Falling Droplet Localization

BSc Thesis

M.E. Kaya
T.A. Haas
N. El Khatibi

Delft University of Technology

TU Delft

Falling Droplet Localization

by

M.E. Kaya
T.A. Haas
N. El Khatibi

Supervisors: Dr. Padmakumar R. Rao
Dr. Sandra K. Raveendran
Tejus Kusur

TU Delft

Abstract & Preface

This report presents the development of a real-time object detection algorithm as part of a larger vision-based tracking system for falling objects. The complete system consists of a Raspberry Pi 5 equipped with a High Quality Pi Camera mounted on a Pimoroni PIM183 pan–tilt actuator. Its purpose is to detect, track, and visually follow a fast-moving target during free fall. The tracking system is divided into two main components: a tracking subsystem and a motion control subsystem.

This report focuses on the visual detection and tracking subsystem, which is responsible for processing a live video stream to determine the vertical position of the falling object in real time. The core algorithm extracts positional data and forwards it to the motion control subsystem, which is implemented by a parallel project group. For validation and testing, a colored cloth ball is used as a proxy for a water droplet, as it offers greater visibility and more consistent shape characteristics.

A key objective of the system is to capture high-quality frames of the object in mid-air, enabling further image-based research by third parties. These frames, centered and stabilized via real-time feedback, are intended for use in subsequent analysis of droplet dynamics, behavior modeling, and related applications in industrial or agricultural research.

This report discusses the design choices and trade-offs involved in building a robust and responsive detection algorithm suitable for integration in a time-critical closed-loop control system.

M.E. Kaya
T.A. Haas
N. El Khatibi
Delft, July 2025

Contents

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
HSV	Hue Saturation Value
KCF	Kernelized Correlation Filter
MMSE	Minimum Mean Squared Error
MOSSE	Minimum Output Sum of Squared Errors
NMN	Non-Maximum Suppression
ORB	Oriented Fast and Rotated Brief

Symbols

Symbol	Definition	Unit
FPS	Frames per second	1/s
px	Pixel	-

1 Introduction

1.1. Background and Motivation

Many industries rely more on computer-systems with embedded object-tracking functionalities. It is important in various fields in these industries to understand and be able to simulate the object which is tracked. In high-speed inkjet-printing machines for example the droplet formation needs to be fully understood in order to prevent misprints [1], in fertilizing spray machines a proper droplet size and density need to be ensured for effective crop treatment [2].
This report addresses the problem of tracking individual falling droplets in real time. Accurately determining a droplet's position during free fall is a critical challenge in systems where precise droplet behavior affects overall performance. However, due to the complexity and variability of real droplet dynamics, such as transparency, size variation, and background interference, a direct study can be difficult to carry out.
Therefore, a proxy problem is introduced: a solid spherical cloth ball is used to represent a falling droplet. The proxy object is easier to control and observe, especially against a plain white background, allowing the development and evaluation of the object-tracking system under simplified but representative conditions. This approach serves as a proof of principle for the underlying method and allows for clearer validation of the system's tracking and image-capturing performance.
The objective of the proposed method is to vertically track the real-time location of the proxy object on a 2D image plane and capture high-quality images as it falls. The real-time position of the falling proxy can then be fed forward to a tilting motion control system, which aims to center the object in mid-air and thereby maximize the number of frames captured during its descent.

1.2. Problem Definition

The central challenge addressed in this work is reliably capturing high-quality images of a free-falling object during its descent. This task becomes particularly demanding when tracking droplets, as their transparency, high reflectance, and lack of color contrast against typical backgrounds make them exceptionally difficult to image clearly [3]. Conventional imaging systems struggle to maintain consistent focus and framing on such objects throughout their rapid descent.

To enable controlled development of the imaging system while preserving the essential tracking challenges, we employ a proxy object approach using a solid spherical cloth ball. This simplification maintains the key dynamics of a falling object while providing more consistent visual properties that facilitate algorithm development and system validation. The proxy object serves three critical purposes:

1. Providing a consistent visual target that maintains the essential tracking challenges of a falling droplet
2. Enabling quantitative evaluation of image capture quality under controlled conditions
3. Allowing isolation of tracking performance from the additional complexities of droplet physics

1.3. System Overview

The complete tracking system is designed to maintain a vertically falling object centered within the camera's field of view during free fall. This system is partitioned into two interdependent subsystems whose specifications derive directly from the overall system requirements defined in Chapter 2.

- **Visual Detection Subsystem**
  To meet the functional requirement of tracking at 50 frames per second, this subsystem captures images using a Raspberry Pi HQ Camera at 50 FPS and greater, providing timely and accurate 2D vertical position measurements of the object. The detection algorithm is tailored to the known object characteristics (a 5 cm diameter colored ball) and operates reliably under the controlled laboratory lighting. Accuracy requirements for object localization ensure the motion control subsystem receives precise data for maintaining the object's position.
- **Motion Control Subsystem**
  The control subsystem uses the positional data to compute actuator commands within the mechanical limits of the Pimoroni PIM183 pan–tilt unit, ensuring the camera's tilt remains within -90° to $+90^\circ$. The control loop is designed to complete within 20 ms ($\sim$ 50 FPS) mandated by the real-time operation requirement, thus ensuring responsiveness and stable tracking. All control computations run exclusively on the Raspberry Pi 5, simplifying hardware dependencies and adhering to system constraints.

The subsystems work in tandem within a closed-loop architecture where the Visual Detection Subsystem continuously supplies the Motion Control Subsystem with positional feedback at 50 FPS. This frame rate matches the mechanical and computational limits of the hardware, balancing detection fidelity and control responsiveness.

The proxy object (a colored cloth ball of approximately 5 cm diameter) was selected to enable robust detection with known color and size properties, as required by the environment and detection reliability specifications.

The overall system is designed to fulfill the mandatory and trade-off requirements by combining real-time image acquisition, accurate object localization, and fast, bounded actuation to maintain the object centered vertically in the frame despite the high-speed fall.

Figure 1.1 illustrates the high-level data flow between these subsystems, demonstrating how they are connected.

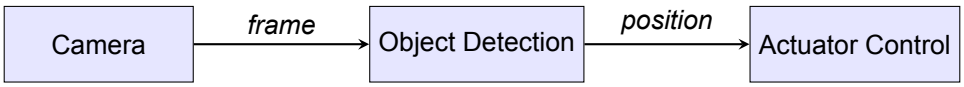


Figure 1.1: High-level system overview showing image acquisition, detection, and control

1.4. State-of-the-Art Methods

Building upon the high-level system overview presented above, the object detector can be implemented using a range of image-processing methods. Over the past decades, both traditional algorithms and AI-based techniques have been developed for object detection and motion tracking, each with distinct strengths and limitations. The following section reviews a selection of these state-of-the-art methods that are relevant for detecting and tracking objects in motion.

Frame Differencing

Frame differencing is a simple and computationally cheap technique for motion detection that functions by comparing consecutive frames in an image sequence. The method calculates the absolute difference between pixel intensities of two successive gray-scaled frames, and significant differences between each pixel indicate motion. Example of such indications for motion are seen in Figure 1.2.
While computationally efficient and suitable for real-time applications, it struggles with detecting objects using a tilting-camera and is sensitive to various changes in the background such as dynamic illumination and shadows.

This method's implementation and performance are further discussed in Chapter 4 and Chapter 5 respectively.

(a) Frame 1
Ball at n=169

(b) Frame 2
Ball at n=170

(c) Processed difference

Figure 1.2: Visualization of the frame differencing algorithm: (a) previous frame, (b) current frame, and (c) processed absolute difference between Frame 1's and Frame 2's respective gray-scaled representations.

Edge Detection

A classic approach to detect objects in images is edge detection. There are a number of well-known and understood edge detection algorithms. As discussed by [4], some popular options are Sobel, Roberts, Canny and Laplacian. These operations detect the change in image intensity, which is then classified as an *edge*. They accomplish this by taking the derivative of the image intensity, where the first derivative or the rate of change of the pixel value uses a gradient operator. The second derivative can also be used instead with a Laplacian operator.
In the gradient-based detection algorithms like Roberts, Canny and Sobel, the gradient of x and y are separately computed by using a kernel or mask. This kernel can be seen as a matrix that then gets convolved with the original image. Combined with an adjustable threshold for the magnitude of the gradient, the edges in an image can be detected. Some starting points for the thresholds for these algorithms are discussed in [5], in which the author advised a threshold of 0.35 to 0.45 for Canny, while for Roberts and Sobel the thresholds 0.1 to 0.45 were best.

MOSSE

The Minimum Output Sum of Squared Error (MOSSE) filter is an algorithm in the family of correlation-based trackers [6]. The core principle of these kind of trackers is to learn a correlation filter which represents the object of interest to track. This filter is then used to find the object in subsequent frames. It is renowned for its exceptional processing speed, often achieving several hundred frames per second. Although MOSSE is extremely fast, its primary limitation is its simplicity. The algorithm in itself also requires manual selection of the object of interest, which is not ideal for object tracking as this calls for a separate algorithm to perform object detection. The OpenCV library provides a direct and highly optimized implementation of this, making it easily accessible for real-time applications.

KCF

Similarly to MOSSE, kernelized correlation filter(KCF) is part of the correlation based trackers. Previous trackers, such as MOSSE use linear filters. KCF addressed the limitation of linear filters. Building directly on the high speed framework established by MOSSE [7]. This led to KCF being viewed as the state of the art tracking for several years. Just like MOSSE the KCF function is included in the widely used OpenCV library. KCF also requires an initial object selection.

Background Subtraction

Background Subtraction is similar to frame differencing, a simple yet effective technique used to detect moving objects in videos with a static background. This method is based on first building a background model, which is a representation of the scene without any moving objects. This model can be as simple as an average of the initial few frames or a more complex statistical representation, such as a Mixture of Gaussians (MoG), which can adapt to gradual changes in lighting [8], [9]. Once this background model is established, each incoming frame is compared to it by subtracting the current frame from the background model pixel-by-pixel. The resulting difference image highlights regions where significant changes have occurred. By applying a threshold to this difference image, a binary foreground mask is generated that segments the moving objects from the static background. While conceptually simple and computationally efficient, this method is highly sensitive to dynamic environments, such as sudden lighting changes or camera movement. The results and performance of this approach is further discussed in Chapter 5.

Deep Learning models

Instead of relying on algorithms a different option would be deep learning models like Convolutional Neural Networks (CNNs). One of the use cases of a CNN is image classification, which is accomplished by feeding this model a labeled images and training it. An example model as described in [10] uses such model with five layer neural networks and 60 million parameters.
More specific to this paper's objective would be something like an edge detection model. In order to train such models, large datasets are required. A classical data set would be something like the method of Arbalez et al, described in [11].
A computionally expensive high performing model would be something like the work described by Zhong et al in their model DETR with 304 million parameters [12]. The other end of the spectrum would be something like Tiny and Efficient Edge Detector (TEED) by Soria X. et al [13] with only 58 thousand parameters, which can be trained in only 30 minutes.

1.5. Report Outline

This report is structured as follows: Chapter 2 provides a review of the Program of Requirements, Chapter 3 describes the overall physical system, Chapter 4 presents the implemented image processing techniques, the results obtained by testing these implementations and their evaluation are discussed in Chapter 5. Finally Chapter 6 concludes this report.

2

Program of Requirements

This chapter specifies the requirements for the complete tracking system, composed of a visual detection subsystem and a motion control subsystem. The aim is to track a falling object using a camera mounted on a tilting platform, ensuring the object remains centered in the image during free fall.

2.1. Target Environment

The system operates in a controlled laboratory setting with consistent lighting and a fixed drop zone. The falling object is a colored ball approximately 5 cm in diameter. The system uses a Raspberry Pi 5, Raspberry Pi HQ Camera, and a Pimoroni PIM183 pan–tilt unit.

2.2. Mandatory Requirements

(Must be fully satisfied; non-compliance means failure.)

1. Functional Requirements
   (a) The complete system must track a vertically falling object using tilt-only camera motion at 50 FPS, consistent with actuator and processing limitations.
   (b) The control loop, comprising frame acquisition, object detection, and actuator command computation, must execute within 20 ms to ensure real-time responsiveness.
   (c) All computations, including image processing and control, must run locally on the Raspberry Pi 5 without external processing support.
   (d) The visual detection subsystem must detect the object's vertical position in the image with precision better than the ratio of the ball's pixel height to the frame height, such that the object may be centered within $\pm 5\%$ of the image height by the motion control system. This ball is known to have a diameter of approximately 5 cm and a fall velocity around 5–7 m/s.
   (e) The detection subsystem must output the object's 2D image coordinates (vertical position) per frame to the motion control subsystem.
2. Non-functional Requirements
   (a) The object detection algorithm must be robust to minor lighting variations and leverage the known color and spherical shape (approximately 5 cm diameter) for reliable segmentation.
   (b) The image resolution and lens parameters must be chosen to ensure the object occupies a sufficient number of pixels (e.g., ≥ 25 pixels in diameter) to permit reliable detection.
   (c) The system must avoid exceeding actuator torque or PWM signal limits to prevent hardware damage.
   (d) The lens of the system should not need to be refocused during the fall of the object.

2.3. Trade-off Requirements

(Optimize where possible; performance may vary.)

1. The system should maintain detection robustness under small lighting variations such as shadows or minor reflections.
2. The actuator response should minimize overshoot and oscillations to ensure stable tracking.
3. Motion blur should be minimized in captured frames to preserve detection accuracy.

3

System Overview

3.1. Overall System Architecture

The overall prototype can be split in two parts. The PIM183 Pan-Tilt actuator that controls the tilting of the camera and the Raspberry pi 5 connected to the HQ camera that detect the falling object. This is also how the two subgroups are divided. With this project report detailing the image processing and detection of the overall project. The motion control subgroup handles the programming and controlling of the actuator. In figure 3.1 the complete prototype can be seen. On the left of the image is the actuator, camera and lens. On a custom mount. On the right is the Raspberry pi with some accessories.

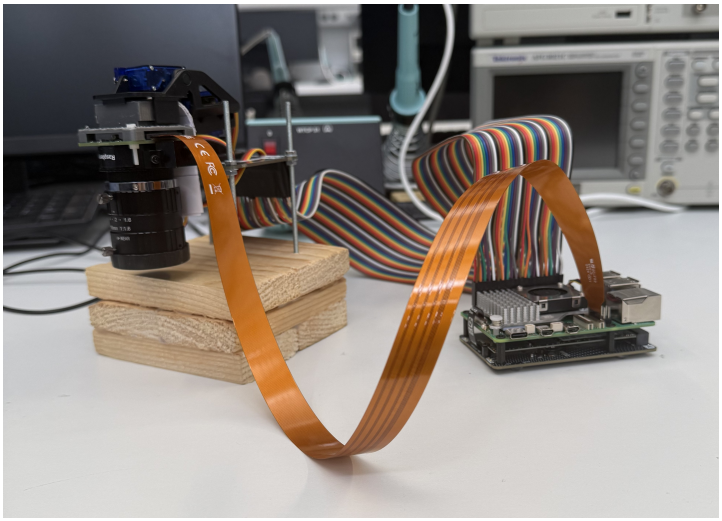


Figure 3.1: Complete Prototype

3.1.1. Raspberry Pi 5

The Raspberry Pi 5 (referred to as Pi) serves as the central processor of the prototype. It houses the complete pipeline: from image capturing to object detection and controlling the actuator. It uses several accessories called Hardware Attached on Top (HAT) modules. It has a active cooler with an aluminum heat sink and fan. This cooler makes sure that the Pi stays within its safe operation conditions with regards to temperature; it will throttle its own performance when it gets above 80°. Additionally, it has a 512 GB SSD card in its SSD slot to store data. Figure 3.2 shows the Pi itself, the array of pins on the upper right of the image are the GPIO pins. They connect the actuator to the Pi using jumper cables.

Figure 3.2: Raspberry pi 5

3.1.2. Raspberry pi High Quality Camera

The camera used was the Raspberry Pi HQ camera which consists of the Sony IMX477R image sensor, with a 12.3 megapixel sensor. This sensor allows for up to 120 FPS, which meets requirment MR-1a in the PoR. The minimum shutter speed of this camera of 0.5 ms also works to lower possible motion blur [14], required by Trade-off requirement section 3. Additionally, this camera allows for adjustable lenses, giving more freedom and flexibility.

3.1.3. Lens Selection

A few Pi lenses fit the program of requirements. The 8 mm, 16 mm, 25 mm and 35 mm. Each number denotes the focal length both in terms of weight as well as the portion of image the ball would take up. The vertical field of view of each of the lenses is shown in table 3.1. These are calculated using the formula 3.1 from [15]. H_{sensor} is the height of the sensor and fl the focal length, both in millimeters. The vertical height for the HQ camera sensor is 4.712 mm [14]. This gives the various field of views for each lens in table 3.1.

$$FOV = 2 \cdot \arctan\left(\frac{H_{\text{sensor}}}{2fl}\right) \tag{3.1}$$

Focal Length	Vertical FOV [°]
8 mm	32.8
16 mm	16.8
25 mm	10.8
35 mm	7.7

Table 3.1: Vertical FOV for respective lenses

A simple diagram of the experiment setup can be seen in Figure 3.3. Where the Field of view of course changes depending on the exact lens used. The falling object here being the stand-in of the proxy problem, namely the 50 mm cloth ball.

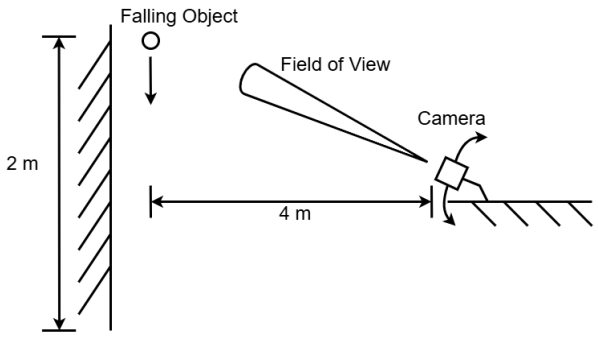
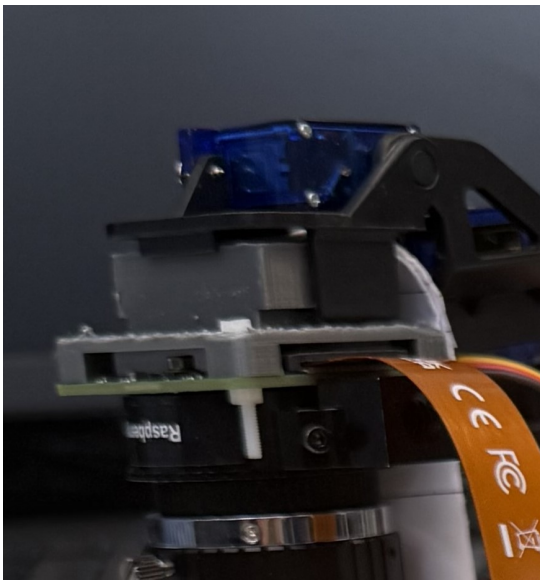


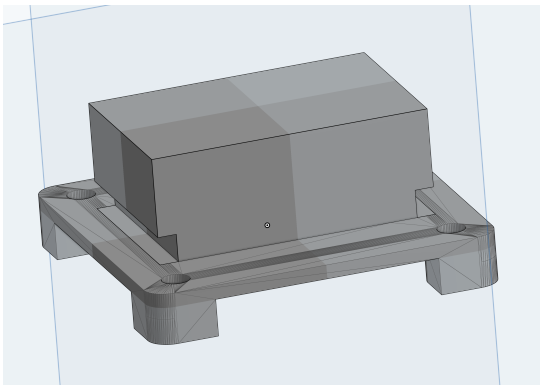
Figure 3.3: Camera Setup

3.1.4. Actuator setup

Normally the PIM183 Pan-Tilt would function as a HAT and press on the GPIO pins of the Raspberry Pi. However due to the active cooler and SSD HAT, there was now not enough room the connect the pins directly, so a 40-pin male to female header cable was used to bridge the gap instead. The pan-tilt hat experience a not insignificant amount of force when it tries to tilt the camera at full speed. In order to stabilize the system during operation it it was mounted to a block of wood using long bolts. This was not exactly an original Pi part, but worked to hold the actuator and camera down.
A final system setup problem that had to be solved was fitting the camera module to the PIM183. The Pan-Tilt HAT was designed for older camera modules that have a smaller and slimmer PCBs. To connect the PCB to a 3D printed backing plate was used that simply screwed into the already made holes of the raspberry pi camera module. The other side has a block with a small overhanging ledge where the original clips of the PIM183 interface attach to. The model and a zoomed in version of the prototype can be seen in figure 3.4. Here the orientation is similar of the model view as in the picture of the prototype. The PCB of the sensor is screwed in and the block clips into the actuator.



(a) Zoomed in Prototype



(b) 3D Model

Figure 3.4: Model and installed adapter side by side

4

Detailed Design

4.1. Frame Capturing

A typical free fall in the setup lasts approximately 0.5 seconds. Given the system's operational frame rate of 50 frames per second (FPS), this corresponds to capturing around 25 frames during the entire drop:

$$\text{Number of frames} = \text{fall duration} \times \text{frame rate} = 0.5\,\text{s} \times 50\,\text{FPS} = 25\,\text{frames}$$

Buffering enables smooth and reliable processing under varying number of input frames. By maintaining a small, efficiently managed buffer of incoming camera frames, the system improves robustness against processing delays and prevents frame drops, such that the latency is kept within strict timing constraints (e.g., 20ms per control cycle in this system).
The camera continuously streams frames into a buffer in shared memory, allowing the processing thread to always access the latest data without waiting. This approach decouples the camera capture mechanism from processing, ensuring that the control loop operates on the most recent available frame while the buffer absorbs minor timing variations between the camera and the image processor.

4.2. Frame differencing

Frame differencing was very appealing due to its simplicity, which also makes it very cheap in terms of computing costs. As long as the camera was static it could very accurately detect the moving object. It also allowed the system to detect the object in the very first frame where the object appeared, even if it wasn't actually in frame in its entirety. For the higher magnification lenses like the 35 mm, one or two extra frames make a significant difference if the ball is only in frame for 10 to 11 frames.
The steps of the frame differencing are shown in the flowchart in Figure 4.1, in which the function takes as input both the previous frame and current frame in grayscale. The difference value is found by taking the absolute difference and is then turned into a binary image by comparing it to a threshold. In this binary image, with purely black and white pixels, it is easier to do subsequent operations on.
This binary image is further cleaned using OpenCV's closing and opening morphology functions [16]. Opening removes noise in the background and closing removes noise inside of the foreground object. The actual results found using his method is discussed in Section 5.2.

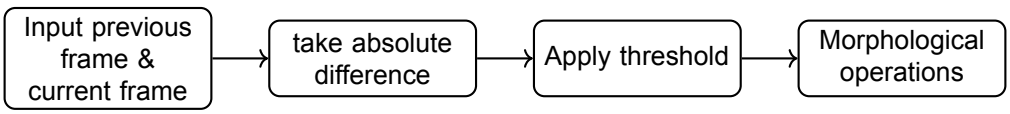


Figure 4.1: Frame Differencing Flowchart

(a) Opening operation **(b)** Closing operation

Figure 4.2: Morphology operation from [16]

4.3. Implementation Moving Camera Setup

4.3.1. Background Subtraction

The background subtraction method tried was using the `createBackgroundSubtractorMOG2` function from the OpenCV library [17]. This function makes a foreground mask by performing a subtraction from a background model, or frame and new frames. This method is most commonly used in static cameras with moving subjects. The most straight forward background was to use a solid color (like white) to separate the falling object from the background, even if the background was also moving.
The core part of the background subtraction code is seen in the flowchart in Figure 4.3. In the process frame box, the previous frame is compared to the current frame to make the background model. Then the background subtraction function is used to find the object inside the image. The masking step is turning this frame also into a binary image. Lastly, on this binary image the contours function from OpenCV finds the contours inside the binary image.

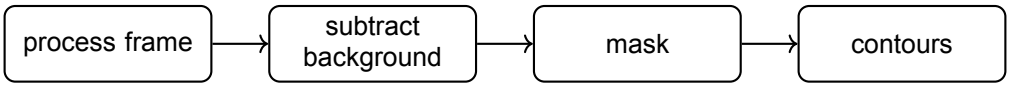


Figure 4.3: Background Subtraction Flowchart

4.3.2. Motion Compensation Attempt

The initial attempts with frame differencing and background subtraction highlighted a fundamental limitation: their reliance on static cameras. Their are several techniques to overcome this limitation. The approach that comes closest to the initial attempt of frame differencing and background subtraction is motion compensation [18]. Once the camera starts moving, the whole background changes from frame to frame, leading to false detections or complete failure. To address this, motion compensation was explored. This technique is designed to estimate and negate global camera motion in order to preserve only the relative motion of objects in the scene.
Typically for motion compensation the inter frame motion is estimated and a geometric transformation is applied to align the current frame with the previous one [19]. This typically begins by extracting keypoints from both frames using a feature detector such as ORB (Oriented Fast and Rotated Brief).Keypoints could be distinct trackable visual features such as corners or blobs. Once these keypoints are extracted in both frames, they are matched with each other, after which a geometric transformation matrix can be estimated. The current frame is then transformed to match the perspective of the previous frame, after which standard motion detection techniques, such as frame differencing and background subtraction, can be applied. An overview of this process is presented in Figure 4.4. This flowchart outlines the key steps of the motion compensation pipeline: capturing consecutive frames, estimating optical flow, computing and compensating for global motion, followed by thresholding and post-processing to identify true motion regions. If motion is detected, regions of interest are tracked and a bounding box is generated.

Even though this approach initially appeared promising, it was ultimately not implemented due to limitations specific to the experimental setup. The reliability of feature based motion estimation methods, such as those relying on ORB, is extremely dependent on the presence of a sufficiently textured back-

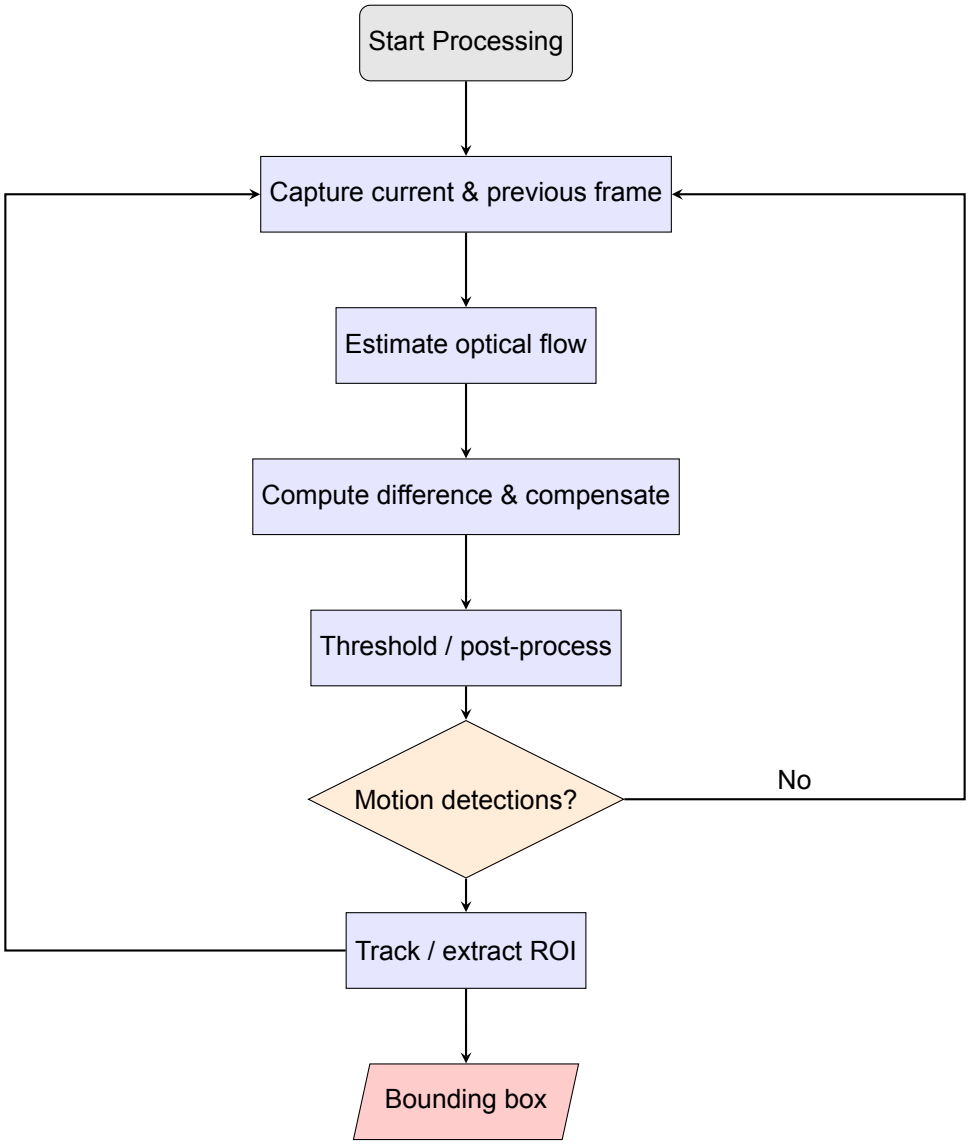


Figure 4.4: Motion Compensation Pipeline Flowchart

ground. However, in this system, a plain and uniform background is deemed ideal to avoid visual noise which could lead to false detections. While this design decision benefits object segmentation, it also leads to the background not containing distinctive features to match between two frames.

4.3.3. Final Implementation: Color Filtering

This final implementation uses a very different method based on color. Unlike frame differencing which relies on change of motion, this technique locates the object by it's color characteristics. Assuming the droplet is captured with a unique hue with no motion-blur, it is possible to extract its location directly from the frame regardless of how fast it is falling. This makes color filtering particularly appealing in scenarios where the background remains in motion. The flowchart of the pipeline of the method described here is presented in Figure 4.6. A general overview of this method and of a non-maximum suppression implementation are given in Algorithm Overview 4.3.3.1 and Non-maximum Suppression 4.3.3.2 respectively. A training-based feature-extraction of the object's color characteristics method is described in HSV Parameter Optimization via Error Minimization 4.3.3.3.

4.3.3.1. Algorithm Overview

This method begins by converting the input color frame from its standard BGR format into HSV. Next, a binary mask is created by thresholding the HSV image. Only pixels with a hue value within a certain range around a predefined target hue, and with sufficient saturation and brightness, are kept. All others are rejected. This results in a binary image with black and white values, where a black pixel indicates that the pixel is rejected and a white value indicates that it's pixel is within the provided target range.
As with frame differencing, the resulting binary image is often noisy. Part of that noise is due to reality rarely being a single color but instead some small amount of other colors being present. So morphological operations are applied to clean it up. Here, opening followed by closing is used. The opening step eliminates small isolated dots, typically background noise, while closing fills small holes inside detected foreground objects.
Contours are then extracted from this cleaned binary mask. These contours represent all the continuous blobs of pixels matching the selected hue. Each is turned into a bounding rectangle and filtered based on size, their area must exceed a minimum threshold to be considered valid. This removes small false positives.

4.3.3.2. Non-maximum Suppression

In cases where multiple overlapping regions are detected, a further step is needed. These overlaps typically arise because the object is approximated using several disjoint sets of pixels, which are each enclosed in separate rectangular bounding boxes. As rectangles are not shape-adaptive, their boundaries often overlap when the underlying pixel regions are spatially close. To address this, the bounding boxes are passed through a Non-maximum Suppression (NMS) method, which suppresses redundant overlapping boxes in favor of the one with the highest confidence, here defined by area. This step ensures that a single, best-fitting detection is selected. The flowchart of this method is presented in Figure 4.5.
Only if exactly one bounding box remains after this suppression step is the detection considered valid. As shown in the flowchart in Figure 4.5, the NMS process begins by collecting all candidate bounding boxes and sorting them by area, assuming larger area implies higher confidence. Then, as long as boxes remain in the list, the box with the largest area is selected. All other boxes that have a high overlap with it (measured by Intersection over Union (IoU) exceeding a threshold θ) are removed. This loop continues until no more boxes are left to evaluate. At the end of the suppression, a check is performed: if exactly one bounding box remains, it is accepted as the final detection. A red box is then drawn on the original frame, and the vertical center position y_{center} is returned. If instead zero or more than one box remain after suppression, the function returns nil and no detection is recorded for that frame. This strict selection criterion reduces false positives but may lead to dropped frames when multiple similar detections exist, which slightly lowers the real-time detection rate.

4.3.3.3. HSV Parameter Optimization via Error Minimization

To ensure reliable color-based detection in a given fixed setup, it is critical to select appropriate threshold ranges—referred to as target intervals—for hue, saturation, and value (HSV) in the color space. These intervals define which pixel values are considered a valid match for the object's color. Rather than selecting them manually, the system uses a per-scene training procedure to automatically determine suitable HSV bounds. It is important that the scene used for training (e.g., background, illumination, and object appearance) remains unchanged when applying the detection method with the trained parameters.

Training Methodology This training process aims to find optimal lower and upper bounds for the HSV parameters that yield the best detection accuracy for a target object across a sequence of video frames. A key aspect of this approach is the choice of the center of the proxy objects as the basis for detection, serving as a consistent spatial reference point. The central idea is to minimize the average squared error between the detected vertical position $\hat{y}$ and the actual position y across frames with known labeled data. The module for labeling data is listed in Appendix B.1.
Given a labeled dataset, where the object's known position per frame is provided, a cost function is

defined based on the MMSE [20] principle:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}_i(\theta) - y_i\right)^2$$

Here, θ represents the HSV filter parameters: the target hue H, the tolerance ΔH, and the lower and upper bounds for saturation and brightness $[S_{\text{min}}, S_{\text{max}}], [V_{\text{min}}, V_{\text{max}}]$. The number of valid samples N is fixed by the dataset and not part of the optimization process. The goal is to find the parameter set θ^* that minimizes the error $J(\theta)$.
If the cost function $J(\theta)$ is computed over the same set of N valid samples and assuming the detector's output is an approximately unbiased [20] estimator of the true y_i, then the standard error SE becomes:

$$SE = \sqrt{J(\theta)}$$

The standard error is used extensively as a performance metric in Chapter 5 Performance Evaluation.

Grid Search The optimization is performed via a brute-force grid search over a pre-defined range of HSV threshold parameters. The ranges are selected based on prior knowledge of the object's color characteristics and practical considerations for computational efficiency:

$$\begin{array}{lcl} H & = & \{6k \mid k \in \mathbb{N},\, 6k < 180\} \quad \text{(Hue center values from 0 to 174)} \\ \Delta H & = & \{2 + 4k \mid k \in \mathbb{N},\, 2 + 4k < 20\} \quad \text{(Hue window widths from 2 to 18)} \\ S_{\text{min}} & = & \{30 + 30k \mid k \in \mathbb{N},\, 30 + 30k < 120\} \quad \text{(Minimum saturation threshold)} \\ S_{\text{max}} & = & \{150 + 50k \mid k \in \mathbb{N},\, 150 + 50k < 256\} \quad \text{(Maximum saturation threshold)} \\ V_{\text{min}} & = & \{30 + 30k \mid k \in \mathbb{N},\, 30 + 30k < 120\} \quad \text{(Minimum value/brightness threshold)} \\ V_{\text{max}} & = & \{150 + 50k \mid k \in \mathbb{N},\, 150 + 50k < 256\} \quad \text{(Maximum value threshold)} \end{array} \tag{4.1}$$

These parameter ranges were chosen to cover a broad spectrum of possible HSV thresholds while avoiding extremes that are unlikely to be useful for typical object colors (e.g., very low brightness or saturation). The step sizes balance the trade-off between search resolution and computation time.
Each configuration is evaluated over a labeled image sequence. The color filtering pipeline is executed using the candidate thresholds, and the vertical object position is estimated per frame. A penalty is added during parameter estimation when a frame yields no valid detection to discourage configurations that frequently miss the object. However, this penalty is not included in the computation of the standard error, as it would bias the evaluation of the detected y-coordinates.
The implementation modules for labeling the image sequences and training the HSV parameters are listed in Listing B.1 and Listing B.2, respectively.

4.4. Conclusion

This chapter described the implementation of several detection methods for tracking a falling object. Frame differencing and background subtraction were effective only with a static camera. Motion compensation was investigated for moving-camera setups but not adopted due to insufficient background texture. The final method, HSV-based color filtering, was selected for its robustness to motion and optimized via grid search. This implementation forms the basis for the evaluation presented in the following chapter.

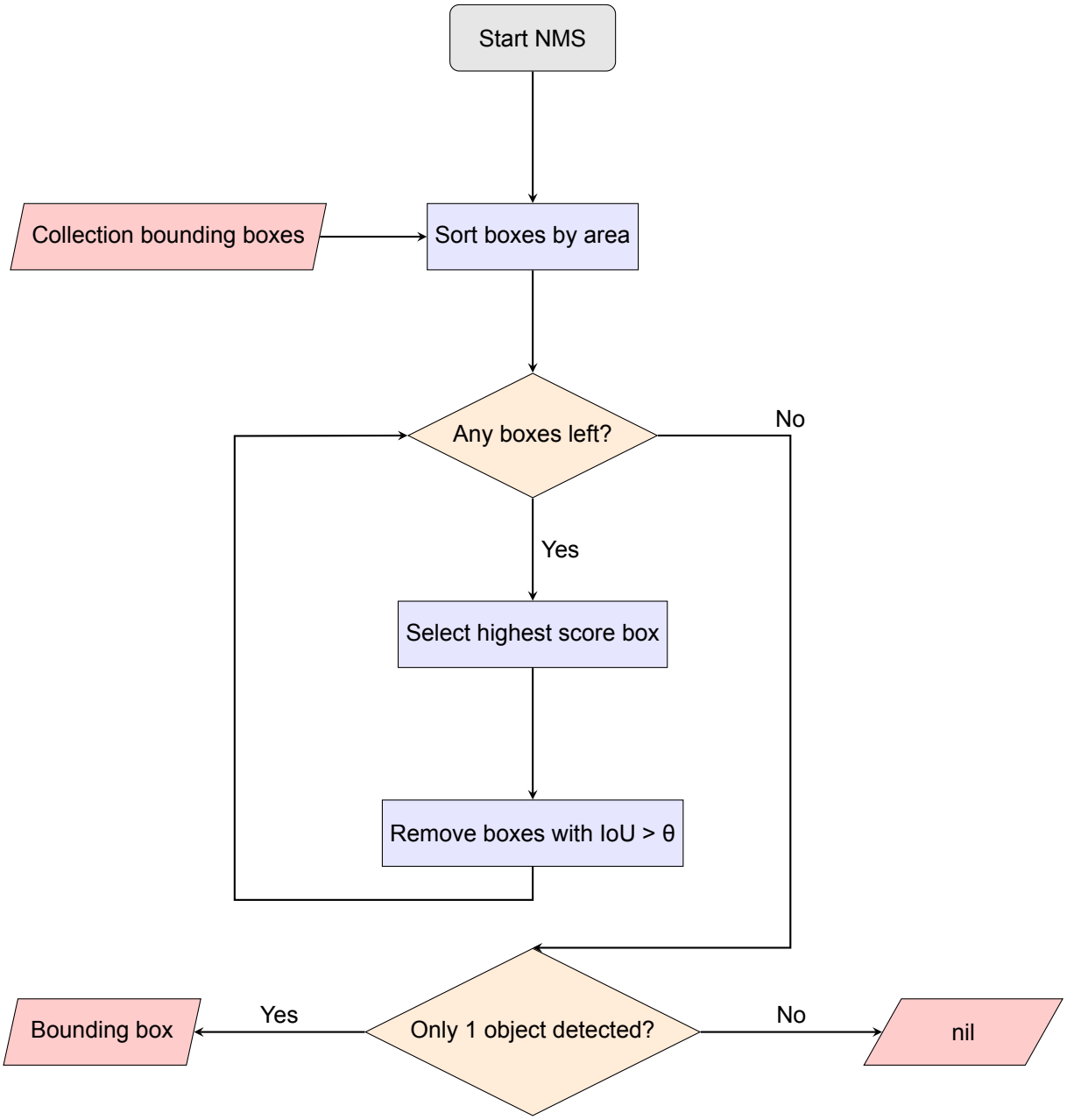


Figure 4.5: NMS Flowchart

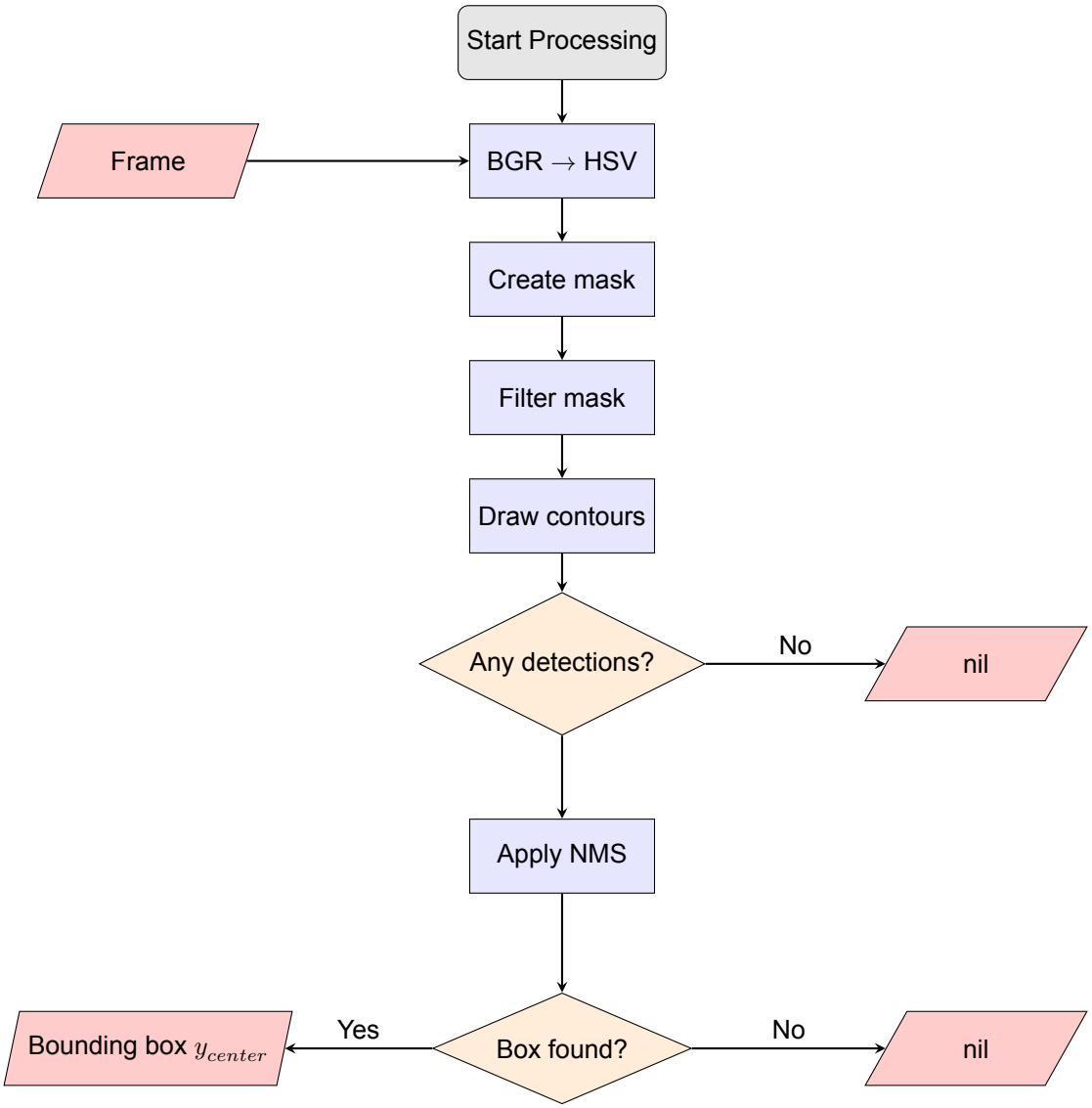


Figure 4.6: Color Filtering Pipeline Flowchart

5

Performance Evaluation

This chapter evaluates the performance of the image processing subsystem. The evaluation is limited strictly to this subsystem; no assessment of the complete system is performed.
Two detection and tracking methods are considered: frame differencing and color filtering. The evaluation focuses on detection accuracy and vertical (y-axis) tracking precision.
To measure tracking performance, the predicted object positions are compared against ground truth positions. These ground truth positions are known from manually labeled sequences created specifically for this purpose. The comparison enables computation of standard error in the vertical position estimates.
Detection accuracy is assessed qualitatively by inspecting whether the detected bounding box tightly encloses the object and a detection-to-object size ratio is estimated as an additional measure of accuracy.

5.1. Process

To evaluate the performance of both frame differencing and color filtering methods, experiments were conducted under distinct conditions for each approach.
For the frame differencing evaluation, static camera setups were used where images were captured from a fixed position with a frame height of 2000 px. Sequences were repeated for three lens configurations (8 mm, 16 mm, and 25 mm) to assess focal length effects.
For the color filtering evaluation, dynamic camera setups were employed with sequences captured with a frame height of 640 px while tilting the camera. All images used in these moving-camera experiments were manually labeled with ground truth y-positions of the target object's center point. These labels enabled both optimization of HSV parameters during training and subsequent error quantification during testing. These labels were also used to optimize HSV parameters through the process described in Section 4.3.3.3.

5.2. Frame Differencing

To evaluate the performance of the frame differencing method, a series of experiments was conducted using labeled image sequences captured with a static camera. For the three different lenses, the experiments are repeated several times to ensure a better estimate of the accuracy. Each setup consists of consecutive video frames and the corresponding known vertical positions of the target object. The goal of the evaluation is to measure how accurately the frame differencing method can detect and localize the target object using the different selection of lenses available.
All image sequences used for the evaluation have their three example frames listed in Appendix D.1. All static-images have a height of 2000 px, and their respective sequences are referred to by their directory names (/lens/index). The height of the target object in all frame differencing setups is close to 195 px, as indicated in Figure 5.1

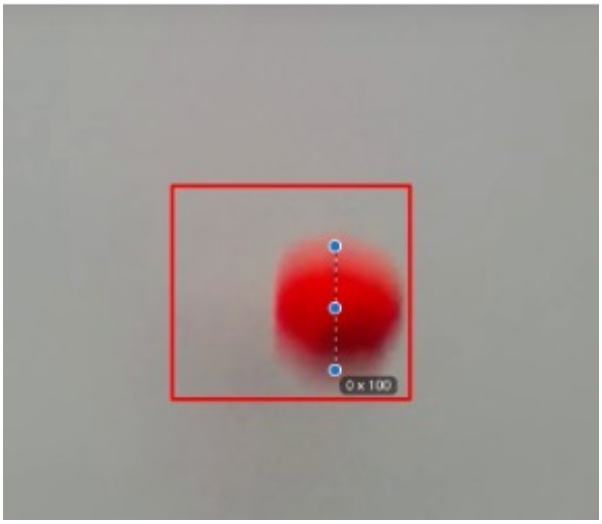


Figure 5.1: ball height in frame differencing setup 8 mm lens

5.2.1. Results

For each of the lenses, the experiments are repeated several times. After each experiment the standard error is determined and plotted for each separate lens in Figure 5.2. The average standard error in all frames for each of the experiments can be seen in Figure 5.2. A clear trend emerges: the tracking accuracy tends to decrease with increasing focal length. The eight mm lens consistently achieved the lowest standard error, with an average standard error across the three experiments of 26 px. In contrast, the 25 mm lens resulted in a significantly higher average of 87 px. Note that this average is heavily influenced by the last sequence for this focal length. As can be seen in Figure 5.2 the last sequence has a significant peak of 139 px. This significant peak is most likely the result of the quality of the images. As with higher focal lengths the field of view becomes significanlty smaller. This means that a small motion by the object results in a larger shift in pixels in the image, which leads to more motion blur. These errors should be put into perspective with the frame height. From the requirements it follows that the estimated object location must not deviate from the true location by more than one object height in the image plane. In Table 5.1 the different focal lengths with their corresponding object heights are shown. With the different object heights and the same framehieght across the different focal lengths a threshold can be determined for which object tracking is possible according to the set requirement.

Focal Length (mm)	Object Height (px)
8 mm	100 px
16 mm	200 px
25 mm	350 px

Table 5.1: Object height in pixels for different focal lengths

This object tracking threshold (OTB) can be determined with the following formula:

$$\frac{\text{object height}}{\text{frame height}} \times 100\%$$

In Figure 5.3 the normalized standard error for each focal length is plotted against the corresponding OTB. This normalized standard error is determined by dividing the average of the standard error for a focal length by the frame height. From the plot it follows that the demonstrated results for the standard error remained well below the threshold derived from the object's apparent height. This indicates that the predicted object location consistently remained within a range that can be considered acceptable in relation to the object's size in the frame for the static case.

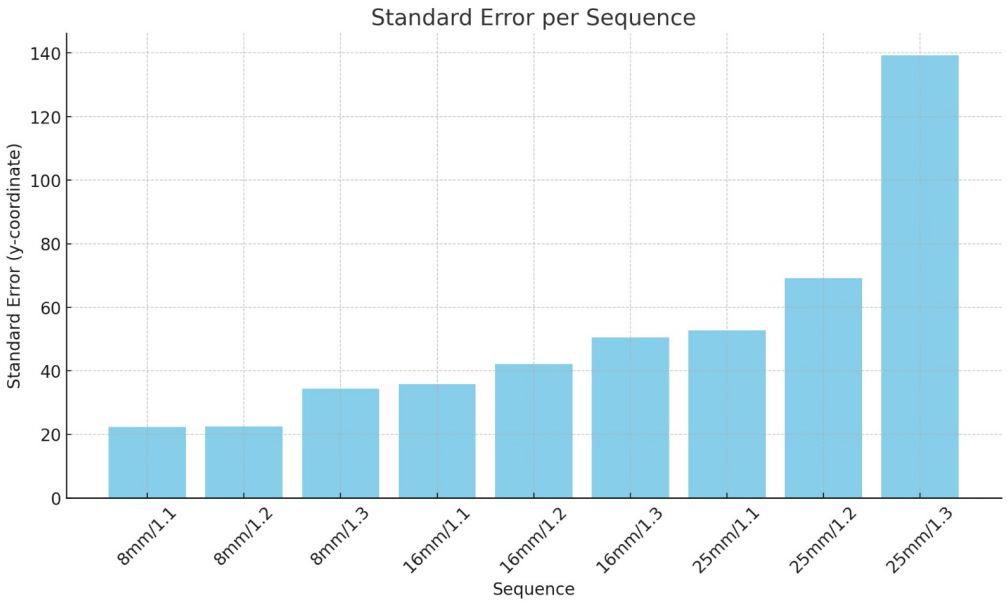


Figure 5.2: Standard error of the y-coordinate per sequence for each lens type.

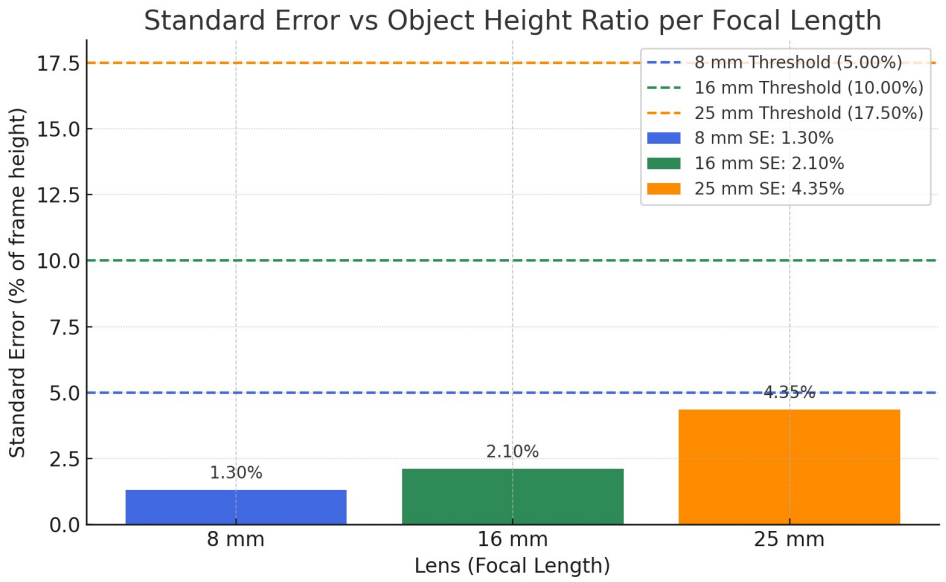


Figure 5.3: Normalized standard error for each focal length compared to object-height-based thresholds.

Since the result for the static situation satisfied the requirement of the precision, the next attempt was to implement this same approach for a moving camera. In the attemps for the moving camera it can be immediately noticed that this approach is not feasible for non static camera's. As can be seen in Figure 5.4 with a moving camera, several false detections are made. This is due to the fact that with any camera motion the entire frame is shifted, which results in large differences between the subsequent frames. These differences are mistakenly interpreted as motion, leading to false detections throughout the frame. Even attempts to only preserve the boxes with the biggest area are not trustworthy. In figure 5.4 the largest box does not contain the moving target. This shows that the approach of frame differencing is not sustainable for a moving camera.

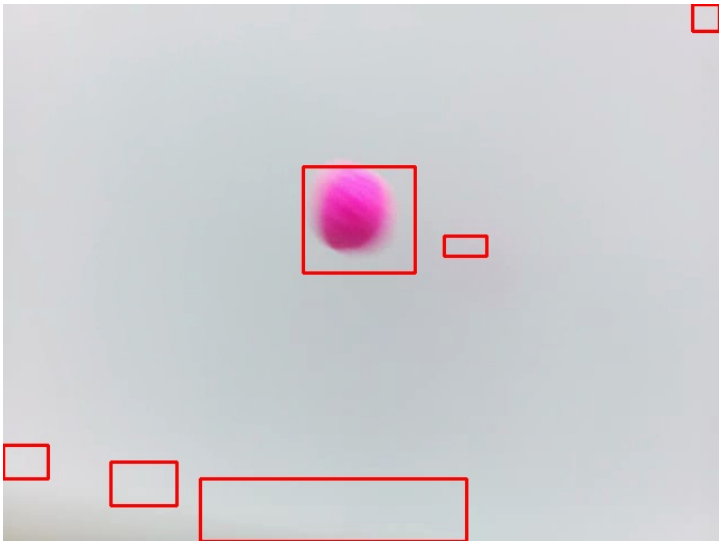


Figure 5.4: Example output of the frame differencing method with multiple detected bounding boxes. The target object (pink ball) is correctly detected, but several false positives are also present due to background noise and camera motion.

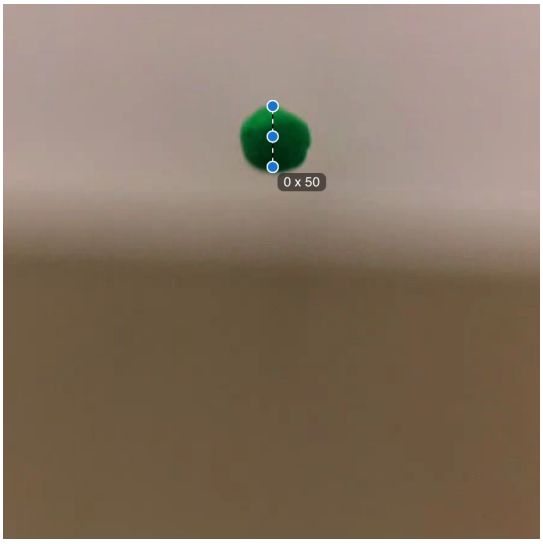
5.2.2. Discussion Frame Differencing

The frame differencing method was initially evaluated under controlled conditions using static camera setups. Across the tested focal lengths a clear relationship was observed between focal length and the standard error. Shorter focal lengths resulted in lower standard errors, with the 8 mm lens achieving an average SE of 26 pixels. In contrast to the 25 mm lens setup which resulted in a significantly higher average SE of 87 pixels, a result likely caused by motion blur in one of the sequences.
To contextualize these errors, a threshold was defined based on the object's apparent size in the image: the maximum allowable deviation was constrained to remained within the objects own height in the image frame. Once the standard errors were normalized with respect to the frame height, it was found that all configurations remained well below their respective thresholds, thereby satisfying the precision requirement for the static case.
Building further on that to a movable camera setup the method was found to be unsuitable for moving camera's. As demonstrated in 5.4, camera motion introduced false positives. These findings underscore a fundamental limitation of the frame differencing approach: its dependency on static backgrounds and stationary camera positioning.

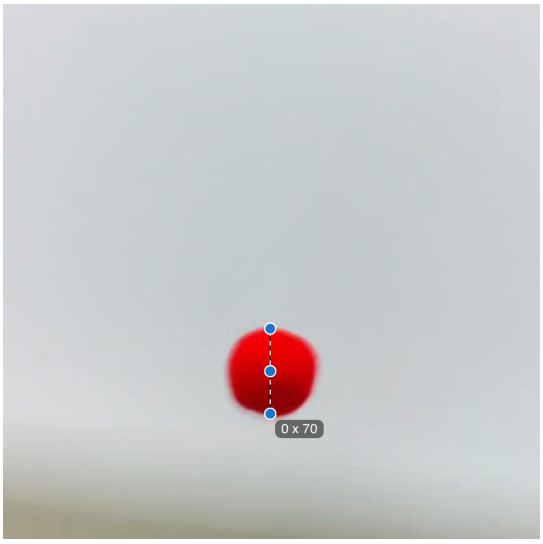
5.3. Color Filtering

To evaluate the performance of the HSV-based color filtering method just like in section 5.2.1, a series of experiments were conducted using labeled image sequences. Each setup consists of a sequence of video frames and corresponding known vertical positions of the target object. The goal of the evaluation is to measure how well the trained HSV parameters generalize both to the data they were trained on and to previously unseen labeled sequences in the same setup.
All image sequences used for the evaluation have their 3 example frames listed in Appendix C. All images captured with the tilting camera have a height of 640 px and their respective sequences are referred to by their directory name (`/lens/index`).

Note that the target object's pixel height differs slightly between lenses due to varying optical properties. Specifically, the object appears with a height of approximately 50 px in sequences recorded with the 25mm lens, and approximately 70 px in those recorded with the 35mm lens. This difference is visually illustrated in Figure 5.5, where the two setups are shown side by side for comparison.



(a) Ball captured with 25mm lens, showing a ball height of 50 px



(b) Ball captured with 35mm lens, showing a ball height of 70 px

Figure 5.5: Comparison of the target object's pixel height in different lens setups.

Some image samples could not be unambiguously labeled with an expected y-coordinate due to excessive motion-blur or due to the object being cut off at either the top or bottom frame edge. In this case, the standard error SE of each image sequence evaluation was computed with only the residuals whose expected and measured y-coordinates are known.

5.3.1. Results

For each setup, the HSV filter parameters $\theta = (H, \Delta H, S_{\text{min}}, S_{\text{max}}, V_{\text{min}}, V_{\text{max}})$ were optimized using the method described in Section 4.3.3.3. The cost function minimized during training was the average squared error between the detected and expected vertical positions, with a penalty of ten times the frame-height added for missed detections. These optimized HSV parameters are listed by their image sequence along with their calculated cost in Table 5.2. Note that the listed cost values are dimensionless metrics representing this cost function, and so their numerical values alone do not carry direct physical meaning but are useful for comparing optimization performance across setups.

Once the optimal HSV intervals were identified for a setup, these parameters were then evaluated on the trained image sequence, and for some intervals also on a different image sequence with the same falling object, captured in the same setup (background, illumination, target-object, etc. are all the same). The latter effectively measures the performance of the color filter with the automatically generated HSV target-intervals for constrained unseen data, called cross-validation.

The evaluation results of the former and latter are presented in Table 5.3 and Table 5.4 respectively, where n_{valid} signifies the number of samples where a labeled expected y-coordinate as well as a measured y-coordinate (non-nil) are known. n_{total} signifies the total number of samples in the given test setup. Finally, SE is the standard error in pixels.

Note also that in previous figures (e.g., Figure 5.2) the test setups included additional focal lengths such as 8 mm and 16 mm, which are not shown here but were part of earlier experiments.

Table 5.2: Estimated HSV parameters and associated cost per test image sequence

Test Setup	$Cost$	H	ΔH	S_{min}	S_{max}	V_{min}	V_{max}
/35mm/1.1	612.57	162	6	90	250	30	250
/35mm/2.1	774.34	162	18	30	200	30	250
/35mm/2.2	2697.49	6	10	60	250	30	250
/25mm/1.1	1895.35	60	10	30	250	90	200
/25mm/1.2	1832.07	156	14	30	250	30	250
/25mm/1.3	1538.00	72	14	30	250	30	150
/25mm/2.1	2566.02	6	6	90	250	30	150
/25mm/2.2	595.64	54	18	30	250	30	150

$Cost$ is the minimized cost function value. H is the central hue value; ΔH is the hue tolerance range. S_{min}, S_{max}, V_{min}, and V_{max} define the saturation and value ranges for the HSV filter.

Table 5.3: Detection validity and standard error in test image sequences using their respective estimated parameter intervals

Test Setup	$n_{\text{valid}} \,/\, n_{\text{total}}$	SE [px]	$SE_{\%}$
/35mm/1.1	33 / 35	49.96	7.81%
/35mm/2.1	23 / 25	36.13	5.64%
/35mm/2.2	14 / 25	35.85	5.60%
/25mm/1.1	11 / 25	41.81	6.53%
/25mm/1.2	16 / 25	37.48	5.86%
/25mm/1.3	10 / 15	45.16	7.06%
/25mm/2.1	5 / 24	49.05	7.66%
/25mm/2.2	9 / 17	29.43	4.60%

$n_{\text{valid}}/n_{\text{total}}$ is the count of valid detections versus total frames. SE is the standard error in pixels between detected and expected vertical positions. $SE_{\%}$ is the relative standard error as a percentage of the frame height.

Table 5.4: Cross-validated performance testing using HSV parameters from their respective counterpart image sequences in the same setup

Test Setup	$n_{\text{valid}} \,/\, n_{\text{total}}$	SE [px]	$SE_{\%}$	Counterpart Setup	$n_{\text{valid}} \,/\, n_{\text{total}}$	SE [px]	$SE_{\%}$
/35mm/1.1	18 / 35	64.34	10.05%	/35mm/2.1	23 / 25	36.44	5.69%
/25mm/1.3	10 / 15	46.32	7.24%	/25mm/2.2	9 / 17	31.99	5.00%
/25mm/1.1	0	-	-	/25mm/1.3	8 / 15	50.29	7.86%
/25mm/1.1	13 / 25	49.26	7.70%	/25mm/2.2	6 / 17	34.51	5.39%

5.3.2. Evaluation of Results

5.3.2.1. Optimized HSV Parameters

The estimated hue values H presented in Table 5.2 correspond to the dominant color of each target object. For visual reference, these colors and their associated parameters can be seen in Figures C.1–C.8 in Appendix C, where each row displays the segmented object using the respective HSV intervals.

It is also notable that the V_{max} parameter for setups /25mm/1.1 through /25mm/2.2 is consistently lower than for the /35mm setups. This difference likely reflects variations in illumination intensity across the experimental configurations, due to changes in lighting. Since the system is intended to operate under controlled laboratory conditions (see Section 2, Target Environment), this variation is expected to differ across different setups.

5.3.2.2. Measured Object Position

The system was tested with two lens configurations producing different ball heights: 50 px (25mm lens) and 70 px (35mm lens) against a constant frame height of 640 px.
Using the normalized evaluation metric, namely that the estimated object location must not deviate from the true location by more than one object height in the image plane, we establish the following benchmarks:

- For 25mm lens: 50 / 640 $\times$ 100% $\approx$ 7.81% threshold
- For 35mm lens: 70 / 640 $\times$ 100% $\approx$ 10.94% threshold

All measured relative standard errors ($SE_{\%}$) for both lens configurations in Table 5.3 satisfy this constraint. This demonstrates that the detection method maintains appropriate precision relative to the object size in the image frame.

This performance, combined with the system's higher valid detection rate (compared to alternatives like frame differencing), suggests an effective balance between detection consistency and precision. The stable positional estimates prove particularly beneficial for the downstream motion control subsystem, which prioritizes reliable, frequent updates of the object's position.

5.3.2.3. Cross-validated Color Filtering Method

Table 5.4 shows the cross-validation results assessing HSV parameter transferability between sequences of similar setups.
The highest detection accuracy was observed for test sequence `/25mm/2.2` when using HSV parameters derived from `/25mm/1.3`, achieving a relative standard error of 5.00% and 9 valid detections out of 17 frames. This indicates that parameters calibrated within a specific setup generalize well to other recordings under the same conditions.
Conversely, the reverse test (`/25mm/1.3` using `/25mm/2.2` parameters) resulted in a higher relative standard error of 7.24%, showing some variation in parameter robustness, which may be attributed to small differences in lighting or excessive motion-blur.
For the `/35mm` configuration, detection performance also remained within a reasonable margin, with relative standard errors such as 5.69% and 10.05% and moderate detection rates. Although performance varied between sequences, the overall trend indicates that HSV parameters can be effectively reused across sequences of the same experimental setup, provided the target object is similar.

Given that training on the target object is part of the system workflow, such per-setup calibration is sufficient for meeting the functional requirement of consistent local processing (see PoR, MR-1.c). While adaptive thresholding could further improve robustness under changing illumination, static HSV intervals remain a viable and practical solution in controlled environments.

5.3.3. Discussion of Final Implementation

The HSV color filtering approach was ultimately chosen as the final object detection method due to its robustness in a moving-camera setup, as demonstrated by the experimental results. Compared to frame differencing, color filtering delivered a significantly higher number of valid detections per image sequence and proved more resilient against the visual disturbances introduced by camera motion.

Tables 5.3 and 5.4 provide quantitative insight into the performance and limitations of this method. Comparing the relative standard errors ($SE\%$) against the ball-height-proportional metric (SE% < ball height / frame height × 100%), this color-filtering method demonstrates consistent compliance across both lens configurations, with the majority of measurements satisfying these constraints. This performance validates the method's ability to maintain appropriate precision relative to the object's size in the image frame.
A key strength of the method is its ability to produce a high valid-detection rate. Most sequences yielded over 50% valid detections (Table 5.3), a substantial improvement over the frame differencing method, which suffered from false positives in the presence of background motion. This makes color filtering particularly suitable for real-time tracking systems where detection dropout can disrupt downstream control subsystems.
Cross-validation experiments further support the robustness of the method under similar test conditions. Table 5.4 shows that HSV parameters trained on one sequence typically generalize well to another se-

quence within the same setup and lighting conditions. For example, the parameters from `/35mm/2.1` achieved low standard error and high validity when applied to `/35mm/1.1`, suggesting that in controlled environments, retraining is not always required for each recording instance.
The consistency of the optimal parameters across such related sequences (Table 5.2) also indicates that in stable laboratory setups, per-sequence calibration of HSV intervals can be an effective and low-overhead strategy. While some variation in detection quality remains, especially when setups differ in lighting or background, this is manageable given that training on a representative target object is part of the system's expected deployment workflow.

5.4. Conclusion

The evaluation of the frame differencing method demonstrated that, under static camera conditions, the standard error remained within acceptable bounds relative to the object height across all tested focal lengths. The method proved capable of maintaining adequate precision in detecting and localizing the object when the background remained stationary. However, in dynamic scenarios involving camera motion, the approach consistently produced false positives due to global frame shifts, rendering it unsuitable for use in moving-camera systems. These limitations highlight the method's dependency on static environments and its lack of robustness under practical deployment conditions. Due to these limitations, an alternative approach was considered namely HSV color filtering. This approach demonstrates appropriate precision when evaluated through the ball-height-proportional metric while offering superior robustness in dynamic conditions and a high valid detection rate. Its suitability for integration with a moving actuator-based camera system, combined with acceptable generalization within constrained conditions, justifies its selection as the final implementation. Future improvements could include adaptive thresholding or dynamic color modeling to reduce sensitivity to illumination variation and setup-specific tuning.

6

Conclusion

This report investigates the development and evaluation of an image processing subsystem for real-time tracking of a vertically falling object. The primary goal was to detect and localize the object accurately across a variety of camera configurations and motion scenarios.
Two object detection approaches were implemented and assessed: frame differencing and HSV color filtering. The evaluation focused on standard error in the vertical (y-axis) position of the object as a metric for tracking precision and employed a proportional threshold based on the object's height within the frame.
Frame differencing demonstrated acceptable tracking accuracy under static conditions. For all tested focal lengths (8 mm, 16 mm, and 25 mm), the normalized standard error remained below the object-height-based threshold, indicating sufficient positional precision when the camera remained fixed. However, the method was shown to be highly sensitive to background motion and entirely unsuitable for dynamic scenarios. Even minor camera movement resulted in extensive false detections due to global frame shifts, thereby undermining the method's reliability for real-time applications involving an actuated camera.

To overcome these limitations, the HSV color filtering approach was adopted and subjected to equivalent evaluation criteria. The experimental evaluation demonstrates that the HSV color filtering method achieves reliable object detection under controlled conditions, meeting the specified accuracy requirements. The measured standard errors remain within the permissible margin relative to the object height in the image plane, with relative errors consistently below their respective calculated thresholds. This method outperforms frame differencing in robustness, particularly in dynamic camera scenarios, yielding higher valid detection rates and greater resilience against motion-induced artifacts.
Cross-validation tests confirm that optimized HSV parameters generalize effectively across sequences with similar setups, though performance varies slightly due to lighting differences or motion blur. The method's consistency supports its suitability for laboratory environments where conditions are stable and per-setup calibration is feasible.
It can therefore be concluded that while frame differencing may serve as a baseline method under constrained conditions, its practical applicability is limited. HSV color filtering, by contrast, meets both the precision and robustness requirements necessary for real-time object tracking in dynamic environments. As such, it has been selected as the final implementation for the image processing subsystem.

It should be noted that real-time performance aspects, such as computational delays or scheduling constraints, were not explicitly evaluated in this study. Additionally, the shutter speed of the camera was automatically set at 8 ms or 1/120. By manually increasing the shutter speed to its maximum of 0.5 ms would have decreased motion blur. At the expense of having needed extra lighting. Future work should include latency measurements and real-time system integration to assess temporal performance. Further improvements could explore adaptive thresholding techniques to enhance robustness under variable illumination.

References

[1] K.-S. Kwon, “Speed measurement of ink droplet by using edge detection techniques,” *Measurement*, vol. 42, no. 1, pp. 44–50, 2009, ISSN: 0263-2241. DOI: https://doi.org/10.1016/j.measurement.2008.03.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0263224108000663.

[2] J. Liu, S. Yu, X. Liu, G. Lu, Z. Xin, and J. Yuan, “Super-resolution semantic segmentation of droplet deposition image for low-cost spraying measurement,” *Agriculture*, vol. 14, no. 1, 2024, ISSN: 2077-0472. DOI: 10.3390/agriculture14010106. [Online]. Available: https://www.mdpi.com/2077-0472/14/1/106.

[3] A. Lukežič, Ž. Trojer, J. Matas, and M. Kristan, “A new dataset and a distractor-aware architecture for transparent object tracking,” *International Journal of Computer Vision*, vol. 132, no. 8, pp. 2729–2742, 2024.

[4] M. Vikram, “Methods of image edge detection: A review,” *Journal of Electrical & Electronic Systems*, vol. 04, Jan. 2015. DOI: 10.4172/2332-0796.1000150.

[5] İ. Avci, “Threshold values of different classical edge detection algorithms,” *Traitement du Signal*, vol. 39, pp. 1775–1780, Nov. 2022. DOI: 10.18280/ts.390536.

[6] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, “Visual object tracking using adaptive correlation filters,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2544–2550. DOI: 10.1109/CVPR.2010.5539960.

[7] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “High-speed tracking with kernelized correlation filters,” *CoRR*, vol. abs/1404.7584, 2014. arXiv: 1404.7584. [Online]. Available: http://arxiv.org/abs/1404.7584.

[8] C. Stauffer and W. Grimson, “Adaptive background mixture models for real-time tracking,” in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 2, 1999, 246–252 Vol. 2. DOI: 10.1109/CVPR.1999.784637.

[9] M. Piccardi, “Background subtraction techniques: A review,” in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, IEEE, 2004, pp. 3099–3104. DOI: 10.1109/ICSMC.2004.1400815.

[10] I. Krizhevsky A. Sutskever and G. Hinton, “Imagenet classification with deep convolutional neural networks,” *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386.

[11] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik, “Contour detection and hierarchical image segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898–916, 2011. DOI: 10.1109/TPAMI.2010.161.

[12] Z. Zong, G. Song, and Y. Liu, “Detrs with collaborative hybrid assignments training,” in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 6725–6735. DOI: 10.1109/ICCV51070.2023.00621.

[13] X. Soria Poma, Y. Li, M. Rouhani, and A. Sappa, “Tiny and efficient model for the edge detection generalization,” Oct. 2023, pp. 1356–1365. DOI: 10.1109/ICCVW60793.2023.00147.

[14] “Raspberry pi camera documenation,” 2025, Accessed on 12-6-2025. [Online]. Available: https://www.raspberrypi.com/documentation/accessories/camera.html.

[15] E. Gamy, *Camera sensor size fov (3)*, [Online], Appendix: “How to calculate angle of view”, 2015. [Online]. Available: https://www.academia.edu/34473790/Camera_Sensor_Size_FOV_3_.

[16] “Morphology, open source computer vision.” Accessed on 2025-05-28. (2025), [Online]. Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html.

[17] “Background subtraction, open source computer vision.” Accessed on 2025-06-08. (2025), [Online]. Available: https://docs.opencv.org/4.x/d1/dc5/tutorial_background_subtraction.html.

[18] M.-N. Chapel and T. Bouwmans, “Moving objects detection with a moving camera: A comprehensive review,” *Computer Science Review*, vol. 38, p. 100 310, 2020, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100310.

[19] J. Ke, A. J. Watras, J.-J. Kim, H. Liu, H. Jiang, and Y. H. Hu, “Efficient online real-time video stabilization with a novel least squares formulation and parallel ac-ransac,” *Journal of Visual Communication and Image Representation*, vol. 96, p. 103 922, 2023, ISSN: 1047-3203. DOI: https://doi.org/10.1016/j.jvcir.2023.103922. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1047320323001724.

[20] R. D. Yates and D. J. Goodman, *Probability and Stochastic Processes: A Friendly Introduction for Electrical and Computer Engineers*, 3rd ed. Wiley, 2014.

A

Algorithms

Listing A.1: Frame Differencing Module

```
"""
MIT License

Copyright (c) 2025 Muhammed Enes Kaya, Naufal el Khatibi, Timo Haas
Copyright (c) 2022 Isaac Berrios

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""
import cv2 as cv
import numpy as np


def process_frames(prev_gray, curr_gray, curr_color):
    diff = cv.absdiff(curr_gray, prev_gray)
    _, bw = cv.threshold(diff, 5, 255, cv.THRESH_BINARY)
    kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
    bw = cv.morphologyEx(bw, cv.MORPH_OPEN, kernel)
    bw = cv.morphologyEx(bw, cv.MORPH_CLOSE, kernel)

    contours, _ = cv.findContours(bw, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    output = curr_color.copy()
    detections = []

    for cnt in contours:
        x, y, w, h = cv.boundingRect(cnt)
        area = w * h

        MIN_AREA_THRESH = 400
        if area > MIN_AREA_THRESH:
            detections.append([x, y, x + w, y + h, area])

    boxes = [det[:4] for det in detections]
    scores = [det[4] for det in detections]

    keep = _non_max_suppression(np.array(boxes), np.array(scores), 0)

    for box in keep:
        x1, y1, x2, y2 = box
        cv.rectangle(output, (x1, y1), (x2, y2), (0, 0, 255), 2)

    if len(keep) > 1:
        return None, output, bw

    return (y1 + y2) / 2, output, bw


def _remove_contained_bboxes(boxes):
    """
    From: https://github.com/itberrios/CV_projects/blob/main/motion_detection/
        detection_with_frame_differencing.ipynb
    """
    check_array = np.array([True, True, False, False])
    keep = list(range(len(boxes)))
```

```
    for i in keep[:]:
        for j in range(len(boxes)):
            if i != j and np.all((np.array(boxes[j]) >= np.array(boxes[i])) == check_array):
                try:
                    keep.remove(j)
                except ValueError:
                    continue
    return keep


def _non_max_suppression(boxes, scores, threshold):
    """
    From: https://github.com/itberrios/CV_projects/blob/main/motion_detection/
        detection_with_frame_differencing.ipynb
    """
    boxes = boxes[np.argsort(scores)[::-1]]
    order = _remove_contained_bboxes(boxes)
    keep = []

    while order:
        i = order.pop(0)
        keep.append(i)
        new_order = []
        for j in order:
            xx1 = max(boxes[i][0], boxes[j][0])
            yy1 = max(boxes[i][1], boxes[j][1])
            xx2 = min(boxes[i][2], boxes[j][2])
            yy2 = min(boxes[i][3], boxes[j][3])

            inter_area = max(0, xx2 - xx1) * max(0, yy2 - yy1)
            area_i = (boxes[i][2] - boxes[i][0]) * (boxes[i][3] - boxes[i][1])
            area_j = (boxes[j][2] - boxes[j][0]) * (boxes[j][3] - boxes[j][1])
            union = area_i + area_j - inter_area

            iou = inter_area / union if union != 0 else 0

            if iou <= threshold:
                new_order.append(j)
        order = new_order

    return boxes[keep]
```

Listing A.2: Color Filtering Module

```
"""
MIT License

Copyright (c) 2025 Muhammed Enes Kaya, Naufal el Khatibi, Timo Haas
Copyright (c) 2022 Isaac Berrios

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""
import cv2 as cv
import numpy as np

```

```
28
29 def process_frames(curr_color, target_hue, hue_tolerance, s_min=50, s_max=255, v_min=50,
       v_max=255):
30     hsv = cv.cvtColor(curr_color, cv.COLOR_BGR2HSV)
31
32     h_low = max(target_hue - hue_tolerance, 0)
33     h_high = min(target_hue + hue_tolerance, 179)
34
35     lower_bound = np.array([h_low, s_min, v_min])
36     upper_bound = np.array([h_high, s_max, v_max])
37
38     mask = cv.inRange(hsv, lower_bound, upper_bound)
39
40     kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))
41     mask = cv.morphologyEx(mask, cv.MORPH_OPEN, kernel)
42     mask = cv.morphologyEx(mask, cv.MORPH_CLOSE, kernel)
43
44     contours, _ = cv.findContours(mask, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
45     output = curr_color.copy()
46     detections = []
47
48     for cnt in contours:
49         x, y, w, h = cv.boundingRect(cnt)
50         area = w * h
51         if area > 10:
52             detections.append([x, y, x + w, y + h, area])
53
54     boxes = [det[:4] for det in detections]
55     scores = [det[4] for det in detections]
56     keep = _non_max_suppression(np.array(boxes), np.array(scores), 0)
57
58     if len(keep) != 1:
59         return None, output, mask
60
61     x1, y1, x2, y2 = keep[0]
62     y_measured = (y1 + y2) / 2
63     output = cv.rectangle(output, (x1, y1), (x2, y2), (0, 0, 255), 2)
64     return y_measured, output, mask
65
66
67 def _remove_contained_bboxes(boxes):
68     """
69     From: https://github.com/itberrios/CV_projects/blob/main/motion_detection/
           detection_with_frame_differencing.ipynb
70     """
71     check_array = np.array([True, True, False, False])
72     keep = list(range(len(boxes)))
73     for i in keep[:]:
74         for j in range(len(boxes)):
75             if i != j and np.all((np.array(boxes[j]) >= np.array(boxes[i])) == check_array):
76                 try:
77                     keep.remove(j)
78                 except ValueError:
79                     continue
80     return keep
81
82
83 def _non_max_suppression(boxes, scores, threshold):
84     """
85     From: https://github.com/itberrios/CV_projects/blob/main/motion_detection/
           detection_with_frame_differencing.ipynb
86     """
87     boxes = boxes[np.argsort(scores)[::-1]]
88     order = _remove_contained_bboxes(boxes)
89     keep = []
90
91     while order:
92         i = order.pop(0)
93         keep.append(i)
94         new_order = []
95         for j in order:
```

```
            xx1 = max(boxes[i][0], boxes[j][0])
            yy1 = max(boxes[i][1], boxes[j][1])
            xx2 = min(boxes[i][2], boxes[j][2])
            yy2 = min(boxes[i][3], boxes[j][3])

            inter_area = max(0, xx2 - xx1) * max(0, yy2 - yy1)
            area_i = (boxes[i][2] - boxes[i][0]) * (boxes[i][3] - boxes[i][1])
            area_j = (boxes[j][2] - boxes[j][0]) * (boxes[j][3] - boxes[j][1])
            union = area_i + area_j - inter_area

            iou = inter_area / union if union != 0 else 0

            if iou <= threshold:
                new_order.append(j)
        order = new_order

    return boxes[keep]
```

B

Modules

Listing B.1: Labeling Module

```
"""
MIT License

Copyright (c) 2025 Muhammed Enes Kaya

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""
import os
import sys
import cv2
import time
import json

from algorithms.color_filter import process_frames
from util import load_frames, replace_last


def main():
    images_dir = "./training-set/moving-camera/"

    instance_reference_xys_map = {
        "35mm/1.1": [],
        "35mm/2.1": [],
        "35mm/2.2": [],
        "25mm/1.1": [],
        "25mm/1.2": [],
        "25mm/1.3": [],
        "25mm/2.1": [],
        "25mm/2.2": []
    }

    for inst_idx, (instance, xys) in enumerate(instance_reference_xys_map.items(), 1):
        frames = load_frames(images_dir + instance)

        for frame_idx, frame in enumerate(frames, 1):
            display = frame.copy()
            status = f"{inst_idx}/{len(instance_reference_xys_map)} {instance} {frame_idx}/{
                len(frames)}"
            cv2.putText(display, status, (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,255), 2)

            cv2.imshow("Reference: click on the center of the ball", display)
            cv2.setMouseCallback("Reference: click on the center of the ball",
                                 on_click, (instance, instance_reference_xys_map, frame_idx))

            if cv2.waitKey(60000) in (ord('c'), 27):
                continue

        with open(f"{replace_last(instance, "/", "_")}.json", 'w') as f:
            json.dump({instance: xys}, f)


def on_click(event, x, y, flags, param):
```

```
    global _user_input_lock
    if event == cv2.EVENT_LBUTTONDOWN:
        _user_input_lock = True

        instance = param[0]
        xys = param[1][param[0]]

        clicked_point = (x, y)
        xys.append((param[2], clicked_point))

        print(f"Reference for {instance} at {param[2]} in JSON: {(param[2], xys[len(xys) - 1])}")


def load_frames(folder_path):
    frame_files = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(
        folder_path, f))]

    image_extensions = ['.jpg', '.jpeg', '.png', '.bmp', '.tiff', '.tif']
    frame_files = [f for f in frame_files if os.path.splitext(f)[1].lower() in
        image_extensions]

    frame_files.sort(key=lambda x: int(''.join(filter(str.isdigit, x))))

    frames = []
    for frame_file in frame_files:
        frame_path = os.path.join(folder_path, frame_file)
        frame = cv2.imread(frame_path)
        if frame is not None:
            frames.append(frame)
        else:
            print(f"Warning: Could not load image {frame_file}")

    return frames


def replace_last(source_string, replace_what, replace_with):
    head, _sep, tail = source_string.rpartition(replace_what)
    return head + replace_with + tail


if __name__ == "__main__":
    main()
```

Listing B.2: Training Module

```
"""
MIT License

Copyright (c) 2025 Muhammed Enes Kaya

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""
import json
import os
```

```
26
27 import numpy as np
28 import cv2 as cv
29
30 from itertools import product
31 from algorithms.color_filter import process_frames
32 from util import load_frames, load_expected_positions
33
34
35 def compute_cost(expected_dict, frames_data,
36                  h_range, h_tol_range,
37                  s_min_range, s_max_range,
38                  v_min_range, v_max_range):
39
40     best_params = None
41     min_cost = float("inf")
42
43     for h, htol, s_min, s_max, v_min, v_max in product(
44         h_range, h_tol_range, s_min_range, s_max_range, v_min_range, v_max_range
45     ):
46         if s_min >= s_max or v_min >= v_max:
47             continue  # skip invalid intervals
48
49         cost = 0
50         total_frames = 0
51
52         for frame_idx, expected_pos in expected_dict.items():
53             frame = frames_data[frame_idx]
54             y_measured, _, _ = process_frames(
55                 frame, h, htol, s_min, s_max, v_min, v_max
56             )
57
58             y_expected = expected_pos[1]
59             if y_measured is None:
60                 # Penalize missed detections heavily
61                 cost += 5000
62             else:
63                 cost += (y_measured - y_expected) ** 2
64
65             total_frames += 1
66
67         avg_cost = cost / total_frames if total_frames > 0 else float("inf")
68
69         if avg_cost < min_cost:
70             min_cost = avg_cost
71             best_params = (h, htol, s_min, s_max, v_min, v_max)
72
73     return best_params, min_cost
74
75
76 instances = [
77     "35mm/1.1",
78     "35mm/2.1",
79     "35mm/2.2",
80
81     "25mm/1.1",
82     "25mm/1.2",
83     "25mm/1.3",
84     "25mm/2.1",
85     "25mm/2.2"
86 ]
87
88 for instance in instances:
89     images_path = "./training-set/moving-camera/" + instance
90     json_path = f"./training-set/Batch␣1/{instance}.json"
91
92     expected = load_expected_positions(json_path, instance)
93     frames_data = load_frames(images_path)
94     h_range = range(0, 180, 6)
95     h_tol_range = range(2, 20, 4)
96     s_min_range = range(30, 120, 30)
```

```
    s_max_range = range(150, 256, 50)
    v_min_range = range(30, 120, 30)
    v_max_range = range(150, 256, 50)


    best_out, best_cost = compute_cost(
        expected, frames_data,
        h_range, h_tol_range,
        s_min_range, s_max_range, v_min_range, v_max_range
    )

    print(f"Final best Params for instance {instance} (hue, hue_tolerance): {best_out}, with
        cost: {best_cost}")
```

Listing B.3: Testing Module

```
"""
MIT License

Copyright (c) 2025 Muhammed Enes Kaya

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
"""
import os
import sys
import cv2
import time

from algorithms.color_filter import process_frames
from util import load_frames


test_instance_color_map = {
    "/35mm/1.1": (162, 6 , 90, 250, 30, 250),  # cost=612.5735294117648
    "/35mm/2.1": (162, 18, 30, 200, 30, 250),  # cost=774.3382352941177
    "/35mm/2.2": (6  , 10, 60, 250, 30, 250),  # cost=2697.4895833333335

    "/25mm/1.1": (60 , 10, 30, 250, 90, 200),  # cost=1895.35
    "/25mm/1.2": (156, 14, 30, 250, 30, 250),  # cost=1832.0714285714287
    "/25mm/1.3": (72 , 14, 30, 250, 30, 150),  # cost=1538.0
    "/25mm/2.1": (6  , 6,  90, 250, 30, 150),  # cost=2566.0208333333335
    "/25mm/2.2": (54 , 18, 30, 250, 30, 150),  # cost=595.6428571428571
}

recording_id = time.strftime('%y%m%d%H%M%S', time.gmtime())
dir_root = f"/Users/enoks/Downloads/{recording_id}/"
os.makedirs(dir_root, exist_ok=True)
for index, (test_key, test_val) in enumerate(sorted(list(test_instance_color_map.items()))):
    folder_path = "./training-set/moving-camera" + test_key
    frames_list = load_frames(folder_path)

    for i in range(0, len(frames_list)):
        frame =  frames_list[i]

        processor_output = process_frames(
```

```
56              frame,
57              test_val[0],
58              test_val[1]
59              )
60
61          y, output_image, mask = processor_output
62
63          cv2.imshow(f"[{test_key}] Colored frame differencing", output_image)
64          cv2.imwrite(dir_root + f"frame_{index:06d}.jpg", frame)
65          cv2.waitKey(1000)
```

C

Moving Camera Sample Frames from Each Test Setup

Figure C.1: 35mm lens, setup 1.1

Figure C.2: 35mm lens, setup 2.1

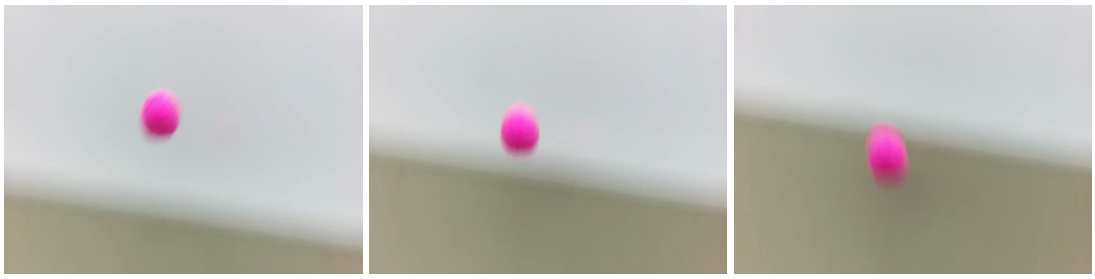


Figure C.3: 35mm lens, setup 2.2

Figure C.4: 25mm lens, setup 1.1

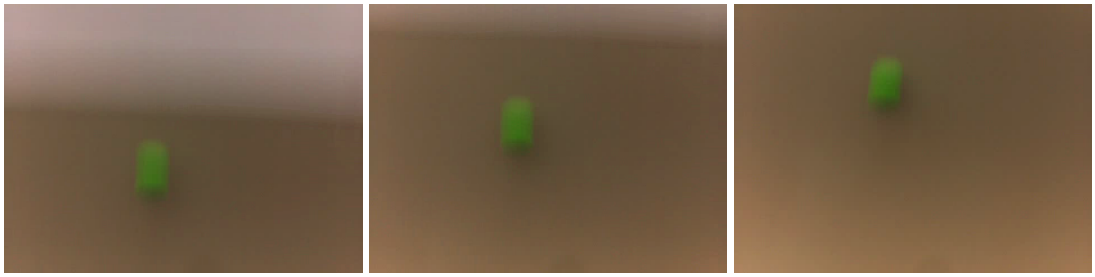


Figure C.5: 25mm lens, setup 1.2

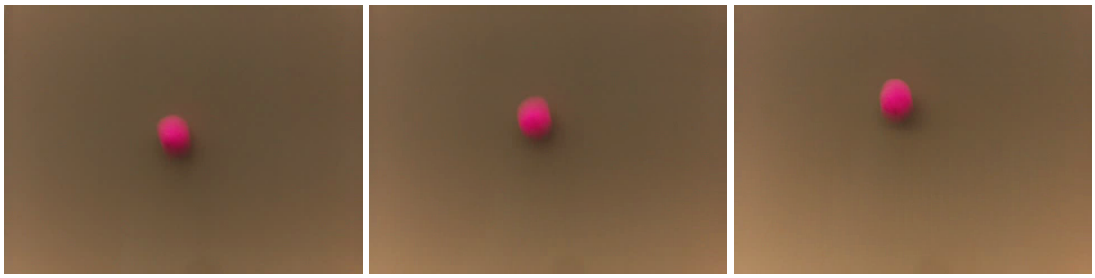


Figure C.6: 25mm lens, setup 1.3

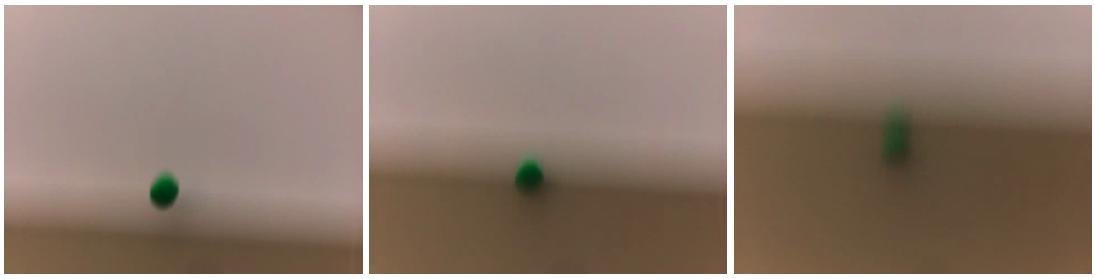


Figure C.7: 25mm lens, setup 2.1

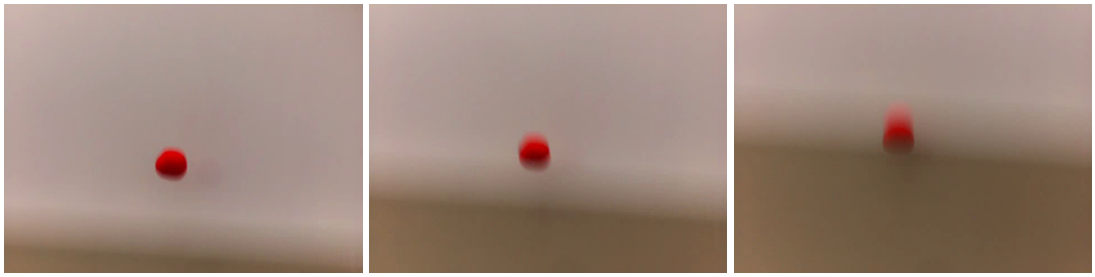
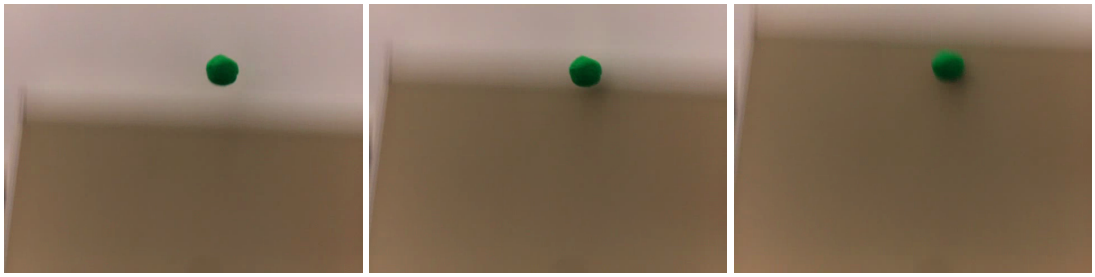


Figure C.8: 25mm lens, setup 2.2



D

Static Camera Sample Frames from Each Test Setup

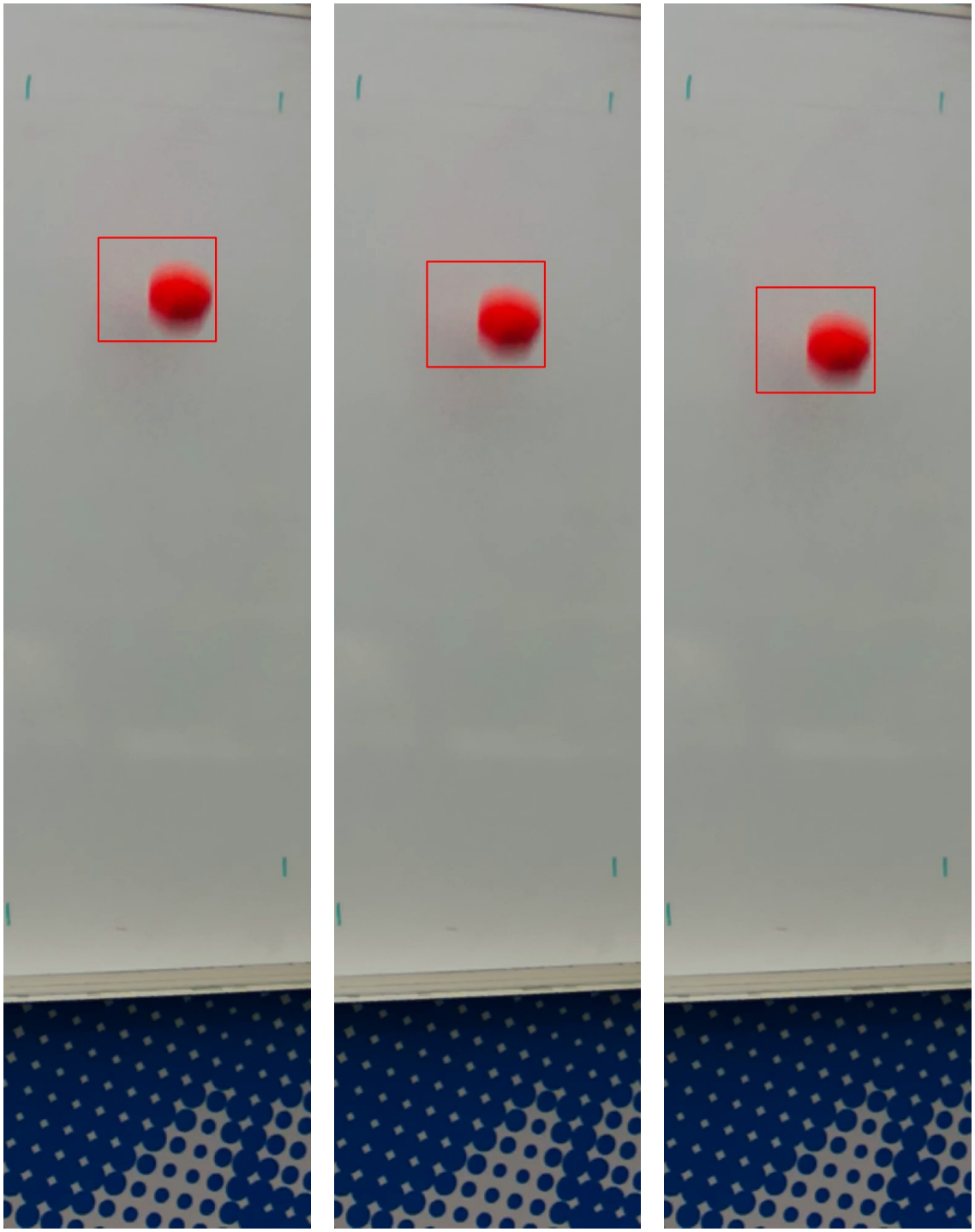


Figure D.1: 8mm lens, setup 1.1

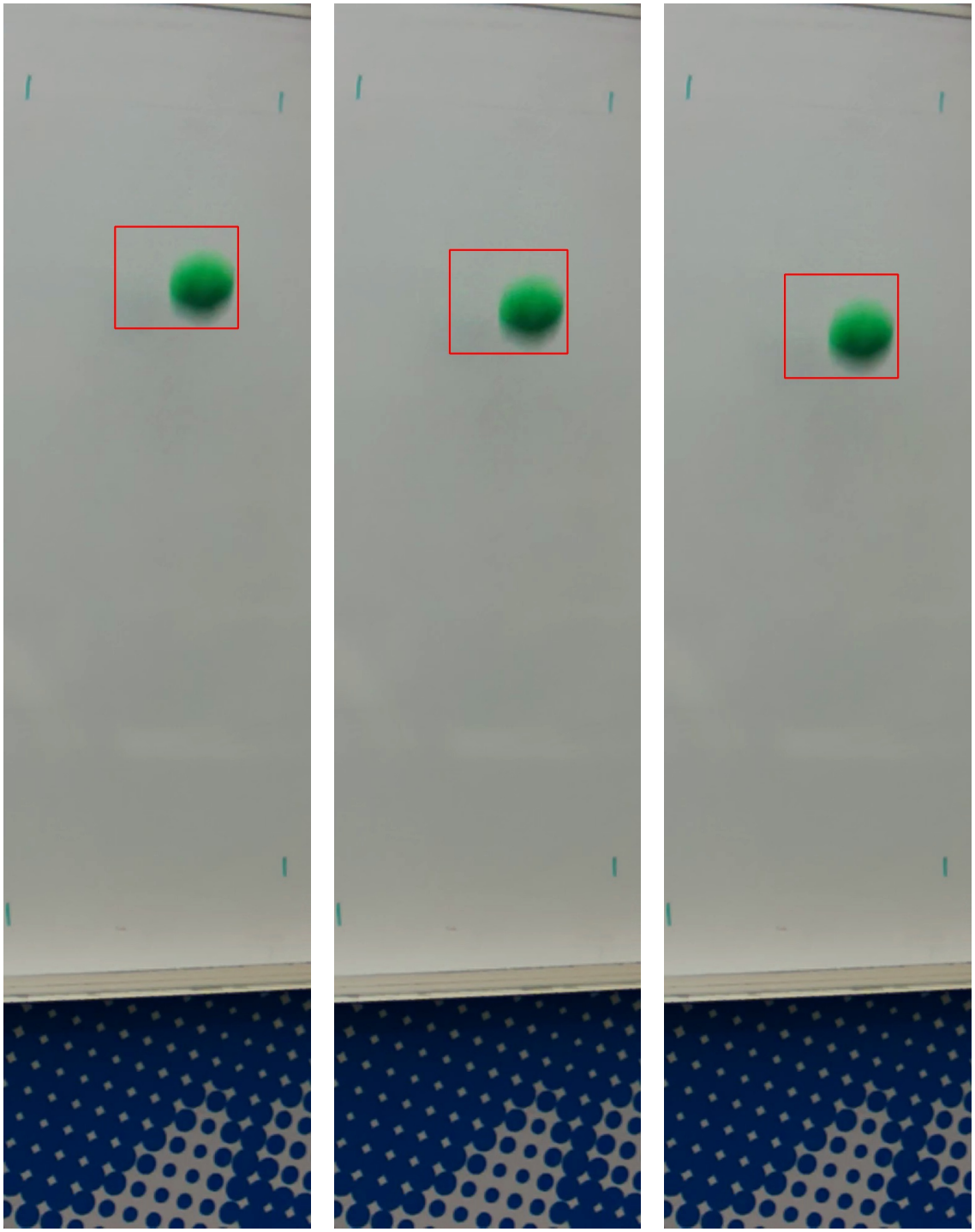


Figure D.2: 8mm lens, setup 1.2

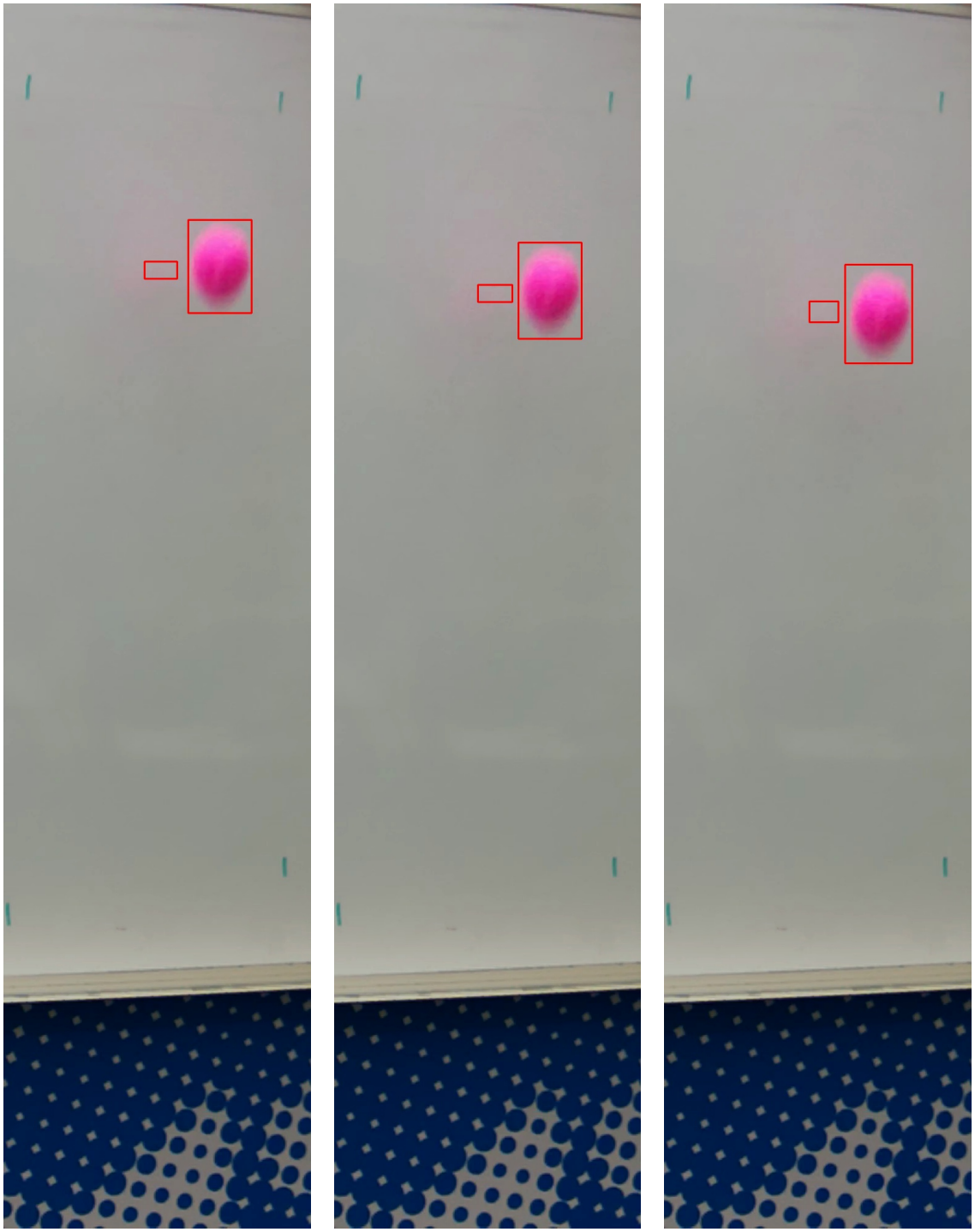


Figure D.3: 8mm lens, setup 1.3

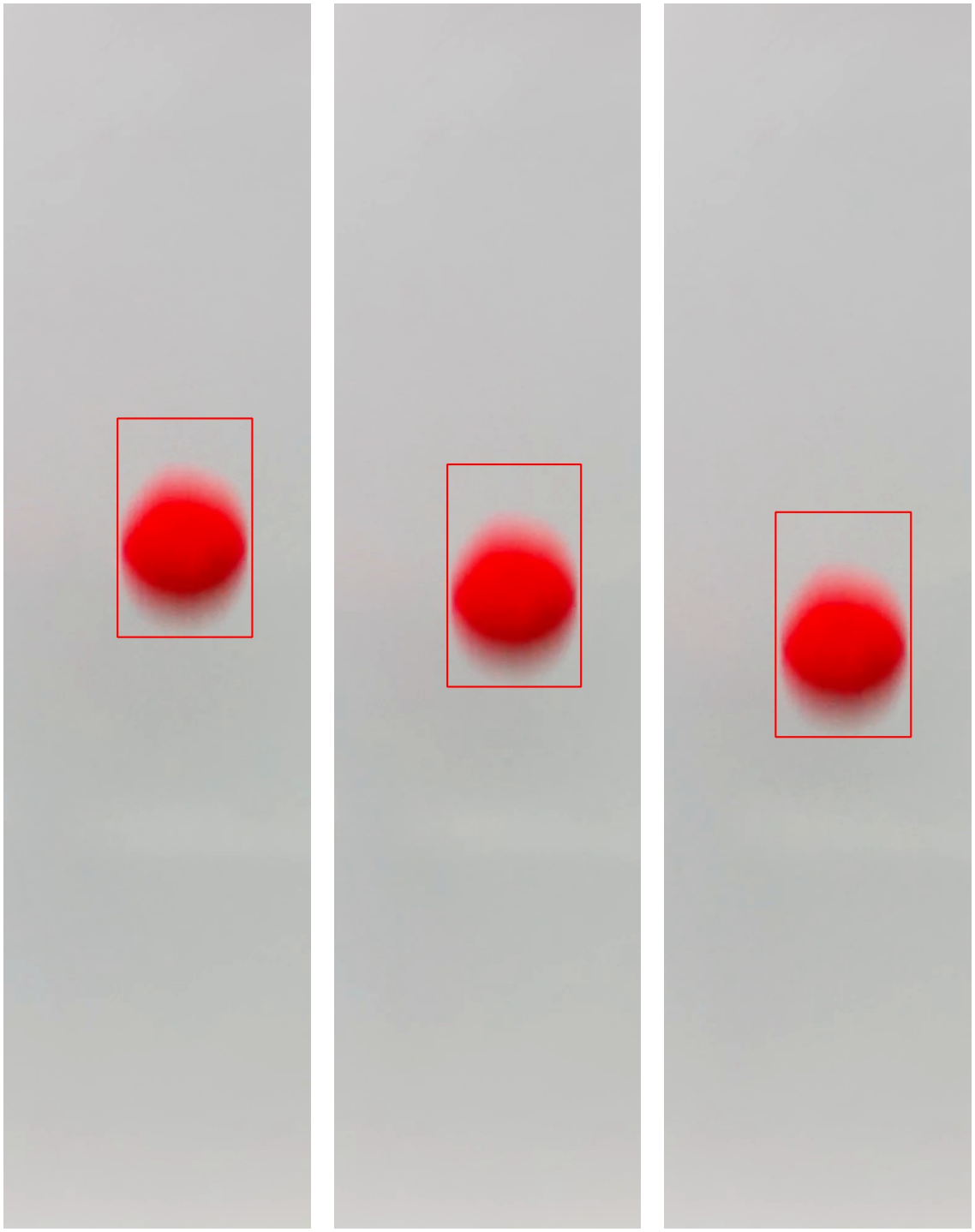


Figure D.4: 16mm lens, setup 1.1

Figure D.5: 16mm lens, setup 1.2

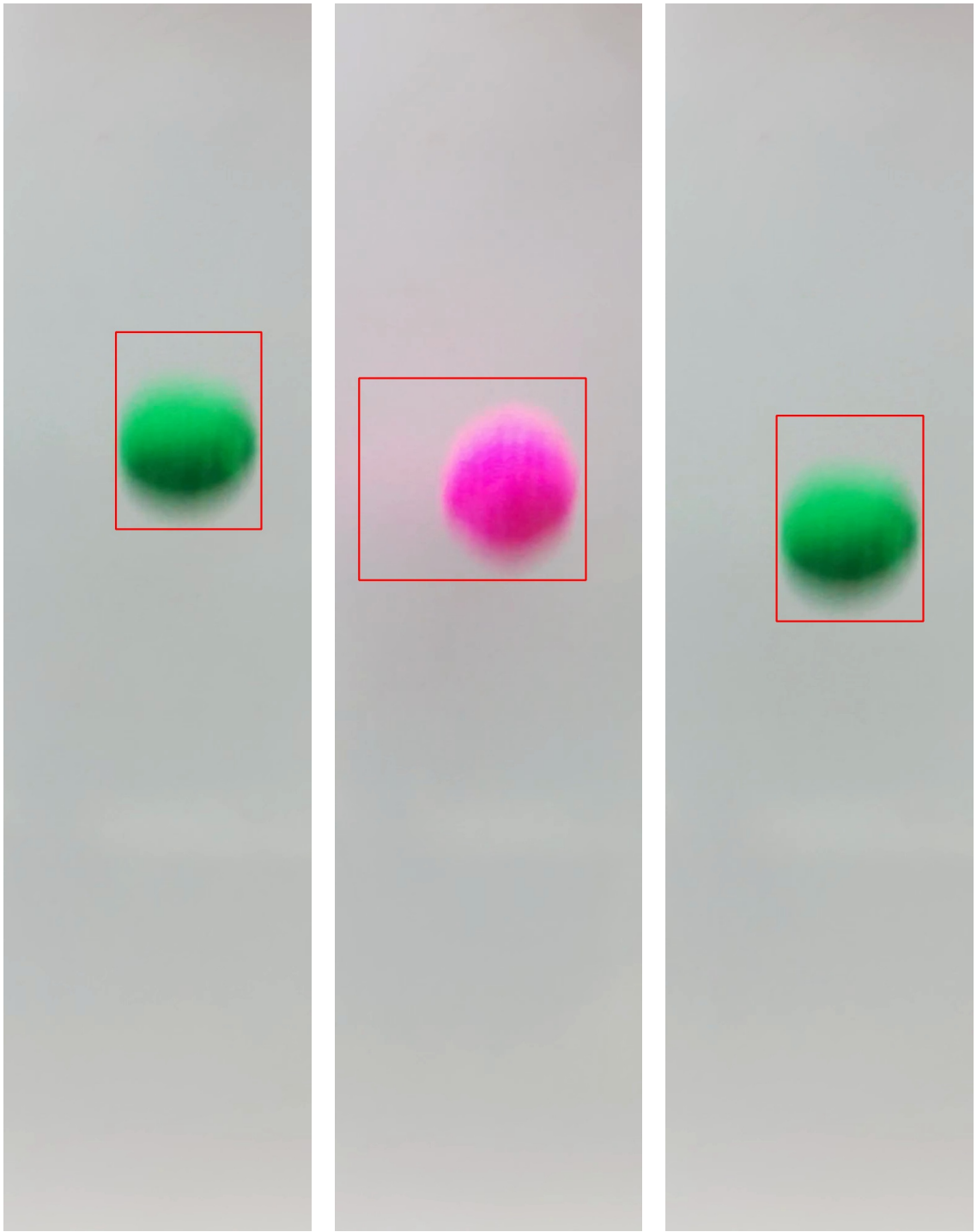


Figure D.6: 16mm lens, setup 1.3

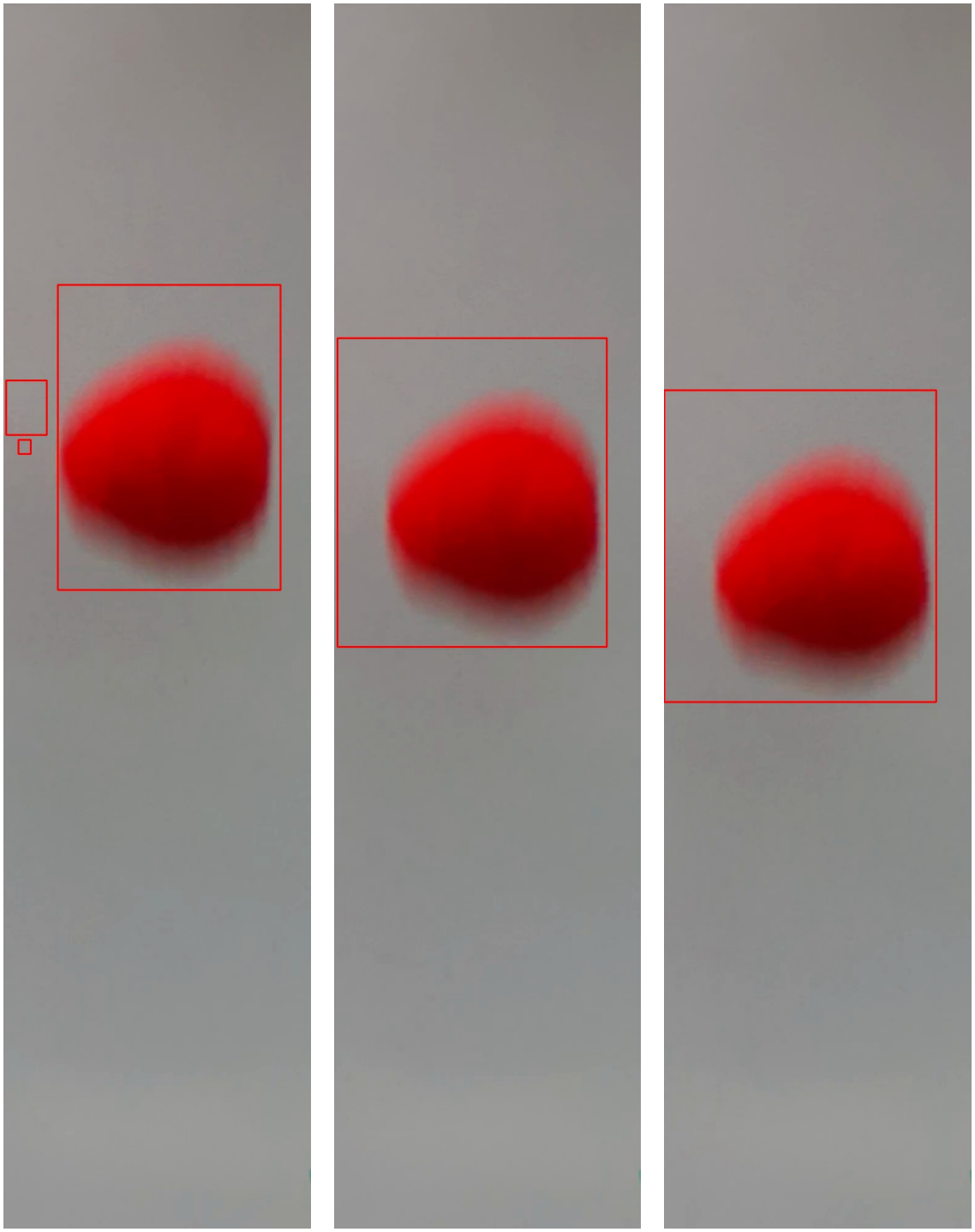


Figure D.7: 25mm lens, setup 1.1

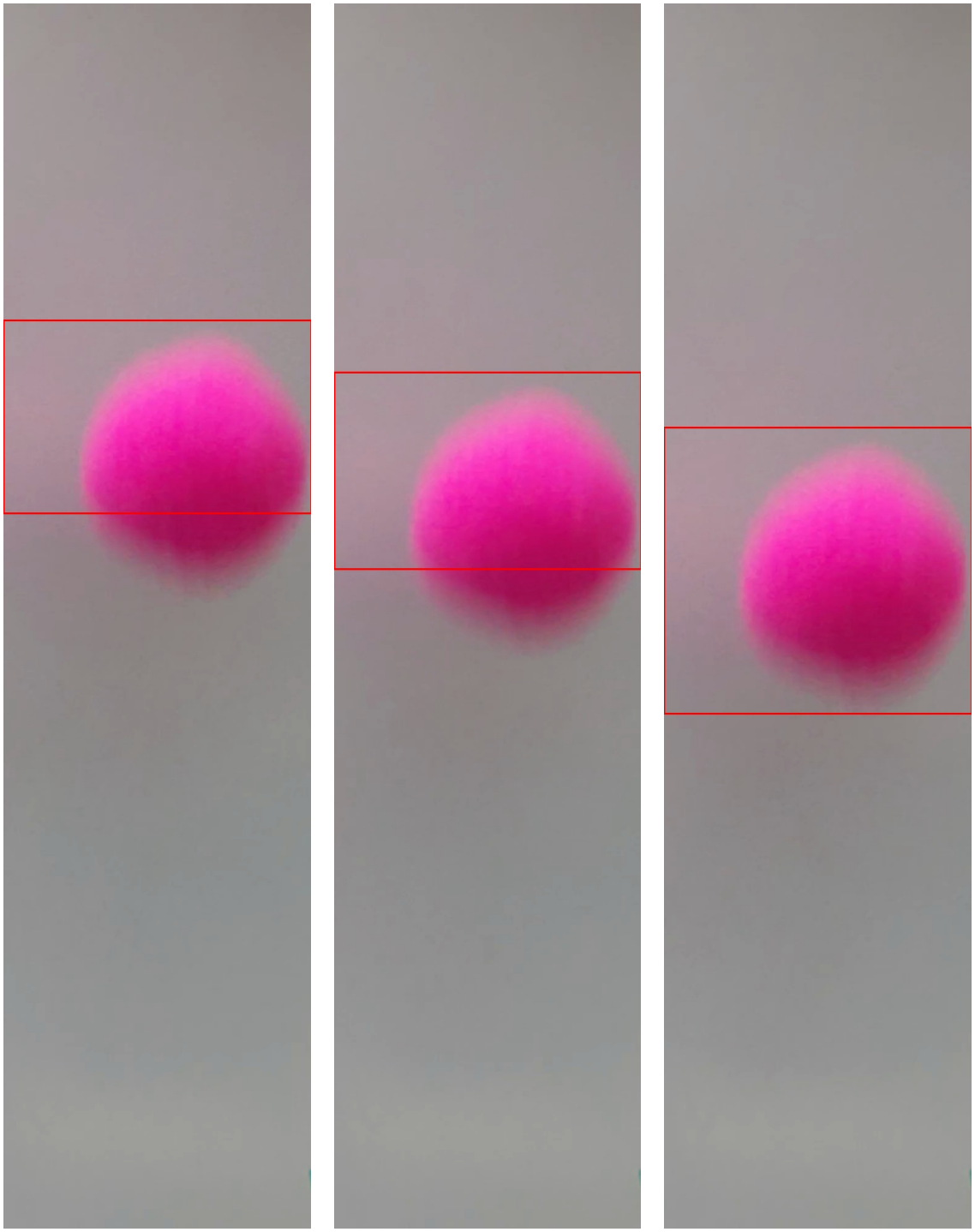


Figure D.8: 25mm lens, setup 1.2

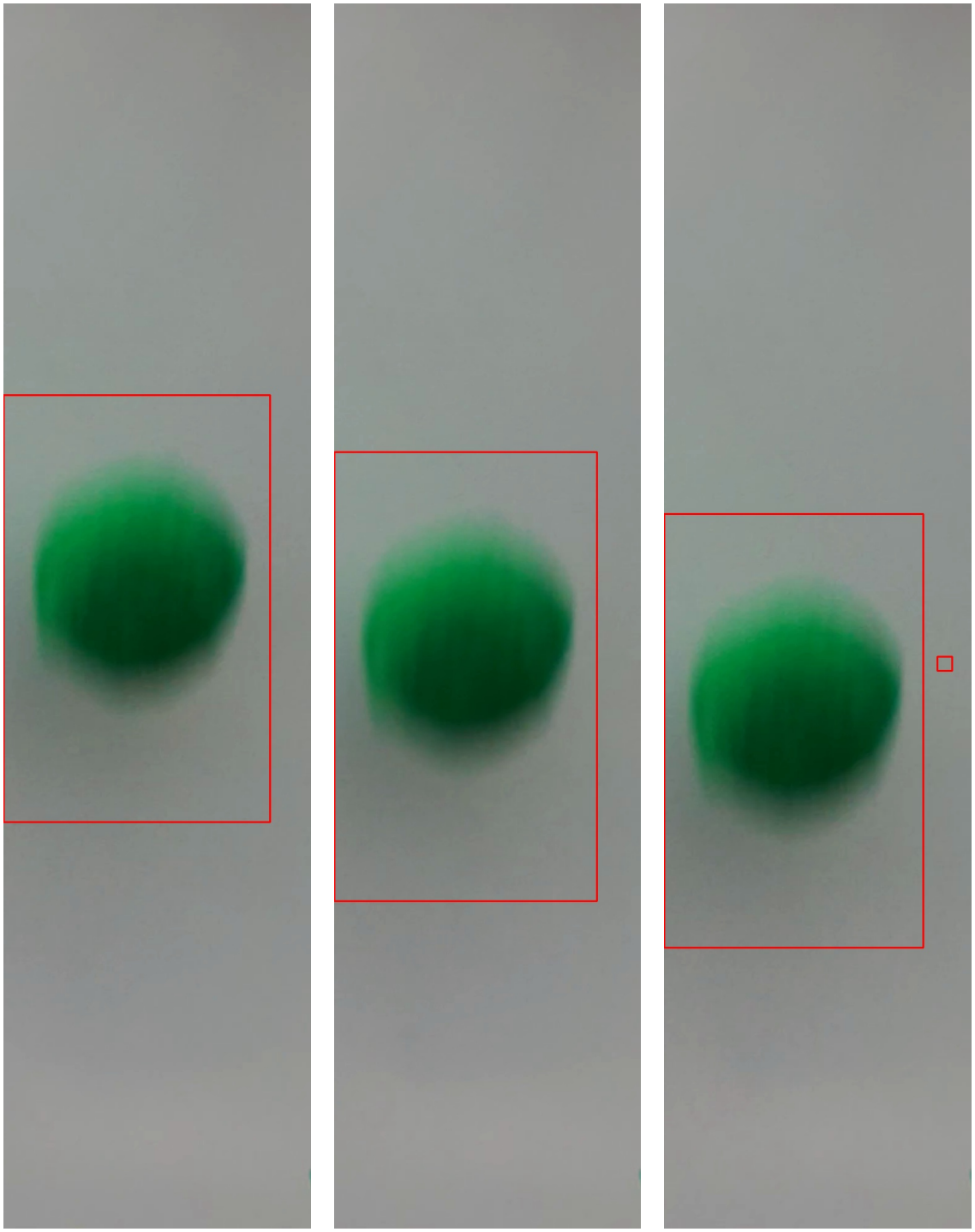


Figure D.9: 25mm lens, setup 1.3