VISVESVARAYA TECHNOLOGICAL UNIVERSITY

**BELGAVI, KARNATAKA -590 018**

 **A Minor Project Report on**

**“object size measurement in opencv ”**

***Submitted in partial fulfillment for the Computer Graphics and Image Processing Laboratory [21CSL66] Course of Sixth Semester of Bachelor of Engineering in Computer Science & Engineering during the academic year 2023-24.***

**By**

|  |  |
| --- | --- |
| **AFREEN SUHAN** | **4MN21CS001** |
| **SADAF SULTANA** | **4MN21CS038** |
| **YUKTHA SARODE J** | **4MN21CS056** |

**Under the Guidance of**

**Prof. Bharath Bharadwaj B S**

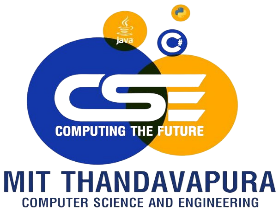
**Assistant Professor**

**Dept. of CS&E**

**MIT Thandavapura**



**2023 – 24**



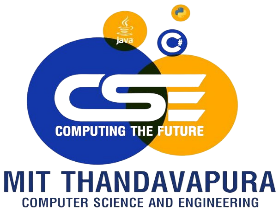
**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**MAHARAJA INSTITUTE OF TECHNOLOGY THANDAVAPURA**

**NH 766, NANJANGUD TALUK, MYSURU – 571302**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**MAHARAJA INSTITUTE OF TECHNOLOGY THANDAVAPURA**



**CERTIFICATE**

*Certified that the minor project work entitled “****OBJECT SIZE MEASUREMENT IN OPENCV****” is a bonafide work carried out by* ***AFREEN SUHAN*** *(4MN21CS001),* ***SADAF SULTANA*** *(4MN21CS038) &* ***YUKTHA SARODE J*** *(4MN21CS056) for the course* ***Computer Graphics and Image Processing Laboratory*** *with course code* ***21CSL66*** *of Sixth Semester in Computer Science & Engineering under Visvesvaraya Technological University, Belagavi during academic year* ***2023-24****.*

*It is certified that all corrections/suggestions indicated for Internal Assignment have been incorporated in the report.* *The report has been approved as it satisfies the course requirements.*

|  |  |  |
| --- | --- | --- |
| \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_  Signature of Lab Staff In-Charge  **Prof. Bharath Bharadwaj B S**  Assistant Professor  Dept. of CS&E  MIT Thandavapura |  | \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_  Signature of the HoD  **Dr. Ranjit K N**  Associate Professor & Head  Dept. of CS&E  MIT Thandavapura |

**External viva**

**Name of the Examiners Signature with date**

1)…………………………………………………………………………………….

2)…………………………………………………………………………………….

**ACKNOWLEDGEMENT**

It is the time to acknowledge all those who have extended their guidance, inspiration and their whole hearted co-operation all along our project work.

We are grateful to **Dr. Y T Krishne Gowda**, Principal, MIT Thandavapura, **Dr.H K Chethan,** Professor and Mentor, CS&E, MIT Thandavapura and **Dr.Ranjit K N**, Associate Professor and Head, CS&E, MIT Thandavapura for having provided us academic environment which nurtured our practical skills contributing to the success of our project.

We would like to sincerely thank our project guide **Prof. Bharath Bharadwaj B S**, Assistant Professor, CS&E, MIT Thandavapura for providing relevant information, valuable guidance and encouragement to complete this project.

We wish to place a deep sense of gratitude to all Teaching and Non-Teaching staffs of Computer Science and Engineering Department for whole-hearted guidance and constant support without which this endeavor would not have been possible.

Our gratitude will not be complete without thanking our parents and also our friends, who have been a constant source of support and aspirations.

|  |
| --- |
|  |
| **AFREEN SUHAN**  **[4MN21CS001]** |
|  |
| **SADAF SULTANA**  **[4MN21CS038]** |
|  |
| **YUKTHA SARODE J**  **[4MN21CS056]** |

**ABSTRACT**

In this project, Accurate measurement of object sizes is crucial in various fields such as industrial automation, robotics, and computer vision. This project explores the implementation of object size measurement using OpenCV, a powerful computer vision library. The objective is to develop a system that can automatically determine the size of objects captured by a camera in real-time or from static images.

The project involves several key steps: image acquisition, preprocessing, object detection, contour detection, and size calculation. OpenCV provides robust tools for image processing and computer vision tasks, making it suitable for this application. Techniques such as thresholding, edge detection, and contour analysis are employed to isolate objects of interest and compute their dimensions accurately.

The system is implemented using Python, leveraging OpenCV’s functionalities to process images, detect objects, and measure their sizes. A graphical user interface (GUI) is developed to facilitate user interaction, allowing for easy input of images and displaying the measured dimensions.

Experimental results demonstrate the effectiveness of the proposed method in accurately measuring object sizes under various conditions. The project contributes to advancing the field of computer vision applications by providing a practical framework for object size measurement using readily available tools and libraries.

In these days of the 4th industrial revolution, real-time object detection and dimensioning is an important aspect from an industrial point of view. These are requisite topics of computer vision problems. This project presents an augmented technique for detecting objects and computing their real-time measurements from an IoT video device such as a webcam. We have suggested an object measurement technique in real-time using AI and IoT technologies like OpenCV libraries and webcam respectively. OpenCV includes many libraries and algorithms that are used.

Object size measurement has become very vital in today’s world, it has widened its applications in various fields; Measurement is not only crucial in science and the chemical industry but also in farming, engineering, building, manufacturing, industries, business, and many more professions and pursuits technique that uses a readily measure an object’s overall size. Especially in the context of the COVID-19 pandemic, where contactless measurements have gained significance to reduce the risk of virus transmission which is aided by this techinique.

This paper presents a comprehensive approach for object size measurement using the OpenCV library, a widely adopted computer vision tool. The proposed method leverages state-of-the-art image processing techniques to provide reliable and precise object size estimates. The method utilizes OpenCV’s extensive functionality for image manipulation and object recognition, the calibration process necessary for converting pixel measurements into real-world units. Calibration methods based on known reference objects and camera parameters are explored to ensure accurate size estimation. The program initially detects the object that interests it in the picture and then applies an array of geometrical procedures to determine its borders and dimensions.

This project created an accurate, dependable, and cost-effective method for measuring item measures (distance, width, and height) in the real world employing only a camera system. Experimental results are presented to validate the proposed approach’s performance under various conditions, including object types, sizes, and environmental factors. The comparative analysis with existing methods demonstrates the superiority of the OpenCV-based approach in terms of accuracy and computational efficiency.

**CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **SL. No.** | | **Index** | **Page No.** |
| **1** |  | **INTRODUCTION** | **1-4** |
|  | 1.1 | Aim of the Project | 3 |
|  | 1.2 | Overview of the Project | 3 |
|  | 1.3 | Outcome of the Project | 4 |
| **2** |  | **DESIGN AND IMPLEMENTATION** | **5-13** |
|  | 2.1 | Algorithm | 8 |
|  | 2.2 | Flowchart | 11 |
|  | 2.3 | OpenCV API’s Used with Description | 12 |
| **3** |  | **RESULT ANALYSIS** | **14-17** |
|  | 3.1 | Snapshots | 14 |
|  | 3.2 | Discussion | 17 |
|  |  | **CONCLUSION AND FUTURE ENHANCEMENT** | **18-19** |
|  |  | **REFERENCES** |  |
|  |  | **APPENDIX A – Source Code** | **A1-A5** |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **SL. No.** | **Index** | **Page No.** |
| 1.1 | Library organization for OpenCV | 5 |
| 2.1 | Flow chart demonstrating working of object size measurement | 11 |
| 2.2 | Steps in object measurement | 11 |
| 3.1 | Snap Shot measuring size of object placed at bottom of image | 14 |
| 3.2 | Snap Shot measuring size of object placed at top of image | 14 |
| 3.3 | Snap Shot measuring size of object placed at right of image | 15 |
| 3.4 | Snap Shot measuring size of object placed at left of image | 15 |
| 3.5 | Measuring size of logo | 16 |
| 3.6 | Measuring size of camera | 16 |
| 3.7 | Measuring size of switch board | 17 |
| 3.8 | Measuring size of clock | 17 |
|  |  |  |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **SL. No.** | **Index** | **Page No.** |
| 2.1 | OpenCV API’S | 12-13 |

**CHAPTER – 1**

**INTRODUCTION**

“ Real- Time Object Dimension ” is a program that can be used to describe real- time object’s confines. There aren't numerous real- time object dimension models and this prototype can be used tremendously further. This is an essential content of computer vision problems. As stated, this design presents a fashion for calculating the measures in real-time from images. To explain it’s working it principally uses a webcam and a white paper background to descry the object. After detecting the object, it displays its confines in specified measuring units at real time. In the perpetration of the proposed fashion, we designed a system that used OpenCV software library. Some advantages of using this methodology are that it's veritably useful in the artificial field, it simplifies mortal work, and numerous further which are noted below in the advantages and disadvantages section. To calculate the size of each object, the prerequisite is that we need to determine the reference object. In this case, it is, plain white paper. After that, the confines of the objects inside the reference are measured or it'll be calculated and hence the size of the object is displayed.

In recent years, advancements in computer vision have revolutionized various industries by enabling automated analysis and measurement of objects in images and videos. Object size measurement, a fundamental task in fields such as manufacturing, healthcare, and robotics, plays a crucial role in quality control, process automation, and object recognition systems. This project focuses on developing a method to accurately measure the size of objects using OpenCV (Open Source Computer Vision Library), a popular tool for real-time computer vision applications.

The ability to measure the size of objects from images or video streams is essential for tasks ranging from determining the dimensions of products on an assembly line to monitoring the growth of cells in biomedical research. Traditional methods often involve manual measurements or specialized hardware, which can be time-consuming, prone to errors, and limited in scalability. Computer vision techniques offer a promising alternative, allowing for automated, precise, and efficient measurement of object dimensions.

Because of stereo vision, the human eye can roughly estimate the gap and size of an object. We proposed employing a stereo vision system to properly assess the gap and size (height and breadth) of the article visible during this project. Identifying the scale of an object is extremely useful in building systems or applications, particularly in autonomous system navigation. Many recent studies have begun to use multiple vision sensors or cameras for a spread of applications, including 3D picture generation, occlusion detection, and so on. Because cameras are now the lowest and straightforward to deploy and use, multiple camera systems are getting more prevalent. Item recognition on stereo pictures, blob extraction, distance and size calculation, and object identification are all a part of the proposed measurement system.

The system also incorporates a rapid algorithm, with real-time measurement. it is a lot easier to calibrate and might provide you with a more precise result. the subsequent are the goals of this project's work: -Pre-processing of images, object segmentation, reference object, and computation of results.

OpenCV provides a comprehensive suite of algorithms and tools for image processing and computer vision tasks. Leveraging its capabilities, this project aims to implement a system that can automatically detect objects in images, extract relevant features, and calculate their sizes based on established calibration parameters. Techniques such as edge detection, contour analysis, and geometric transformations will be utilized to achieve accurate size estimation.

The project will not only focus on the technical implementation but also explore practical considerations such as camera calibration, lighting conditions, and noise reduction techniques to enhance measurement accuracy. A graphical user interface (GUI) will be developed to facilitate user interaction, allowing for easy input of images and visualization of the measured dimensions.

By developing a robust method for object size measurement using OpenCV, this project aims to contribute to the advancement of computer vision applications in various domains, offering a versatile and cost-effective solution for precise object analysis.

Object size measurement is a crucial process across various fields, including engineering, manufacturing, and everyday tasks. It involves determining the dimensions or volume of an object to ensure it meets design specifications and functional requirements. Accurate measurements are vital for quality control, proper fit in assembly, and efficient inventory management. Tools used for measuring include simple instruments like rulers and tape measures, as well as more precise devices such as calipers and micrometers. Advanced techniques like laser scanning and photogrammetry provide detailed 3D measurements for complex objects. Adhering to measurement standards, such as ISO or ANSI, is essential for consistency and reliability. Challenges in measurement include ensuring accuracy and precision, understanding different units of measurement, and accounting for environmental factors that can affect results. Mastering these measurement techniques is key to achieving desired outcomes in both technical and practical applications.

Concept of size is something that we all learn at a young age. As children, we have a sense of what is enormous and what is tiny, especially in comparison to ourselves. However, there is one key feature to this that we will discover later: the relative character of what we term size. The size of an object, or anything for that matter, has no meaning in and of itself. Size is only an approximation of one entity in relation to another.

Size is relative, big or small, tiny or tall, all this can be said only when compared with something else. For example, the cricket ball is small in comparison with the basketball but is large when compared to a tennis ball. This ambiguity shows that size is a relative term.

**1.1 Aim**

OpenCV, short for Open Source Computer Vision Library, is a versatile open-source library designed for computer vision and image processing tasks. It offers a comprehensive set of tools and algorithms that enable developers to manipulate images, perform real-time computer vision tasks, and integrate machine learning models seamlessly. OpenCV is widely used across various industries including robotics, healthcare, automotive, and entertainment, providing essential functionalities for tasks such as object detection, facial recognition, and video analysis. Its cross-platform compatibility and active community support make it a preferred choice for both academic research and commercial applications in the field of computer vision.

**1.2 Overview**

The ability to accurately measure object dimensions is crucial across various fields such as manufacturing, robotics, and biomedical research. Traditional methods often involve manual measurements or specialized equipment, which can be time-consuming and error-prone. By leveraging OpenCV's capabilities, this project aims to automate the process, providing a reliable and efficient solution for determining the size of objects captured by a camera.

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associate with that image. Usually Image Processing system in-clades treating images as two-dimensional signals while applying already set signal processing methods to them. Image processing basically includes the following three steps. Importing the image with optical scanner or by digital photography. Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs. Output is the last stage in which result can be altered image or report that is based on image analysis.

**1.3 Outcome**

The project will focus on implementing fundamental computer vision techniques such as image preprocessing, object detection, contour analysis, and geometric transformations. These techniques will be optimized to ensure accurate and precise size estimation under various conditions, including different object types, lighting environments, and camera settings.

A crucial aspect of the project is the development of a user-friendly graphical interface (GUI). This interface will allow users to interact with the system seamlessly, inputting images or video streams and viewing the measured dimensions in real-time. The GUI will enhance usability and accessibility, making the measurement system practical for users across different industries and applications.

We will use a real-time technique to live the physical properties of things within the in close this project. We started by putting in a camera for live data entry. we can also use a video that has been recorded. We use the OpenCV module to use the gaussian blur after the camera setup is complete. The image is blurred by employing a low-pass filter kernel to convolve it. It may be wont to reduce noise. It removes high-frequency material from the image (for example, noise and edges). As a result, edges are blurred slightly during this process (however other blurring techniques don't blur the edges). Then we use a procedure referred to as erosion. The boundaries of the realm are being eroded thanks to this. it's a foreground object that's wanting to blur the image's details. The contour parameters are then determined. A curve will be wont to describe contours. All continuous points (along the boundary) with the identical colour or intensity are joined together. The contours are a useful gizmo for object detection and recognition further as form analysis. Warp perspective and transform: Warping perspective is converting or transforming a view. Images are transformed from an angle to a birds' eye view perspective to vary. We'll use the OpenCV warp perspective function to convert the image to a Veronica chamaedrys view. Calculating Scale Factor: looking on the precision we select it approximates a contour form to a different shape with a fewer number of vertices.

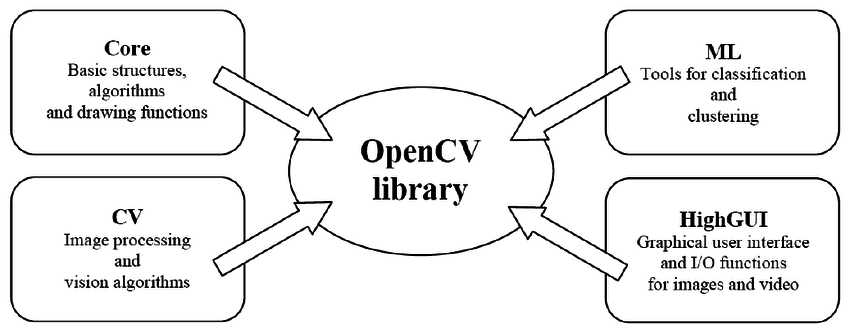
**CHAPTER – 2**

**DESIGN AND IMPLEMENTATION**

**About OpenCV**

OpenCV, short for Open Source Computer Vision Library, is an open-source computer vision and machine learning software library. Originally developed by Intel, it is now maintained by a community of developers under the OpenCV Foundation.

**OpenCV** is a huge open-source library for computer vision, machine learning, and image processing. OpenCV supports a wide variety of programming languages like Python, C++, Java, etc. It can process images and videos to identify objects, faces, or even the handwriting of a human. When it is integrated with various libraries, such as [Numpy](https://www.geeksforgeeks.org/python-numpy/) which is a highly optimized library for numerical operations, then the number of weapons increases in your Arsenal i.e whatever operations one can do in Numpy can be combined with OpenCV.



**Fig 1.1 : Library organisation for OpenCV**

OpenCV was erected to give a common structure for computer vision operations and to accelerate the use of machine perception in the marketable products. Being a BSD certified product, OPENCV makes it easy for businesses to use and modify the law. The library has further than 2500 optimized algorithms, which includes a comprehensive set of both classic and state of heart computer vision and machine literacy algorithms. These algorithms can be used to descry and fete faces, identify objects, classify mortal conduct in vids, track camera movements, track moving objects, excerpt 3D models of objects, produce 3D point shadows from stereo cameras, sew images together to produce a high resolution image of an entire scene, find analogous images from a n image database, remove red eyes from images taken using flash, follow eye movements, fete decor and establish labels to overlay it with stoked reality, etc. OpenCV has further than 47 thousand people of stoner community and estimated number of downloads exceeding 18 million. The library is used considerably in companies, exploration groups and by governmental bodied.

Measuring the size of an object using OpenCV typically involves several steps: object detection, determining scale, and calculating dimensions based on known references. Here’s a basic outline of how you can approach this:

**1. Object Detection**

First, you need to detect the object whose size you want to measure. This can be done using various techniques such as color thresholding, contour detection, or machine learning-based object detection models.

Object detection in OpenCV involves the process of finding and identifying objects within an image or a video stream. Here's a brief summary of how it's typically approached:

1. **Image Preprocessing**: The input image is often preprocessed to enhance features that make object detection easier, such as contrast enhancement or noise reduction.
2. **Feature Extraction**: Various methods are used to extract meaningful features from the image, such as edges, corners, or textures, which help in identifying potential object locations.
3. **Object Localization**: Techniques like sliding window or region proposal methods are used to localize potential object regions within the image.
4. **Feature Matching**: Features extracted from potential object regions are compared against features of known objects (stored in a model or database) using methods like template matching, histograms of oriented gradients (HOG), or deep learning-based methods.
5. **Classification**: Once potential objects are detected, a classifier is used to determine the class of each detected object based on the features extracted.
6. **Bounding Box Generation**: Detected objects are often enclosed within bounding boxes to indicate their location and size within the image.
7. **Post-processing**: Techniques like non-maximum suppression may be applied to refine object detections and reduce duplicate detections.

OpenCV provides various built-in functions and modules to facilitate these steps, including but not limited to:

1. **cv2.dnn module**: For deep learning-based object detection using frameworks like TensorFlow, PyTorch, etc.
2. **cv2.CascadeClassifier**: For Haar cascade classifiers, useful for basic object detection tasks.
3. **cv2.HOGDescriptor**: For Histogram of Oriented Gradients (HOG) based object detection.

Overall, object detection in OpenCV involves a combination of preprocessing, feature extraction, classification, and post-processing steps to accurately identify and locate objects within images or video frames.

**2. Determine Scale**

To measure the size of the object accurately, you need to establish a scale reference. This can be done in several ways:

**Known Distance**: If you know the physical distance between two points in the image (e.g., distance between two corners of a known object of fixed size), you can use this to establish a scale.

**Camera Calibration**: If you know the camera parameters (intrinsic and extrinsic), you can use these to estimate distances more accurately.

Determining scale in OpenCV is essential for accurately interpreting and measuring objects within images or video streams. This process involves establishing a reference size or dimension within the scene, enabling the conversion of pixel measurements to real-world units. This can be achieved through several methods: using known objects of fixed size as references, calibrating the camera to understand its intrinsic properties like focal length, employing feature-based techniques such as keypoints with known dimensions, or leveraging calibrated cameras with precise parameters. By accurately determining scale, OpenCV applications can perform tasks such as object detection, measurement, and augmented reality with greater precision and reliability, bridging the gap between digital image data and real-world dimensions effectively.

**3. Calculate Object Size**

Once you have the object detected and the scale determined, you can calculate its size. Calculating the size of objects in OpenCV involves several fundamental steps. First, an image containing the object of interest is loaded and processed. Object detection techniques, such as deep learning-based models or traditional methods like Haar cascades, are then applied to identify and localize the object within the image. Once detected, the object is typically enclosed within a bounding box, allowing the extraction of its dimensions in pixels. To convert these pixel dimensions into real-world units like meters or centimeters, camera calibration is crucial. This calibration involves determining the camera's intrinsic parameters, such as focal length, which are used to establish a scale factor for converting pixel measurements to physical dimensions accurately. By following these steps, OpenCV enables the precise measurement and analysis of object sizes, supporting various applications in fields like computer vision, robotics, and augmented reality.

OpenCV (Open Source Computer Vision Library) is a robust, open-source library designed for real-time computer vision and image processing tasks. Developed by Intel and maintained by a global community, OpenCV offers an extensive suite of tools and functions that cater to a wide range of visual data analysis needs, making it a vital resource for developers, researchers, and engineers.

At its core, OpenCV provides a comprehensive set of functions for basic image processing operations. This includes essential tasks such as reading and writing images, resizing, rotating, and converting between different color spaces. The library also excels in advanced image processing techniques, including filtering methods like Gaussian and median blurring, edge detection using the Canny algorithm, and complex geometric transformations such as affine and perspective warping. These capabilities allow users to manipulate and analyze images with high precision.

In the realm of computer vision, OpenCV is highly regarded for its powerful object detection and recognition features. The library includes pre-trained classifiers, such as Haar cascades, which are effective for detecting faces, eyes, and other objects within images. OpenCV also supports feature detection and matching with advanced algorithms like SIFT (Scale-Invariant Feature Transform) and ORB (Oriented FAST and Rotated BRIEF). These algorithms are crucial for tasks such as image stitching, 3D reconstruction, and motion tracking. By leveraging these tools, users can develop sophisticated vision systems that identify and track objects in real time.

OpenCV's machine learning capabilities are also noteworthy. The library's `ml` (machine learning) module provides algorithms for classification, regression, and clustering. This includes models like Support Vector Machines (SVM), Decision Trees, and K-Nearest Neighbors (KNN). These models enable users to train and apply machine learning techniques directly within OpenCV, though the library also integrates seamlessly with other machine learning frameworks for more advanced tasks. This integration supports a broad spectrum of applications, from predictive analytics to complex pattern recognition.

The `dnn` (Deep Neural Networks) module of OpenCV further enhances its capabilities by facilitating the use of pre-trained deep learning models. Users can import models from popular frameworks such as TensorFlow, Caffe, and PyTorch, allowing for sophisticated tasks like object detection, image classification, and semantic segmentation. This deep learning support positions OpenCV at the cutting edge of computer vision technology, enabling the implementation of state-of-the-art neural network architectures.

In terms of video analysis, OpenCV provides tools for processing and analyzing video streams. This includes features for motion detection, background subtraction, and video stabilization, which are valuable for applications such as surveillance systems, video editing, and real-time analytics. The ability to handle and analyze video data in real time expands the library's utility across various domains.

OpenCV also offers GUI functionalities through its `highgui` module, which allows users to create graphical user interfaces for displaying images and videos, and interacting with them. This feature is particularly useful for developing interactive applications and visualization tools.

The library’s versatility is further evidenced by its cross-platform support and multi-language accessibility. OpenCV is compatible with major operating systems like Windows, macOS, and Linux, and supports multiple programming languages, including C++, Python, Java, and JavaScript (via OpenCV.js). This broad compatibility ensures that OpenCV can be integrated into diverse development environments and projects.

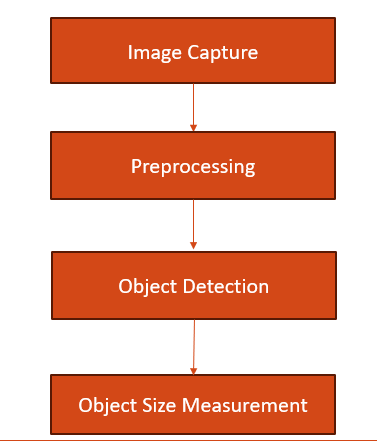
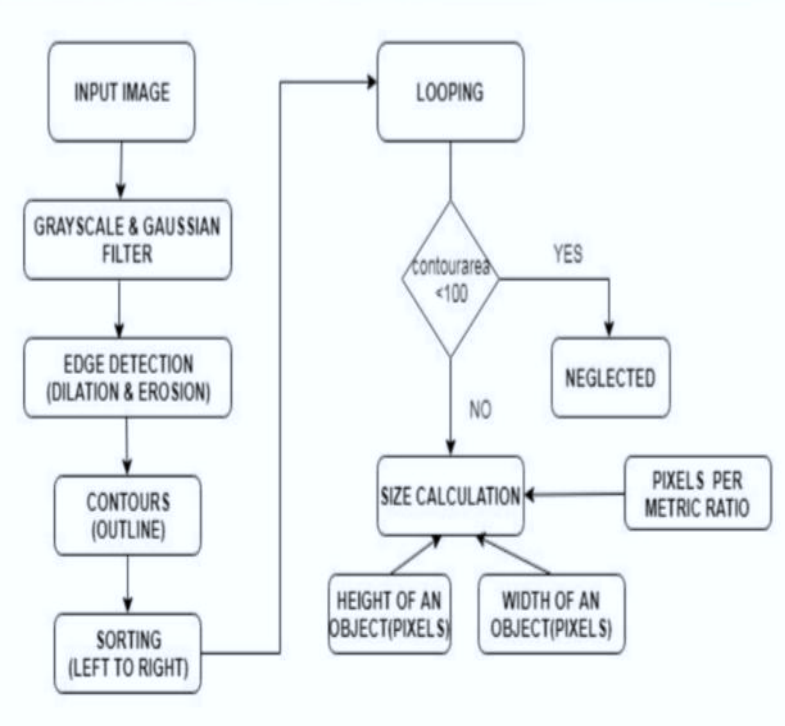
Overall, OpenCV stands out as a powerful and flexible library for computer vision and image processing. Its extensive functionality, deep learning support, and cross-platform capabilities make it an indispensable tool for tackling a wide array of visual analysis challenges. With a wealth of documentation and an active community, OpenCV provides the resources and support needed to leverage its full potential in both research and practical applications.

**2.1 Algorithm**

1. **Imports and Helper Function**:
   1. The code imports necessary libraries (cv2 for OpenCV, numpy for numerical operations, imutils for image processing utilities, and argparse for command-line argument parsing).
   2. Defines a helper function midpoint(ptA, ptB) to compute the midpoint between two points.
2. **Argument Parsing**:
   1. Command-line arguments are parsed using argparse. Key arguments include -m for mode (either "image" or "video"), -i for input image path (required if mode is "image"), and -w for the width of a known object in inches, used for calibration.
3. **Initialization**:
   1. Depending on the mode ("image" or "video"), the appropriate initialization is performed. If in video mode, it initializes the video capture from the default camera (cv2.VideoCapture(0)).
4. **Processing Loop**:
   1. Continuously loops over frames from the video stream or processes a single image depending on the mode.
   2. Resizes each frame to a width of 800 pixels for consistent processing.
   3. Converts each frame to grayscale and applies Gaussian blur to reduce noise.
5. **Edge Detection and Contours**:
   1. Performs Canny edge detection to find edges in the grayscale frame.
   2. Applies dilation and erosion to close gaps between detected edges (cv2.dilate and cv2.erode).
   3. Finds contours (cv2.findContours) in the processed edge map.
6. **Contour Processing**:
   1. Sorts contours from left to right (imutils.contours.sort\_contours).
   2. Initializes pixelsPerMetric to None for calibration.
7. **Iterating Through Contours**:
   1. Iterates over each contour detected.
   2. Skips contours that are too small (cv2.contourArea(c) < 100).
   3. Computes the rotated bounding box (cv2.minAreaRect(c)) around each contour.
   4. Orders the points of the box (perspective.order\_points).
8. **Dimension Calculation**:
   1. Computes midpoints between vertices of the bounding box.
   2. Computes Euclidean distances (dist.euclidean) between these midpoints to determine dimensions dA and dB.
   3. If pixelsPerMetric is None, calculates it as the ratio of the known width (args["width"]) to dB, establishing a scaling factor for pixel-to-inch conversion.
9. **Dimension Display**:
   1. Converts dA and dB from pixels to inches using pixelsPerMetric.
   2. Draws the dimensions on the original frame (cv2.putText).
10. **Display and User Interaction**:
    1. Displays the annotated frame with dimensions (cv2.imshow).
    2. Checks for a 'q' key press to exit the loop (cv2.waitKey).
11. **Cleanup**:
    1. Releases the video stream (cap.release()) and closes all OpenCV windows (cv2.destroyAllWindows()).

The provided Python script utilizes OpenCV for real-time object dimension measurement from a video stream or an input image. It begins by parsing command-line arguments to specify the operational mode (either "image" or "video"), the path to the input image (if applicable), and the known width of an object in inches for calibration purposes. Depending on the mode selected, the script either initializes a video capture or loads a single image. Within the processing loop, each frame undergoes resizing, conversion to grayscale, Gaussian blurring, and edge detection using Canny edge detection. These edges are further refined through dilation and erosion operations to enhance contour detection. Contours are then identified using cv2.findContours, sorted from left to right using imutils.contours.sort\_contours, and processed individually. For each contour, a rotated bounding box is calculated (cv2.minAreaRect) and its vertices are ordered to ensure correct perspective. Midpoints between specific vertices of the bounding box are computed to establish reference points for dimension measurement. The Euclidean distance between these midpoints provides the basis for calculating object dimensions in pixels. By comparing these dimensions to the known width of the reference object (converted to pixels), the script determines a scaling factor (pixelsPerMetric) for converting pixel measurements to real-world units (inches). Finally, the dimensions are annotated on the original frame using cv2.putText, and the annotated frame is displayed. User interaction is facilitated by checking for a 'q' key press to exit the processing loop. Overall, this script demonstrates a structured approach to real-time object dimension measurement using OpenCV, suitable for applications requiring accurate size estimation of objects within a video stream or static images.

**2.2 Flowchart**

**** **Fig 2.1: Flow chart demonstrating working of object size measurement.**

**Fig 2.2: Steps in object measurement.**

**2.2 OpenCV API’s Used with Description**

OpenCV (Open Source Computer Vision Library) is a powerful open-source computer vision and machine learning software library. It provides a wide range of functionalities for processing images and videos. Here are some key APIs and functions in OpenCV along with brief descriptions:

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| **cv::Mat**: | Represents an n-dimensional dense numerical single-channel or multi-channel array.Used to store and manipulate image and matrix data. |
| **cv::imshow**: | Displays an image in a window on the screen.Useful for visualizing images during development and debugging. |
| **cv::imread**: | Reads an image from a file into a Mat object.Supports various image formats (JPEG, PNG, TIFF, etc.). |
| **cv::imwrite**: | Saves an image stored in a Mat object to a file.Supports various image formats (JPEG, PNG, TIFF, etc.). |
| **cv::resize**: | Resizes an image to a specified size.Allows for resizing while maintaining aspect ratio or by specifying interpolation methods. |
| **cv::cvtColor**: | Converts an image from one color space to another.Supports conversions between RGB, HSV, GRAY, etc. |
| **cv::threshold**: | Applies a fixed-level threshold to each pixel in an image.Useful for binary image segmentation based on pixel intensity. |
| **cv::findContours**: | Detects contours in a binary image and is represented as a list of points. |
| **cv::drawContours**: | Draws contours onto an image.Useful for visualizing detected contours or regions of interest. |
| **cv::CascadeClassifier**: | Implements the Haar cascade classifier for object detection. Used for detecting faces, eyes, and other objects based on pre-trained classifiers. |
| **cv::VideoCapture**: | Provides an interface to capture video from a camera or a file. Allows for frame-by-frame processing of video streams. |
| **cv::VideoWriter**: | Writes video frames to a file.Supports various video codecs and formats. |
| **cv::BackgroundSubtractor**: | Base class for background/foreground segmentation. Provides methods for segmenting moving objects from static scenes. |
|  |

**Table 2.1: OpenCV API’s**

OpenCV (Open Source Computer Vision Library) is a highly regarded library used for computer vision, image processing, and machine learning tasks. It provides a comprehensive set of tools and functions designed to enable real-time processing and analysis of visual data. OpenCV's API (Application Programming Interface) is a powerful feature that allows developers to integrate its functionalities into their applications.

Overview of OpenCV API

1. Core Features:

- Image Processing: Functions for filtering, transformations, color space conversions, and geometric transformations.

- Computer Vision: Tools for object detection, face recognition, feature extraction, and tracking.

- Machine Learning: Algorithms for classification, clustering, and regression, as well as integration with deep learning frameworks.

- \*\*Video Analysis\*\*: Capabilities for motion detection, background subtraction, and video stabilization.

\*\*2. \*\*Key Components\*\*:

- \*\*Core Module\*\*: Includes fundamental data structures, functions for basic image manipulation, and mathematical operations.

- \*\*Imgproc Module\*\*: Provides functions for image processing tasks like resizing, blurring, and edge detection.

- \*\*HighGUI Module\*\*: Facilitates image and video input/output, and includes GUI functionalities for creating windows and handling user interactions.

- \*\*Features2D Module\*\*: Contains algorithms for feature detection, description, and matching, such as SIFT, SURF, and ORB.

- \*\*Object Detection\*\*: Includes methods for detecting objects using pre-trained classifiers, such as Haar cascades and deep learning models.

- \*\*Machine Learning Module\*\*: Offers tools for training and using machine learning models, including support for algorithms like Support Vector Machines (SVM), Decision Trees, and K-Nearest Neighbors (KNN).

\*\*3. \*\*Common API Functions\*\*:

- \*\*Image Reading/Writing\*\*: `cv2.imread()` and `cv2.imwrite()` for loading and saving images.

- \*\*Image Display\*\*: `cv2.imshow()` to create and show image windows.

- \*\*Image Transformation\*\*: Functions like `cv2.resize()`, `cv2.rotate()`, and `cv2.warpAffine()` for geometric transformations.

- \*\*Filtering\*\*: `cv2.GaussianBlur()`, `cv2.medianBlur()`, and `cv2.filter2D()` for smoothing and sharpening images.

- \*\*Feature Detection\*\*: `cv2.SIFT\_create()`, `cv2.ORB\_create()` for detecting and describing features.

- \*\*Object Detection\*\*: `cv2.CascadeClassifier()` for using pre-trained classifiers to detect faces and other objects.

\*\*4. \*\*Integration with Deep Learning\*\*:

- \*\*DNN Module\*\*: Facilitates the use of deep learning models with functions to load and run models from frameworks like TensorFlow, Caffe, and PyTorch.

- \*\*Pre-trained Models\*\*: Provides access to a variety of pre-trained models for tasks like object detection (YOLO, SSD) and image classification (ResNet, Inception).

\*\*5. \*\*Programming Languages\*\*:

- OpenCV's API is accessible through multiple programming languages, including C++, Python, Java, and even JavaScript (via OpenCV.js), making it versatile for different development environments.

\*\*6. \*\*Documentation and Support\*\*:

- Comprehensive documentation is available on the [OpenCV website](https://docs.opencv.org/), providing detailed explanations of functions, classes, and usage examples.

- The community support includes forums, tutorials, and an active development community that contributes to continuous improvement and problem-solving.

Overall, OpenCV's API offers a robust set of tools for a wide range of computer vision and image processing tasks, making it a popular choice among developers and researchers working in these domains.

The OpenCV API is a powerful toolset for computer vision and image processing, providing a broad array of functions for tasks like filtering, object detection, and feature extraction. It includes core modules for basic image manipulation, advanced features for object recognition, and machine learning algorithms for model training and prediction. Key functions allow for image reading, display, transformation, and processing, while integration with deep learning frameworks extends its capabilities. Accessible through languages such as C++, Python, and Java, OpenCV is widely used for both real-time and complex visual analysis, supported by extensive documentation and an active community.

Top of Form

Bottom of Form

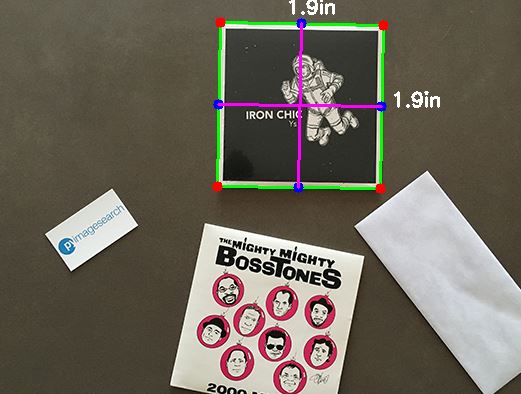
**CHAPTER – 3**

**RESULT ANALYSIS**

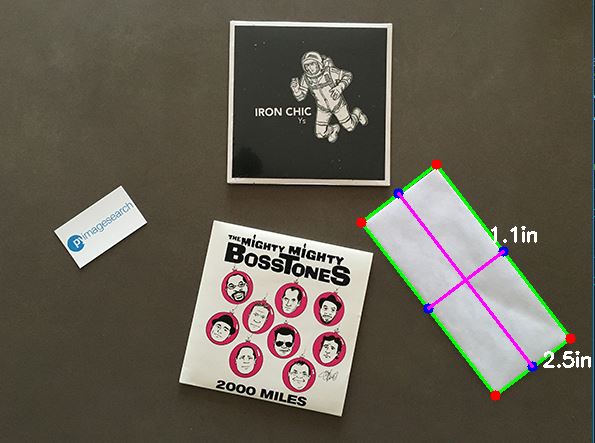
**Snapshots**

****

**Fig 3.1:Snap Shot measuring size of object placed at bottom of image**

****

**Fig 3.2:Snap Shot measuring size of object placed at top of image**

****

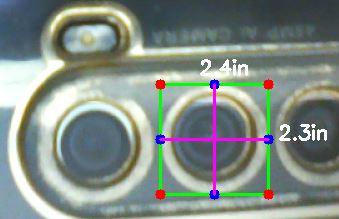
**Fig 3.3:Snap Shot measuring size of object placed at right of image**

****

**Fig 3.4:Snap Shot measuring size of object placed at left of image.**

****

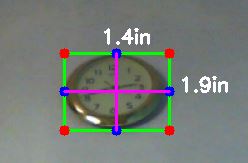
**Fig 3.5:Measuring size of logo.**

****

**Fig 3.6:Measuring size of camera.**

****

**Fig 3.7: Measuring size of switch board**

****

**Fig 3.8: Measuring size of clock**

**Discussion**

The above snapshots represents the object size measurement of both static images and the captured live images .we can notice that in static image the size of each object present in image is measured one after other by measuring all the objects present in it. similarly in live video streaming it measures the size of objects in inch by calculating euclidean distance that is encountered on the screen .

**CONCLUSION AND FUTURE ENHANCEMENT**

**Conclusion**

To conclude, measuring the size of objects using OpenCV in Python offers a powerful and versatile solution for various applications. By leveraging image processing techniques such as edge detection, contour detection, and perspective transformation, we can accurately determine dimensions like length, width, and area from digital images or video streams. This capability is particularly useful in fields ranging from industrial automation and quality control to medical imaging and augmented reality. With OpenCV's robust library and Python's flexibility, measuring object sizes becomes efficient, precise, and adaptable to diverse real-world scenarios.

Because of this framework, numerous enhancements can be made to the modern area. The task effectively gauges the elements of the article progressively. Consequently the PC vision (webcam gadget and code) is utilized to gauge the aspects progressively. It catches the picture from the real-time video casing and afterward shows its aspects. A Canny edge locator is effectively used to identify the aspects. This procedure works quick and enjoys many benefits and remarkable highlights that can be carried out in reality.

**Future Enhancement**

In the realm of computer vision and image processing, the future enhancement of object size measurement using OpenCV and Python holds exciting possibilities. Current methodologies typically involve detecting objects within images, often using techniques like contour detection or object segmentation. These methods can accurately locate and outline objects but may face challenges in precisely measuring their sizes due to factors like perspective distortion, occlusion, or varying lighting conditions.

Future enhancements could focus on integrating advanced machine learning models, such as deep neural networks, to improve object size measurement accuracy. These models could learn to estimate object dimensions more robustly by accounting for context, scale,

and environmental factors. Additionally, leveraging advancements in sensor technology, such as depth sensors or LiDAR, could provide more accurate spatial information for size estimation. Furthermore, incorporating real-time feedback loops and adaptive algorithms could enhance the system's ability to dynamically adjust measurements based on changing conditions or user-defined requirements. Overall, future enhancements in object size measurement using OpenCV and Python are likely to blend sophisticated algorithms with improved sensor capabilities, enabling more precise and versatile applications in fields like industrial automation, robotics, and augmentedreality.

**REFERENCES**

[1] Learning OpenCV 4: Computer Vision with Python 3 by Adrian Kaehler and Gary Bradski 2019

[2] OpenCV 4 with Python Blueprints" by Michael Beyer 2019

[3] OpenCV Essentials by Oscar Deniz Suarez, Madel Milagro Fernandez Carrobles, Noelia Vallez Enano, Gloria Bueno Garcia 2014

[4]OpenCV 4 Computer Vision Application Programming Cookbook" by Robert Laganière 2019

**APPENDIX – A**

**SOURCE CODE**

# import the necessary packages

from scipy.spatial import distance as dist

from imutils import perspective

from imutils import contours

import numpy as np

import argparse

import imutils

import cv2

def midpoint(ptA, ptB):

return ((ptA[0] + ptB[0]) \* 0.5, (ptA[1] + ptB[1]) \* 0.5)

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-m", "--mode", type=str, default="image",

help="mode of operation: 'image' or 'video'")

ap.add\_argument("-i", "--image", required=False,

help="path to the input image (only required if mode is 'image')")

ap.add\_argument("-w", "--width", type=float, required=True,

help="width of the left-most object in the image (in inches)")

args = vars(ap.parse\_args())

# initialize the video stream and allow the camera sensor to warm up

if args["mode"] == "video":

print("[INFO] starting video stream...")

cap = cv2.VideoCapture(0)

# allow the camera to warm up

while not cap.isOpened():

pass

# initialize variables for calibration

pixelsPerMetric = None

# loop over the frames from the video stream

while True:

# grab the next frame from the video stream, resize it, and convert it to grayscale

ret, frame = cap.read()

if not ret:

break

# resize the frame and convert it to grayscale

frame = imutils.resize(frame, width=800)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

gray = cv2.GaussianBlur(gray, (7, 7), 0)

# perform edge detection, then perform a dilation + erosion to

# close gaps in between object edges

edged = cv2.Canny(gray, 50, 100)

edged = cv2.dilate(edged, None, iterations=1)

edged = cv2.erode(edged, None, iterations=1)

# find contours in the edge map

cnts = cv2.findContours(edged.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

# check if no contours were found

if len(cnts) == 0:

cv2.imshow("Video", frame)

key = cv2.waitKey(1) & 0xFF

if key == ord('q'):

break

continue

# sort the contours from left-to-right and initialize the

# 'pixels per metric' calibration variable

(cnts, \_) = contours.sort\_contours(cnts)

pixelsPerMetric = None

# loop over the contours individually

for c in cnts:

# if the contour is not sufficiently large, ignore it

if cv2.contourArea(c) < 100:

continue

# compute the rotated bounding box of the contour

orig = frame.copy()

box = cv2.minAreaRect(c)

box = cv2.cv.BoxPoints(box) if imutils.is\_cv2() else cv2.boxPoints(box)

box = np.array(box, dtype="int")

# order the points in the contour such that they appear

# in top-left, top-right, bottom-right, and bottom-left

# order, then draw the outline of the rotated bounding

# box

box = perspective.order\_points(box)

cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)

# loop over the original points and draw them

for (x, y) in box:

cv2.circle(orig, (int(x), int(y)), 5, (0, 0, 255), -1)

# unpack the ordered bounding box, then compute the midpoint

# between the top-left and top-right coordinates, followed by

# the midpoint between bottom-left and bottom-right coordinates

(tl, tr, br, bl) = box

(tltrX, tltrY) = midpoint(tl, tr)

(blbrX, blbrY) = midpoint(bl, br)

# compute the midpoint between the top-left and top-right points,

# followed by the midpoint between the top-righ and bottom-right

(tlblX, tlblY) = midpoint(tl, bl)

(trbrX, trbrY) = midpoint(tr, br)

# draw the midpoints on the image

cv2.circle(orig, (int(tltrX), int(tltrY)), 5, (255, 0, 0), -1)

cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (255, 0, 0), -1)

cv2.circle(orig, (int(tlblX), int(tlblY)), 5, (255, 0, 0), -1)

cv2.circle(orig, (int(trbrX), int(trbrY)), 5, (255, 0, 0), -1)

# draw lines between the midpoints

cv2.line(orig, (int(tltrX), int(tltrY)), (int(blbrX), int(blbrY)),

(255, 0, 255), 2)

cv2.line(orig, (int(tlblX), int(tlblY)), (int(trbrX), int(trbrY)),

(255, 0, 255), 2)

# compute the Euclidean distance between the midpoints

dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))

dB = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))

# if the pixels per metric has not been initialized, then

# compute it as the ratio of pixels to supplied metric

# (in this case, inches)

if pixelsPerMetric is None:

pixelsPerMetric = dB / args["width"]

# compute the size of the object

dimA = dA / pixelsPerMetric

dimB = dB / pixelsPerMetric

# draw the object sizes on the image

cv2.putText(orig, "{:.1f}in".format(dimA),

(int(tltrX - 15), int(tltrY - 10)), cv2.FONT\_HERSHEY\_SIMPLEX,

0.65, (255, 255, 255), 2)

cv2.putText(orig, "{:.1f}in".format(dimB),

(int(trbrX + 10), int(trbrY)), cv2.FONT\_HERSHEY\_SIMPLEX,

0.65, (255, 255, 255), 2)

# show the output image

cv2.imshow("Video", orig)

# check for 'q' key pressed, if pressed break from the loop

key = cv2.waitKey(1) & 0xFF

if key == ord('q'):

break

# release the video stream and close any open windows

cap.release()

cv2.destroyAllWindows()

# load the image, convert it to grayscale, and blur it slightly

image = cv2.imread(args["image"])

# USAGE

# python object\_size.py --image images/example\_01.png --width 0.955

# python object\_size.py --image images/example\_02.png --width 0.955

# python object\_size.py --image images/example\_03.png --width 3.5

def midpoint(ptA, ptB):

return ((ptA[0] + ptB[0]) \* 0.5, (ptA[1] + ptB[1]) \* 0.5)

# construct the argument parse and parse the arguments

ap = argparse.ArgumentParser()

ap.add\_argument("-i", "--image", required=True,help="path to the input image")

ap.add\_argument("-w", "--width", type=float, required=True,help="width of the left-most object in the image (in inches)")

args = vars(ap.parse\_args())

# load the image, convert it to grayscale, and blur it slightly

image = cv2.imread(args["image"])

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

gray = cv2.GaussianBlur(gray, (7, 7), 0)

# perform edge detection, then perform a dilation + erosion to

# close gaps in between object edges

edged = cv2.Canny(gray, 50, 100)

edged = cv2.dilate(edged, None, iterations=1)

edged = cv2.erode(edged, None, iterations=1)

# find contours in the edge map

cnts = cv2.findContours(edged.copy(), cv2.RETR\_EXTERNAL,

cv2.CHAIN\_APPROX\_SIMPLE)

cnts = imutils.grab\_contours(cnts)

# sort the contours from left-to-right and initialize the

# 'pixels per metric' calibration variable

(cnts, \_) = contours.sort\_contours(cnts)

pixelsPerMetric = None

# loop over the contours individually

for c in cnts:

# if the contour is not sufficiently large, ignore it

if cv2.contourArea(c) < 100:

continue

# compute the rotated bounding box of the contour

orig = image.copy()

box = cv2.minAreaRect(c)

box = cv2.cv.BoxPoints(box) if imutils.is\_cv2() else cv2.boxPoints(box)

box = np.array(box, dtype="int")

# order the points in the contour such that they appear

# in top-left, top-right, bottom-right, and bottom-left

# order, then draw the outline of the rotated bounding

# box

box = perspective.order\_points(box)

cv2.drawContours(orig, [box.astype("int")], -1, (0, 255, 0), 2)

# loop over the original points and draw them

for (x, y) in box:

cv2.circle(orig, (int(x), int(y)), 5, (0, 0, 255), -1)

# unpack the ordered bounding box, then compute the midpoint

# between the top-left and top-right coordinates, followed by

# the midpoint between bottom-left and bottom-right coordinates

(tl, tr, br, bl) = box

(tltrX, tltrY) = midpoint(tl, tr)

(blbrX, blbrY) = midpoint(bl, br)

# compute the midpoint between the top-left and top-right points,

# followed by the midpoint between the top-righ and bottom-right

(tlblX, tlblY) = midpoint(tl, bl)

(trbrX, trbrY) = midpoint(tr, br)

# draw the midpoints on the image

cv2.circle(orig, (int(tltrX), int(tltrY)), 5, (255, 0, 0), -1)

cv2.circle(orig, (int(blbrX), int(blbrY)), 5, (255, 0, 0), -1)

cv2.circle(orig, (int(tlblX), int(tlblY)), 5, (255, 0, 0), -1)

cv2.circle(orig, (int(trbrX), int(trbrY)), 5, (255, 0, 0), -1)

# draw lines between the midpoints

cv2.line(orig, (int(tltrX), int(tltrY)), (int(blbrX), int(blbrY)),

(255, 0, 255), 2)

cv2.line(orig, (int(tlblX), int(tlblY)), (int(trbrX), int(trbrY)),

(255, 0, 255), 2)

# compute the Euclidean distance between the midpoints

dA = dist.euclidean((tltrX, tltrY), (blbrX, blbrY))

dB = dist.euclidean((tlblX, tlblY), (trbrX, trbrY))

# if the pixels per metric has not been initialized, then

# compute it as the ratio of pixels to supplied metric

# (in this case, inches)

if pixelsPerMetric is None:

pixelsPerMetric = dB / args["width"]

# compute the size of the object

dimA = dA / pixelsPerMetric

dimB = dB / pixelsPerMetric

# draw the object sizes on the image

cv2.putText(orig, "{:.1f}in".format(dimA),

(int(tltrX - 15), int(tltrY - 10)), cv2.FONT\_HERSHEY\_SIMPLEX,

0.65, (255, 255, 255), 2)

cv2.putText(orig, "{:.1f}in".format(dimB),

(int(trbrX + 10), int(trbrY)), cv2.FONT\_HERSHEY\_SIMPLEX,

0.65, (255, 255, 255), 2)

# show the output image

cv2.imshow("Image", orig)

cv2.waitKey(0)

#command to run

Python filename.py –-mode video –width 0.955 //video

Python filename.py –-image images/example\_01.png –width 0.955 //image