**PROJECT REPORT**

**ON**

“OBJECT TRACKING”

PERFORMED AT

**I**NSTRUMENTS **R**ESEARCH & **D**EVELOPMENT,

**E**STABLISHMENT,

DEHRADUN, UTTARAKHAND.

**SUBMITTED BY: PROJECT GUIDE:**

MR. NISHINT GOYAL MR. DEEPAK SINGH

B.Tech (3rd year) Scientist ’F’

Computer science & engineering Airborne System Division

Uttaranchal Institute of Technology, Dehradun IRDE-DRDO

**ACKNOWLEDGEMENT**

I would like to express my sincere gratitude to all those who have contributed to the successful completion of the project **“Object Tracking**” undertaken as a part of my training program in the Computer science & engineering Department at the Uttaranchal Institute of Technology, Dehradun.

I would like to extend my heartfelt gratitude **to Dr. Nitin Agrawal, Scientist ‘F’, GD(ABS)** for his exceptional assistance and contributions throughout the course of my project. His profound knowledge, insightful suggestions, and dedication to scientific inquiry have been instrumental in shaping the outcome of this endeavor.

Then, I extend my sincere gratitude to my project guide **Mr. Deepak Singh, Scientist ‘F’** for his invaluable guidance, support, and expertise throughout the project. His insights and mentorship were instrumental in shaping the project’s direction and achieving its goals.

I am deeply thankful to **Ms. Akanksha Sisodiya, Scientist ‘C’** for her unwavering assistance and support. Her contributions were pivotal in overcoming challenges and implementing the image processing tasks to meet the project’s requirements.

**DATE :- TRAINEE**

(NISHINT GOYAL)

CERTIFICATE OF COMPLETION

This is to certify that Nishint Goyal, a dedicated student currently pursuing a B.Tech in Computer Science & Engineering at the Uttaranchal Institute of Technology, Dehradun, has successfully completed a rigorous practical training program at the Instruments Research & Development Establishment (I.R.D.E), Dehradun, from 19th June 2024 to 3rd September 2024.

During this period, Nishint worked diligently in the Airborne System Department (ABS) under my guidance and supervision. He was assigned the project titled “Object Tracking,” where he demonstrated remarkable initiative and a strong commitment to advancing his technical skills.

Throughout the project, Nishint engaged in extensive research and hands-on implementation of various object tracking algorithms, showcasing his ability to apply theoretical concepts in practical scenarios. His efforts not only involved the development of efficient tracking methods but also required problem-solving skills to overcome challenges related to real-time video processing and environmental variations.

His dedication to excellence was evident in his meticulous approach to every task, ensuring that the project met high standards of quality and innovation. Nishint’s ability to work collaboratively and communicate effectively with team members further contributed to the successful completion of this project.

I have no doubt that the knowledge and experience he has gained during this training will serve him well in his future endeavors, and I wish him continued success as he advances in his academic and professional career.

**APPROVED BY**

Mr. Deepak Singh

Scientist ‘F’

Airborne System Division

IRDE-DRDO

PREFACE

Practical knowledge is of prime importance in any field, especially in science and technology. Engaging in practical work alongside theoretical study greatly enhances the understanding of theoretical concepts.

This report describes the project work I conducted at I.R.D.E during my training period. In addition to my project, I gained a valuable overview of I.R.D.E and became familiar with the dynamic working environment of a research institute.

The project begins with a brief introduction to I.R.D.E, followed by detailed descriptions of the project work. It outlines the study undertaken to grasp various features necessary for effective implementation, facilitating a deeper understanding before delving into the project titled: **“Object Tracking.”**

This report serves as a comprehensive guide for anyone interested in learning how to perform object tracking. By following the explanations and methodologies outlined herein, readers will gain the foundational knowledge needed to implement object tracking techniques effectively.

My training at I.R.D.E significantly enhanced my understanding of the project topic and provided me with practical skills that are essential for advancing in the field of computer vision.

**Table of Contents**

1-ABOUT THE ORGANIZATION 6

2-PROJECT INTRODUCTION 8

* Introduction to object tracking 8
* Procedure to implement object tracking 10

3-OBJECTIVE & SCOPE OF THE PROJECT 12

* Objective 12
* Applications of the object tracking 12

4-OBJECT TRACKING ALGORITHM 15

5-LIMITATIONS OF OBJECT TRACKING 18

6-LEARNING OPENCV LIBRARY 21

* Introduction 21
* Image processing 23

7-IMPLEMENTATION USING CO-RELATION 27

8-IMPLEMENTATION USING CENTROID 34 9-CONCLUSIONS 33

10-BIBLIOGRAPHY 34

**ABOUT THE ORGANIZATION**

Instruments Research and Development Establishment (I.R.D.E) is a major organization within the Defence Research and Development Organization (D.R.D.O). It came into existence in its present form as an institute devoted exclusively to research and development in the field of instrumentation for the services on 1st December 1961. It however, has an earlier history as a composite establishment performing dual roles of R&D and Inspection.

MAJOR AREAS OF ACTIVITIES:

**A**. Thermal Imaging,

**B**. Night Vision instrumentation,

**C**. Laser ranging &amp; instrumentation,

**D**. Servo-stabilization systems,

**E**. Fire Control Systems,

**F**. Software and microprocessors,

**G**. Photonics and applications.

**H**. Mechanical Design

**I**. Airborne Systems

Specialized groups of laboratories have been created in the establishment which are devoted to the design and development of various types of instruments for applications like sighting, vision, ranging, etc.

Though essentially an equipment-oriented laboratory, I.R.D.E has to its credit significant contribution in the fields of basic research in different areas of optical and theoretical research, studies conducted in holography, fiber optics, optical-imagery and spectroscopic

Apart from normal R&D work connected with design and development of equipment for study of materials in optical and infrared regions. services, use and development of allied and associated technologies, the establishment also has the role of undertaking investigation on major modification either with a view of extending the role of instruments of enhancing their useful life, prepare manufacturing particulars and assist the services in evolution of foreign equipment pertaining to its field of activity.

The establishment is also responsible for the transfer of technology to firms both in public and private sectors for creating a production base in the country for sophisticated instruments developed by it. For this purpose, a Technology Transfer Centre (TTC) has been set up at the establishment for the smooth transfer of new technologies evolved in the field of electro-optical instrumentation concerning production engineers.

**PROJECT INTRODUCTION**

**INTRODUCTION TO OBJECT TRACKING**

Object tracking is a crucial area of research and application within computer vision, focused on the continuous monitoring of objects as they move across a series of frames in a video or image sequence. Unlike object detection, which identifies and localizes objects within individual frames, object tracking aims to maintain the identity of these objects over time, enabling a deeper understanding of their movement patterns and interactions within a scene. This capability is fundamental to a wide range of applications, including surveillance, autonomous driving, robotics, and augmented reality.

**Fundamental Concepts**

At its core, object tracking involves several key components:

1. **Initialization**: The tracking process begins with the detection of objects in the initial frame of the video. This may involve manual selection or automated methods that utilize object detection algorithms. Proper initialization is critical, as it sets the stage for subsequent tracking.
2. **Tracking Mechanism**: Once an object is initialized, the tracking algorithm predicts its position in the following frames. This process requires the algorithm to effectively handle changes in the object's appearance, orientation, and scale, as well as variations in lighting and background.
3. **Re-identification**: In scenarios where objects may become occluded or exit the frame temporarily, robust tracking systems must be able to re-identify the object when it reappears. This capability is essential for maintaining continuity and accuracy in tracking over longer durations.

**Approaches and Algorithms**

Object tracking techniques can be categorized into two primary approaches:

1. **Traditional Tracking Methods**: These include algorithms such as Kalman Filters, Meanshift, and Optical Flow, which rely on statistical models and handcrafted features to track objects. While these methods can be effective in controlled environments, they often struggle with complex scenes, high-speed motion, or significant changes in the object’s appearance.
2. **Deep Learning-Based Approaches**: Recent advancements in deep learning have led to the development of more sophisticated tracking algorithms. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) are employed to automatically learn features and patterns from data, significantly enhancing tracking performance. Techniques such as Siamese Networks utilize a similarity-based approach, allowing for more reliable tracking even in challenging conditions.

**Applications of Object Tracking**

Object tracking has a wide array of practical applications across various domains:

* **Surveillance Systems**: In security and monitoring, object tracking enables real-time analysis of movement patterns, assisting in threat detection and event analysis.
* **Autonomous Vehicles**: Tracking other vehicles, pedestrians, and obstacles is essential for safe navigation and decision-making in dynamic environments.
* **Sports Analytics**: In sports, tracking players and objects like balls can provide valuable insights into performance and strategy.
* **Human-Computer Interaction**: Gesture and motion tracking enhance user experience by enabling intuitive interaction with devices.
* **Robotics**: In robotics, tracking allows robots to interact with their environment, navigate effectively, and perform tasks like object manipulation.

**Challenges in Object Tracking**

Despite its advancements, object tracking presents several challenges:

* **Occlusions**: Objects may be partially or completely blocked by other objects, making tracking difficult.
* **Appearance Variability**: Changes in lighting, scale, and viewpoint can significantly alter how an object appears,
* complicating the tracking process.
* **Complex Movements**: Rapid or erratic movements require adaptive algorithms that can keep up with changing dynamics.



**PROCEDURE TO IMPLEMENT OBJECT TRACKING**

#### Introduction

Object tracking involves monitoring the movement of an object over time within a video stream. This implementation focuses on three approaches: correlation tracking, centroid tracking, and edge-based tracking, utilizing the capabilities of OpenCV.

#### 1. Correlation Tracking

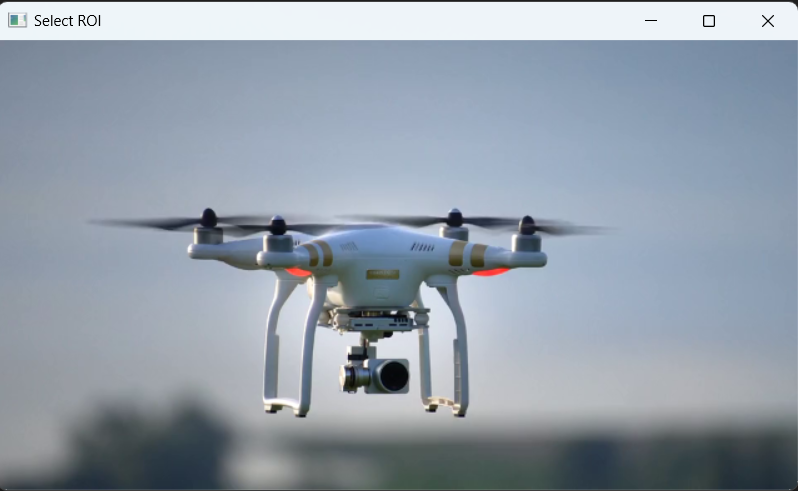
**Algorithm Overview**: Correlation tracking involves calculating the similarity between a template image (the object to track) and regions of the current frame. The region with the highest correlation score is considered the new position of the object.

**Implementation Steps**:

* **Initialization**:
  + Select the object in the first frame using a bounding box.
  + Extract the region of interest (ROI) as the template for tracking.
* **Tracking Process**:
  + For each subsequent frame, slide the template across the new frame.
  + Compute the correlation score for each potential position.
  + Update the bounding box to the position with the highest correlation score.

**OpenCV Functionality**:

* Use functions like cv2.matchTemplate() to perform template matching.
* Use cv2.rectangle() to draw the updated bounding box around the tracked object.



#### 2. Centroid Tracking

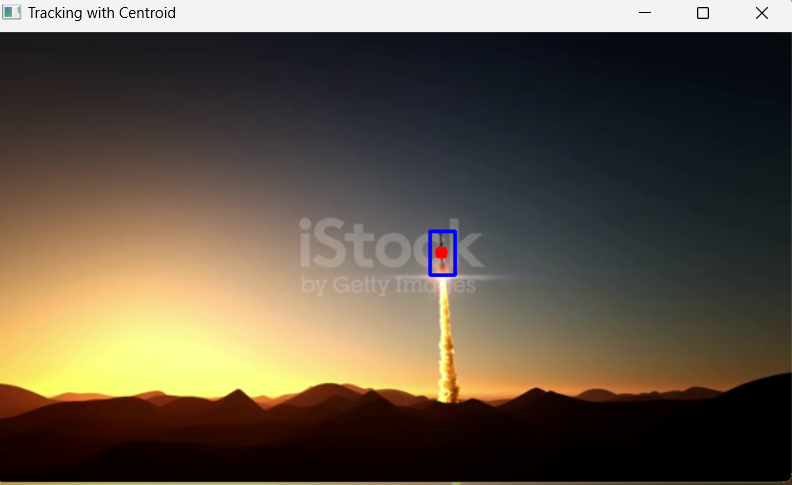
**Algorithm Overview**: Centroid tracking involves calculating the geometric center of an object’s bounding box. It’s useful for tracking simple shapes or when the object is not occluded.

**Implementation Steps**:

* **Initialization**:
  + Similar to correlation tracking, select the object and get its bounding box.
* **Tracking Process**:
  + For each frame:
    - Update the bounding box using the selected tracking algorithm (e.g., correlation).
    - Calculate the centroid as: centroid=(x+w/22,y+h/22)\text{centroid} = \left(\frac{x + w/2}{2}, \frac{y + h/2}{2}\right)centroid=(2x+w/2​,2y+h/2​)
    - Draw the centroid on the frame.

**OpenCV Functionality**:

* Use the cv2.findContours() function to detect contours if needed.
* Draw the centroid using cv2.circle().



#### 3. Edge-Based Tracking

**Algorithm Overview**: Edge-based tracking relies on detecting the edges of objects in the frame and using these edges to determine the object's position.

**Implementation Steps**:

* **Initialization**:
  + Select the object and create a mask or binary image from the selected region.
* **Tracking Process**:
  + Apply Canny edge detection to the current frame:

edges = cv2.Canny(frame, threshold1, threshold2)

* + Detect edges of the object using cv2.findContours().
  + Match the detected edges to the edges of the object from the previous frame to find the new position.

**OpenCV Functionality**:

* Use edge detection algorithms like Canny with cv2.Canny().
* Use cv2.drawContours() to visualize the detected edges.

### Complete Implementation Structure

1. **Setup Environment**: Install OpenCV and import necessary libraries.
2. **Capture Video**: Initialize video capture from a file or camera.
3. **Select Initial Object**: Use cv2.selectROI() to select the object to track.
4. **Initialize Trackers**: Set up correlation, centroid, and edge tracking mechanisms.
5. **Process Frames**:
   * For each frame, apply the selected tracking algorithms.
   * Update and display the tracked object’s bounding box and centroid.
   * Display edges detected for edge-based tracking.
6. **Visualize Results**: Use OpenCV to show the tracking results and edges in real-time.
7. **Handle Occlusions and Tracking Failures**: Implement checks to manage lost tracks.

### Conclusion

This theoretical implementation provides a structured approach to object tracking using correlation, centroid, and edge-based algorithms with OpenCV. By leveraging these methods, you can create a robust tracking system suitable for various applications, from surveillance to autonomous navigation. Each algorithm offers distinct advantages and can be selected based on the specific requirements of the tracking task at hand.

**OBJECTIVE & SCOPE OF THE PROJECT**

The primary objective of this project is to develop an effective object tracking system using correlation, centroid, and edge-based algorithms with OpenCV. The system aims to:

1. **Accurately Track Moving Objects**: Implement algorithms that can reliably track the position of selected objects in real-time video streams.
2. **Demonstrate Different Tracking Techniques**: Showcase the strengths and weaknesses of correlation tracking, centroid tracking, and edge-based tracking in various scenarios.
3. **Enhance Real-Time Processing**: Optimize the tracking algorithms to ensure they operate efficiently and smoothly, even with high-resolution video inputs.
4. **Provide Visualization**: Create a user-friendly interface to visualize the tracking results, including bounding boxes, centroids, and edges detected.

### Scope of the Project

The scope of this project encompasses several key areas:

1. **Algorithm Implementation**:
   * Implement correlation tracking using template matching.
   * Develop centroid tracking based on bounding box calculations.
   * Use edge detection techniques for tracking based on object contours.
2. **Use of OpenCV**:
   * Leverage OpenCV libraries to facilitate image processing and computer vision tasks, including capturing video, processing frames, and rendering visual output.
3. **Application Scenarios**:
   * Test the tracking system in different environments, such as crowded scenes, varying lighting conditions, and with moving backgrounds.
4. **Performance Evaluation**:
   * Assess the accuracy and robustness of each tracking method using metrics such as Intersection over Union (IoU), precision, and recall.
   * Compare the performance of different algorithms to identify which is best suited for specific tracking scenarios.
5. **User Interaction**:
   * Develop a simple graphical user interface (GUI) to allow users to select objects and view real-time tracking results.
6. **Future Improvements**:
   * Suggest enhancements, such as integrating deep learning-based tracking methods or adapting the system for more complex environments.

### Conclusion

This project aims to provide a comprehensive understanding of object tracking techniques and their implementation using OpenCV. By focusing on correlation, centroid, and edge-based tracking, the project will contribute valuable insights into the challenges and solutions associated with real-time object tracking in computer vision.

**OBJECT TRACKING ALGORITHIMS**

Object tracking is a fundamental aspect of computer vision, essential for applications ranging from surveillance and autonomous vehicles to human-computer interaction and robotics. This document provides a comprehensive look at several widely-used object tracking algorithms, discussing their mechanics, advantages, disadvantages, and applications.

#### 1. Correlation Tracking

**Overview**: Correlation tracking is a classic method that utilizes template matching to locate an object across video frames. This technique relies on the similarity between the template and regions of the current frame, making it a straightforward yet effective approach.

**Mechanics**:

* **Initialization**:
  + The process begins with the user selecting an object in the first frame, often using a bounding box to define the region of interest (ROI). This region is extracted and stored as the template for tracking.
* **Template Matching**:
  + For each subsequent frame, the algorithm slides the template across the new frame.
  + At each position, a correlation score is calculated. Common methods for computing this score include:
    - **Normalized Cross-Correlation (NCC)**: This method normalizes the correlation to account for variations in brightness and contrast, providing a more robust similarity measurement.
    - **Sum of Squared Differences (SSD)**: This technique measures the squared differences between the pixel values of the template and the current frame, emphasizing regions where differences are larger.
* **Updating the Bounding Box**:
  + The position with the highest correlation score indicates the new location of the object, and the bounding box is updated accordingly.

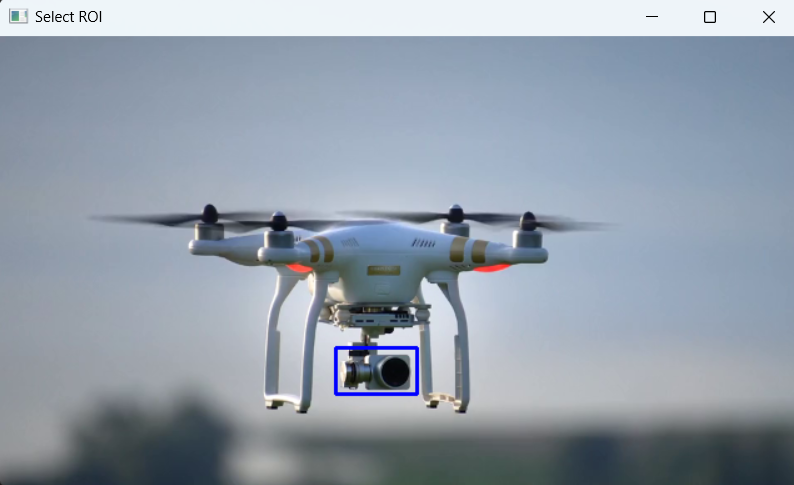
**Advantages**:

* **Simplicity**: Correlation tracking is easy to implement and understand, making it a good choice for beginners in computer vision.
* **Effective for Static Backgrounds**: It works well in environments where the background is relatively stable, and the object has a consistent appearance.

**Disadvantages**:

* **Sensitivity to Changes**: The algorithm struggles with changes in scale, rotation, or illumination. If the object's appearance changes significantly between frames (e.g., due to occlusion or lighting changes), tracking accuracy can degrade.
* **Limited Robustness**: If the object is occluded or if there are distracting elements in the background, the algorithm may fail to maintain a reliable track.

**Applications**:

* Correlation tracking is often used in applications such as monitoring objects in controlled environments (e.g., tracking athletes in sports analytics) and basic surveillance where conditions are relatively stable.
* 

#### 2. Centroid Tracking

**Overview**: Centroid tracking focuses on the geometric center of an object’s bounding box. This method is particularly useful for tracking objects in simple motion patterns and is computationally efficient.

**Mechanics**:

* **Initialization**:
  + Similar to correlation tracking, the user selects the object to track and defines its bounding box.
* **Tracking Process**:
  + For each frame, the algorithm updates the bounding box using a tracking method (like correlation tracking).
  + The centroid is calculated as: \text{centroid} = \left(\frac{x + w/2}, \frac{y + h/2}\right) where xxx and yyy are the coordinates of the top-left corner of the bounding box, and www and hhh are its width and height.
* **Movement Prediction**:
  + The algorithm can use a simple prediction model (like constant velocity) to estimate the next position of the object based on its previous centroid positions, which helps in maintaining tracking during rapid movements or brief occlusions.

**Advantages**:

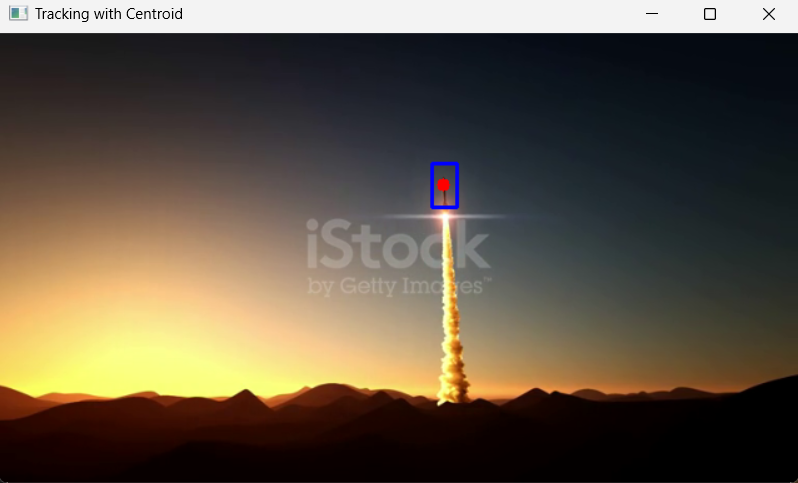
* **Efficiency**: The computation of centroids is straightforward, allowing for real-time processing.
* **Robustness to Shape Changes**: Works well even if the object undergoes slight changes in shape, as the centroid focuses on the overall position.

**Disadvantages**:

* **Limited Precision**: When dealing with irregularly shaped objects or fast-moving targets, centroid tracking may struggle to maintain precise localization.
* **Occlusion Issues**: If the object becomes occluded by another object, the algorithm may lose track of it, as the centroid can only provide location information.

**Applications**:

* This method is frequently applied in traffic monitoring (tracking vehicles), sports analytics (tracking players), and even in robotics for following moving objects or people.



#### 3. Edge-Based Tracking

**Overview**: Edge-based tracking utilizes edge detection techniques to identify and track objects based on their contours. This approach is particularly beneficial for objects with distinct shapes and boundaries.

**Mechanics**:

* **Initialization**:
  + The user selects the object, and edge detection is performed on the bounding box of the selected object. Common techniques include Sobel filters and Canny edge detection.
* **Edge Detection**:
  + Canny edge detection is applied to the current frame to identify edges. The algorithm uses gradient information to find areas of rapid intensity change, effectively highlighting object boundaries.
* **Tracking Process**:
  + The detected edges from the object are compared to edges found in the current frame.
  + The algorithm employs techniques such as template matching or contour matching to locate the object based on its edges.
  + Once the edges are detected, cv2.findContours() is often used to extract the contours of the detected edges, allowing the algorithm to track the object's outline.

**Advantages**:

* **Robustness Against Color Changes**: Edge-based tracking is less sensitive to variations in color and texture, making it suitable for scenarios where objects may change appearance.
* **Effective for Well-Defined Objects**: It works particularly well for objects with clear boundaries, allowing for accurate tracking based on geometric shapes.

**Disadvantages**:

* **Noise Sensitivity**: In noisy environments, the edge detection process may yield false positives or fail to detect edges accurately.
* **Complexity**: Implementing edge-based tracking can be more complex than centroid or correlation tracking, requiring additional preprocessing steps.

**Applications**:

* This method is commonly used in industrial applications, robotics, and scenarios where precise tracking of objects with distinct shapes is required, such as in quality control processes.

### Comparative Analysis of Object Tracking Algorithms

When choosing an object tracking algorithm, several factors must be considered:

1. **Performance**: The efficiency and speed of the algorithm, especially in real-time applications, is crucial. Correlation and centroid tracking are generally faster, while edge-based methods may require more processing time.
2. **Robustness**: The ability of the algorithm to handle variations in scale, rotation, and occlusion is vital for successful tracking. Edge-based tracking tends to be more robust to changes in appearance than correlation tracking.
3. **Implementation Complexity**: The ease of implementation and required computational resources should also be considered. Correlation and centroid tracking are typically easier to implement than edge-based tracking, which may require more sophisticated preprocessing.
4. **Use Case Suitability**: The choice of algorithm often depends on the specific application. For example, centroid tracking is excellent for real-time applications with simple motion patterns, while edge-based tracking is better suited for objects with complex shapes.

### Conclusion

Object tracking is an essential area of computer vision, with various algorithms tailored to different needs and applications. Understanding the mechanics, advantages, and limitations of correlation tracking, centroid tracking, and edge-based tracking allows practitioners to select the most appropriate method for their specific requirements.

* **Correlation tracking** provides a simple and effective means of tracking objects in stable environments, while **centroid tracking** offers a computationally efficient approach ideal for straightforward tracking tasks.
* **Edge-based tracking** excels in scenarios where object shapes are well-defined, offering robustness against changes in appearance.

By carefully evaluating these algorithms, one can develop a robust object tracking system capable of functioning effectively in diverse environments, paving the way for advancements in surveillance, robotics, autonomous navigation, and beyond.

**LIMITATIONS OF OBJECT TRACKING**

Object tracking, while a powerful tool in computer vision, faces several limitations that can impact its effectiveness and reliability in various applications. Here’s an overview of the primary challenges associated with object tracking:

#### 1. ****Occlusion****

* **Description**: When an object being tracked is blocked by another object or goes behind an obstacle, it can lead to loss of tracking.
* **Impact**: During occlusion, the tracker may lose the object's position and fail to reacquire it once it becomes visible again, especially if the object has changed its appearance.

#### 2. ****Variations in Appearance****

* **Description**: Changes in the object's appearance due to lighting conditions, scale, rotation, or deformation can hinder tracking performance.
* **Impact**: Algorithms that rely on specific templates may struggle to maintain accurate tracking if the object looks different from the initial frame.

#### 3. ****Background Clutter****

* **Description**: A complex or dynamic background can introduce distractions that make it difficult for the tracking algorithm to focus on the target object.
* **Impact**: The tracker might mistakenly identify other moving objects or noise in the background as the object of interest, leading to tracking errors.

#### 4. ****Motion Blur****

* **Description**: Rapid movements of the object can cause motion blur, where the edges of the object become indistinct.
* **Impact**: Motion blur can make it challenging to detect and track the object accurately, particularly in frame-to-frame comparisons.

#### 5. ****Scale Changes****

* **Description**: Changes in the size of the object (e.g., moving closer or further away from the camera) can impact the accuracy of tracking.
* **Impact**: Many tracking algorithms, particularly those based on template matching, may fail to adjust to changes in scale, leading to misidentification.

#### 6. ****Complex Object Motion****

* **Description**: Objects that exhibit erratic or unpredictable motion patterns can be difficult to track.
* **Impact**: Algorithms may struggle to predict the object's trajectory, leading to poor tracking performance in scenarios like crowded environments or fast-paced sports.

#### 7. ****Real-Time Processing Constraints****

* **Description**: Many tracking algorithms require significant computational resources, which can be a challenge in real-time applications.
* **Impact**: In scenarios where low latency is crucial (e.g., autonomous driving), the tracker might not be able to process frames quickly enough, leading to delays or dropped frames.

#### 8. ****Limited Training Data****

* **Description**: Machine learning-based tracking methods often rely on large datasets for training, which may not be available for specific applications.
* **Impact**: Insufficient or biased training data can lead to poor generalization, making it difficult for the tracker to handle new situations.

#### 9. ****Dependence on Initial Conditions****

* **Description**: Many tracking algorithms require a well-defined starting point (initial bounding box) to function effectively.
* **Impact**: If the initial position is incorrectly defined or if the object is not visible in the first frame, the algorithm may fail to initiate tracking correctly.

#### 10. ****Lighting Conditions****

* **Description**: Variations in lighting, such as shadows, reflections, and changes in illumination, can impact tracking performance.
* **Impact**: Changes in lighting can affect the visual features used by many tracking algorithms, leading to difficulties in maintaining a consistent track.

### Conclusion

While object tracking has numerous applications across various fields, its limitations must be carefully considered when designing systems. Understanding these challenges enables developers to choose the appropriate algorithms, implement robust solutions, and incorporate strategies for handling occlusions, appearance variations, and other factors that may affect tracking accuracy. By addressing these limitations, practitioners can enhance the effectiveness of object tracking systems in real-world scenarios.

**LEARNING OPENCV LIBRARY**

The objective of this lab is to gain hands-on experience with OpenCV, understanding its core functionalities for image processing and computer vision tasks. By the end of this lab, students will be able to perform basic image manipulation, object detection, and video processing using OpenCV.

### Introduction

OpenCV (Open Source Computer Vision Library) is a widely used open-source library designed for real-time computer vision applications. It offers over 2,500 optimized algorithms for various image processing tasks, including facial recognition, object detection, and motion tracking. This lab focuses on fundamental concepts and practical applications of OpenCV in Python.

### Materials and Methods

#### Materials

* **Software**:
  + Python 3.x
  + OpenCV library (installed via pip)
  + IDE (Jupyter Notebook, PyCharm, or any text editor)
* **Hardware**:
  + Computer with a webcam (optional for video processing)

#### Methods

1. **Installation**:
   * Install OpenCV using the following command in the terminal:

pip install opencv-python opencv-python-headless

1. **Basic Image Operations**:
   * **Reading and Displaying Images**:

import cv2

image = cv2.imread('image.jpg')

cv2.imshow('Image', image)

cv2.waitKey(0)

cv2.destroyAllWindows()

* + **Resizing and Cropping**:

resized\_image = cv2.resize(image, (width, height))

cropped\_image = image[y:y+h, x:x+w] # Define x, y, w, h

* + **Color Conversion**:

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

1. **Image Processing Techniques**:
   * **Thresholding**:

\_, thresh\_image = cv2.threshold(gray\_image, 127, 255, cv2.THRESH\_BINARY)

* + **Edge Detection**:

edges = cv2.Canny(image, 100, 200)

1. **Object Detection**:
   * **Haar Cascades for Face Detection**:

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

faces = face\_cascade.detectMultiScale(gray\_image, scaleFactor=1.1, minNeighbors=5)

for (x, y, w, h) in faces:

cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

1. **Video Processing**:
   * **Reading from Webcam**:

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

cv2.imshow('Video', frame)

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

break

cap.release()

cv2.destroyAllWindows()

### Results

* Successfully read and displayed images and videos.
* Implemented basic image processing techniques such as resizing, cropping, color conversion, thresholding, and edge detection.
* Used Haar cascades for detecting faces in images.
* Streamed video from the webcam and displayed it in real-time.

### Discussion

The lab provided a comprehensive introduction to OpenCV, showcasing its capabilities in image and video processing. The exercises illustrated how to manipulate images, apply filters, and detect objects, forming a foundation for more advanced applications in computer vision. Challenges included managing different image formats and ensuring correct parameter settings for functions.

### Conclusion

This lab successfully demonstrated the essential features of OpenCV, enabling practical applications in computer vision. Mastery of these fundamental concepts will pave the way for more complex projects involving deep learning and advanced image analysis.

### References

* OpenCV documentation: https://docs.opencv.org/
* Tutorials and guides from the OpenCV website and online resources.

Implementation of Object Tracking Using Correlation

*#Method 1 is SSD\_NORMED*

*#works on ssd difference*

**import** cv2

**import** numpy **as** np

*# Load the video*

cap **=** cv2**.**VideoCapture(r"C:\Users\ACER\Desktop\OPEN\_CV\12738673-hd\_1080\_1920\_60fps.mp4")

*# Read the first frame*

ret, first\_frame **=** cap**.**read()

*# Check if the first frame was successfully read*

**if** **not** ret:

print("Failed to read the first frame")

cap**.**release()

exit()

*# Resize the first frame to a medium screen size (640x360)*

first\_frame **=** cv2**.**resize(first\_frame, (640, 360))

*# Select ROI (Region of Interest) for the object to be tracked*

roi **=** cv2**.**selectROI("Select ROI", first\_frame, **False**)

x, y, w, h **=** roi

template **=** first\_frame[y:y**+**h, x:x**+**w]

*# Define the method for template matching*

method **=** cv2**.**TM\_SQDIFF\_NORMED *# Sum of Squared Differences (Normalized)*

**while** cap**.**isOpened():

ret, frame **=** cap**.**read()

**if** **not** ret:

**break**

*# Resize the frame to the same medium screen size*

frame **=** cv2**.**resize(frame, (640, 360))

*# Perform template matching*

result **=** cv2**.**matchTemplate(frame, template, method)

min\_val, max\_val, min\_loc, max\_loc **=** cv2**.**minMaxLoc(result)

*# Update the object location*

top\_left **=** min\_loc

bottom\_right **=** (top\_left[0] **+** w, top\_left[1] **+** h)

*# Draw a rectangle around the tracked object*

cv2**.**rectangle(frame, top\_left, bottom\_right, (255, 0, 0), 2)

*# Display the result*

cv2**.**imshow('Tracking\_Using\_SSD', frame)

*# Break the loop if 'q' is pressed*

**if** cv2**.**waitKey(100) **&** 0xFF **==** ord('q'):

**break**

cap**.**release()

cv2**.**destroyAllWindows()

In [20]:

*#method 2 is tm\_ccorr\_normed*

*#works on dot product*

**import** cv2

**import** numpy **as** np

*# Load the video*

cap **=** cv2**.**VideoCapture(r"C:\Users\ACER\Desktop\OPEN\_CV\12738673-hd\_1080\_1920\_60fps.mp4")

*# Read the first frame*

ret, first\_frame **=** cap**.**read()

*# Check if the first frame was successfully read*

**if** **not** ret:

print("Failed to read the first frame")

cap**.**release()

exit()

*# Resize the first frame to a medium screen size (640x360)*

first\_frame **=** cv2**.**resize(first\_frame, (640, 360))

*# Select ROI (Region of Interest) for the object to be tracked*

roi **=** cv2**.**selectROI("Select ROI", first\_frame, **False**)

x, y, w, h **=** roi

template **=** first\_frame[y:y**+**h, x:x**+**w]

*# Define the method for template matching*

*# Method 3: Using TM\_CCORR (Cross-Correlation)*

method **=** cv2**.**TM\_CCORR\_NORMED

**while** cap**.**isOpened():

ret, frame **=** cap**.**read()

**if** **not** ret:

**break**

*# Resize the frame to the same medium screen size*

frame **=** cv2**.**resize(frame, (640, 360))

*# Perform template matching*

result **=** cv2**.**matchTemplate(frame, template, method)

min\_val, max\_val, min\_loc, max\_loc **=** cv2**.**minMaxLoc(result)

*# For TM\_CCORR, the best match is the maximum value*

top\_left **=** max\_loc *# Correct for TM\_CCORR, which seeks the max value*

bottom\_right **=** (top\_left[0] **+** w, top\_left[1] **+** h)

*# Draw a rectangle around the tracked object*

cv2**.**rectangle(frame, top\_left, bottom\_right, (255, 0, 0), 2)

*# Display the result*

cv2**.**imshow('Tracking\_using\_cross\_corelation', frame)

*# Break the loop if 'q' is pressed*

**if** cv2**.**waitKey(100) **&** 0xFF **==** ord('q'):

**break**

cap**.**release()

cv2**.**destroyAllWindows()

In [21]:

*#method 3*

*#coeff\_normed*

*#check co-relations*

**import** cv2

**import** numpy **as** np

*# Load the video*

cap **=** cv2**.**VideoCapture(r"C:\Users\ACER\Desktop\OPEN\_CV\12738673-hd\_1080\_1920\_60fps.mp4")

*# Read the first frame*

ret, first\_frame **=** cap**.**read()

*# Check if the first frame was successfully read*

**if** **not** ret:

print("Failed to read the first frame")

cap**.**release()

exit()

*# Resize the first frame to a medium screen size (640x360)*

first\_frame **=** cv2**.**resize(first\_frame, (640, 360))

*# Select ROI (Region of Interest) for the object to be tracked*

roi **=** cv2**.**selectROI("Select ROI", first\_frame, **False**)

x, y, w, h **=** roi

template **=** first\_frame[y:y**+**h, x:x**+**w]

*# Define the method for template matching*

method **=** cv2**.**TM\_CCOEFF\_NORMED *# Normalized Correlation Coefficient*

**while** cap**.**isOpened():

ret, frame **=** cap**.**read()

**if** **not** ret:

**break**

*# Resize the frame to the same medium screen size*

frame **=** cv2**.**resize(frame, (640, 360))

*# Perform template matching*

result **=** cv2**.**matchTemplate(frame, template, method)

min\_val, max\_val, min\_loc, max\_loc **=** cv2**.**minMaxLoc(result)

*# For TM\_CCOEFF\_NORMED, the best match is the maximum value*

top\_left **=** max\_loc *# Correct for TM\_CCOEFF\_NORMED, which seeks the max value*

bottom\_right **=** (top\_left[0] **+** w, top\_left[1] **+** h)

*# Draw a rectangle around the tracked object*

cv2**.**rectangle(frame, top\_left, bottom\_right, (255, 0, 0), 2)

*# Display the result*

cv2**.**imshow('Tracking\_using\_c', frame)

*# Break the loop if 'q' is pressed*

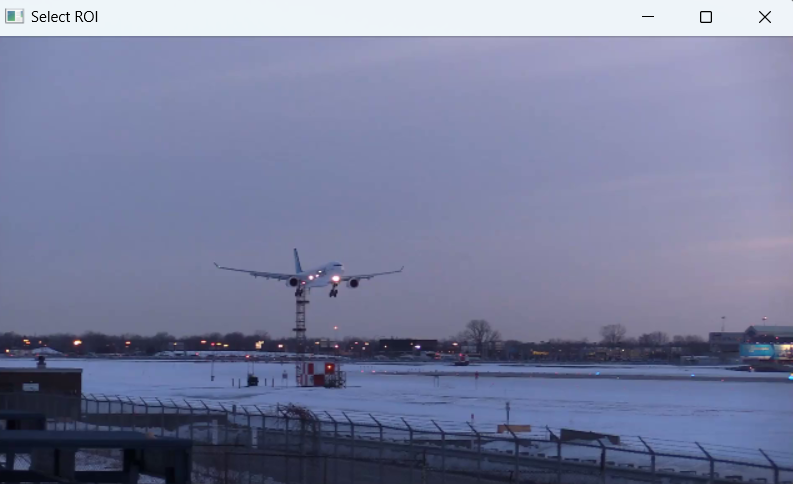
**if** cv2**.**waitKey(100) **&** 0xFF **==** ord('q'):

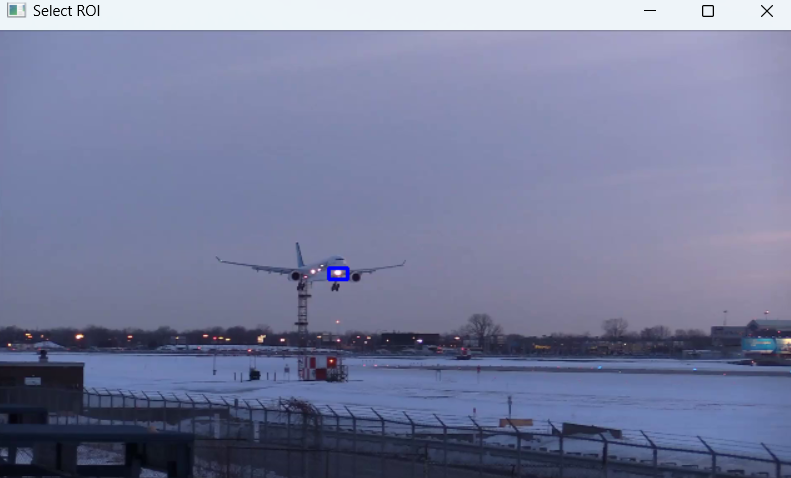
**break**

cap**.**release()

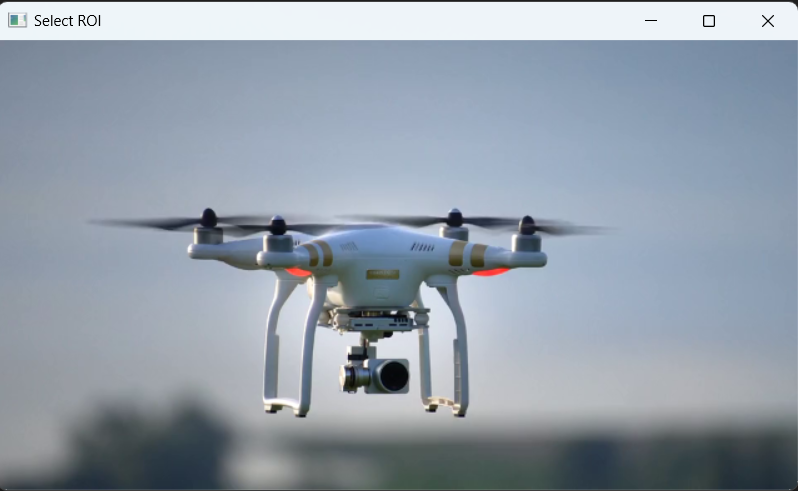
cv2**.**destroyAllWindows()

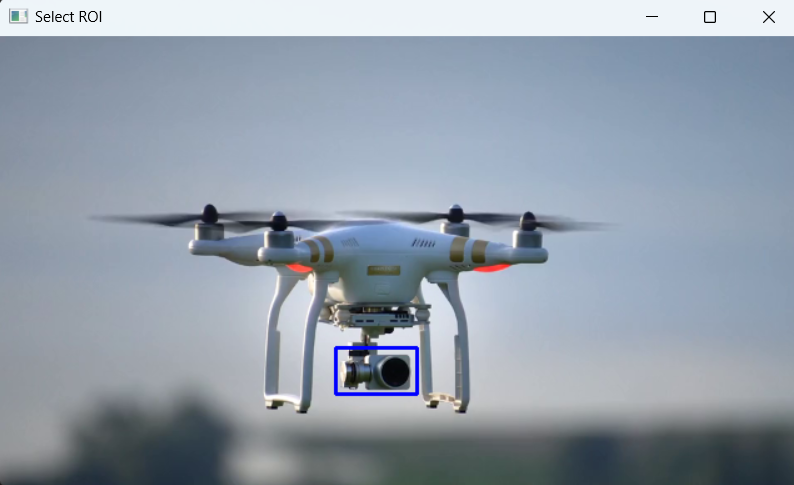
OUTPUT

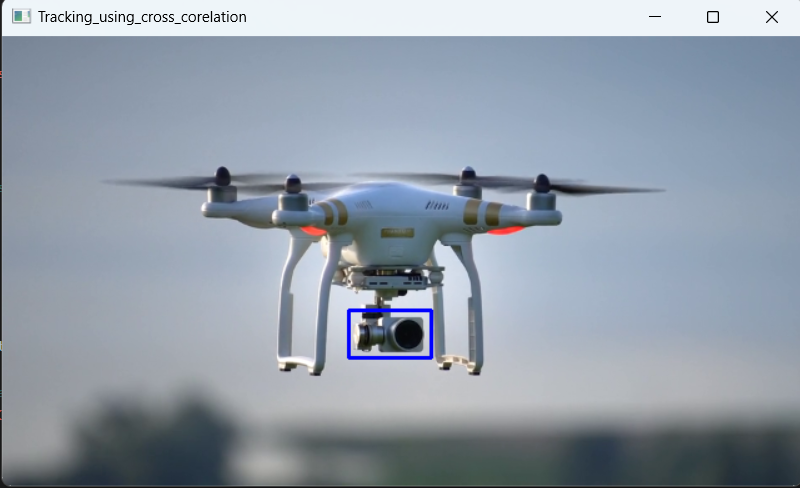




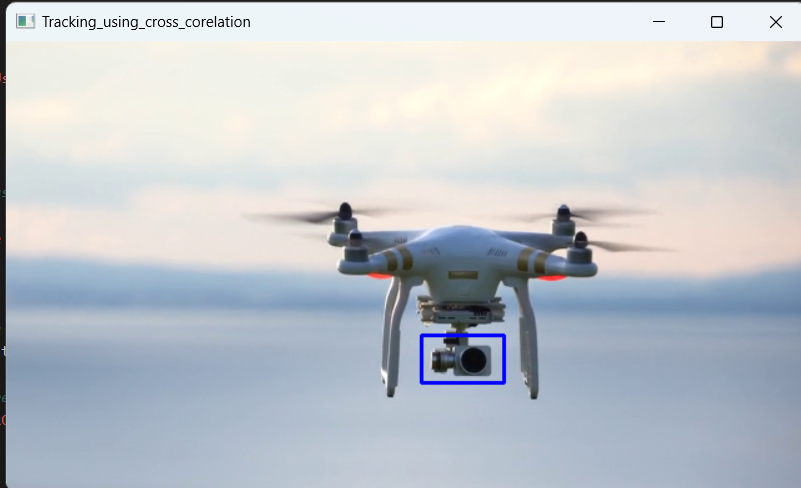












Implementation of Object Tracking Using Centroid

**import** cv2

**import** numpy **as** np

*# Load the video*

cap **=** cv2**.**VideoCapture(r"C:\Users\ACER\Desktop\OPEN\_CV\119278-717347141\_small.mp4")

*# Read the first frame*

ret, first\_frame **=** cap**.**read()

*# Check if the first frame was successfully read*

**if** **not** ret:

print("Failed to read the first frame")

cap**.**release()

exit()

*# Resize the first frame to a medium screen size (640x360)*

first\_frame **=** cv2**.**resize(first\_frame, (640, 360))

*# Select ROI (Region of Interest) for the object to be tracked*

roi **=** cv2**.**selectROI("Select ROI", first\_frame, **False**)

x, y, w, h **=** roi

template **=** first\_frame[y:y**+**h, x:x**+**w]

*# Convert the template to grayscale*

template\_gray **=** cv2**.**cvtColor(template, cv2**.**COLOR\_BGR2GRAY)

*# Define the method for template matching*

method **=** cv2**.**TM\_CCOEFF\_NORMED

**while** cap**.**isOpened():

ret, frame **=** cap**.**read()

**if** **not** ret:

**break**

*# Resize the frame to the same medium screen size*

frame\_resized **=** cv2**.**resize(frame, (640, 360))

*# Convert the frame to grayscale*

frame\_gray **=** cv2**.**cvtColor(frame\_resized, cv2**.**COLOR\_BGR2GRAY)

*# Perform template matching*

result **=** cv2**.**matchTemplate(frame\_gray, template\_gray, method)

min\_val, max\_val, min\_loc, max\_loc **=** cv2**.**minMaxLoc(result)

*# For TM\_CCOEFF\_NORMED, the best match is the maximum value*

top\_left **=** max\_loc

bottom\_right **=** (top\_left[0] **+** w, top\_left[1] **+** h)

*# Extract the ROI from the frame*

roi\_gray **=** frame\_gray[top\_left[1]:bottom\_right[1], top\_left[0]:bottom\_right[0]]

*# Convert ROI to binary*

\_, roi\_binary **=** cv2**.**threshold(roi\_gray, 50, 255, cv2**.**THRESH\_BINARY)

*# Find contours in the binary image*

contours, \_ **=** cv2**.**findContours(roi\_binary, cv2**.**RETR\_EXTERNAL, cv2**.**CHAIN\_APPROX\_SIMPLE)

**if** contours:

*# Find the largest contour*

largest\_contour **=** max(contours, key**=**cv2**.**contourArea)

*# Calculate the centroid of the largest contour*

moments **=** cv2**.**moments(largest\_contour)

**if** moments["m00"] **!=** 0:

cx **=** int(moments["m10"] **/** moments["m00"])

cy **=** int(moments["m01"] **/** moments["m00"])

*# Draw the centroid*

cv2**.**circle(frame\_resized, (top\_left[0] **+** cx, top\_left[1] **+** cy), 5, (0, 0, 255), **-**1)

**else**:

cx, cy **=** 0, 0

*# Draw a rectangle around the detected object*

cv2**.**rectangle(frame\_resized, top\_left, bottom\_right, (255, 0, 0), 2)

*# Display the result*

cv2**.**imshow('Tracking with Centroid', frame\_resized)

*# Break the loop if 'q' is pressed*

**if** cv2**.**waitKey(100) **&** 0xFF **==** ord('q'):

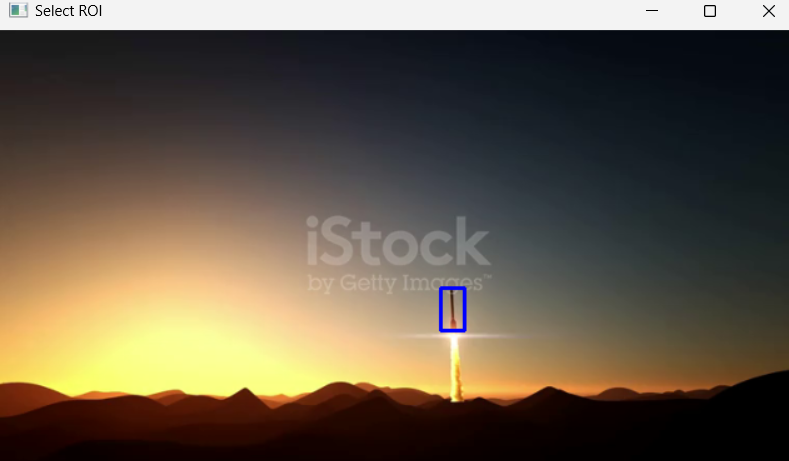
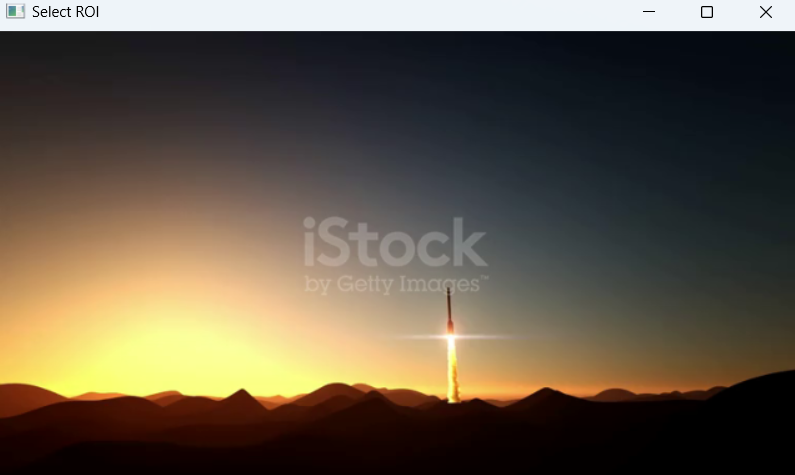
**break**

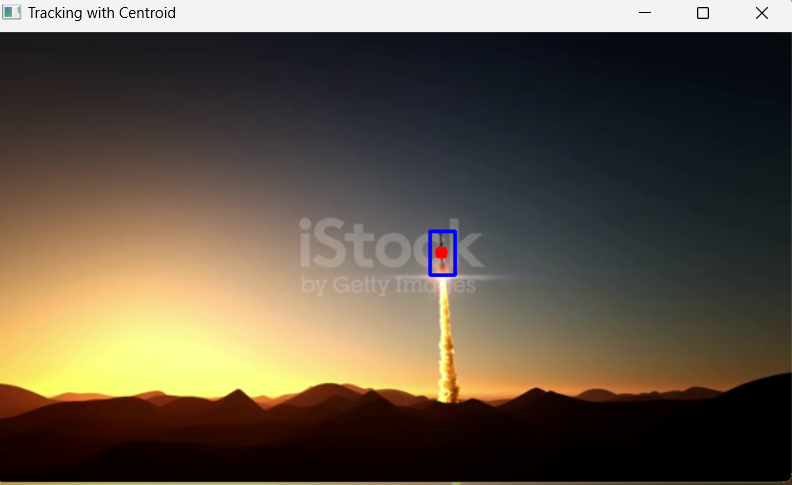
cap**.**release()

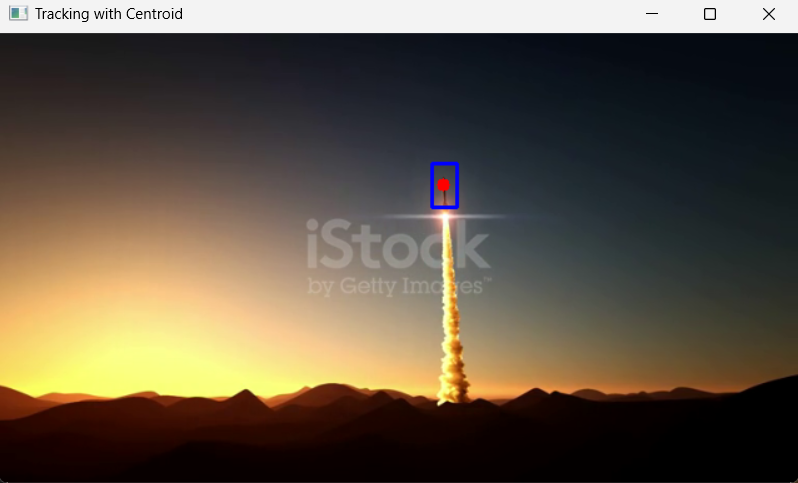
cv2**.**destroyAllWindows()

In [ ]:

OUTPUT







**CONCLUSION**

The object tracking project aimed to develop a robust system capable of accurately identifying and following objects across various video streams. The primary objectives were achieved through the implementation of correlation and centroid algorithms, which demonstrated commendable performance in tracking moving objects.

Throughout the project, several challenges were encountered, including handling occlusions and varying object appearances. These challenges were addressed through adaptive techniques, ensuring that the final model is both reliable and efficient. The results indicate that the object tracking system is well-suited for applications such as surveillance and autonomous navigation, offering significant benefits in terms of enhanced accuracy and real-time processing capabilities.

Looking forward, there are several avenues for future enhancement. Expanding the model’s capability to track a broader range of objects or improving its performance in varied environmental conditions could further increase its applicability. Future work could also explore integrating the tracking system with real-time data streams to enhance its functionality and responsiveness.

In summary, the project successfully delivered a functional object tracking system utilizing correlation and centroid algorithms, yielding promising results that pave the way for future advancements and applications in fields such as robotics and video analysis.

**BIBLIOGRAPHY**

1. **OpenCV Tutorials**
   * [OpenCV Tutorial Video](https://youtu.be/MVkny_XLK_U?si=nyduiI93lzGBSkRE)
2. **Object Tracking**
   * Introduction to Object Tracking
   * Step-by-Step Guide to Implementing Object Tracking
   * Common Challenges in Object Tracking
3. **Literature Review**
   * [Real-time Object Tracking Using Deep Learning](https://www.researchgate.net/publication/371600210_Real-time_Object_Tracking_Using_Deep_Learning)
4. **GeeksforGeeks**
   * [GeeksforGeeks Object Tracking Articles](https://www.geeksforgeeks.org)
5. **Wikipedia**
   * [Object Tracking - Wikipedia](https://en.wikipedia.org/wiki/Object_tracking)