# LAB: Dimension Measurement with 2D camera

**Date:** 2024-April-30

**Author:** Taegeon Han 21900793

**Partner:** Gyeonheal An 21900416

**Demo Video:** https://youtu.be/UL6ffoYS9oc

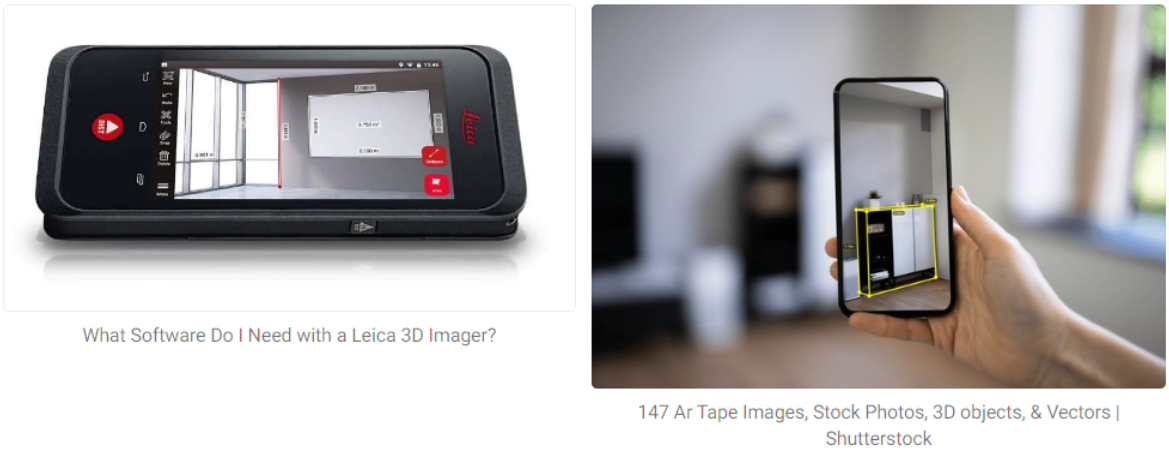
---

## I. Introduction

A company wants to measure the whole dimension of a rectangular object with a smartphone.

We are asked to make an image processing algorithm for an accurate volume measurement of the small object.

- Smartphone camera
- object
- reference



What Software Do I Need with a Leica 3D Imager?

147 Ar Tape Images, Stock Photos, 3D objects, & Vectors | Shutterstock

Figure 1. Completed example photo

## 1. Problem Conditions

- Measure the 3D dimensions (LxWxH) of a small rectangular object
- Assume we know the exact width (**W**) of the target object. We only need to find **L** and **H**.
- The accuracy of the object should be within **3mm**
- We can only use a smartphone (webcam) 2D camera for sensors. No other sensors.
- We cannot know the exact pose of the camera from the object or the table.
- We can use other known dimension objects, such as A4-sized paper, check-board, small square boxes etc

- Try to make the whole measurement process to be as simple, and convenient as possible for the user
    - Using fewer images, using fewer reference objects, etc

## 2. Preparation

### Software Installation

- OpenCV 4.9.0 with Visual Studio 2022
- GML Camera Calibration

### Dataset

This is a picture of the object and reference.

**Angle1** : Download the test image

**Angle2 :** Download the test image

**Angle3 :** Download the test image

# II. Problem

Measure the 3D dimensions (LxWxH) of a rectangular object

## 1. Procedure

1. First, understand fully about the design problem.
2. Design the algorithm flow
    - Calibration, Segment the object from the background, Finding corners etc
3. We can use additional reference objects such as A4 paper, known-sized rectangular objects, etc.
    - We will get a higher point if we use the reference object as simple as possible.
4. We must state all the additional assumptions or constraints to make our algorithm work.
    - We are free to add assumptions and constraints such as the reference object can be placed in parallel to the target block etc
    - But, we will get a higher point if we use fewer constraints/assumptions.
5. Use our webcam or smartphone to capture image(s)
    - Use the given experimental setting of the background and 3D object.

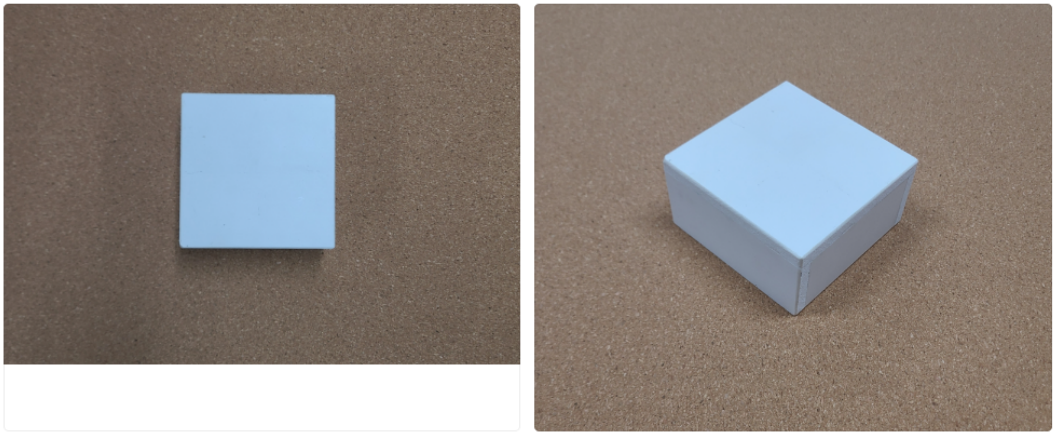


Figure 2. Given rectangular object

6. Measure each dimension of the test rectangular object.
    - The exact width (W) of the target object is given.
    - Measure only Height and Length
7. The output image or video should display the measurement numbers.
8. Your algorithm will be validated with other similar test object

# 2. Algorithm

## 1. Overview

It measures the length of an object by taking a picture of it with a camera.

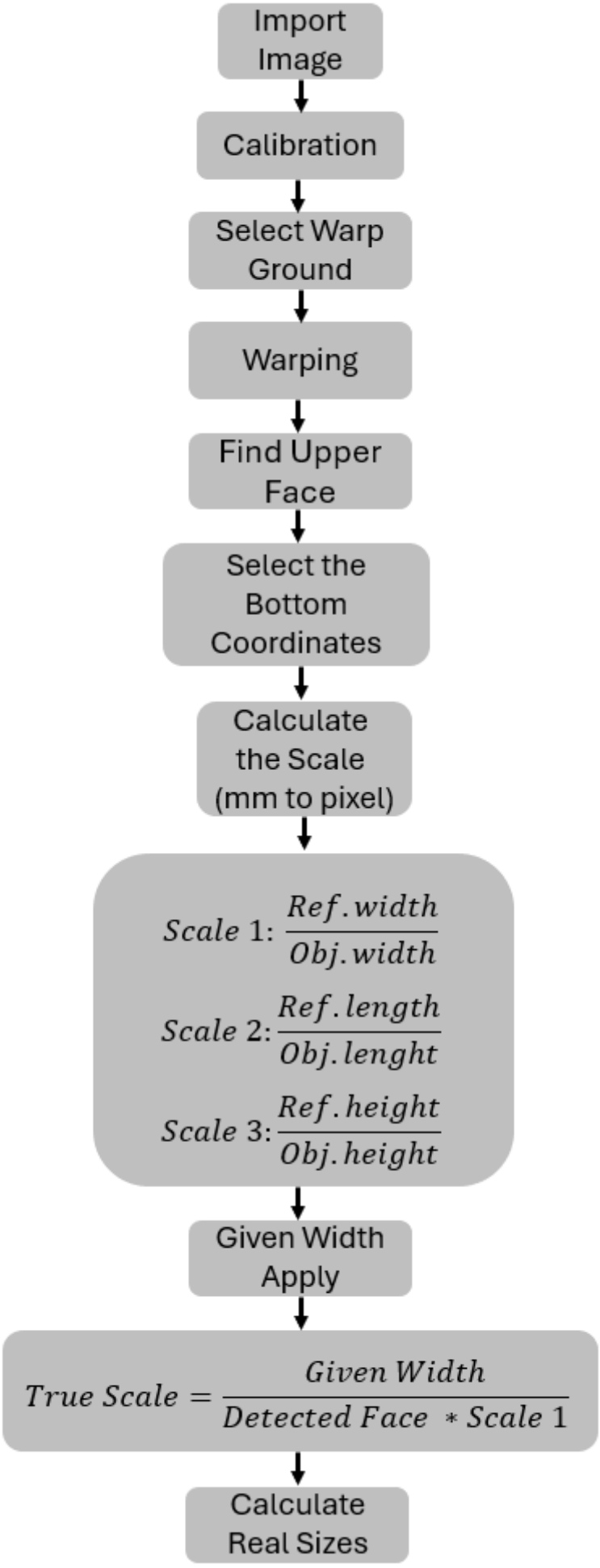



Figure 3. Outline to results

## 2. Assumption

- Box and bottom plate parallel
- All four corners of the Warp plane should be visible in the picture.
- The reference object and the measurement object are located in the same plane (the two objects must be parallel to the floor plate)
- Reference object is located to the right of the measurement object
- You have to see all three sides in the picture

## 3. Process

### Import Image & Calibration

LAB was conducted with 3 photos taken from different angles. Using the GML Camera Calibration app, we calibrated photos taken with a smartphone camera. Other tasks were performed in the calibrated state.

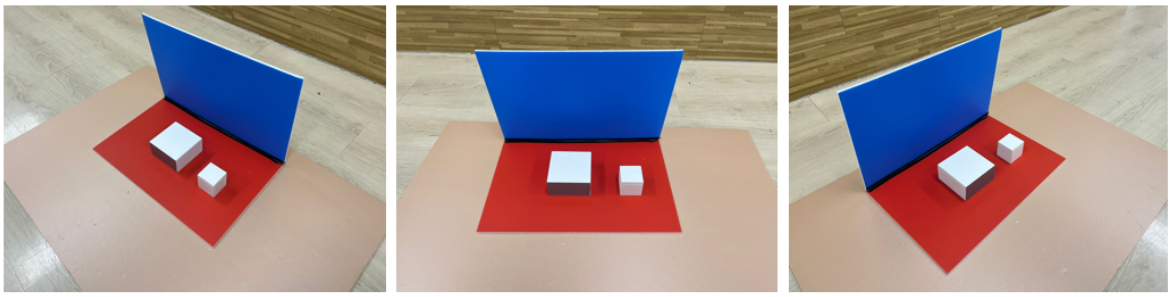


Figure 4. test Image

Figure 5. Calibrated test Image

### Warped Image

There are parts that seem distorted for reasons such as perspective. In image processing, warped images are used to correct the perspective shown in the picture. Select four points to display them on the image and map them to a new plane to correct the distortion. In Figure 5, the object is placed on a red rectangular plate. In order to measure the length of the object accurately, it is important to select four corners of the red plate as reference points. When selecting a point, if it is not accurate, the image may be inaccurate and the length may be measured differently. Selecting four red corners can make the object appear as seen above. The point to note here is that you have to point clockwise from the top left. That way, the image comes out properly.

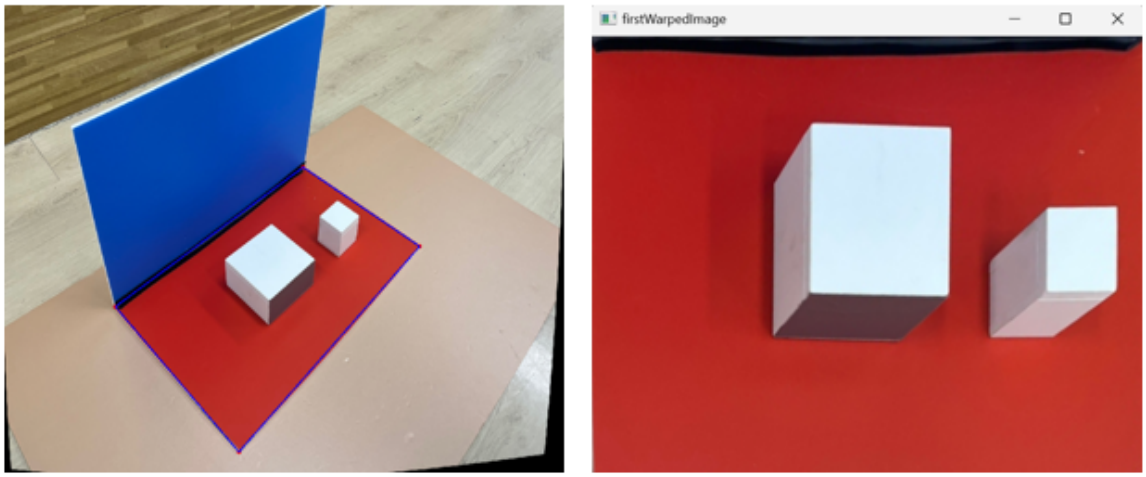



Figure 6. Four points and Warped Image

**Thresholding(Find Upper Face)**

In the warped image, most objects have bright colors, so the threshold value was largely set. In addition, in order to separate the background and the object well, the type was set to Threshold_binary.

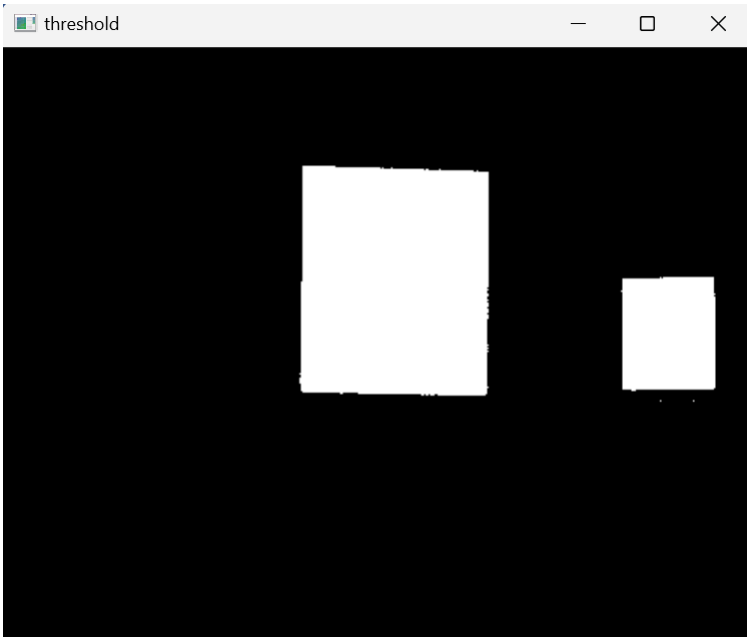



Figure 7. Threshold Image

**Find contours**

Contour is used to represent the outline of an object in an image. In Figure 7, the right side is the reference object, and the left side is the object to be measured. Also, since the shape is square, it is contoured in the form of a square. At this time, the contour must be executed in the Threshold image. The horizontal and vertical lengths can be obtained with the contoured image. Therefore, contour must be done to know the lengths of both objects.

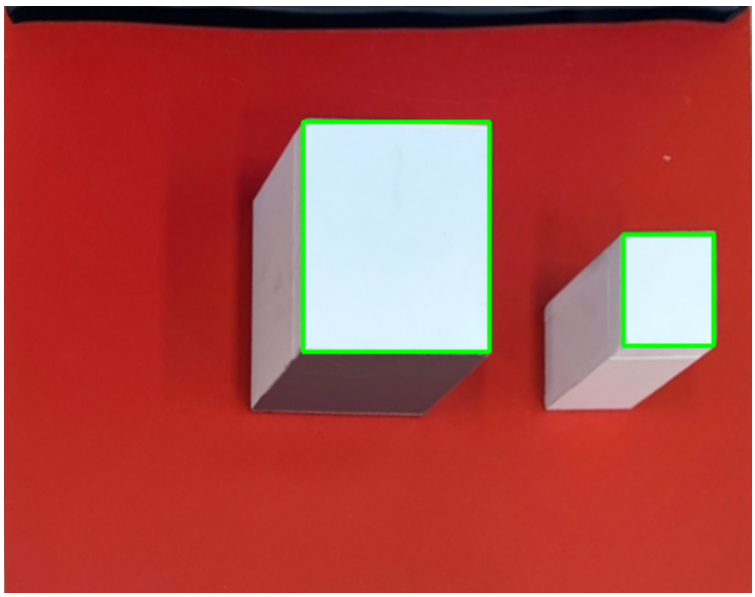


Figure 8. Contour Image

**Select the bottom coordinates**

In the contour image, we need to know where the bottom of each object is. Therefore, if we click the bottom with the mouse, we will recognize that it is the bottom of the object. We can find the coordinates of the clicked point and obtain the height of the object from this.

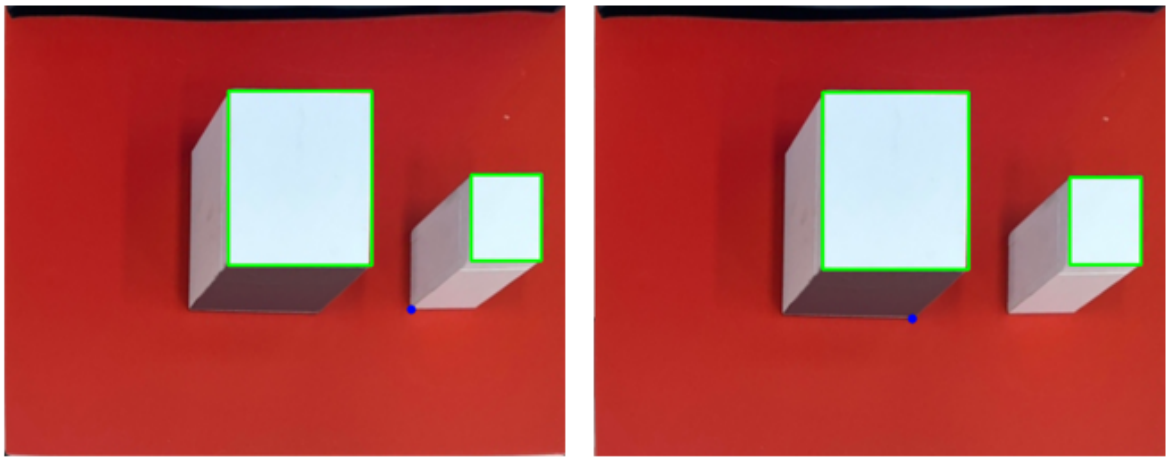


Figure 9. Set bottom of objects

**Calculate Weight, Length, Height**

We are aware of the width, length, and height of the reference object, as well as the width of the object we wish to measure. Using this known information, we can determine the length and height. Since the pixel dimensions (W, L, H) of the contour box are known, conversion to actual length units is possible. By first comparing the widths and then applying the derived scale value consistently to both the length and height, we can obtain the desired measurements.

# III. Result and Discussion

# 1. Final Result

As shown in Figure 4, it was carried out in different photos at different angles. In the process of outputting the results of the three photos, dots were drawn as identically as possible, and as a result, the error rate was within 3mm.

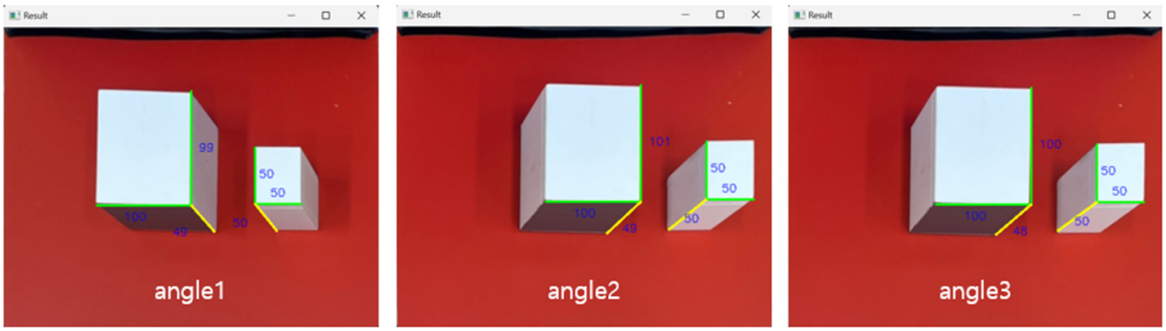



Figure 10. Measurement of the length of each angle

We can see the bottom coordinates where the dots were pointed in the console window.

```
Box1 Coordinates: (359, 272)
Box2 Coordinates: (277, 276)
```

Figure 11. Bottom coordinates

# 2. Discussion

The table below shows the accuracy of object all length.

| Picture | Part | True | Estimated | Accuracy |
|---|---|---|---|---|
| Angle1 | Length | 100 | 99 | 99% |
| | Height | 50 | 49 | 98% |
| Angle2 | Length | 100 | 101 | 98% |
| | Height | 50 | 49 | 98% |
| Angle3 | Length | 100 | 100 | 100% |
| | Height | 50 | 48 | 96% |

Through the procedure, the length of the object could be measured. At first, I tried to use a picture with only one side visible, but if I did this, I had to use two pictures. It's a picture of width, length and width, height. That is, I had to do it twice. To reduce this hassle, I went with a picture with width, length and height shown at once. As a result, I had to click on the warped image and bottom part with the mouse, but I finally succeeded. As you can see from the table above, the accuracy is high.

We didn't set a value separately according to the angle. If we knew all the lengths of the reference object and knew the width of the other object, we could get the length ratio between the two objects. Moreover, we thought that if we applied this to the other object and calculated the length, we could get it regardless of the angle.

You need to accurately measure the edge of the plate on which the object is rising. To extract the warped image, mark four points, but do not mark any points. In the image, objects far from the camera appear smaller due to perspective distortion. The edge of the red plate is parallel to the plane of the object being measured, so you need to mark the points as accurately as possible.

### 3. Demo Video

Link : https://youtu.be/UL6ffoYS9oc

## IV. Conclusion & Troubleshooting

### 1. Conclusion

In this project, it was aimed to reduce the length of the measuring object to less than 3mm. As a result of the progress, it was confirmed that the length and height were given within 3mm. When four points are taken for warped images, it is expected that more accurate lengths can be measured if the exact location of them is known. I n addition to this, fewer errors can be extracted if a method that accurately knows the location of the bottom is applied.

### 2. Troubleshooting

In this project, it was difficult to measure the length through the ratio between the reference object and the object to be measured. Since each width is known, the ratio using it was applied to the remaining length measurement, and the ratio to height and length was also applied. In the process, when only the length ratio to the reference object, not the ratio between the two objects, was calculated and applied in the same manner in other pictures, the result of the value coming out completely strange was also confirmed.

---

## V. Appendix

```
/*
 * DLIP_LAB2.cpp
 * author: Taegeon Han
 * Parter: Gyeonheal An
 * Date: 2024-04-28
 * DLIP_LAB2_Dimension Measurement with 2D camera
 * Measure rectangular objects whole dimension with a smartphone
 */

#include <iostream>
```

```cpp
#include <opencv2/opencv.hpp>
#include "cameraParam.h"

using namespace std;
using namespace cv;

// Global variables for Morphology
int element_shape = MORPH_RECT;      // MORPH_RECT, MORPH_ELIPSE, MORPH_CROSS
int n = 3;
Mat element = getStructuringElement(element_shape, Size(n, n));

Mat src, src_gray, result, result2, result3;
Mat gray, thresh;

// Global variables to store the coordinates of the marked point
Point markedPoint(-1, -1);

vector<Point2f> clickedPoints;
vector<Point2f> secondClickedPoints;  // Store the second set of points
int pointSelectionStage = 1; // Track which set of points is being select
int selectedPointIndex = -1;  // Index of the point being dragged
Mat firstWarpedImage, secondWarpedImage;

// Function headers
void CallBackFunc(int event, int x, int y, int flags, void* userdata);
void updateWarpedImage();
void findBoundingRect(const Mat& image);
void onMouse(int event, int x, int y, int flags, void* userdata);


vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
double scale = 1, scale2 = 1, scale3 = 1;
double sc;
int cnt = 0;
double real_sc;

int box1x;
int box1y;
int box2x;
int box2y;
int clickCount = 0;  // Variables that track the number of clicks


int main()
{
    // Calibration and read image file
    cameraParam param("lab2.xml");
    src = imread("angle3.jpg");
    result = param.undistort(src);

    if (src.empty()) {
        std::cout << "Could not open or find the image!" << endl;
        return -1;
    }
    namedWindow("Source Image", WINDOW_NORMAL);

    // Convert the image to Gray
```

```
    cvtColor(result, src_gray, COLOR_BGR2GRAY);

    setMouseCallback("Source Image", CallBackFunc, nullptr);



    imshow("Source Image", result);

    while (true) {
        char key = static_cast<char>(waitKey(0)); // Wait indefinitely for a
user key press

        if (key == 'c' || key == 'C') {
            clickedPoints.clear();        // Clear first set of points
            firstWarpedImage.release();   // Release the image created by first
warping
            secondClickedPoints.clear();  // Clear second set of points
            secondWarpedImage.release();  // Release the image created by second
warping

            pointSelectionStage = 1;      // Reset the point selection stage to
the first set

            ::imshow("Source Image", result);  // Redisplay the original image
        }

        if (key == 27)
            break;
    }

    waitKey(0);
    return 0;
}


void CallBackFunc(int event, int x, int y, int flags, void* userdata) {
    vector<Point2f>* currentPoints = (pointSelectionStage == 1) ? &clickedPoints
: &secondClickedPoints;

    if (event == EVENT_LBUTTONDOWN) {
        selectedPointIndex = -1;
        for (int i = 0; i < currentPoints->size(); ++i) {
            if (norm(Mat((*currentPoints)[i]), Mat(Point2f(x, y))) < 5) {
                selectedPointIndex = i;
                break;
            }
        }
        if (selectedPointIndex == -1 && currentPoints->size() < 4) {
            currentPoints->push_back(Point2f((float)x, (float)y));
            std::cout << "Point " << currentPoints->size() << ": (" << x << ", "
<< y << ")" << endl;
            if (currentPoints->size() == 4 && pointSelectionStage == 1) {
                updateWarpedImage();
                pointSelectionStage = 2; // Move to second stage after first set
of points
            }

        }
```

```
    }
    else if (event == EVENT_MOUSEMOVE && selectedPointIndex != -1) {
        (*currentPoints)[selectedPointIndex] = Point2f(x, y);
    }
    else if (event == EVENT_LBUTTONUP) {
        selectedPointIndex = -1;
    }


    // Redraw all points
    result2 = result.clone();
    for (size_t i = 0; i < clickedPoints.size(); ++i) {
        circle(result2, clickedPoints[i], 5, Scalar(0, 0, 255), FILLED);
        if (i > 0) line(result2, clickedPoints[i - 1], clickedPoints[i],
Scalar(255, 0, 0), 2);
        if (clickedPoints.size() == 4) line(result2, clickedPoints[3],
clickedPoints[0], Scalar(255, 0, 0), 2);
    }

    imshow("Source Image", result2);
}


void updateWarpedImage() {
    if (clickedPoints.size() == 4) {
        int W = 500, H = 400;  // Arbitrary dimensions
        vector<Point2f> dstCorners = { Point2f(0, 0), Point2f(W - 1, 0),
Point2f(W - 1, H - 1), Point2f(0, H - 1) };
        Mat transformMatrix = getPerspectiveTransform(clickedPoints,
dstCorners);
        warpPerspective(result, firstWarpedImage, transformMatrix, Size(W, H));
        imshow("firstWarpedImage", firstWarpedImage);
        findBoundingRect(firstWarpedImage);
    }
}


void findBoundingRect(const Mat& image) {

    result3 = image.clone();

    // change to grayscale
    cvtColor(image, gray, COLOR_BGR2GRAY);
    threshold(gray, thresh, 220, 255, THRESH_BINARY);
    // Finding contours
    findContours(thresh, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE);

    for (size_t i = 0; i < contours.size(); i++) {

        Rect boundingBox = boundingRect(contours[i]);

        // Restrict of boundingBox
        if (boundingBox.width < 40)
            continue;

        rectangle(firstWarpedImage, boundingBox, Scalar(0, 255, 0), 2); // Draw
a bounding box in green
```

```
        imshow("firstWarpedImage", firstWarpedImage);
    }

    if (clickCount == 0) setMouseCallback("firstWarpedImage", onMouse, nullptr);
}


// Mouse callback function
void onMouse(int event, int x, int y, int flags, void* userdata) {
    if (event == EVENT_LBUTTONDOWN) {
        // Specify coordinates based on the number of clicks
        switch (clickCount) {
        case 0:
            box1x = x;
            box1y = y;
            break;
        case 1:
            box2x = x;
            box2y = y;
            break;
        }
        clickCount++;  // Increase click count

        if (clickCount == 2) {
            cout << "Box1 Coordinates: (" << box1x << ", " << box1y << ")" <<
endl;
            cout << "Box2 Coordinates: (" << box2x << ", " << box2y << ")" <<
endl;
            for (size_t i = 0; i < contours.size(); i++) {

                Rect boundingBox = boundingRect(contours[i]);

                // Restrict of boundingBox
                if (boundingBox.width < 40)
                    continue;

                rectangle(firstWarpedImage, boundingBox, Scalar(0, 255, 0), 2);
// Draw a bounding box in green

                if (boundingBox.width > 45 && cnt == 0) {
                    // Calculate corner points of the bounding box
                    Point2f topLeft(boundingBox.x, boundingBox.y);
                    Point2f topRight(boundingBox.x + boundingBox.width,
boundingBox.y);
                    Point2f bottomRight(boundingBox.x + boundingBox.width,
boundingBox.y + boundingBox.height);
                    Point2f bottomLeft(boundingBox.x, boundingBox.y +
boundingBox.height);

                    double knownWidth = 50.0;  // Desired width in actual units
                    double knownLength = 50.0;  // Desired length in actual
units
                    double knwonHeight = 50.0;  // Desired height in actual
units

                    scale = knownWidth / boundingBox.width;  // Calculate scale
based on the first contour
```

```
                    scale2 = knownLength / boundingBox.height;  // Calculate
scale based on the first contour
                    scale3 = knwonHeight / (sqrt(pow(box1x - bottomLeft.x, 2) +
pow(box1y - bottomLeft.y, 2)));
                    cnt++;

                    line(result3, bottomLeft, bottomRight, Scalar(0, 255, 0),
2);
                    line(result3, topLeft, bottomLeft, Scalar(0, 255, 0), 2);
                    line(result3, bottomLeft, Point(box1x, box1y), Scalar(0,
255, 255), 2);

                    // Calculate the size text
                    string sizeText_W = to_string(static_cast<int>
(ceil(boundingBox.width * scale)));
                    string sizeText_L = to_string(static_cast<int>
(ceil(boundingBox.height * scale2)));
                    string sizeText_H = to_string(static_cast<int>
(ceil(sqrt(pow(box1x - bottomLeft.x, 2) + pow(box1y - bottomLeft.y, 2)) *
scale3)));

                    // Add the size text next to the lines
                    putText(result3, sizeText_W, bottomLeft - Point2f(-20, 10),
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 0, 0), 1.5);
                    putText(result3, sizeText_L, topLeft - Point2f(-5, -40),
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 0, 0), 1.5);
                    putText(result3, sizeText_H, bottomLeft - Point2f(30, -30),
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 0, 0), 1.5);

                }
                else {

                    // Calculate corner points of the bounding box
                    Point2f topLeft2(boundingBox.x, boundingBox.y);
                    Point2f topRight2(boundingBox.x + boundingBox.width,
boundingBox.y);
                    Point2f bottomRight2(boundingBox.x + boundingBox.width,
boundingBox.y + boundingBox.height);
                    Point2f bottomLeft2(boundingBox.x, boundingBox.y +
boundingBox.height);

                    double distance = sqrt(pow(box2x - bottomRight2.x, 2) +
pow(box2y - bottomRight2.y, 2));
                    double realWidth = 100.0;
                    sc = realWidth / boundingBox.width; // scale of object about
width
                    real_sc = sc / scale;   // scale between reference object
and object that we wanted to calculate

                    line(result3, topRight2, bottomRight2, Scalar(0, 255, 0),
2);
                    line(result3, bottomRight2, bottomLeft2, Scalar(0, 255, 0),
2);
                    line(result3, bottomRight2, Point(box2x, box2y), Scalar(0,
255, 255), 2);

                    // Calculate the size text
                    double actualWidth = boundingBox.width * scale * real_sc;
```

```
                    double actualLength = boundingBox.height * scale2 * real_sc;
                    double actualHeight = distance * scale3 * real_sc;

                    string sizeText2_W = to_string(static_cast<int>
(ceil(actualWidth)));
                    string sizeText2_L = to_string(static_cast<int>
(ceil(actualLength)));
                    string sizeText2_H = to_string(static_cast<int>
(ceil(actualHeight)));

                    // Add the size text next to the lines
                    putText(result3, sizeText2_W, bottomRight2 - Point2f(90,
-20), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 0, 0), 1.5);
                    putText(result3, sizeText2_L, topRight2 - Point2f(-10, -80),
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 0, 0), 1.5);
                    putText(result3, sizeText2_H, bottomRight2 - Point2f(25,
-40), FONT_HERSHEY_SIMPLEX, 0.5, Scalar(255, 0, 0), 1.5);
                }

                resize(result3, result3, Size(500, 400));
                imshow("Result", result3);
            }
            // Initialize click count (for additional clicks)
            clickCount = 0;
        }
    }

}
```