Chapter 1
Introduction

1. Background and Importance

In recent years, the use of machine vision to interpret and understand visual information has been in high demand in the fields of Robotics and automation. As the advancement in technology progresses and the invention of more powerful GPUs the limit for intelligence for machine vision has been pushed far back, not only skill wise but also by reducing execution time of highly complex algorithms, which always allows for smooth real time processing of complex tasks. Such applications is very important in multiple fields and domains. For example In agriculture, a robot with the help of a mounted camera can be used to assess the ripeness of fruits or vegetables or other produce, therefore classify whether they are ready for harvesting. To achieve this, the robot must be supplied with a system that contains the algorithms necessary to accurately extract the properties of these fruits, (Size and Color are two of the most important criteria to determine maturity). Another example are autonomous control of mobile or aerial vehicles are in need of such systems.

2. Reference Study

This paper, with the use of an NVIDIA processor (GPU 128-core Maxwell + CPU Quad-core ARM A57), proposes an advanced system that extracts properties of objects from camera frames with the use of advanced techniques that requires minimal user intervention and without interaction with the environment that contains the object. There have been similar systems proposed in different domains with different applications, but each different in its own way. [1] and [2] requires a controlled background, and [3] uses a previous image of the object as reference for determining its dimensions in the current frame. The system we propose in this paper does not require a controlled environment nor a previous image to use as reference, all while achieving accurate and reliable results.

3. Goals to be achieved

- Utilize our stereo camera to achieve reliable depth perception through computing disparities, in order to perform accurate photogrammetry on camera frames.
- Train a machine learning model to be able to precisely localize the desired objects.
- Experiment with different methods of foreground extraction to completely isolate the detected objects from the images.
- Applying Naive K-Means Clustering to extract the dominant color of objects.

- Create an entire package in ROS2 that contains the algorithms and systems proposed, to allow for further improvement.

Chapter 2
Hardware and Software System Architecture

1. Introduction

The Entire System is an Integration of multiple layers made up of mostly smaller software systems. The softwares used were purposely selected to be the most advanced algorithms and techniques

2. Software Architecture

Software wise, The two main tools used is the python programming language along with its modules, as well as the Robot Operating System because of its diverse packages that allows for swift and efficient programming.

2.1. Python Programming Language

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed. The two main packages used are the OpenCV-Python and TensorFlow packages, as well as Matplotlib/Seaborn for visualization purposes.

- Opencv is an open-source python library that has a multitude of functions that are used in in computer vision and image processing applications, its considered the most advances library that serves this purpose.
- TenserFlow is an open-source software for Machine leaning and Artificial Intelligence applications, implemented in python. It's particularly articulated towards training and inferring deep neural networks.

2.2. Robot Operating System (ROS)

As the official ROS documentation recites, ROS is a collection of open-source software libraries and tools that provide a flexible framework for developing and running Robotics applications. It is intended to be the primary development platform for robots that operate in complex, dynamic environments, and it is designed to be an open, scalable, and Interoperable framework for building robot applications.

ROS2 is the successor to ROS (Robot Operating System), which was created by Willow Garage, a robotics research institute, in 2007. ROS was created to provide a common framework for developing robotics applications, and it has since grown in popularity as a platform for building robots in research and industry. ROS2 expands on ROS's strengths while also introducing new features and capabilities to meet the evolving needs of the robotics community.

3. Hardware Architecture

3.1. NVIDIA Jetson Nano

Since the project involves deep information processing that consumes an extravagant amount of computing power, the NVIDIA Jetson Nano 4g was chose as the processor for the project since it has a strong CPU (Quad-core ARM Cortex-A57 MPCore processor) as well as a GPU (NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores)

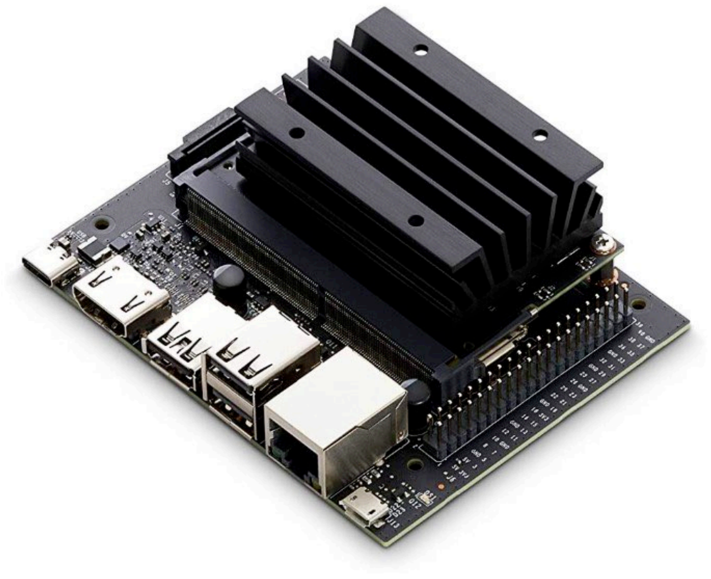


Fig 2.1. Jetson Nano

The specs of the Jetson Nano are displayed In the table below:

GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA® cores
CPU	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	16 GB eMMC 5.1
Connectivity	Gigabit Ethernet, M.2 Key E
Display	HDMI 2.0 and eDP 1.4
USB	4x USB 3.0, USB 2.0 Micro-B
Communication Protocols	GPIO, I2C, I2S, SPI, UART

Table 2.1. Jetson Nano Specs

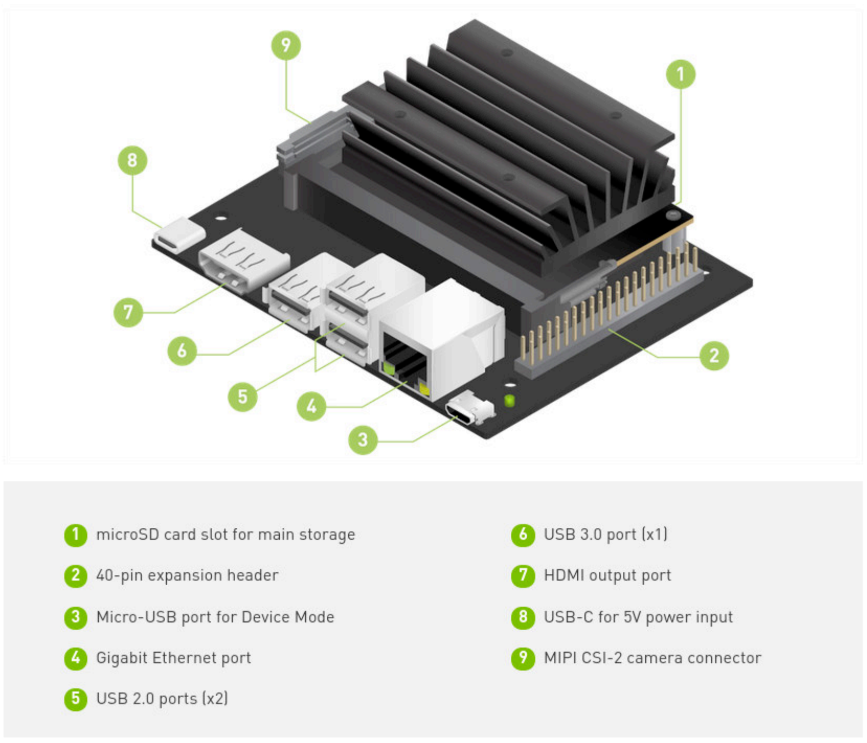

9
1
8
7
6
5
4
3
2
1 microSD card slot for main storage
2 40-pin expansion header
3 Micro-USB port for Device Mode
4 Gigabit Ethernet port
5 USB 2.0 ports (x2)
6 USB 3.0 port (x1)
7 HDMI output port
8 USB-C for 5V power input
9 MIPI CSI-2 camera connector

Fig 2.2. Jetson nano I/O

3.2. Stereo Camera

A Stereo Camera is a camera with two or more lenses with a separate image sensor for each lens. This allows the camera to stimulate human binocular vision, Therefore allowing it to capture 3D images by computing depth perception. In a nutshell, Stereo Vision enables us with depth perception by generating multiple points of view of the same scene. It's widely used in applications where photogrammetry is applied, as it has generated more accurate results then any other off-the-shelf sensors.

3.2.1. The Stereo Camera Used

The Stereo Camera used in the project is (Camera Binoculars 3D-1MP02)

The camera outputs a full stereo image that is a combination of the left and right camera frames (640 x 240 for each frame for one camera). The Following table contains the main specifications of the camera.

Camera	3D-1MP02-V92
Sensor	OV9750
Lens Size	1/3 Inch
Baseline	6.5 cm
Pixel Size	3.75um X 3.75um
Output Image Format	MJPEG
Signal to Noise Ratio	39dB
Sensitivity	3.7V/lux-sec at 550nm
Minimum illumination	0.1 lux
Supported Resolutions	640*240 pixels at 60FPS 1280*48 pixels at 60FPS 2560*72 pixels at 60FPS 2560*96 pixels at 60FPS

Table 2.2. Camera Binoculars 3D-1MP02 Specifications

3.2.2. Stereo Geometry

The definition of Stereo Geometry is the extraction of the mathematical formulas that can be applied to extract the depth of a desired object or point in spade from a reference plane. The Following diagram illustrates the Stereo Geometry.

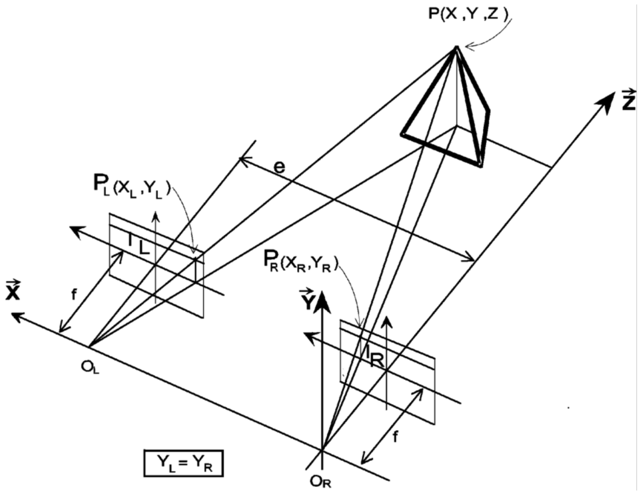


Fig 2.3. Stereo Vision Geometry

From the previous diagram:

OL and OR are the centers of each camera, f is the focal length of the camera, e is the baseline or the horizontal Distance between both cameras and z is the depth of the target object from the camera. From the previous geometry, we can extract the mathematical formulas that we can apply:

$$Disparity = abs(X_R - X_L)$$

$$baseline = abs(X_R - X_L) = e$$

$$Depth = \frac{Baseline * FocalLength}{Disparity} = z$$

4. Conclusion

The tools mentioned in this chapter will all be used to to design and implement our algorithms and methods to ensure the best possible results are to be achieved.

Chapter 3
Methodology

1. Introduction

The system contains a multi stage algorithm that outputs a crisp isolation of the object from the image before beginning to extract properties. The algorithm begins by detecting and classifying the target object and outputs a bounding box surrounding it. Using the bounding box outputted, an advanced foreground extraction algorithms utilizes it to completely extract the object from the image and that is to perform photogrammetry and color extraction accurately. After a crisp isolation is outputted, Photogrammetry is performed using a Stereo vision system that makes use of the relationship between disparity and object length in pixels. As for Color Extraction, K-Means Clustering is implemented to extract the dominant color in the target object. Since size and color the two most vital attributes to determine fruit quality and grade, throughout this paper, the system will be tested on fruits. Specifically, Lemons and Strawberries.

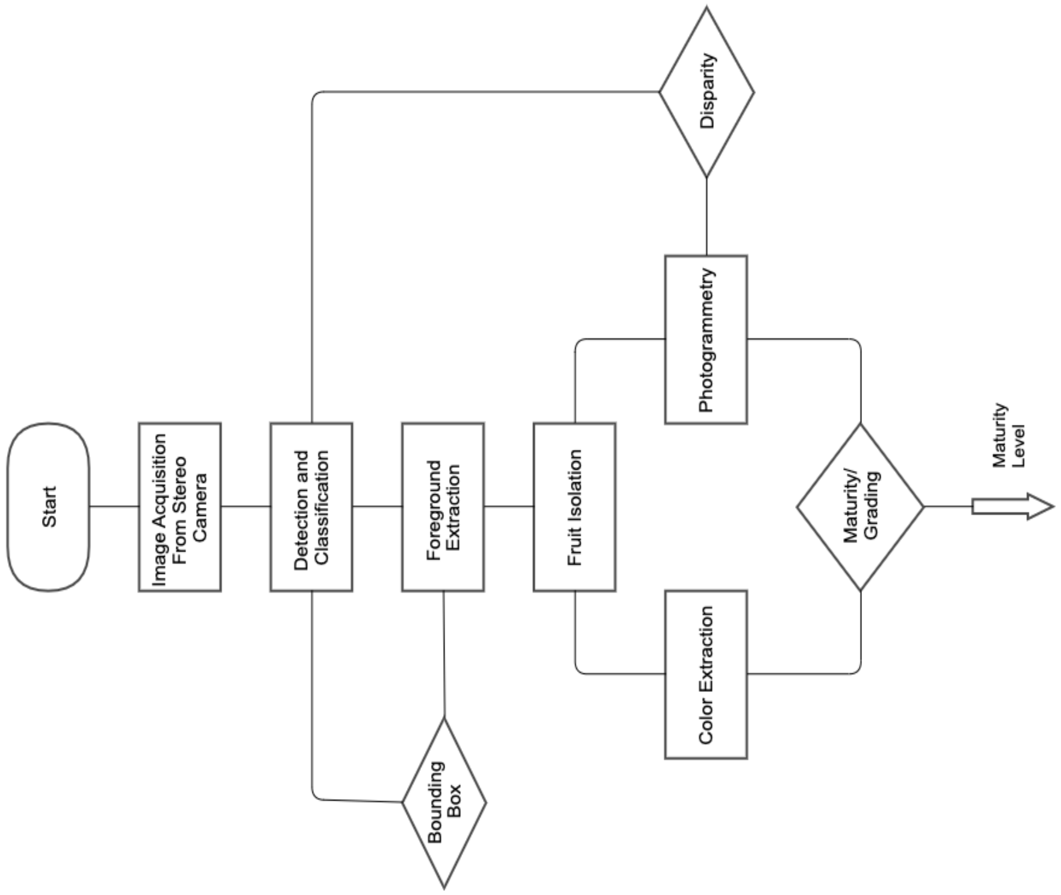


Fig 3.1. Flowchart of Proposed System

2. Image Preprocessing

Before supplying the algorithm with the frames, we need to first perform image preprocessing on it, that includes:

- Frame/Plane Rectification
- Splitting The Stereo Frames
- Removing all distortions from the image
- Applying basic filtering to boost image quality overall.

Non rectified frames as well as any distortions present in the image will severely negatively impact the results of the photogrammetry leading to an increased average error in measurements as well as the disparity computed, therefore its important for the image preprocessing stage to output a crisp, rectified and undistorted image

2.1.Frame/Plane Rectification

Stereo rectification is an important step in stereo vision systems where frame projected by the cameras are rectified onto one common planes in order to perform correlation and calculate the disparity accurately. Even though the stereo camera used had both lenses parallel to each other we still went ahead and rectified the frames in order to be certain that the frames are indeed on the same plane. The rectification process is as follows:

First we detect key points and features In both frames, then we choose the best keypoints to ensure they match in order to calculate the reprojection matrices. After the reprojection matrices are extracted we can rectify both images onto a common plane. The below images visualizes the transformation from unrectified frames to rectified ones using keypoint matching.

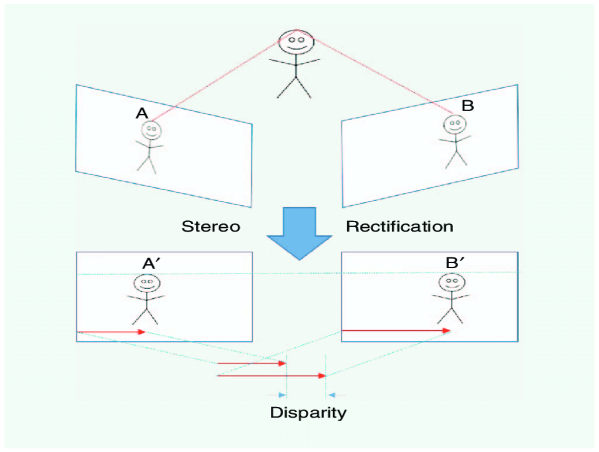


Fig 3.2. Stereo Rectification Visualization

2.2.Removing Distortions

Distortions is the present of curved or unnatural lines in the frame, but in reality they are straight. Distortion is often the result of the lens' geometrics and can significantly disrupt the image's quality. To remove all distortions present in the image we need to Calibrate our camera. The point of camera calibration is to extract the intrinsic and extrinsic parameters as well as the focal length of the camera, which can be used to remove the distortions. Camera calibration was performed using the standard checkerboard pattern method.

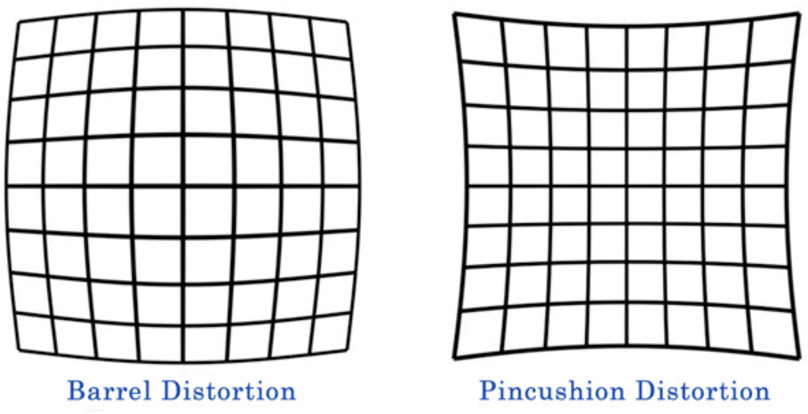


Fig 3.3. Types of distortions

3. Detection and Classification

For the system to be effective, any strong neural network models that classifies and detects objects within images and outputs a bounding box on said object will work. Models such as R-CNN[4] as well as its evolved versions of Fast R-CNN[5] and Faster R-CNN[6] will do fine. For this paper the YOLO(You Only Look Once)[7] Network is implemented for the detection and Classification stage, since it has shown an effective results in the detection of various target objects as well as fruits, as the following papers have proven [8][9].

3.1.YOLOv5 Algorithm

The Yolo Algorithm is a single stage object detector, whose architecture is made up of three main components: Backbone, head and a neck to make its prediction.

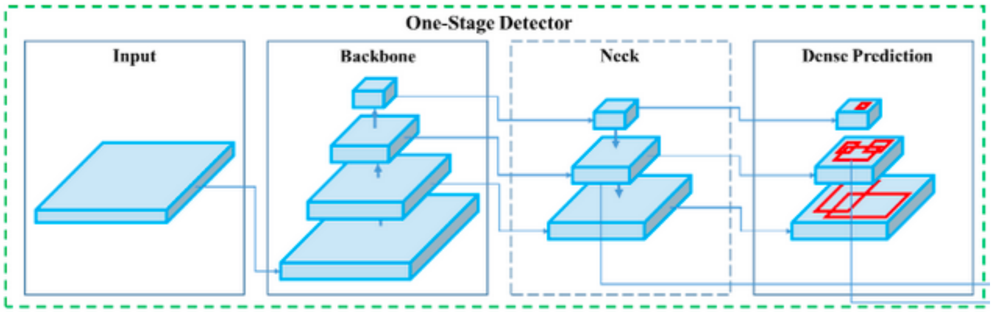


Fig 3.4. One Stage Detectors' Architecture.

Model Backbone: The Backbone is a pre-trained network used to extract rich feature representation for images. This helps to reduce the spatial resolution of the image and increase its feature (channel Resolution). Yolov5 Uses CSP-Darknet53 as its backbone, CSP-Darknet53 is a CNN called Darknet53 which is 53 layers deep, with cross stage partial (CSP) network strategy applied to it. CSP helps to reduce the excessive amount of redundant gradient information by truncating the gradient flow.

Model Neck: The Neck is used to extract feature pyramids, this helps the model to generalize well on different sizes and scales. Yolov5 uses Path Aggregation Network(PANet) and Spatial Pyramid Pooling (SPP) as the model neck.

Model Head: The head is used to perform the final stage of operations, it applies anchor boxes on feature maps and render the final output: Classes, Confidence and Bounding Box. The algorithm uses three convolution layers to make this prediction.

3.2. Activation Function

Activation functions are used to determine the output of a natural network by applying mathematical transformations to the input signals received from other layers of the network, choosing the correct activation function is that determines the overall efficiency and reliability of a trained neural network. The Yolov5 algorithms utilizes two activation functions, SiLU and Sigmoid Activation Function. The Sigmoid Linear Unit (SiLU) function is used with the Convolution operations in the hidden layer, whereas the Standard Sigmoid function is used with the Convolution operations in the Output Layer.

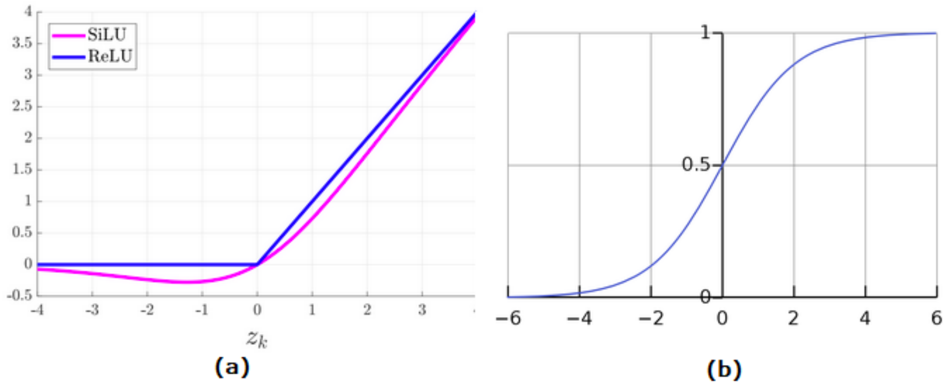


Fig3.5. (a) SiLU Activation Function, (b) Sigmoid Activation Function

The mathematical functions for both of these functions are:

$$ReLU(x) = \frac{1}{1 + e^{-x}}$$

$$SiLU(x) = x * RelU(x)$$

3.3.Loss Function

The Yolov5 Algorithm generates three different outputs: The class of the object, its confidence and a bounding box surrounding the object. It uses BCE (Binary Cross Entropy) To compute the Class loss as well as the confidence Loss, and for computing the Bounding Box Location loss it uses CIoU (Complete Intersection over Union). The Formula for the Final Loss is Given as:

$$Loss = \lambda_1 L_{cls} + \lambda_2 L_{cnf} + \lambda_3 L_{loc}$$

4. Foreground Extraction

4.1. Introduction

Foreground Extraction/Background Subtraction is Isolating the Object completely from the image. The previous stage may be solely enough to isolate the object from the image, but in order for a more robust isolation, further processing is required. The Detection Stage outputs a Rectangular Bounding Box, So the next stage will be to isolate the object from within the bounding box. This can be achieved by various methods, the most obvious is to apply a binary mask (Thresholding) within the bounding box but for varying backgrounds with different lighting that is not always the most effective (The results of binary masking will be discussed later on). The best method to perform Foreground extraction in our case is by using the GrabCut Algorithm[10].

4.2. Grabcut Algorithm

The Grabcut is a very powerful iterative model that takes a bounding box which fully encloses the foreground region as input. Afterwards, a gaussian mixture model GMM is used to model the foreground and background. A graph is generated depending on the learning of the GMM model and it's pixel distribution output. Nodes in the graphs are pixels. Additional two nodes are added, Source node and Sink node. Every foreground pixel is connected to Source node and every background pixel is connected to Sink node. The weights of edges connecting pixels to source node/end node are defined by the probability of a pixel being foreground/background. The weights between the pixels are defined by the edge information or pixel similarity. If there is a large difference in pixel color, the edge between them will get a low weight. Then a mincut algorithm is used to segment the graph. It cuts the graph into two separating source node and sink node with minimum cost function. The cost function is the sum of all weights of the edges that are cut. After the cut, all the pixels connected to Source node become foreground and those connected to Sink node become background. The process is continued until the classification converges.

5. Photogrammetry

Extracting actual dimensions from lengths in pixels with varying depth from the camera is not possible with a single camera. Of course, it is possible to use a laser sensor to measure the depth of the object, but the point is not to determine the depth of the object from the camera, but rather Measure between two significant points from the image. This is why the best option is to use a stereo vision system that contains two or more cameras. Since our stereo vision setup gives us two different perspectives of the same object, photogrammetry can be computed by making use of the relationship between the disparity extracted from both frames and the desired length measured in pixels and use it to convert the measured length in pixels to actual length in cm in real time. If L is the actual length in cm and α is some coefficient and l is the length measured in pixels, then:

$$L = \alpha l$$

This converts the measured length in pixels to the actual length in cm, The goal will be to find the value of α. since the length in pixels varies depending on the depth, that means the value of α also varies. To extract the function that represents the relationship between α and the computed disparity, an experiment is conducted. The results will be discussed in the later chapter.

5.1. The Experiment

We'll use an Aruco Marker which Is basically a distinct figure that is easy to detect and track in camera frames, similar to a barcode. The width of the Aruco marker is measured and used as the reference length in our experiment, this is the Length L. The stereo camera is mounted in a fixed spot and the marker is placed 5cm in front of the camera. Initially the Disparity (D) is Calculated from the two frames that contains the detected marker, the value of the disparity is noted down. After that, we will measure the width of the detected marker in one of the frames, this length is l. Now in this case, for the specific value D we can compute α using the equation above. Now we have our fist point (D,α). The distance of the marker from the camera is incremented by 1 cm each time and the same process is repeated. The accumulated points are then plotted on a graph and a using a Regression Algorithm we will extract the function that represents the relationship between our two independent variables, disparity and α coefficient. After a large amount of data is collected, we plot all the points and apply a curve fitting technique called dogbox to extract the function (f(disparity) = α) that represents the relationship between disparity and α. Afterwards, anytime we want to perform photogrammetry, we substitute the disparity in our function and we get the value α that me can multiply with the length in pixels to get the length in cm. The results of this experiment will be discussed later on.

6. Dominant Color Extraction

Extracting the dominant color of the object is an essential property to extract, as many applications require the accurate extraction of the object's color for decision making. This can be achieved in a number of ways, the most basic is by extracting the color straight from the histogram of the frame, but with RGB purposes it becomes a little complicated. An advanced alternative to achieve reliable output is using either K-means clustering or the K-median clustering.In our system, we propose applying the K-Means Clustering Algorithm for this purpose as it has shown more effectiveness, the results of both will be discussed later.

6.1. Naive K-Means Clustering

K-means Clustering is a form of an unsupervised machine learning model that maps n observations into k clusters of color, in which each observation belongs to the cluster with the nearest mean which is often referred to as cluster center or cluster centroid. K-means clustering minimizes within-cluster variances using squared euclidean distances.

Chapter 4
Results and Discussions

1. Introduction

The previous demonstrated the algorithm and techniques applied to achieve our goals, This chapter shows the results of each one of these applications and also discusses their output. Kindly view Appendix A if you are interested in the coding of the algorithm and its individual stages.

2. Camera Calibration

The results of the camera calibration were as follows:

Parameters	Left Camera	Right Camera
Camera Matrix	$A = \begin{pmatrix} 273.825 & 0 & 170.776 \\ 0 & 273.216 & 122.397 \\ 0 & 0 & 1 \end{pmatrix}$	$A = \begin{pmatrix} 273.618 & 0 & 183.712 \\ 0 & 272.298 & 129.469 \\ 0 & 0 & 1 \end{pmatrix}$
Distortion Coefficients	[-0.082 0.545 -0.079 0.003 -1.126]	[-0.012 0.646 -0.001 0.006 -0.946]

table 4.1. Camera Calibration Result

3. Foreground Extraction

The GrabCut algorithms was applied to the To the output of the detection and classified stage, with the following parameters:

The result was as follows

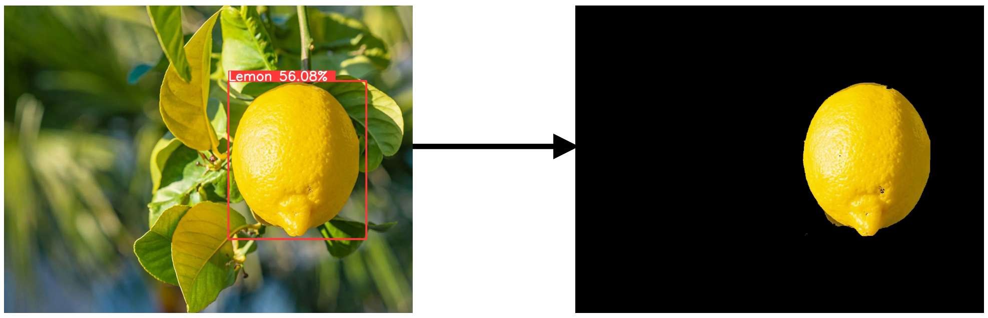


Fig 4.1. GrabCut Result

3.1. Discussion

As shown in the image, the results is very exceptional and completely isolates the objects from the image. The only downside to the algorithm is that is it is considerably slow and time consuming which would drastically effect real time applications on live frames. To make the algorithm more efficient we could decrease the amount of pixels in the image by altering its resolution, but since our stereo camera outputs two frames each 360x240 pixels, then decreasing the resolution would result in serious loss in features. So, the best way is to not supply a fixed number of iterations to the algorithm, but rather comparing the difference between the foreground extracted in the current iteration of the segmentation with the previous one, if it is less than a threshold value of 8% we stop iterating further and output the current segmentation as the final result. This method decreases the efficiency by a almost 5% but results in 14% faster Execution time.

4. Photogrammetry

4.1. Results

The results of the experiment explained previously in shown in the following table

Disparity [pixels]	α **[cm/pixel]**
122	0.0294
108	0.0344
96	0.0392
88	0.0431
82	0.04797
76	0.05277
72	0.0565
67	0.0612
63	0.0673
61	0.0698
58	0.0748
57	0.0805

Disparity [pixels]	α [cm/pixel]
55	0.08333
51	0.08878
51	0.0959
48	0.0979
47	0.1021
46	0.1079
44	0.1130
43	0.11875
42	0.1233

Table 4.2. Results of the experiment

The following plot visualizes the table above:

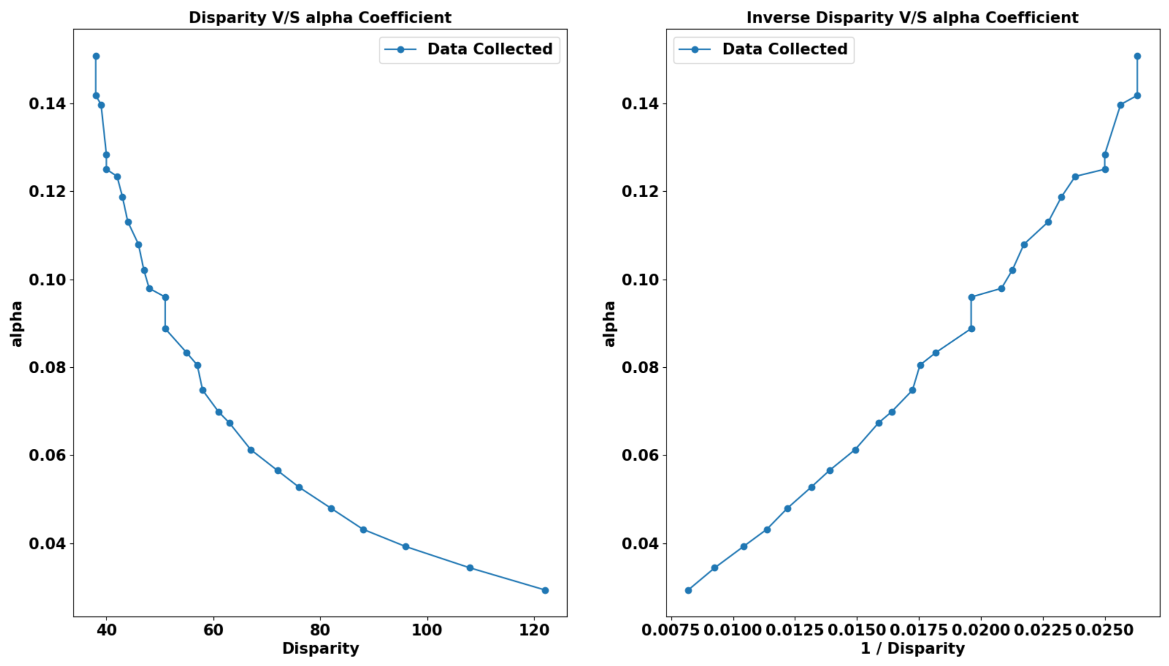


Fig 4.2. Result Visualization

To extract the function that represents the relationship between the two variables, we need to Curve Fit these points (Inverse Disparity V/S alpha is linear, So standard linear regression will do) . The conventional way of performing curve fitting is using the least squares method, this method is considered the most common. The least square method is a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets ("the residuals") of the points from the curve. As is it is clear in the following plot, this method will not work efficiently

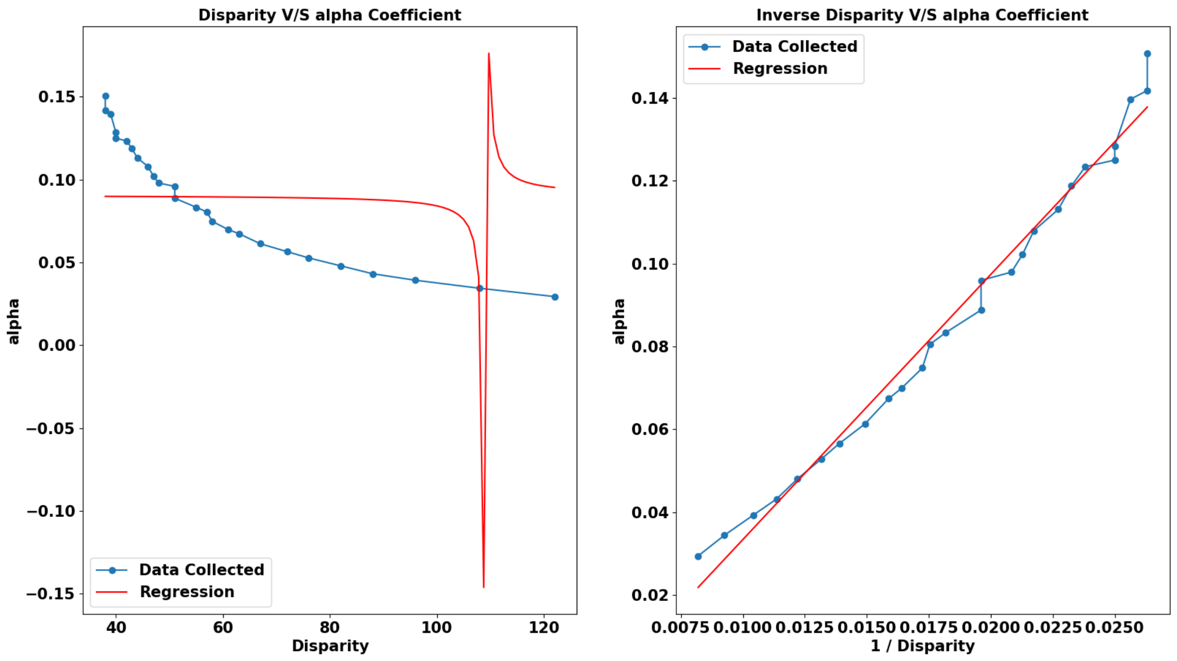


Fig 4.3. Regression with the traditional method.

Linear Regression works like a charm, but the Curve fitting is terrible. The Reason for that is the number of observations in our data is less than the number of variables, and that's because we want to fit our data to the function $f(x) = \dfrac{a}{x-b} + c$. So instead, we will have to use either the dogbox or the trf method for optimization. The dogbox algorithm does trust-region iterations, the shape of trust regions is rectangular as opposed to conventional elliptical. The intersection of a trust region and an initial feasible region is again some rectangle. Thus, on each iteration a bound-constrained quadratic optimization problem using a specific quadratic formula.

<u>How the The Curve Fit function was chosen</u>: we can take reference on the relationship between disparity and depth. It's known that $depth = \dfrac{bf}{disparity}$ where b is the baseline that represents

the horizontal distance between both lenses of the stereo vision camera, and f is the focal length. The relationship between the disparity and depth is a reciprocal function of $f(x) = \frac{a}{x}$, so for our data we try to fit the curve with a more generalized reciprocal function of $f(x) = \frac{a}{x - b} + c$ using the dogbox method mentioned above.

The Result of the curve fitting algorithm using the dogbox method and the function $f(x) = \frac{a}{x - b} + c$ as input, we get the following result

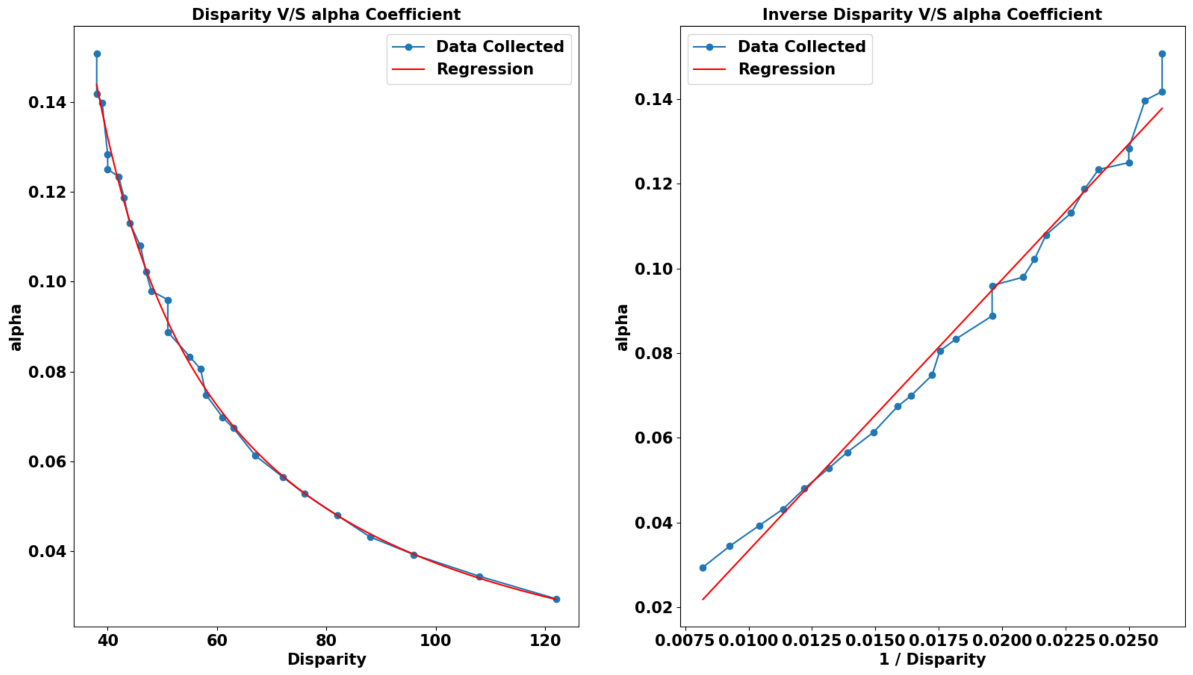


Fig 4.4. Final Result of Regression using dogbox method

It's clear that the results are much better and more reliable to use than the previous output.

The output function for the (Disparity - alpha) data is:

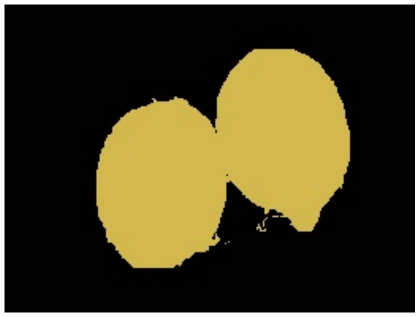
$$alpha = \frac{3.34713891}{disparity - 15.0538816} - 2.02645145e - 03.$$

Whereas the linear function for the (Inverse Disparity - alpha) data is

$$alpha = 6.402 * (1/disparity) - 0.03065.$$

5. Dominant Color Extraction

The Means clustering Algorithm was Applied to an image that shows two lemons on a tree waiting to be harvested, the dominant colors were extracted and visualized as follows:



6. Discussion

The algorithm carries out all the stages and extracts the color clusters from the output of the isolation stage. The result of the cluster is highly reliable to use as criteria to determine maturity/ripeness, and is an excellent color extraction overall.

Chapter 5
References

[1] S. M. T. Islam, M. Nurullah and M. Samsuzzaman, "Mango Fruit's Maturity Status Specification Based on Machine Learning using Image Processing," 2020 IEEE Region 10 Symposium (TENSYMP), Dhaka, Bangladesh, 2020, pp. 1355-1358, doi: 10.1109/TENSYMP50017.2020.9230951

[2] M. Muladi, D. Lestari and D. T. Prasetyo, "Classification Of Eligibility Consumption Of Manalagi Apple Fruit Varieties Using Backpropagation," 2019 International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA), Batu, Indonesia, 2019, pp. 75-79, doi: 10.1109/ICAMIMIA47173.2019.9223398.

[3] I. B. Mustaffa and S. F. B. M. Khairul, "Identification of fruit size and maturity through fruit images using OpenCV-Python and Rasberry Pi," 2017 International Conference on Robotics, Automation and Sciences (ICORAS), Melaka, Malaysia, 2017, pp. 1-3, doi: 10.1109/ICORAS.2017.8308068.

[4] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[J]. Computer Science,2013:580-587

[5] Girshick R. Fast R-CNN[C]//IEEE International Conference on Computer Vision.IEEE,2015:1440-1448.

[6] Ren S,He K,Girshick R,et al.Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J].IEEE Transactions on Pattern Analysis & Machine Intelligence, 2015:1-1.

[7] Redmon J, Farhadi A. YOLO9000:Better, Faster, Stronger.arXiv preprint arXiv:1612.08242v1,2016.

[8] W. Lan, J. Dang, Y. Wang and S. Wang, "Pedestrian Detection Based on YOLO Network Model," 2018 IEEE International Conference on Mechatronics and Automation (ICMA), Changchun, China, 2018, pp. 1547-1551, doi: 10.1109/ICMA.2018.8484698.

[9] K. R. B. Legaspi, N. W. S. Sison and J. F. Villaverde, "Detection and Classification of Whiteflies and Fruit Flies Using YOLO," 2021 13th International Conference on Computer and Automation Engineering (ICCAE), Melbourne, Australia, 2021, pp. 1-4, doi: 10.1109/ICCAE51876.2021.9426129.

[10] C. Rother, V. Kolmogorov, and A. Blake, GrabCut: Interactive foreground extraction using iterated graph cuts, ACM Trans. Graph., vol. 23, pp. 309–314, 2004.

Appendix A
Python Code

Frame.py

```
from Detection import YOLO
import Isolation
from SizeExtraction import SizeExtraction
import cv2
import numpy as np
import os
path = os.path.realpath(os.path.dirname(__file__))
path = path.replace('src', '')

class FRAME(YOLO, SizeExtraction):
    def __init__(self):
        self.GrabcutIsolation = None
        self.Detections = None
        self.frame = None
        super().__init__(conf=0.4)



    def ProcessFrame(self, frame):
        #Split Stereo
        # h, w = frame.shape[:2]
        # left = frame[:, :w//2]
        # self.frame = left
        self.frame = frame
        self.GrabcutIsolation = frame
        self.height, self.width = self.frame.shape[:2]
        self.getDetections(self.frame)
        self.getDisparity()
        self.getIsolation()

    def getDetections(self, frame):
        # Detection And Classification
        pred, im, im0 = self.Detect(frame)
        self.Detections = self.getParameters(pred, im0)
        self.AnnotatedFrame, self.XYXY = self.Annotate(pred, im, im0)
        self.XYWH = self.XYXYtoXYWH(self.XYXY)

    def NormalizeWidth(self, x):
        if x >= self.width // 2:
            return int(x - self.width // 2)
        return x

    def getDisparity(self):
        if len(self.Detections) == 2:
            [x1, x2] = self.XYWH[0][0], self.XYWH[1][0]
            x1 = self.NormalizeWidth(x1)
            x2 = self.NormalizeWidth(x2)
            self.disparity = abs(x1 - x2)

    def getIsolation(self):
        if len(self.Detections) > 0:
            # Isolation
            self.BigXYXY = self.XYXYtoBB(self.XYXY)
            self.BigXYWH = self.XYXYtoXYWH(self.BigXYXY)
            self.BigBoundingBoxIsolation = Isolation.IsolateBoundingBox(self.frame, self.BigXYWH)
            self.BoundingBoxIsolation = Isolation.IsolateBoundingBox(self.frame, self.XYWH)
            self.GrabcutIsolation = Isolation.GrabCutIsolation(self.frame, self.BigXYWH)
            self.ThresholdedImage, self.CrispThreshold = self.Threshold(self.GrabcutIsolation)

            return self.GrabcutIsolation
        return None

    def getPhotogrammetry(self):
        isolated = self.getIsolation()
        if isolated is not None and len(self.Detections) == 1:
            # Size extraction
            self.quadrants = self.getQuadrants(isolated, self.XYWH)
            self.VisualizedQuadrants = self.VisualizeQuadrants(self.frame, self.quadrants)
            top, bottom, left, right = self.quadrants[0]
            print(f'THE LENGTH: {self.getLengths(top, bottom, self.disparity)}')
            return self.VisualizedQuadrants

        return None
```

```
def Threshold(self, img,  val=100, kernel=np.ones((15, 15), np.uint8)):
    frame = np.copy(img)
    if frame.ndim == 3:
        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    _, ThresholdedImage = cv2.threshold(frame, val, 255, cv2.THRESH_BINARY)
    CrispThreshold = cv2.morphologyEx(ThresholdedImage, cv2.MORPH_OPEN, kernel)
    return ThresholdedImage, CrispThreshold

def BGRtoRGB(self, img):
    return img[...,::-1].copy()
```

Detection.py

```
import sys
sys.path.insert(1, '/Users/baherkherbek/Desktop/Graduation/yolov5-master')
sys.path.insert(1, '/Users/baherkherbek/Desktop/Graduation/weights')

from utils.torch_utils import select_device
from models.common import DetectMultiBackend
from utils.general import xyxy2xywh, check_img_size, non_max_suppression, Profile, scale_boxes
from utils.augmentations import letterbox
from utils.plots import Annotator, colors, save_one_box
import numpy as np
import torch
import cv2

class YOLO():
  def __init__(
        self,
        imgsz=320,
        half=False,
        device='cuda' if torch.cuda.is_available() else 'cpu',
        conf=0.25,
        iou=0.45,
        classes=None,
        agnostic=False,
        max_det=1000,
        line_thickness=5

  ):
    self.weights = '/Users/baherkherbek/Desktop/Graduation/weights/best2.pt'
    self.data = 'cus.yaml'
    self.imgsz = (imgsz, imgsz)
    self.half = half
    self.model = None
    self.device = select_device(device)
    self.conf_thres = conf
    self.iou_thres = iou
    self.classes = classes
    self.agnostic_nms = agnostic
    self.max_det = max_det
```

```
        self.line_thickness = line_thickness
        self.device = select_device(self.device)
        self.model = DetectMultiBackend(self.weights, device=self.device, dnn=False, data=self.data, fp16=self.half)
        self.imgsz = check_img_size(self.imgsz)

    def Detect(self, img):
        frame = np.copy(img)
        self.model.warmup(imgsz=(1 if self.model.pt or self.model.triton else 1, 3, *self.imgsz))
        dt = (Profile(), Profile(), Profile())

        im0 = frame
        im = letterbox(im0, self.imgsz, stride=self.model.stride, auto=True)[0]
        im = im.transpose((2, 0, 1))[::-1]
        im = np.ascontiguousarray(im)

        with dt[0]:
            im = torch.from_numpy(im).to(self.model.device)
            im = im.half() if self.model.fp16 else im.float()  # uint8 to fp16/32
            im /= 255  # 0 - 255 to 0.0 - 1.0
            if len(im.shape) == 3:
                im = im[None]  # expand for batch dim

        # Inference
        with dt[1]:
            pred = self.model(im, augment=False)

        with dt[2]:
            pred = non_max_suppression(pred, self.conf_thres, self.iou_thres, self.classes, self.agnostic_nms,
                                       max_det=self.max_det)

        return pred, im, im0

    def getParameters(self, pred, image0):
        im0 = np.copy(image0)
        gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
        det = pred[0]

        objects = []
        for *xyxy, conf, cls in reversed(det):
            xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
            object = (cls, *xywh, conf)
            objects.append(object)
            #li.append(xyxy)

        return objects

    def Annotate(self, pred, image, image0, hide_labels=False, hide_conf=False, additionalText=""):
        im0 = np.copy(image0)
        im = np.copy(image)
        annotator = Annotator(np.ascontiguousarray(im0), line_width=self.line_thickness,
example=str(self.model.names))
        det = pred[0]
        XYXY = []
        if len(det):
            # Rescale boxes from img_size to im0 size
            det[:, :4] = scale_boxes(im.shape[2:], det[:, :4], im0.shape).round()

        for *xyxy, conf, cls in reversed(det):
            c = int(cls)
            label = None if hide_labels else (
                self.model.names[c] if hide_conf else f'{self.model.names[c]} {100 * conf:.2f}% {additionalText}')
            annotator.box_label(xyxy, label, color=colors(c, True))
            im0 = annotator.result()
            xyxy = list(map(int, xyxy))
            XYXY.append(xyxy)

        return im0, XYXY

    def showXYXY(self, frame, XYXY, COLOR=(0,255,255)):
        img = np.copy(frame)
        for Object in XYXY:
            p1 = (Object[0], Object[1])
            p2 = (Object[2], Object[3])

            cv2.circle(img, p1, 5, COLOR, -1)
            cv2.circle(img, p2, 5, COLOR, -1)

        return img
```

```
def XYXYtoXYWH(self, BBX):
    Points = np.copy(BBX)

    for idx, point in enumerate(Points):
        width = point[2] - point[0]
        height = point[3] - point[1]
        XYWH = [point[0], point[1], width, height]
        Points[idx] = XYWH

    return Points

def XYXYtoBB(self, xywh):
    x1 = np.amin(xywh, axis=0)[0]
    y1 = np.amin(xywh, axis=0)[1]
    x2 = np.amax(xywh, axis=0)[2]
    y2 = np.amax(xywh, axis=0)[3]
    XYXY = (x1, y1, x2, y2)

    return [XYXY]
```

Photogrammetry.py

```
import cv2
import numpy
from Isolation import IsolateBoundingBox
#from frame import FRAME
import math
import os

# path = os.path.realpath(os.path.dirname(__file__))
# path = path.replace('src', 'data/')
# popt_alpha = numpy.load(path + 'popt_alpha_new.npy')
path = '/home/baher/Desktop/projects/Graduation/data/popt_alpha_new.npy'
popt_alpha = numpy.load(path)

class SizeExtraction():
    def __init__(self):
        pass

    def RegressionFunc(self, x, a, b, c):
        return (a / (x - b)) + c
    def getQuadrants(self, IsolatedImage, XYWH):
        gray = numpy.copy(IsolatedImage)

        gray = cv2.cvtColor(gray, cv2.COLOR_BGR2GRAY)
        quadrants = []
        i = 0
        for xywh in XYWH:
            isolated = IsolateBoundingBox(gray, [xywh])

            # Vertical
            height = xywh[1] + xywh[3] // 2
            slice = isolated[height, :]
            NonZero = numpy.where(slice != 0)
            isolated[height, :] = 255
            left, right = (numpy.min(NonZero), height), (numpy.max(NonZero), height)

            # Horizontal
            width = xywh[0] + xywh[2] // 2
            slice = isolated[:, width]
            NonZero = numpy.where(slice != 0)
            isolated[:, width] = 255
            up, down = (width, numpy.min(NonZero)), (width, numpy.max(NonZero))

            quadrants.append([up, down, left, right])

        return quadrants

    def VisualizeQuadrants(self, img, quadrants, COLOR=(0, 255, 255), THICKNESS=1):
        frame = numpy.copy(img)

        for quadrant in quadrants:
            top, bottom, left, right = quadrant

            if THICKNESS is not None:
                frame = cv2.line(frame, top, bottom, COLOR[::-1], THICKNESS)
                frame = cv2.line(frame, left, right, COLOR, THICKNESS)
            else:
                frame = cv2.circle(frame, top, 5, COLOR, -1)
                frame = cv2.circle(frame, bottom, 5, COLOR, -1)
                frame = cv2.circle(frame, left, 5, COLOR[::-1], -1)
                frame = cv2.circle(frame, right, 5, COLOR[::-1], -1)
        return frame

    def getLengths(self, p1, p2, disparity):
        #get angle:
        opposite = p1[1] - p2[1]
        adjacent = p1[0] - p2[0]
        slope = (abs(opposite / adjacent) if adjacent != 0 else None)
        angle = (math.pi/2 if (slope is None) else math.atan(slope))

        alpha = self.RegressionFunc(disparity, *popt_alpha)
        Length = ((alpha * adjacent) * math.cos(angle) if adjacent != 0 else (alpha * opposite))
        return abs(Length)
```

Color.py

The following Demonstrate the main functions used:

```
from sklearn.cluster import KMeans
import numpy

def KMEANS(x, numClusters):
    if x.ndim == 3:
        x = x.reshape((x.shape[0] * x.shape[1], 3))
    kmeans = KMeans(n_clusters=numClusters, n_init='auto')
    kmeans.fit(x)
    return kmeans

def VisualizeColors(img, colors, labels):
    centers = numpy.uint8(colors)
    ClusteredImage = centers[labels.flatten()]
    ClusteredImage = ClusteredImage.reshape(img.shape)
    return ClusteredImage
```