HOCHSCHULE
MITTWEIDA
University of Applied Sciences

BACHELOR THESIS

Ms.
Ngoc Uyen Nguyen

AT ONCE POST-PROCESSING OF FLUID STRUCTURES IN MICRO GRAVITY EXPERIMENT

Mittweida, September 2023

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

AT ONCE POST-PROCESSING OF FLUID STRUCTURES IN MICRO GRAVITY EXPERIMENT

Author:
Ngoc Uyen Nguyen

Course of Study:
Applied Mathematics

Seminar Group:
MA19w1-B

First Examiner:
Prof. Dr. rer. nat. Florian Zaussinger

Second Examiner:
M.Sc. Jan Auth

Submission:
Mittweida, 23.09.2023

Defense/Evaluation:
Mittweida, 2023

Faculty of **Applied Computer Sciences and Biosciences**

BACHELOR THESIS

AT ONCE POST-PROCESSING OF FLUID STRUCTURES IN MICRO GRAVITY EXPERIMENT

Author:
Ngoc Uyen Nguyen

Course of Study:
Applied Mathematics

Seminar Group:
MA19w1-B

First Examiner:
Prof. Dr. rer. nat. Florian Zaussinger

Second Examiner:
M.Sc. Jan Auth

Submission:
Mittweida, 23.09.2023

Defense/Evaluation:
Mittweida, 2023

Bibliographic Description

Nguyen, Ngoc Uyen:
AT ONCE POST-PROCESSING OF FLUID STRUCTURES IN MICRO GRAVITY EXPERIMENT. – 2023. – 38 S.
Mittweida, Hochschule Mittweida – University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences, Bachelor Thesis, 2023.

Referat

The GeoFlow II experiment aims to replicate Earth's core dynamics using a rotating spherical container with controlled temperature differences and simulated gravity. During the GeoFlow II campaign, a massive dataset of images was collected, necessitating an automated system for image processing and fluid flow visualization in the northern hemisphere of the spherical container. From here, we aim to detect the special structures appearing on the post processed images. Recognizing YOLOv5's proficiency in object detection, we apply Yolov5 model for this task.

Contents

Contents I

List of Figures III

List of Tables V

Listings VII

1 Introduction 1
1.1 Paper Overview . . . 1
1.2 Literature Review . . . 1
1.2.1 The GeoFlow II Experiment . . . 1
1.2.2 YOLOv5 . . . 2
1.2.2.1 An Overview of the YOLOv5 Architecture . . . 2
1.2.2.2 Preliminary YOLOv5 Evaluation Metrics . . . 3
1.2.2.3 An Overview of the YOLOv5 Training . . . 3
1.2.3 Ray Casting Algorithm . . . 4

2 Methodology 7
2.1 Object Detection . . . 7
2.2 Model Training . . . 7
2.2.1 Data Preparation . . . 7
2.2.1.1 Annotation . . . 7
2.2.1.2 Data Splitting . . . 7
2.2.2 Model Configuration . . . 8
2.2.2.1 Yolov5 Model Selection . . . 8
2.2.3 Environment Setup . . . 9
2.2.3.1 Data Management with GitHub . . . 9
2.2.3.2 Computing Power of Google Colab Resources . . . 9
2.2.4 Training Process . . . 9
2.2.5 Model Deployment . . . 10
2.3 Image Processing . . . 10
2.3.1 Images Sorting and Flow Visualization . . . 10
2.3.2 Rotation (Affine Transformation) . . . 11
2.3.2.1 Rotation Center . . . 12
2.3.2.2 Image Expansion . . . 12
2.3.2.3 Anticlockwise Rotation . . . 12
2.3.3 Sequential Transformation . . . 13
2.3.4 Assemble . . . 13
2.3.4.1 Combining Individual Images . . . 13
2.3.4.2 Addition Operation . . . 13
2.4 Object Detection on Post-processed Images . . . 14

3 Results 17
3.1 Trained Yolov5 Model . . . 17
3.1.1 Initial Training . . . 17

3.1.2 Optimal Training 19
3.2 Image Processing 21
3.3 Object Detection on Post-processed Images 23

4 Discussion **27**
4.1 Triangle Cut Position 27
4.2 Temporal Discrepancy Assumption 27
4.3 Angular Span Limitation 27
4.4 Pre-processed Image Optimization Insight 27
4.5 Potential Overfitting in YOLOv5 Object Detection 28
4.6 Proposal 1: Marking The Detected Objects Prior To Image Processing. 28
4.7 Proposal 2: Angle-independent Structure Detection 28

5 Annex **31**

Bibliography **37**

Statutory Declaration in Lieu of an Oath **39**

List of Figures

1.1 (a) International Space Station [1], (b) GeoFlow core [2], (c) GeoFlow container [3] 1
1.2 Architecture of an object detection network.[14] 2
1.3 The number of intersections for a ray passing from outside the polygon to any point determines its location: If the number is odd, the point is inside the polygon; if even, the point is outside. [16] 5
2.1 An example of using LabelImg tool to annotate the double-eye structure in YOLO format made of image `OPS_1165695908721` and how the lable is saved in a '.txt' file. 8
2.2 An example of a Yolov5 detected image `OPS_1165682608795` at confidence score at 0.25 and how the detected label is saved in '.txt' file. 10
2.3 A simplified presentation of merging alternately 25 'patches' in two turns 2.3a and in four turns 2.3b 11
2.4 An example of a 'triangle-masked' image in the original size 480×480 pixel. 12
2.5 An example of expanding an image to ensure a full presentation of the triangular 'patch' during image rotation. 13
2.6 Example of a two-rotation sequence. 14
2.7 Example of a four-rotation sequence 15
2.8 A full presentation of merging alternately 25 triangular 'patches' in two turns 2.8a and four turns 2.8b. 16
3.1 Two examples of the initial Yolov5m object detection. 18
3.2 Two examples of Yolov5x object detection. 18
3.3 Two examples comparing the initial Yolov5m to detect folder c20_69. 19
3.4 Two examples of the optimally trained Yolov5m to detect folder c21_95. 20
3.5 Two unsuccessful presentations of the aligned 'patches' that do not form a double-eye structure. 21
3.6 Two well-made presentations of the aligned 'patches' that do form a double-eye structure. 22
3.7 Two examples of false detection 3.7a, 3.7b and two examples of correct detection 3.7c, 3.7d made by the optimally trained Yolov5m model at confidence score 0.25. 24
3.8 Two examples of false detection 3.8a, 3.8b and two examples of correct detection 3.8c, 3.8d made by the optimally trained Yolov5m model at confidence score 0.1. 25
4.1 Proposed triangle cut position 27
4.2 Proposal 1: Marking the detected objects prior to image processing. 28
4.3 Proposedly collecting the first images to each 25-image-sequence. 29
4.4 Example collection of three images at the same position, separated by two rotations, in folder c20_95. 29

List of Tables

3.1 Performance metrics of two Yolov5 vaiants in training. 17
3.2 Two YOLOv5 variants detection result by confidence score. 17
3.3 Performance metrics of Yolov5m model in training. 19
3.4 Performance metrics of Yolov5m model in training. 19

Listings

script_sort_mid.py 31
script_sort_high.py 32
script_image_processing_25.py 33
yolov5.py 35

1 Introduction

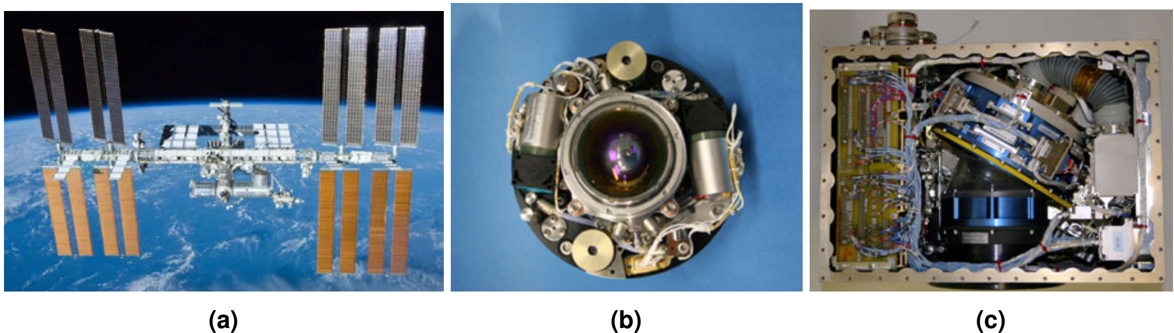
1.1 Paper Overview

The thesis is structured as follows:

- **Chapter 1** includes a literature review of the GeoFlow II experiment and its objectives, introduces the YOLO object detection model, and provides a basic overview of the ray casting algorithm.
- **Chapter 2** explains the theoretical foundations behind training the YOLOv5 model to recognize the double-eye structure within the GeoFlow II experiment dataset, together with the image processing techniques used for data visualization and how the trained YOLOv5 model performs object detection on these processed images.
- **Chapter 3** presents the key results, including the trained YOLOv5 model, post-processed images providing insights into the GeoFlow II experiment, and the outcomes of applying the YOLOv5 model to these images.
- **Chapter 4** discusses the results, and challenges faced during the thesis, and proposes potential improvements for the project.

1.2 Literature Review

1.2.1 The GeoFlow II Experiment



(a) (b) (c)

Figure 1.1: (a) International Space Station [1], (b) GeoFlow core [2], (c) GeoFlow container [3]

The GeoFlow II [4] experiment is a high-precision scientific investigation conducted aboard the International Space Station (ISS). It is designed to simulate and study the thermo-electro hydrodynamics (TED) occurring within the Earth's core. This experiment holds immense significance in the field of geophysics and Earth sciences due to its potential to provide crucial insights into the generation and maintenance of Earth's magnetic field.

The primary focus of GeoFlow II is to replicate the extreme conditions found in Earth's core, where molten iron and nickel are subject to intense heat and pressure. The core's TED processes, including the interplay between thermal convection, electrical currents, and magnetic field generation, are believed to be responsible for Earth's magnetic field. Understanding these processes is essential because the planet's magnetic field shields us from harmful solar radiation and plays a pivotal role in various geophysical phenomena.

GeoFlow II achieves this simulation by utilizing a cylindrical container filled with a low melting-point gallium alloy. The fluid within the container serves as an analog for the molten outer core of the Earth. Researchers carefully control the temperature, rotation, and magnetic field conditions to mimic the core's behavior.

The experiment's results are expected to provide valuable data on the dynamo processes occurring in Earth's core, revealing how magnetic fields are generated and maintained. These findings can contribute to a deeper understanding of the Earth's interior and its geophysical processes.

Furthermore, the GeoFlow II experiment represents a remarkable fusion of laboratory experimentation and space science. Conducting this research aboard the ISS allows scientists to eliminate the gravitational effects that would interfere with similar experiments conducted on Earth. This unique environment enables more precise observations and simulations of core processes, making GeoFlow II an essential component of Earth and space sciences research.

1.2.2 YOLOv5

The YOLOv5 [5] repository refers to the codebase and associated resources for the YOLOv5 (You Only Look Once [6] version 5) object detection model. YOLOv5 is an open-source deep learning model for real-time object detection and has gained popularity across various applications, such as traffic regulation [7] [8], civil infrastructure [9], manufacturing quality inspection [10], microfluidic droplet detection [11], brain cancer segmentation [12], marine micro-objects detection [13].

The official YOLOv5 repository is typically hosted on platforms like GitHub, making it accessible to a broad user base and encouraging collaboration and contributions from the community. Users can clone or fork the repository to customize and utilize the YOLOv5 model for their specific object detection tasks.

1.2.2.1 An Overview of the YOLOv5 Architecture

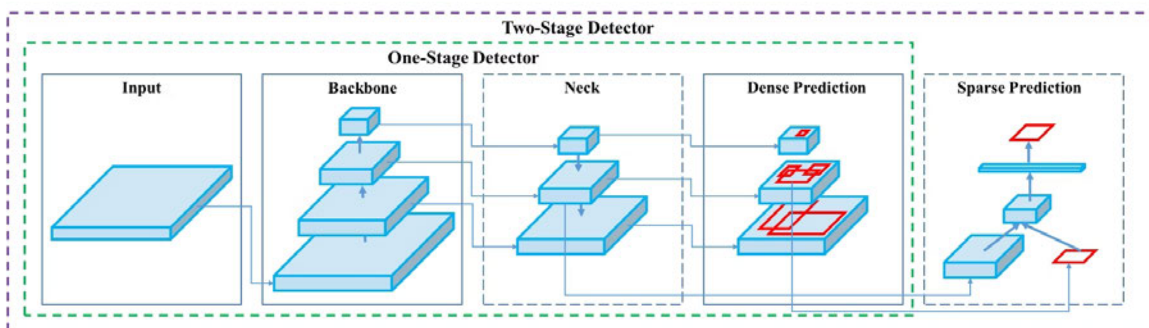


Figure 1.2: Architecture of an object detection network.[14]

The YOLOv5 architecture is a deep learning model designed for object detection. It comprises three main components:

1. **Backbone**:
   - The backbone is the initial part of the YOLO network and serves as the feature extractor. It takes the input image and processes it through a series of convolutional layers.
   - Common backbone architectures used in YOLO include Darknet and more recently, CSPDarknet and PANet. These architectures are designed to capture hierarchical features from the input image.

2. **Detection Head**: The detection head is responsible for making predictions based on the features extracted by the backbone. It typically consists of several convolutional layers and is divided into two main branches:

- Bounding Box Regression: This branch predicts bounding boxes for potential objects in the image. Each bounding box is represented by its center coordinates (x, y), width, height, and confidence score.
- Class Prediction: This branch predicts the conditional probabilities of object classes. For each bounding box, it estimates the likelihood of the detected object belonging to different predefined classes.

3. **Neck (Optional)**: In some YOLO variants, there is an optional component called the "neck" that connects the backbone and detection head. The neck can include additional convolutional layers or skip connections to further refine feature representations.

The YOLO network processes the entire image in a single pass, making it highly efficient for real-time object detection tasks. It predicts bounding boxes and class probabilities for multiple objects simultaneously, which distinguishes it from other object detection methods. This design enables YOLO to achieve both speed and accuracy in object detection across a wide range of applications.

1.2.2.2 Preliminary YOLOv5 Evaluation Metrics

Preliminary YOLOv5 Evaluation Metrics typically refer to the initial set of performance metrics used to assess the performance of the YOLOv5 model in object detection tasks. These metrics are often computed during the early stages of model development and training to provide an initial indication of how well the model is performing. Common evaluation metrics for object detection tasks include Precision (P), Recall (R), Mean Average Precision (mAP), and various variations of mAP, such as mAP at different Intersections over Union (IoU) thresholds (e.g., mAP50, mAP75).

- Precision (P) measures the fraction of true positive detections among all the positive predictions made by the model. It reflects how accurate the model's detections are.
- Recall (R) measures the fraction of true positive detections among all the actual positive instances in the dataset. It reflects how well the model is at capturing all relevant objects.
- Mean Average Precision (mAP) is a composite metric that considers the precision-recall trade-off over a range of confidence thresholds. It provides an overall assessment of the model's performance.

Preliminary YOLOv5 Evaluation Metrics help researchers and practitioners understand how well the model is performing early in the development process and guide further adjustments and optimizations. These metrics are often refined and validated in later stages of model training and evaluation.

1.2.2.3 An Overview of the YOLOv5 Training

The training loop in **YOLOv5** is a crucial component of the model training process, responsible for iteratively updating the model's weights to improve its performance in detecting objects within images. It involves several key steps and concepts, which I'll explain in paragraphs.

1. **Batch Processing**: During each iteration of the training loop, a batch of preprocessed images and their corresponding ground truth annotations is passed through the YOLOv5 model. Batch processing allows for efficient use of computational resources and helps generalize the model's learning from a variety of examples.

2. **Forward Pass**: In the forward pass, each image in the batch is processed by the model. YOLOv5 divides the input image into a grid and makes predictions for bounding boxes, class probabilities, and objectness scores within each grid cell. These predictions are generated for multiple anchor boxes, which are predefined sizes and aspect ratios.
3. **Loss Calculation**: After the model generates predictions, the loss is calculated. YOLOv5 uses a combination of loss functions, including localization loss (CIoU loss), confidence loss, and class loss. The localization loss measures the accuracy of bounding box coordinates, the confidence loss measures the accuracy of objectness predictions (i.e., whether an object exists in a bounding box), and the class loss measures the accuracy of class predictions (identifying the object's class). The losses are calculated for each grid cell and anchor box.
4. **Backpropagation**: With the loss calculated, the backpropagation process begins. This step involves computing gradients of the loss with respect to the model's weights. These gradients indicate how much each weight should be adjusted to minimize the loss.
5. **Weight Updates**: After backpropagation, the model's weights are updated using optimization algorithms like stochastic gradient descent (SGD) or its variants. The learning rate, a hyperparameter, determines the size of these weight updates. Lower learning rates can lead to slower convergence but may result in a more accurate final model.
6. **Epochs**: The training loop repeats this process for a predefined number of epochs. An epoch refers to one complete pass through the entire training dataset. Multiple epochs are typically needed to allow the model to learn and improve its performance.
7. **Validation**: Periodically, the model's performance is evaluated on a separate validation dataset. Metrics like mean average precision (mAP) are commonly used to assess how well the model detects objects and avoids false positives. This evaluation helps in monitoring the model's progress and making decisions about early stopping or fine-tuning hyperparameters.
8. **Checkpointing**: Model checkpoints are saved at regular intervals during training. These checkpoints allow us to resume training from a specific point or choose the best-performing model for inference. They also serve as backups to prevent data loss in case of training interruptions.
9. **Monitoring**: Throughout the training process, various metrics such as loss, mAP, and learning rate are monitored. These metrics provide insights into how the model is learning and help in making decisions about model selection and hyperparameter tuning.

The training loop in YOLOv5 iteratively refines the model's weights, optimizing it to accurately detect objects within images. The process involves forward and backward passes, loss calculation, weight updates, and repeated iterations through the training dataset. It is essential for achieving a well-trained and accurate object detection model.

1.2.3 Ray Casting Algorithm

The Ray Casting Algorithm [15] is a fundamental technique used in computer graphics and computational geometry. Its primary application is determining whether a point is inside a polygon, identifying intersections between rays and polygons, and performing other geometric calculations. Here's an overview of the Ray Casting Algorithm:

- **Basic Concept**: At its core, the Ray Casting Algorithm involves casting rays from a specific point (usually a test point or the origin) and counting the number of intersections between these rays and a polygon or a set of line segments.

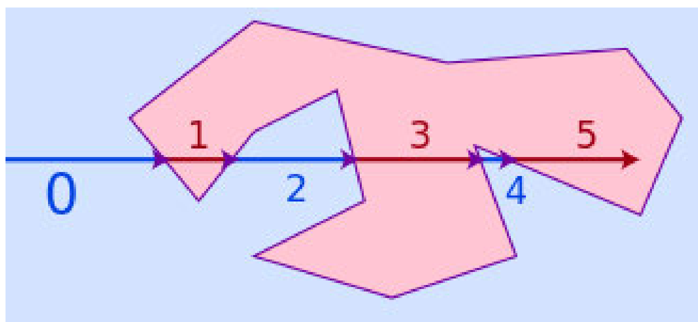


Figure 1.3: The number of intersections for a ray passing from outside the polygon to any point determines its location: If the number is odd, the point is inside the polygon; if even, the point is outside. [16]

- **Point-in-Polygon Test**: One of the most common applications of the Ray Casting Algorithm is determining whether a point is inside a polygon. To perform this test, a ray is cast from the test point in a specific direction (e.g., to the right) and checked for intersections with the edges of the polygon. The number of intersections is counted. If the number of intersections is even, the point is considered outside the polygon; if it's odd, the point is inside.
- **Ray Direction**: The choice of ray direction can affect the results. For point-in-polygon tests, it's common to cast rays horizontally (e.g., to the right) or vertically (e.g., upward). However, any direction can be chosen depending on the problem's requirements.
- **Applications**: Ray casting has various applications beyond point-in-polygon tests, including line-of-sight calculations, shadow rendering in computer graphics, and collision detection in video games.
- **Limitations**: While the Ray Casting Algorithm is efficient, it may not work correctly for polygons with self-intersections or for more complex geometric shapes. In such cases, more advanced algorithms like the Ray-Segment Intersection Algorithm or the Weiler-Atherton Clipping Algorithm may be used.

In general, the Ray Casting Algorithm is a versatile technique used in various computational geometry and computer graphics. It efficiently determines relationships between points and polygons, benefiting numerous applications in image processing and computer vision.

2 Methodology

2.1 Object Detection

Object Detection [17] is a fundamental computer vision task that involves identifying and locating objects within an image or video frame. Unlike image classification, which assigns a single label to an entire image, object detection provides detailed information about the objects present, including their classes and precise positions in the image.

The goal of object detection is to draw bounding boxes around each object of interest and label them with their corresponding class or category. This enables computers to understand and interpret visual information. It is regularly used in autonomous vehicles, surveillance, robotics, and medical imaging.

Object detection techniques commonly rely on deep learning models. It partitions the image into a grid and, within each grid cell, predicts bounding boxes, class probabilities, and objectness scores, resulting in precise object detection.

2.2 Model Training

2.2.1 Data Preparation

We collect images from the GeoFlow II experiment's c20_95 folder to create a diverse and representative dataset containing the double-eye structure for detection. It is advisable to ensure the dataset encompasses a broad spectrum of relevant scenarios and variations.

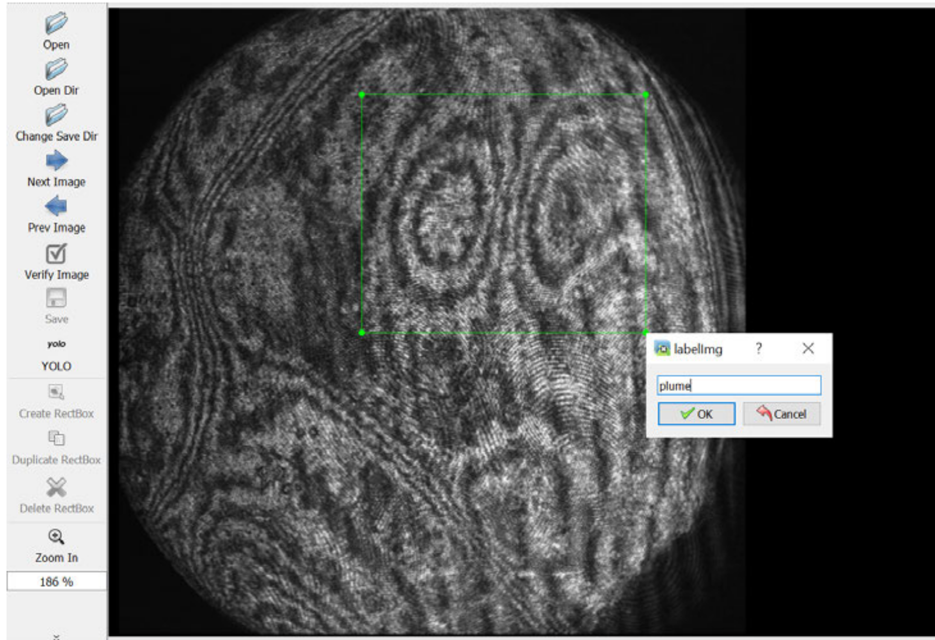
2.2.1.1 Annotation

This thesis employs LabelImg [18], an open-source graphical image annotation tool tailored for the task of outlining bounding boxes around objects exhibiting double-eye structures in images, and labeling them as "plume" in the context of this study. LabelImg simplifies the annotation process by letting users draw these bounding boxes interactively.

The annotations generated using LabelImg are saved in the YOLO format, where each image is associated with a corresponding ".txt" file. Each line in this file represents an object in the image, containing the object's class identifier and the bounding box's normalized coordinates relative to the image dimensions. The YOLO format follows this structure: `<class id>` `<center x>` `<center y>` `<width>` `<height>` , with coordinates normalized from 0 to 1. This standardized format ensures compatibility with YOLO-based object detection models and simplifies the image processing pipeline's following stages.

2.2.1.2 Data Splitting

In this step, we divide the dataset into training set (70%) and validation set (30%). The training set is used to train the model, while the validation set helps fine-tune hyperparameters and monitor training progress.




```
0   0.488281   0.31875   0.304688   0.333333
```

Figure 2.1: An example of using LabelImg tool to annotate the double-eye structure in YOLO format made of image `OPS_1165695908721` and how the lable is saved in a '.txt' file.

2.2.2 Model Configuration

2.2.2.1 Yolov5 Model Selection

In the context of this thesis, we employ a YOLOv5 detection model to identify double-eye structures in GeoFlow II experiment images. Four variants of the YOLOv5 object detection model were designed to cater to different needs in terms of model size, speed, and accuracy. Here's a brief overview of each variant:

- **YOLOv5s (Small)**: This is the smallest and fastest variant of YOLOv5. It has fewer layers and parameters compared to other variants, making it suitable for resource-constrained environments. YOLOv5s is designed for tasks where speed is crucial.
- **YOLOv5m (Medium)**: YOLOv5m strikes a balance between model size and performance. It is a mid-sized variant that offers good accuracy while maintaining reasonable inference speed. This variant is often a good choice for general-purpose object detection tasks
- **YOLOv5l (Large)**: YOLOv5l is a larger variant designed to achieve higher accuracy. It has more layers and parameters compared to smaller variants, making it suitable for tasks where accuracy is the top priority. YOLOv5l may have slightly slower inference times compared to smaller variants.
- **YOLOv5x (Extra Large)**: YOLOv5x is the largest and most powerful variant in the YOLOv5 family. It has a significantly larger number of layers and parameters, making it capable of achieving state-of-the-art accuracy on various object detection benchmarks. This variant is ideal for tasks where the highest level of accuracy is required, even if it comes at the cost of slower inference times.
- **YOLO Nano**: YOLO Nano is an ultra-lightweight variant designed for extremely resource-constrained environments, such as edge devices and IoT devices. It sacrifices some accuracy for the sake of speed and minimal model size.

2.2.3 Environment Setup

2.2.3.1 Data Management with GitHub

To optimize our dataset management during the GeoFlow II experiment, we create a dedicated GitHub repository and commit the dataset to it.

GitHub significantly aids our data management for this thesis. It acts as a secure backup, reducing the risk of data loss, and offers convenient access from anywhere with an internet connection, making it ideal for remote work and collaboration. Additionally, GitHub's integration with tools like Google Colab simplifies our model training and evaluation processes.

2.2.3.2 Computing Power of Google Colab Resources

Google Colab is a cloud-based platform that provides access to Jupyter notebook environments, with the added benefit of GPU (Graphics Processing Unit) acceleration. We utilize Google Colab's GPU for its substantial computational power, which significantly accelerates both YOLOv5 training and detection processes. This reduction in computation time greatly enhances our experimentation efficiency and speeds up object detection.

2.2.4 Training Process

We begin training the YOLOv5 model on our annotated dataset within the Google Colab notebook as our chosen training environment.

Throughout the training process, the model learns to predict bounding boxes and class probabilities for objects in the images. This process involves optimizing the model's parameters based on the annotated data to make accurate predictions.

Simultaneously, we employ a validation dataset to assess the model's generalization performance and to detect overfitting, which happens when the model excels on the training data but falters on unseen data.

Meanwhile, YOLOv5 generates loss curves and computes mAP (mean average precision) scores. These help us understand how well the model is learning and how accurately it can detect objects.

Guided by the validation results, we fine-tune the model and make necessary adjustments to hyper-parameters to improve its overall performance.

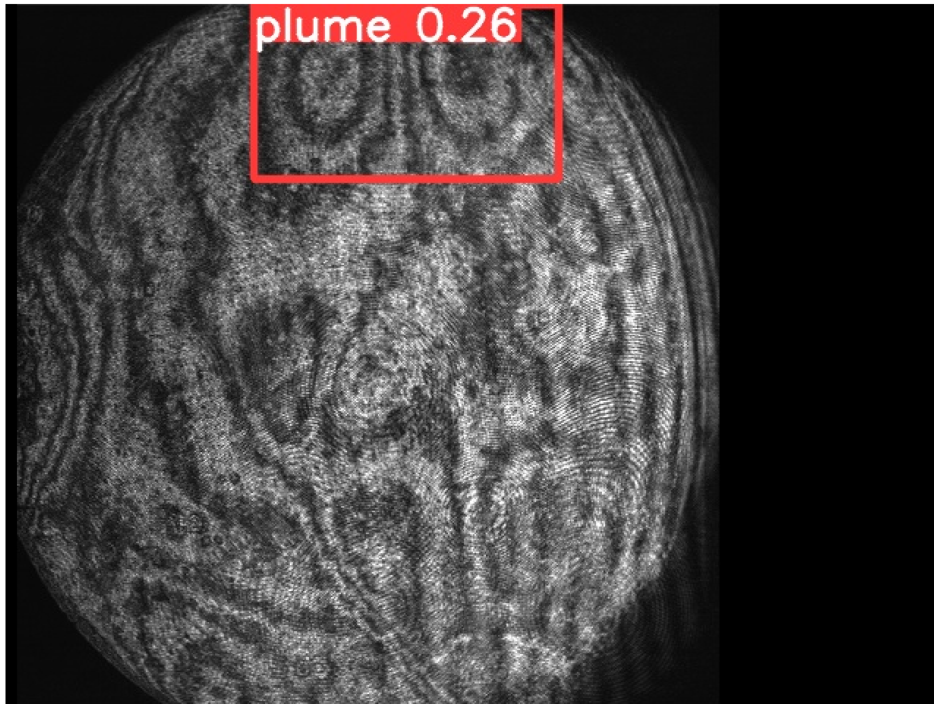
Upon completing the training process, the trained weights of the YOLOv5 model are typically saved. These saved weights are valuable resources and can be applied in different ways, including some of the following:

- **Inference and Deployment**: Trained weights enable the model to make detections on new data, as they contain its learned knowledge.
- **Transfer Learning**: These weights are valuable for transfer learning, where a pre-trained model is fine-tuned on a smaller dataset, saving time and resources.
- **Continual Improvement**: Keeping trained weights allows iterative model improvement over time, starting from an informed state.
- **Reproducibility**: Storing trained weights ensures research can be easily reproduced and validated, fostering collaboration in the scientific community.

2.2.5 Model Deployment

Once the model meets the desired performance criteria, we are ready to make detection on new unseen images. The final output is a list of detected objects, each represented by a bounding box with coordinates (x, y) and dimensions (width, height), a class label, and a confidence score.

We review the detected images and regularly retrain or fine-tune the model with new data to adapt to changing conditions and ensure accuracy.




```
0 0.426562 0.123958 0.325 0.247917
```

Figure 2.2: An example of a Yolov5 detected image `OPS_1165682608795` at confidence score at 0.25 and how the detected label is saved in '.txt' file.

2.3 Image Processing

2.3.1 Images Sorting and Flow Visualization

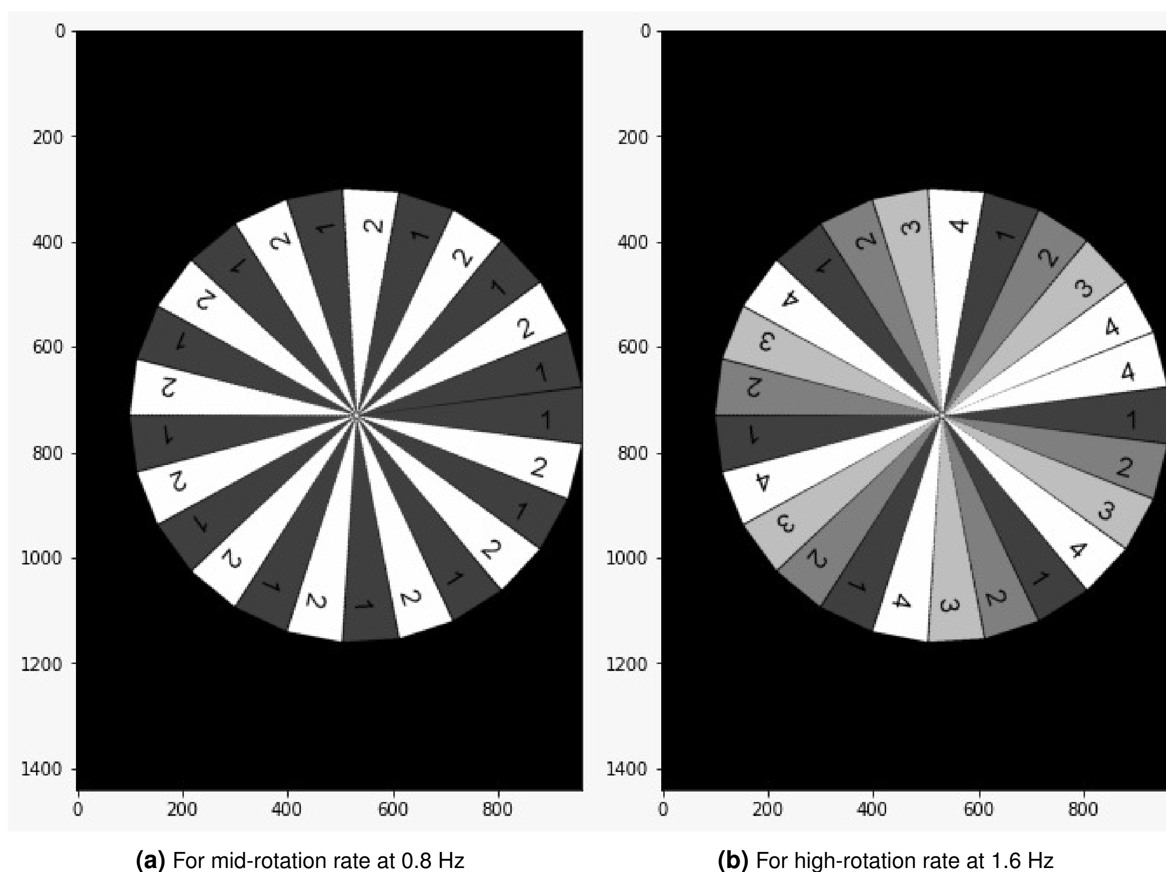
In this thesis, we focus on analyzing images with dimensions of 640 $\times$ 480 pixels obtained from GeoFlow version IIc. Short-term experiment runs captured by a 10 Hz frame rate camera. This frame rate allows us to precisely track turbulent convective flows, resulting in a total of 1250 images captured for a full rotation at 0.008 Hz.

In experiments with a mid-rotation rate of 0.8 Hz, we capture a sequence of 25 images spanning two complete turns. For experiments with a high-rotation rate of 1.6 Hz, we capture a sequence of 25 images spanning four complete turns. In both cases, these sequences of 25 consecutive images allow us to visualize the flow field across the entire northern hemisphere, with each image representing an angular span of $360^{\circ} \div 25 = 14.4^{\circ}$. Subsequently, these image sequences are alternately merged based on either two or four turns. In this thesis, the mid-rotation rate experiments include folders c20_95 and c20_69, while the high-rotation rate experiments comprise folder c21_95.

To create a comprehensive global map from these 25 consecutive images, each labeled from 1 to 25 with time stamps increasing in 100 ms intervals, we use a specific sorting method. This sorting pattern involves combining segments ('patches') of 14.4°into a full circular map, following a predetermined order:

- For mid-rotation rates: 13, 1, 14, 2, ..., 24, 12, 25.
- For high-rotation rates: 19, 13, 7, 1, ... 24, 18, 12, 16, 25.

Each 'patch' created using this approach resembles an isosceles triangle spanning 14.4°, with one corner located at the North Pole. The height of these triangles aligns with the sphere's longitude, as shown in the accompanying image. By systematically assembling these 25 patches in a specified pattern, we achieve a comprehensive visualization of the entire northern hemisphere. This systematic approach enables a thorough analysis of the double-eye structures within the GeoFlow II experiment.



(a) For mid-rotation rate at 0.8 Hz

(b) For high-rotation rate at 1.6 Hz

Figure 2.3: A simplified presentation of merging alternately 25 'patches' in two turns 2.3a and in four turns 2.3b

2.3.2 Rotation (Affine Transformation)

In Python, we performed a masking operation on a specified triangular region within the image. This operation preserved the selected area while setting the pixel values of the remaining portion to zero, resulting in the creation of a "masked" image. Here's a breakdown of the process:

1. For each pixel position (x, y) in the image, we used the path.contains_point(x, y) method from the Matplotlib library's matplotlib.path module. This method employs the Ray Casting Algorithm to determine whether each pixel is inside the defined triangular region.
   - The Ray Casting Algorithm, as used here, traces a ray from a given point and counts its intersections with the edges of the triangle.

- ◦ The point-in-polygon method assesses whether a pixel is inside the triangle, returning a True value for points within the triangle (indicating they should be retained) and a False value for points outside the triangle, aiding in the creation of a binary mask matrix.

2. Using this point-in-triangle function, we created a binary matrix that matches the dimensions of the original image. This binary matrix serves as a mask, marking pixels that should be retained.
3. With both the binary mask matrix and the original image matrix in hand, we performed an element-wise multiplication operation between these matrices.
   This operation retained the original image's pixel values wherever the corresponding mask element was set to 1, while it set the pixel values to 0 where the mask element equaled 0.
4. The result of this element-wise multiplication procedure is a new image. This newly generated image preserves the selected triangular region while masking and setting to zero the remaining portions.

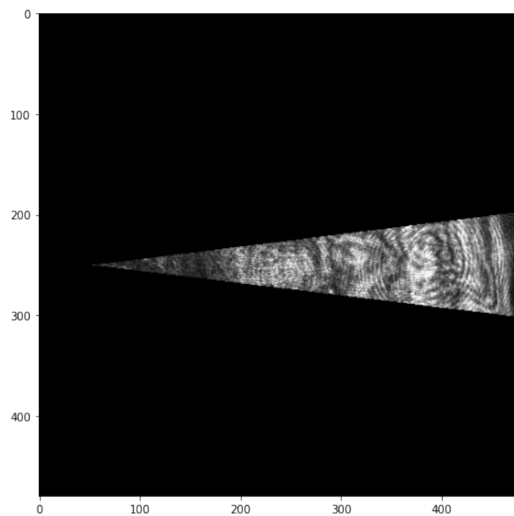


Figure 2.4: An example of a 'triangle-masked' image in the original size 480×480 pixel.

2.3.2.1 Rotation Center

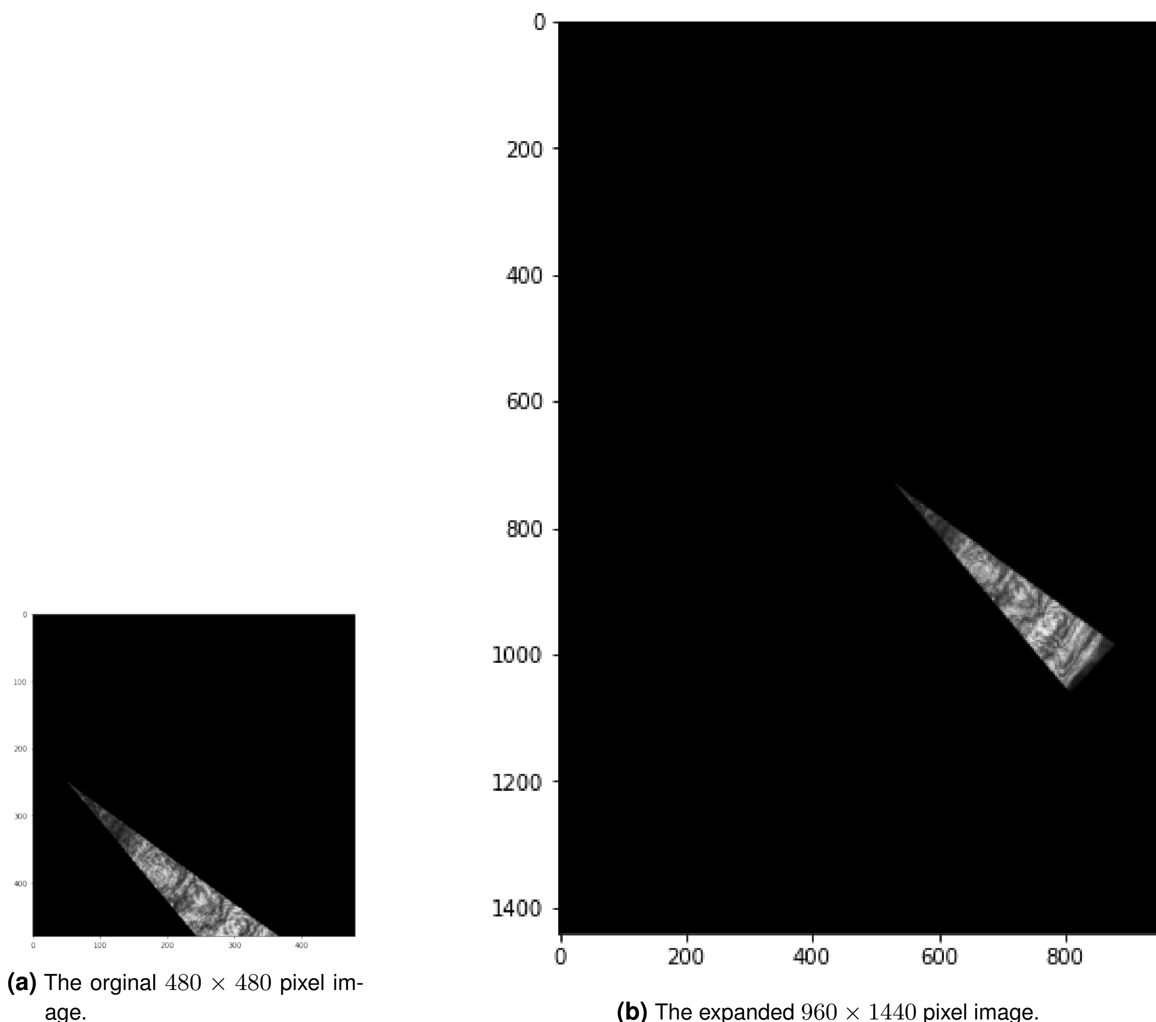
A designated center point is established at coordinates N(53, 250) within the original 480×480 image matrix. This specified center point corresponds to the geographic North Pole of the sphere within the context of the GeoFlow II experiment.

2.3.2.2 Image Expansion

To account for the rotation, we expanded the image matrix by adding additional rows and columns to the top, left, and bottom as shown in Fig.2.5b. This modification resulted in a new rotated image with dimensions of (960, 1440) pixels, whereas the original dimensions were (480 $\times$ 480). This augmentation ensures a full presentation of the triangular 'patch' during the rotation process.

2.3.2.3 Anticlockwise Rotation

The masked image is subsequently rotated anticlockwise around the designated center point. Each image is rotated by an angle by an angle determined by 'n' multiples of 14.4°as shown in Fig.2.5a. The specific 'n' value varies depending on the image's position within the sequence of 25 images.



(a) The orginal 480×480 pixel image.

(b) The expanded 960×1440 pixel image.

Figure 2.5: An example of expanding an image to ensure a full presentation of the triangular 'patch' during image rotation.

2.3.3 Sequential Transformation

The rotation process is performed iteratively for all 25 images in the sequence. Each image is subjected to the masking, rotation, and expansion procedures, employing a unique 'n' value determined from its sequential order within the sequence.

2.3.4 Assemble

2.3.4.1 Combining Individual Images

After applying the masking, rotation, and expansion procedures to each image individually as described earlier, we then proceed to combine these 25 processed images into a cohesive group. This collection of 25 images collectively represents the sphere's perspectives from various angles during its two rotations in Fig.2.6 or four rotations in Fig.2.7.

2.3.4.2 Addition Operation

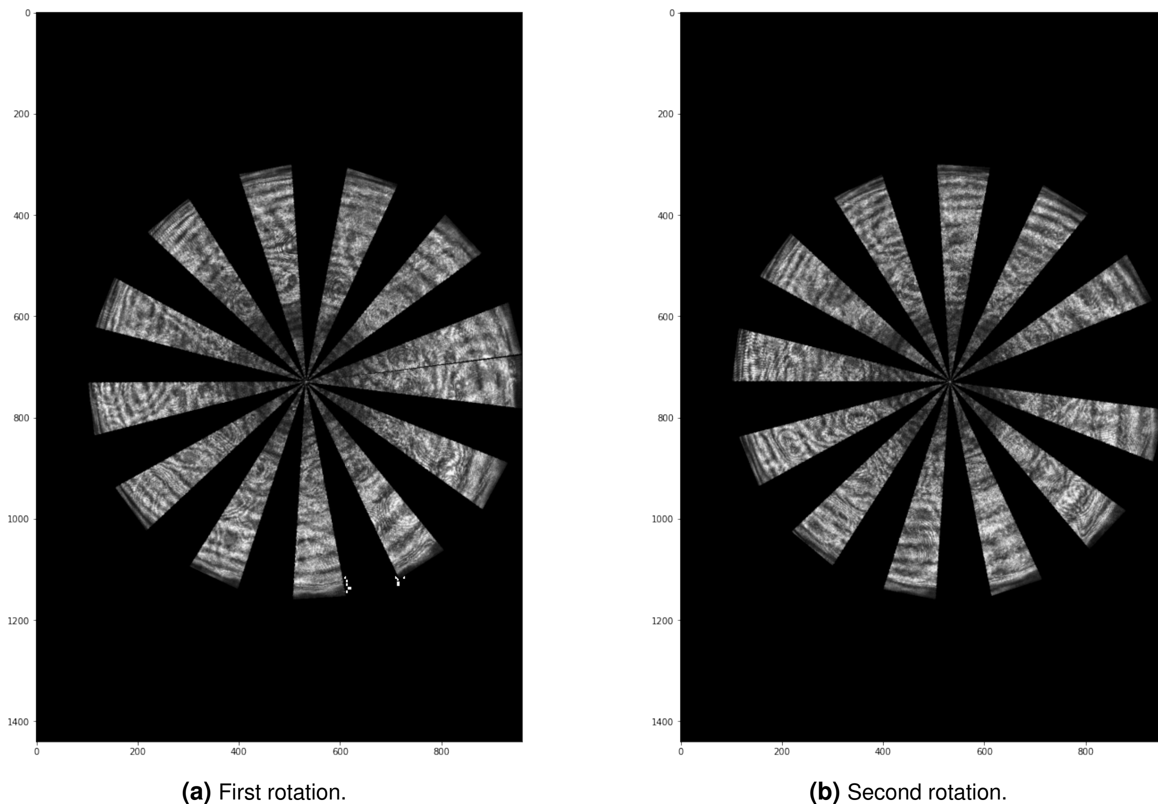
The combination of these new output images is straightforward, assisted by a simple addition operation. This operation is effective because each image exclusively contains non-zero values within the triangular region of interest. As a result, when these images are added together, they form a coherent representation with non-zero values aligning to create a circular-like shape as shown in Fig.2.8.

By following these rotation and assembly procedures, we efficiently process the original images, allowing us to capture the sphere's behavior over two or four rotations and generate a single, comprehensive circular-like image. This unified image contains all the essential information required for subsequent analysis.

It's important to note that this process relies on a critical assumption: that the images were captured at precisely 100ms spaced intervals. This temporal consistency enables us to observe the northern hemisphere over time, providing valuable insights into the dynamics and characteristics of the double-eye structures we seek to analyze. By implementing these operations as described, we have effectively prepared our dataset for further analysis and object detection tasks.

2.4 Object Detection on Post-processed Images

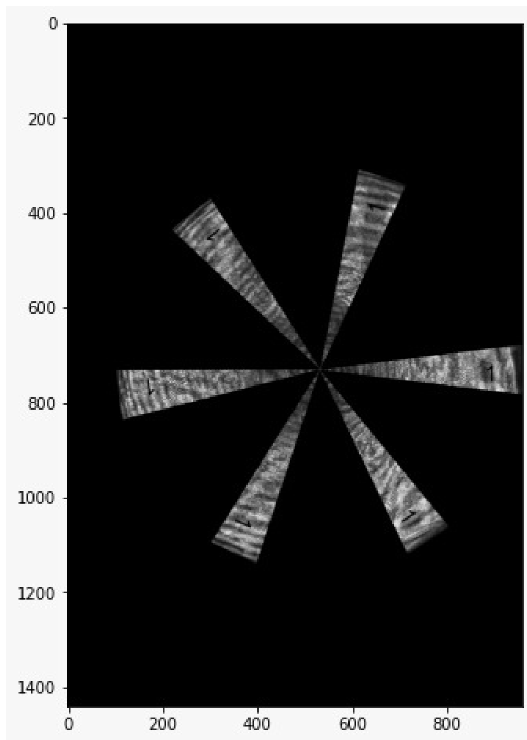
The post-processed images form a complete circle, each one representing the northern hemisphere at different time intervals. During our observations, we've identified double-eye structures at various moments within this circular representation. Our goal is to use the previously trained Yolov5 model to detect these double-eye structures within the post-processed data.



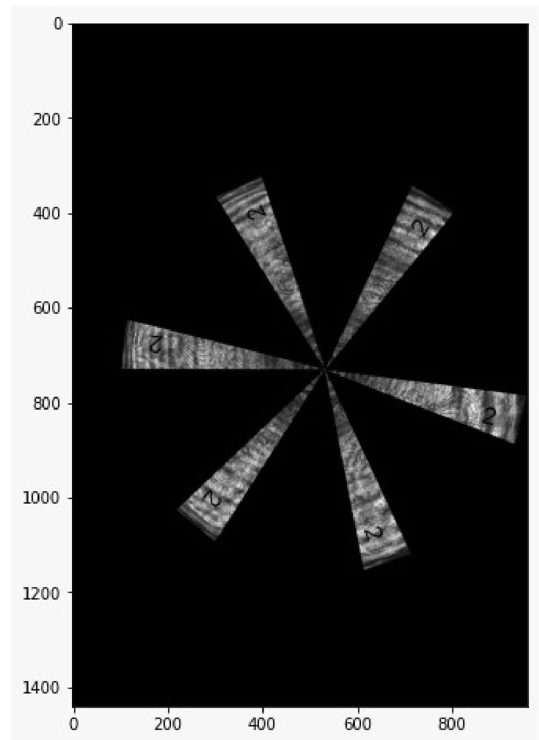
(a) First rotation.

(b) Second rotation.

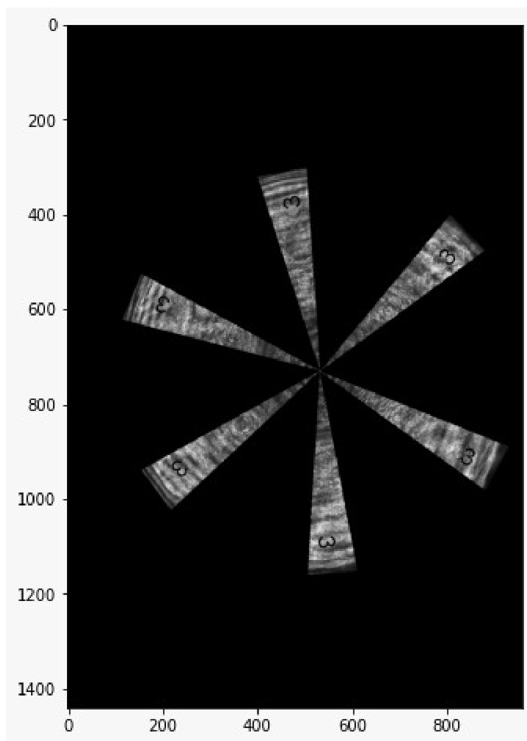
Figure 2.6: Example of a two-rotation sequence.



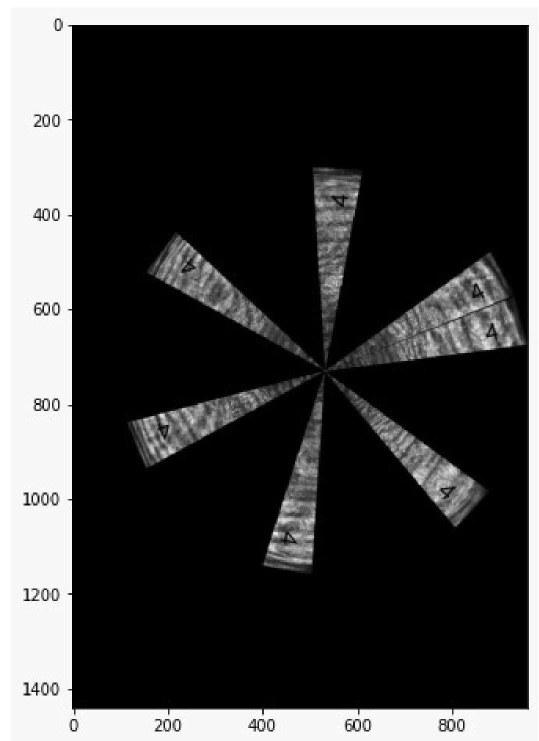
(a) First rotation



(b) Second rotation

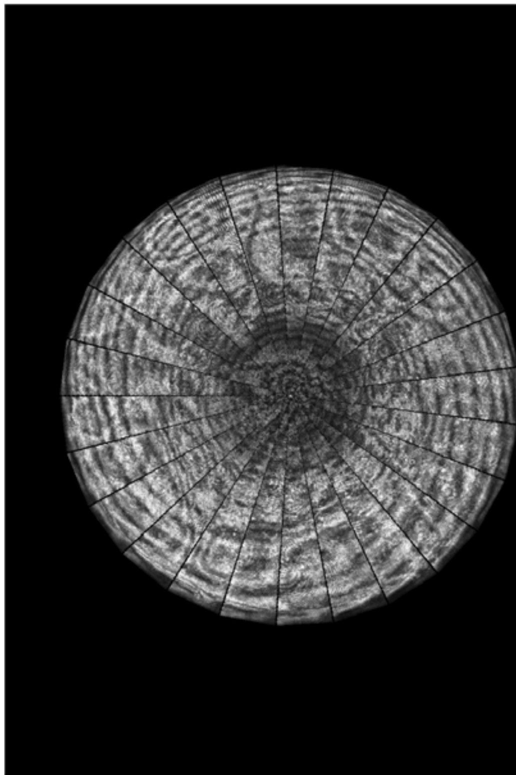


(c) Third rotation

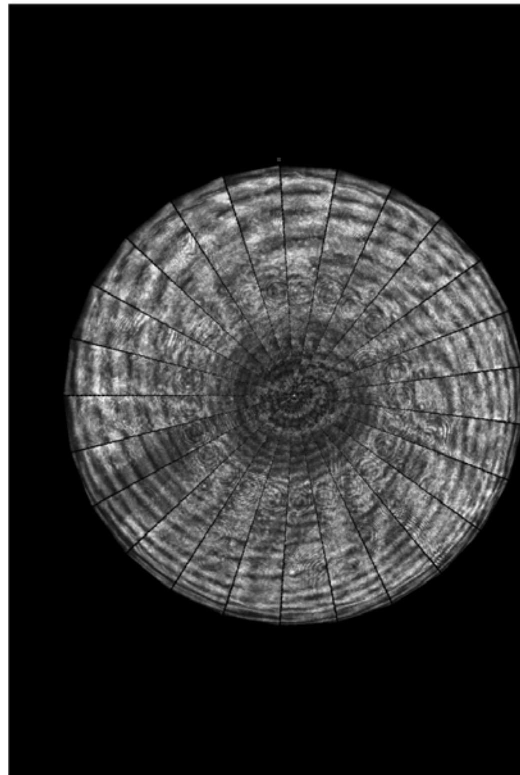


(d) Fourth rotation

Figure 2.7: Example of a four-rotation sequence



(a) Two-rotation sequence.



(b) Four-rotation sequence.

Figure 2.8: A full presentation of merging alternately 25 triangular 'patches' in two turns 2.8a and four turns 2.8b.

3 Results

3.1 Trained Yolov5 Model

3.1.1 Initial Training

In this thesis, we trained two variants of the YOLOv5 model, YOLOv5m and YOLOv5x, using the dataset from the c20_95 folder, which consists of 1851 images. Our dataset was divided into a 70% training dataset and a 30% validation dataset. During the training sessions for both models, we conducted 50 epochs of training with a batch size of 16.

Variant	P	R	mAP50	mAP50-95	Weight Size	Time
Yolov5m	0.538	0.46	0.455	0.257	41,182 KB	35m36s
Yolov5x	0.0142	0.552	0.0316	0.00961	169,007 KB	1h16m21s

Table 3.1: Performance metrics of two Yolov5 vaiants in training.

The results in Table 3.1 show that the YOLOv5m model outperforms the YOLOv5x model in terms of precision (0.538 vs. 0.0142) and mAP50 (0.455 vs. 0.0316), indicating that YOLOv5m achieves more accurate detections and localization. However, the YOLOv5x model exhibits a slightly higher recall (0.552) compared to YOLOv5m (0.46), suggesting that it captures a larger portion of the relevant objects. Despite the differences in performance, both models have relatively low mAP50-95 scores, indicating room for improvement in precise object localization. Additionally, the YOLOv5x model comes with a significantly larger model size and longer training time, which should be considered when choosing between the two models for specific applications.

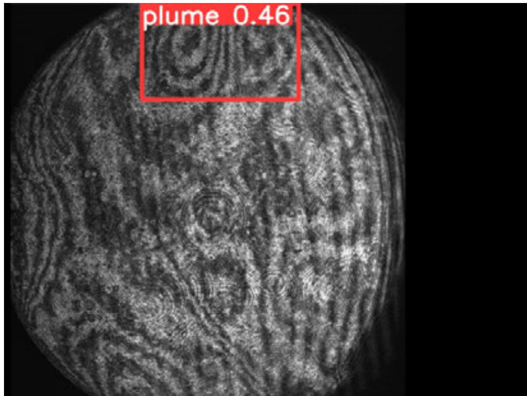
We used these trained models to detect objects in 1879 new, unseen images from the c20_69 folder. Results are shown in Table 3.2. Here's what we found:

Variant	Images	Detection	Time	Confidence Score
Yolov5m	1879	172	53s	0.25
Yolov5x	1879	1880	1m35s	0.25

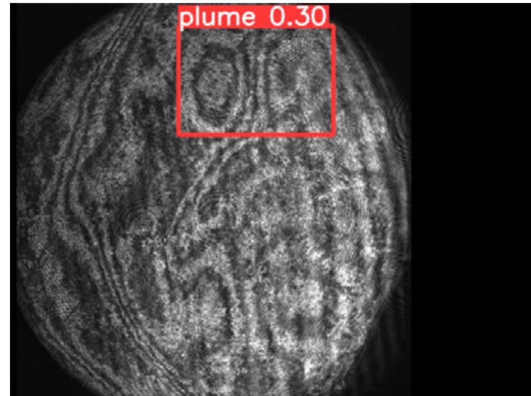
Table 3.2: Two YOLOv5 variants detection result by confidence score.

- **Yolov5m** made 172 detections in 53 seconds, upon visual inspection, we see that most of these detections were accurate, as shown in 3.1.
- **Yolov5x** on the other hand, Yolov5x made 1880 detections in 1 minute 35 seconds. However, upon visual inspection, we see that this model struggled to learn and accurately detect the double-eye structures in the images from the c20_69 folder, as shown in 3.2.

These results highlight the differences in performance between the two models when faced with the specific task of detecting double-eye structures in this dataset.

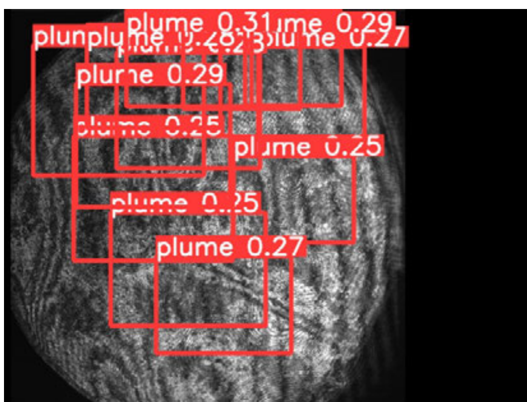


(a) Detected `OPS_1165682627610`.



(b) Detected `OPS_1165682627813`.

Figure 3.1: Two examples of the initial Yolov5m object detection.



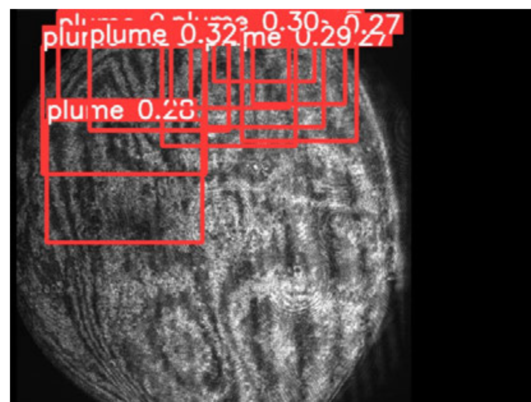


Figure 3.2: Two examples of Yolov5x object detection.

3.1.2 Optimal Training

The training results 3.1.1 indicate that YOLOv5x outperforms YOLOv5m in terms of detection accuracy, showcasing the potential for further enhancements in recall and mAP scores. Consequently, we extended the training of YOLOv5m for an additional 200 epochs. By epoch 150, Google Colab indicated that the model had ceased to show further improvement, suggesting it had reached its optimal training performance. The results, presented in Table 3.4, also support this conclusion.

Variant	P	R	mAP50	mAP50-95	Weight Size	Time
Yolov5m	0.903	0.888	0.862	0.749	41,182 KB	1h35m1s

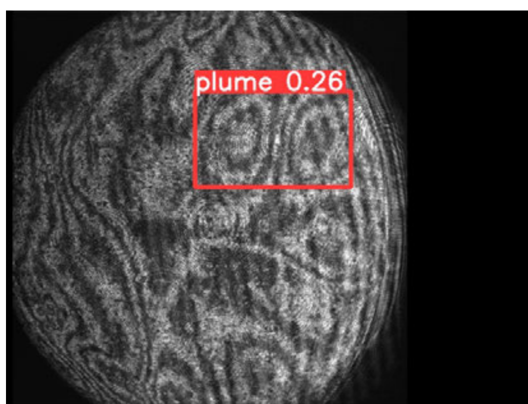
Table 3.3: Performance metrics of Yolov5m model in training.

We employed the optimally trained YOLOv5m model to detect objects in new, unseen images in two folders. Here is our observations:

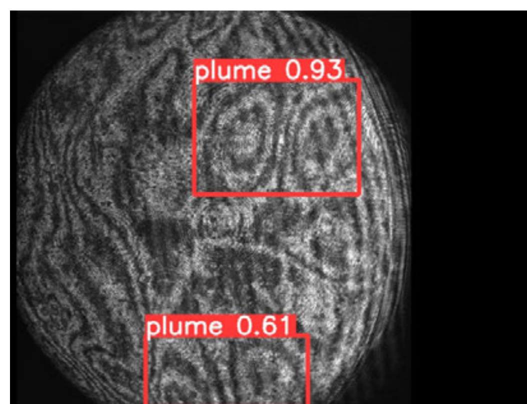
- Model YOLOv5m, trained on mid-rotation rate experiment folder c20_95, detecting mid-rotation rate experiment folder c20_69. It's worth noting that within experiments of the same rotation rate, the double-eye structures are relatively similar. As shown in Fig.3.3, the 150-epoch-trained model recognizes the same double-eye structures as the 50-epoch-trained model but with a higher confidence score. Additionally, the 150-epoch-trained model can detect another double-eye structure that the 50-epoch-trained model cannot.
- Model YOLOv5m, trained on mid-rotation rate experiment folder c20_95, detecting high-rotation rate experiment folder c21_95. There are significant differences in object structures between mid-rotation rate and high-rotation rate experiments. As shown in Fig.3.4, the optimally trained YOLOv5m model makes impressive detections of distorted double-eye structures.

Folder	Images	Detection	Time	Confidence Score
c20_69	1880	718	8m3s	0.25
c21_95	1880	262	11m4s	0.25

Table 3.4: Performance metrics of Yolov5m model in training.

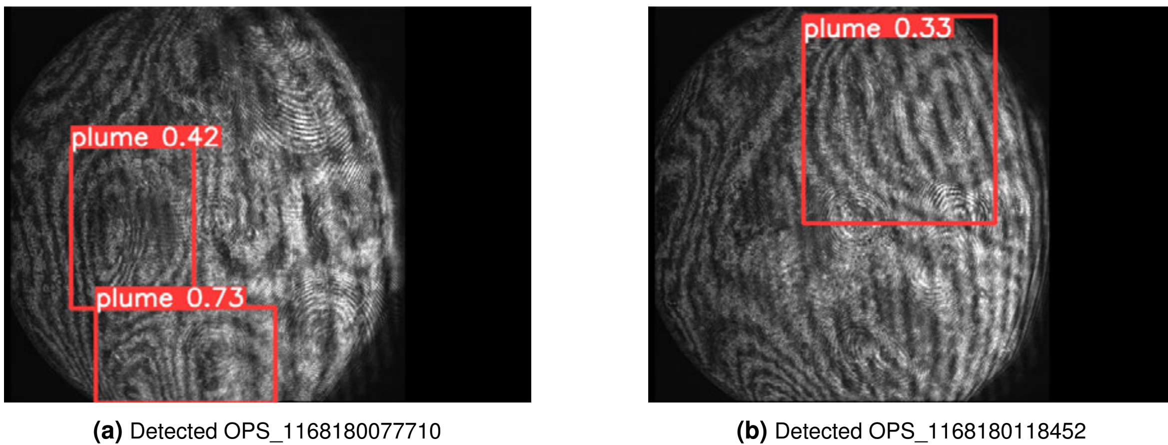


(a) 50-epoch trained Yolov5m



(b) 150-epoch trained Yolov5m

Figure 3.3: Two examples comparing the initial Yolov5m to detect folder c20_69.



(a) Detected OPS_1168180077710

(b) Detected OPS_1168180118452

Figure 3.4: Two examples of the optimally trained Yolov5m to detect folder c21_95.

3.2 Image Processing

In this thesis, we have successfully built a systematic procedure that transforms a sequence of images that initially captured incomplete views of a rotating sphere into a full northern hemisphere representation. While most assembled images didn't align perfectly to reveal proper structures, as shown in Fig.3.5, a few iterative assemblies showcase the evolving double-eye structure, as shown in Fig.3.6. This suggests the potential of our methodology but calls for further enhancements.

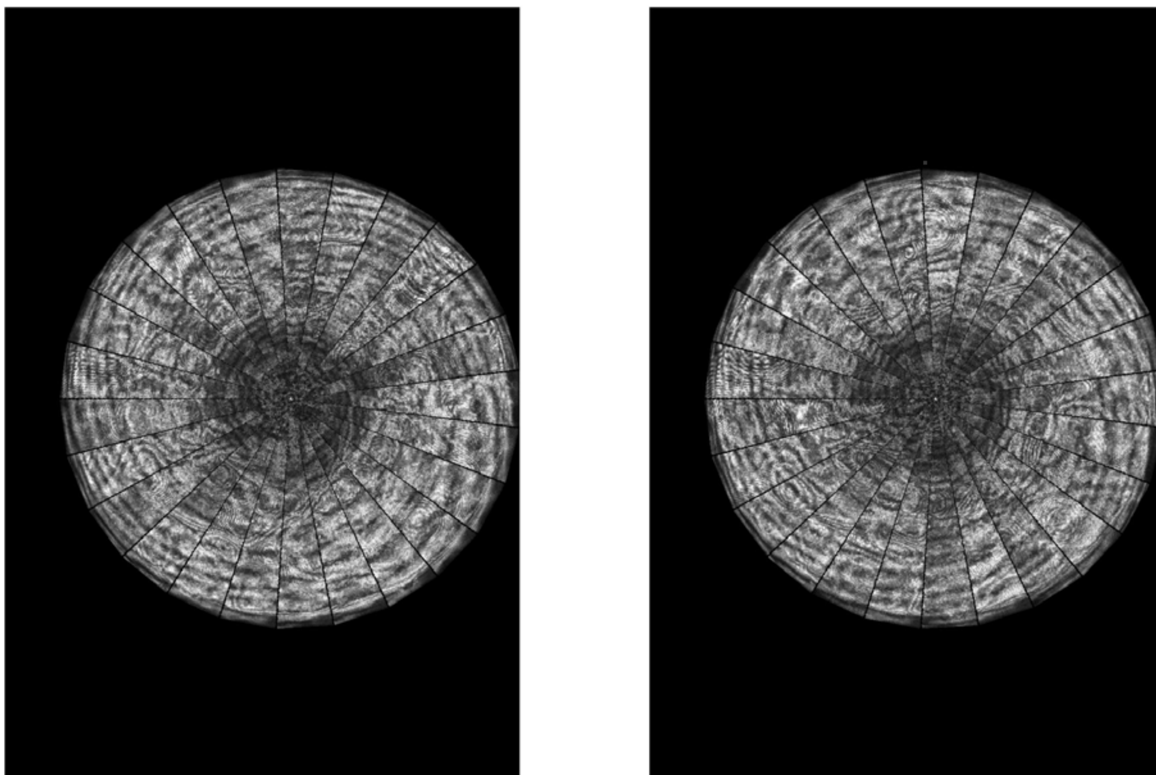
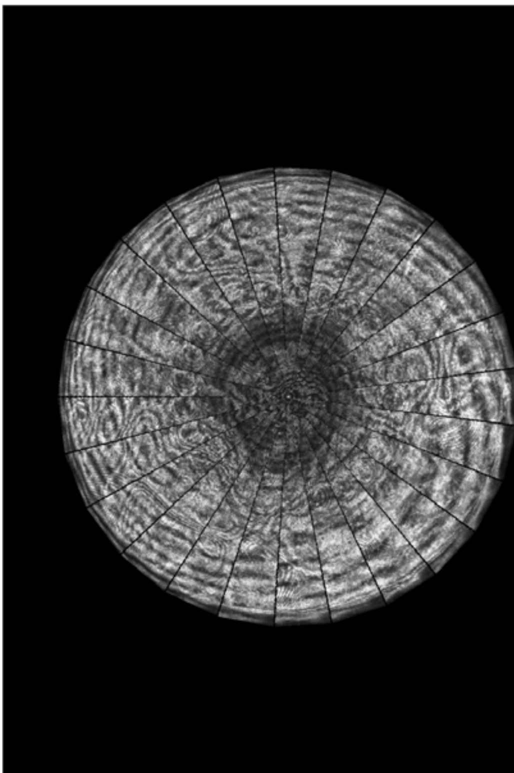
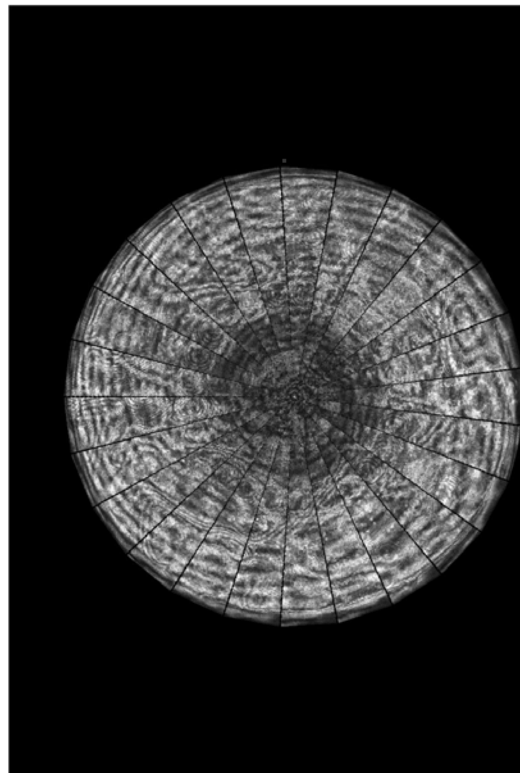


Figure 3.5: Two unsuccessful presentations of the aligned 'patches' that do not form a double-eye structure.



(a) Caption for the first image



(b) Caption for the second image

Figure 3.6: Two well-made presentations of the aligned 'patches' that do form a double-eye structure.

3.3 Object Detection on Post-processed Images

We employ the newly trained YOLOv5m model to detect objects in the post-processed images. The model was previously trained on the pre-processed c20_95 dataset, and we apply it to detect objects in the post-processed c20_95 dataset, which consists of 75 assembled images representing the northern hemisphere.

At Yolov5 default confidence score of 0.25, the model made 6 detections in 11 seconds, 3 of which were accurate upon visual inspection, as shown in Fig.3.7.

Adjusting the confidence score to 0.1, the model made 9 detections in 12 seconds, 6 of which were accurate upon visual examination, as shown in Fig.3.8.

Despite extensive training, the model did not produce convincing detection results. However, these results suggest the potential for automated detection of the double-eye structures and visualization of the northern hemisphere within the GeoFlow II experiment. The limited results also indicate the need for further improvement in our methodology. The model faced challenges in automatically identifying double-eye structures in the post-processed images, which will be discussed in the following section.

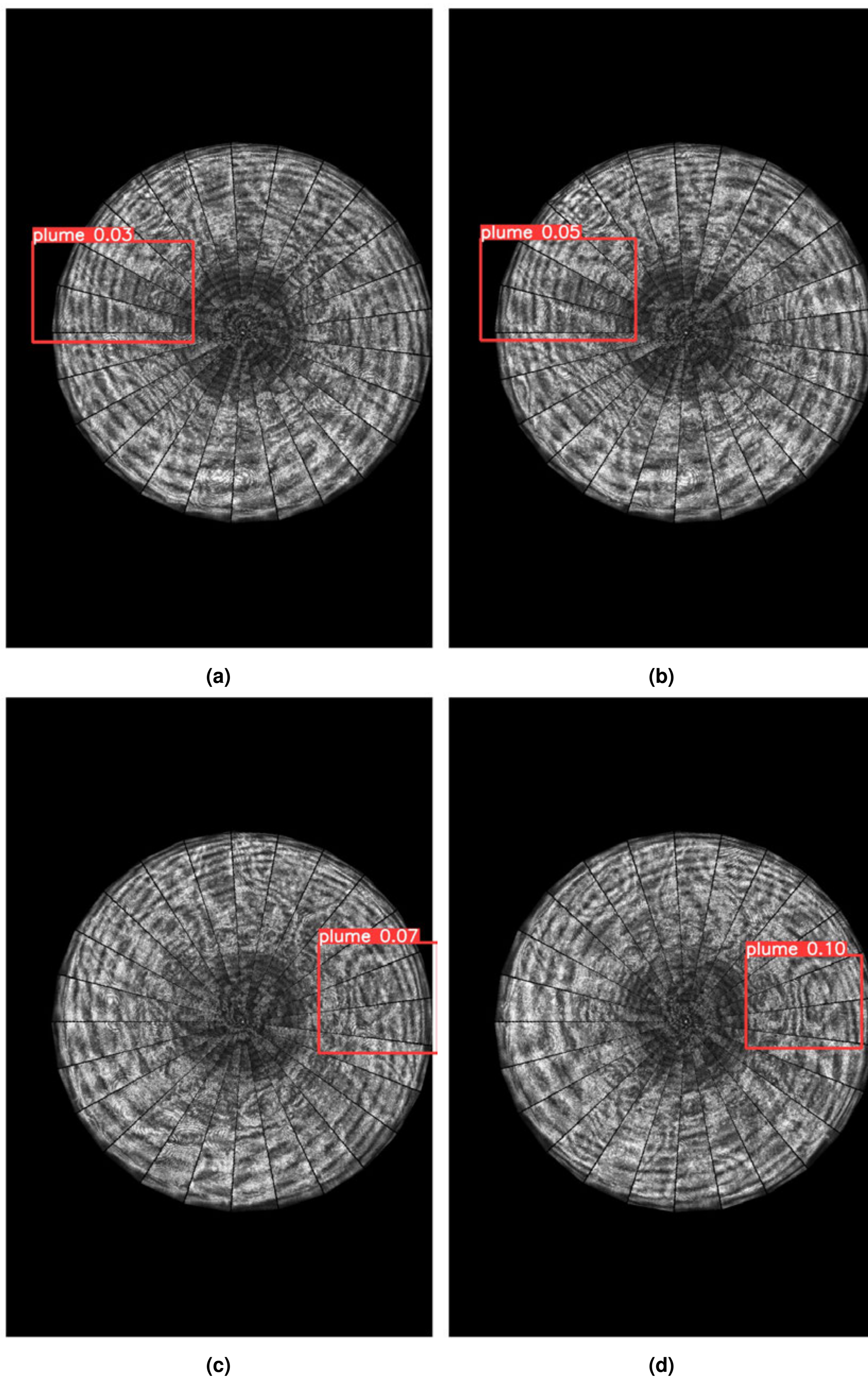


Figure 3.7: Two examples of false detection 3.7a, 3.7b and two examples of correct detection 3.7c, 3.7d made by the optimally trained Yolov5m model at confidence score 0.25.

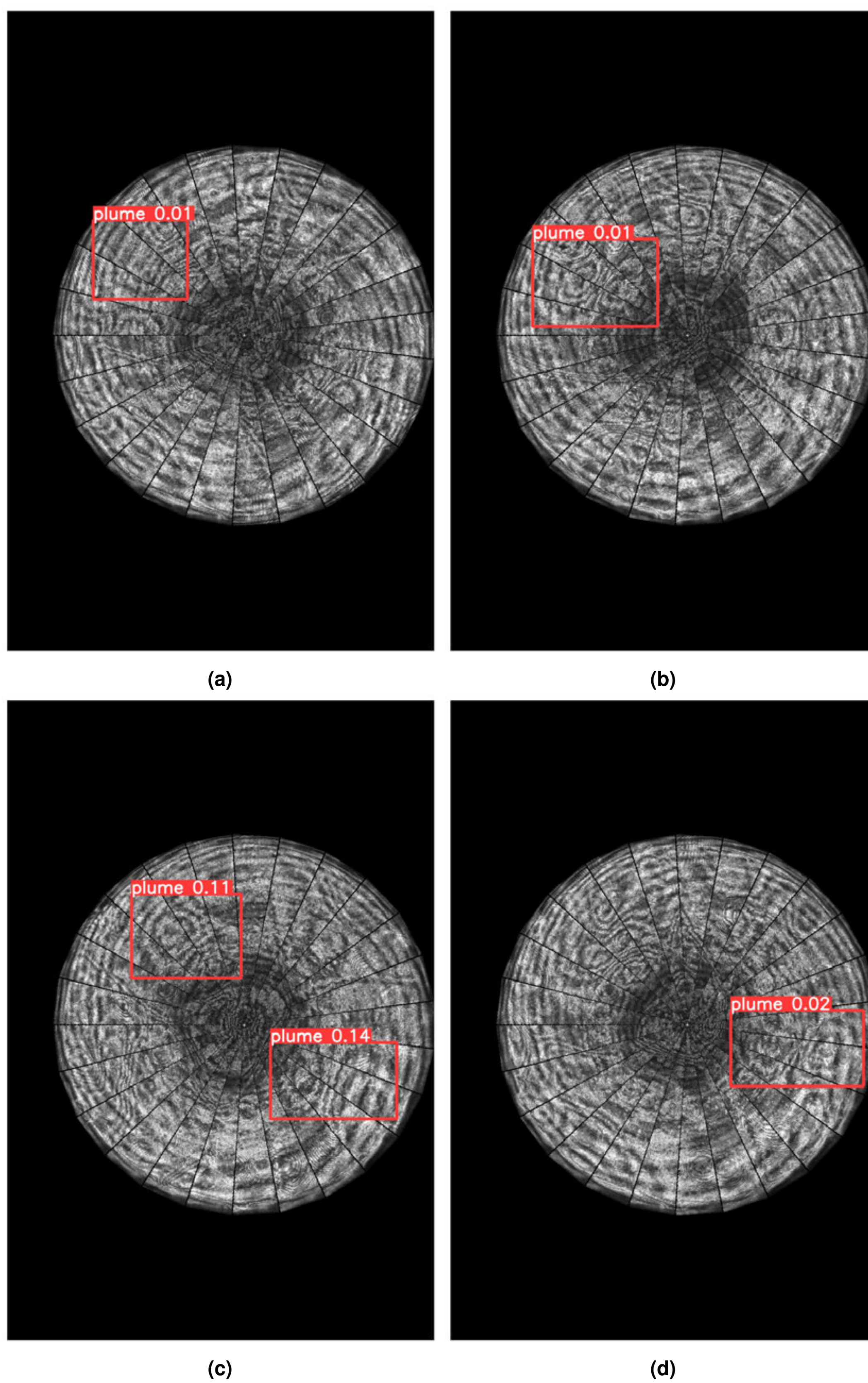


Figure 3.8: Two examples of false detection 3.8a, 3.8b and two examples of correct detection 3.8c, 3.8d made by the optimally trained Yolov5m model at confidence score 0.1.

4 Discussion

4.1 Triangle Cut Position

The placement of the triangle cut in image processing did not yield optimal results in detecting double-eye structures within the GeoFlow II experiment. Initially, the decision to set the triangle cut at 7.2°to - 7.2°, as Fig.2.4, was influenced by observations made from the internship project [19]. However, we now acknowledge that this approach is not always correct for experiments with mid to high-rotation rates. From the result in 3.1.1, we see that the model repeatedly detected objects at the upper right corner. This suggests that we should change the position of the triangle cut. The proposed angle is shown in Fig.4.1. This conclusion is made by observing the detected pre-processed images from folder c20_95 in mid-rotation rate experiments, it might not be applicable for high-rotation rate experiments, nor it is applicable for other cases of the same mid-rotation rate experiments like c20_69, c20_56, etc.

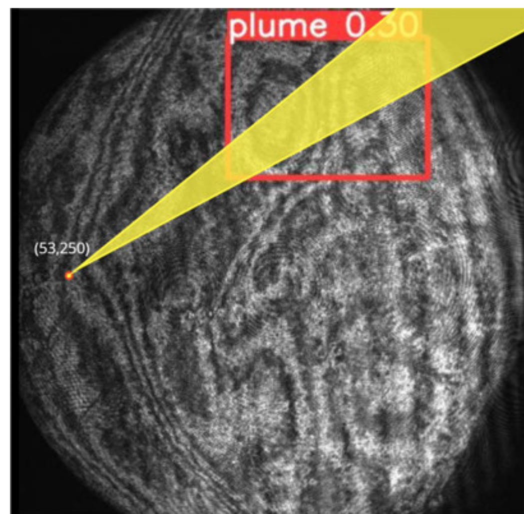


Figure 4.1: Proposed triangle cut position

4.2 Temporal Discrepancy Assumption

One of the fundamental assumptions in our methodology was that each image was captured at precisely spaced time intervals of 100ms. However, in practice, we discovered the images did not strictly adhere to this assumption, occasionally exhibiting a displacement of ±2ms. This discrepancy could have potentially misaligned the annulus during image processing.

4.3 Angular Span Limitation

Many structures exceeded an angular span of 14.4°, necessitating more than one angle to capture one complete structure. This factor introduced a significant risk of incomplete structure formation.

4.4 Pre-processed Image Optimization Insight

An important observation from our detection model is that the model's performance reached its best when applied to post-processed images. Notably, the model's performance has no improvements after 150 epochs of training. This implies that the detection model is currently operating at its optimal capacity, and further enhancements should primarily focus on improving the quality of the pre-detected dataset.

4.5 Potential Overfitting in YOLOv5 Object Detection

The detections obtained from the post-processed images in Section 3.3 were achieved by optimally training the YOLOv5 model on the pre-processed c20_95 dataset and subsequently applying it to detect objects in the post-processed c20_95 images. However, this approach may raise concerns about potential overfitting, as the model could have essentially memorized the objects in the pre-processed dataset and merely identified them in the post-processed images. This situation might undermine the intended purpose of YOLOv5, which is designed to learn from data and make predictions on new, unseen images.

4.6 Proposal 1: Marking The Detected Objects Prior To Image Processing.

In the context of the GeoFlow II experiment, our findings suggest that the YOLOv5 model performs optimally when conducting object detection on pre-processed images. We recommend adopting this approach for further experiments. The workflow involves initial object detection on pre-processed images, followed by the marking of detected objects. This marking, achieved by using the bounding box centers inherited from the YOLOv5 model, provides a suitable representation of the objects of interest, as shown in Fig.4.2. Subsequently, we proceed with image processing, resulting in a sequence of assembled images with object markings. By analyzing the displacement of these markings, we can gain valuable insights into the object dynamics within the experiment.

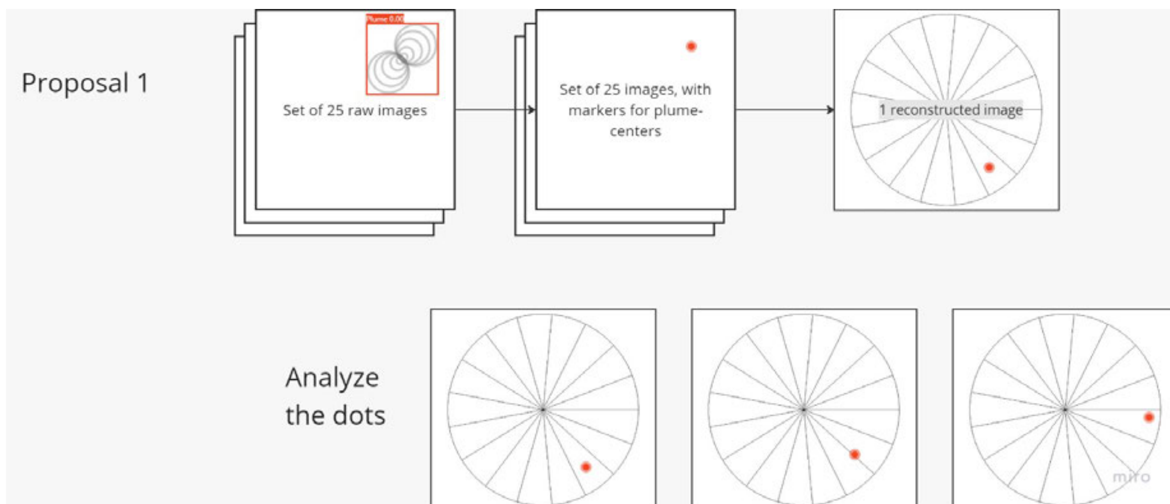


Figure 4.2: Proposal 1: Marking the detected objects prior to image processing.

4.7 Proposal 2: Angle-independent Structure Detection

We recommend conducting object detection on pre-processed images. Instead of collecting a sequence of images over two, or four, rotations, we collect images spaced by two, or four, turns at the same rotating position. This new collection contains images taken at consistent angles, allowing us to perform object detection and compare the displacement of objects every two, or four, rotations, as shown in Fig.4.3 and Fig.4.4.

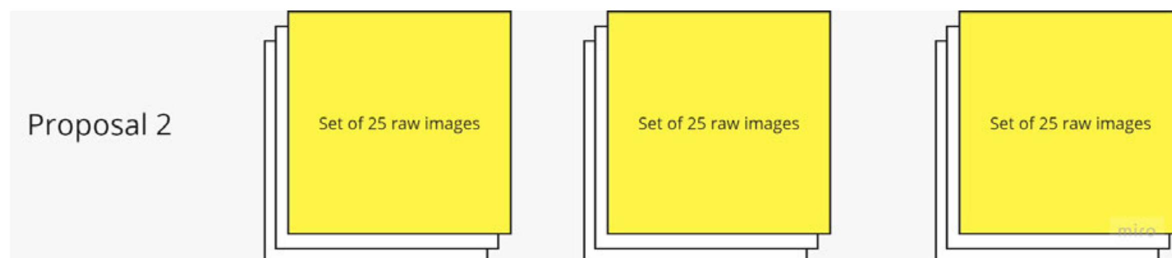
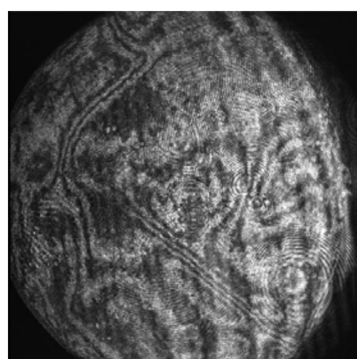
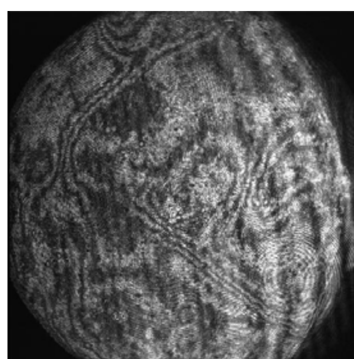


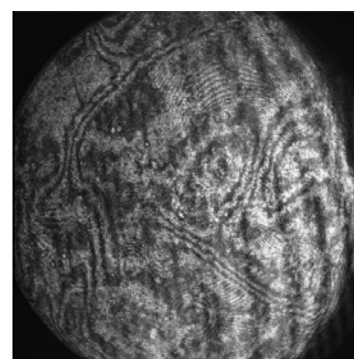
Figure 4.3: Proposedly collecting the first images to each 25-image-sequence.



(a) First image of rotation 5



(b) First image of rotation 7



(c) First image of rotation 9

Figure 4.4: Example collection of three images at the same position, separated by two rotations, in folder c20_95.

5 Annex

Sorting & Labeling of Mi2R folder

This script takes an original folder of more than 1,800 images and splits it into a structure of sub-folders of 25 images which corresponds with 2 rotations. The first 12 images capture the first rotation, and the next 13 images capture the second rotation. Then these 25 images are reordered and labeled in a way that is ready for the next assembly step, i.e. Image-1 will be put next to Image-2, and so on.

```
import os
import shutil
from PIL import Image

path= r"path" # path to folder c20_95

j = 1
a = 0
b = 0
c = 0

def create_directory(directory):
    if not os.path.exists(directory):
        os.makedirs(directory)

for file_name in os.listdir(path):
    if file_name.endswith(".jpg"):
        c += 1
        if c <= 12:
            a += 2
            i = a
        else:
            b += 1
            i = b
            b += 1

        output_directory = os.path.join(path,str(j))
        create_directory(output_directory)

        k = f"{i:02d}"

        img = Image.open(os.path.join(path,file_name))
        # Crop to 480x480 pixels with an offset of 5 pixels
        cropped_img = img.crop((5, 0, 485, 480 ))
        cropped_img.save(os.path.join(output_directory, f"{k}.jpg"))
```

```
        if i == 25:
            j += 1
            a = 0
            b = 0
            c = 0
```

Sorting & Labeling of Hi4R folder

This script does almost the same process as the above section, except that 25 images correspond to 4 rotations. Approximately every 6 images now capture a rotation.

```
import os
from PIL import Image

path=r'path' # path to folder c21

j = 1
a = 0
b = 0
c = 0
d = 0
e = 0
i = 0
directory=os.path.join(path, str(j))
os.makedirs(directory, exist_ok=True)

for file_name in sorted(os.listdir(path)):
    if file_name.endswith('.jpg'):
        c += 1
        if c <= 6:
            a += 4
            i = a
        if (c>= 7) & (c<=12):
            b += 3
            i = b
            b +=1
        if (c >= 13) & (c<=18):
            d += 2
            i = d
            d +=2
        if c >= 19:
            e += 1
            i = e
            e +=3

        k = f'{i:02d}_'+str(os.path.splitext(os.path.basename(file_name))[0])+'.jpg'

        img = Image.open(os.path.join(path, file_name))
```

```
        cropped_img = img.crop((5, 5, 485,485))
        cropped_img.save(os.path.join(directory, k))

        if i == 25:
            j += 1
            directory=os.path.join(path, str(j))
            os.makedirs(directory, exist_ok=True)
            a = 0
            b = 0
            c = 0
            d = 0
            e = 0
            i = 0
```

Image Processing

This script takes the images, then cuts, rotates, and assembles them together. This process is repeated for all sub-folders which hold 25 images each.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.path import Path
from PIL import Image as im
from PIL import Image, ImageDraw, ImageFont,ImageOps
import os
from os import listdir

fnt = ImageFont.truetype("Pillow/Tests/fonts/arial.ttf", 30)

### Triangle matrix for multiplying element-wise
def create_triangle_matrix():
    # Create a 480x480 matrix filled with zeros
    matrix = np.zeros((480, 480), dtype=int)

    # Define the vertices of the triangle
    vertices = np.array([[53, 250], [480, 197], [480, 302]])

    # Create a Path object to represent the triangle
    path = Path(vertices)

    # Iterate through each point in the matrix
    for y in range(480):
        for x in range(480):
            # Check if the point lies inside the triangle
            if path.contains_point((x, y)):
                matrix[y, x] = 1

    return matrix
```

```
def add_margin(pil_img, top, right, bottom, left, color):
    width, height = pil_img.size
    new_width = width + right + left
    new_height = height + top + bottom
    result = im.new(pil_img.mode, (new_width, new_height), color)
    result.paste(pil_img, (left, top))
    return result

original_folder_path = r"path"
save_path = r"path"

# Create a triangle matrix
triangle_matrix = create_triangle_matrix()

# MAIN
# The order by which images will be assembled anti-clockwise
order=[i for i in range(1,26)]

mydict ={order[i]:i for i in range(25)}
list_images = [None]*25

# Loop through all 76 sub-folders of origin folder.
for folder in [i for i in range(1,77)]:
    # get the path/directory
    folder_dir = os.path.join(original_folder_path,str(folder))
    for images in os.listdir(folder_dir): # Enforce the correct format
        # check if the image ends with jpg
        if (images.endswith(".jpg")):
            label=int(images[:2])
            # creating an image object
            im1 = Image.open(folder_dir+r"\\"+images)
            im2 = im1.resize((480, 480))
            # applying grayscale method
            im3 = ImageOps.grayscale(im2)
            list_images[mydict[label]] = im3

    # Now 25 images are sorted into the order of assembly already.
    Images_before = list_images
    Images_after = []

    padtop = 480
    padleft=480
    padbottom=480

    # Loop through 25 images
    for i in range(25):
```

```
        # take new image
        image =  Images_before[i]

        # cut
        array_cut = np.multiply(np.asarray(image), triangle_matrix)
        image_cut = im.fromarray(array_cut)

        # add padding and text
        image_pad = add_margin(image_cut, padtop, 0, padbottom, padleft, 0)
        #I1 = ImageDraw.Draw(image_pad)
        #I1.text((400+480, 250+480), str(order[i]), font=fnt)

        # rotate by multiples of 14.4 degree anti-clockwise
        resultimage_pad = image_pad.rotate(-14.4*i,
                                        center=(53+padleft, 250+padtop))

        # append to list
        Images_after.append(np.asarray(resultimage_pad))
        # Now each of the 25 images has non-zero values only in the
        # corresponding triangle area.

    # Combine 25 images by addition operation into 1 final image
    padarray_plus = sum(Images_after)
    FinalImage =  im.fromarray(padarray_plus)

    # Plot 1 final image and save
    plt.figure(figsize=(20,15))
    plt.imshow(FinalImage)
    plt.axis('off')
    plt.savefig(os.path.join(save_path,str(folder)+".jpg"),bbox_inches='tight')
    plt.close('all')

    # Next folder if applicable
```

Custom training YOLOv5 model

This script loads the YOLOv5 model from the source repository, trains the model with our custom dataset, and runs detection on another arbitrary dataset. YOLOv5 model should be run with GPU resources. An affordable method to run experiments on GPU is through a Google Colab Pro subscription for approximately €11/month.

```
import tensorflow as tf

# Check GPU
print(tf.config.list_physical_devices('GPU'))

# Clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5.git
```

```
TRAIN_DATA = r'path'
DETECT_DATA = r'path'

!nvidia-smi

!pip install -r requirements.txt

import os
import shutil

def move_folder_contents(source_folder, destination_folder):
    # Get a list of all items (files and subfolders) in the source folder
    items = os.listdir(source_folder)

    # Move each item to the destination folder
    for item in items:
        source_item_path = os.path.join(source_folder, item)
        destination_item_path = os.path.join(destination_folder, item)

        # use shutil.move to move content
        shutil.move(source_item_path, destination_folder)

    print(f"Moved contents from '{source_folder}' to '{destination_folder}'.")

# Example usage:
source_folder = TRAIN_DATA
destination_folder = '/content/yolov5/data'
move_folder_contents(source_folder, destination_folder)


"""Instructions:
+ From TRAIN_DATA, move folder ('train', 'valid', 'data.yaml', 'best.pt'),
+ to folder '/content/yolov5/data'
"""
# Train
!python train.py --data /content/yolov5/data/data.yaml --epochs 50
--cfg yolov5m.yaml --batch-size 16 #--weights /content/yolov5/data/best.pt

"""From 'yolov5/runs/train/exp/weights/best.pt', download 'best.pt'
and save for later.
 """

# Detect
!python detect.py --weights yolov5/runs/train/exp/weights/best.pt
--source DETECT_DATA --img 480
--data /content/yolov5/data/data.yaml --save-txt --conf 0.25
```

Bibliography

[1] NASA. (2010). International Space Station after undocking of STS-132.jpg [Image]. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:International_Space_Station_after_undocking_of_STS-132.jpg

[2] ESA. (2008). Kugel in der Kugel: das Experiment GEOFLOW [Image]. ESA Multimedia. https://www.esa.int/ESA_Multimedia/Images/2008/03/Kugel_in_der_Kugel_das_Experiment_GEOFLOW

[3] ESA. (2012). Geoflow experiment [Image]. ESA Multimedia. https://www.esa.int/ESA_Multimedia/Images/2012/06/Geoflow_experiment

[4] Zaussinger, F., Egbers, C., Kuhlmann, H., Hollerbach, R. (2021). Modeling and Simulation of Thermo-electro Hydrodynamics. BTU Cottbus - Senftenberg. https://opus4.kobv.de/opus4-btu/frontdoor/index/index/docId/5517

[5] Ultralytics. "YOLOv5: Open-source deep learning framework." GitHub, https://github.com/ultralytics/yolov5.

[6] Redmon, J., Farhadi, A. (2018). YOLOv3: An Incremental Improvement. https://arxiv.org/abs/1804.02767

[7] Sanyam Jain. (2023). Adversarial Attack On Yolov5 For Traffic And Road Sign Detection. https://arxiv.org/abs/2306.06071

[8] Geoffery Agorku, Divine Agbobli, Vuban Chowdhury, Kwadwo Amankwah-Nkyi, Adedolapo Ogungbire, Portia Ankamah Lartey, Armstrong Aboah. (2023). Real-Time Helmet Violation Detection Using YOLOv5 and Ensemble Learning. https://arxiv.org/abs/2304.09246

[9] Arunabha M. Roy, Jayabrata Bhaduri. (2023). A Computer Vision Enabled damage detection model with improved YOLOv5 based on Transformer Prediction Head. https://arxiv.org/abs/2303.04275

[10] Longlong Li and Zhifeng Wang and Tingting Zhang. (2023). Photovoltaic Panel Defect Detection Based on Ghost Convolution with BottleneckCSP and Tiny Target Prediction Head Incorporating YOLOv5. https://arxiv.org/abs/2303.00886

[11] Mihir Durve, Sibilla Orsini, Adriano Tiribocchi, Andrea Montessori, Jean-Michel Tucny, Marco Lauricella, Andrea Camposeo, Dario Pisignano, Sauro Succi. (2023). Benchmarking YOLOv5 and YOLOv7 models with DeepSORT for droplet tracking applications. https://arxiv.org/abs/2301.08189

[12] Sudipto Paul and Dr. Md Taimur Ahad and Md. Mahedi Hasan. (2022). Brain Cancer Segmentation Using YOLOv5 Deep Neural Network. https://arxiv.org/abs/2212.13599

[13] Aleksandr N. Grekov and Yurii E. Shishkin and Sergei S. Peliushenko and Aleksandr S. Mavrin. (2022). Application of the YOLOv5 Model for the Detection of Microobjects in the Marine Environment. https://arxiv.org/abs/2211.15218

[14] Bochkovskiy, A., Wang, C. Y., Liao, H. Y. M. (2020). Yolov4: Optimal speed and accuracy of object detection. https://arxiv.org/abs/2004.10934

[15] Wikipedia contributors. (2023, September 15). Ray Casting. In Wikipedia. https://en.wikipedia.org/wiki/Ray_casting

[16] "Point in Polygon." Wikipedia. Wikimedia Foundation. Retrieved [insert date of access], https://en.wikipedia.org/wiki/Point_in_polygo

[17] Lin, Tsung-Yi et al. "COCO: Common Objects in Context." (2014). https://arxiv.org/abs/1405.0312

[18] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg

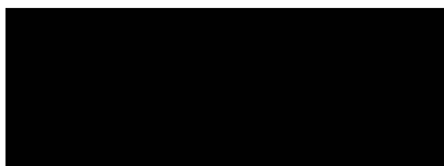
[19] Nguyen. Object detection using deep learning model in the GeoFlow experiment (2022). Hochschule Mittweida.

Statutory Declaration in Lieu of an Oath

I – Ngoc Uyen Nguyen – do hereby declare in lieu of an oath that I have composed the presented work independently on my own and without any other resources than the ones given.

All thoughts taken directly or indirectly from external sources are correctly acknowledged.

This work has neither been previously submitted to another authority nor has it been published yet.



Mittweida, 23. September 2023

Location, Date

Ngoc Uyen Nguyen