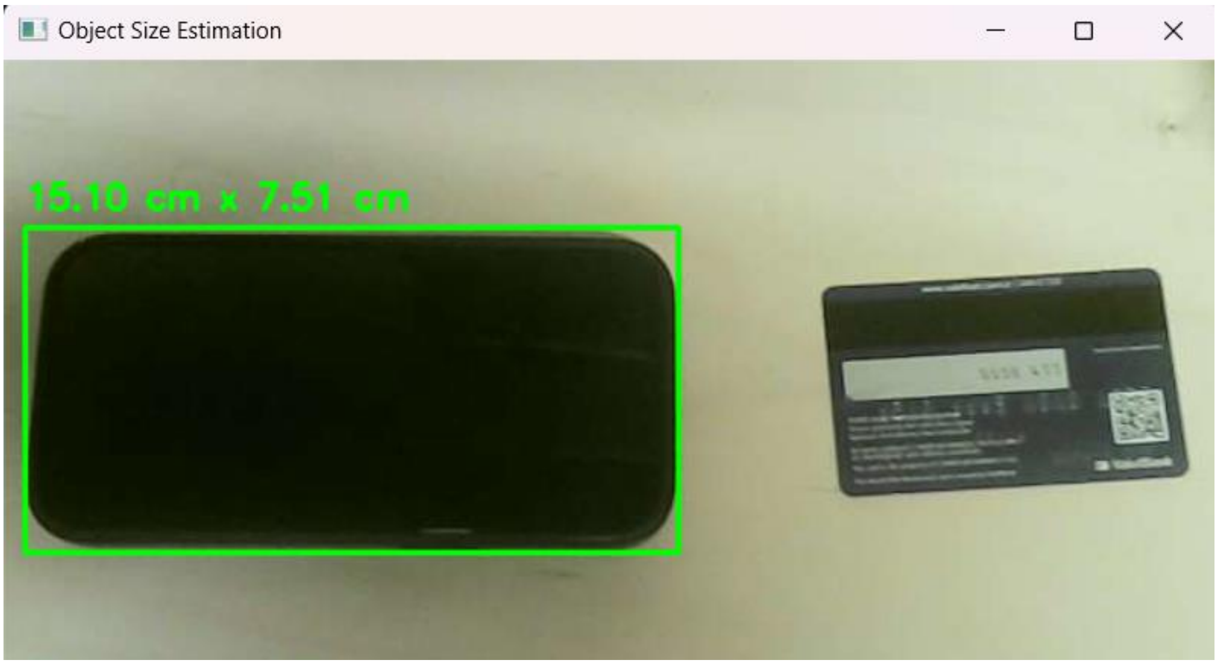
8. Conclusion

The Object Size Estimation System effectively applies computer vision techniques to measure objects in real-time using a known reference object. By leveraging OpenCV and NumPy, the system captures video input, detects contours, and calculates object dimensions based on a predefined scale. Its modular design allows flexibility for improvements and practical applications in inventory management, packaging, and automation.

While the system provides accurate measurements, its performance depends on proper reference object placement, lighting conditions, and camera angles. Perspective distortions can introduce minor errors, as the system assumes a planar view. Future improvements could include machine learning for automatic reference object detection, advanced camera calibration, and depth-sensing technology to enhance accuracy.

Overall, this system demonstrates a practical and adaptable approach to object measurement. With refinements, it could be extended for more complex applications in industrial and commercial environments.

9. Example Output



```
15.10 cm x 7.51 cm
```

The object used to calculate is an iPhone 15 and its dimensions are 147.6 mm and 71.6 mm. We can see from the output that the size estimation is pretty close to the real dimensions.