Deep-Learning to Analyse Self-Assembly Processes

Thesis Bachelor Graduation Project Electrical Engineering

W.P. de Bruin, G. van Huizen, B.R. Metz, B.R. van Osch, E.D.N. de Rooij, S.L. Smilde

Delft University of Technology

TU Delft

Deep-Learning to Analyse Self-Assembly Processes

by

W.P. de Bruin, G. van Huizen, B.R. Metz, B.R. van Osch, E.D.N. de Rooij, S.L. Smilde

Student Name	Student Number
de Bruin	5341353
van Huizen	5343372
Metz	5295742
van Osch	5412099
de Rooij	5287642
Smilde	5375274

Supervisor:	L. Abelmann
Subject Experts:	J. Dauwels, M. Mastrangeli
Project Duration:	April 2024 - June 2024
Faculty:	Faculty of Electrical Engineering, Delft

TU Delft

Abstract

This report presents the design process of a project aimed at the automatic recognition of 3D structures formed by magnetic spheres in a turbulent water-filled cylinder. This field of research holds promise for future technologies, as macroscopic self-assembly might be the key to three-dimensional storage on chips. With the macroscopic setup, the microscopic self-assembly process is imitated. To automatically recognise a 3D structure, this report is divided into three subgroups that research the optimal test setup, develop an image processing program and create a deep learning model. A labelled result of the 3D formation will be outputted, all while limiting the data size, computation time and inaccuracies. The subgroup responsible for the setup and the underlying physics produces images of the magnetic spheres forming a structure in the test setup. The Image Processing subgroup extracts the properties of the spheres from the image. Finally, the subteam for deep learning, in combination with data management, gives the extracted properties as input to a neural network model, which determines the structure of the spheres. Each submodule has demonstrated successful functionality on its own. However, due to time constraints, a fully integrated system with high accuracy has not been achieved yet. Future work will involve expanding the dataset to enhance the robustness of the recognition algorithms.

Preface

This state-of-the-art project dives into the concept of macroscopic self-assembly by focusing on the automatic recognition of three-dimensional structures. The project is divided into three interconnected components: a module responsible for the experimental setup and physics, the image processing module and the deep learning module, supported by a data management part.

The design process has primarily involved reiterating design choices based on the program of requirements, planning and laying the groundwork for subsequent implementation and, eventually, acquiring experimental results and testing the algorithms.

We would like to express our gratitude to everyone who has contributed to this project. To Leon Abelmann, especially, thank you for your valuable guidance throughout this project. Your commitment to answering all our practical and theoretical questions has been of great value in what we deem a successful project. In particular, the Vaseline has saved us a lot of effort. We also extend our thanks to Justin Dauwels and Massimo Mastrangeli for their expertise and helpful insights on the matter. We are looking forward to any progress and discoveries that will emerge in this research field, and we are excited to see what the future holds.

Contents

1 Introduction

In the dynamic and rapidly advancing field of semiconductor manufacturing, a fascinating innovation emerges: chip self-assembly. Imagine a watery domain within a controlled chamber, where minute spheres, each harbouring a magnetic force, frolic in an orchestrated choreography. They connect, disconnect, and reassemble, forming intricate structures. Introducing the captivating realm of autonomous fabrication, where science meets spectacle, and every sphere tells a story.

1.1. Macroscopic Self-Assembly

The pursuit of enhanced computational power and miniaturization in semiconductor technology has led to a pressing need for innovative fabrication techniques to accommodate the demand for increased information density. Traditional 2D chip architectures are reaching their limits, with the scaling of device dimensions becoming increasingly challenging. While 3D chips offer a glimpse of promise, current methods are constrained. Relying primarily on stacking to achieve information density in the z-direction, will not suffice as this remains relatively low compared to the x and y directions.

This limitation presents an obstacle in the search for a more compact and efficient chip design. However, with the prospect of chip self-assembly, possibilities arise. By utilizing the principles of autonomous fabrication, self-assembly holds the potential to overcome the constraints of traditional manufacturing techniques, offering a pathway towards unlocking the full potential of 3D chip architectures. Self-assembly enables the seamless integration of components in three dimensions, thereby achieving a more equitable distribution of information density across all spatial dimensions [1].

Microscopic self-assembly is challenging to observe due to the minuscule size of the particles and the rapid time scales involved. However, to optimize the self-assembly process and minimize crystal defects, it is crucial to gain insights into all interactions. While computer simulations offer a potential solution, this is computationally expensive and impractical for large systems. An alternative approach involves using a macroscopic self-assembly reactor to make simulations. This can significantly enhance visibility and allow for the detailed study of self-assembly dynamics at a larger scale.

To best mimic the microscopic scenarios, it is important to replicate the interplay between the disruptive forces and the assembling forces that are present. By analysing the formations formed in the macroscopic set-up, researchers gain insights into the mechanisms of self-assembly, observe the formation and evolution of structures in real-time, and identify factors that lead to defects. This approach can complement computer simulations and experimental studies [12].

The ultimate aim is to pave the way for the realization of next-generation chip architectures. The next step in realising this is to automate the analysis of formations that are composed in the self-assembly reactor to gain a better understanding of the automation principles.

1.1.1. Motivation

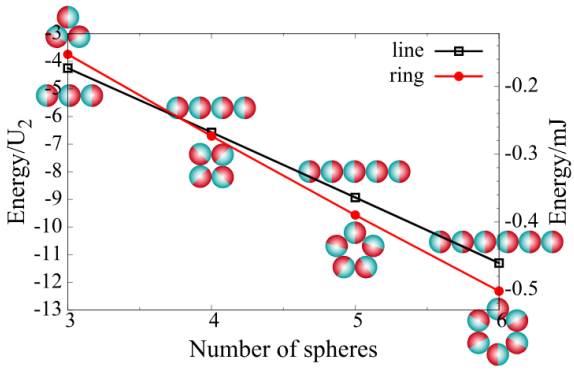


Figure 1.1: (Normalized) energy for formations of different number of spheres

One example of the research that can be conducted with the constructed product is illustrated by analyzing the energy levels of different formations of spheres. Figure 1.1 shows the energy levels for ring and line formations of varying numbers of spheres, as calculated in [14]. The energy is normalized to the energy of two dipole spheres. The formulas for the energies of both formations are:

$$u_N^{\text{line}} = -\frac{2}{N}\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\frac{1}{(j-i)^3} \tag{1.1}$$

$$u_N^{\text{ring}} = -\frac{1}{4}\sin^3\left(\frac{\pi}{N}\right)\sum_{k=1}^{N-1}\frac{3+\cos\left(\frac{2\pi k}{N}\right)}{\sin^3\left(\frac{\pi k}{N}\right)} \tag{1.2}$$

Particles tend to go to a state of minimum energy. For instance, when the experiment involves six spheres, Figure 1.1 indicates that the probability of a line formation occurring is nearly zero. This raises the question of the necessity of our product for experiments with six spheres.

However, the experiment also incorporates two coils placed on either side, capable of generating a homogeneous field of up to 10 mT. As the external magnetic field increases, observations indicate that ring formations will eventually break into a line formation aligned with the magnetic field. The energy of the external field per sphere is given by:

$$U_{\text{ext}} = -\mathbf{m}\cdot\mathbf{B}_{\text{ext}} \tag{1.3}$$

where **m** is the magnetic moment of the magnets inside the spheres. Since the external energy is the dot product of the magnetic moment of the spheres and the external field, there is no energy contribution to a perfectly aligned ring formation. However, there is (negative) energy added to the line formation. Assuming the line is perfectly aligned with the field, the total external energy is the product of the magnetic moment of the spheres, the external field, and the number of spheres. In Figure 1.2, the same energy levels from Figure 1.1 are plotted. Additionally, a green line representing the total energy of the line formation under a specific external field is added. It is evident that for six spheres, an external field of 0.16 mT is required to equalize the total energy of line and ring formations. Investigating the external field required to overcome the energy barrier and transition from a ring to a line formation would be intriguing. Automatic recognition of ring and line formations would be highly beneficial in such research.

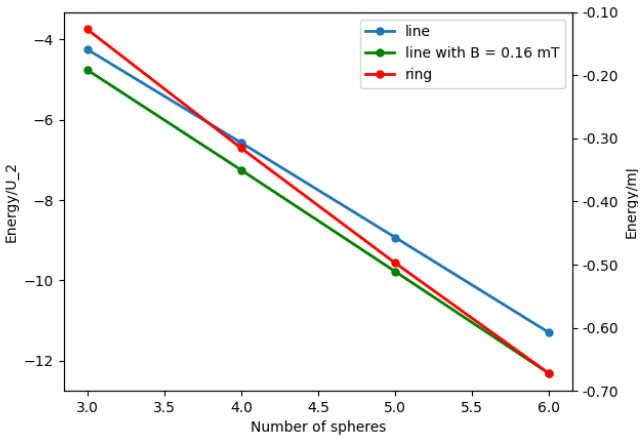


Figure 1.2: Having an external field of 0.16 mT results in ring and line formations with an equal total amount of energy

1.2. Research Objective

This study aims to investigate and realise a highly accurate system that automates the analysis of magnetic dipole spheres that form structures in a turbulence-driven fluid.

Due to the magnetic interactions between the spheres and the turbulence, these spheres may form chains or rings with a certain probability. Distinguishing between these formations is crucial for calculating their respective probabilities and advancing research. Our research question is:

"How do we automatically recognize 3D formations of magnetic spheres in line and ring formations?"

1.3. Experimental Set-Up

The setup consisted of spherical components, each containing a magnet. These were suspended in a water-filled tank, featuring a steady upward flow, enabling buoyancy of the spheres and allowing them to interact and form structures freely. The magnetic properties of the spheres facilitate their connection when in close proximity.

The setup can use up to six spheres for structure formation analyses. This limitation arises from the dimensions of the tank and the spheres. To facilitate individual analysis, each sphere is assigned a unique colour, enabling one to track and analyse their behaviour independently.

Furthermore, the setup includes two large conductors attached to the tank, capable of inducing a controlled magnetic field of up to 10 mT. This external magnetic field serves as a tunable parameter, influencing the conditions under which the spheres connect and form structures. By modulating the strength and orientation of the magnetic field, a systematic exploration of its effects on the self-assembly process and the resulting structures formed by the spheres can be performed.

1.4. Outline of this Thesis

The experiment is composed of a combination of experimental observations, simulations, and automated analysis techniques. This thesis explains and describes the experiment and is structured in the following way:

Chapter 2 outlines the top-level requirements that must be considered and adhered to throughout every stage of the design process. It describes the goals and constraints of the project, ensuring that all subsequent decisions align with the overarching objectives.

Chapter 3 explores the different approaches to fulfilling these top-level requirements. It evaluates various design options, discussing their merits and drawbacks to identify the best possible solution.

Chapter 4 provides a design overview of the selected implementation. It details the chosen architecture, components and workflow.

Chapter 5 delves into the setup and physics that are required for the project. It covers the hardware configurations, environmental conditions and physical interactions of the setup.

Chapter 6 focuses on the image processing techniques utilized in the project. The algorithms and methodologies used to process and analyze the visual data are described.

Chapter 7 examines the deep learning models integrated into the system. The chapter discusses the selection, training and optimization of the models.

Chapter 8 addresses the strategies for managing data throughout the project's pipeline. It covers data collection and storage, ensuring data integrity and accessibility are maintained.

Chapter 9 brings all system components together, describing how they interact and function as a unit.

Chapter 10 provides a critical analysis of the results and findings from the implementation and integration stages. It discusses the project's successes, limitations, and potential areas for improvement, offering insights into the effectiveness of the chosen design and implementation strategies.

Chapter 11 summarizes the project's achievements and lessons learned. It reflects on the overall process, highlighting key takeaways and suggesting directions for future research and development. This chapter closes the document by reaffirming the project's contributions to the field and its potential impact.

2

Top-Level Requirements

The goal of this project is to determine a structure at several consecutive moments in time automatically. The idea for achieving this goal is that for each time instance, a physical or digital system can recognize how each sphere is connected to another sphere. From this, the structure is determined and subsequently, the information about the structure is stored and analyzed.

To measure whether the developed design method serves the described goal, the method should adhere to the Program of Requirements(PoR). During the design process, the requirements were iterated and design choices were made based on the program of requirements. This chapter describes the top-level requirements. The subgroup-specific requirements are described in the chapter of each subgroup.

2.1. Practical Requirements

To ensure that the project is practically implementable, the development of the algorithm must adhere to practical requirements.

1.1 **Time Frame:** The implementation of the project must be feasible in a time frame of three weeks.

1.2 **Costs:** The total cost budget of this project is €2000.

1.3 **Setup Compatibility:** The measurement method must be practically feasible and compatible with the current setup without necessitating any modifications.

1.4 **Portability:** The developed automated system must be portable. It should be designed such that the transportation of the system is convenient.

1.5 **Accessibility:** The system should be easy and remotely accessible.

2.2. System Requirements

The system requirements specify the criteria that define the necessary software, hardware and environmental conditions for the developed system to function according to the project goal.

2.1 **Training Time Efficiency:** Training of the automated recognition module should take up less time than the manual evaluation of the spheres' formations.

2.2 **Data Storage Size:** The maximum allowable size for long-term data collected from a single experimental trial is 500 MB.

2.3 **Reproducible Experiment:** The measurement methods must allow for reproducible experiments to ensure consistent results.

2.3. Functional Requirements

The functional requirements are used to specify the functionalities and features that the developed design must perform to serve the overall aim of the project.

3.1 **Formation Recognition:** The method must be able to distinguish between a ring, a line and an undefined formation of a variable number of spheres.

3.2 **Automated Process:** The developed algorithm/method must yield an automated execution process.

3.3 **Accuracy:** The overall system must have an accuracy of at least 95%, meaning that false classifications should not take up more than 5% of all the results.

3.4 **Generalisability:** The developed method must be generalisable to any structure and any shape of the spheres in the structure.

3.5 **Validatability:** The designed overall system must be capable of being validated to show the reliability and usability of the system.

3.6 **Experiment Duration:** The test setup must be able to run the experiment for a duration of 12 to 24 hours. Also, the developed method must be able to handle and store the corresponding amount of data.

3.7 **Computation Time:** The computation time of the automatised recognition module should not take more than 1 hour past the experiment duration.

2.4. Most Relevant Requirements

In summary, the seven most relevant top-level requirements will be iterated underneath and in the Conclusion, Chapter 11, solutions to these major requirements will be presented:

2.1 **Training Time Efficiency,**

2.2 **Data Storage Size,**

3.1 **Formation Recognition,**

3.2 **Automated Process,**

3.3 **Accuracy,**

3.5 **Validatability,**

3.7 **Computation Time.**

3

Top-Level Design Choices

As with any robust project design pipeline, various ideas, both effective and ineffective, were explored, leading to the adoption or rejection of these strategies. At the outset of the project, two primary strategies were assessed: a hardware solution and a software solution. Subsequently, sub-strategies were devised and explored. Ultimately, one solution triumphed (see Figure 3.1 for an overview of the decision tree).

3.1. Hardware

Within the hardware approach, three distinct options were considered: the implementation of reed switches, Hall effect sensors or high-frequency EM sensors.

3.1.1. Reed Switches

Reed switches, which are sensitive to magnetic fields, close to allow current flow when two opposing spheres come into contact due to their magnetic properties. However, since the experiment uses an external field to manipulate sphere formations, the effective operational range of the reed switches is significantly limited. They must resist closure induced by the external field from the coils while remaining sensitive enough to close in response to the magnetic field produced by an adjacent sphere. This dual demand presents a challenge in differentiating between the impact of the external field and the proximity of another sphere [7].

3.1.2. Hall Effect Sensors

Hall effect sensors detect magnetic fields and convert these into electrical signals by creating a voltage differential across a conductor exposed to a magnetic field. They require continuous power, making them less suitable for power-sensitive applications compared to more energy-efficient reed switches. Additionally, Hall effect sensors are also sensitive to interference from the external fields of coils. Therefore, in this power-sensitive setup, reed switches would be a better alternative [7].

3.1.3. High-Frequency EM Sensors

High-frequency EM sensors were identified as the most promising solution among the three options. This category includes sensors employing Bluetooth Low Energy (BLE), which offers a practical means to detect connectivity in compact spheres. By calibrating the attenuation of EM waves through the medium, it becomes possible to sense whether adjacent spheres are in contact. This solution, intended to identify sphere interactions accurately, remains unaffected by the experiment's conditions and can efficiently collect the required data. However, its potential success was considered lower due to the experiment's size constraints and the challenges associated with accessing the data, both of which significantly influenced the decision-making process regarding all three solutions.

3.2. Software

Within the software solutions, three different strategies are considered: image processing, deep learning and a hybrid approach. Each utilises image processing techniques to isolate individual spheres.

3.2.1. Image Processing

Image processing, although inadequately described by its title, involves two distinct strategies: template matching and 3D modelling.

Template matching

Template matching utilises a binary 2D circle template, where pixels that represent the circle are set to 1, and its diameter includes an additional margin to accommodate the maximum observed length in the structure. Initially, the input image undergoes binary processing, the template is then systematically moved across the image. If the sphere structure forms a complete circle, it fully overlaps the template, matching all its pixels, and indicating the area of the template circle. This process is repeated for images captured from the x, y, and z directions to evaluate the circularity of the structure. A consistent match in all directions suggests a circular structure; otherwise, it is classified as linear. However, this method is prone to incorrectly classifying larger structures with elliptical appearances that intersect the template boundary as linear, necessitating the use of multiple cameras.

3D modelling

3D modelling employs advanced algorithms designed to analyse images captured from different perspectives, typically from multiple cameras positioned at various angles (x, y, and z directions). These algorithms utilise principles of computer vision and geometric reconstruction to identify and localise the centres of spheres within a 3D coordinate system.

The process begins with the acquisition of 2D images of the sphere formations from different viewpoints. These images are then fed into the 3D modelling algorithm, which systematically analyses each image to identify and extract features corresponding to the spheres. The algorithm reconstructs a 3D representation of the scene, with each sphere represented by its centre point in the 3D coordinate system. Once the centres of the spheres have been accurately localised in 3D space, the algorithm can analyse their spatial arrangement to determine whether they form a circular or linear configuration. This analysis may involve geometric computations, such as measuring distances between sphere centres and assessing their alignment along different axes.

3.2.2. Deep Learning

The deep learning approach included two strategies: formation tracking and connectivity analysis.

Formation tracking

Formation tracking involves analysing entire sphere formations to distinguish between linear and circular configurations. This model requires training with a comprehensive dataset of all possible sphere and formation combinations, which can be captured using a single camera. Therefore, it demands a large dataset, especially if the number of spheres varies significantly, and entails extensive training time due to the complexity of handling large arrays of RGB data.

Connectivity analysis

The connectivity analysis approach identifies connections between spheres to infer the formation type. For instance, if three spheres are linked by two connections, it suggests a linear formation, whereas three connections indicate a circular formation. This method is less influenced by the number of spheres, substantially reducing the dataset size as fewer combinations need to be learned. Its primary focus is on detecting whether a link exists between any two spheres, a factor that remains constant regardless of the number of spheres. However, despite its potential, relying solely on image data has disadvantages.

3.2.3. Hybrid Model

The third alternative is the use of a hybrid model. Image processing methods would be employed to extract radius and centre information from the sphere formations, which would then serve as input for the deep learning model.

3.2.4. Incomplete Data Handling

Given that not all spheres may be visible in a single image, it's crucial to devise a strategy for handling incomplete data. Two approaches were considered: deletion and insertion of null values.

Deletion

The *"deleting"* strategy utilizes a visibility algorithm to evaluate the accuracy of radius predictions. This algorithm applies a threshold to determine the visibility coefficient of each sphere. If a sphere's visibility falls below the threshold, the corresponding set of images — whether from one or multiple cameras — is eliminated.

However, this approach may result in the loss of a significant number of images, particularly if stringent visibility standards are applied based on the number of cameras used. For instance, with two cameras, if one image lacks sufficient visibility, both associated images are discarded. Consequently, three out of four possible image combinations may be deleted. Historical data indicates that the camera positioned in the x-direction typically has a higher likelihood of failing to capture certain spheres, thereby increasing the potential for image loss under this strategy.

Insert Null

The *"insert null"* strategy also employs a visibility algorithm to evaluate the accuracy of radius predictions. A threshold is applied to determine whether a sphere's visibility coefficient falls below a certain level. If it does, all parameters related to that sphere are set to zero in the dataset.

This approach enables the retention of images in which some spheres are not visible while still training the model to handle missing or incomplete data without bias. By incorporating data where certain spheres are absent, the model learns to perform in scenarios where visibility is compromised. This method mitigates the excessive discarding of data, potentially enhancing the overall effectiveness and reliability of the system, especially in diverse operational environments.

However, it's important to note that this approach may influence the accuracy of predictions in scenarios with good visibility, as the model is trained on data with artificially inserted null values.

The choice between these two strategies depends on the volume of data that can be collected, which is influenced by the attempt frequency (f_A); the frequency of formation changes. Given the constraints of processing and uploading images via a Raspberry Pi, the setup requires careful consideration of data flow.

When the attempt frequency is relatively low, the sampling frequency can exceed the Nyquist frequency. In this scenario, even with a relatively low sampling rate, the data volumes remain manageable, allowing for a significant portion of data to be deleted without risking the loss of transitional information.

Conversely, in situations where the frequency of formation changes is high, the system generates a substantial amount of data. Here, the sampling frequency must be at least twice that of the formation change frequency to ensure no transitions are missed. In such cases, to avoid missing critical transitions, the algorithm may have to utilize data with poorer visibility. Balancing this trade-off is crucial for maintaining the integrity of the experiment's outcomes.

3.3. Design Comparison

To determine the most suitable implementation, whether one or both, a comparison between the alternatives will be done using the described top-level requirements from Chapter 2.

3.3.1. Hardware vs Software

The comparison made for the top-level design choice is illustrated in Table 3.1. This comparison has been made early on in the project, which is why the requirements in the Table do not fully match the final requirements from Chapter 2. While specific weights for each category are not necessary, it's clear that the image-based approach holds an advantage. The top four categories (Sphere Size, Generalisability, Measurement Neutrality and Experiment Duration) make the most significant difference in the design selection; hence, their scores will be explained.

Due to the installation of a sensor within the sphere, the size of the sphere will increase, which is a significant disadvantage for sensor-based approaches. Generalisability includes various aspects such as sphere size, variability, different structures, and formations. Because sensor-based methods only establish a connection between spheres, this method cannot make a difference between different formations like a cube. This presents a challenge for image processing as well, but one that is likely manageable.

Sensor-based experiments may be sensitive to external electric field interference, which could influence the experiment results. Additionally, the energy consumption of sensors poses a challenge, especially when running experiments for extended periods such as 12-24 hours. In contrast, image-based methods do not have these disadvantages in either of these categories.

Table 3.1: Hardware vs. software comparison based on top-level requirements

Top-level requirements	**Image-based**	**Sensor-based**
Sphere Size	5	3
Generalisability	5	3
Measurement Neutrality	4	2
Experiment Duration	4	2
Accessibility	4	3
Costs	4	4
Setup Stability	4	3
Data Storage Size	4	5
Data Variability	5	3
Synchronized Data Collection	3	4
Future Expansion Prospects	4	2
Validatability	4	5
Total weight	**50**	**39**
Weighted average	**4,2**	**3,3**

3.3.2. Image Processing vs Neural Network vs Hybrid

In Table 3.2, clearly one method results in the best option. It distinguishes itself from the storage size, as only a file with sphere arrays needs to be saved, instead of images with the other options. Secondly, this method has good generalisable options, as this can be trained with the coordinates and radii of the spheres. Also, an educated guess on the success rate of the method has been made, considering aspects like debugging availabilities, precision estimation and potential unforeseen challenges.

Thus, after considering all these different methods, Figure 3.1 depicts the design plan, showing all alternatives.

Table 3.2: Image processing vs neural network comparison based on image-based requirements

Image-based requirements	Weights	Image Processing	Neural Network	Hybrid
Structure Dependencies	*3*	3	2	3
Data Storage Size	*7*	1	2	5
Data Variability	*4*	5	3	4
Generalisability	*9*	4	2	4
Future Expansion Prospects	*2*	3	2	4
Synchronized Data Collection	*8*	2	5	5
Accessibility	*7*	1	4	3
Validatability	*10*	5	3	4
Portability	*7*	4	4	4
Measurement Neutrality	*10*	2	4	4
Time Frame	*4*	3	4	5
Setup Stability	*5*	1	4	4
Training Time Efficiency	*4*	4	1	2
Chance of Success	*10*	3	2	4
Computational Time	*3*	2	4	4
Costs	*10*	5	5	5
Traceability	*7*	4	2	3
Total weight	***110***	**346**	**356**	**444**
Weighted average		**3,15**	**3,24**	**4,04**
% difference with max		**28,3%**	**24,7%**	**0,0%**

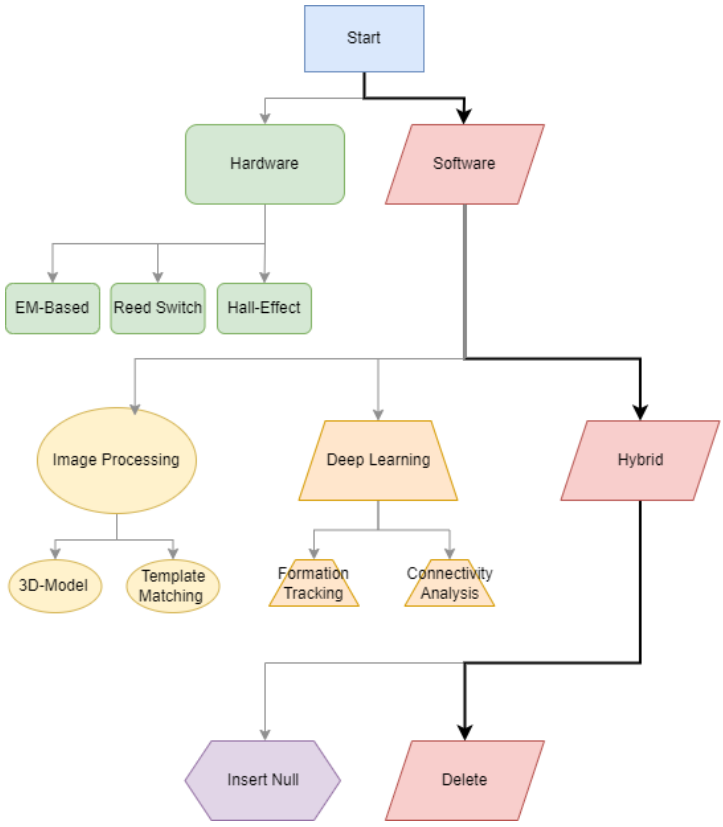


Figure 3.1: Alternative design decision tree

3.3.3. Language Selection

To ensure consistency in the software program, it was opted to select one language for script writing.

1.1 Python

1.2 Matlab

1.3 C++

Python was chosen as the primary language for development due to its extensive libraries, ease of use, and rapid prototyping capabilities. MATLAB and C++ were not selected because MATLAB, is less efficient for large-scale applications, and C++ requires more complex memory management and development time. The table substantiating this choice can be found in Appendix D.3, Table D.12.

4

Top-Level Implementation

This chapter provides an overview of the designed top-level implementation of the experimental setup and the analysis that automates the detection of formations of spherical objects using image processing and deep learning techniques. The implementation integrates various components, such as data acquisition, processing and analysis workflow.

4.1. System Overview

The system consists of several interconnected modules managed by different teams. The process begins with acquiring images at a certain sample frequency and ends with predicting the structure formed at specific time stamps. The primary components involved are the experimental setup, the image acquisition system, preprocessing and data extraction modules and the deep learning model for structure prediction.

The steps that lead to automatised recognition of 3D structures of magnetic spheres are as follows:

1. A camera takes images of the experiment and a separate device uploads them to a server.
2. Image processing recognises the individual spheres suspended in a cylinder filled with turbulent water.
3. The acquired data represents the position of the centre point and the radius of the spheres in a 2D plane.
4. The data is processed by inserting it into a neural network that is trained by (generated) labelled data.
5. Output is a predicted 3D structure, either: *Ring*, *Line* or *None*.

4.1.1. Experimental Setup and Physics Analysis

Camera Setup

One camera is implemented in the setup, with a possible expansion to two cameras, and is used to capture images at a predetermined sample frequency. The decision to use one camera comes from the Time Frame requirement from Section 2.1 as it saves time, and from the Setup Compatibility requirement (Section 2.2) so that it takes the existing setup into account. In addition, the camera is chosen such that it can recognise individual spheres to meet the Formation Recognition requirement (Section 2.3).

Sphere Design

The spheres must meet certain calculated size values to meet the requirements of Setup Compatibility. According to requirements Reproducible Experiment and Experiment Duration, the spheres should be designed in such a way that multiple experiments can be performed, having a duration of 12 to 24 hours. Therefore, the spheres themselves and the paint should be waterproof and not break apart in the water

flow. The training data is designed according to the requirement Generalisability, as the forced training formations can be easily reproduced and extended to more spheres.

4.1.2. Data Management

The acquired images are temporarily stored in a cloud. First as the primary data source for subsequent processing steps. Second, for post-inspection and to ensure no retake of the experiment has to be performed in case the process fails. The images from the cloud have to be manually deleted whenever the client does not need them anymore. These decisions can be traced back to the Data Storage Size requirement in Section 2.2 and the Validatability requirement in Section 2.3.

4.1.3. Preprocessing and Feature Extraction

- **Preprocessing:** The image quality enhancement and preparation for feature extraction (Accuracy requirement from Section 2.3).
- **Sphere Shape Extraction:** Individual spherical shapes are identified and extracted from the images, these are stored in separate files for further analysis.
- **Extrapolation and Coordinate Determination:** The edges of each sphere are extrapolated and the 2D coordinates of each sphere's centre and the radius are determined.
- **Visibility:** The ratio of the visible and extrapolated sphere (Accuracy requirement from Section 2.3).

4.1.4. Data Integration

The coordinates, radii and visibility of all identified spheres are compiled and saved into a single file, consolidating the data for easier access and analysis.

4.1.5. Deep Learning for Structure Prediction

- **Logic Implementation:** implementing the necessary logic to feed the extracted data into the deep learning model. Mainly, determining which images will be thrown away based on the visibility factor (Accuracy requirement from Section 2.3).
- **Model Training:** Deep learning is used to process the data, and the model is trained to predict the formation of the spheres (Formation Recognition requirement from Section 2.3).

4.1.6. Outcome and Analysis

The output of the deep learning model includes the predicted structure of the spheres: ring, line or none. This will be saved accordingly to the final formatted file, so data can be validated. This final format includes per image its spheres with their respective metadata (Validatability requirement from Section 2.3).

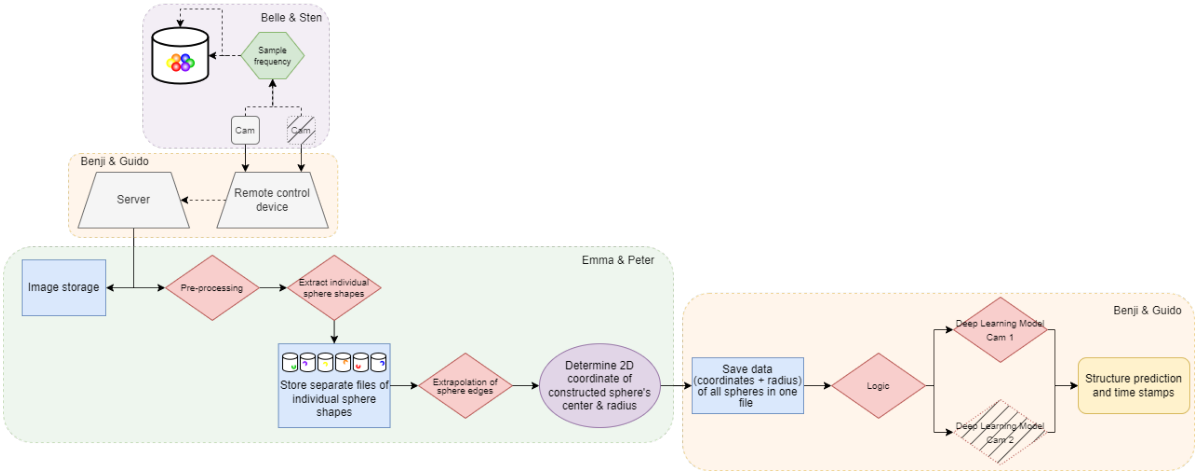


Figure 4.1: Pipeline of the top-level implementation (can also be found in Appendix E.78).

5

Experimental Setup & Physics

5.1. Introduction

The research question of the overall project is:

“How do we automatically recognise 3D formations of magnetic spheres in line and ring formations?”

This submodule is responsible for the experimental setup and the underlying physics principles. In this subdivision, experiments on the setup will supply data to be analysed and processed by the subsequent modules, Image Processing and Deep Learning. First, to be able to recognise 3D formations of magnetic spheres, the spheres should be designed and produced. The Image Processing and Deep Learning modules require clear images of the sphere formations, with the right illumination and enough pixels to recognise the different spheres. Besides, the Deep Learning model needs to be trained on the three optional results: *ring*, *line* and *none*. Therefore, a training dataset labelled with *ring*, *line* and *none* formations needs to be produced. Therefore the subgroup-specific research question will be formulated as follows:

”How can we supply the right data to the Image Processing and Deep Learning modules that will enable them to recognise the formation of the 3D structures?”

The experimental setup is shown in Figure 5.1. It consists of an orange-coloured container on the ground filled with water. A water pump produces a constant and adjustable upward water flow in the cylinder. Inside this cylinder, a cone (see Figure 5.2) is located in which the spheres move. An ”aquarium” is stationed around the cylinder to oppose the reflection of the light. On top of that, there is also a narrow orange cylinder placed 65 cm above the bottom of the cone, see the simulation in Figure 5.3. A flashlight is placed in the cylinder, directing light on the setup from above.

This chapter will cover all the steps that are needed to produce clear pictures of the spheres. It will start with the selection of the camera and lighting source. Furthermore, the designing and assembling of the magnetic spheres will be dissected, including their colours. Last but not least, decisions will be made on the generation of the training data: which formations should be trained and how are these being produced?

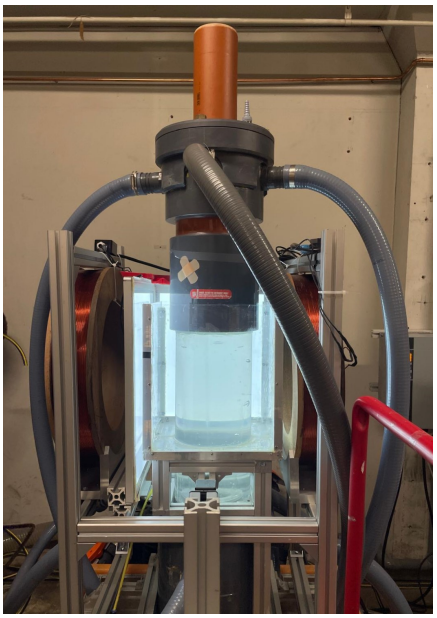


Figure 5.1: Setup as seen from the front

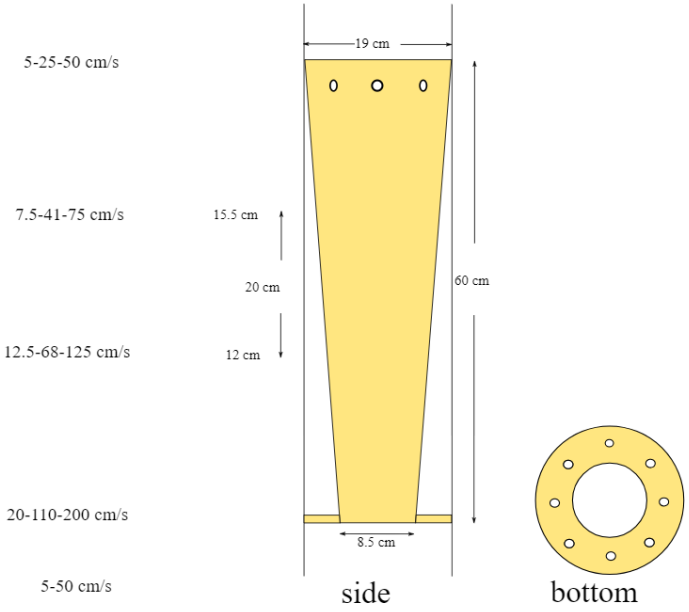


Figure 5.2: Cone in which the spheres are in motion

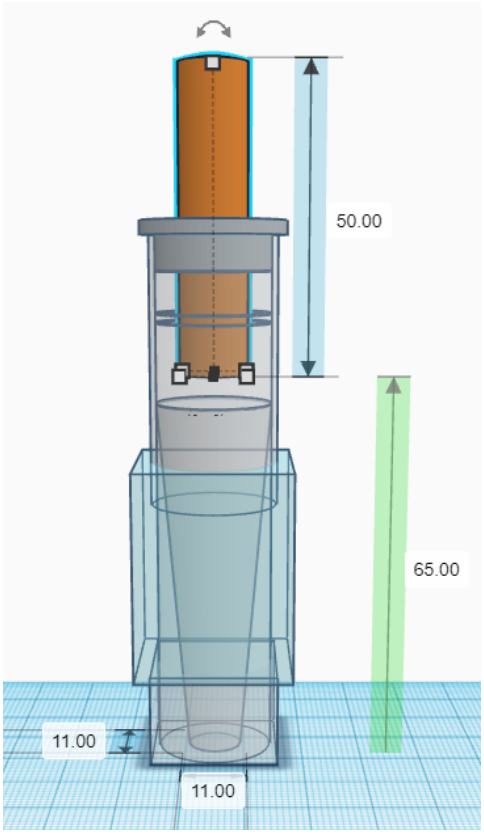


Figure 5.3: Undetailed simulation of the setup, including the cone within and cylinder above

5.2. Program of Requirements

This submodule is responsible for the experimental setup and the underlying physics principles. Specific requirements for this subgroup based on the top-level requirements have been formulated to en-

sure clear objectives and guide design choices. These requirements are divided into three categories, namely sphere design, setup and training data.

5.2.1. Sphere Design

Sphere design requirements describe the conditions the spheres and the magnets inside are limited to.

1.1 **Sphere Size:** To accommodate a line formation of six spheres within the water tube with a minimum diameter of 12 cm, each sphere's diameter must be at most 2 cm. This requirement comes from the Setup Compatibility requirement.

1.2 **Mass Density Sphere:** The spheres should have a mass density, dependent on their weight and size, such that they will 'float' in the area of sight of the cone. This is to ensure the Formation Recognition requirement is met.

1.3 **Energy Barrier:** The energy barrier between the magnets in two contacting spheres should be 100 μJ to ensure proper interaction and stability within the experimental setup. This requirement is derived from the accuracy objective.

1.4 **Colour Selection:** The Image Processing module (Chapter 6) prefers unique colours per sphere and we will work with a maximum of 6 spheres. So 6 colours should be selected that can be distinguished from each other by the colour thresholding of Image Processing to allow for Formation Recognition and Training Time Efficiency.

1.5 **Paint Durability:** The paint on the spheres must remain intact and undamaged for the entire duration of the experiment (so this comes from the requirement Experiment Duration), withstanding long exposure to water and any other conditions presented by the setup.

5.2.2. Setup Design

The setup requirements involve optimal equipment placement, consistent environmental conditions and system stability to achieve accurate experimental results.

2.1 **Costs:** The budget for the experimental setup and hardware part is 1500 euros. This comes from the top-level Costs requirement and is determined such that the majority of the budget is available for this module.

2.2 **Camera Position:** The camera should be positioned to provide a clear and unobstructed view of the spheres for Image Processing to allow for Formation Recognition. From this module follows a position directly above the cylinder, looking down.

2.3 **Camera Size:** The physical dimensions of the camera are limited by the 11 cm diameter cylinder on top of the experiment in which the camera is positioned. This requirement is derived from the top-level requirement Setup Compatibility.

2.4 **Camera Resolution:** The pixel rate of the camera should be sufficient so that the spheres at a maximum distance (at the bottom of the cone) depict a minimum of 10 pixels in radius, but not too high that the size of the acquired data becomes too large to store. This is determined such that requirements for Formation Recognition, Accuracy and Data Storage Size are met.

2.5 **Camera Field of View:** The camera should have a narrow enough field of view to minimise noise from capturing areas outside the water cylinder, thereby maximising the resolution and focus on the relevant experimental area. This requirement is derived from the same top-level requirements as Camera Resolution.

2.6 **Camera Colour Accuracy:** The camera must have high colour accuracy to clearly differentiate and recognise colours, this is to improve the overall accuracy and recognition.

2.7 **Lighting Position:** The light should be directed from above, in line with the camera's perspective, to minimise shadows and reflections that could interfere with sphere detection. This comes from the requirements Formation Recognition and Accuracy.

2.8 **Lighting Size:** The size of the light source is limited by the same cylinder as the camera, the lighting must fit together with the camera in an 11 cm diameter cylinder, according to requirement Setup Compatibility.

2.9 **Lighting Brightness:** The right luminosity is crucial for image recognition and should be chosen such that it is bright enough that the spheres can be detected by the camera at their lowest point but not too intense that overexposure occurs. This is derived from the same requirements as Lighting Position.

2.10 **Sampling Frequency:** The camera should have a sampling frequency sufficient to capture changes in sphere formations accurately, yet not so high as to produce redundant data. This comes from Data Storage Size and Formation Recognition.

2.11 **Water Pump Force:** The water pump must be able to generate a flow rate sufficient to exert an upward force equal to the gravitational force acting on the magnetic spheres at their terminal velocity, ensuring that the spheres are suspended in the water without rising or sinking.

2.12 **Waterproof Setup:** The setup, including camera and lighting, must be fully waterproof to prevent damage if an accident in the water-filled environment occurs. This is to ensure the Setup Compatibility requirement is met.

2.13 **Water Quality and Clarity:** The water inside the cylinder must be clean and free of particles that could obscure the view of the spheres, to allow for high accuracy and recognition.

2.14 **Setup Stability:** The entire setup should be stable to avoid any movements that could blur the images and affect detection accuracy. This is derived from the requirements Accuracy and Reproducible Experiment.

2.15 **Contrasting Background:** The background should be uniform and provide sufficient contrast with the spheres, so that Formation Recognition is possible.

2.16 **Uploading to Server:** The camera must upload the collected data required by the Image Processing module to a server. This comes from Accessibility, Data Storage Size and Formation Recognition.

5.2.3. Training Data

The requirements for generating training data are based on its value for the deep learning algorithms, as it should represent reality as much as possible to achieve high accuracy in the overall model.

3.1 **Data Labelling :** The system should facilitate easy labelling of the captured training data, indicating the specific formation and number of spheres in each sample. This requirement is based on Training Time Efficiency, Reproducible Experiment and Validatability.

3.2 **Secure Attachment:** The spheres should be attached strong enough to keep their shape intact during the experiment, forming either a line or a ring. This comes from Validatability and Experiment Duration.

3.3 **Easy Assembly:** The setup should allow for easy assembly and disassembly of different formations with varying numbers of spheres to generate diverse training data. This comes from the Time Frame requirement.

3.4 **String Visibility:** In the case of using a string to attach the spheres, this string should be transparent to avoid interference with the image recognition algorithms and to ensure it does not undermine the accuracy.

3.5 **Realistic Training Data:** The generated training data must closely mimic the actual experimental data in terms of sphere size and weight, formation flexibility, lighting conditions and image quality. This is derived from the Training Time Efficiency, Accuracy and Validatability requirements.

5.3. Design Choices

In this section, all the alternatives, analyses and choices in terms of setup and sphere design will be presented and discussed. This will be on the topics of camera selection, lighting selection, sphere and magnet size considerations, paint colour selection, determination of the sampling frequency and the chosen method for generating training data.

5.3.1. Sphere and Magnet Size

In the design of the spheres, there are five requirements listed in Section 5.2.1 above, of which the first three will influence the design choices concerning sphere and magnet size.

The first requirement is about the sphere size. Figure 5.2 shows the cone in which the spheres are moving. The 20 cm height of the cone is the area that is visible in Figure 5.1. This is also the area of interest for the model. If the spheres move outside of this area, then the data can be discarded. The spheres should be able to form a line anywhere in this area, thus also at the lower side of the visible area. The diameter of this lower side is 12 cm. Furthermore, the experiment should be performed with up to six spheres. It is thus required that the spheres have a maximum diameter of 2 cm.

The second requirement is about the mass density of the sphere. This follows from the fact that the spheres should not be too heavy or too light. Whenever the spheres are too heavy, they remain at the bottom of the cylinder with the water flowing upwards. If they are too light, they will float on the water. When a sphere is 'floating' in a certain position in the water, the gravity force pointing downwards and the drag force pointing upwards are equal to each other. The formula for the gravity force is:

$$F_{\mathsf{g}} = (m_{\mathsf{sphere}} - m_{\mathsf{H_2O}})g \tag{5.1}$$

The mass of the sphere includes the mass of the sphere itself and the magnet inside of it:

$$F_{\mathsf{g}} = \rho_{\mathsf{3D}} V_{\mathsf{sphere}} + \rho_{\mathsf{magnet}} V_{\mathsf{magnet}} - \rho_{\mathsf{H_2O}} V_{\mathsf{sphere}} \tag{5.2}$$

With V_{sphere} being the total volume of the sphere minus the volume of the magnet. The mass density of the 3D material used for the sphere is 1169.2 kg/m^3. This is calculated by 3D printing a sphere with a certain volume and weighing it. The magnet that is used is a Neodymium magnet, having a mass density of 7640 kg/m^3. The mass density of the water is assumed to be 1000 kg/m^3. The formula for the drag force is:

$$F_{\mathsf{drag}} = \frac{1}{2} A C_{\mathsf{d}} v^2 \rho_{\mathsf{H_2O}} \tag{5.3}$$

With A being the surface of the sphere and C_d the drag coefficient, which is previously measured to be 0.58 [11], V is the terminal velocity and this is determined by the characteristics of the water pump. The pump can deliver a flow upward with a speed between 7 cm/s to 30 cm/s. If the speed goes below 7 cm/s, the flow will not be constant anymore and 30 cm/s is the maximum flow that the pump can deliver. However, this speed is measured at the top of the cone, so the speed inside the visible area is about twice as large due to the smaller area through which the water has to flow. Calculations show that for the design a speed of 33 cm/s is required, thus having a flow at the top of the cone of 16.5 cm/s.

The third requirement is about the energy barrier between two spheres. The deep-learning model will classify formations into three cases: either a *ring*, a *line* or *none*. Therefore, there must be an energy barrier between two magnets in surrounding spheres such that the spheres remain connected most of the time. This energy barrier represents the energy required to overcome the repulsive or attractive forces between two magnets to bring them to a desired position or to separate them. This can be quantified in terms of the magnetic potential energy, which depends on the magnetic moments, the distance between the magnets and their alignment, see equations 5.4 and 5.5.

Our spheres are designed to have an energy barrier of around 100 μJ for two reasons. First of all, in [11] the ideal disturbing energy between two spheres has been determined to be around 100 μJ. This 100 μJ follows from a combination of cylindrical magnets of 4 mm by 4 mm and a sphere diameter of 18 mm. Secondly, this assumption has been confirmed by experiments with spheres with an energy barrier of around 90 μJ. A lot of the time some spheres were disconnected, resulting in a "none" classification, which is the least desirable classification (see Figure 5.4). It can be concluded that an energy barrier of 90 μJ is too low. The formula for the energy of a single sphere (1) in the presence of a magnetic field of another sphere (2) is:

$$U_1 = -\mathbf{m_1} \cdot \mathbf{B_2} \tag{5.4}$$

Assuming that the spheres are aligned, the magnetic field is calculated as follows:

$$\mathbf{B_2} = \frac{\mu_0 \mathbf{m_2}}{2\pi \mathbf{r^3}} \tag{5.5}$$

Since all spheres contain the same magnet, $\mathbf{m_1} = \mathbf{m_2}$.

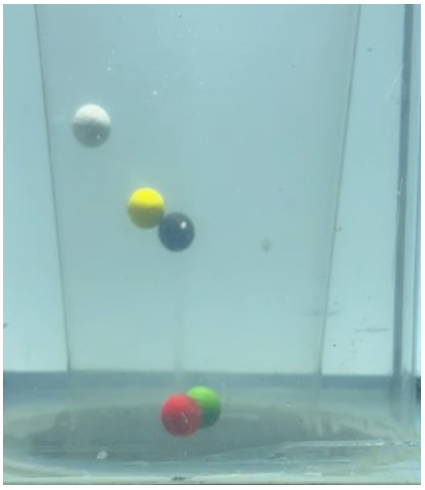


Figure 5.4: Experiment with spheres having an energy barrier of 90 μJ, resulting in a *none* classification

5.3.2. Sphere Colours

The fourth and fifth requirements of the sphere design are about colour and paint selection. The type of paint that was deemed most suitable was spray paint, as that would be the easiest to apply evenly on round shapes. The colour choices are based on maximum contrast between them to simplify the image recognition algorithm.

A spray paint was found, Edding Permanent Spray, that was expected to meet the Paint Durability requirement. This acrylic paint is suitable for almost all types of materials, dries quickly, is weatherproof, gives good coverage and - most promising and conforms to the requirement Colour Selection - it is available in 36 different colours, see Figure 5.5.

Figure 5.5: Colour options of the Edding 5200 Permanent Spray Premium Acrylic Paint

Based on knowledge from previous experiments with coloured spheres (see the colours in Figure 5.4), it was found that it is quite hard to distinguish white from the rest, when the camera is positioned on the side of the setup. Assumed is that this has something to do with the low contrast with the background

and that it is technically not a colour, but a combination of all colours. Black, on the other hand, was no problem to isolate.

The colours from Figure 5.5 were tested on the colour thresholding module found in Appendix C.2.1. It was noticed that some of those are RAL colours, which is a standardized colour system. Considering future applications, utilizing RAL colours provides a reliable reference for 'standard' colours. This is why it was chosen for the colours seen in Figure 5.6. These are red (*Verkeersrood*), black (*Diepzwart*), orange (*Neon oranje*), yellow (*Verkeersgeel*), green (*Geelgroen*), dark blue (*Elegant midnight*) and pink (*Telemagenta*). Despite its impracticality in colour isolation, the colour white (*Verkeerswit*) was also ordered as it can be useful to restore the spheres to their original colour, facilitating the application of a new colour if needed.

There are some uncertainties in the colour selection, in particular with blue and orange, that can only be confirmed or refuted through experimentally testing the colour recognition. Initially, the lighter blue (*Gentiaanblauw*) was preferred, as it has a larger contrast with black, but that colour was unfortunately unavailable. Additionally, it was found that the colour orange can easily be isolated. However, the only orange option is a neon colour; it cannot be said for certain how the colour recognition will respond to that.

Figure 5.6: Colour selection of the Edding 5200 Permanent Spray Premium Acrylic Paint

5.3.3. Camera

Certain camera specifications have been relevant in the selection of a camera. First and foremost, the camera size has been the biggest limitation, since a camera position from above is required and there is a relatively narrow cylinder on top of the setup (see requirements Camera Position and Camera Size in Section 5.2.2). 'Standard' cameras like DSLR and mirrorless cameras are instantly unsuitable because of their large sizes; even smaller compact cameras are too big. Action cameras were considered but those are, no matter how small, not compatible because of their typical wide-angle lenses (see requirement Camera Field of View in Section 5.2.2). Besides, action cameras often lack the necessary control over sampling frequency and lighting conditions. What was left, is the category of webcam cameras. Those tend to be compact and easy to integrate into a controlled experimental setup, with adjustable fields of view, resolutions and frame rates. Additionally, many webcams allow for real-time data capture via a direct connection to a computer.

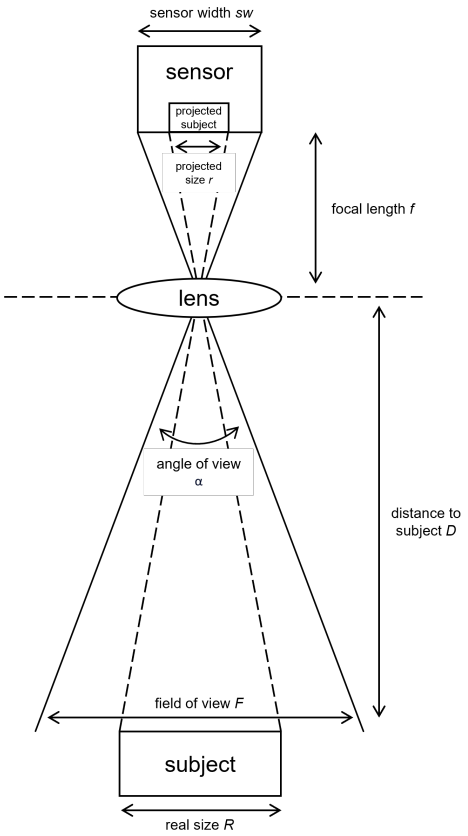


Figure 5.7: Schematic of camera proportions and distances

Finding the resolution

As can be seen in the requirements (Section 5.2.2), for Image Processing a minimum of 10 pixels must be seen in a sphere's radius at maximum distance, so underneath in the cone. The number of pixels is in close correlation with the field of view and the focal length of a camera. Both specifications influence the resolution and perceived image of a camera [8]. The number of pixels is calculated according to Figure 5.7. From this, an equation can be drawn up for the ratio between actual and perceived distances by the sensor:

$$\frac{r}{f} = \frac{R}{D} \tag{5.6}$$

Using the specific focal length *f* of a camera, the real radius of a sphere *R* of 1 cm and then the maximum distance to a sphere *D* of 46 cm, one can find the perceived sphere radius at the sensor *r*. Subsequently, the perceived radius can be divided by the size of one pixel in the camera sensor to find how many pixels this radius projection represents.

Finding the field of view

For the Camera Field of View requirement, finding the field of view F (m) equals:

$$F = 2\tan(\frac{\alpha}{2})D \tag{5.7}$$

with α the angle of view (degrees).

From this follows the width the camera sees at a certain distance, the minimum in the setup is 15,5 cm at a distance of 26 cm (the top of the 20 cm visible area).

Camera comparison

One webcam was selected, the Logitech StreamCam, whose specifications met all the requirements. However, another camera option emerged during the research, namely a Raspberry Pi Camera Module 3. This comes in four versions, normal and wide angle, and with and without infrared filter. The wide angle does not meet the field of view requirement, so only a normal angle with an infrared filter and a normal angle without an infrared filter were considered.

The requirements and camera specifications of the Logitech Streamcam and the Raspberry Pi cameras are compared in Table 5.1. Based on this analysis, the Raspberry Pi camera with an infrared filter emerged as the superior option.

Table 5.1: Camera requirements comparison

Requirements	**Weights**	**Logitech Streamcam**	**Raspberry Pi Camera Module 3**	**Raspberry Pi Camera Module 3 NoIR**
Costs	*3*	3 145 euros	4 56 euros	5 50 euros
Camera Size	*5*	4 48 x 58 x 66 mm	5 25 x 24 x 11,5 mm	5 25 x 24 x 11,5 mm
Camera Resolution	*5*	4 1920 x 1080 pixels focal length 3,7 mm pixel size 2,2 um –>min sphere radius 33 pixels[1]	5 4608 x 2592 pixels focal length 4,74 mm pixel size 1,4 um –>min sphere radius 66 pixels[1]	5 4608 x 2592 pixels focal length 4,74 mm pixel size 1,4 um –>min sphere radius 66 pixels[1]
Camera Field of View	*4*	4 diagonal angle of view 78° –>diagonal field of view 42 cm[2]	4 diagonal angle of view 75° –>diagonal field of view 40 cm[2]	4 diagonal angle of view 75° –>diagonal field of view 40 cm[2]
Camera Colour Accuracy	*5*	4 larger sensors generally have higher color accuracy	4 larger sensors generally have higher color accuracy	2 this camera has no infrared filter, meaning that colours will be distorted
Waterproof Setup	*2*	4 not completely waterproof, but has enclosure	3 open circuit board, measures must be taken to make watertight	3 open circuit board, measures must be taken to make watertight
Uploading to Server	*3*	4 direct interface with computers via usb	5 raspberry pi camera module directly connects to raspberry pi board	5 raspberry pi camera module directly connects to raspberry pi board
Total weight	***26***	**97**	**120**	**107**
Weighted average		**3,73**	**4,62**	**4,12**
% difference with max		**23,7%**	**0,0%**	**12,2%**

[1]: Here the maximum distance to subject is D=46 cm.
[2]: Here the minimum distance to subject is D=26 cm.

5.3.4. Lighting

In the lighting selection, there were three requirements to keep in mind: Lighting Position, Lighting Size and Lighting Brightness, see Section 5.2.2. In terms of position, lighting coming from the same direction as the camera was preferred. A bright flashlight would be ideal, as it is a narrow cylinder the lamp has to fit in. So after choosing the Raspberry Pi Camera, the size of the lamp was constrained to around 7 cm in diameter. For the brightness requirement, the value for the required luminous flux (in Lumen) was based on a back-of-the-envelope calculation of the existing flashlight in the cylinder which has about 12000 lm. This one was not suitable for the experiment because it was too large. The selection for lighting landed on the Ansmann T12000R flashlight which has a brightness of 12000 lm and is 64 mm in diameter, meaning that it meets all the requirements.

5.3.5. Sampling Frequency

The sampling frequency, in other words, the rate at which the camera takes pictures, is based on empirical observations. It is quite difficult to theoretically determine a value for this, so testing our spheres and observing their connecting behaviour was deemed as most useful in determining a sampling frequency. For the Sampling Frequency requirement, the picture rate is based on how much the formations seem to have changed after different time intervals. It is anticipated that a realistic value for accurately capturing formation changes will be around 1 image per second.

5.3.6. Training Data

Some considerations have been made about how to generate the training data. First and foremost, the generated data should be labelled so that it can serve as a verification of our actual experimental data, see requirements in Section 5.2.3. Another requirement is secure attachment, this is so that training data that needs to represent a ring, for example, cannot suddenly disconnect during the experiment and show a line formation. Keeping these two in mind, two general methods for making this training data were considered. One way is glueing spheres together in specific line and ring formations. Another method is pulling a string through the spheres to create more flexible formations. Returning to the rest of the training data requirements (easy assembly of formations, transparent strings and realistic data) led to the conclusion of combining the two methods, as both have upsides and downsides, see Table 5.2.

Table 5.2: Training data requirements comparison

Training data requirements	**Glue**	**String**
Data labelling	5	5
Secure attachment	5	5
Easy assembly	5	4
String visibility	5	4
Realistic training data	2	5
Total weight	**22**	**23**
Weighted average	**4,4**	**4,6**

It can be noted that magnets are not necessary in the case of the training spheres, since specific formations will be forced. It was decided to make use of lead fishing weights. Because of their specific shape, it is easier to use these in combination with a string, because the string can be pulled through the weights (see Figure 5.8).

Figure 5.8: Fishing weights from lead

Some calculations have been done to ensure that the training spheres have the same effective mass density as the spheres containing a magnet, and thus the same weight and size. Using a fishing weight of 0,3 g meets the *Realistic training data* requirement of Section 5.2.3.

5.4. Implementation

This section sums up the implementation of the design choices from Section 5.3 that are made based upon the requirements described in Section 5.2.

5.4.1. Sphere Design

There are five requirements for the design of the spheres: the size, the mass density, the energy barrier, the colour and the paint selection. First of all, the diameter of the sphere must be below 2 cm. Secondly, the mass density should have a certain value such that the spheres have a terminal velocity between 7 cm/sec and 30 cm/sec. Thirdly, the energy barrier between two magnets in connecting spheres is 100 μJ. Then, the six spheres must have a unique colour and lastly, the paint for the spheres must stay intact during the whole operation.

Taking everything into consideration, the spheres are designed with the following specifications:

- The spheres contain a cylindrical Neodymium magnet with a diameter and a height of both 4 mm;
- The diameter of the spheres is 18mm.
- 3D models for the spheres are designed and printed using the Ultimaker S5 3D printer, present in the EEMCS building.
- Seven printed spheres are painted for the experiment with the following colours: red, black, neon orange, yellow, green, dark blue and pink (see Figure 5.6).

Printing of the magnetic spheres

For the printing of the spheres, a 3D model is designed using the program Tinkercad. The design is illustrated in Figure 5.9 and the seven printed spheres can be seen in Figure 5.10.

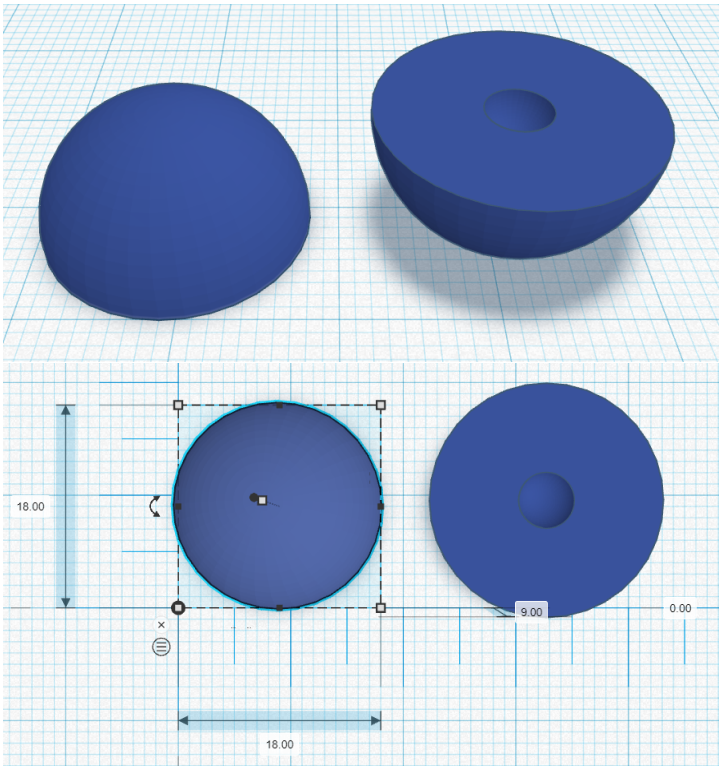


Figure 5.9: 3D model of two halves of a sphere with space left for a 4x4 mm cylindrical magnet inside

Figure 5.10: Printed magnetic spheres in seven different colours

5.4.2. Setup Design

The experimental setup has been implemented and the necessary components have been ordered and installed. This includes the camera with the appropriate field of view and resolution, lighting and all the components required for the Data Management module. In addition, a compressed air supply into the cylinder has been implemented to serve as ventilation and cooling for the components within. All the spheres necessary are glued and printed for the experiment.

Camera mould

A custom insert has been placed at the bottom of the cylindrical cone, which has precisely printed spaces for the camera and the flashlight, to keep these in place as much as possible. The design is shown in Figure 5.11.

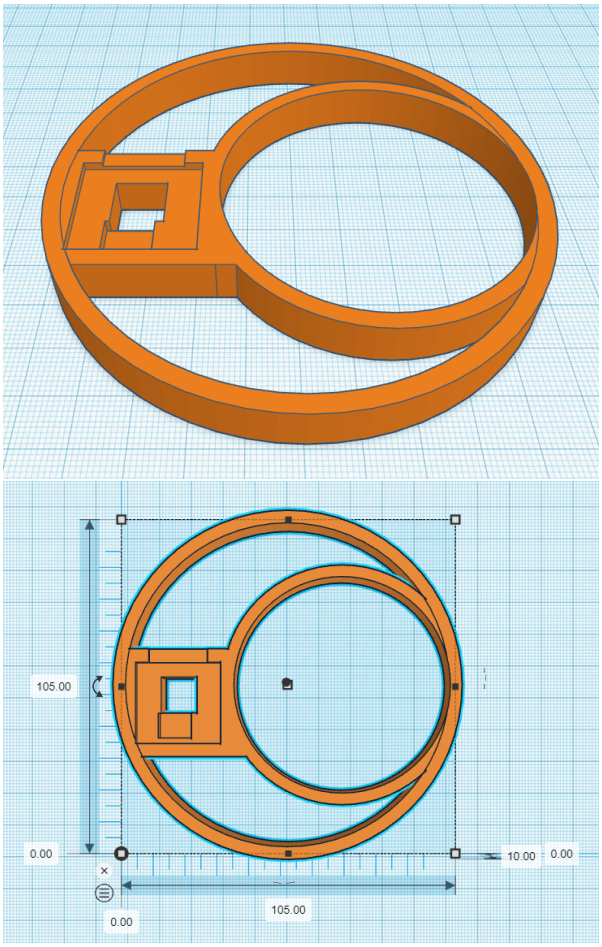


Figure 5.11: 3D model of the camera mould with space left for the Raspberry Pi camera and the flashlight

5.4.3. Training Data

Due to time constraints, initially training data using only four spheres has been generated, arranged in line, ring and none formations.

To meet the last two requirements of the sphere design - colour and paint selection - the paint colours from Figure 5.6 have been ordered and used on seven spheres. The colours red (*Verkeersrood*), yellow (*Verkeersgeel*), green (*Geelgroen*) and pink (*Telemagenta*) seem to be the most effective for the image processing model.

For the creation of the training data, two options have been considered and both are applied: glueing the spheres together in a specific formation and pulling a string through them, enabling a certain flexibility (see Section 5.3.6).

It has been decided that the ring will be glued and the line will be formed with a string. According to the requirements in Section 5.2.3, the data should be easily labelled and securely attached. This follows automatically using this method, given that the line could never form a ring and the glued ring is strong enough to never lose connectivity between all spheres.

Furthermore, the setup should be easily assembled and the string should not be visible to prevent interference with the image recognition program. Currently, four spheres are used to generate training

data. Additionally, the same method could be applied to five or more spheres: using a string for a line and glueing the ring. The string is transparent to prevent interference and the amount of string that is visible is very small (the spheres are tightly connected). It is anticipated that it is not necessary to generate more rings and lines with interchanged colours, since the deep learning model relies on relative positions and can interchange the coordinates of the different coloured spheres.

Lastly, the training data must be realistic. By using the string instead of just glueing a line, the line becomes more flexible and thus better represents reality.

Printing of the training spheres

As explained in Section 5.3.6, fishing weights made of lead are used for the training spheres, instead of magnets. These leads are spherical, not cylindrical, and thus somewhat different in size than the magnets. The 3D model had to be adjusted to this. Additionally, a very narrow, sort of hollow groove was implemented in both sphere halves to allow a string to be threaded through it. The design is shown in Figure 5.12.

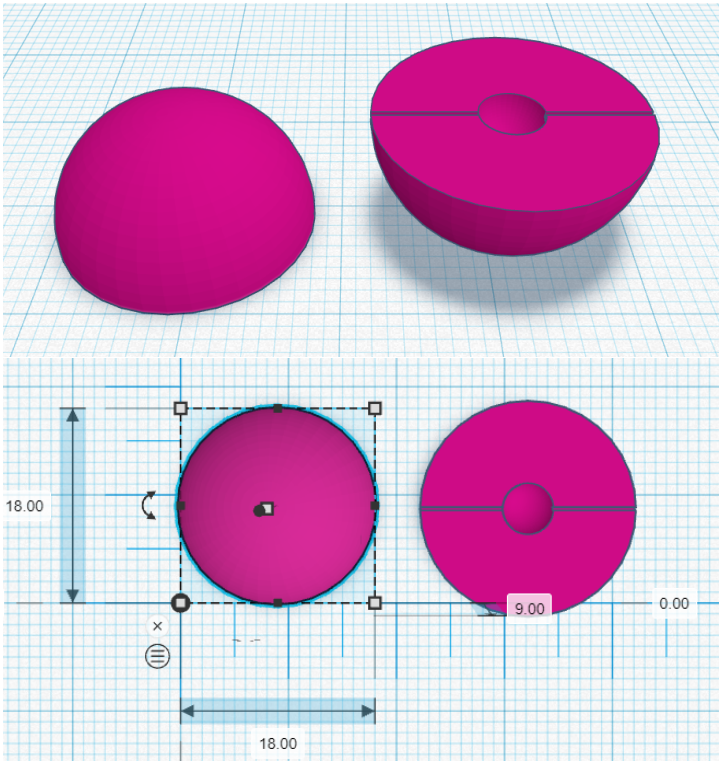


Figure 5.12: 3D model of two halves of a sphere, to be used for generating training data and with space left for a lead fishing weight and a string to be threaded through

Several of these designs have been printed and the training formation of four spheres in a line can be seen in Figure 5.13.

Figure 5.13: Printed training spheres formed in a line formation and connected via a string

5.5. Results

5.5.1. Sphere Design

To test whether the sphere design meets the requirements, some experiments have been executed. First of all, only one sphere was tested. The sphere was observed to float perfectly in the area of sight at a speed of 15 cm/s. As explained before, a test has been performed with an energy barrier of 90 μJ, which was a bit too low. Therefore, our spheres are designed to have an energy barrier of 100 μJ. However, the 3D print material of our spheres has a higher mass density than the previously designed spheres, resulting in a higher speed than in previous experiments. Although this speed is perfectly within the limits of the water pump, it does produce more turbulence. This extra turbulence results in more ”none” cases than was expected. The paint remains intact in the water, even after performing multiple experiments for a couple of hours.

5.5.2. Camera

Initially, it was found that the pictures taken from above were less clear than anticipated. After filtering the water, the quality of the images greatly improved. In addition, the camera fits perfectly in the mould. However, if the spheres move too high, they might disappear from view, since the camera in the mould is positioned somewhat off-centre, due to the flashlight. These unusable pictures will be discarded by the image processing model.

After printing, it was discovered that the mould did not quite fit in the narrow cylinder. It was made 1 or 2 mm too large. To solve the problem, the mould was cut open on one side and a small segment was removed so that the mould could be compressed and would fit in the cylinder.

5.5.3. Lighting

An unanticipated problem occurred with the flashlight. Unfortunately, it cannot charge and produce light simultaneously. For this reason, the original flashlight is used.

5.5.4. Sampling Frequency

Observations of the experiment showed that taking fewer pictures than one every two seconds would lead to insufficient information, as the spheres in the water moved quite quickly. It was also experienced that there was a limit to how fast data could be uploaded to the server (requirement Uploading to Server), a frequency higher than 1 image per second would deteriorate the performance of the Raspberry Pi. Eventually, the determined sampling frequency is one picture taken every 1.5 seconds.

5.5.5. Training Data

As said before, the training data will be created by glueing together a ring and using a string for a line. The training data for the none case has been produced by demolishing the ring formation and letting all four spheres float individually. A script is run on the Raspberry Pi (see 8), making it possible to automatically generate one picture per 1.5 seconds. Therefore, it is possible to generate 2400 images per hour. Since the deep learning model only needs 2500 pictures to train its model, the experiment has to run 62.5 minutes to generate enough training data for one formation. Some captures of the three different formations (*ring*, *line* and *none*) are shown in Figure 5.14.

(a) Ring **(b)** Line **(c)** None

Figure 5.14: Captures from the different training data cases

5.6. Discussion

5.6.1. Energy Barrier

The energy barrier between two magnets inside two connecting spheres is the main point of discussion. The spheres are ideally designed to have very little *none* cases. Due to the heavier mass density of the 3D printing material compared to the material that is used in previous experiments, the overall mass density, and thus weight, of the sphere becomes higher. Therefore, it is needed to have a higher water pump force, which in turn results in increased turbulence. Due to this higher turbulence, an energy barrier of 100 μJ is insufficient to have every sphere in a connected state most of the time. To omit the *none* cases, one solution could be to use a 3D print material with a reduced effective density. A lower mass of the spheres requires less force from the water pump and, consequently, less kinetic energy caused by turbulence. As an alternative, it is recommended to replace the 4 by 4 mm magnets with a 4 by 5 mm magnet, requiring only a few adjustments to the current design of the 3D sphere. The spheres do increase in weight, resulting in more turbulence. However, the step from a 4 by 4 mm magnet to a 4 by 5 mm magnet is also quite big in terms of magnetic force, increasing the energy barrier to 156 μJ.

Another consequence of the heavier spheres is the area that they can be found in. Since a higher velocity results in more turbulence and thus more *none* cases, the water pump force is kept relatively low. That means that the spheres will move generally in the lower parts of the cone. This can have a significant impact on the image processing and deep learning models, as these had expected the spheres to be higher up in the cone.

5.6.2. Light Source

The flashlight chosen as a light source from the top works perfectly fine. It produces enough light and is small enough to fit in the cylinder next to the camera. However, as is mentioned in the Results (Section 5.5), it is not possible to charge the flashlight while it is on. With only 2,5 hours of operation time and a disproportional 6 hours of charging time, this particular flashlight became unfunctional for this implementation as it conflicts with top-level requirement Experiment Duration. Therefore, it has been decided to use the original flashlight. It does not fit perfectly in the camera mould next to the camera but it does produce enough light for the spheres to be clearly visible, even when positioned on top of the camera.

5.6.3. Camera Position

The decision is made to put the camera in the cylinder, looking down at the spheres from the top. This has been decided because the formations tend to move along a horizontal axis. This implies that spheres forming a ring will have a higher visibility when looking from above. Considerations have been made that the deep learning model better predicts the formation when looking from the top. Another consideration is the light source placed at the top. This produces shadows at the bottom half of the spheres, which are visible at the front. The camera from above does not see these shadows, which is an advantage for the image processing model.

The alternative is placing a camera at the front. The camera placed at the top had much more difficulty producing clear pictures than at the front, so the water needed to be filtered first. Another downside of the camera positioned above is the background. The camera from above sees the bottom of the setup, while a camera from the front would have a white and uniform background, which matches sub-requirement 2.15, Contrasting Background. Since the image processing model is looking for the colours of the spheres, it has more difficulty finding these from the top than from the front. It is unclear to us whether the advantages of having the camera at the top outweigh the disadvantages.

5.6.4. Extend Number of Spheres

According to top-level requirement 3.4, the developed method must be generalisable to any structure and any shape of the spheres in the structure. The spheres can be easily reproduced and painted in different colours. The same yields for producing the training data. After the formations are made, the system can be run and automatically produce enough training data in one hour per formation.

5.6.5. Future Recommendations

Moving forward, several key areas should be addressed to enhance the overall effectiveness and integration of the project. To begin with, looking into the practicalities of the setup could improve user-friendliness. At present, the mould used for the camera at the bottom of the orange cylinder barely fits in there. In addition, it was designed to fit the smaller flashlight of around 6 cm in diameter. The final flashlight used is larger than that and does not fit in the mould. Besides this, the layout of the camera, lighting and the Raspberry Pi 5 inside the cylinder is quite complex and not effortless to put into place. In the future, the use of a sort of tube that holds everything at the right spot could be considered.

Secondly, as of right now, the setup is not waterproof. Due to time limitations, it is not yet ensured that in case of an accident, all the components are secured from water damage. To achieve this, the cylinder can be covered, for example, in a way that still allows for ventilation and cooling of the components.

Furthermore, another recommendation is the use of a second camera or a different camera position. Choosing to take pictures from above the setup, has introduced some complications, such as the non-uniform background of the bottom of the cone. The ideal would be to adjust the bottom and create more contrast with the sphere formations In practice, on the other hand, this would not be in alignment with the Setup Compatibility requirement. For this reason, experimenting with another camera perspective could yield some useful insights into image recognition. Moreover, the ability to take pictures from a second perspective is most likely to improve accuracy.

5.7. Conclusion

The objective of this subgroup is to provide the subsequent modules Image Processing and Deep Learning with the required data for 3D structure recognition. For this, the optimal test setup and sphere design have been examined and implemented. The experimental findings will be concluded in this section by reiterating the comparison of all the requirements with the design choices.

5.7.1. Results and Performance Evaluation

In conclusion, the Experimental Setup & Physics submodule has been quite successful in providing an answer to the research question, that is:

”How can we supply the right data to the Image Processing and Deep Learning modules that will enable them to recognise the formation of the 3D structures?”

Supplying the right data involved researching the optimal test setup and experimental spheres. The test setup has functioned accordingly and also the sphere design has met all the requirements, as illustrated in the section below. Only the Energy Barrier requirement remains to be disputed. The design resulted in an energy barrier that would deliver favourable data, in theory. In practice, however, it was experienced that the experimental spheres would float in the *none* classification the majority of the time. This formation is least desirable in acquiring useful results. It was found that merely focusing on the energy barrier will not suffice, the combination with the effective density is crucial for the structure formations.

Furthermore, successful training spheres have been designed and training data has been delivered. This part of the submodule has functioned properly and meets the requirements. For the future, the expansion of the training dataset is a real opportunity.

5.7.2. Alignment with Requirements

Top-level requirements

1.1 **Time Frame:** Due to time limitations, only a demo of four spheres can be performed. However, an extension to more spheres is easily performed (see Section 5.6.5).

1.3 **Setup Compatibility:** Since modifications to the current setup were not possible, there were some constraints that needed to be dealt with. However, this has not restricted the achievements of this submodule, all the implementations could be performed without disrupting the integrity of the setup.

2.1 **Training Time Efficiency:** Only two sphere formations have to be prepared: one ring and one line. After that, the experiment can run for hours generating the training data automatically. And finally, to train on *none* data, the ring and line formations can be demounted to form disconnected spheres and an experiment can run, again, for hours.

2.3 **Reproducible Experiment:** By using the same lighting, camera and water pump settings, all the factors apart from the independent variable stay constant, ensuring the reproducibility of the experiments.

3.6 **Experiment Duration:** A requirement was to run the experiment for 12 to 24 hours. This amount of time has not been tested yet, but it is expected that all the physical elements will endure, such as the paint and glued sphere formations. Only the duration of the connection between the camera and the server remains uncertain, but this requires further testing to confirm its stability.

Sphere design requirements

1.1 **Sphere Size:** The sphere diameter is taken to be 18 mm.

1.2 **Mass Density Sphere:** To ensure this requirement is met, calculations have been done with the effective density of the 3D print material, the size of the magnet and its density and the size of the sphere.

1.3 **Energy Barrier:** Cylindrical neodymium magnets have been chosen of 4 mm in diameter and 4 mm in height. This in combination with 18 mm sphere size gives an energy barrier between two spheres of around 100 μJ. Although this requirement is met, in hindsight a stronger magnet or smaller sphere size would be preferred.

1.4 **Colour Selection:** 7 colours have been ordered, namely: red, neon orange, green, yellow, pink, dark blue and black. This is one more than required to have the ability to alternate between colours.

1.5 **Paint Durability:** The paint selection fell on Edding Permanent Spray. Empirical observations have shown that the paint stays intact and that this requirement has been met.

Setup design requirements

2.1 **Costs:** In total, the expenses have cost around 300 euros, well under budget.

2.2 **Camera Position:** The camera provides a clear view of the spheres in the cone. Because of the flashlight in the same cylinder, the camera is positioned around 3 cm off-centre, which may result in a skewed perspective. It appears, however, to capture all the relevant information.

2.3 **Camera Size:** A camera is chosen that is truly small, only 25 by 24 mm, so this requirement has been met.

2.4 **Camera Resolution:** This requirement has been met with a focal length of 4,74 mm and a pixel size of 1,4 μm. From this follows a minimal sphere radius of 66 pixels.

2.5 **Camera Field of View:** The selected camera has a diagonal field of view at a 26 cm distance of 40 cm. This is not minimal (which is 15,5 cm), but it is sufficient.

2.6 **Camera Colour Accuracy:** Contrast between the chosen colours can clearly be detected.

2.7 **Lighting Position:** It was possible to direct light from the same perspective as the camera.

2.8 **Lighting Size:** The flashlight was able to fit within the same cylinder as the camera.

2.9 **Lighting Brightness:** Eventually, the already existing flashlight was used. The luminosity of this light could be adjusted so that the right brightness could be selected.

2.10 **Sampling Frequency:** The sampling frequency was determined at 1,5 s per image. More images per second would be preferable because of the quick changes in formations. However, the frequency was limited by the Raspberry Pi's capacity.

2.11 **Water Pump Force:** During the experiments, it was found that a pump velocity of around 15 cm/s ensures the floating of the spheres. However, this relatively high speed also caused quite some undesired turbulence in the sphere formations.

2.12 **Waterproof Setup:** It was not yet ensured the setup is waterproof, since the cylinder containing the camera, flashlight and Raspberry Pi is left uncovered, which is an area for future improvement.

2.13 **Water Quality and Clarity:** Initially, it was found that the camera position from above resulted in low-quality pictures. After refreshing the water, however, the images became a lot clearer, showing that it is essential to keep the water quality in mind.

2.14 **Setup Stability:** To improve the stability of the setup, a mould for the camera and flashlight has been designed. Although it was printed a bit too large, after some effort the mould could still fit and effectively hold the camera in place.

2.15 **Contrasting Background:** When using a camera position from above, instead of from the side of the setup, the background is not uniform anymore. This has also caused some problems in the Image Processing module with background removal. The fact that this requirement is not met in the current setup could account for some inaccuracies.

2.16 **Uploading to Server:** Uploading to the server runs successfully. First, data is transmitted from the camera to the Raspberry Pi, which in turn is connected to a computer via the internet, enabling smooth uploading of the acquired data.

Training data requirements

3.1 **Data Labelling :** Three experiments have been conducted with the training data: a ring formation, a line formation and four individual spheres disconnected. These three experiments are clearly labelled with the specific formation.

3.2 **Secure Attachment:** The training formations have held their required connectivity for the whole duration of the experiment, meeting this requirement.

3.3 **Easy Assembly:** It was quite easy to assemble the formations. Disassembling, on the other hand, will be a challenge because of the superglue used. Some future effort can be put into simplifying this process.

3.4 **String Visibility:** A transparent string is used, one that is not visible on the images, meaning that this requirement is followed successfully.

3.5 **Realistic Training Data:** Both the line and ring formation closely mimicked the actual experimental data. The line was flexible, to a certain extent in that it would never form a ring. The ring was robust and accurately represented this formation type.

6

Image Processing

6.1. Introduction

The image processing team's objective is to develop an algorithm that can accurately and efficiently process images to detect spheres and extract the desired properties: the centers, radii and visibility. This algorithm not only needs to handle clear and unobstructed images but must also be robust against partial occlusions and varying lighting conditions. This is an important step in the overall system, that aims to automatically identify structures such as lines or rings.

Various image processing techniques have been employed to detect and analyze geometric shapes. Traditional methods include Hough Transform, edge detection algorithms like Canny, and contour-based approaches. These methods have been successful in controlled environments but often struggle with occlusions and noise present in real-world scenarios. To enhance robustness, more recent advances incorporating deep learning techniques are employed in the developed algorithm for example by removing the background. However, these approaches require substantial annotated data and computational resources. Given the constraints and the specific needs of this project, a hybrid approach leveraging both traditional and modern techniques was considered.

After researching potential strategies, the sub-team developed an algorithm that applies image processing techniques to automatically compute the properties requested by the Deep Learning system. The algorithm was tested on preliminary images recorded with mobile phones of team-members, allowing the image processing team to work in parallel with the other teams. Additionally, binary images of occluded circles were generated to facilitate the comparison of different algorithms and ensure robustness. Also, this gave access to a much larger dataset of testing images ensuring controlled adjustment of parameters in the test images.

The project adopted an iterative approach, continuously evaluating and refining the requirements and design to align with the top-level goals and meet the expectations of the team supervisor. The image processing component serves as a bridge between the raw images and the data required for the deep learning algorithm, which is a subsequent stage in the system.

The following sections will provide a breakdown of the image processing team's work. A simplified overview of the final implemented subsystem can be seen in Figure 6.1 and was designed to meet the requirements, including sphere detection, feature extraction, noise resistance, computation efficiency, and data management. A robust sequence of steps ensures that the system is resilient to noise and capable of processing multiple sequential tasks efficiently.

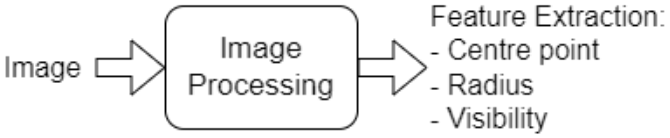


Figure 6.1: Black Box Pipeline of Image Processing

Image processing black box overview

1. **Colour Enhancement:** Improves the visibility of spheres by enhancing the colours in the images.
2. **Background Removal:** Eliminates irrelevant background details, isolating the spheres more effectively.
3. **Sphere Isolation via Color Detection:** Uses colour-based techniques to distinguish spheres from the background.
4. **Extrapolation using Hough Transform:** Applies the Hough Transform to compute centre coordinates and radii of the isolated spheres.
5. **Visibility Calculation:** Assesses the visibility of each sphere by comparing it to the respective reconstructed complete circle.
6. **Data Output:** The computed features for post-inspection and further analysis.

6.1.1. Outline of the Chapter

The outline of this chapter is as follows: Section 6.2 will discuss the specific requirements for the image processing component. It will detail the constraints and specifications that the image processing algorithm needs to meet to contribute to the overall system goals.

In Section 6.3, the design choices made during the development of the image processing algorithm will be explained. This section will also explore alternative methods that were considered, providing insights into the decision-making process and the rationale behind selecting the final approach.

Section 6.4 will dive into the details of the implementation of the chosen image processing algorithm. It will describe the steps taken to develop the system, the challenges encountered, and how these were addressed.

The performance of the image processing system will be discussed in Section 6.5 and Section 6.6. It will present the results obtained from testing the system on various images. The accuracy of the system in detecting the centers, radii, and visibility of the spheres will be evaluated and compared to the initial requirements.

Finally, Section 6.7, will review the developed image processing system, summarizing its performance and the extent to which it meets the project goals. Potential future improvements will be analyzed to enhance the system's accuracy and efficiency.

6.2. Program of Requirements

The overall goal is to compute individual spherical objects' center coordinates, radius, and visibility in a two-dimensional image of the self-assembled structure of magnetic spheres in a turbulent driven flow of water. This section outlines the Program of Requirements (PoR) for the image processing part, detailing the necessities to determine the features mentioned before. The PoR ensures clear objectives and guided design choices. The requirements are divided into four categories: Overall Subsystem, Sphere Isolation, Feature Extraction, and Data Storage.

6.2.1. Overall Subsystem

1.1 **Accuracy:** The accuracy of the radius and centre point should be 98.2% for 3 spheres, 96.1% for 4 spheres, 96% for 5 spheres and 95.4% for six spheres (see Figure 7.14 from the Deep Learning sub-team to achieve an overall system accuracy of 95%).

1.2 **Self-Contained Algorithm:** The algorithm must operate independently without relying on external applications.

1.3 **Iterative Sequential Performance:** The system should perform multiple sequential processing and measurement tasks iteratively.

1.4 **Computation Time:** The time it takes to process all images obtained during an experiment should not lie above 2 hours.

1.5 **Multi-Sphere Computation:** The algorithm must detect and compute the required parameters for up to six individual spheres.

1.6 **Automated Process:**: The algorithm must automatically compute the features of the detected spheres when an image is inputted.

6.2.2. Sphere Isolation

2.1 **Noise Resistance:** The model should effectively remove environmental and system noise caused by factors such as light reflections, shadows, and water disturbances, without losing important information.

2.2 **Sphere Detection:** The spheres' colour should be distinguishable by the software algorithm in the images taken of the observed structure.

6.2.3. Feature Extraction

3.1 **Position:** The system must determine and store the center position coordinates of each sphere.

3.2 **Radius Computation:** The system must determine the radius of each observable circle in the 2D image taken from the 3D structure of spheres.

3.3 **Visibility:** The system must calculate the visibility of each sphere by comparing it to the reconstructed complete circle with the identical computed centre coordinates and radius.

6.2.4. Data Management

4.1 **Input File Format:** The system should be able to process `.jpg` images, as this is the standard format received from the camera.

4.2 **Output File Format:** The computed features should be in a format readable by the Deep Learning Neural Network: a NumPy array, consisting of the centre (x,y), radius, and visibility.

6.3. Design Choices

In developing a robust solution for the image processing component of the system, various design choices were explored and evaluated. This section delves into multiple design options and their respective analyses, providing insights into the selection process based on specific criteria and performance metrics. With the use of weighted tables comparing the amount of adherence to the top-level and sub-group specific requirements per alternative, the best approach for every choice is substantiated.

6.3.1. Camera Position

In a similar project executed by Tijmen [12], one camera was placed in a horizontal recording direction on the side of the test tank. However, the camera could also be placed at the top of the test setup and record the structure in a vertical direction pointing down. Recordings taken with the camera recording horizontally at the side of the tank resulted in a significant loss in the ability to have a significant portion of the sphere captured in the camera. Mainly when the structure of spheres formed a ring, some spheres were behind other spheres such that the sphere was not visible. Also, when the camera records the spheres horizontally at the side of the tank and the light source is placed vertically at the top of the tank, shadows cause implications in sphere isolation. Example images of these implications can be found in Appendix E.1.1. In Appendix D.1 the comparison between the two camera positions can be found. Eventually, the choice of having a camera from the top scores significantly higher than having a camera from the side. Therefore, it was decided to place the camera vertically at the top of the test setup.

6.3.2. Feature Extraction Method

To extract circle features a distinction between the two methods was made:

Alternatives

1.1 Immediate Circle Detection

1.2 Binary Thresholding + Reconstruction

Analysis and Choice

Table D.2 weighed the selected method to be sphere isolation by binary thresholding, followed by circle reconstruction to extract the circle features. This method is preferred over immediate circle reconstruction as it is less accurate and robust. Another main advantage is that the binary thresholding and reconstruction is expected to have better processing speed and flexibility in the objects that needs to be extracted.

6.3.3. Pre-processing Techniques

To increase the chance of successful sphere detection and feature extraction, the images had to be prepared to enhance the quality.

Alternatives

2.1 LAB Image - CLAHE Enhanced L-channel

2.4 Background removal

2.5 Non-Local Means Denoising

2.6 Contrast and Brightness Adjustment

2.7 Remove Shade with Low Frequencies

2.8 Glare Removal using CLAHE

2.9 Temperature Adjustment

Analysis and Choise

The selected pre-processing techniques include: **LAB Image - CLAHE Enhanced L-channel**: Underwater digital images generally suffer from more blur, low contrast, non-uniform lighting, and diminished colour, due to different colour absorption underwater. To conquer the problem of uneven illumination, a proposed [13] pre-processing technique based on CLAHE in the LAB colour space improves the quality of underwater digital images. **Background Removal**: To limit thresholding of parts in the image that are not part of the spheres. **Non-Local Means Denoising**: to smooth the artefacts that may have arisen during colour enhancement. These techniques collectively improve image quality, enhance features of interest, and reduce noise. In Appendix D.1, Table D.3 the considerations can be found. The decision can be traced back to requirements 2.1 and 2.2.

6.3.4. Sphere Colours

The first step in detecting spherical objects in the image depends on the appearance of the spheres.

Alternatives

3.1 All Spheres are the Same Colour

3.2 All Spheres are a Unique Colour

3.3 Half the Sphere One Colour, the Other Half Another Colour (Consistent Across All Spheres)

Analysis and Choice

Spheres with unique colours or half-and-half colouring provide distinct advantages for colour-based segmentation, simplifying detection and differentiation. Therefore, both options were explored. The implementation process of both algorithms can be found in Section 7.4. The weighted table can be found in Appendix D.4. The decision can be traced back to requirements 1.5 and 2.2.

6.3.5. Sphere Detection Techniques

Alternatives

4.1 Colour Threshold - Binary Isolation

4.2 Contour, Concave, Segment grouping - Estimates Contours

4.3 Canny Edge Detection

4.4 Watershed Algorithm

4.5 Gradient detection [9]

Analysis and Choice
Colour Threshold - Binary Isolation is selected for its simplicity, minimizing the computation time, and effectiveness in isolating spheres based on their unique colour. This choice is substantiated by the table from Appendix D.5. For half-and-half colouring, the Contour, Concave, Segment grouping alternative was explored. The decision can be traced back to requirement 2.2.

6.3.6. Circle reconstruction Algorithms

Due to possible overlapping of spheres, they need to be reconstructed to find the radius and center point.

Alternatives

5.1 Bayesian Extrapolation [18]

5.2 Matched Filter

5.3 Minimum Enclosing Circle

5.4 Hough Transform [3]

Analysis and Choice
Minimum Enclosing Circle algorithm and Hough Transform are selected for their robustness and accuracy in detecting circular shapes in images. See Table D.6. The decision can be traced back to requirements 3.1 and 3.2.

6.3.7. Intermediate Temporal Image Storage

To check processing steps, a temporal storage format is selected:

Alternatives

6.1 PNG

6.2 BIN

6.3 PSD

6.4 JPEG

Analysis and Choice
PNG is chosen for its lossless compression, ensuring high image quality throughout the processing stages. The table arguing for this choice can be found in Appendix D.8. The decision can be traced back to requirement 1.1.

6.3.8. Visibility Metrics

To determine the overlap of the detected and constructed circles a visibility metric is introduced. This is used by the Deep Learning sub-team to give a measure of the probability of the extracted parameters being accurate up to a certain point.

Alternatives

7.1 Jaccard Similarity Coefficient

7.2 Dice Similarity Coefficient

7.3 Fuzzy Logic

7.4 Easy Ratio: Detected / Reconstructed

Analysis and Choice
Jaccard and Dice coefficients are selected for their effectiveness in evaluating the overlap and similarity between detected spheres and ground truth data. The weighted table, D.9, substantiated the selected method. The decision can be traced back to requirement 3.3.

6.3.9. Background Theory

To gain a deeper understanding of the design choices that were made, a background study is performed. In this section the explanation and theory behind every processing step can be found, supported by mathematical equations where applicable.

Preprocessing Theory

The **LAB colour space** is a colour-opponent space with dimensions L for luminance and a and b for the colour-opponent dimensions(chromaticity). It is designed to approximate human vision and is often used in image processing because it separates the luminance (lightness) from the colour information. This separation allows for more effective color-based segmentation and analysis.

CLAHE (Contrast Limited Adaptive Histogram Equalization) is applied to the L-channel to improve local contrast and enhance the visibility of features in the image. The CLAHE method optimizes maximum entropy equalization, while the contrast gets limited. [2]

$$\text{CLAHE}(L) = clip(\frac{cdf(L) - min(cdf(L))}{max(cdf(L)) - min(cdf(L))} \cdot (L_{max} - L_{min}) + L_{min}) \tag{6.1}$$

Where $cdf(L)$ is the cumulative distribution function of the L-channel, and L_{max} and L_{min} are the maximum and minimum pixel values.

Contrast enhancement ensures more accurate **background removal**: where a pre-trained deep learning model is used to segment the foreground (objects of interest) from the background.

$$I_{foreground} = Model(I) \tag{6.2}$$

I is the input image, and $I_{foreground}$ is the segmented image with the background removed.

To filter artefacts in the enhanced image, a denoising method is applied that reduces noise while preserving edges by averaging similar pixel patches throughout the image. This is called **Non-local Means (NLM) Denoising** [5]:

$$I'(x,y) = \frac{1}{C(x,y)} \sum_{p \in \Omega} I(p) \cdot exp\big(-\frac{||I(x,y) - I(p)||^2}{h^2}\big) \tag{6.3}$$

Where $I'(x,y)$ is the denoised image, Ω is the search window, h is the filtering parameter, and $C(x,y)$ is the normalization factor.

Sphere Isolation Theory

To isolate spheres, selecting distinct colours that can be easily recognized is essential. The **HSV (Hue, Saturation, Value) colour space** is used because it separates chromatic content (hue and saturation) from intensity (value), making it easier to segment based on colour.

The conversion from RGB to HSV is [10]:

$$V = max(R,G,B) \tag{6.4}$$

$$S = \begin{cases} 0 & \text{if } V = 0 \\ 1 - \frac{min(R,G,B)}{V} & \text{otherwise} \end{cases} \tag{6.5}$$

$$H = \begin{cases} 0 & \text{if max(R,G,B) = min(R,G,B)} \\ 60° \cdot \frac{G-B}{V-min(R,G,B)} + 0° & \text{if V = R} \\ 60° \cdot \frac{B-R}{V-min(R,G,B)} + 120° & \text{if V = G} \\ 60° \cdot \frac{R-G}{V-min(R,G,B)} + 240° & \text{if V = B} \end{cases} \tag{6.6}$$

Binary selection is done using thresholding:

$$I_{bin}(x,y) = \begin{cases} 1 & \text{if } H_{min} \leq H(x,y) \leq H_{max} \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

Feature Extraction Theory

The **Minimum Enclosing Circle algorithm** uses geometry to find the smallest possible circle enclosing the detected segmentation. This can define the size and position of the spherical objects within the image. The general equation of a circle with centre (a,b) and radius r is:

$$(x-a)^2 + (y-b)^2 = r^2 \tag{6.8}$$

Circle detection using the **Hough Transform** [3], works by transforming points in the image space to a parameter space where circles can be identified as peaks. It is defined in a parametric form, where (a, b) is the centre and r the radius:

$$x = a + r\cos\theta \tag{6.9}$$

$$y = b + r\sin\theta \tag{6.10}$$

Points in the image that lie on the circumference of a circle, map to curves in the Hough space, and the intersection of these curves indicates the presence of a circle.

Visibility Theory

The **Dice Similarity Coefficient(DSC)** is defined as:

$$DSC = \frac{2|A \cap B|}{|A| + |B|} \tag{6.11}$$

and the **Jaccard Similarity Coefficient(JSC)** as:

$$JSC = \frac{|A \cap B|}{|A \cup B|} \tag{6.12}$$

Where A and B are two sets representing the original and the reconstructed segmentation. Both metrics quantify the similarity between these segmentations, with values ranging from 0 (no overlap) to 1 (perfect overlap). For a consistent visibility evaluation, the JSC is selected.

6.4. Implementation

This section will describe the implementation and the design choices made to meet the top-level and image processing requirements.

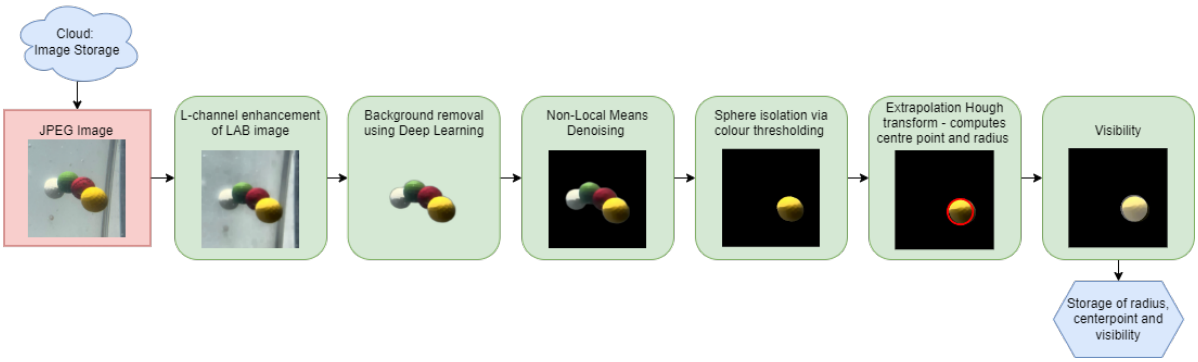


Figure 6.2: Pipeline of the Image Processing Implementation (Appendix E.32)

The strategies and algorithms were selected based on the background information in Section 6.3.9 to comply with the PoR. The input image for the processing algorithm is provided by the hardware team, with the image-taking process detailed in Chapter 5. An example image processed by the algorithm is shown in Figure 6.2. The following sections discuss the implementation details of the image processing algorithm.

6.4.1. Radius and Center coordinates extraction

The final implementation was developed in Python, as explained in Section 3.3.3. Primarily using the OpenCV library due to its optimized C++ codebase, ensuring robustness, computational efficiency, and real-time image processing capabilities [21].

To improve the accuracy of the contour-finding algorithm, colour enhancement was applied to the input image. Initially, histogram equalization was attempted to enhance contrast; but it increased noise due to unequal pixel value representation across different colour channels. The Contrast Limited Adaptive Histogram Equalization (CLAHE) method (explained in Section 6.3.9), significantly improved contrast while reducing noise, enhancing the detection of spheres in binary images. After applying `CLAHE`, a `GaussianBlur` was used to smooth edges and remove sharp transitions. This was followed by image dilation to connect fragmented edges and erosion to refine these connections. The process is illustrated in Figure 6.3.

Input image Pre-processed color enhanced image

Figure 6.3: Pre-processing and colour enhancement result

To limit computational complexity, the background is removed to make only the spheres visible. The background removal algorithm, `rembg`, efficiently handles relatively simple backgrounds using a deep learning approach. This results in a significantly reduced amount of pixel information to process [20]. The result of applying the rembg background removal algorithm can be seen in Figure 6.4.

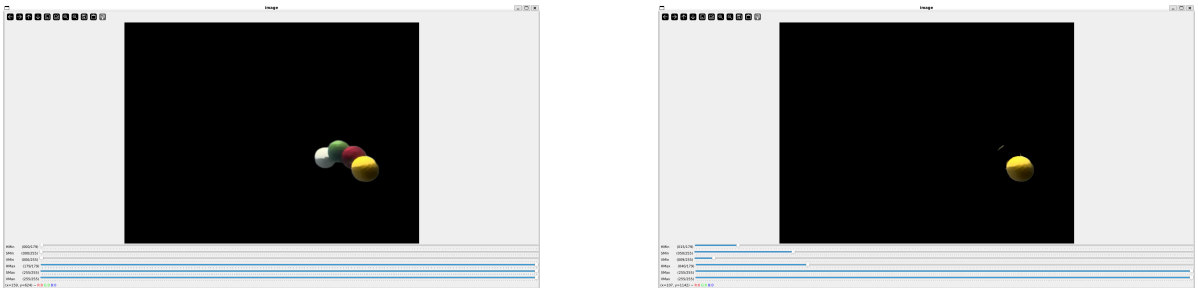
Pre-processed color enhanced image Removed background image

Figure 6.4: Rembg remove background result

To denoise the image, the `fastNlMeansDenoisingColored` function from the `cv2` library was applied to the background-removed image. This function searches for similar patches in an image and averages them to filter out noise [4] enhancing the overall accuracy of the contour-finding algorithm and property computation.

In Section 6.3.9 the HSV colour space was chosen for its invariance to illumination brightness [15]. A

trackbar was used to visualize the minimum and maximum values for the Hue, Saturation, and Value channels. The trackbar adjusts the H, S, and V thresholds, as illustrated in Figure 6.5. The Python script can be found in C.2.1.



Full HSV Range HSV Range for Yellow

Figure 6.5: HSV Range Selection

The image is converted from BGR to HSV colour space. Each pixel is compared to the lower and upper H, S and V threshold values to create a binary image. Accurate threshold values ensure the binary image contains only the visible part of the occluded sphere. This process is shown in Figure 6.6. Finally, a `GaussianBlur` is applied and the image is dilated and eroded to smooth and connect the edges. This process is repeated for all spheres in the input image, isolating and storing each sphere in a binary image. The isolated yellow sphere in the binary image can be seen in Figure 6.6.

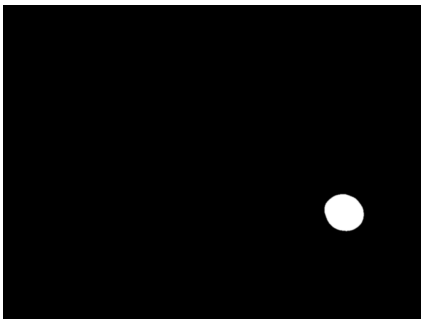


Figure 6.6: Binary image of yellow isolated sphere

Sphere Isolation:Unique colours vs Consistent Half-and-Half Colours

Two methods were evaluated for isolating spheres in the image processing algorithm: the Unique Colour method and the Consistent Half-and-Half Colour scheme.

Unique colour combined with binary thresholding method:

- Advantages: This method is straightforward and computationally efficient, making it well-suited for scenarios where spheres can be distinctly coloured.
- Limitations: It lacks flexibility and generalizability for more complex and varied scenarios where sphere colours might overlap or be indistinct.

Consistent half-and-half colour scheme - contour and concave segment grouping:

- Advantages: Potentially more generalizable and future-proof, capable of handling more complex scenarios.
- Limitations: Higher computational complexity and prone to errors, particularly in overlapping contour segments. An example trial image of this method is shown in Appendix E.19.

Ultimately, the Unique Colour method was chosen due to its simplicity and efficiency, despite the potential future benefits of the Consistent Half-and-Half Colour scheme. The latter method was not fully implemented due to time constraints.

Circle Reconstruction Algorithms: Hough Transform vs. Min Enclosing Circle

In Section 6.3.6 various algorithms for determining the radius and centre coordinates of a binary image were discussed. Two main algorithms were considered: the textttmin_Enclosing circle method and the Hough Transform.

`min_Enclosing Circle` Method:

- Initial Assessment: Initially seemed optimal due to its efficiency in specific geometric conditions.
- Limitations: Proved inaccurate when less than half of the circle's perimeter was visible, failing in scenarios with partial occlusion.

Hough Transform:

- Initial Assessment: Scored 9% lower in initial tests but demonstrated higher overall robustness across varied scenarios.
- Performance: Chosen for its ability to handle cases with partial visibility of the sphere's perimeter.

To compare the accuracy of these algorithms, an initial accuracy test was conducted using artificial images. A dataset was generated with a white circle (radius of 73 pixels) in the centre of each image. A black circle, representing an occluding sphere, was placed on top of the white circle and displaced incrementally until it was completely out of the white circle. This setup simulated the occlusion scenarios expected in real experiments, as shown in Figure 6.7.

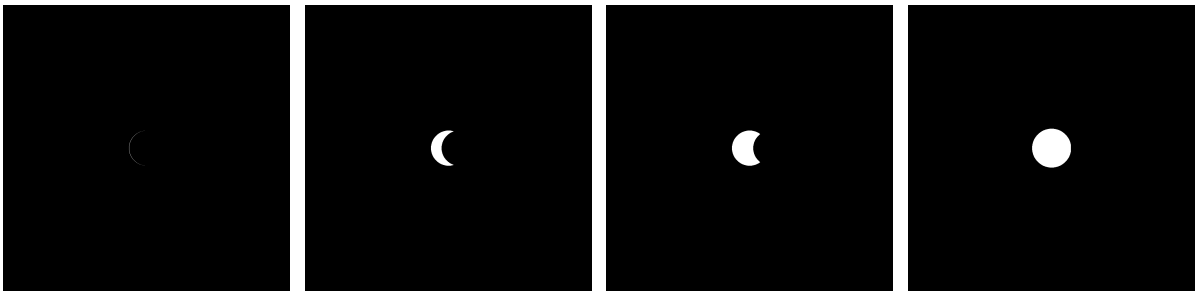


Figure 6.7: Illustrative examples of images from the accuracy test set

The error in the computed radius for each algorithm was calculated using Equation 6.13:

$$e_{radius}[\%] = \frac{\text{computed radius} - \text{correct radius}}{\text{correct radius}} * 100 \tag{6.13}$$

Comparing the computed radius to the actual radius provides an accurate measure of the inaccuracy of the computed radius. The following graph, Figure 6.8, shows the error in the computed radius when the different algorithms are applied.

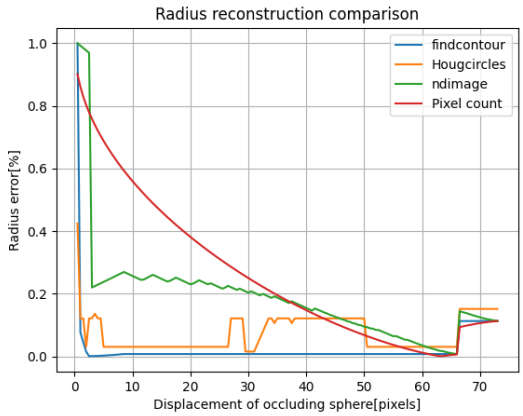


Figure 6.8: Comparison of radius reconstruction when displacing one occluding sphere

Initial evaluations suggested that the `min_Enclosing` circle method might provide the most accurate results. However, further investigation revealed that this method's accuracy heavily depends on specific geometric characteristics. Notably, when less than half of the circle's perimeter is visible, the `min_Enclosing` circle method yields inaccurate radius estimates. Examples of common fault reconstructed spheres using the min_Enclosing algorithm can be found in Appendix E.11. Given the varying radii observed in images of spheres within the tube, it is impractical to establish a definitive threshold for the perimeter visibility required for accurate results using this method.

Consequently, an updated evaluation of the circle detection requirements, detailed in Appendix D.1.6, led to the exclusive implementation of the Hough Transform due to its superior performance against the defined criteria. The following code snippet demonstrates the implementation of the Hough Transform using the OpenCV library:

```
# Apply the HoughCircles function on the edges
circles = cv2.HoughCircles(edges,
                cv2.HOUGH_GRADIENT,
                dp=4.7,
                minDist=10,
                param1=100,
                param2=50,
                minRadius=66,
                maxRadius=117)
```

Listing 6.1: Python code: Hough Transform circle implementation in cv2

Below an overview of definitions of the Hough Circles parameters is given.

- dp : Size of accumulator in comparison to the input image.
- minDist : Minimum distance between neighbouring centres (when more circles can be detected).
- param1 : Internal Canny-edge detector threshold value.
- param2 : Accumulator threshold for detecting circle centers.
- minRadius: Minimum radius of the circle to be reconstructed.
- maxRadius: Maximum radius of the circle to be reconstructed.

The number of pixels per radius can be computed by Equation 5.6 of the Experimental Setup & Physics team. Depending on the distance used, one can find the `minRadius` and `maxRadius` in pixels. In the beginning of the implementation phase it was decided that the spheres would be detectable in a height range of 20 centimeters, resulting in a minimum radius of 66 pixels and a maximum radius of 116 pixels for the detected sphere. The `minDist` parameter, which specifies the minimal distance between detected circle centres, is calculated as:

$$\texttt{minDist} = 2 * \texttt{minRadius} \tag{6.14}$$

With a minimum radius of 66, the `minDist` equals 132.

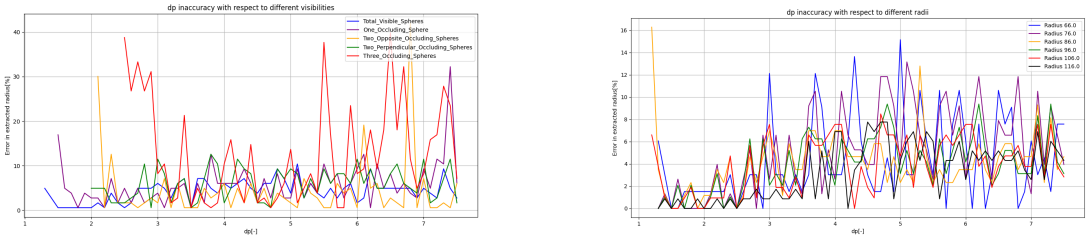
Parameter Optimization

To identify the optimal values for `dp`, `param1`, and `param2`, multiple test images were analyzed. These tests examined the impact of each parameter on radius and centre accuracy across various visibilities and radii, as detailed in Appendices E.1.3 and E.1.4.

dp Parameter

Figures 6.9 and 6.10 illustrate the dependency of radius and centre errors on the `dp` parameter for different visibilities and radii.

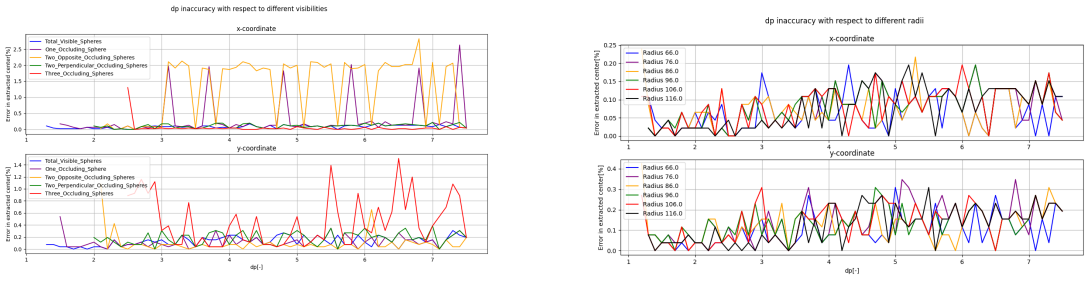
For different visibilities, a `dp` value of 4.7 results in the lowest radius error. In contrast, a `dp` value of 2 minimizes the radius error for circles of varying radii. The centre coordinate error does not show a significant correlation with the `dp` value for different radii. However, for varying visibilities, the y-coordinate error is minimized at `dp` = 4.7 when only part of the sphere is visible. Thus, `dp` was set to 4.7 based on these observations.



Visibility based dp dependency

Radii based dp dependency

Figure 6.9: dp Dependency when the radii error is calculated



Visibility based dp dependency

Radii based dp dependency

Figure 6.10: dp Dependency when the centre coordinate error is calculated

param1 Parameter

The effect of `param1` on radius and centre coordinate error is detailed in Appendix E.1.5 The results indicate no significant correlation between `param1` and the errors, leading to a default value of 100 for `param1`.

param2 Parameter

Figures 6.11 and 6.12 show the impact of `param2` adjustments on radius and center errors.

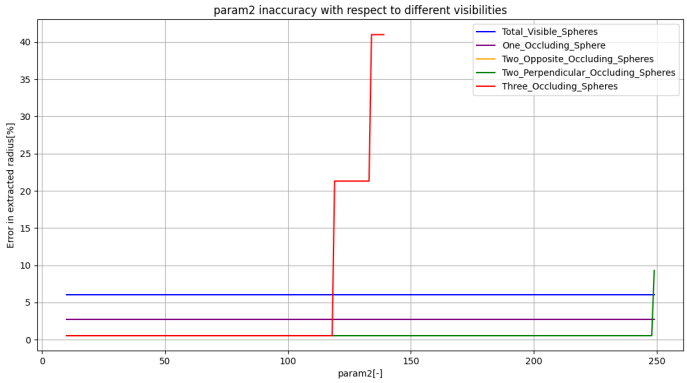


Figure 6.11: Effect of param2 adjustment on the computed radius inaccuracy for different visibilities

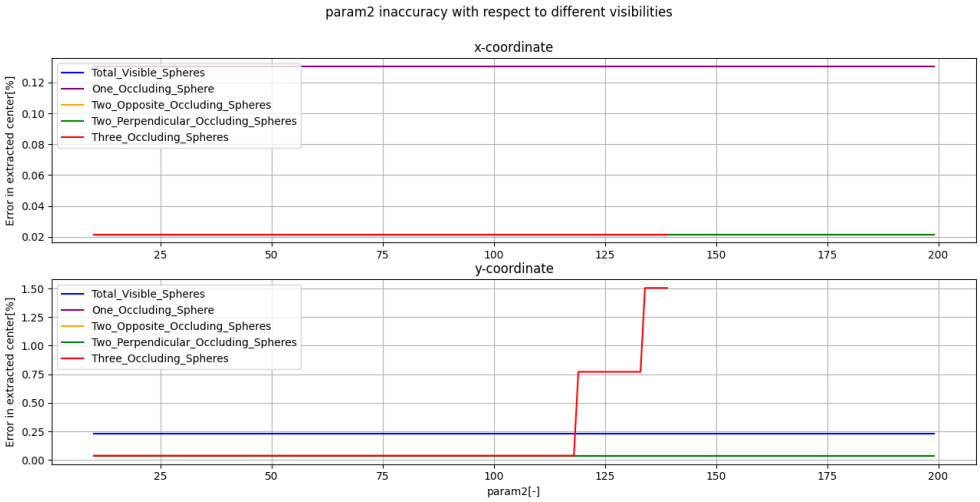


Figure 6.12: Effect of param2 adjustment on the computed center inaccuracy for different visibilities

A steep increase in error is observed at $\texttt{param2} = 118$ when three occluding spheres are present. Thus, $\texttt{param2}$ was set to 100 to ensure accuracy under various conditions.

6.4.2. Visibility

The visibility is computed using the Jaccard Similarity Coefficient (JSC), as detailed in Section 6.3.9. The extracted radius and center coordinates are used to reconstruct the circle, and the binary isolated sphere image is compared to this reconstruction using the JSC.

Final implementation in Python

The final overall implementation in python code can be found in appendix C.2.2.

6.5. Results

Since images were only available at the end of the project, the image processing algorithm was first tested on generated images. Generated images allow for the evaluation of accuracy, as the centre and radius are known inputs in the image generation algorithm. This enables a comparison between the computed reconstructed centre and radius and the actual centre and radius.

6.5.1. The Generated Image Set

The Generated Image set is divided into two categories:

1. **Radius Variation Set:** This set examines the effect of varying radii on the performance of the image processing algorithm. The radii range from 66 to 116 pixels, with increments of 2.5 pixels. The colours tested are green, red, pink, white, and yellow. Sample images are provided in Appendix E.1.6.
2. **Visibility Variation Set:** This set assesses the impact of varying levels of occlusion on the algorithm's performance. A circle with a centre in the middle of the image and a radius of 91 pixels is used. The same colours as the radius variation set are applied. An identical black circle is used to occlude the target circle in increments of 20 pixels until it no longer covers any part of the target circle. This is tested with one, two, and three occluding spheres. Sample images are in Appendix E.1.6.

Influence of radius on performance

The algorithm's performance was assessed using the radius variation test set. The following equations were used to compute the accuracy of the reconstructed centre and radius:

$$\text{radius accuracy} = 100 - \frac{|\text{reconstructed radius} - \text{actual radius}|}{\text{actual radius}} * 100$$

$$\text{x-coordinate accuracy} = 100 - \frac{|\text{extracted x-coordinate} - \text{correct x-coordinate}|}{\text{correct radius}} * 100$$

$$\text{y-coordinate accuracy} = 100 - \frac{|\text{extracted y-coordinate} - \text{correct y-coordinate}|}{\text{correct radius}} * 100$$

Figures 6.13 and 6.14 show the accuracy of radius and centre coordinate extraction for circles of varying radii and colours.

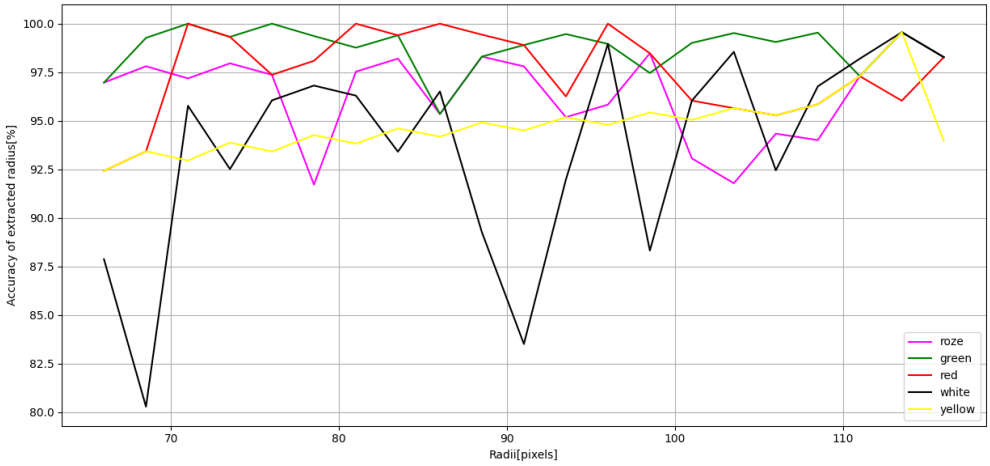


Figure 6.13: Radius accuracy concerning different radius lengths [pixels]

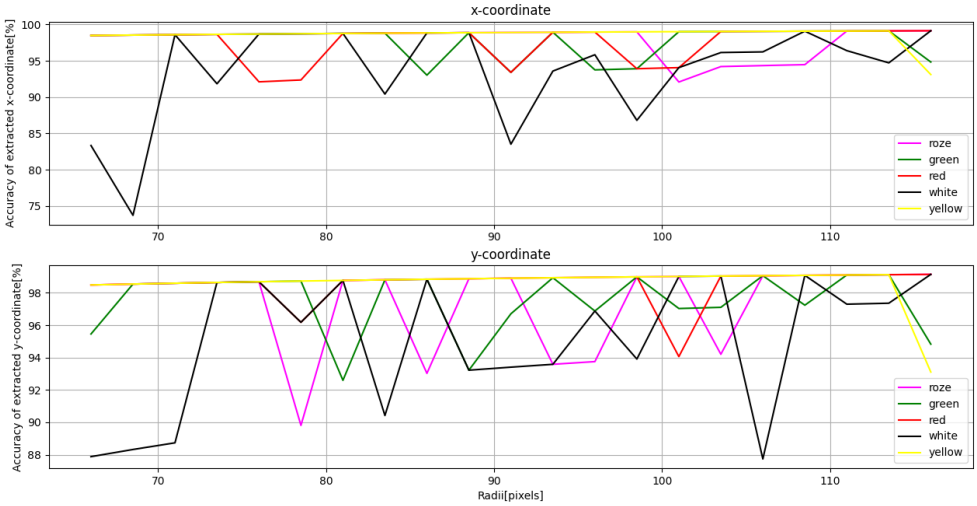


Figure 6.14: Centre coordinates accuracy concerning different radius lengths [pixels]

The results indicate that:

- The radius extraction accuracy for green and red circles is above 95%.
- White circles exhibit the lowest and most unstable accuracy.

- Excluding white circles, the radius extraction accuracy is at least 92%, with an average of approximately 96%.

For centre coordinates:

- Yellow circles have consistently high accuracy.
- Different colours yield varying accuracies without a clear bias.
- Excluding white circles, the accuracy for x-coordinate extraction is at least 92% and for y-coordinate extraction, at least 90%.
- The average accuracy for both coordinates is approximately 96%.

Influence of visibility on performance

The influence of occlusion was assessed, comparing it to the radius, x-coordinate and y-coordinate accuracy. The displacement percentage was calculated using:

$$\text{displacement}[\%] = \frac{\text{displacement}[\text{pixels}]}{2 * \text{radius}} * 100$$

Figures 6.15 and 6.16 show the impact of occlusion on the reconstructed radius and centre accuracy.

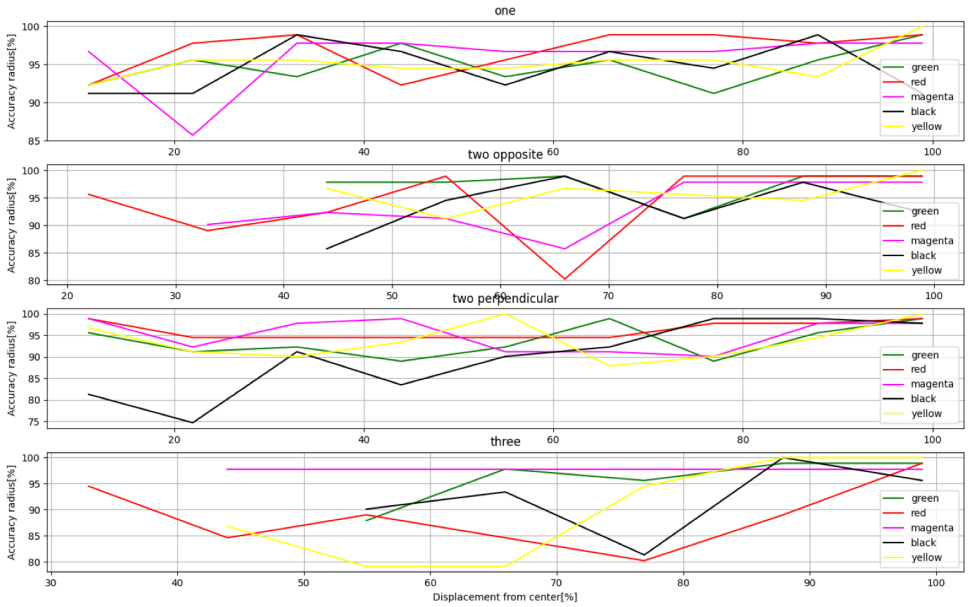


Figure 6.15: Influence on the amount of occlusion on the reconstructed radius accuracy

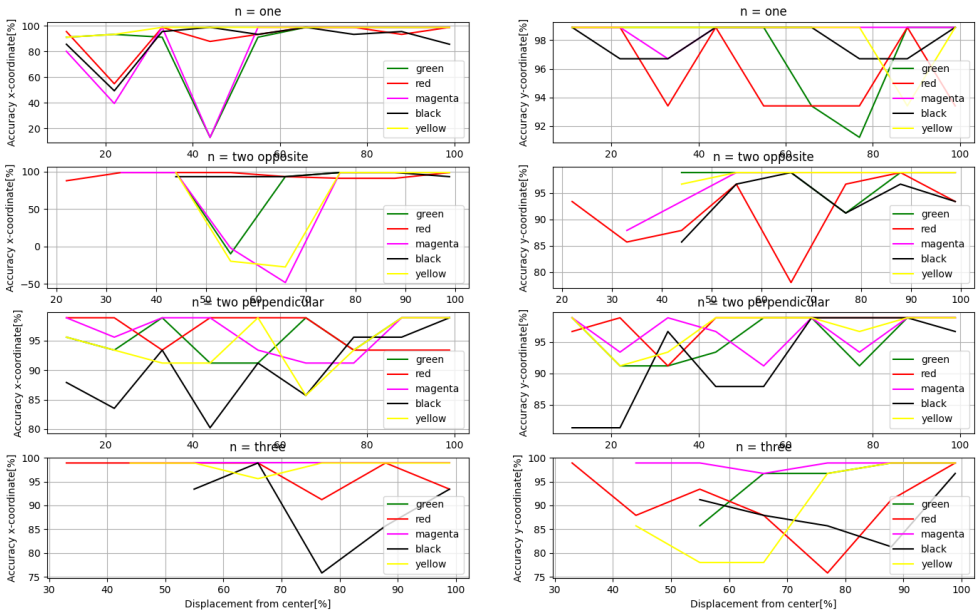


Figure 6.16: Influence on the amount of occlusion on the reconstructed centre accuracy

Key observations include:

- A greater visible area of the sphere leads to higher radius extraction accuracy.
- For centre accuracy, a higher visible area improves accuracy.
- The accuracy shows significant variance when three occluding spheres are present.
- Minimal accuracy for visible parts is 80%.

Influence of noise on performance

The effect of noise, such as glare and shadows, was analyzed. Figures 6.13 and 6.18 display the radius and centre accuracy in noisy conditions.

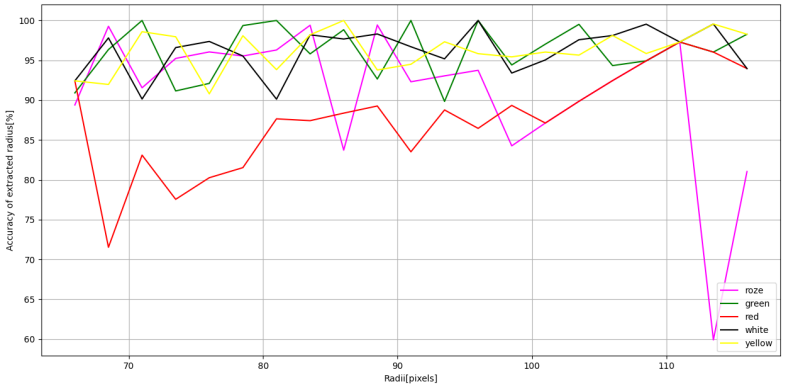


Figure 6.17: Radius accuracy concerning different radius lengths [pixels] and the influence of unique sphere colours with added noise

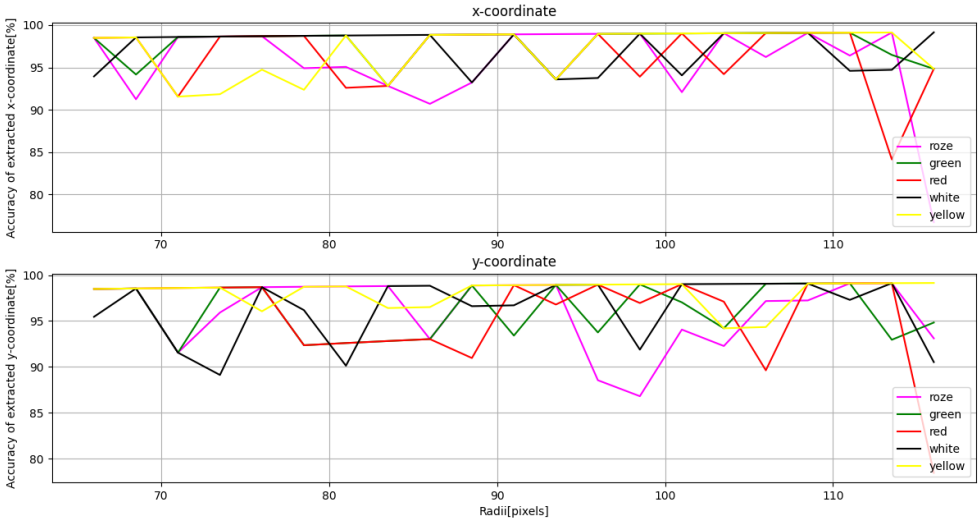


Figure 6.18: Centre accuracy concerning different radius lengths [pixels] and the influence of unique sphere colours with added noise

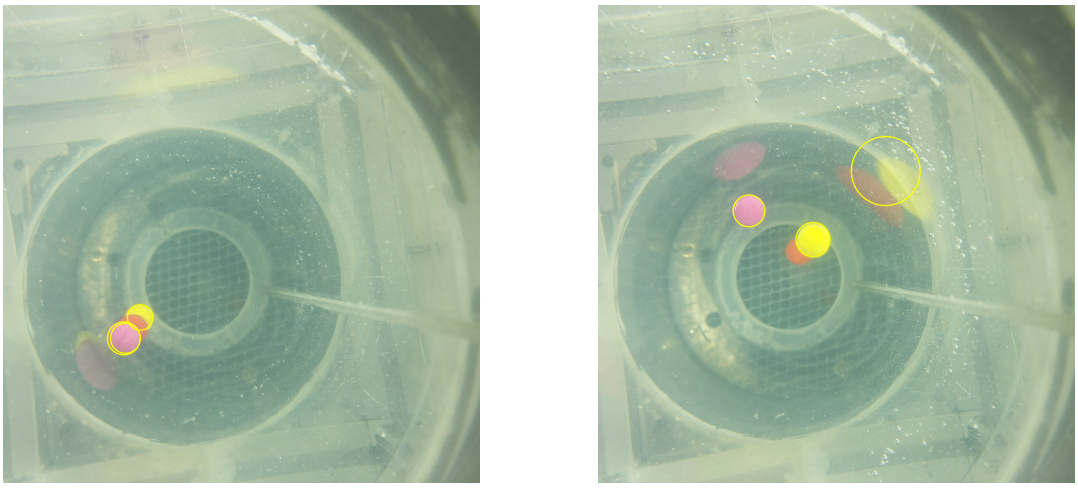
Findings are:

- Noise reduces radius accuracy by 2% on average.
- The x-coordinate accuracy remains similar to noise-free conditions, averaging 96%.
- The y-coordinate accuracy decreases by 1%, averaging 95%.
- Radius accuracy drops to 94% on average.

6.5.2. Images from test setup

From a set of 750 images, 63 were qualitatively assessed to determine an estimation of the accuracy. There were three main types of errors:

Wrong detection of Sphere

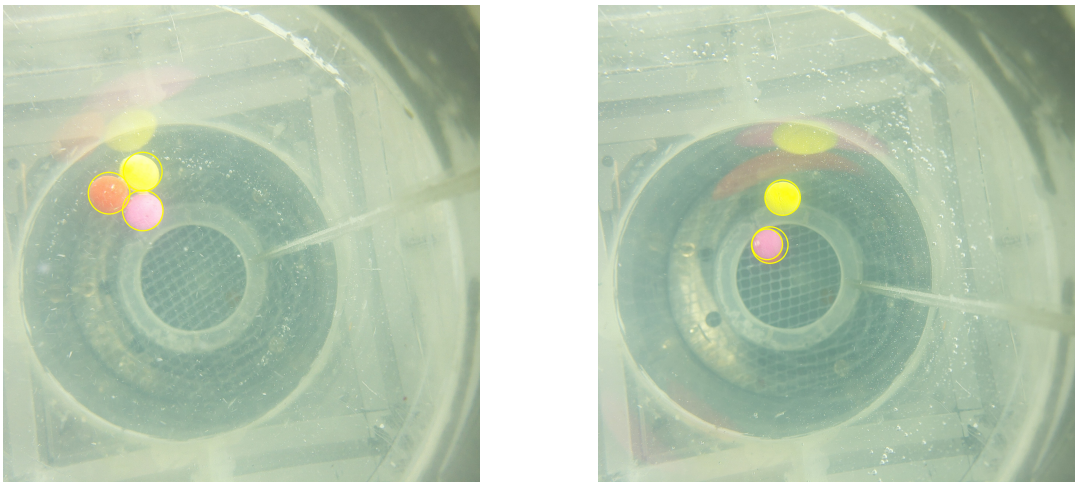


Test Image Line Test Image None

Figure 6.19: Wrong Detection of Entire Sphere

In Figure 6.19 The algorithm completely missed the correct detection of the sphere, resulting in a 100% wrong localization.

Inaccuracy in Radius and Centre Point

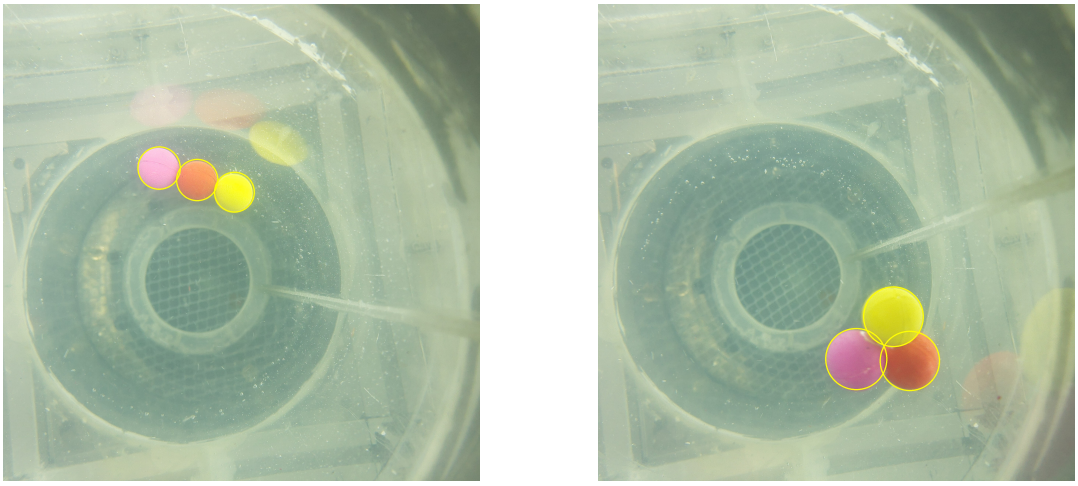


Test Image Ring Test Image None

Figure 6.20: Inaccuracy in Detection of Sphere

Figure 6.20 show inaccuracies of the determined centre coordinates and radius of the reconstructed sphere.

Accurate Detection of Sphere



Test Image Line Test Image Ring

Figure 6.21: Accurate Detection of Spheres

Figure 6.21 depicts accurately reconstructed spheres.

The average inaccuracies and standard deviation are calculated using the tables found in Appendix E.1, E.2 and E.3. The results can be found in Table 6.1.

		Line	Ring	None	Combined
Radius	Average Inaccuracy [%]	1.0	2.0	1.7	1.6
	Standard Deviation [%]	3	4	5	4
Centre Point	Average Inaccuracy [%]	0.1	0.2	0.3	0.2
	Standard Deviation [%]	1	1	2	1

Table 6.1: Inaccuracies [%] of the Test Images

6.5.3. Bias and Error

To determine possible bias and errors, dine images were taken from the same sphere formation by turning off the water upflow, causing the spheres to settle at the bottom of the cone in a net without movement. Features were extracted from these images and compared using standard deviation. Results are summarized in Appendix E.1.8. The average standard deviation for the x and y axes is 4.1 pixels. The standard deviation of the radius per colour is as follows:

- Red: 4.66 pixels
- Green: 4.19 pixels
- Blue: 3.87 pixels
- Yellow: 3.16 pixels
- Black: 4.72 pixels
- Pink: 3.18 pixels

These results indicate that the centre point lies within a box defined by these standard deviations and that the algorithm maintains a relatively consistent accuracy across different colours, with some variation due to noise and occlusion. One of the reconstructed images can be found in Figure 6.22.

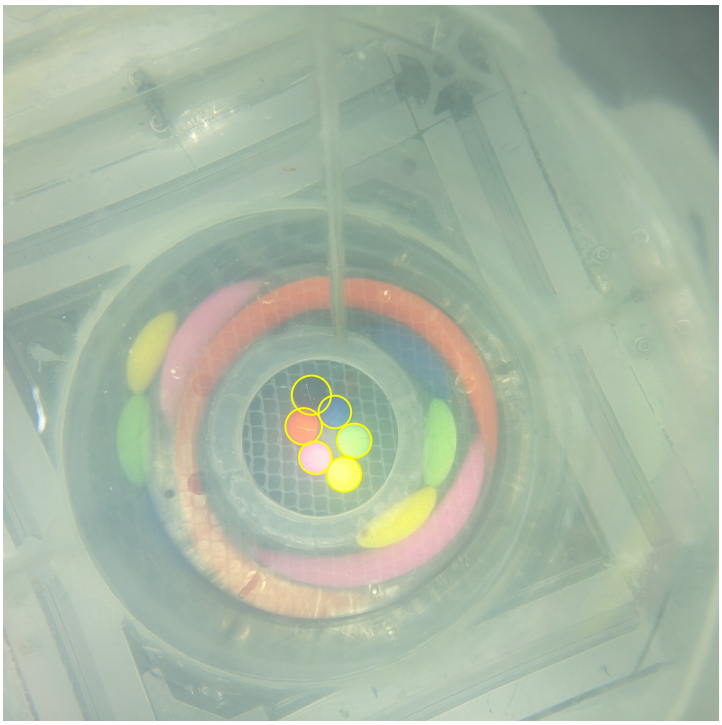


Figure 6.22: Bias and Error Determination

6.6. Discussion

The discussion expands on the analysis of the applied methods and highlights potential improvements for future research. Recommendations are provided to enhance the developed methods and address current limitations.

6.6.1. Sphere isolation

The current method for sphere isolation, based on colour thresholding, generally performs well on real images produced by the Hardware & Test Setup team. Each sphere is assigned a different colour to facilitate efficient computational isolation.

- **Shadow Interference:** When a sphere has a significant shadow, the isolation algorithm struggles

to detect and isolate it accurately. This issue reduces the robustness of the current method under varying lighting conditions.

- **Colour Limitation:** The range of usable colours is limited due to potential overlap in colour threshold values, which restricts the number of distinguishable spheres.
- **Background Colour Bias:** The detection of green spheres in the test images, was notably problematic, probably due to the greenish hue of the background. This colour interference significantly reduced the detection accuracy for green spheres. Figures 6.23 shows an image with correct detection of the green sphere, however in most cases the detection is more like in Figure 6.24. The green sphere was omitted during further experimentation.
- **Background Noise:** Due to unexpected background noise from the alternative camera setup (top view), the background removal function sometimes struggles to correctly identify and remove the background. This inconsistency leads to the deletion of important features in some instances and the retention of excessive background in others, causing significant noise in colour isolation (prior testing was only done from the side view as the camera needed for the top view had not yet arrived).

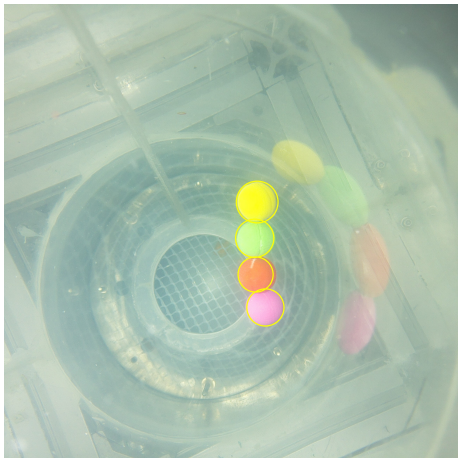


Figure 6.23: Correct detection of the green sphere

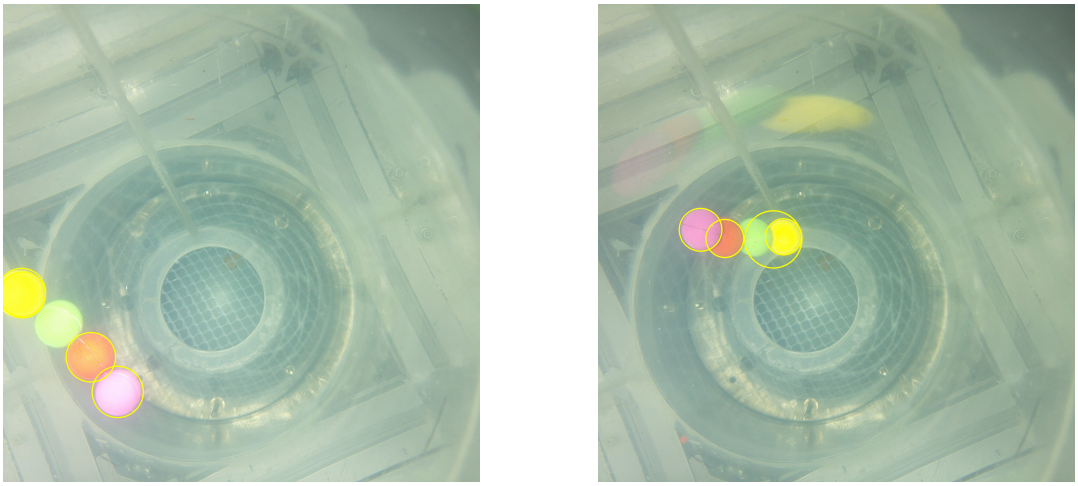


Figure 6.24: Incorrect detection of green sphere

To overcome these limitations, future research could explore algorithms that extract sphere contours based on shape and edges rather than relying solely on colour. Implementing shape-based methods

would make the algorithm more aligned with the generalisability requirement 3.4 in Section 2.3, allowing for the detection of more spheres and diverse structures. A promising approach is the development of algorithms based on concave point detection and Gaussian processes, which were preliminarily researched but not fully implemented due to time constraints [23].

6.6.2. Sphere reconstruction

The developed algorithm employs the Hough Transform to extract the centre and radius of spheres. The Hough Transform was selected for its robustness and general accuracy in most cases. However, several issues were noted:

- **Fluctuating Accuracy:** The accuracy of the extracted radius and centre can vary, with some images yielding less reliable results. Initially, an agreement was made that the sphere radius size would be between 66 and 117 pixels to limit noise in the algorithm. However, in practice, the sphere radii now range between 50 and 150 pixels, making the algorithm more prone to errors. This was noted late in the project due to the final test images being generated only at the end of the project.
- **Processing Time:** The image processing algorithm takes approximately 6 seconds per image, with the Hough Transform being a significant contributor to this duration. For large datasets, this results in relatively long processing times.
- **Minimum and Maximum radius:** The minimum and maximum radius were first calculated based on the visible part that could be captured by a camera horizontally from the side resulting in a height range of 20 cm. However, with the camera vertical on top of the tube, the minimum and maximum height of the sphere that could be captured with the camera was significantly higher.Therefore, also the minimum and maximum radius of the captured spheres were lower and higher, respectively. The minRadius and maxRadius values of the Hough Transform were adjusted accordingly. Resulting in less accurate results of the Hough Transform, because the Hough Transform can detect more false positive circles.

Future research should investigate alternative algorithms that could provide more accurate and computationally efficient computation of radii and centre coordinates. Optimization techniques may offer improvements in both accuracy and processing time.

6.6.3. Characterizing depth

Currently, depth is inferred from the computed radius using a single-camera setup. This method has inherent limitations:

- **Accuracy Measurement:** Without knowing the exact correct centre and radius of the image, the accuracy of the centre and radius extraction algorithm can only be visually assessed. Adding reconstructed circles to the original images provides a rough measure of accuracy.
- **Single-Camera Limitation:** Using a single camera restricts depth computation accuracy. Employing two cameras with perpendicular observation angles could enhance accuracy by capturing x, y, and z coordinates. However, this approach doubles the data and computation time, potentially conflicting with project requirements.
- **Light attenuation:** During testing it was found that spheres located at the bottom or lower part of the testing tube, showed a glare and reduced contrast with the background due to the light attenuation of the water.
- **Reflection:** The presence of reflections in conical surfaces complicates the accurate detection of the sphere's location, see Figure 6.25.
- **Bottom of the Cone:** The net at the bottom of the cone introduces additional noise, as the camera's focus on the net results in clear images that background removal algorithms misidentify as foreground, sometimes deleting the spheres instead of the net, see Figure 6.25.

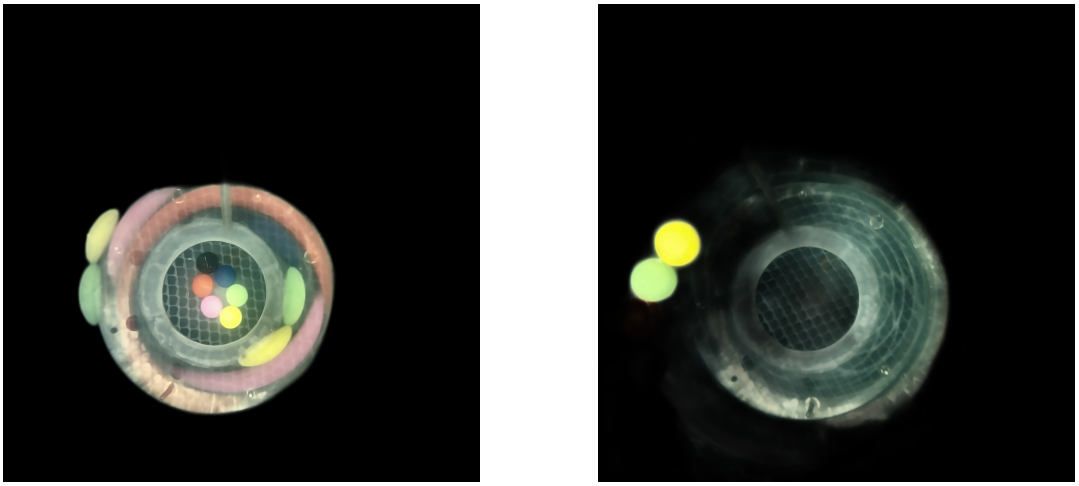


Figure 6.25: Incorrect Removal of the Background

Further research could focus on optimizing dual-camera systems to balance accuracy and efficiency. Additionally, exploring depth estimation using light intensity variations could be beneficial. As observed, spheres at different depths exhibit varying light intensities due to underwater light attenuation and refraction. Establishing a relationship between light intensity and depth could offer an alternative method for depth measurement. This approach warrants further investigation to determine its practical application in improving overall system accuracy.

6.6.4. Future Recommendations

To build on the current work, the following recommendations are proposed:

- **Advanced Sphere Isolation Techniques:** Develop and test shape-based sphere isolation algorithms to improve robustness and scalability.
- **Alternative Sphere Reconstruction Algorithms:** Explore and validate more efficient and accurate methods for computing sphere parameters, reducing reliance on the Hough Transform.
- **Enhanced Depth Characterization:** Investigate dual-camera setups and light intensity-based depth estimation [22] methods to improve depth accuracy.
- **Comprehensive Testing:** Conduct extensive testing under various environmental conditions and with different sphere configurations to ensure the robustness of the developed methods.
- **Background and Color Interference Mitigation:** Address the issue of colour interference from backgrounds by adjusting image capture conditions or applying advanced colour filtering techniques.
- **Background and Color Interference Mitigation:** Address the issue of colour interference from backgrounds by adjusting image capture conditions or applying advanced colour filtering techniques.

In summary, while the current image processing system shows promise, addressing the identified limitations and implementing the recommended improvements will enhance its robustness, accuracy and applicability in diverse practical scenarios.

6.7. Conclusion

The goal of the image processing sub-team was to design an algorithm capable of automatically computing the radius, centre coordinates, and visibility of spheres from an image of a structure of magnetic spheres in water. Throughout the project, the design of the algorithm was iteratively refined based on the program of requirements. The final algorithm isolates each sphere into a binary file from which

the radius and centre are reconstructed. Using these parameters, the circle is reconstructed using the Hough transform, and the visibility is computed using the Jaccard Similarity Coefficient (JSC).

6.7.1. Results and Performance Evaluation - Generated Images

Preliminary validation of the image processing algorithm with generated images was necessary due to the late availability of real images. Generated images allowed for mathematical computation of the accuracy of radius, centre coordinates, and visibility. Key performance metrics derived from these tests are:

- **Sphere Detection:** The algorithm is able to successfully detect generated circles.
- **Feature Extraction:** The algorithm accurately computes the centre coordinates, radius, and visibility for fully visible spheres and those partially occluded up to a specific threshold.
- **Accuracy:** For fully visible spheres, the average radius and centre coordinate accuracy is 96%. For five and six spheres, the required accuracy threshold of respectively 96% and 95.4% is met, confirming that the algorithm reconstructs the radius and centre within acceptable limits. However, for three or four spheres a minimum accuracy of respectively 98.2% and 96.1% must be achieved. Therefore, when three or four spheres are used for the experiment, the current developed algorithm is, based on generated images, not accurate enough.
- **Noise Resistance:** Noise introduced into the generated images reduced radius accuracy by 2% and centre accuracy by 1%. When occluding spheres are at least 50% displaced from the centre of the occluded sphere, the noise still results in an average accuracy of 95%. Although assessing actual noise resistance is challenging with generated images alone, these findings suggest plausible robustness against noise.

6.7.2. Results and Performance Evaluation - Test Images

The proposed image processing system was further validated with real images from the Hardware & Test Setup team. Although a comprehensive analysis of a large dataset was not possible due to late image availability, the evaluation of a smaller set provided valuable insights:

- **Sphere Detection:** Out of 63 inspected images containing 3 spheres per image, 187 spheres were detected accurately, achieving a detection accuracy of 98.9%.
- **Feature Extraction:** For the 187 correctly detected spheres, the average accuracy of the computed centre coordinates was 99.8% and radii was 98.4%.
- **Data Output:** The output data, containing all features, is saved as a `.h5` file.
- **Multi-Sphere Computation:** The system effectively handled up to 6 spheres per image.

6.7.3. Alignment with sub-team specific and top-level relevant Requirements

The image processing system aligns well with the Program of Requirements (PoR) specified in Chapter 6.2:

- **Accuracy:** The system achieves an overall accuracy of 98.9% for sphere detection, 99.8% for correct centre determination and 98.4% for correct radius determination. All lie well above the 95% accuracy requirement.
- **Self-Contained Algorithm:** The algorithm operates independently without the need for external applications, ensuring reliability and ease of integration.
- **Iterative Sequential Performance:**The system consistently performs multiple sequential processing tasks, maintaining performance across large image sets.
- **Computation Time:** Average processing time is maintained at 6-7 seconds per image for up to six spheres, this is time-intensive for large data sets. If the experiment runs 12 hours and every second an image will be taken, then the total processing time of processing this data is 72 hours of image processing not adhering to the requirement of the total processing time of a set of images must be less than 2 hours.
- **Automated Process:** The system is fully automated, requiring no manual intervention during the feature computation process, which enhances usability and efficiency.

- **Noise Resistance and Sphere Detection:** The model effectively removes noise and accurately distinguishes sphere colours, achieving a detection accuracy of 98.9%.
- **Feature Extraction:** The system determines the position, radius, and visibility of each sphere with an average accuracy of 99.8% for centre coordinates and 98.4% for radii.
- **Data Management:** Input and output file formats are handled as specified, with processed features outputted in a format compatible with Deep Learning applications.
- **Validatability:** The algorithm's performance can be visually verified for sphere detection.
- **Generalisability:** The algorithm can detect a maximum number of distinct spheres based on color differentiation, within threshold margins.

6.7.4. Recommendations

Based on the results and performance evaluation, the following is recommended:

- **Enhancing Detection Accuracy:** Further refinement of the detection algorithm to improve the accuracy and reliability of sphere detection, especially under varying environmental conditions. Exploring more flexible and generalizable detection methods to get rid of the colour variation limitation.
- **Expanding Testing Scenarios:** Conduct additional tests with different quantities and configurations of spheres to ensure robustness and generalisability, targeting an accuracy variation within ±1
- **Optimizing Computation Time:** Investigate methods to further reduce computation time, aiming for a processing time of less than 5 seconds per image, enhancing efficiency for real-time applications.
- **Improving Feature Extraction:** Enhance the accuracy of feature extraction, particularly the computation of centre coordinates and radii, aiming for an accuracy improvement to 99%.
- **Validation Method:** Determine a more accurate validation method that does not rely solely on visual assessment.

In conclusion, the proposed image processing system demonstrates significant potential in achieving high accuracy and efficiency in sphere detection and feature extraction. By addressing the identified areas for improvement, the system can be further optimized to meet the demands of practical applications and ensure seamless integration with Deep Learning algorithms for formation identification.

7

Deep Learning

7.1. Introduction

To classify the formations, a deep learning algorithm (a multi-layer perceptron) was chosen. Deep learning, a subset of machine learning, generally utilizes multi-layered neural networks to decipher complex patterns in data. This technique automatically identifies features through training, facilitating tasks such as image recognition and natural language processing without the need for manually programmed feature detection.

The algorithm processes a dataset comprising coordinates and radii of each formation, training to detect patterns by interpreting complex correlations within the data. Given the complex non-linear relationships between random formations suspended in water and their 2D plane projections captured by a camera, deep learning is an effective method for handling such complexities.

Due to timeline constraints, the deep learning sub-team has assumed responsibility for creating accurate simulations of the tank and suspended spheres. This proactive approach reduces the dependency on the progress of other sub-teams. By doing so, the deep learning team can generate its data and independently develop its model in parallel with the ongoing work of other groups, ensuring continuous progress without delays. This approach also has the potential to generate data in many more formations than might be feasible when artificially glueing spheres together to form formations for creating training data.

The deep learning pipeline will function as follows:

1. Simulate formations to generate data.
2. Create a 2D projection from the simulation from the perspective of a camera.
3. Extract x, y-coordinates, and radii information from the projection.
4. Transform the data so each coordinate is relative to one of the spheres in the formation and normalize it.
5. Train the model on this transformed data.
6. Use the trained model to make predictions on unseen data.

7.2. Program of Requirements

For deep learning, there are two sets of requirements specific to the subsystem: requirements for the simulation and requirements for deep learning itself.

7.2.1. Simulation Requirements

A simulation is required due to timeline constraints, as the neural network heavily depends on training data that cannot be collected until other processes are completed. The simulation must be as accurate as possible so that when actual data is collected, the model won't require significant adjustments,

rendering the initial tuning effort futile. The simulation must be based on existing research but cannot be fully validated without real data, however, it is challenging to quantify the accuracy of the simulation as it encompasses many dynamics. Alternatively, if the simulation is very accurate, future work might be able to generate more complex formations without needing to collect actual data through cumbersome methods such as glueing or tying formations to maintain their shape.

1.1 **Realness:** The simulation must be as realistic and true-to-life as possible. (This requirement is hard to give a value to as it encompasses many dynamics.)

1.2 **Generation Count:** The simulation must be able to generate at least 2500 to 3000 formations for each type of formation (ring/line/undefined).

1.3 **Generation Centre Points:** The simulation must efficiently generate data points (the centres of the spheres) in three-dimensional space.

1.4 **Relative Position:** The simulation must provide the data points relative to each other to ensure independence from the calibration of the coordinate system.

1.5 **Generation Time:** The generation of data should not take longer than 15 minutes.

7.2.2. Deep Learning Requirements

2.1 **Accuracy:** The deep learning submodule must achieve at least 95% accuracy, as specified by the functional top-level requirement for accuracy, (see Section 2.3).

2.2 **Noise Resistance:** The model trained on simulated data must be resistant to noise up to 5% (see Section 6.5).

2.3 **Required Data:** The model must process coordinates and radii per the top-level requirement for required data (see Section 2.3).

2.4 **Non-GPU Training:** To reduce computation time, the model must be trainable on a laptop without a GPU, as required by the top-level requirement for non-GPU training and training time (see Section 2.3).

2.5 **Multi-Sphere Computation:** The algorithm must isolate and compute the required parameters for a variable number of spheres, in line with the top-level requirement for multi-sphere computation (see Section 2.3).

2.6 **Training Time:** The model must be trained within 24 hours, as per the top-level requirement for training time (see Section 2.3).

7.3. Design Choices

As discussed in Chapter 3, one of the alternative top-level designs required the neural network to be trained on image data instead of coordinate data. This approach would have increased both the physical size of the data and the computation time for training, making it unfeasible without using a graphics card, thus violating the graphics requirement in Section 7.2.2. Additionally, this approach would have necessitated training multiple networks in different amounts of spheres.

After deciding to use a hybrid approach (coordinates/radii extraction before training), data processing became much more manageable. However, to fulfil the multi-sphere requirement in Section 7.2.2, multiple models still need to be trained.

Another important design decision was the architecture of the simulation. It was crucial for the simulation to be completed as quickly as possible since the major bottleneck was generating data without having any of the other processes finished. Without a camera setup, a database, an image processing system, and a coordinate extraction algorithm, the deep learning sub-team could not begin tuning and training its models. Therefore, it was decided that a simulation was necessary.

The simulation could take one of two approaches: generating spherical formations by creating images in 3D space and projecting the spheres onto a 2D plane, or directly generating spheres on a 2D plane. While the latter approach sounds simpler, key considerations included the ease of validating the model's accuracy and simulating the spatial constraints of the tank. A 3D simulation facilitates easier plotting

of probability density and more accurate simulation of spatial constraints, which would be challenging with a direct 2D projection.

The accuracy of the model is crucial, as the training and tuning of the model depend heavily on it. If the model is not sufficiently accurate, the tuning process would need to be completely revised, potentially altering its characteristics. Therefore, the decision was made to generate spheres in three-dimensional space. This approach not only ensures better accuracy but also holds significant potential for future expansion, allowing for the generation of more complex structures without needing to run actual experiments and glue formations together.

In the field of machine learning every decision is a consideration between computational efficiency and accuracy of the model. Models like Multi-Layer Percetrons (MLP's), Convolutional Neural Networks etc, have a lot of variables to tune. With enough time and a fast enough computer any variable can be tuned to gain a perfect accurate model, however there is not enough time to do so. Therefore a significant decision was determining the deep learning architecture—specifically whether to use multiple classifier models, a single model utilizing a "wisdom of the crowd" approach, or a single expert model. To aid this decision, preliminary tests were conducted to evaluate various models, both linear and non-linear, including neural networks. These tests aimed to provide a preliminary understanding of the models' accuracy before proceeding further. Although the simulation had already been developed, programming the simulation took longer than expected, necessitating an accelerated timeline for developing the proper architecture. Nevertheless, sufficient care was taken in the evaluation of the architecture. The comparison uses the following models (see Table 7.1). These models were implemented using the Scikit-learn library, as it was the library with which the sub-group had the most experience [16]. Due to time constraints, it was decided to use Scikit-learn despite its limited flexibility for deep learning models.

Model Type	Description	Good At	Not Good At
SGDClassifier	A linear classifier (e.g., linear SVM or logistic regression) optimized using stochastic gradient descent.	Large-scale datasets, high-dimensional spaces, online learning, fast training.	Non-linear relationships, small datasets, handling noisy data without careful tuning.
SVC	A classifier that finds the hyperplane that best separates the classes, using kernel functions for non-linear decision boundaries.	Non-linear classification with kernel trick, high-dimensional feature spaces, and clear margin of separation.	Large datasets (computationally intensive), handling many classes without significant tuning.
NuSVC	A variant of SVC that uses a parameter ν to control the number of support vectors and the margin.	Similar advantages to SVC, more control over the number of support vectors, effective for imbalanced classes with proper tuning.	Same limitations as SVC regarding large datasets require careful tuning of the ν parameter.
RandomForestClassifier	An ensemble method using multiple decision trees to improve classification accuracy.	Handling high-dimensional spaces, robustness to overfitting, dealing with missing values, and interpretability.	Requires more computational resources, can be less effective on very noisy data, and may not perform well on datasets with a very small number of features.
XGBoost	An efficient implementation of gradient boosting for decision trees.	Handling high-dimensional data, excellent predictive accuracy, handling missing values, regularization to prevent overfitting, fast training with parallel processing.	Requires careful tuning of hyperparameters, computationally intensive for very large datasets.
MLPClassifier	A neural network-based classifier with one or more hidden layers.	Learning complex, non-linear relationships, handling high-dimensional data, and flexibility with various architectures.	Requires a lot of data to avoid overfitting, computationally expensive and slower training, hyperparameter tuning can be complex and time-consuming.

Table 7.1: Comparison of Different Models

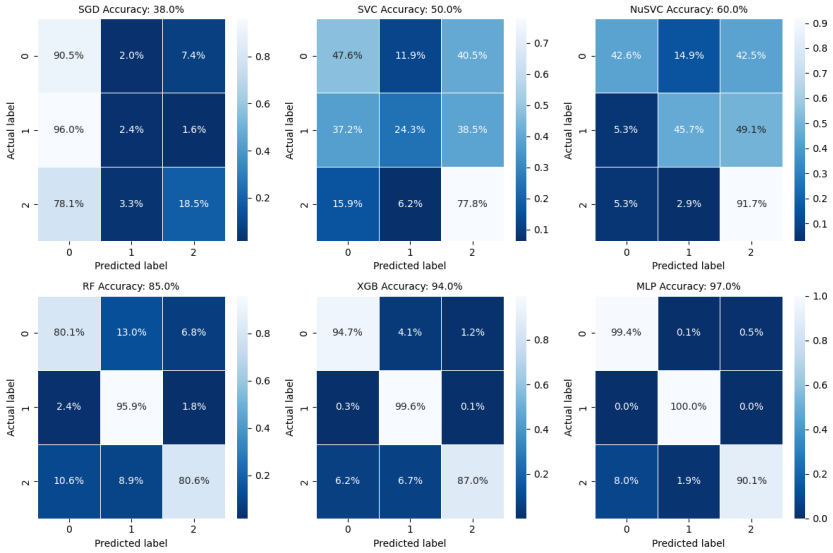


Figure 7.1: Confusion matrix model comparison

Having evaluated the accuracy of these non-optimized models and considering the secondary requirements for the architecture, it was found that models like XGB and MLP already show promising accuracy. This suggests that a single expert model could achieve sufficient accuracy while being computationally less expensive than training multiple models. This advantage becomes even more significant when considering that multiple architectures would need to be trained for different amounts of spheres potentially violating the computation time requirement, see Section 7.2.2. Therefore, the decision was made to implement a single MLP architecture.

7.4. Implementation

This section discusses the implementation and design choices made in creating the simulation and deep learning algorithm.

7.4.1. Simulation Implementation

The simulation began simply by generating points in 3D space. Since the simulation needed to generate points relative to each other, it could proceed in one of two ways: randomly generating formations in 3D space or generating points on a 2D plane and then randomly translating and rotating the formation. Randomly generating formations in 3D space is a more complex task, as it involves generating multiple points in random directions relative to each other while spatially restraining consecutive points to form a line or ring formation within the tank's dimensions.

It is much easier to first generate points in a specific formation and then randomly rotate and translate the entire formation. Therefore, this approach was adopted. A point is first generated at the origin, after which consecutive points are generated in a formation and then rotated and translated.

To handle these calculations, several algorithms were developed and utilized. The file 'generatePoints.py' generates consecutive points based on initialization parameters. It calculates the minimum allowed angle between the vectors of consecutive sphere centres to form a ring of spheres, effectively determining the angle between vertices of regular polygons. Using this angle and the sphere radius, it calculates the centre coordinates of the next sphere. To ensure the next point is generated in the correct location, a 2D rotational matrix is multiplied with a set of 2D coordinates where y equals zero.

This results in the following equation:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} 2R \\ 0 \end{pmatrix} = \begin{pmatrix} 2R\cos\theta \\ 2R\sin\theta \end{pmatrix} \tag{7.1}$$

A similar approach was taken when randomly rotating using 3D rotational matrices in their simplest form. However, for random translations, the simulation needed to be highly accurate (to fulfil the realness requirement in Section 7.2.1). To achieve this, the observed distribution of the spheres when suspended in the tank was utilized. Specifically, a particle has a 26% chance of being in the upper half of the tank, a 48% chance of being in the positive x-axis half of the tank, and a 62% chance of being in the positive y-axis half of the tank [12]. This distribution was implemented using a simple if statement combined with a uniform random distribution. This characteristic distribution is evident when plotting thousands of these formations, as shown in Figure 7.12.

When the formation is generated and randomly transformed, it is crucial to check whether it remains within the boundaries of the cone. The file 'generateShapes.py' generates all necessary components to calculate the distances from sphere centres to the tank's boundaries. It includes both the mathematical description of the tank (a frustum of a cone) and its physical dimensions.

Efficiently determining whether a sphere collides with the tank is essential, as this check must be performed for every point in the thousands of formations generated (in alignment with the generating data requirement in Section 7.2.1). A strategy like ray tracing is impractical due to computational intensity. Instead, the frustum of a cone with a specified height can be described as the set of all points traced by a line segment rotated about a fixed axis, where the endpoints of the segment trace two circles with specified radii. Using this approach, each generated point can be evaluated by comparing it to the nearest line segment that defines the frustum, ensuring it remains within the tank's boundaries.

To find the distance from a point to a line segment on the cone, it is simplest to first determine the corresponding points on the circles traced by the rotated line segment. This is done by calculating the angle the point makes with the x-axis when projected onto the XY-plane. Using this angle, the 3D coordinates of the endpoints of the line segment can be found, given the measured height and radii of the cone.

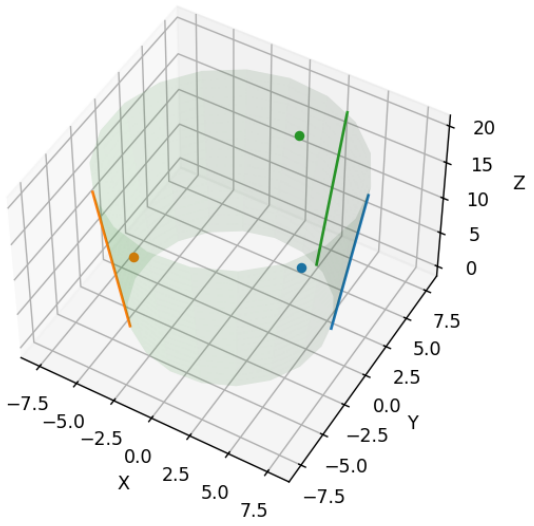


Figure 7.2: Points and their closest line segment describing the cone

With three points calculated—one representing the centre of a random sphere and two representing the endpoints of the line segment on the cone—the next step is to project the sphere centre orthogonally onto the vector spanning the line segment of the cone. This can be achieved using the projection formula shown in Equation 7.2, where $\mathbf{u}$ is the vector of the line segment onto which the point is projected:

$$x_p = \frac{x \cdot \mathbf{u}}{\mathbf{u} \cdot \mathbf{u}} \mathbf{u} \tag{7.2}$$

Having calculated the orthogonal projection, the distance to the cone is now equal to the magnitude of the vector spanning from the projection to the centre of the randomly generated sphere.

Since the positions of the spheres will be captured using a camera, the spheres will be projected onto a certain plane. This implies that the radii of the spheres will depend on their distance from the lens. Consequently, the file ‘projectPoints.py‘ calculates the perceived radius and converts all coordinates into pixels.

The ‘generateData.py‘ script generates tens of thousands of formations and formats them into HDF5, enabling rapid training and evaluation of the deep learning algorithm, meeting the computation time requirements in 7.2.1.

7.4.2. Deep Learning Implementation

As discussed in Chapter 7.3, the decision to use a neural network was made quickly due to time constraints. The next step involved tuning the neural network and evaluating its performance.

To begin, it was essential to estimate the amount of data required for training to achieve accurate predictions on new data. This estimation was done by initially generating 15,000 random formations, which was deemed sufficient. A model was then trained using normalized data following a relative coordinate transformation (relative coordinate transformation will be explained later). The training was conducted using k-fold cross-validation, iteratively decreasing the training dataset size and calculating the accuracy of unseen data. This process resulted in the graph shown in Figure 7.3. The x-axis is inverted to better illustrate the effect of iteratively decreasing the training dataset size.

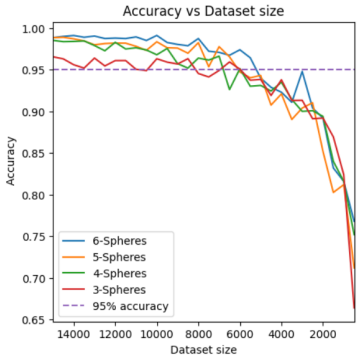


Figure 7.3: Accuracy with relative coordinate transformation depending on the size of training set

With this information, the other sub-teams now have clear requirements regarding the amount of data they need to collect, from which the requirement in 7.2.1 sprouted. To further evaluate the performance of the neural network, the accuracy was plotted against the number of nodes in a network with one hidden layer. For every dataset, a model was trained using k-fold cross-validation while incrementing the number of nodes to determine the accuracy and when the model converged to a stable accuracy. This resulted in the following plots (see Figure 7.7).

The three graphs in Figure 7.7 correspond to different transformations of the data to make it translation-invariant. Figure 7.4 shows the relationship between the number of nodes and the accuracy of the network when trained with unaltered data. The figure next to it (Figure 7.5) shows the same relationship, but this time the coordinates of each formation are not absolute but relative to each other, with one sphere being the origin to ensure the data is translation-invariant. In Figure 7.6, this concept is applied differently: instead of using relative coordinates, the angle between vectors spanning consecutive points is calculated.

The relative coordinate transformation subtracts the coordinates of the first sphere in the formation from every other coordinate within the formation.

The relative angle transformation generates a 1D vector of x and y coordinates, creates another set of vectors shifted by one element, and then takes the point-wise difference between the corresponding vectors. This process generates two one-dimensional vectors that hold the directional information from one point to another in the x and y directions. By combining these vectors and using the definition of the dot product (Equation 7.3), the angle between them can be calculated.

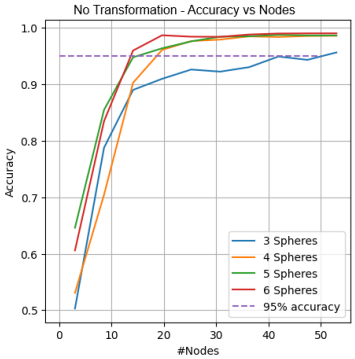


Figure 7.4: Accuracy depending on the number of nodes in hidden-layer with non-transformed training data

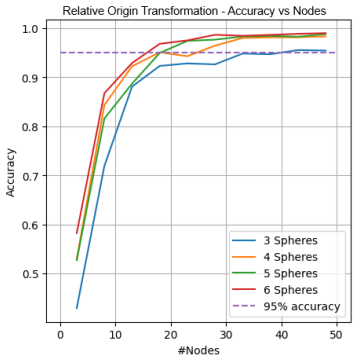


Figure 7.5: Accuracy depending on the number of nodes in hidden-layer with relative-coordinate-transform training data

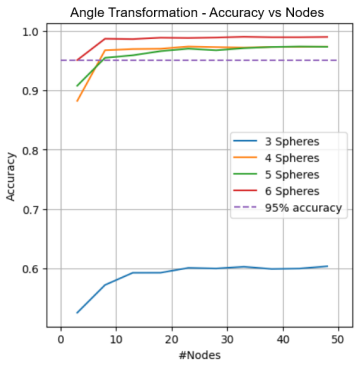


Figure 7.6: Accuracy depends on the number of nodes in the hidden layer with relative-angle-transform training data.

Figure 7.7: Accuracy depending on amount of node in hidden layer

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos(\theta) \tag{7.3}$$

Looking at the graphs, none of them converge to 100%. Figures 7.4 and 7.5 exhibit similar characteristics, while Figure 7.6 shows distinct differences. Additionally, there appears to be a correlation between the ratio of inputs (number of spheres) and the number of nodes in the hidden layer, which affects the model's accuracy. This is evident in the initial values of the plots, where the initial accuracy increases with the number of input nodes corresponding to a larger number of spheres. If the hidden layer has 40 nodes, the accuracy declines beyond a certain number of spheres, with the threshold being around 13 spheres.

The relative coordinate transformation however seems to have a larger spread in converging characteristics with different amounts of spheres. The relative angle transformation converges much faster to a stable accuracy, but this accuracy only exceeds 95% when more than three spheres are present. This is likely because the angles between the points contain much less information compared to configurations with four, five, or six spheres. For example, consider three spheres in a line formation compared to a formation with two connected spheres and one separated sphere. It is much harder to distinguish these formations based solely on angles, as it is unclear whether the spheres are connected or in a specific formation. This is also the reason why, in Figure 7.8, the confusion matrices show a high false positive rate for undefined formations.

Therefore, the decision was made to evaluate the model and transform the data using the relative coordinate transformation. As shown in Figure 7.5, the topology of the neural network will consist of one hidden layer with 40 nodes.

Taking this topology into account, the confusion matrices for each number of spheres are shown in Figure 7.8. The network was trained using k-fold cross-validation with one layer and 40 nodes on 15,000 transformations. The confusion matrices were tested using the same dataset.

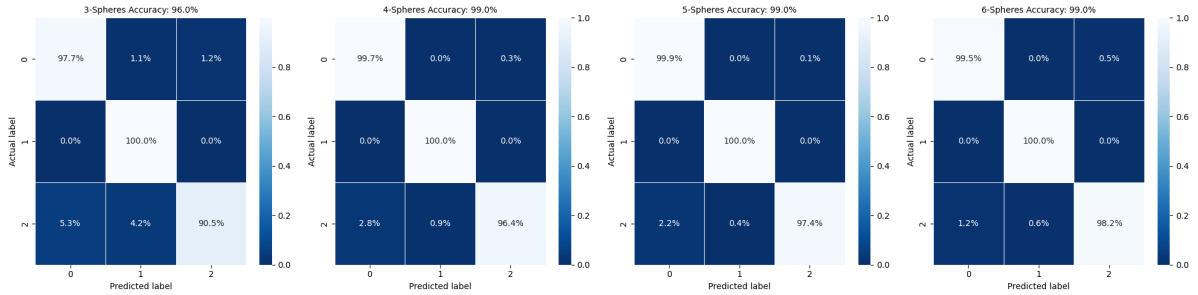


Figure 7.8: Confusion matrix per amount of spheres with relative coordinate transformation. Labels 0, 1 and 2 are line-, ring- and undefined-formations respectively.

From this analysis, it was determined that the chosen topology provides sufficient accuracy and performance to meet the requirements outlined in Section 7.2.2. The results will be discussed in detail in Section 7.5.

To determine the model's susceptibility to noise, it was first trained with 'clean' data and then evaluated on noisy data. This approach helps to understand how noisy the data will be when gathered from the actual setup using a model trained with simulated data, and conversely, how accurate the simulation is.

Noisiness is defined as the accuracy of measured centre point coordinates and radius measurements compared to the actual centre points and radii. Since the classification of the formation heavily depends on the relative coordinates of the spheres and the accuracy of the radii, a thorough understanding of the impact of inaccuracies is essential.

In this test, the model was trained on 'clean' or noiseless data and then evaluated on (newly generated) data with noise added incrementally, expressed as a percentage of the mean radius. This is done because the Noise levels ranged from 0% to 5%, applied to the x, y coordinates, and radius. Before being fed into the model, the values were normalized. This process resulted in the graph shown in Figure 7.14.

Finally, data was collected, but it was insufficient for the requirements in Section 7.2 to train the model on a new dataset (approximately 760 different formations). The data was normalized and fed into the trained model, producing the Figure 7.8.

7.5. Results

Since not enough data has been gathered, there isn't a way to compare the simulation's accuracy to real-world results. However, understanding the generated data is crucial for identifying discrepancies in the simulation. To facilitate this, three graphs have been plotted in Figure 7.12.

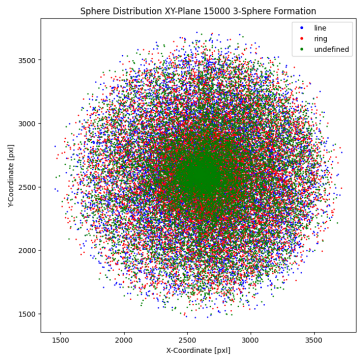


Figure 7.9: XY-Plane distribution

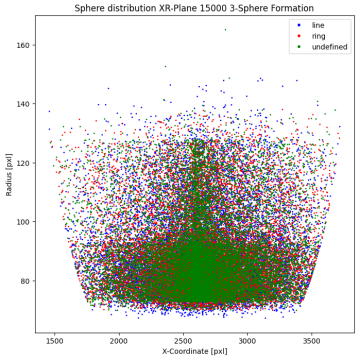


Figure 7.10: XR-Plane distribution

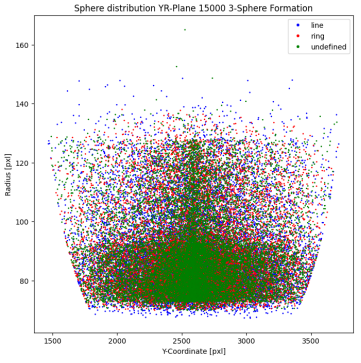


Figure 7.11: YR-Plane distribution

Figure 7.12: Distribution of Generated Formations

In the diagram, there is a clear divide between the lower and upper parts of the tank. This is intentional, as research ([12]) indicates a 26% chance of finding a sphere in the upper part of the tank. The same research also states there is a 62% chance of finding a sphere along the positive x-axis of the tank and a 48% chance along the positive y-axis. These distributions are less visible but still present in the diagrams.

The simulation attempts to generate thousands of formations accurately, but as shown in Figure 7.12, there are a few inaccuracies. For example, there are clear lines around radius values of 130 and 60 pixels, which roughly correspond to height values of 20 and 0 cm, virtually the bounds of the tank. Some formations are generated above these bounds because the formations are translated with respect to the origin. If a formation is rotated 90 degrees (with respect to the x or y-axis), some spheres in the formation might pass the 20 cm mark, but generally, they do not. There are exceptions, such as some undefined formations that end up much higher than intended. This occurs because when separate formations are generated to collide, one of the formations is randomly translated, causing its new origin to have a z-coordinate that isn't equal to zero, allowing it to be generated much higher. This is not a problem, however, since the simulation includes a limit that translates formations back down when they are generated above 30cm. Although there are some inaccuracies, they are mitigated by the large size of the dataset, which averages them out.

Using the simulation, 15,000 formations were generated, producing the following diagrams (see Figure 7.13). These diagrams were created by a neural network, a multi-layer perceptron with one hidden layer and 40 nodes, trained on the generated data and evaluated on newly generated data.

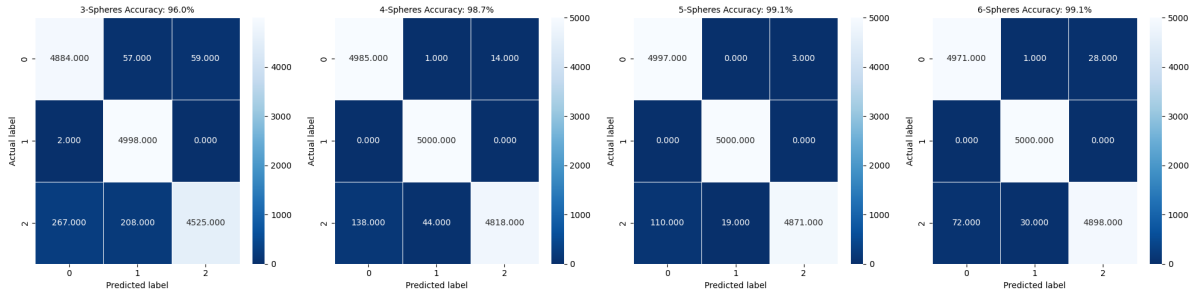


Figure 7.13: Confusion matrix of final Neural Network with MLP(40,) topology. Labels 0, 1 and 2 are line-, ring- and undefined-formations respectively.

The neural network achieves accuracies of 96.0%, 98.7%, 99.1%, and 99.1% respectively with 3-spheres, 4-spheres, 5-spheres, and 6-spheres, respectively. It can be observed that every model exhibits a similar characteristic: the undefined formation has the highest misclassification ratio of all the formations. This is likely due to the fact that, in rare situations, some separated formations might be touching or nearly touching. In such cases, it is difficult to recognize the type of formation. Additionally, there seems to be a correlation between the model's accuracy and the number of spheres the model is trained on, likely due to the ability to extract more information from larger formations.

Following the measurement of the model's accuracy, the noise resistance of each model was evaluated.

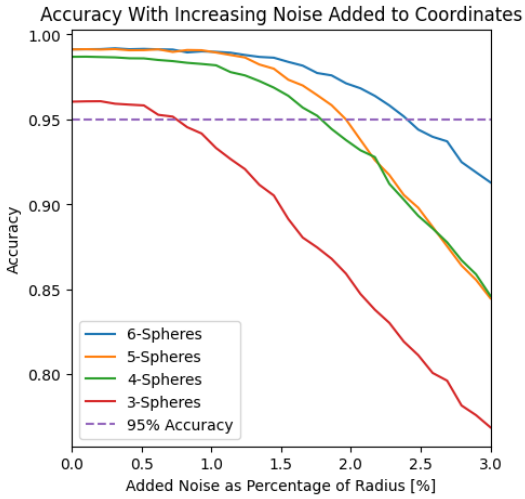


Figure 7.14: Noise resistance of final Neural Network with MLP(40,) topology

Note that when 0% noise is added, the predicted accuracy is similar to the accuracies in 7.8, as expected. Additionally, there is a clear correlation between the number of spheres and the accuracy plots. Specifically, the 6-sphere formation prediction accuracy starts and ends with the highest value, while the other models, although offset lower, follow a similar trajectory. The example plot in Figure 7.14 shows a different trajectory in some cases. Although the accuracy plot for 5-spheres starts with a higher initial accuracy, its final accuracy is lower than that of the 4-sphere plot. This discrepancy is likely due to a bias within the model, which would likely disappear if the model is re-trained.

In general, there seems to be a trend that Figure 7.8 agrees with: the more spheres present in the data, the higher the model's accuracy. This is because the model can extract more information about the structure's formation when more spheres are included. This is reinforced through the noise resistance plot, where there appears to be a correlation between noise resistance and the number of spheres in the formation. This correlation is a direct result of having more information and stronger relationships between a higher number of spheres.

The model for 6-spheres has the best resistance to noise, being able to handle added noise of about 2.4%. Following that, the 4-sphere model, 5-sphere model, and 3-sphere model have noise resistances of 2.0%, 1.9%, and 0.7%, respectively, until they fall below the 95% accuracy threshold.

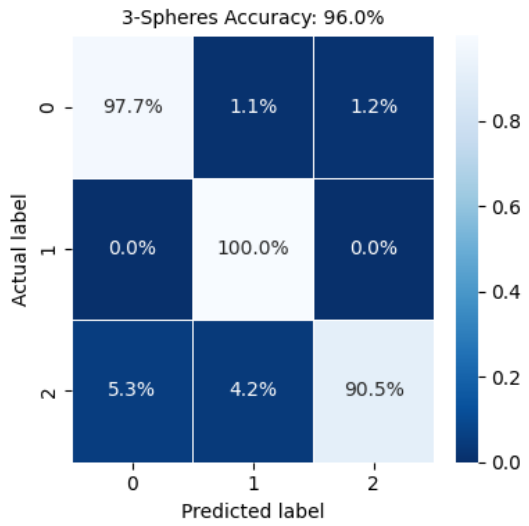


Figure 7.15: Confusion matrix of the model trained with 3-sphere formation evaluated on generated data with no added noise.

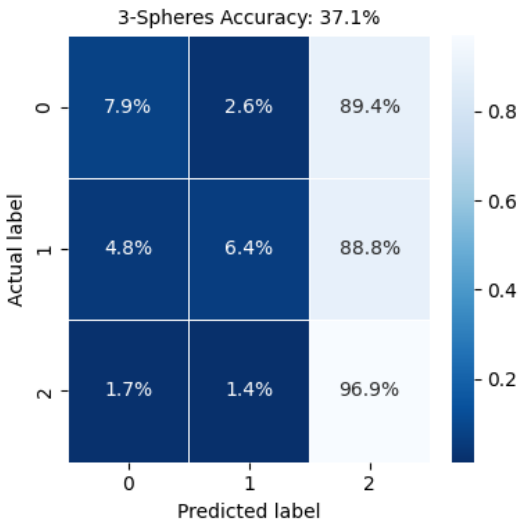


Figure 7.16: Confusion matrix of the model trained with 3-sphere formation evaluated on generated data with 50% added noise to radius only

Figure 7.17: Labels 0, 1 and 2 are line-, ring- and undefined-formations respectively.

The tuning parameter for the image processing radius and coordinate extraction algorithm (Section 8.4) determined a bias in coordinate or radii extraction accuracy. Therefore, to determine what the optimal bias is, it was necessary to investigate the accuracy of coordinates and radii data depending on added noise, as this bias accuracy would translate to noise added to the data. The element that exhibits the worst noise resistance characteristics will be the element for image processing to bias towards. The results of this evaluation are shown in Figure 7.18. Assuming that both coordinates and radius are equally sensitive to noise, it is expected that the coordinates will perform worse, as they encompass two values (x and y coordinates) instead of one (the radius of a sphere).

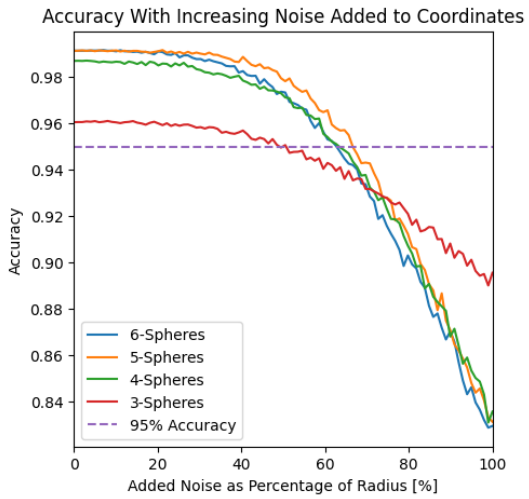


Figure 7.18: Accuracy of model with increasing added noise as a percentage of the coordinate data

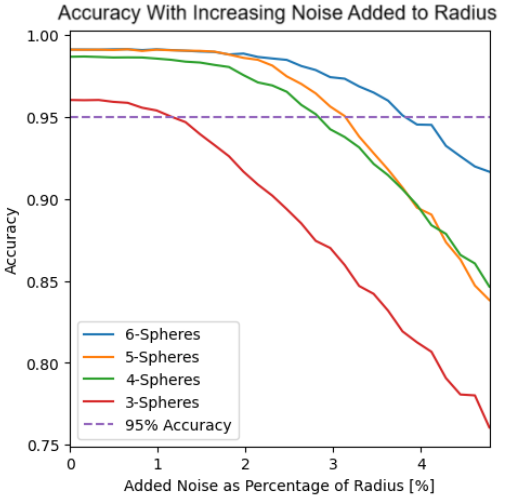


Figure 7.19: Accuracy of model with increasing added noise as a percentage of the radii data

Figure 7.20: Distribution of Generated Formations

Examining the graphs in Figure 7.20, it is evident that the assumption that both coordinates and radius are equally sensitive to noise is incorrect. The radius is approximately 20 times more sensitive to noise.

The model reaches the 95% accuracy threshold at 0.7%, 1.9%, 2.0%, and 2.4% of added noise to the radius (see Figure 7.19), while the coordinates exhibit greater noise resistance, achieving the 95% accuracy threshold at around 50%, 60%, 60%, and 65% of added noise to the coordinates (see Figure 7.18).

The difference in sensitivity can be attributed to the fact that changes in the radius of a sphere can cause parts of the formation to appear disconnected, rendering the formation undefined. This theory is supported by the confusion matrices of formations with and without noise (see Figures 7.15 and 7.17). These figures show a shift from high prediction accuracy to predicting mostly undefined formations as noise increases. Essentially making the undefined formation category a catch-all class.

The differences in noise sensitivity between the two elements might initially seem excessive. However, they fit well when compared to the ranges of the coordinates and radii. The coordinates range from 0 to 2600, while the radii range from 60 to 200. Adding a certain percentage of the mean radius has a less significant effect. Plotting the x-axis as the ratio between the added percentage of the mean radius and the maximum value of either the x or y coordinates reveals that the coordinates are only half as sensitive to noise compared to the radius. This explains why the plot in Figure 7.19 exhibits similar characteristics to Figure 7.14. The maximum coordinate is used because the relative coordinate transformation ensures the mean of the coordinates is zero.

Note that the requirement for noise resistance in Section 7.2 has not been met, as none of the radii noise resistances are above 5%.

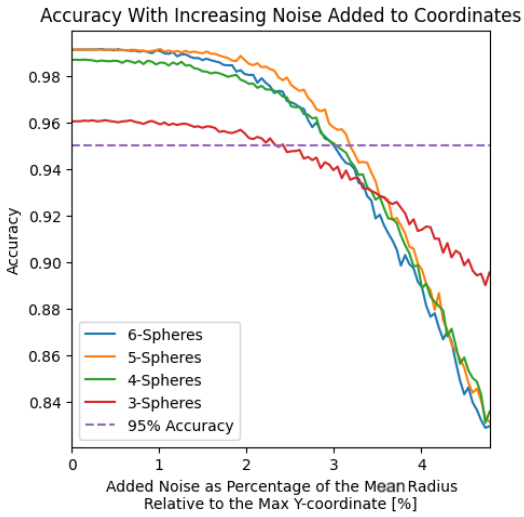


Figure 7.21: Accuracy of model with increasing added Noise as Percentage of the mean radius relative to the max Y-coordinate

Finally, the collected data was evaluated, resulting in Figure 7.22. Unlike the model trained with generated data, there is a significant bias toward misclassifying formations as undefined and a tendency not to classify any formation as a line formation. When the formation is undefined, the model has a 98% accuracy. This is likely because it is easier to discern when spheres are far apart, indicating they are not in a formation, and the undefined formation category acts as a catch-all for misclassified instances.

As seen in Figure 7.19 and discussed earlier, the radius is much more sensitive to noise/inaccuracies. This sensitivity is because a change in radius can alter the perception of the formation from being connected to appearing unconnected.

Other discrepancies in the misclassification of formations can be explained by the difference in the distribution of values between the simulation and the collected data (see Figures 7.24 and 7.23).

From this evaluation, an error was identified in the simulated data: it was generated with the wrong resolution (the maximum value of the x and y coordinates should be 2592 pixels as per the resolution of the camera). This mistake went unnoticed throughout the simulation process. This mistake was easily fixed in the code (see the Discussion in Section 7.6). The newly generated data was re-evaluated and

Data Distribution	x-coordinate range	y-coordinate range	radius range
Collected Data	1458 - 3719	1468 - 3715	52 - 147
Simulation Data	106 - 2343	425 - 2484	67 - 174
Fixed Simulated Data	162 - 2423	172 - 2419	67 - 165

resulted in the same accuracy. The coordinate translation was invariant, maintaining the accuracy of the results.

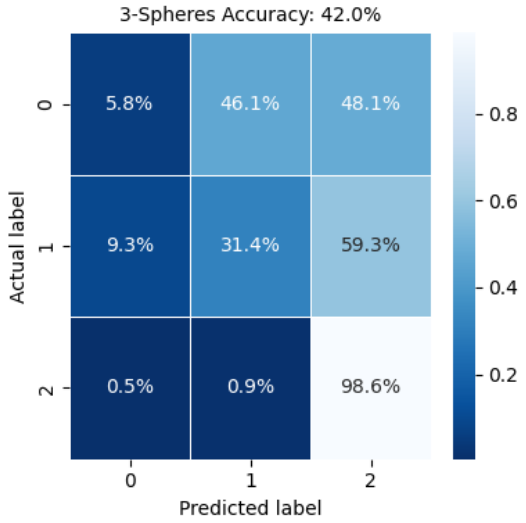


Figure 7.22: Confusion matrix of the model trained on generated data and evaluated on collected data that has been modified to match the range of the simulated data. Labels 0, 1 and 2 are line-, ring- and undefined-formations respectively.

This discrepancy raised concerns about the simulation, prompting an evaluation and comparison of the data distributions (see Figures 7.23 and 7.24). There is a clear bell curve in the collected data for the x and y-coordinate distribution, indicating a Gaussian distribution. The generated data, however, has a more triangular shape due to the uniform distribution over the length of the axis as discussed in the implementation.

The mean of the collected X-coordinate distribution appears to be shifted to the left compared to the simulated data. This can be explained by the camera being off-centre relative to the cone's centre, which shifts the mean of the X-coordinate. The Y-coordinates in both datasets seem to have the same mean value, as only the X-axis of the camera is off-centre. It is also observed that the range of the distributions for both X and Y-coordinates is the same in the dataset. However, this is not the case for the slope of the distributions and the relative peak values. Further investigation is needed to better understand the distribution discrepancy.

The distributions for the radius exhibit significant differences. Both peaks are shifted to the left, indicating the tendency of the spheres to stay in the lower half of the tank, as per the research done in [12]. However, the difference in the frequency of sphere occurrences between the lower and upper halves of the tank is smaller in the collected data. There is a higher probability of finding a sphere with a radius lower than 60 pixels, corresponding to the tendency of the spheres to settle at the lowest points of the tank (as discussed in Section 5.6). The inaccuracy in extracting the radius is much higher than the neural network was expected to handle, as the requirement for noise resistance at 5% was set much later (as discussed in Section 6.5 and related to the requirement for noise resistance in Sectuib 7.2).

Although there are differences in the distribution, they do not appear significant enough to suggest that the current model topology cannot be trained to reach the required accuracy.

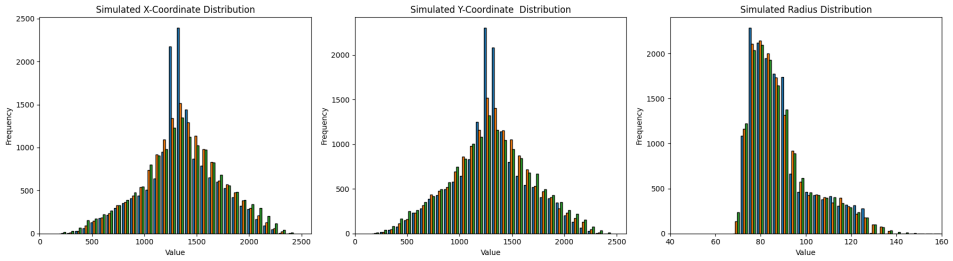


Figure 7.23: Distribution of simulated data

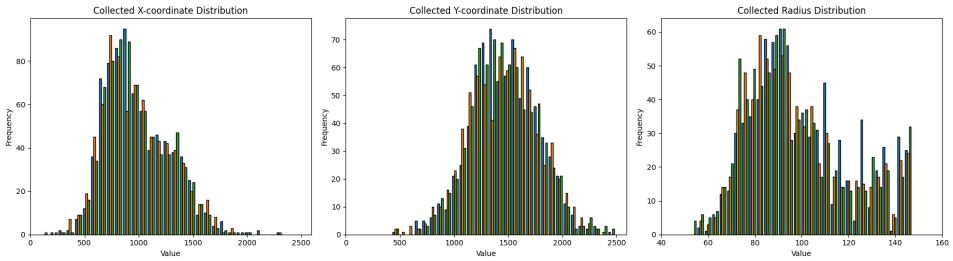


Figure 7.24: Distribution of collected data.

7.6. Discussion

7.6.1. Simulation

The main discussion around the simulation focuses on accuracy. The more obvious issues include incorrect tank measurements and incorrect camera location or resolution. A mistake was found in the scaling algorithm when projecting coordinates on a plane. In the final two lines of code, the centres of the spheres were incorrectly translated by half the resolution.

```
        # translate coordinates first quadrant of plane
        self.sphere_centers[:, 0]+= gS.tank_param.r2
        self.sphere_centers[:, 1]+= gS.tank_param.r2
        # normalize coordinates
        self.sphere_centers[:, 0]/= gS.tank_param.upper_diameter
        self.sphere_centers[:, 1]/= gS.tank_param.upper_diameter

        # scale normalized coordinates
        self.sphere_centers[:, 0] *= gS.tank_param.camera_resolution[0]
        self.sphere_centers[:, 1] *= gS.tank_param.camera_resolution[0]
        # translate to center of image
        self.sphere_centers[:, 0] += gS.tank_param.camera_resolution[0]/2
        self.sphere_centers[:, 1] += gS.tank_param.camera_resolution[0]/2
```

Listing 7.1: Python code: projectionPoint.py, normalization and scaling of coordinates.

However, these issues can be easily adjusted in the simulation code and do not pose a significant problem. The primary reason the model trained on the simulated data is inaccurate is likely due to the collected data being a high deviation from the true value or noisy data. Nevertheless, when sufficient data is available to train the model (see data size requirement in Section 7.2), there is enough evidence to suggest that it will work with this topology.

With sufficient data, the simulation's accuracy can be enhanced. More data is required as the simulation generates 3D coordinates projected onto a 2D plane. By increasing the data volume, the distribution based on the depth of the spheres (i.e., the radius of the sphere) can be analyzed. Currently, coordinates are generated through a uniform distribution over its corresponding axis, with the magnitude of the coordinates dependent on the radius of the cone in that part of the tank. To better emulate the tank's real dynamics, a dataset is required where the distribution of the tank is plotted at discrete heights. This is a cumbersome task and therefore could be more easily approximated by scaling a normal distribution's standard deviation based on the sphere's height in the tank.

Another issue is that formations can only be generated in spheres or arc formations and not in snake-like patterns, which will undoubtedly occur in the actual setup. This effect will be exacerbated when adding more spheres. Consequently, the models have only been trained on formations with up to 6 spheres. It was deemed that beyond this point, the simulation would become too inaccurate. Not only will lines form snake-like formations, but ring formations won't always be perfect circles. They may instead take on elliptical shapes or resemble the edge of a hyperbolic paraboloid (Pringles-shaped ring). This will undoubtedly also impact the data size required for training a model to recognize larger formations. As formations become more complex and deviate from ideal shapes, the simulation's accuracy decreases, necessitating a larger dataset to adequately capture the variability and improve the model's robustness.

Finally, the model will never be trained on certain formations due to the strategy for deleting data when the visibility of the formation is determined to be too low. For example, when a line is perpendicular to the camera in the simulation, all coordinates and corresponding radii are generated. However, in the real world, the camera will only see one sphere. The visibility algorithm removes such datasets as they are deemed useless, and thus the model is not trained on them.

A fix for this problem wouldn't be hard to implement, but it may not be necessary. Since the model will be trained on data it will never encounter in practice, it won't need to make predictions on these specific types of data. In other words, while the model could predict more orientations of formations, these will never actually be fed into the network.

7.6.2. Deep Learning

The main discussion for the neural network concerns the amount of features that can be extracted. Feature selection is a crucial aspect of machine learning and an essential part of model development. However, for this project, feature selection was not implemented due to the initial belief that there was sufficient information available and a lack of experience with machine learning. In hindsight, it is a logical step, but there was insufficient time to implement and redo all the tests as of writing.

A clear understanding of what works and what doesn't has been established, which will be elaborated upon in this section. From this analysis, strategies for future improvements will be suggested.

After collecting the generated data, it will be formatted as a set of $n \times 3$ matrices where n is the number of spheres in the formation. To train or predict data with the model, these matrices will need to be reshaped into a vector of size $3n$. Consequently, the input to the neural network will have $3n$ nodes.

When using the relative centre's transformation, the input size remains $3n$, but this is not the case with the relative angle transformation, which reduces the vector length to n. This reduction explains why the hidden layer is smaller, as there is less information needed to tune weights and biases to reach stable accuracy. This is also why the accuracy for, for example, 3-sphere formations, can't reach higher than 60%, as there is insufficient information.

A good strategy might be to extract both features: relative coordinates and angles between those coordinates, which has generated the plot below, Figure7.25.

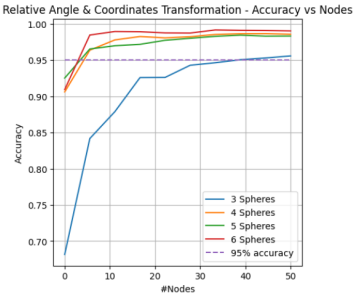


Figure 7.25: Accuracy as a Function of the Number of Nodes in the Hidden Layer, Using Both Relative Coordinate and Relative Angle Transformations on the Data

With this model, it can be observed that for formations with more than 3 spheres, the models start to reach stable accuracy or converge with just above 8 nodes. However, similar to the other plots in Figure

7.7, the model still struggles with formations involving 3 spheres. It might be advantageous to use this strategy with higher-order formations, as it will require less processing time when the model uses fewer nodes.

Keep in mind the concept of diminishing returns. This concept describes the relationship between the number of features used in a model and the resulting accuracy. As more features are added, the model's accuracy typically improves up to a certain point, after which additional features may offer diminishing returns or even degrade performance due to overfitting or noise.

Another issue is the unknown characteristics of the noise, which means the characteristics observed in the noise resistance evaluation tests may not exactly match those in the data eventually collected. The noise has been added using a uniform probability distribution, but it could exhibit different characteristics in reality. Despite this, evaluating noise resistance in this way remains valuable for managing expectations and tuning parameters for other sub-teams.

Additionally, the requirement for noise resistance in Section 7.2 was established very late, only after the radius and coordinate accuracy had been evaluated post-training. This delayed the neural network team's ability to tune the model for noise resistance, which could have been achieved by training the model on noisy data. Unfortunately, there was not enough time to modify the model to meet the noise resistance requirement at that stage. However, new data will be gathered for training the model, which could not be included in this report due to time constraints. This new data is expected to be inherently noisy, allowing the model to be naturally trained for noise resistance, potentially making additional tuning unnecessary. This will also verify whether this topology will work on the gathered data and can be used to increase the simulation's accuracy.

7.7. Conclusion

In conclusion, the model classifies generated formations with up to 6 spheres with an accuracy of up to 99%. Beyond this, the structures become more complex than a perfect circle or arc, which the current simulation cannot generate. The simulation produces tens of thousands of separate formations within seconds, fulfilling the requirements in 7.2.1. Data from these formations can be stored, loaded, and fed into a deep learning algorithm.

The neural network's accuracy varies with the number of spheres in the formation. It achieves over 95% accuracy with 3-sphere formations and up to 99% with 6-sphere formations when trained and evaluated on transformed data (coordinates relative to one sphere) meeting the accuracy requirement in Section 7.2. No network has been trained on formations with more than 6 spheres, as it was determined that the simulation would not accurately represent the true experiment beyond this point. However, there appears to be a correlation between model accuracy and the number of spheres in the formation. A neural network with a MLP 40-node, single hidden layer topology shows higher accuracy with more spheres.

Thus, although the simulation accurately represents the experiment up to a certain number of spheres, it is likely that if the actual experiment collected enough data, the neural network would achieve the desired accuracy with formations involving more than 6 spheres, potentially up to around 13-sphere formations.

This is not supported by the model's noise resistance. When trained on 'clean' data and evaluated on newly generated 'clean' data with added noise as a percentage of the data value, the model can handle up to 2.4% noise for 6-sphere formations and down to 0.8% noise for 3-sphere formations. This does not meet the noise resistance requirement in Section 7.2 , as the requirement was set too late for the model to be re-trained on generated noisy data (see the Discussion in Section 7.6).

Using the generated data, the required data size to train the model was determined to be around 2,500 samples per formation. To gather data from the actual experiment, at least 2,500 images per formation type per sphere amount are needed. This requirement was not met, and therefore, the model could not be trained on the collected data. When making predictions on the collected data, the model had an accuracy of 42%.

This evaluation revealed an error in the range distribution of the coordinates in the simulation, where

the coordinates were unnecessarily translated. This mistake was corrected by ensuring that the simulation's range matched the experiment's range. However, this error did not affect the model's accuracy since the model is translation invariant, suggesting that the generated data is more inaccurate than initially thought.

Due to the difference in data size, further data collection is necessary to enhance the simulation's accuracy by analyzing the distribution of the coordinates and radii within the tank. Nonetheless, the variation in the simulation is not significant enough to conclude that this model topology will be ineffective. There is a high degree of confidence that this topology will work if enough data is gathered.

8

Data Management

8.1. Introduction

Data management consists of two main components:

8.1.1. Image Storage

Captured images need a reliable storage solution. While there are many options, only a few are suitable for this project. As discussed in Chapter 4, a remote device will automatically handle the image-capturing and saving process. However, additional requirements must be specified for this project, including:

- How much storage space is required at different stages of the system?
- What are the client's specific needs?
- How will the images be secured during the experiment?

8.1.2. Metadata Storage

After processing the images, the associated metadata needs to be stored in an appropriate format to ensure data validation, as outlined in Chapter 2. This chapter will detail the specific requirements and solutions for metadata storage.

Following the discussion of design choices, the chosen solutions will be implemented and tested. The results of these experiments will be analyzed, leading to a conclusion on the effectiveness of the design.

Project Scope and Objectives

The aim is to develop a robust system for managing image data and metadata efficiently and securely. The main objectives include:

- Designing a scalable and secure storage solution for both images and metadata.
- Ensuring the system meets the specific requirements of the client.
- Validating the data to maintain integrity and reliability throughout the process.

Outline of the Chapter

The outline of this chapter is as follows: the first section, 8.2, will discuss the requirements for temporarily storing images and long-term metadata storage. Details will be discussed such as data storage size, accessibility and more.

In Section 8.3, alternative designs will be discussed, which are potential solutions regarding the requirements.

Section 8.4 will discuss how the design choices are implemented and the problems that were encountered during this process.

Section 8.5 will explain experiments to prove if the modules are working and show the results of these experiments.

Section 8.6 discusses the experiment's results. The conclusion, section 8.7 goes over the chapter again and summarises the discussion of this chapter.

8.2. Program of Requirements

As discussed in Section 2, key considerations for the system include automation, synchronised data collection, portability and accessibility. This section outlines the requirements for selecting a device that meets these criteria. Each requirement is referenced back to the top-level requirements.

8.2.1. Experiment Control Device Requirements

1.1 **Remote Accessibility**: the device should provide a simple method for connecting to a remote server. This follows from the top-level's 'accessibility' requirement. (1.4)

1.2 **Synchronised Data Collection**: The device must support the synchronisation of captured images. This way the system can be validated to show the reliability and usability of the system, this also follows from the top-level requirements. (3.5)

1.3 **Portability**: The system should be portable, ensuring convenient transportation and easy setup for image saving. (1.3)

1.4 **Budget**: The device, in combination with cameras, should fit within a budget of €400 or less and is available for delivery. (1.2)

1.5 **Performance**: The device must have sufficient performance capabilities to upload high-quality images. Performance will have an effect on the overall accuracy of the whole system. Accuracy is an important top-level requirement.

1.6 **Future Scalability**: the device's function should be scalable to accommodate future expansions. (3.4)

1.7 **Experiment Time**: The device should demonstrate reliability over extended operation periods (≥12 hours). (3.6)

8.2.2. Data Storage Requirements

Temporarily Image Storage: After image upload, temporary storage is needed for image processing. The storage solution must be easily accessible and have sufficient capacity for ≥12 hours of image data. This data will be stored until the images are processed and validated:

2.1 **Data Storage Size**: The device must have enough storage to save ≥12 hours of images temporarily (2.2).

2.2 **Safety**: The device should be resilient to unforeseen issues, especially considering the project's environment (3.5 & 3.6).

2.3 **Accessibility**: Images must be easily accessible for processing (1.4).

2.4 **Data Retention:** The original image dataset should be stored until it is proven that the developed method meets the specified accuracy requirement (3.5).

2.5 **Budget**: The temporary image storage, in combination with the device and cameras; thus should fit within a budget of €100 or less and is available for delivery (1.2).

Data Formatting: The processed and validated data for each image, along with its metadata, must be stored in a well-structured database. This database should be compact and readable to facilitate data validation. The requirements for the database format include:

3.1 **Concise Structure**: The format should allow for a well-organized, concise structure (1.4).

3.2 **Data Storage size**: the format must be compact (2.2).

3.3 **Readability**: the format must be readable to enable data validation (1.4).

Uploading and Downloading: both of these have the same requirement, which is:

4.1 **Computational Time**: the time to upload or download an image should not influence the waiting time for processing the images. This means no longer than 6 seconds of uploading time as image processing takes this amount of time to process an image, see Chapter 6. Downloading time should be milliseconds to minimise download waiting time (3.7).

8.3. Design Choices

8.3.1. Remote Device

To meet criteria such as budget-friendliness, reliability, ease of use, and remote connectivity, a Single-Board Computer (SBC) was chosen over other options like microcontrollers or commercial IP cameras. Among various SBC options, the Raspberry Pi 5 (RPI5) [17] emerged as the best choice due to the team's prior experience with the device.

8.3.2. Image storage

Given the potential for the experiment to generate a large volume of image data over a ≥12-hour period (every 2 to 5 seconds an image is captured), sufficient storage capacity is crucial. Keeping this primary requirement in mind with the requirement of 8.2.2, several options were considered: SD card, external hard disk, and cloud storage. Each advantage and disadvantage is referenced back to their requirement.

Storage Method	Pros	Cons
SD Card	• Enough storage capacity (2.1) • Easily accessible (2.2)	• Vulnerable in experimental conditions (e.g., water) (2.3) • Expensive for large data sizes (2.5)
External Hard Disk	• Enough storage capacity (2.1) • Budget-friendly (2.5)	• Vulnerable in experimental conditions (e.g., water) (2.3)
Cloud Server	• Very large or "unlimited" storage capacity (2.1) • Saves physical storage space (2.3) • Data is secure once uploaded (2.3) • Budget-friendly (client already has a cloud server) (2.5)	• Requires a stable internet connection for live uploading (2.2)

Based on this Table 8.3.2, cloud storage emerged as the preferred choice due to its large capacity, space-saving benefits, and existing availability for the client. To make this option even better, a combination with the use of an SD card will be implemented. The SD card will function as RAM, whenever the internet connection is not stable to upload to the cloud. This will ensure no data will be lost.

To add to this "internet connection" drawback, a brief experiment testing the stability in the project's environment, showed sufficient upload speeds. This further supports the feasibility of cloud storage. 5 measurements in one hour have been conducted with the help [19] (see Table 8.1). The slowest speed

Timestamp	09:09	09:25	09:37	09:56	10:05
Upload Speed (Mbps)	258.3	76.0	80.21	55.6	50.4

Table 8.1: Upload speeds (Mbps) for 5 measurements and their corresponding timestamps

is 50.4 Mbps within this one-hour test, which is still fast enough for uploading images. On average the upload speed is 104.1 Mbps. The results of this test ensure that there is no need for significant concern.

To summarise, the eventual setup involves using a Raspberry Pi 5 with a 128GB SD card for temporary storage, with images uploaded directly to the cloud whenever possible.

8.3.3. Metadata Storage

After image processing, metadata storage options such as JSON, TOML, and HDF5 were evaluated. See Appendix D.11 for a weighted table. HDF5 emerged as the most suitable choice, despite the limitation of not being able to be directly opened like a '.txt' file (this disadvantage applies to most formats). This limitation can be easily overcome by converting the structured data from the HDF5 file into a '.txt' file. Additionally, compression can be applied to mitigate any concerns regarding file size.

8.4. Implementation

This chapter outlines the implementation of the data collection process, data handling, and cloud integration using a Raspberry Pi.

The process pipeline, based on the design alternatives, is as follows: the Raspberry Pi captures images using the attached camera. The images will be temporarily stored on the Raspberry Pi, subsequently, these will be uploaded to a Google Drive. From the cloud, the images are downloaded and processed, and then stored as a .h5 file. The processed data is analysed using a Deep Learning algorithm.

It is important to note that the image downloading and processing pipeline runs simultaneously with the execution of the experiment, to accelerate the overall process.

All these processes will be automated to fulfil the main requirement: automation.

8.4.1. Cloud Integration Using Raspberry Pi

Raspberry Pi

Integration of the Raspberry Pi involves establishing a connection with a remote laptop and explaining the use of the Raspberry Pi.

Initially, the Raspberry Pi was set up and configured with an SSH key, enabling remote access from a laptop to configure its IP address. Later, a VNC connection was established, allowing easier control of the Raspberry Pi through a simple IP address. This improved accessibility, as not every PC needing to communicate with or control the Pi would require its own SSH key.

Using the laptop, the user can execute the main program, specify the saving folder for data storage, and initiate experiments. This initiation means starting the camera and capturing images, which are then uploaded to the cloud.

Image Capturing

Image capture is managed by a simple script on the Raspberry Pi, triggering the camera at set intervals and saving images locally. Each image is uniquely named for automation purposes: "xxxxx_image", where "xxxxx" is a number from 00000 up to 99999.

Uploading

Once the images are saved locally, they are uploaded to a cloud server, currently the project team's Google Drive, enabling easy access for all team members.

The upload algorithm works as follows: at regular time intervals, the script will send a request to upload the images to the Google Drive server, and if the request is successful, the images will be uploaded. Once the images are uploaded, they will be deleted from the Raspberry Pi.

If a connection cannot be established, the program handles this exception instead of crashing. The images will be kept on the Raspberry Pi and during the next interval, the program will try to reconnect and upload the images again.

Downloading

Images can be downloaded in real-time onto a remote system during the experiment. The download process continuously monitors newly uploaded images, downloading them in chronological order. Robust error-handling strategies, such as retry attempts and dynamic API connections, are used to ensure stability and prevent crashes.

While implementing the download process, several errors occur in the threading processes, causing the program to crash. To address this within the limited implementation time frame, an efficient solution

was adopted. Errors were caught using try-except blocks, with a setup to try downloading the image three times. In case of failure after three tries, the program would go ahead and download the next item. This strategy was effective in preventing frequent crashes.

Another problem faced was a 'malloc' error, and this was surprising because Python is known to handle memory problems by itself. The problem occurred while working with Google's authorization process and, to some extent, in combination with Python's threading. At first, in combination with threads, it was found that if a connection were made to Google's API at the start of the program, then eventually a 'malloc' error happened and crashed the program. The solution was for the thread to connect to Google's API upon each execution, during the download of a file. This procedure ensured that there was no crash error and that the download operation proceeded smoothly.

```
def run(self):
        try:
            # Run build (again) within threaded function, otherwise malloc() errors
            creds = authenticate()
            service = build(API_NAME, API_VERSION, credentials=creds)
            request = service.files().get_media(fileId=self.file_id)
            with io.BytesIO() as byte_stream:
```

Listing 8.1: Python code: Google API and Threading.

8.4.2. Data Formatting

The output from image processing, a NumPy array, is transformed into HDF5 format using scripts. This HDF5 format, when printed into a text file, provides a structured and easily interpretable representation of the data, see Figure 8.1.

```
00000_Image
  Formation: 0
  Sphere_1:
    Coordinate_x: -0.4288183443673478
    Coordinate_y: 0.4288183443673477
    Radius: 0.014632492238354855
    Visibility: 0.5382459989951148
  Sphere_2:
    Coordinate_x: -0.6191921770114961
    Coordinate_y: 1.051503093362322
    Radius: 0.013396831614441248
    Visibility: 0.5408016960372727
  Sphere_3:
    Coordinate_x: -0.7755345244206772
    Coordinate_y: 1.6682866191950516
    Radius: 0.012351047042317756
    Visibility: 0.4677212154806605
  Sphere_4:
    Coordinate_x: -0.8977977632013477
    Coordinate_y: 2.2789810436286517
    Radius: 0.011454757087188024
    Visibility: 0.8719386937715661
  Sphere_5:
    Coordinate_x: -0.9859446507865849
    Coordinate_y: 2.88340034322551
    Radius: 0.010678270791698192
    Visibility: 0.9264346771511778
```

Figure 8.1: HDF5 formatting

As the input of the deep learning algorithm requires a NumPy array, another script is designed to deconstruct the HDF5 files to a NumPy array.

Additionally, an option for compressing the HDF5 and .txt files into zip format is provided for long-term storage, addressing the requirement for minimal data storage size.

8.5. Results

This section presents the outcomes of experiments conducted to validate the functionality and reliability of the implemented subsystems.

8.5.1. Cloud Integration Using Raspberry Pi

First, 2 experiments will be explained, which should prove working sub-modules.

Experiment 1

Every individual sub-module within image capturing, uploading, and downloading has been tested individually. The next level of testing integrates these modules into a laptop and runs the tests for an extended period to determine their long-term reliability. A one-hour test period should be enough at this stage. This period is not enough to determine a ≥12-hour reliability but serves as enough evaluation to test the sub-module by itself on a laptop.

The images are captured every 2 seconds and uploads will happen in batches every 5 seconds. Every 5 seconds an image is downloaded. The experiment will be documented by saving the terminal logs and comparing them to ensure that the correct images are downloaded on the laptop.

Experiment 2

A second experiment will be conducted to check the reliability of the Raspberry Pi. A one-hour test will be executed, from which again the terminal log will be stored.

With this experiment, images will be captured every second and uploaded continuously. This will also directly test the reliability of faster uploading.

8.5.2. Experiment 3

A third experiment will purely check the time it takes to download an image. This data will be proved again by logging the terminal. The download speed during this experiment was measured at around 130 Mbps with [19] and conducted at the Technical University of Delft.

Results Experiment 1

The log output found in listing 8.2, shows the beginning and end of the test period. Images were successfully captured between 11:50:33 and 12:50:50. A total of 720 images (numbered 0-719) were uploaded, which had no failed uploads and are available in Google Drive.

```
Image 00001_Image.jpg captured successfully.
2024-06-07 11:50:33,959 - INFO - file_cache is only supported with oauth2client<4.0.0
Image 00002_Image.jpg captured successfully.
File '00000_Image.jpg' uploaded successfully.
# skip to end
File '00719_Image.jpg' uploaded successfully.
2024-06-07 12:50:50,407 - INFO - Contents of folder 'Images' uploaded and deleted
    successfully.
Image 00720_Image.jpg captured successfully.
```

Listing 8.2: log: Uploading timestamps

One error occurred during the one-hour testing period (log 8.3): the server connect error, is trapped in the try-except block, and the algorithm continues.

```
ERROR - An error occurred in upload_and_delete: <!DOCTYPE html>
#...
  <p>The server encountered a temporary error and could not complete your request.<p>Please
    try again in 30 seconds.  <ins>'Thats all we know.</ins>
```

Listing 8.3: Log: Errors in uploading.

This log output shows how the first and last images are successfully downloaded, and linked with their time stamps. The last download happened 17 minutes after the last upload, as the sleep time between downloads is longer than the upload time.

```
What is the name of the folder of the experiment?: test_experiment_1
2024-06-07 11:51:47,598 - INFO - file_cache is only supported with oauth2client<4.0.0
2024-06-07 11:51:49,061 - INFO - Download successful: 00000_Image.jpg
# skip to end
2024-06-07 13:07:35,616 - INFO - file_cache is only supported with oauth2client<4.0.0
2024-06-07 13:07:37,050 - INFO - Download successful: 00719_Image.jpg
```

Listing 8.4: log: Downloading timestamps

This error is a server connection error. It can be resolved in the same way as with the errors during uploading: by using a try-except block, so that the process can keep on going.

```
2024-06-07 12:30:01,193 - ERROR - Error downloading chunk: IncompleteRead(129614 bytes read,
    169937 more expected)
```

Listing 8.5: Log: Errors in downloading.

Result Experiment 2

The experiment was conducted for 45 minutes (10:58:31 - 11:45:48). Within this 45 minutes, 3 different times uploading was started. In the first execution, a connection (IOError) occurred 8.6. This collided with the whole system, and uploading was not reliable. The next 2 executions of the script did work effectively.

```
File '00329_Image.jpeg' uploaded successfully.
2024-06-17 11:45:15,419 - INFO - Contents of folder 'Images' uploaded and deleted
    successfully.
File '00328_Image.jpeg' uploaded successfully.
```

Listing 8.6: Log: Errors in uploading.

The uploading speed was observed to be diverse between a burst upload within a second and sometimes uploading took 2 seconds.

```
Image 00318_Image.jpeg captured successfully.
File '00317_Image.jpeg' uploaded successfully.
2024-06-17 11:44:42,114 - INFO - Contents of folder 'Images' uploaded and deleted
    successfully.
```

Listing 8.7: Log: Uploading speed.

During testing, it was noticed that the cooling fan of the Raspberry Pi did not work. Forcing the cooling fan within the terminal was tried, but not succeeded. However, with the cooling of the flashlight and smart placement, the Raspberry Pi would not overheat.

8.5.3. Results Experiment 3

In the logbook 8.8, the initial timestamp of connecting to the Google Drive is made and the time stamp is shown when the download was finished. The time between these two is 1.8 seconds. The total amount of images downloaded during this experiment was 100 images. By observation, the amount of seconds it took to download an image was around 2 seconds.

```
2024-06-18 16:24:03,686 - INFO - file_cache is only supported with oauth2client<4.0.0
2024-06-18 16:24:05,499 - INFO - Download successful: 00157_Image.jpeg
```

Listing 8.8: Log: Download time.

8.5.4. Data Formatting

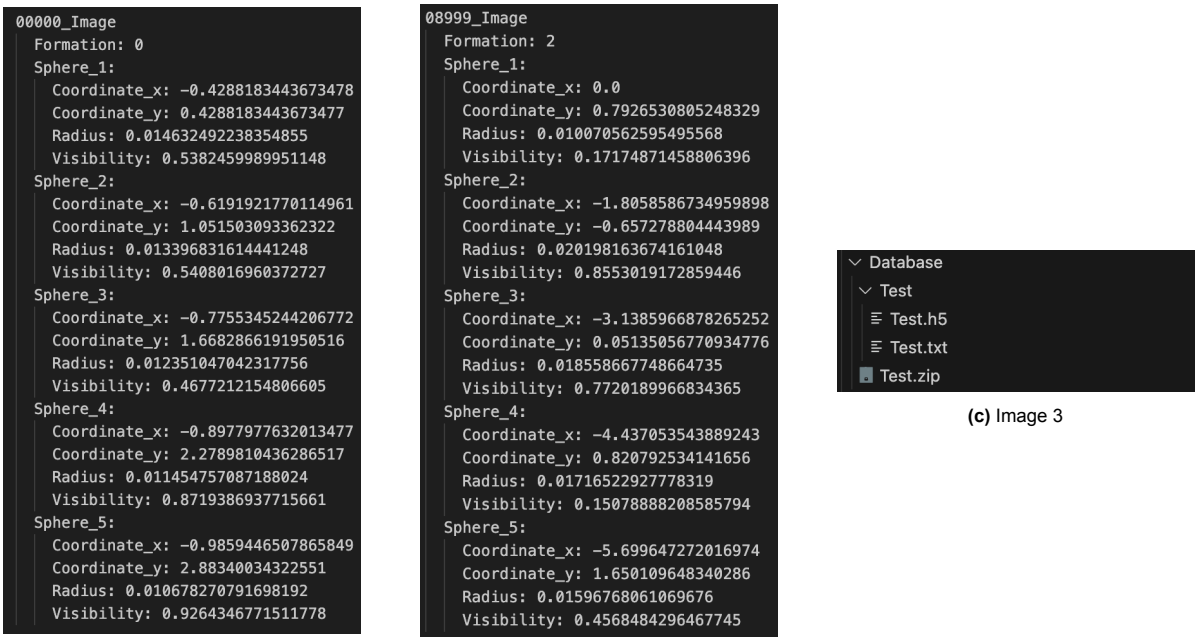
Experiment 4

The experiment aimed to validate the data formatting process by simulating 9000 images and converting them into HDF5 format and a text file. The resulting data was then compressed into a zip folder to assess storage size.

An experiment for this is very straightforward. Give input in a NumPy array per image with spheres and their respective coordinates, radius and visibility generated from the Deep Learning simulation. If the formatting still works correctly for a lot of images and can zip with the use of a script, the sub-module works.

Results experiment 4

In figure 8.2 the metadata of the first and last images are displayed. Next to that, the database is shown with the 'normal' folder and the zip folder. The zip folder size was found to be 8.8 Mb, demonstrating the efficient data compression capabilities of the formatting process.



(a) Image 1

(b) Image 2

(c) Image 3

Figure 8.2: Formatting and database

8.6. Discussion

The Discussion section provides a detailed analysis of the experimental outcomes, focusing on the effectiveness of data formatting procedures, and the performance and reliability of cloud integration using the Raspberry Pi.

8.6.1. Cloud Integration Using Raspberry Pi

Experiment 1

The results of Experiment 1 were positive. During the one-hour testing period, only two errors occurred during uploading and one during downloading, all related to server connection issues. These errors were effectively managed by the algorithm, allowing the process to continue smoothly. The successful capture and upload of 720 images, along with their accurate download, demonstrate the functionality and reliability of the sub-modules.

Although the test duration was limited to one hour, the positive results indicate potential for long-term reliability. Extended testing is recommended to fully assess the system's performance over 24 hours or more.

Experiment 2

The error observed in this experiment could cause problems during execution. While restarting the script resolves this issue, it is not a foolproof solution.

During the creation of the training data and the extended execution of the script on the Raspberry Pi, the sub-module worked as expected whenever the script started correctly, such as with a good connection and proper image capturing. However, initialization sometimes failed, which resulted in later data loss if proceeded. This issue was observed by the team but not officially documented.

The uploading speed either occurred in bursts or took more than 2 seconds. This variation is due to the algorithm checking the upload folder and uploading all contents within that folder directly. Between

these bursts, there can be a longer delay before uploading again, explaining the time differences. This implementation is as expected; it does not slow down the overall pipeline process since uploading is anticipated to be faster than downloading and processing the image. This will be further discussed in the final discussion (see Chapter 10).

8.6.2. Experiment 3

The time required to download an image significantly slows down the total process. This issue can be mitigated by designing a buffer system, where the next image is downloaded while the current image is being processed and analyzed by deep learning algorithms. This would eliminate download waiting time. Although this idea has not been implemented in the system, it should be a priority when focusing on optimizing computation time.

8.6.3. Data Formatting: Experiment 4

The data formatting process performed as expected. The formatted data maintained a concise structure even with a large dataset of 9000 images. The data was easily readable in a text file format, facilitating validation and analysis. Additionally, the compressed data size was efficiently reduced to 8.8 MB, well within the target of 500 MB for long-term storage of experiment data.

The successful implementation of data formatting and cloud integration underscores the system's viability for automated, synchronized data collection and storage. These findings support the system's potential for scalability and reliability in long-term experiments.

8.6.4. Future Recommendations

Overall, the system functions as expected and is correctly set up with the sub-modules. The main issue is with initializing the uploading script. The recommendation is to make this process more robust by improving the handling of IOE errors. Instead of continuing the script, the program should restart and resume uploading from the image that initially caused the error. It is crucial to save the images that were not uploaded when the error occurred. A script implementing this feature already exists but should be verified for correct functionality if this idea is adopted.

After addressing this issue, a long-duration test should be conducted to meet the requirements of **Experiment Time** (1.7), demonstrating the system's reliability for long-term use.

8.7. Conclusion

This chapter reviewed the implementation of remote accessibility, an upload and download script, and storage requirements. Through targeted experiments, the system demonstrated good results.

The implementation of the Raspberry Pi 5 successfully met the requirements of accessibility (1.1), synchronized data collection (1.2), portability (1.3), budget (1.4), performance (1.5) and future scalability (1.6). These requirements were fulfilled effectively because the Raspberry Pi 5 provided a platform for capturing and uploading images while being compact and easily transportable, during the use of multiple experiments. As the system is an OS, the future expansion is limitless. No valid experiment has been done to show a valid proof for the requirement *Experiment Time*(1.7).

The upload script was implemented on the Raspberry Pi, and the download script was run on a laptop. The most important aspect of system reliability during extended experiments was addressed, with the results showing reliability whenever initialisation was conducted properly. The system handled server connection errors properly, not including initialisation, and maintained consistent operation throughout the testing period, which shows potential for long-term use.

The uploading time was within the margin of the requirement (6 seconds) [4.1]. Downloading time did not fulfil the requirement of having 'milliseconds of download waiting time' by the image processing and deep learning pipeline.

Data formatting did not require extensive testing, as it was straightforward to demonstrate its effectiveness. The script successfully formatted a large dataset, creating a readable text file and compressing the data into an 8.8 MB zip file. This satisfied the most important requirement for minimal long-term data storage size (3.2), confirming the efficiency of the data formatting process. Furthermore, all other

data storage-related requirements were met.

In conclusion, the experiments showed positive results, except for the downloading time. This module can however be improved with the help of a buffer. Furthermore, the system's long-term reliability remains sensitive to network initialisation. Further testing and adjustments may be needed to ensure consistent long-term performance.

9

System Integration

9.1. Introduction

This chapter discusses the full integration of all sub-modules and how they are realized as an integrated system. It details the general operation of the program, highlighting the input and output of each sub-module and identifying any implementation bottlenecks. While the system as a whole will be tested, the results of these tests will be discussed in Chapter 10 and the conclusion in Chapter 11.

Automation is a key aspect of the system integration, as it directly influences computational time. Furthermore, all previously discussed requirements from Chapter 2 apply to this full project implementation. This chapter focuses exclusively on the implementation process and the results obtained.

9.2. Implementation

The same figure from Chapter 4 is shown in Figure 9.1 for clarity regarding the entire system. As such, it will not be explained in detail again here. Instead, this section will focus on discussing the exact inputs and outputs of each submodule.

Additionally, the implemented code for system integration can be found on GitHub. In Appendix C.1, each branch relevant to the project is explained.

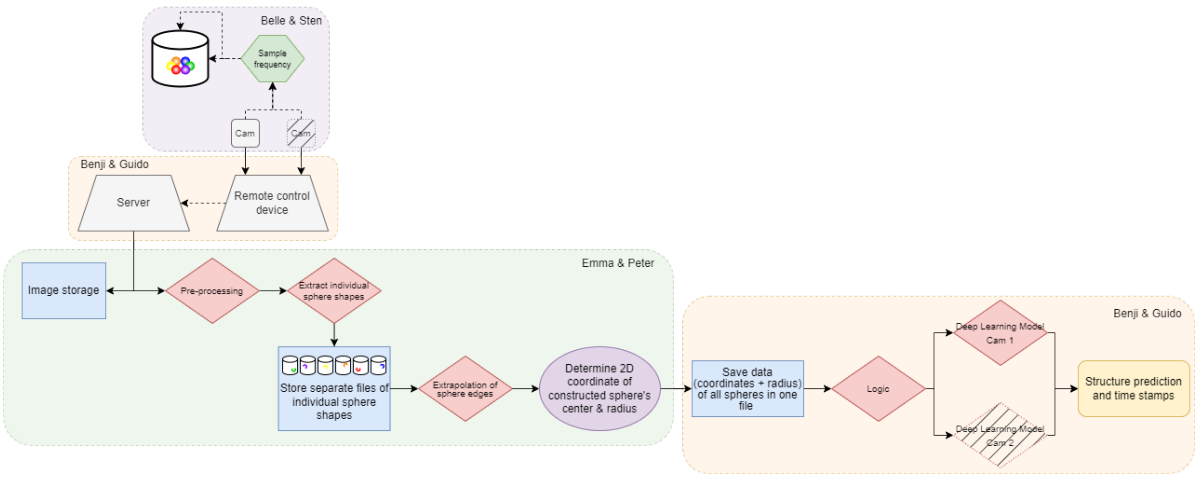


Figure 9.1: Pipeline of the top-level implementation (can also be found in appendix E.78).

9.2.1. General Use

The goal of the system integration is to design a fully automated process for the client. This process is structured as follows: when the user starts the program, they are prompted to choose whether to

start the process of defining a Ring, Line, or none. From here, two more options will appear: using an existing database or conducting a live experiment.

For the existing database option, data that has already been processed by image processing and saved in a zip folder will be used as input for deep learning.

The second option is to run a direct check of the downloaded images. These images could be uploaded live or already exist on a cloud server. In either case, these images will be processed one by one through the image processing module, then through the deep learning algorithm, and finally saved into an HDF5 file. The output for each image will be displayed accordingly.

9.2.2. Setup to Image Processing

The system is designed to directly process newly uploaded images from the cloud that have been uploaded from the setup. The structure is sequential, with image processing receiving a new input only after processing the previous image. This means that one image per iteration will be on the local system, which helps save storage space. This process is repeated until the user stops it by pressing 'Ctrl+C'.

9.2.3. Image Processing to Deep Learning

Once the image processing algorithm completes its task, the output is passed through the deep learning network. If the process is running live, the output of image processing (a NumPy dictionary) is fed directly into the neural network. The neural network will display its prediction alongside the corresponding image. All of this data is saved into an HDF5 file.

If the process runs offline, the saved HDF5 format is deconstructed (without its prior given label) and input into the neural network. The neural network processes the images and updates all the labels according to its new output.

9.3. Results

During the implementation, it was concluded that a green sphere was unrecognizable by the image processing module. This issue impacted the deep learning sub-module, preventing the generation of complete results. Only a small number of images, including three spheres, were successfully processed by image processing and subsequently analyzed by deep learning using a simulated neural network. These results have already been discussed in Chapter 7, so they will not be detailed again here. However, the accuracy of the current trained neural network in defining a ring, line, or none with three spheres is 42.0%. Excluding the 'undefined' category, which likely serves as a "catch-all," the accuracy drops to 18.4%.

9.3.1. Validation

For more comprehensive results, such as those from a 12-hour experiment, the required sample size n for the validation set can be calculated using the finite population correction formula [6]. For a 12-hour experiment, this calculation results in a sample size of approximately $n \approx 380$.

$$n = \frac{N \cdot Z^2 \cdot p \cdot (1-p)}{E^2 \cdot (N-1) + Z^2 \cdot p \cdot (1-p)}$$

Where:

$N =$ Population size for a 12-hour experiment is 43200 images

$Z =$ Z-score corresponding to the desired confidence level (1.96 for 95% confidence)

$p =$ Estimated proportion (0.5 as a conservative estimate)

$E =$ Margin of error (0.05 for 5% error)

10

Discussions

The discussion provides a critical analysis of the results and findings from the implementation and integration stages. It discusses the project's successes, limitations, and potential areas for improvement, offering insights into the effectiveness of the chosen design and implementation strategies.

10.1. Final Result

The current implementation results of the system do not meet the set requirements. This is due to the unavailability of training data for the neural network, which shows currently an achievable accuracy of 42%. However, the result of the neural network itself, with trained simulation data, still shows promising data if trained correctly. Therefore, the primary recommendation and highest priority for proceeding with this project is to gather the required training data for the neural network as originally planned.

10.2. Hardware Setup and Image Processing

In Chapter 6, a conclusion resulted that a green sphere could not be recognized and thus no metadata could be collected. This means no experiments can be conducted with a green sphere, this does reduce the possibility to generalise. The hardware team and image processing team should make further investigation into finding colors that will work effectively, so experiments can be conducted with multiple spheres.

10.3. Image Processing and Deep Learning

As has been explained in Top-Level Design (Chapter 4, the choice was made to process images into 2D coordinates, radius and visibility. This metadata would be fed into the neural network, which would produce an output. Resulting this pipeline, it was concluded that the computational time for downloading would take around 2 seconds image processing would show at least 6 seconds of processing and deep learning was able to produce its result in milliseconds. This result is too large and thus the requirement is not met.

One solution to solve this problem is applying optimization for each process. The optimization for the downloading process has been explained in detail in the discussion of Chapter 8; design a buffer, so other processes don't have to wait on downloading. The image processing algorithm should be investigated to achieve faster computational time. The neural network does not need further optimisation regarding computational time, as it is already fast.

It can also be considered to reevaluate the design choice of extracting metadata from images. As image processing is highly computationally inefficient, this may have not been the correct design choice. Spreading the weight of computation time between image processing and deep learning more evenly may have been a better choice. Ideas for this have not been researched. It is recommended to first investigate optimisation for image processing before considering changing to a whole different design.

10.4. Hardware setup and Deep Learning

A key difference between data collection and simulation lies in the formation possibilities. In the collected data, spheres were connected with thin fishing wire, particularly in line formations, which inhibits the formation of certain curvatures. The small distance between the anchor points of opposing spheres allows for some flexibility but restricts the range of motion between the spheres.

This constraint permits snake-like chain formations, which are not generated in the simulation. The simulation enforces equal angles between the vectors spanning the centers of the spheres, unlike the actual scenario where each angle can vary. However, the simulation allows for a wider range of these angles. Although the simulation cannot generate snake-like formations, it defines a broader range of curvatures as lines, which the hardware physically cannot achieve. This discrepancy might impact model training, as data collected from actual experiments could differ. When the actual experiment is running, the model might misclassify formations, predicting a ring instead of a line due to a larger grey area of formation types in the simulation. For example, a 3/4 arc of a circle might be recognized as a ring instead of a line.

10.5. Future Recommendations

A future recommendation is to generate more data to train the neural network, ensuring it captures the full range of possible formations observed in real-world scenarios. This additional data should include various formations with inherent noise and inaccuracies, enabling the model to better handle real experimental conditions and improve its classification accuracy. By incorporating these diverse formations, the neural network can be more robust and reliable in distinguishing between similar shapes, reducing the likelihood of misclassifications.

The current model operates with three to six spheres. The spheres are small enough to fit a line of six within the cone. Training data is easily generated by reconstructing one ring and one line and running the experiment for a few hours. Six different colors are selected for the image processing model to maintain an accuracy of 99%. The deep learning model exhibits characteristics that show higher accuracy with more spheres. However, beyond a certain number of spheres, the model topology will need to be adjusted to prevent a likely decline in accuracy.

Another recommendation is to improve the image processing algorithms. Enhancing these algorithms can lead to more accurate detection and measurement of the spheres' positions and radii, reducing the noise and errors in the data. This can be achieved by employing more advanced techniques such as deep learning-based image segmentation, edge detection algorithms, or real-time tracking methods. Additionally, incorporating pre-processing steps like noise reduction, contrast enhancement, and image stabilization can further refine the input data quality. By improving the image processing pipeline, the overall accuracy and reliability of the model can be significantly increased, leading to better performance in real-world applications.

11

Conclusion

To better understand self-assembly processes, it is important to be able to research interactions between the particles. A macroscopic experimental setup containing magnetic dipole spheres in a turbulence-driven fluid is yet to be analysed. The turbulence of the water and the magnetic force of the spheres create an environment where the spheres interact with each other and form either a line or a ring. The research question at the beginning of the report stated:

"How do we automatically recognize 3D formations of magnetic spheres in line and ring formations?"

To answer this in short: the system can perform an automated process that provides a result based on a given 3D formation. However, the accuracy of this result is too low to be viable for practical use. On the other hand, promising working sub-modules have been demonstrated, which showed accurate results.

11.1. Submodule Conclusions

The pipeline contains the experimental setup and physics analysis, image processing, deep learning and data management. The camera takes pictures of the experiment from the top. The images are processed and individual spheres are recognized. Data is extracted from these spheres and fed to the deep learning model. The deep learning model is trained on generated labelled data and will give one of three cases as output: ring, line or none. The images and camera control will be done with a Raspberry Pi 5.

Spheres are designed to fulfil the requirements of the experimental setup and the image processing model. The camera together with the light source is placed at the top to take clear pictures without shadow. Training data for the deep learning model is generated by gluing the spheres together (in the case of a ring) and pulling a string through them (in the case of a line).

During the image processing stage, the spheres are detected on the images and their centre coordinates, radii and visibility are computed. The spheres are detected with an accuracy of 98.9%. The centre coordinates are computed with an accuracy of 99.8% and the radii with 98.4%. The average processing time is 6-7 seconds per image.

According to simulations, the deep learning model seems to correlate with the number of spheres and the model's accuracy. Having three spheres, an accuracy of over 95% is reached up to 99% for six spheres. The simulation showed that at least 2500 images per formation type per sphere are needed to train the model. Real training data has not been provided to the model, and therefore, it has to make real predictions based on simulated training data. The overall accuracy is 42.0%, taking into account *none* as well.

A Raspberry Pi 5 is used to control the experiment. The upload script to a cloud server has been successfully tested. Formatting the metadata retrieved from image processing into HDF5 proved to be working as well.

11.2. Top-Level Requirements

The most relevant requirements, as stated in section 2 are summed up below. The other top-level requirements can be found in Appendix B. If applicable, all design choices have been made upon these:

- **Training Time Efficiency:** *Training of the automated recognition module should take up less time than the manual evaluation of the spheres' formations.*

  Training data does not have to be labelled automatically anymore. By forcing a formation of spheres into a line (using the string) or a ring (gluing), the experiment can be performed for hours without any manual intervention.

- **Data Storage Size:** *The maximum allowable size for long-term data collected from a single experimental trial is 500 MB.*

  Every image is processed and certain data is extracted. Saving only the coordinates of the spheres, their radii and visibility drastically reduces the amount of data compared to the images themselves. Besides, formatting the data of a simulated experiment to a HDF5 format and a text file, results in a zip folder of 8.8 MB, thus complying with the requirement.

- **Formation Recognition:** *The method must be able to distinguish between a ring, a line and an undefined formation of a variable number of spheres.*

  As stated in the Accuracy requirement, the overall system lacks real-time training data and thus achieves a result of 42.0%. Although every sub-system has promising results, the overall implementation does not meet this requirement.

- **Automated Process:** *The developed algorithm/method must yield an automated execution process.*

  With the system integration developed as it is now, a fully automated process has been integrated. Capturing and uploading happen at the same time when the downloading and processing pipeline runs. This gives a direct output, which is saved accordingly. This integration meets the requirement of an automated process.

- **Accuracy:** *The overall system must have an accuracy of at least 95%.*

  The overall system has not met the set requirements due to the lack of real-time training data for the neural network. The achieved accuracy was 42.0%.

- **Validatability:** *The designed overall system must be capable of being validated to show the reliability and usability of the system.*

  Each image is able to be validated if its computed output is correct. This is possible by observing the direct output in the terminal and post-evaluation from the saved HDF5 format.

- **Computational Time:** *The computation time of the automatised recognition module should not take more than 1 hour past the experiment duration.*

  The total computational time includes 2 seconds for downloading, 6 seconds for image processing, and milliseconds for deep learning. This sums up to a total of 8 seconds, which does not meet the requirement of processing an image every second. This results in a computational time that is 8 times longer than the experiment duration, as an image is taken every second. For experiments lasting 12-24 hours, this leads to significant delays in computation.

References

[1] Leon Abelmann et al. “Self-Assembled Three-Dimensional Non-Volatile Memories”. In: *Micromachines* 1.1 (2010), pp. 1–18. ISSN: 2072-666X. DOI: 10.3390/mi1010001. URL: https://www.mdpi.com/2072-666X/1/1/1.

[2] Anzar Alam, Mohd. Abdullah, and Ravi Shankar Mishra. “Colour Contrast Enhancement Method by Scaling the DC Coefficients in CIE-LAB Colour Space”. In: *International Journal of Computer Applications* 97 (July 2014), pp. 1–6. ISSN: 0975-8887. DOI: 10.5120/17136-7371. URL: https://dx.doi.org/10.5120/17136-7371.

[3] Donald Bailey, Yuan Chang, and Steven Le Moan. “Analysing Arbitrary Curves from the Line Hough Transform”. In: *Journal of Imaging* 6.4 (2020). ISSN: 2313-433X. DOI: 10.3390/jimaging6040026. URL: https://www.mdpi.com/2313-433X/6/4/26.

[4] G. Bradski. “The OpenCV Library”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[5] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. “Non-Local Means Denoising”. In: *Image Processing On Line* 1 (2011). https://doi.org/10.5201/ipol.2011.bcm_nlm, pp. 208–212.

[6] William Gemmell Cochran. *Sampling techniques*. en. Wiley, 1977, p. 428. ISBN: 978-0-471-16240-7.

[7] A.P. Dijkshoorn. “Design, Fabrication and Testing of 3D-Printed Spheres for Macro Self-Assembly Experiments”. PhD thesis. University of Twente, 2016.

[8] Tim Dobbert. *Matchmoving*. en. Wiley & Sons, Limited, John, 2012, p. 336. ISBN: 978-1-118-52966-9.

[9] U. Gasser and B. Zhou. “Accurate detection of spherical objects in a complex background”. In: *Opt. Express* 29.23 (Nov. 2021), pp. 37048–37065. DOI: 10.1364/OE.434652. URL: https://opg.optica.org/oe/abstract.cfm?URI=oe-29-23-37048.

[10] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing, Global Edition*. en. Pearson Higher Education, 2017, p. 1024. ISBN: 9781292223049.

[11] T. A. G. Hageman et al. “Macroscopic equivalence for microscopic motion in a turbulence driven three-dimensional self-assembly reactor”. In: *Journal of Applied Physics* 123 (Jan. 2018), Not available. ISSN: 0021-8979, 1089-7550. DOI: 10.1063/1.5007029. URL: https://dx.doi.org/10.1063/1.5007029.

[12] Tijmen Antoon Geert Hageman. “Observing magnetic objects in fluids”. English. PhD thesis. Netherlands: University of Twente, Saarland University, Mar. 2018. ISBN: 978-90-365-4481-8. DOI: 10.3990/1.9789036544818.

[13] Ramandeep Kaur and Dipen Saini. “Image Enhancement of Underwater Digital Images by Utilizing L*A*B* Color Space on Gradient and CLAHE based Smoothing”. In: *Communications on Applied Electronics* 4 (Apr. 2016), pp. 22–30. ISSN: 2394-4714. DOI: 10.5120/cae2016652166. URL: https://dx.doi.org/10.5120/cae2016652166.

[14] Per Arvid Löthman. “Macroscopic magnetic Self assembly”. English. PhD thesis. Netherlands: University of Twente, Mar. 2018. ISBN: 978-90-365-4512-9. DOI: 10.3990/1.9789036545129.

[15] Kohtaro Ohba, Yoichi Sato, and Katsusi Ikeuchi. “Appearance-based visual learning and object recognition with illumination invariance”. In: *Machine Vision and Applications* 12 (Dec. 2000), pp. 189–196. ISSN: 0932-8092, 1432-1769. DOI: 10.1007/s001380050138. URL: https://dx.doi.org/10.1007/s001380050138.

[16] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[17] *Raspberry Pi 5 Product Brief*. Raspberry Pi Ltd. Apr. 2024.

[18] Manish Singh and Jacqueline M. Fulvio. “Visual extrapolation of contour geometry”. In: *Proceedings of the National Academy of Sciences* 102.3 (2005), pp. 939–944. DOI: 10.1073/pnas.0408444102. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.0408444102. URL: https://www.pnas.org/doi/abs/10.1073/pnas.0408444102.

[19] *Speedtest by Ookla*. https://www.speedtest.net/nl.

[20] Yiqi Tew, Wee Yan Lee, and Ga Men Tam. “Parallel Computing Using CUDA and MultiThreading in Background Removal Process”. In: *2024 3rd International Conference on Digital Transformation and Applications (ICDXA)*. 2024, pp. 237–242. DOI: 10.1109/ICDXA61007.2024.10470572.

[21] Alberto Fernández Villán. *Mastering OpenCV 4 with Python*. en. Packt Publishing Ltd, 2019, p. 532. ISBN: 9781789344912.

[22] Atsushi Yamashita, Megumi Fujii, and Toru Kaneko. “Color Registration of Underwater Images for Underwater Sensing with Consideration of Light Attenuation”. In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 4570–4575. DOI: 10.1109/ROBOT.2007.364183.

[23] Sahar Zafari et al. “Resolving overlapping convex objects in silhouette images by concavity analysis and Gaussian process”. In: *Journal of Visual Communication and Image Representation* 73 (Nov. 2020), p. 102962. ISSN: 1047-3203. DOI: 10.1016/j.jvcir.2020.102962. URL: https://dx.doi.org/10.1016/j.jvcir.2020.102962.

A

Report organisation

A.1. Contributions to the report

Table A.1: The contributions of each team member to the report

Chapter of the report	**Contributed individual(s)**
Abstract & Preface	Belle & Peter
Introduction	Emma & Sten
Top-Level Requirements	Belle & Peter
Top-Level Design Choices	Benji
Top-Level Implementation	Emma
Experimental Setup & Physics	Belle & Sten
Image Processing	Emma & Peter
Deep Learning	Benji
Data Management	Guido
System Integration	Guido
Discussion	Benji & Guido
Conclusion	Sten
References	Emma
Appendix	All team members

B

Conclusion Top-Level Requirements

- **Time Frame:** *The implementation of the project must be feasible in a time frame of three weeks.*

  Every subsystem has been able to implement its design and generate promising results. The overall design, however, faced some difficulties which led to a low overall accuracy. Even though the final product is not fully functioning, some very useful results and conclusions have been gathered in the process. Therefore, this requirement has been met.
- **Costs:** *The total cost budget of this project is €2000.*

  The total costs of the project have been around €300 for the setup and €200 for data management. Consequently, the requirement has been met.
- **Setup Compatibility:** *The measurement method must be practically feasible and compatible with the current setup without necessitating any modifications.*

  In fact, it is not a question of whether this requirement has been met, but multiple design decisions come from the compatibility with the current setup. A camera mould has been designed and assembled to fit the camera and flashlight into the small cylinder at the top. Both the camera and the flashlight had to be small to fit in the cylinder. The only thing that has been changed to the setup is the filtering of the water to increase visibility.
- **Portability:** *The developed automated system must be portable. It should be designed such that the transportation of the system is convenient.*

  The Raspberry Pi 5 provides a platform for capturing and uploading images while being compact and easily transportable, during the use of multiple experiments. Therefore, this requirement is met.
- **Accessibility:** *The system should be easy and remotely accessible.*

  The images that are generated are uploaded to a Google Drive and therefore very easily and remotely accessible. Also, the Raspberry Pi itself is easily accessible through the use of a VNC connection.
- **Reproducible Experiment:** *The measurement methods must allow for reproducible experiments to ensure consistent results.*

  The training formations (the forced line and ring formations) are created in such a way that it is possible to reproduce new data without changing the formations. Besides, the camera mold is produced to ensure the stability of the experiment.
- **Generalisability:** *The developed method must be generalisable to any structure and any shape of the spheres in the structure.*

  The spheres can easily be reproduced and painted in different colours. The training data can be assembled the same way. For image processing, there is a limit for the different amount of

colours that can be used (and thus the maximum number of spheres).

- **Experiment Duration:** *The test setup must be able to run the experiment for a duration of 12 to 24 hours. Also, the developed method must be able to handle and store the corresponding amount of data.*

  During the design of the spheres, especially the choice of the paint was important as it had to remain intact for multiple experiments of 12 to 24 hours. Furthermore, the shape of the spheres needs to remain intact during these experiments. However, the flashlight did not meet this requirement, as it could not charge and produce light at the same time.

C

Source Code

C.1. GitHub Relevant Branches

Link to Git repository: GitHub

(https://github.com/gvanhuizen/BEPSelfAssembly)

Github Branches	
Main	Overall main branch for the main program (not updated yet as no working version)
Version1	The first implementation version; no trained neural network as for now
RPI5	The Raspberry configuration code
MaskAlgorithm	The Neural Network design and analysis
ImageProcessingPeter	Part of image processing analysis
ImageProcessingEmma	Part of image processing analysis
Implementation	Design and testing of the whole system implementation

C.2. Image Processing

C.2.1. HSV Colour Range Selection

```
import cv2
import numpy as np

def nothing(x):
    pass

# Load image
image = cv2.imread('noise_remove_background_removed_00000_bias_image_colorenhance_no_clahe.
    png')
if image is None:
    print("Error: Unable to open/read file")
    exit()

# Create a window
cv2.namedWindow('image')

# Create trackbars for color change
# Hue is from 0-179 for OpenCV
cv2.createTrackbar('HMin', 'image', 0, 179, nothing)
cv2.createTrackbar('SMin', 'image', 0, 255, nothing)
cv2.createTrackbar('VMin', 'image', 0, 255, nothing)
cv2.createTrackbar('HMax', 'image', 0, 179, nothing)
cv2.createTrackbar('SMax', 'image', 0, 255, nothing)
cv2.createTrackbar('VMax', 'image', 0, 255, nothing)

# Set default value for Max HSV trackbars
cv2.setTrackbarPos('HMax', 'image', 179)
```

```
cv2.setTrackbarPos('SMax', 'image', 255)
cv2.setTrackbarPos('VMax', 'image', 255)

# Initialize HSV min/max values
hMin = sMin = vMin = hMax = sMax = vMax = 0
phMin = psMin = pvMin = phMax = psMax = pvMax = 0

while True:
    # Get current positions of all trackbars
    hMin = cv2.getTrackbarPos('HMin', 'image')
    sMin = cv2.getTrackbarPos('SMin', 'image')
    vMin = cv2.getTrackbarPos('VMin', 'image')
    hMax = cv2.getTrackbarPos('HMax', 'image')
    sMax = cv2.getTrackbarPos('SMax', 'image')
    vMax = cv2.getTrackbarPos('VMax', 'image')

    # Set minimum and maximum HSV values to display
    lower = np.array([hMin, sMin, vMin])
    upper = np.array([hMax, sMax, vMax])

    # Convert to HSV format and color threshold
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower, upper)
    result = cv2.bitwise_and(image, image, mask=mask)

    # Print if there is a change in HSV value
    if (phMin != hMin or psMin != sMin or pvMin != vMin or phMax != hMax or psMax != sMax or
    pvMax != vMax):
        print(f"(hMin = {hMin}, sMin = {sMin}, vMin = {vMin}), (hMax = {hMax}, sMax = {sMax},
     vMax = {vMax})")
        phMin = hMin
        psMin = sMin
        pvMin = vMin
        phMax = hMax
        psMax = sMax
        pvMax = vMax

    # Display result image
    cv2.imshow('image', result)
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

cv2.destroyAllWindows()
```

Listing C.1: HSV Thresholding Tool

C.2.2. Final implementation

Image Processing Main Function code

```
import sys
import os
import cv2
import numpy as np

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), 'Sphere_Isolation')))
from run_files_main import main_sphere_isolation

sys.path.append(os.path.abspath(os.path.join(os.path.dirname(__file__), 'Property_extraction'
    )))
from Main_Property_Extraction import main_property_extraction

# The main function of the property extraction algorithm
def extract_sphere_feature_main(image_path):

    # Load the image
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)

    # Isolate each sphere in the image and save each sphere in a list of binary images of
    isolated spheres
    dictionary_of_binary_isolated_circles = main_sphere_isolation(image_path)
```

```
    sphere_properties = []

    # Iterate through all the binary files of the isolated spheres
    for color, bin_image in dictionary_of_binary_isolated_circles.items():
        # Compute for each isolated sphere its radius, center and visibility
        properties_dictionary = main_property_extraction(bin_image, color, image)
        print(f"Extracted {color}")

        # Remove the unnecessary 'found' parameter from the properties_dictionary
        properties_dictionary.pop('found')

        # Append the property information of the isolated sphere
        sphere_properties.append(properties_dictionary)

    return sphere_properties
```

Listing C.2: Image Processing Main function

Image Processing sphere isolation code

```
import cv2
import numpy as np
from PIL import Image
from LAB_process_image import enhance_and_process_image
from background_removal_process import remove_background
from denoise_process import denoise_image
from colour_threshold_process import process_and_blend_colors

def main_sphere_isolation(image_path):

    # Step 1: Enhance and process the image
    processed_image = enhance_and_process_image(image_path)

    # Convert the processed image to a format compatible with the rembg library
    processed_image_pil = Image.fromarray(cv2.cvtColor(processed_image, cv2.COLOR_BGR2RGB))

    # Step 2: Remove the background
    processed_image_pil = remove_background(processed_image_pil)

    # Convert back to OpenCV format\sub
    processed_image = cv2.cvtColor(np.array(processed_image_pil), cv2.COLOR_RGB2BGR)

    # Step 3: Denoise the image
    processed_image = denoise_image(processed_image)

    # Step 4: Process and blend colors
    isolated_spheres_binary_images = process_and_blend_colors(processed_image)

    return isolated_spheres_binary_images
```

Listing C.3: Image Processing Main sphere isolation function

```
import cv2
import numpy as np

def enhance_and_process_image(image_path):
    # Load image
    image = cv2.imread(image_path)

    # Convert the denoised image to the LAB color space
    lab_image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)

    # Split the LAB image into its channels
    l, a, b = cv2.split(lab_image)

    # Apply CLAHE to the L-channel
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8, 8))
    cl = clahe.apply(l)

    # Merge the CLAHE enhanced L-channel with the a and b channels
```

```
    enhanced_image = cv2.merge((cl, a, b))

    # Convert the LAB image back to BGR color space
    enhanced_image = cv2.cvtColor(enhanced_image, cv2.COLOR_LAB2BGR)

    # Smooth edges using Gaussian blur
    blurred_image = cv2.GaussianBlur(enhanced_image, (5, 5), 0)

    # Create a kernel for the morphological operations
    kernel = np.ones((5, 5), np.uint8)

    # Apply dilation to connect edges
    dilated_image = cv2.dilate(blurred_image, kernel, iterations=3)

    # Apply erosion to refine the edges
    eroded_image = cv2.erode(dilated_image, kernel, iterations=3)

    return eroded_image
```

Listing C.4: Image Processing enhance and process image function

```
from rembg import remove
from PIL import Image

def remove_background(image):
    # Remove the background
    output_image = remove(image)
    return output_image
```

Listing C.5: Image Processing remove background function

```
import cv2

def denoise_image(image):
    # Apply Non-Local Means Denoising
    denoised_image = cv2.fastNlMeansDenoisingColored(image, None, 10, 3, 11, 21)
    return denoised_image
```

Listing C.6: Image Processing denoise image function

```
import cv2
import numpy as np

def process_and_blend_colors(image):
    # Convert the image to HSV color space
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

    # Define the color ranges
    color_ranges = {
        'red': (np.array([1, 0, 0]), np.array([19, 255, 114])),
        'yellow': (np.array([27, 104, 233]), np.array([35, 255, 255])),
        'roze': (np.array([155, 37, 186]), np.array([179, 255, 255]))
    }

    def process_color(mask, color_name):
        # Smooth edges using Gaussian blur
        blurred_mask = cv2.GaussianBlur(mask, (5, 5), 0)

        # Create a kernel for the morphological operations
        kernel = np.ones((5, 5), np.uint8)

        # Apply dilation and erosion to connect edges
        dilated = cv2.dilate(blurred_mask, kernel, iterations=3)
        eroded = cv2.erode(dilated, kernel, iterations=3)

        # Save the resulting binary image to a file
        cv2.imwrite(f'Image_Procesing_Implementation/Test_Images/{color_name}_binary_WA0093.
    png', eroded)

        return eroded

```

```
    # Initialize a dictionary to hold the binary images
    binary_images = {}

    # Iterate through each color range and process the corresponding mask
    for color_name, (lower, upper) in color_ranges.items():
        mask = cv2.inRange(hsv, lower, upper)
        binary_images[color_name] = process_color(mask, color_name)

    return binary_images
```

Listing C.7: Image Processing color thresholding function

Image Processing Sphere Reconstruction code

```
from Radius_Center_V2 import compute_center_radius
from Visibility import main_visibility
from Radius_Center_V2 import find_largest_region_features
import cv2
from skimage import measure
import numpy as np


def main_property_extraction(isolated_sphere_binary_image, color, image):
    min_radius = 50
    max_radius = 150

    # Load the binary image of the isolated sphere as binary file
    _, binary_image = cv2.threshold(isolated_sphere_binary_image, 1, 255, cv2.THRESH_BINARY)

    # Compute the x and y center coordinates and the radius of the visible part of the sphere
    x, y, radius, found = compute_center_radius(binary_image, None, None, max_radius, None)

    # Make an empty image with the same dimensions as the binary input image
    height, width = binary_image.shape
    empty_image = np.full((height, width), 0, dtype=np.uint8)

    # Try to make a binary picture by reconstructing the circle from the isolated contour
    try:
        reconstructed_circle = cv2.circle(empty_image, (int(x), int(y)), int(radius), 255,
    -1)

        # Determine the visibility
        visibility = main_visibility(binary_image, reconstructed_circle)
    except:
        visibility = None
        pass

    # The output of the main property function is a dictionary with the following values
    properties = {
        "x" : x,
        "y" : y,
        "radius" : radius,
        "visibility" : visibility,
        "found" : found,
        "color" : color
        }
    print(properties)

    return properties
```

Listing C.8: Image Processing Sphere Reconstruction main function

```
    def compute_center_radius(binary_image, labeled_image, largest_region, max_radius,
    regions):
    # Compute the parameters of the visible part of the circle by applying the Hough
    Transform
    x, y, radius, found = HoughCicles_Reconstruction(binary_image)

    return x, y, radius, found
```

Listing C.9: Image Processing Sphere apply Hough Transform function

```
    import cv2
import numpy as np

def HoughCicles_Reconstruction(image):

    # Find the edges in the binary image
    edges = cv2.Canny(image, 50, 150)

    # Apply the HoughCircles function on the edges
    circles = cv2.HoughCircles(edges,
                               cv2.HOUGH_GRADIENT,
                               dp=4.7,
                               minDist=132,
                               param1=100,
                               param2=100,
                               minRadius=66,
                               maxRadius=117)

    # Determine whether a circle is found and set the x, y and radius values
    if circles is not None:
        found = True
        for circle in circles:
            circles = np.round(circles[0, :]).astype("float")
            circle = circles[0]
            x, y, radius = circle
    else:
        found = False
        x = None
        y = None
        radius = None
    return x, y, radius, found
```

Listing C.10: Image Processing Sphere Hough Transformfunction

```
import numpy as np
import cv2

def jaccard_similarity(img1, img2):
    intersection = np.logical_and(img1, img2)
    union = np.logical_or(img1, img2)
    if np.sum(union) == 0:
        return 0
    else:
        return np.sum(intersection) / np.sum(union)

def dice_coefficient(img1, img2):
    intersection = np.logical_and(img1, img2)
    if np.sum(img1) + np.sum(img2) == 0:
        return 0
    else:
        return 2.0 * np.sum(intersection) / (np.sum(img1) + np.sum(img2))

def main_visibility(detected_circle, reconstructed_circle):
    # Binarize the images (in case they are not binary)
    _, detected_circle = cv2.threshold(detected_circle, 127, 255, cv2.THRESH_BINARY)
    _, reconstructed_circle = cv2.threshold(reconstructed_circle, 127, 255, cv2.THRESH_BINARY
    )

    # Calculate Jaccard similarity coefficient
    jaccard_score = jaccard_similarity(detected_circle, reconstructed_circle)
    print("Jaccard Similarity Coefficient:", jaccard_score)

    # Calculate Dice coefficient
    dice_score = dice_coefficient(detected_circle, reconstructed_circle)
    print("Dice Coefficient:", dice_score)

    return jaccard_score
```

Listing C.11: Image Processing compute visibility function

D

Requirement Trade-Off Tables

D.1. Image Processing

D.1.1. Camera position

Camera position	Weights	Side/Horizontal	Top/Vertical
Time frame feasibility: Consideration of the implementation time compared to the acquired results	8	5	4
Detection Accuracy	7	2	4
Chance of success: How big is the risk factor when using a certain method?	10	2	4
Computational efficiency: How much computational power is required when recognizing the formation?	8	4	4
Ease of Segmentation	5	2	5
Traceability: Ability to track the calculations/steps used in the method and debug possible errors	7	3	4
Generalisability: Does the method allow for generalisability in size, variability of sphere, structure and formation?	8	4	4
Validatability: Can the accuracy of the results be verified?	10	3	3
Robustness: How robust is the algorithm to minor changes in the environment of the test setup?	5	2	4
Bias prone: How likely will a bias be introduced in the algorithm with the applied method?	7	3	3
Total	75	230	288
Weighted average	-	3,06	3,84

Table D.1: Camera Position Selection

D.1.2. Circle Features Extraction

Criterion	Weight	Immediate Circle Detection	Binary Thresholding + Reconstruction
Accuracy	9	3	4
Processing Speed	6	2	4
Complexity	6	4	3
Robustness	6	3	5
Scalability	3	3	4
Flexibility	3	2	4
Resource Usage	2	3	3
Result	35	102	138
Weighted Score	-	2.91	3.94

Table D.2: Circle Features Extraction Methods

D.1.3. Preprocessing Techniques

Technique	Selected (Yes/No)	Reason
LAB Image - CLAHE Enhanced L-channel	Yes	Enhances contrast in the luminance channel.
Gaussian Blur	Yes	Reduces noise and detail.
Background Removal	Yes	Removes unnecessary parts of the image.
Non-Local Means Denoising	Yes	Effectively removes noise while preserving details.
Contrast and Brightness Adjustment	No	Less effective compared to other methods.
Remove Shade with Low Frequencies	No	Redundant with other selected methods.
Glare Removal using CLAHE	No	Removed too much datail and information.
Temperature Adjustment	No	Not necessary for the scope of this project.

Table D.3: Preprocessing Technique Alternatives

D.1.4. Sphere Colours

Spheres	Weights	Unique Colour	Same Colour	Half-Half
Time frame feasibility: Consideration of the implementation time compared to the acquired results	8	4	2	1
Detection Accuracy	7	5	2	4
Chance of success: How big is the risk factor when using a certain method?	10	5	1	4
Computational efficiency: How much computational power is required when recognizing the formation?	8	5	3	2
Ease of Segmentation	5	5	2	4
Traceability: Ability to track the calculations/steps used in the method and debug possible errors	7	5	2	3
Generalisability: Does the method allow for generalisability in size, variability of sphere, structure and formation?	8	2	5	5
Validatability: Can the accuracy of the results be verified?	10	2	5	4
Robustness: How robust is the algorithm to minor changes in the environment of the test setup?	5	4	3	4
Bias prone: How likely will a bias be introduced in the algorithm with the applied method?	7	3	5	4
Total	75	294	228	261
Weighted average	-	3,92	3,04	3,48

Table D.4: Sphere Colour Selection

D.1.5. Sphere Detection

Circle detection algorithms	Weights	Colour Threshold - Binary Isolation	Contour, Concave, Segment Grouping	Canny Edge Detection	Watershed Algorithm	Gradient Detection
Time frame feasibility: Consideration of the implementation time compared to the acquired results	8	5	1	2	2	3
Accuracy	10	3	5	4	4	3
Chance of success: How big is the risk factor when using certain method?	10	4	3	2	2	2
Computational efficiency: How much computational power is required when recognizing the formation?	8	3	1	2	2	3
Traceability: Ability to track the calculations/steps used in the method and debug possible errors	7	4	4	3	4	4
Generalisability: Does the method allow for generalisability in size, variability of sphere, structure and formation?	5	3	5	5	5	4
Validatability: Can the accuracy of the results be verified?	10	3	4	3	3	3
Robustness: How robust is the algorithm to minor changes in the environment of the test setup?	5	3	5	4	3	3
Data variability (features): What is the variance in data that can be collected? What type of data can we collect?	4	4	4	2	2	2
Data storage: What is the amount of data required for the algorithm? And how much data is outputted?	8	5	3	3	3	5
Bias prone: How likely will a bias be introduced in the algorithm with the applied method?	4	2	3	3	3	3
Total	79	286	266	232	234	251
Weighted average	-	3,62	3,37	2,94	2,96	3,18

Table D.5: Sphere Detection Algorithms

D.1.6. Circle Reconstruction Algorithm

Initial Evaluation

Circle detection algorithms	Weights	Bayesian extrapolation	Matched Filter	minEnclosing	Hough Transform
Time frame feasibility: Consideration of the implementation time compared to the acquired results	8	4	4	5	5
Accuracy	10	2	3	5	4
Chance of success: How big is the risk factor when using certain method?	10	4	3	5	5
Computational efficiency: How much computational power is required when recognizing the formation?	8	2	2	3	2
Traceability: Ability to track the calculations/steps used in the method and debug possible errors	7	4	4	4	4
Generalisability: Does the method allow for generalisability in size, variability of sphere, structure and formation?	5	4	5	5	3
Validatability: Can the accuracy of the results be verified?	10	4	4	4	4
Robustness: How robust is the algorithm to minor changes in the environment of the test setup?	5	5	3	5	5
Data variability (features): What is the variance in data that can be collected? What type of data can we collect?	4	3	3	4	3
Data storage: What is the amount of data required for the algorithm? And how much data is outputted?	8	5	3	5	5
Bias prone: How likely will a bias be introduced in the algorithm with the applied method?	4	3	4	4	4
Total	79	285	268	354	322
Weighted average	-	3,61	3,39	4,48	4,08

Table D.6: Circle Reconstruction Algorithms

Updated Circle Reconstruction Algorithm Evaluation

Circle detection algorithms	Weights	Bayesian extrapolation	Matched Filter	minEnclosing	Hough Transform
Time frame feasibility: Consideration of the implementation time compared to the acquired results	8	4	4	5	5
Accuracy	10	2	3	**2**	5
Chance of success: How small is the risk factor when using certain method?	10	4	3	**2**	5
Computational efficiency: How much computational power is required when recognizing the formation?	8	2	2	3	2
Traceability: Ability to track the calculations/steps used in the method and debug possible errors	7	4	4	4	4
Generalisability: Does the method allow for generalisability in size, variability of sphere, structure and formation?	5	4	5	**3**	3
Validatability: Can the accuracy of the results be verified?	10	4	4	4	4
Robustness: How robust is the algorithm to minor changes in the environment of the test setup?	5	5	3	**3**	5
Data variability (features): What is the variance in data that can be collected? What type of data can we collect?	4	3	3	4	3
Data storage: What is the amount of data required for the algorithm? And how much data is outputted?	8	5	3	5	5
Bias prone: How likely will a bias be introduced in the algorithm with the applied method?	4	3	4	**3**	4
Total	79	285	268	270	332
Weighted average	-	3,61	3,39	3,42	4,20

Table D.7: Circle Reconstruction Algorithms

D.1.7. Temporal Image Storage

Temporal Image Storage	Weights	.PNG	.BIN	PSD	JPEG
File size	10	3	2	2	5
Quality	8	5	5	5	3
Re-editing	8	5	3	5	3
Compatibility	7	5	3	3	5
Lossless Compression	7	5	5	5	2
Total	37	180	140	156	147
Weighted Average	-	4,86	3,78	4,22	3,97

Table D.8: Temporal Image Storage Format

D.1.8. Visibility Metrics

Visibility Metrics	Weights	Jaccard	Dice	Fuzzy Logic	Easy Ratio
Accuracy	10	5	5	3	2
Robustness	10	4	4	3	2
Simplicity	7	4	4	3	5
Detail	7	4	4	5	1
Computational Efficiency	5	3	3	2	5
Ease of Implementation	10	4	4	3	5
Result	49	201	201	156	157
Weighted Average		4.10	4.10	3.18	3.20

Table D.9: Visibility Metrics Evaluation

D.2. Data Management

Requirement	Mini-PCs/SBCs	Microcontrollers	IP Cameras	Smartphones/Tablets	Streaming Devices	Custom-built Systems
Budget-friendly	++	++	-	-	+	+/-
Performance	+	-	+/-	+	+/-	+
Remote Connectivity	++	+	+	++	+	+
Scalability	+	-	-	+	-	+
Ease of Use & Setup	++	+	-	-	+	-
Reliability	+	+	+	+	+	+
USB Connectivity for Cameras	+	+	-	-	-	-

Table D.10: Comparison of various devices based on different requirements; Without numbers very clear which one is better.

File Format	Weight	CSV	JSON	YAML	HDF5	TOML
Small storage size	7	2	2	2	4	3
Speed (high)	5	1	2	2	4	2
Metadata accessibility	8	2	3	3	4	4
Human Readable	2	2	2	1	3	2
Structure design complementary	10	1	3	3	5	3
Easiness in use	5	5	4	2	2	1
”Add-ons”	2	1	1	2	1	2
Total	**39**	**76**	**104**	**94**	**148**	**106**
Weighted Average	-	1.948717949	2.666666667	2.41025641	3.794871795	2.717948718
% Difference with Max	-	94.74%	42.31%	57.45%	0.00%	39.62%

Table D.11: Comparison of file formats based on various criteria

D.3. Top Level

D.3.1. Language Selection

Programming Language	Weights	Python	Matlab	C++
Open source	7	5	1	5
Future proof How well is the algorithm able to adopt to future improvements and updates of the programming language?	5	5	4	5
Prior knowledge of the programming language	10	5	4	2
Simplicity How simple is it to implement the algorithms? What libraries are available?	7	5	4	2
Prototyping speed	5	5	4	2
Performance	5	3	3	5
Total	39	145	95	94
Weighted average	-	3,72	2,44	2,41

Table D.12: Language Selection for the Image Processing Program

E

Results and Figures

E.1. Image Processing

E.1.1. Images captured from the side and top for comparison

Illustrative sample images captured from the side

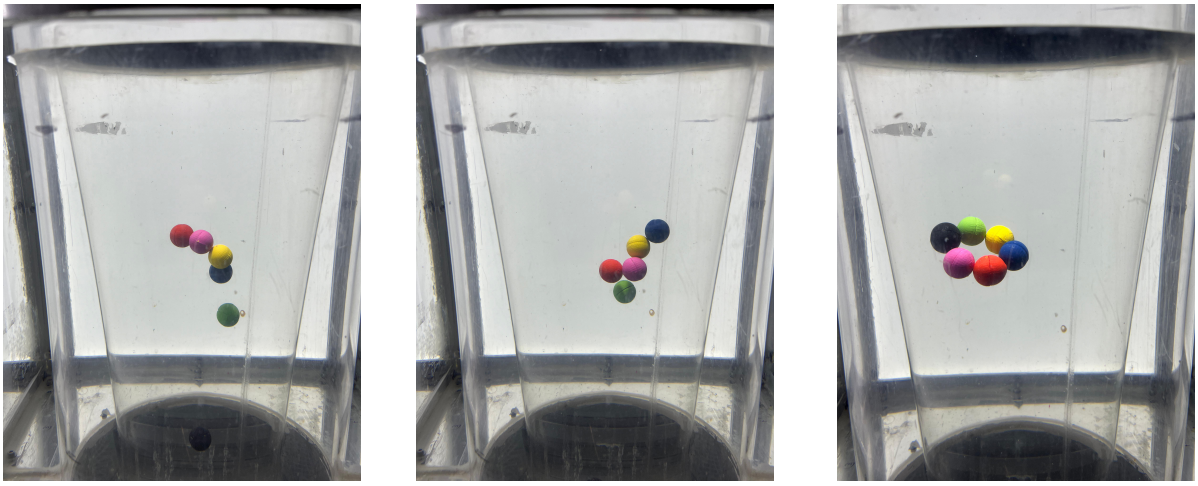


Figure E.1: Illustrative images from the side of the test setup captured with mobile phone

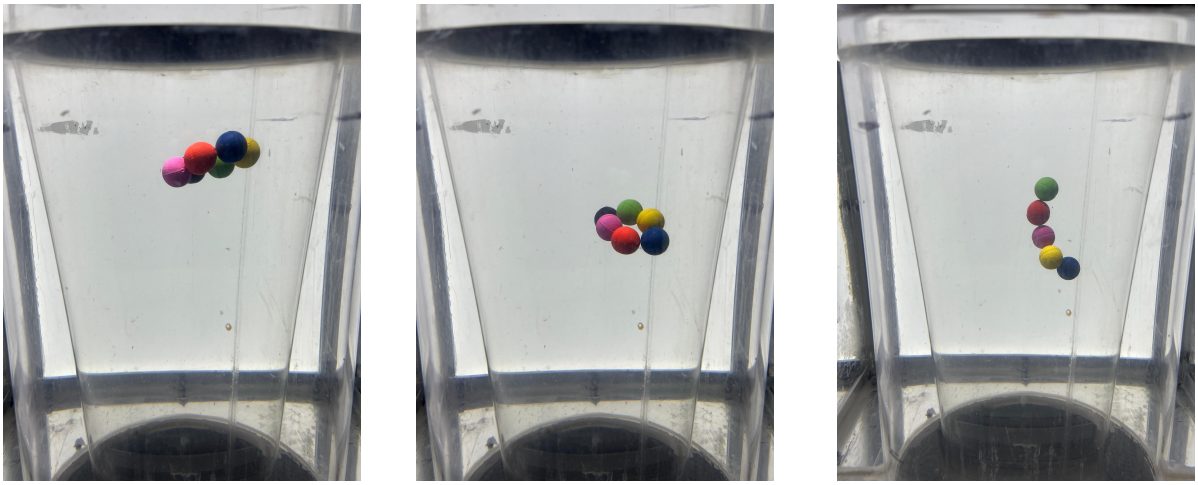


Figure E.2: Illustrative images from the side of the test setup captured with mobile phone

Illustrative sample images captured from the top

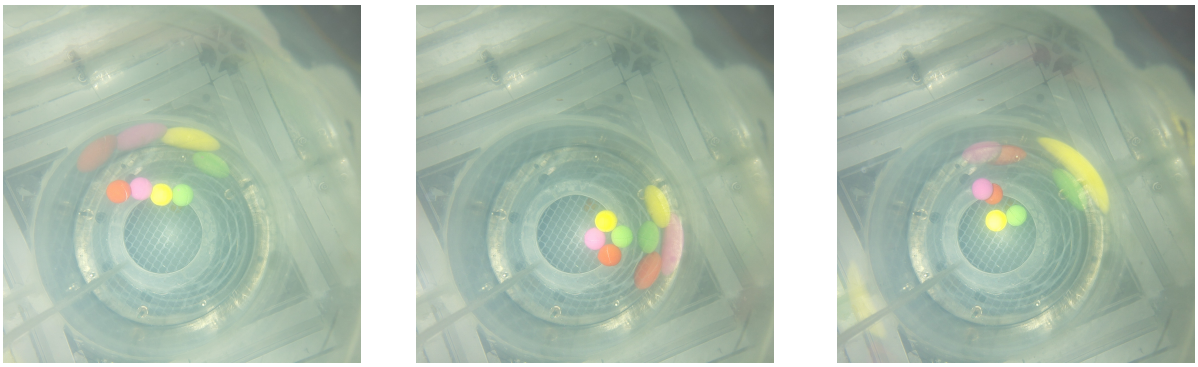


Figure E.3: Illustrative images from the top of the test setup captured with mobile phone

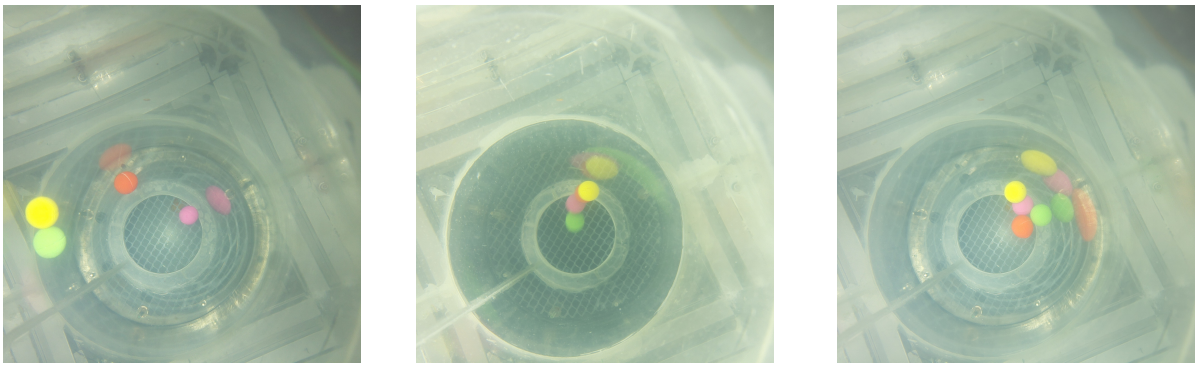


Figure E.4: Illustrative images from the top of the test setup captured with mobile phone

E.1.2. min_Enclosing result images

The min_Enclosing method can accurately compute the radius and center of the circle when more than half of the circumference of the circle is visible. However, due to the varying radius of the to be reconstructed sphere in the test setup due to a varying depth of the sphere in the tube, it is not possible to set up correct boundary conditions when the min_Enclosing algorithm should be applied.

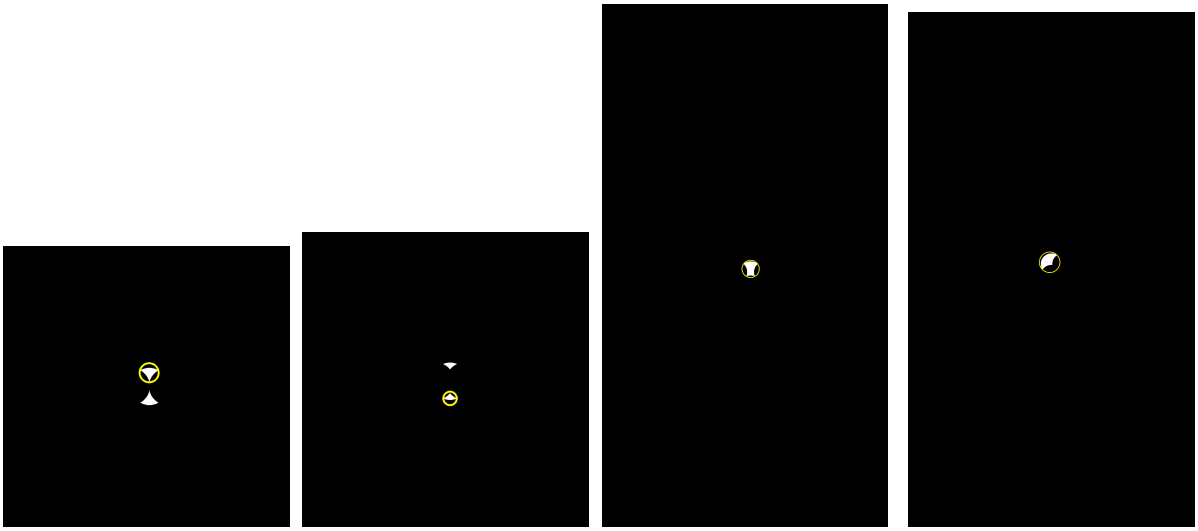


Figure E.5: Reconstructed sphere computed with the min_Enclosing algorithm

E.1.3. Visibility test images set with # occluding spheres

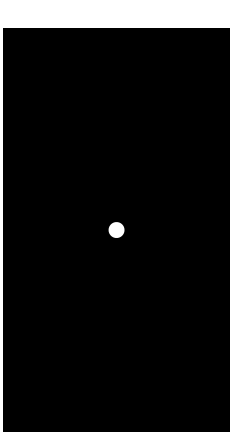


Figure E.6: None

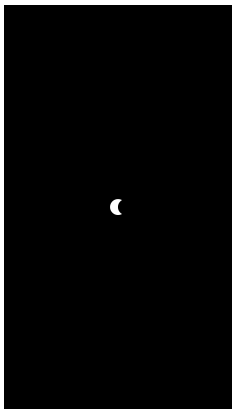


Figure E.7: One

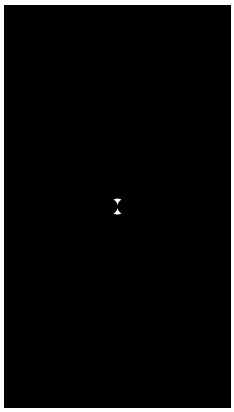


Figure E.8: Two Opposite

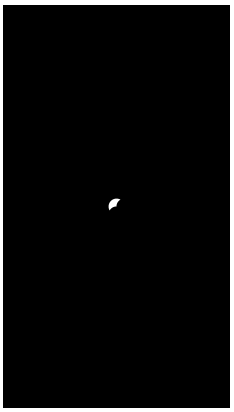


Figure E.9: Two Perpendicular

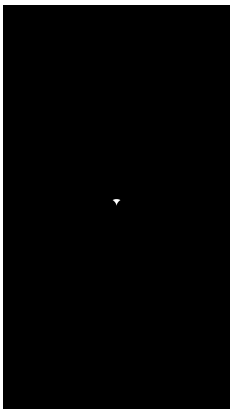


Figure E.10: Three

Figure E.11: Visibility test images set with # occluding spheres

E.1.4. Radii test images set with # occluding spheres

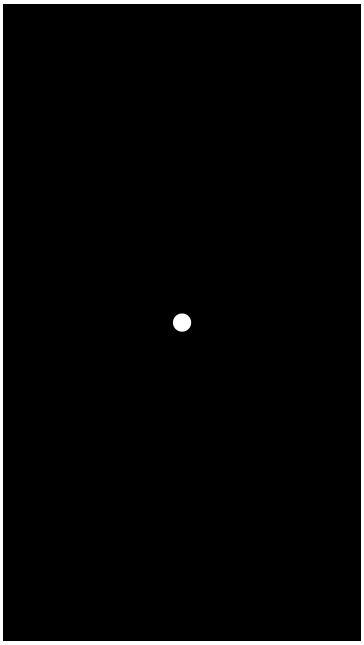


Figure E.12: 66

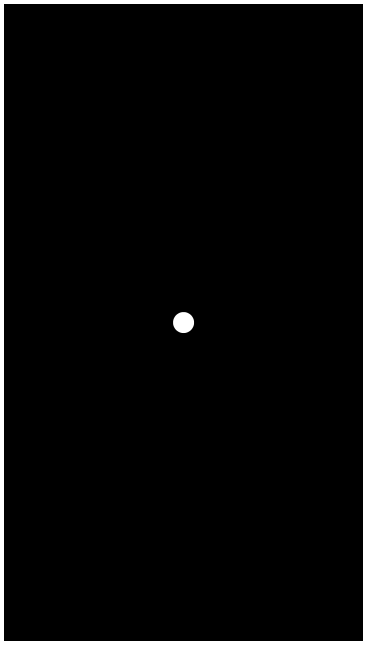


Figure E.13: 76

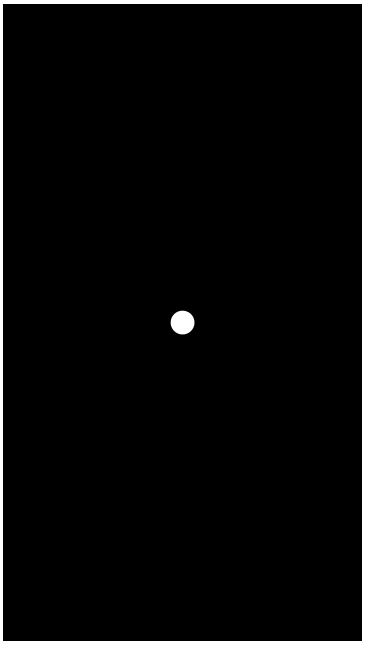


Figure E.14: 86

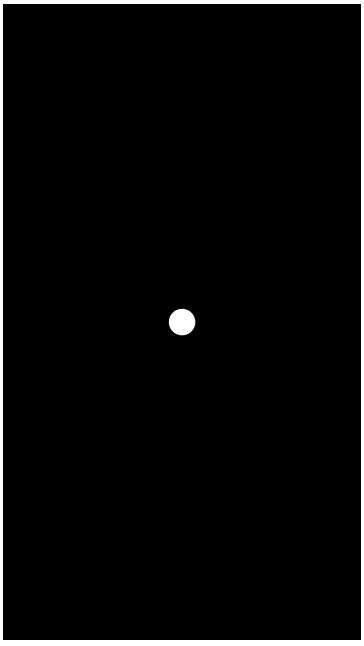


Figure E.15: 96

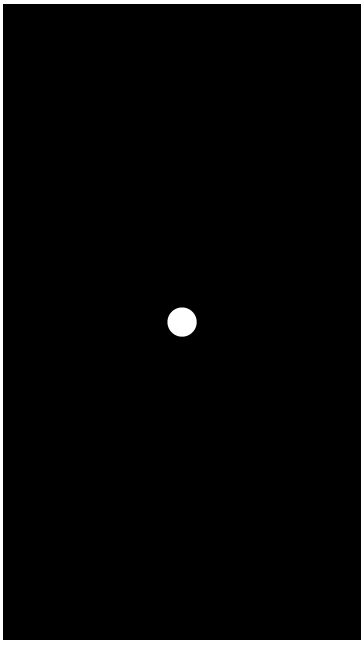


Figure E.16: 106

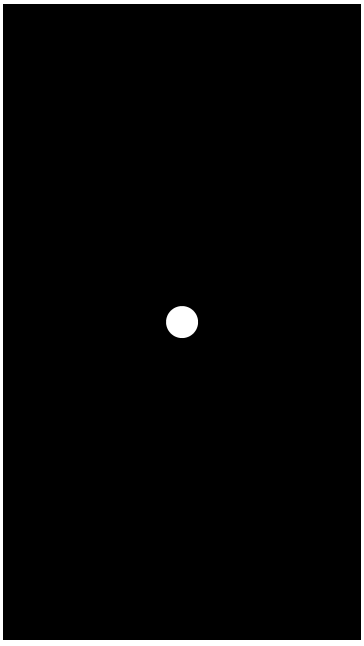


Figure E.17: 116

Figure E.18: Radius test images set with [#] pixels radius

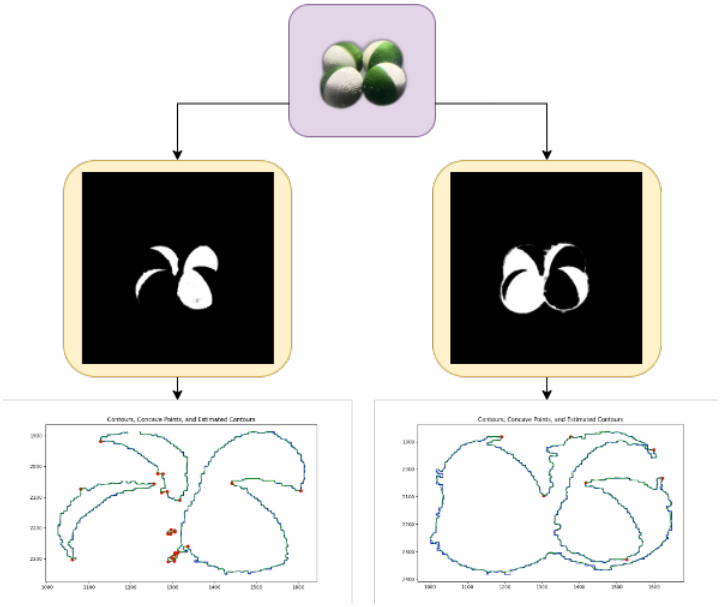


Figure E.19: Half-and-Half Spheres Isolation using Contour, Concave, Segment Algorithm

E.1.5. Graphs exploring parameter adjustment Hough Transform

Exploring dp adjustment

The following two graphs show the effect of changing the dp value on the amount of error in the computed radius.

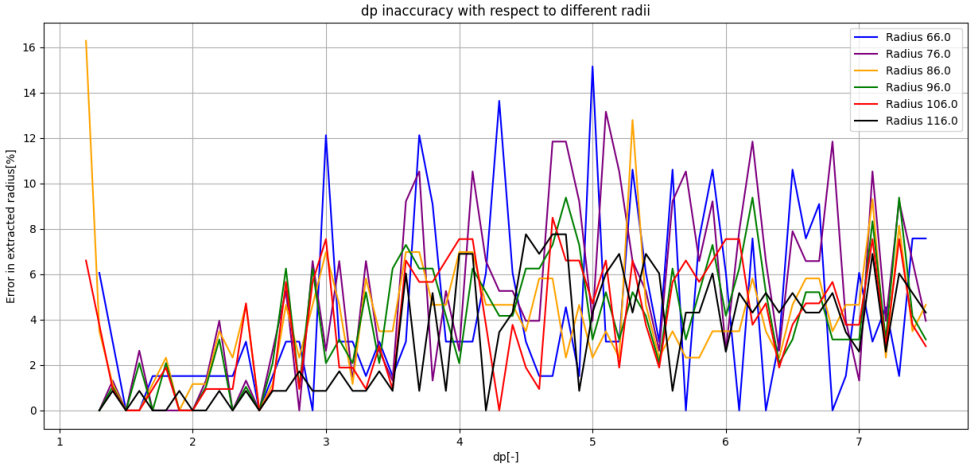


Figure E.20: Effect of dp adjustment on the computed radius inaccuracy for different radii

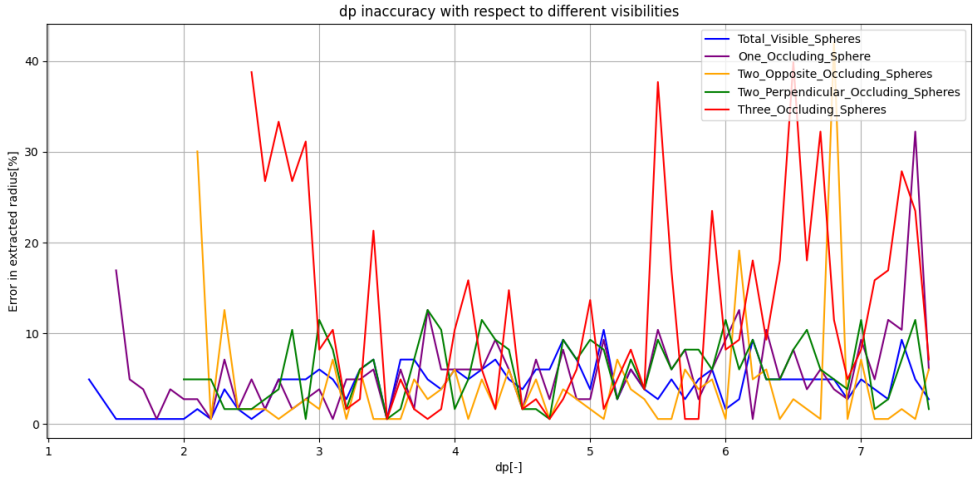


Figure E.21: Effect of dp adjustment on the computed radius inaccuracy for different visibilities

The following graphs show the effect of changing the dp value on the amount of error in the computed center x- and y-coordinate.

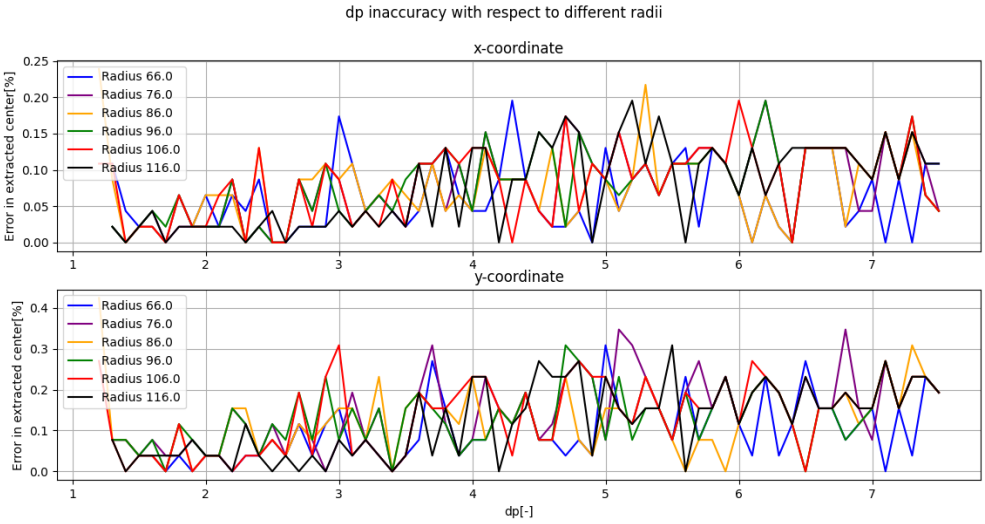


Figure E.22: Effect of dp adjustment on the computed center inaccuracy for different radii

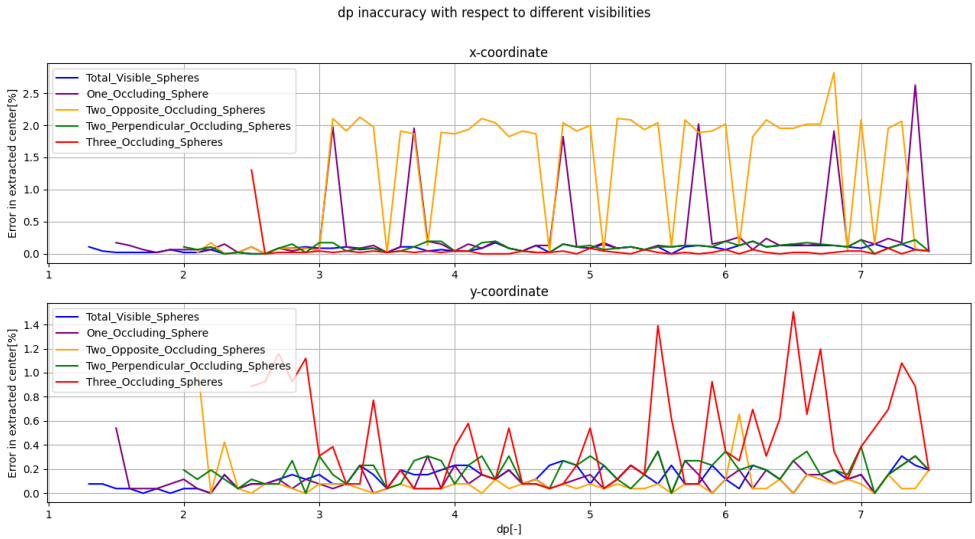


Figure E.23: Effect of dp adjustment on the computed center inaccuracy for different visibilities

Exploring param1 adjustment

The following two graphs show the effect of changing the param1 value on the amount of error in the computed radius.

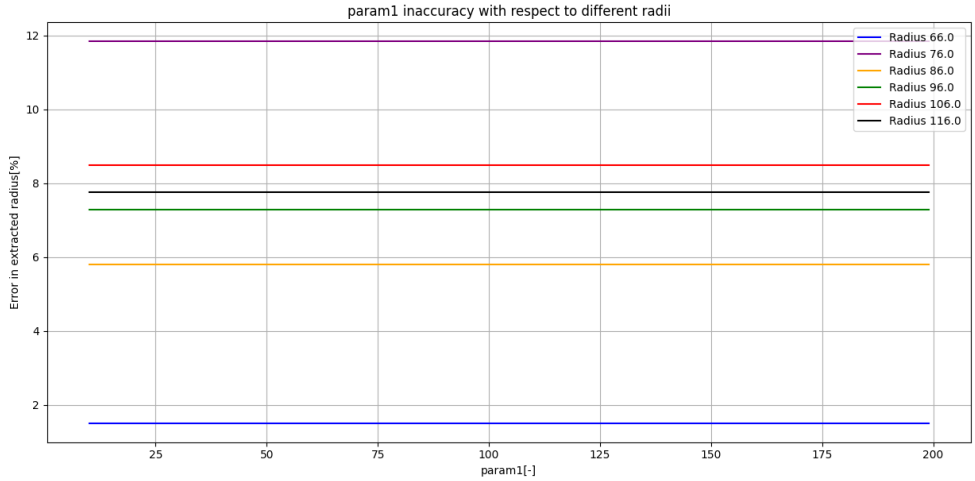


Figure E.24: Effect of param1 adjustment on the computed radius inaccuracy for different radii

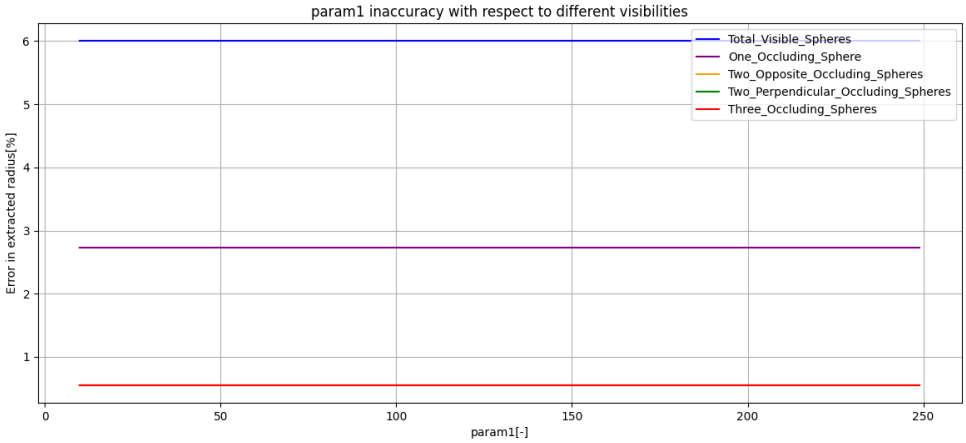


Figure E.25: Effect of param1 adjustment on the computed radius inaccuracy for different visibilities

The following graphs show the effect of changing the param1 value on the amount of error in the computed center x- and y-coordinate.

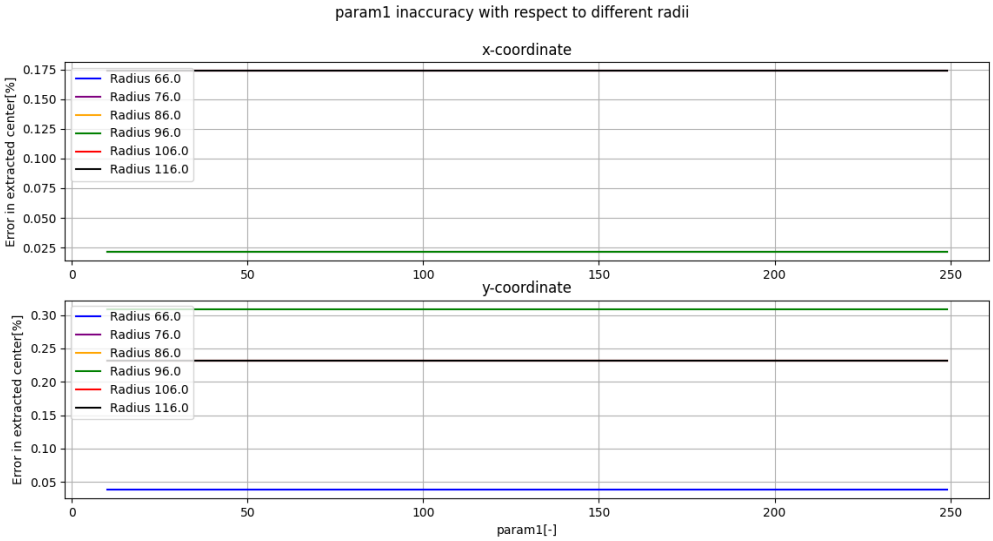


Figure E.26: Effect of param1 adjustment on the computed center inaccuracy for different radii

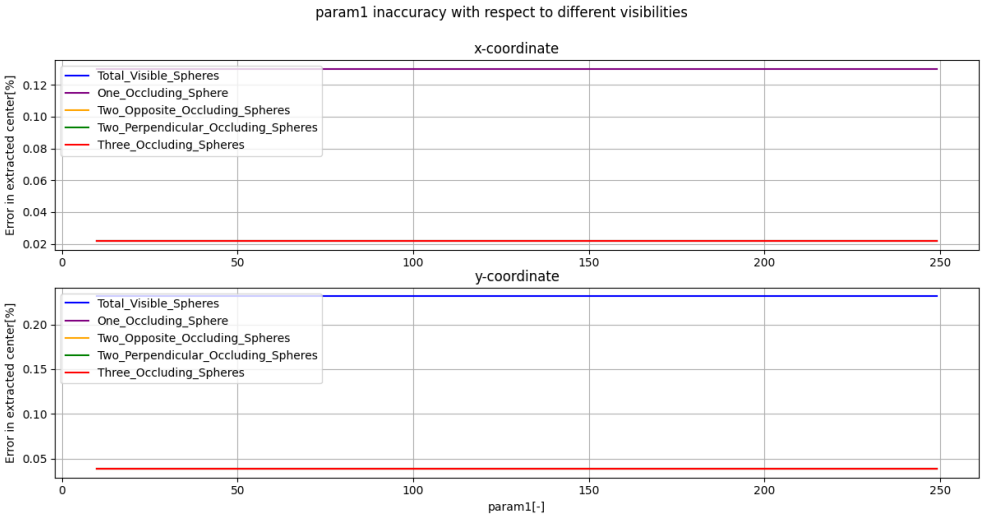


Figure E.27: Effect of param1 adjustment on the computed center inaccuracy for different visibilities

Exploring param2 adjustment

The following two graphs show the effect of changing the param2 value on the amount of error in the computed radius.

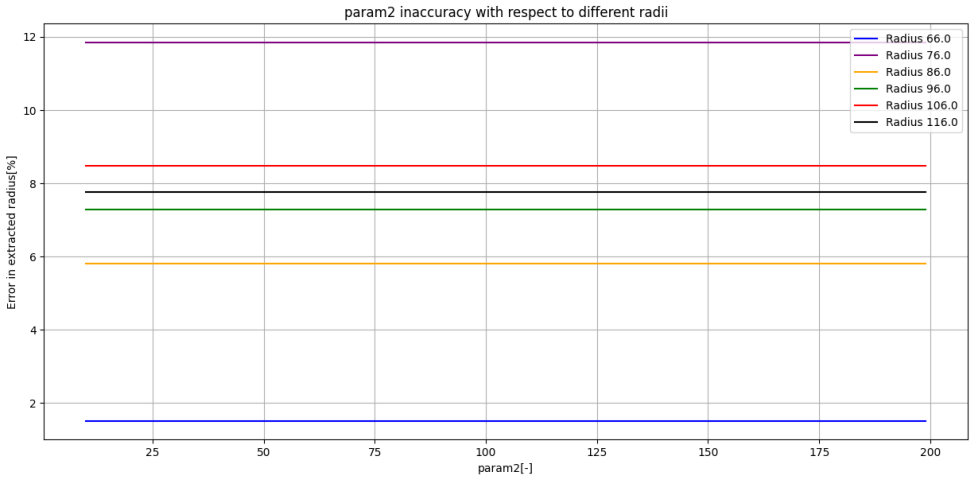


Figure E.28: Effect of param2 adjustment on the computed radius inaccuracy for different radii

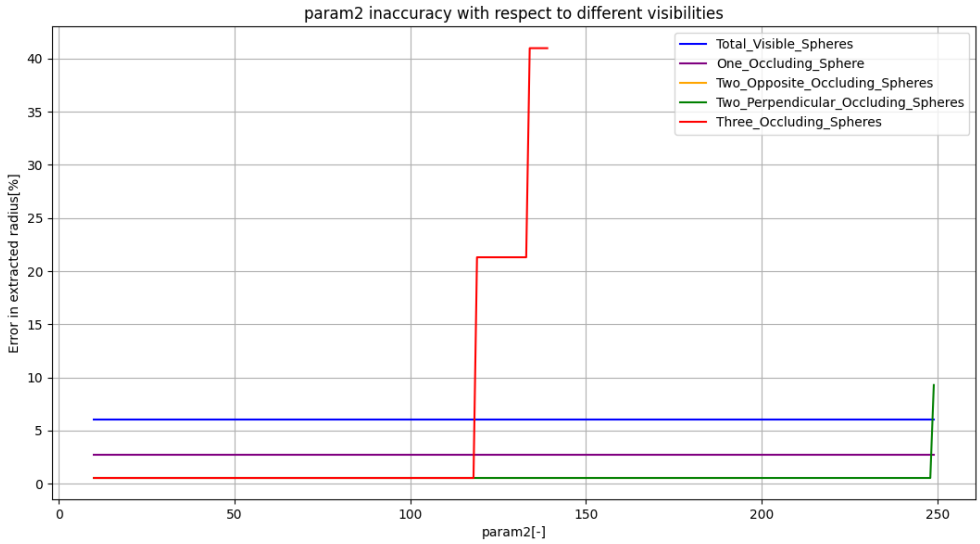


Figure E.29: Effect of param2 adjustment on the computed radius inaccuracy for different visibilities

The following graphs show the effect of changing the param2 value on the amount of error in the computed center x- and y-coordinate.

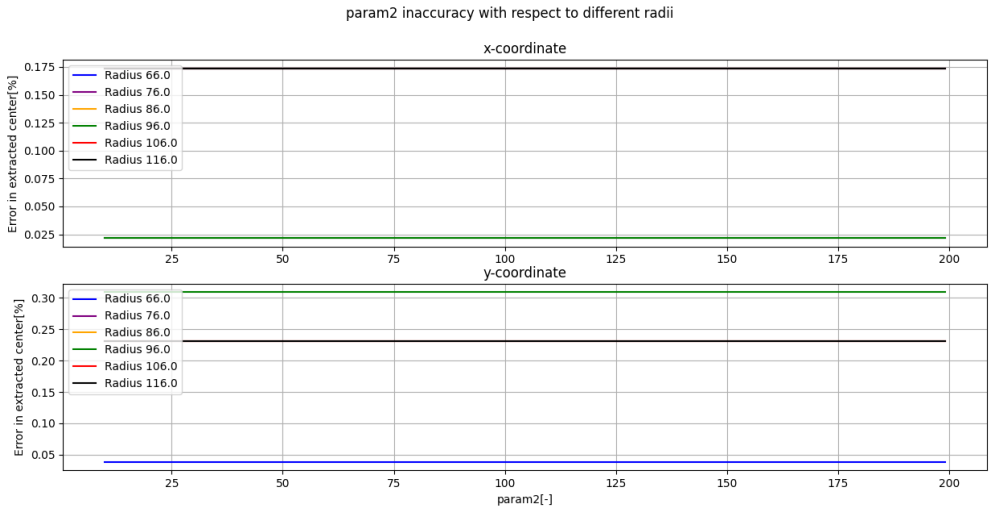


Figure E.30: Effect of param2 adjustment on the computed center inaccuracy for different radii

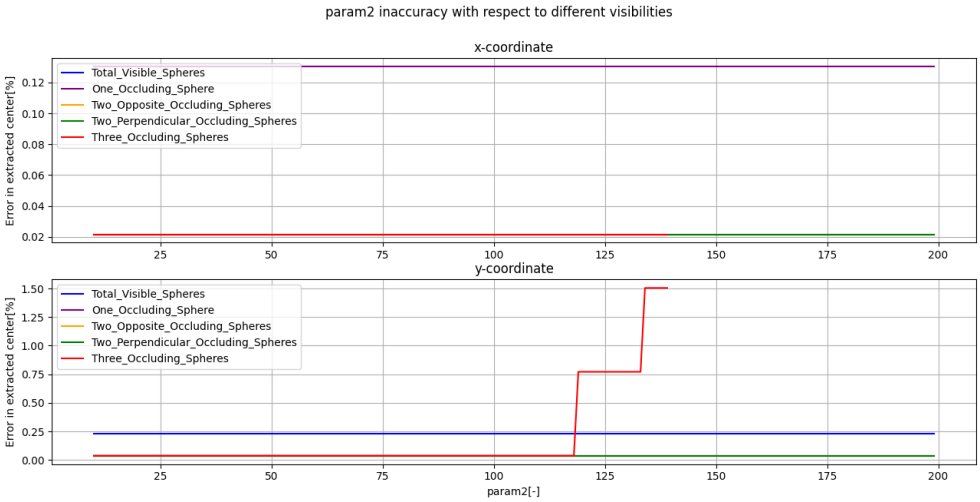


Figure E.31: Effect of param2 adjustment on the computed centre inaccuracy for different visibilities

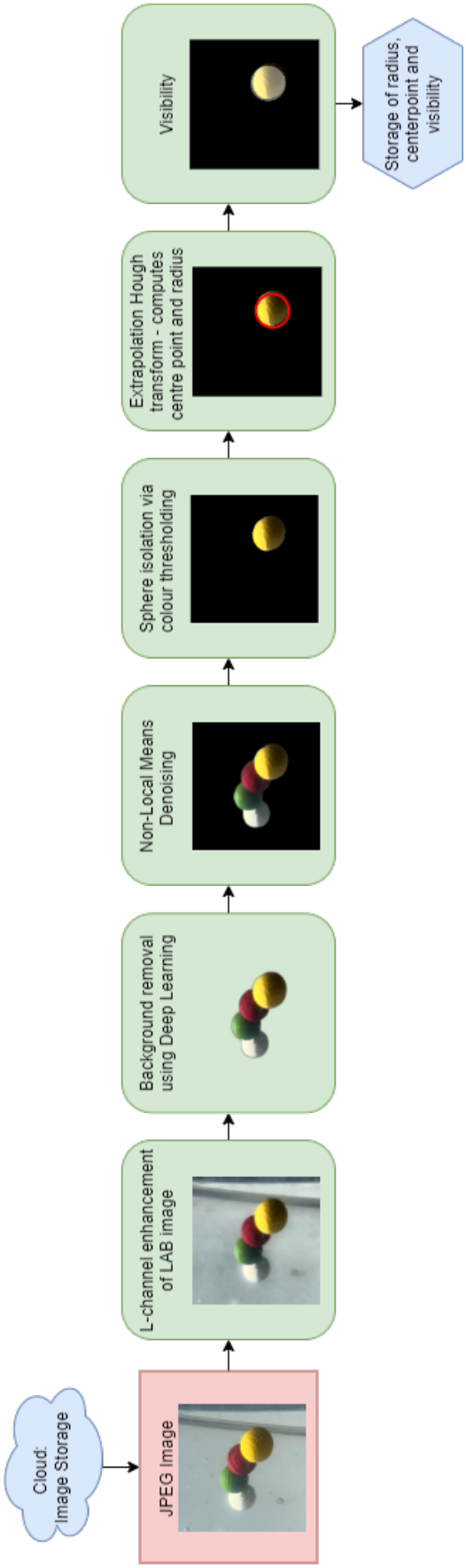


Figure E.32: Pipeline of the Image Processing Implementation

E.1.6. Test Set images for testing the image processing algorithm

Sample images of radii based test images set

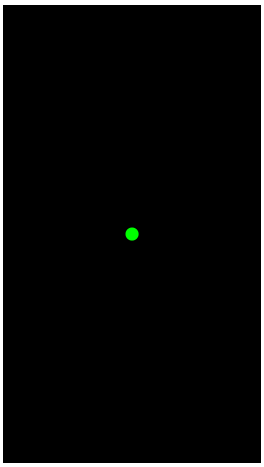


Figure E.33: r = 66.0

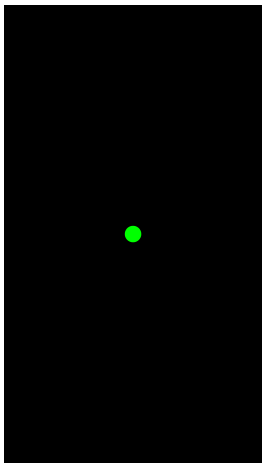


Figure E.34: r = 83.5

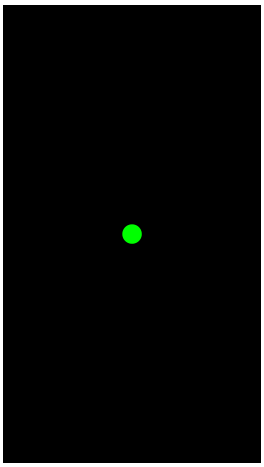


Figure E.35: r = 98.5

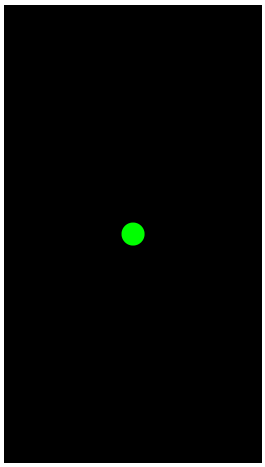


Figure E.36: r = 116.0

Figure E.37: Radii differing color test images set

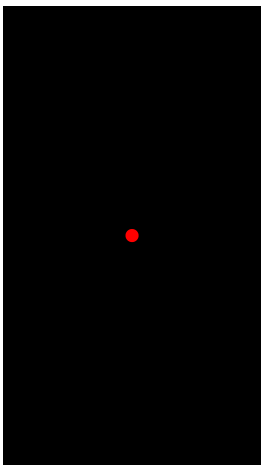


Figure E.38: r = 66.0

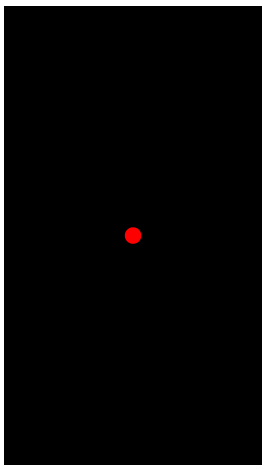


Figure E.39: r = 83.5

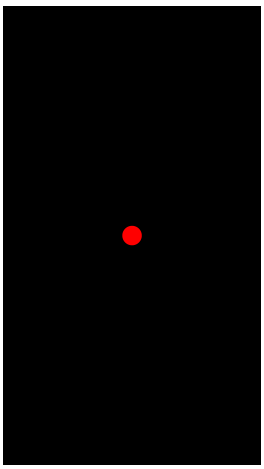


Figure E.40: r = 98.5

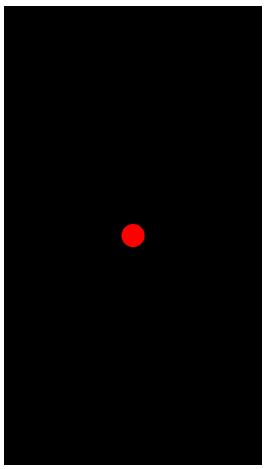


Figure E.41: r = 116.0

Figure E.42: Radii differing color test images set

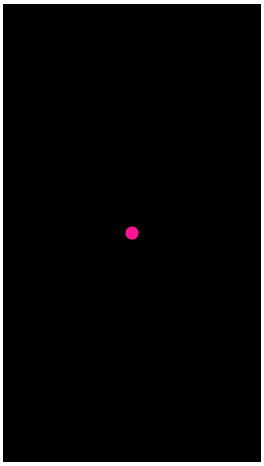


Figure E.43: r = 66.0

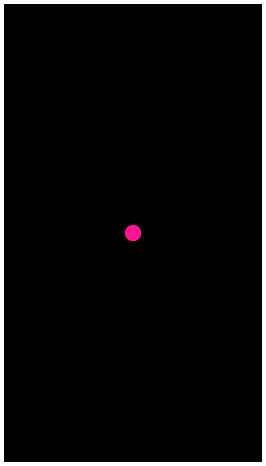


Figure E.44: r = 83.5

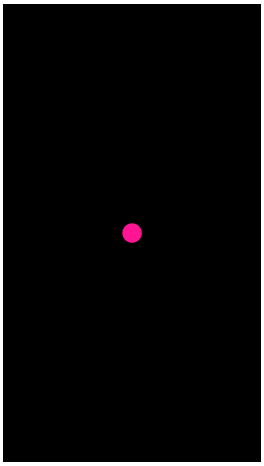


Figure E.45: r = 98.5

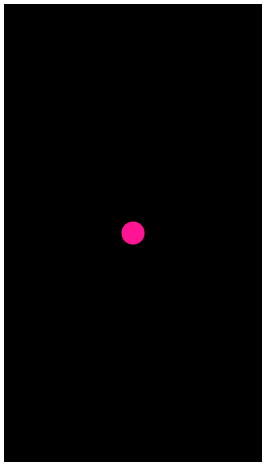


Figure E.46: r = 116.0

Figure E.47: Radii differing color test images set

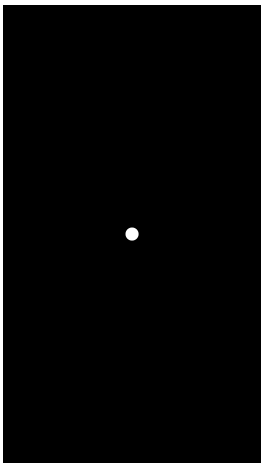


Figure E.48: r = 66.0

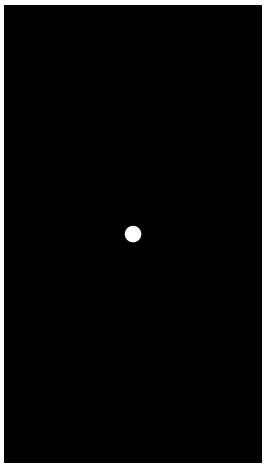


Figure E.49: r = 83.5

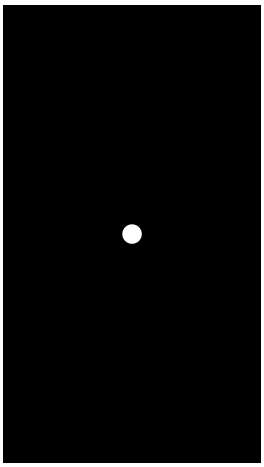


Figure E.50: r = 98.5

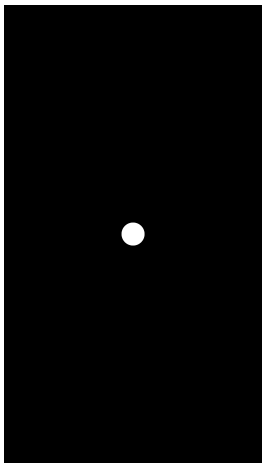


Figure E.51: r = 116.0

Figure E.52: Radii differing color test images set

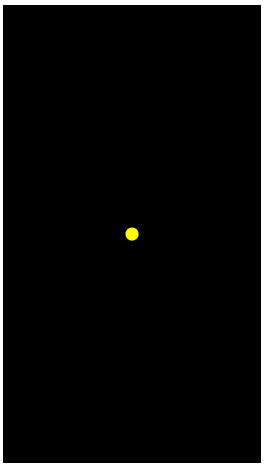


Figure E.53: r = 66.0

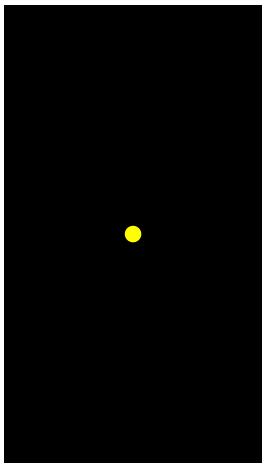


Figure E.54: r = 83.5

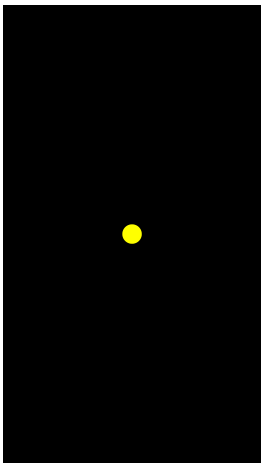


Figure E.55: r = 98.5

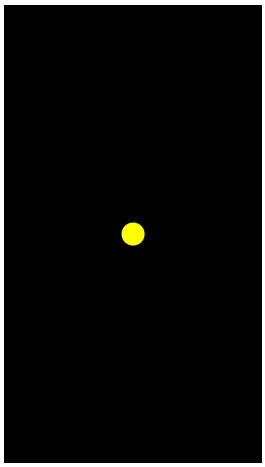


Figure E.56: r = 116.0

Figure E.57: Radii differing color test images set

Sample images of visibility based test images set

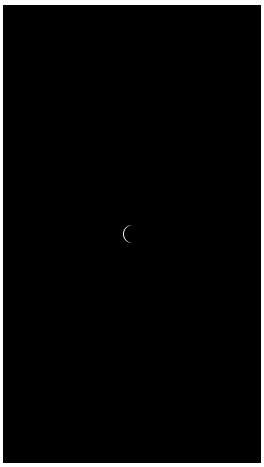


Figure E.58: d = 10

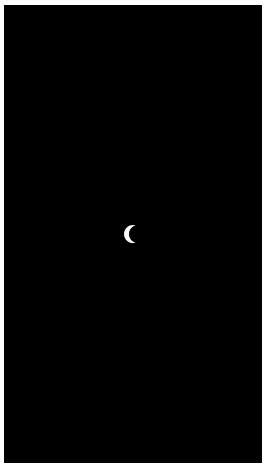


Figure E.59: d = 50

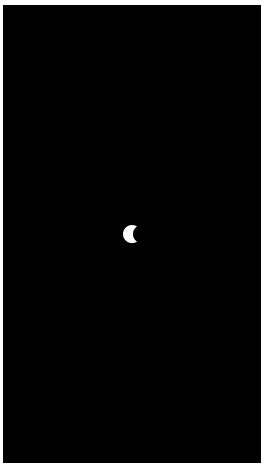


Figure E.60: d = 100

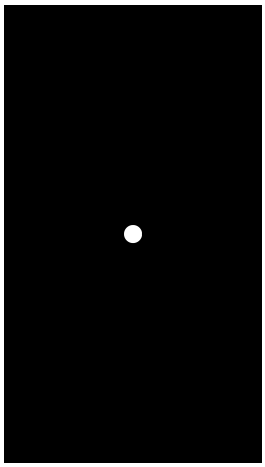


Figure E.61: d = 180

Figure E.62: Visibility differing color test images set with 1 occluding sphere with center displacement d pixels

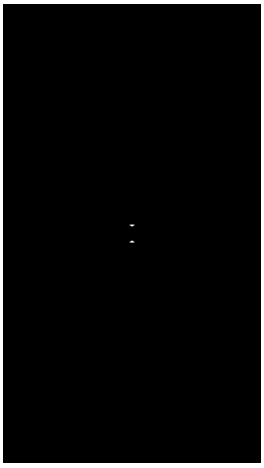


Figure E.63: d = 60

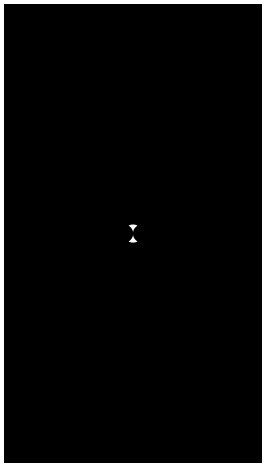


Figure E.64: d = 90

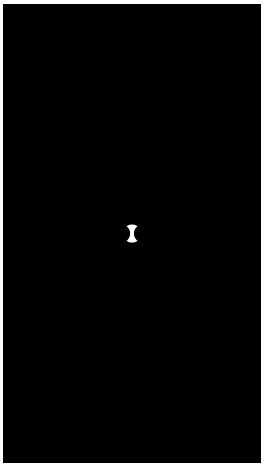


Figure E.65: d = 110

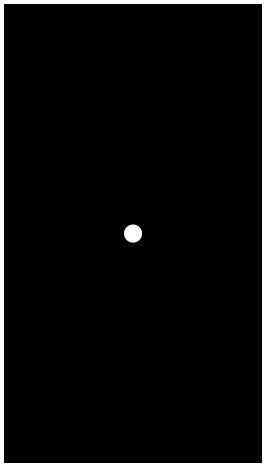


Figure E.66: d = 180

Figure E.67: Visibility differing color test images set with 2 opposite occluding spheres with center displacement d pixels

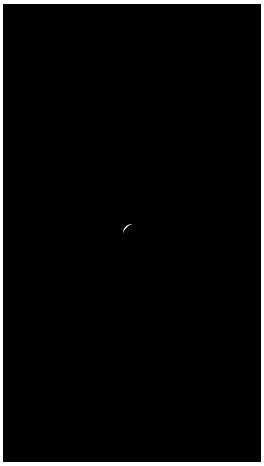


Figure E.68: d = 20

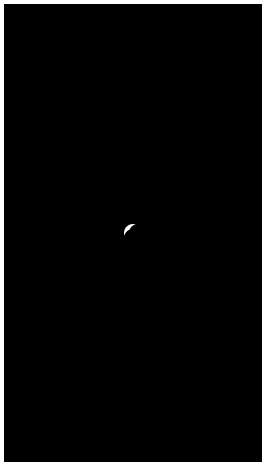


Figure E.69: d = 50

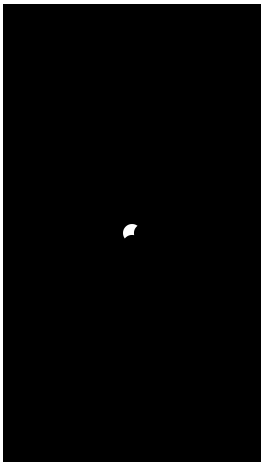


Figure E.70: d = 110

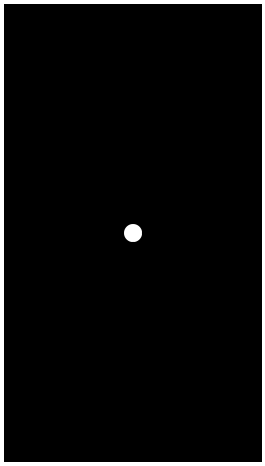


Figure E.71: d = 180

Figure E.72: Visibility differing color test images set with 2 perpendicular occluding spheres with center displacement d pixels

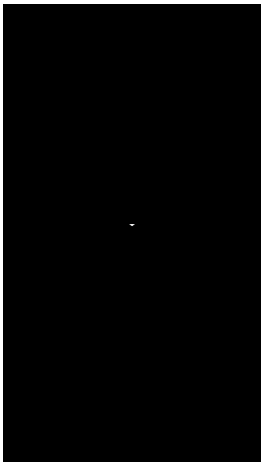


Figure E.73: d = 60

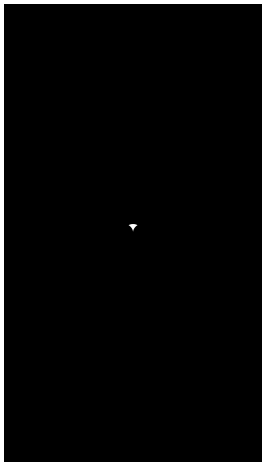


Figure E.74: d = 90

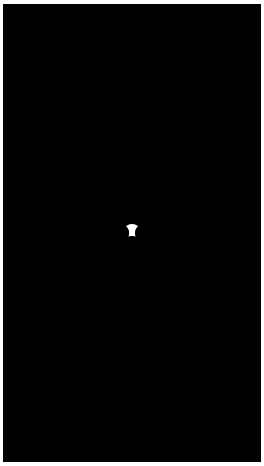


Figure E.75: d = 120

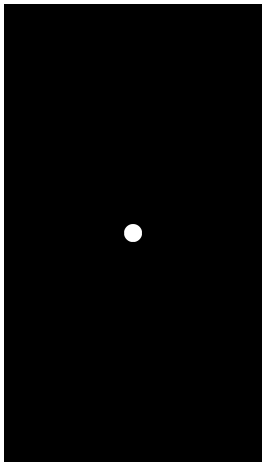


Figure E.76: d = 180

Figure E.77: Visibility differing colour test images set with 3 occluding spheres with centre displacement d pixels

E.1.7. Qualitative Assessment of Test Images

Line

Image	Detection	Radius[%]			Centre Point[%]		
		Pink	Red	Yellow	Pink	Red	Yellow
1	3	0	0	0	0	0	0
2	3	0	0	0	0	0	0
3	3	0	0	0	0	0	0
4	3	0	0	0	0	0	0
5	3	0	0	0	0	0	0
6	3	0	5	0	5	0	0
7	3	0	0	10	0	0	0
8	3	0	0	10	0	0	0
9	3	0	0	0	0	0	0
10	3	0	0	10	0	0	0
11	2	0	-	0	0	-	0
12	3	0	0	0	0	0	0
13	3	0	0	0	0	0	0
14	3	0	0	0	0	0	0
15	3	0	0	0	0	0	0
16	3	0	0	5	0	0	0
17	3	0	0	5	0	0	0
18	3	0	10	0	0	0	0
19	3	5	0	0	0	0	0
20	3	0	0	0	0	0	0
21	3	0	0	0	0	0	0
Average		0.2	0.8	1.9	0.2	0	0
Standard Deviation		1	2	4	1	0	0

Table E.1: Qualitative Assessment of the Test Images, where Three Spheres Could be Detected. The screening was done with increments of 5%.

Ring

Image	Detection	Radius Inaccuracy [%]			Centre Point Inaccuracy [%]		
		Pink	Red	Yellow	Pink	Red	Yellow
1	3	5	5	0	0	0	10
2	3	0	0	0	0	0	5
3	3	0	0	0	0	0	0
4	3	5	0	5	0	0	0
5	3	0	0	0	0	0	0
6	3	0	0	0	0	0	0
7	3	10	0	0	0	0	0
8	3	0	10	5	0	0	0
9	3	0	0	0	0	0	0
10	3	0	0	0	0	0	0
11	3	20	0	0	0	0	0
12	3	0	0	0	0	0	0
13	3	10	0	0	0	0	0
14	3	0	0	0	0	0	0
15	3	0	0	0	0	0	0
16	3	5	0	5	0	0	0
17	3	0	5	0	0	0	0
18	3	5	0	0	0	0	0
19	3	10	0	0	0	0	0
20	3	0	0	15	0	0	0
21	3	5	0	0	0	0	0
Average		3.6	1.0	1.4	0	0	0.7
Standard Deviation		5	2	3	0	0	2

Table E.2: Qualitative Assessment of the Test Images, where Three Spheres Could be Detected. The screening was done with increments of 5%.

None

Image	Detection	Radius Inaccuracy [%]			Centre Point Inaccuracy [%]		
		Pink	Red	Yellow	Pink	Red	Yellow
1	3	10	0	0	0	0	0
2	3	0	0	0	0	0	0
3	3	0	0	5	0	0	0
4	3	0	0	0	0	0	0
5	3	0	2	0	0	0	0
6	3	5	5	5	0	0	0
7	3	10	0	5	0	0	0
8	2	0	-	0	0	-	0
9	3	0	0	5	0	0	0
10	3	0	0	0	0	0	0
11	3	0	0	0	0	0	0
12	3	0	0	5	0	0	0
13	3	10	0	0	5	0	0
14	3	0	0	0	0	0	0
15	3	0	0	0	0	0	0
16	3	0	0	0	0	0	5
17	3	0	0	0	0	0	0
18	3	0	0	0	0	0	10
19	3	10	0	5	0	0	0
20	3	0	0	0	0	0	0
21	3	0	30	0	0	0	0
Average		2.1	1.9	1.1	0.2	0	0.7
Standard Deviation		4	7	2	1	0	2

Table E.3: Qualitative Assessment of the Test Images, where Three Spheres Could be Detected. The screening was done with increments of 5%.

E.1.8. Bias and Error Detection

Radius

Image	Red	Green	Blue	Yellow	Black	Pink
0	68	62	63	62	60	59
1	62	53	57	67	62	56
2	66	59	55	67	56	57
3	51	56	59	62	67	59
4	60	63	57	67	62	60
5	62	53	60	72	62	68
6	66	52	66	67	62	59
7	66	64	59	67	72	62
8	64	56	66	67	56	62
9	66	60	55	72	67	59
mean	63,1	57,8	59,7	67	62,6	60,1
Standard deviation	4,7	4,2	3,9	3,2	4,7	3,2
Averaged Standard deviation						4,0

Table E.4: Radius variability

x-Coordinate

Image	Red	Green	Blue	Yellow	Black	Pink
0	1093	1267	1210	1238	1116	1135
1	1088	1257	1210	1243	1121	1135
2	1088	1271	1215	1243	1121	1135
3	1079	1271	1210	1234	1116	1130
4	1088	1271	1210	1243	1116	1135
5	1083	1257	1206	1234	1116	1140
6	1088	1257	1201	1238	1112	1130
7	1088	1267	1210	1243	1112	1140
8	1083	1271	1201	1238	1121	1135
9	1088	1267	1210	1234	1112	1130
Mean	1086,6	1265,6	1208,3	1238,8	1116,3	1134,5
Standard deviation	3,7	5,9	4,2	3,8	3,5	3,5
Averaged Standard deviation						4,1

Table E.5: x-Coordinate variability

y-Coordinate

Image	Red	Green	Blue	Yellow	Black	Pink
0	1525	1577	1464	1704	1398	1638
1	1530	1582	1469	1699	1403	1638
2	1530	1582	1478	1704	1403	1638
3	1539	1577	1478	1708	1412	1643
4	1525	1586	1469	1708	1408	1647
5	1530	1582	1478	1699	1403	1638
6	1530	1582	1478	1699	1408	1643
7	1530	1586	1478	1704	1417	1643
8	1525	1577	1478	1699	1403	1643
9	1530	1577	1478	1699	1412	1643
Mean	1529,4	1580,8	1474,8	1702,3	1406,7	1641,4
Standard deviation	3,9	3,4	5,1	3,6	5,4	3,0
Averaged Standard deviation						4,1

Table E.6: y-Coordinate variability

E.2. Top-Level

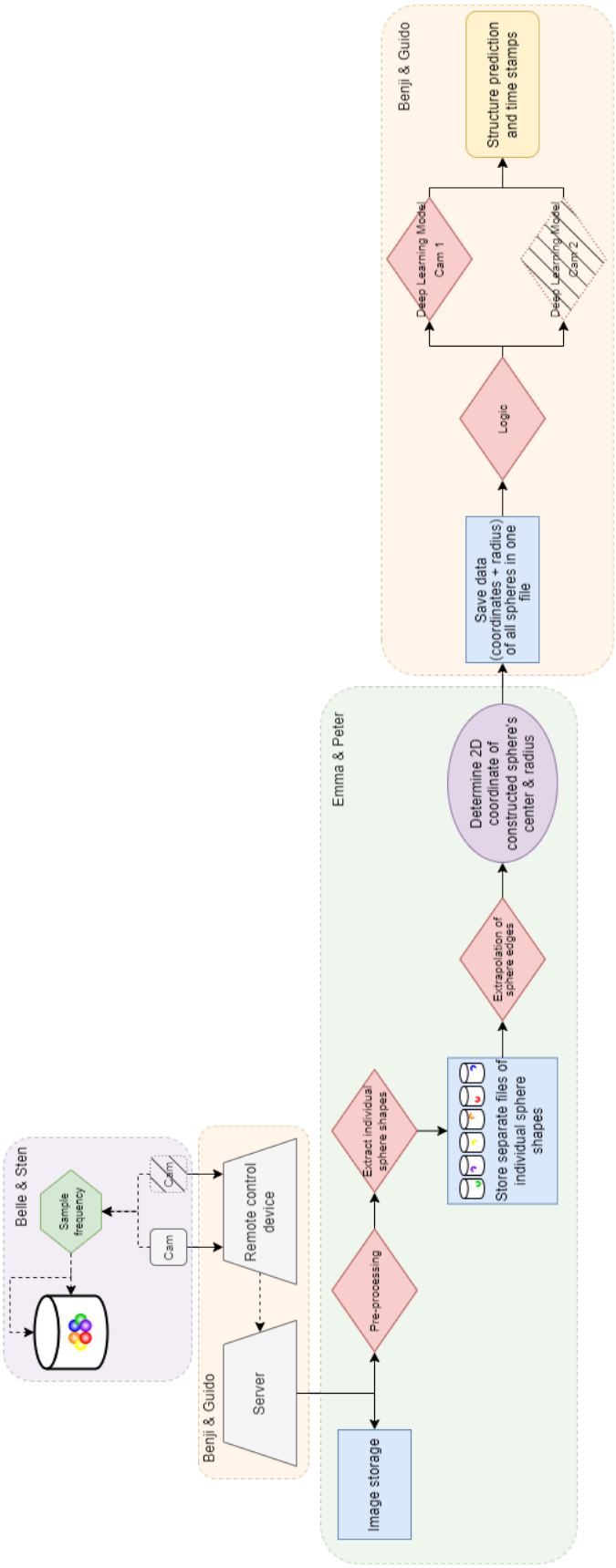


Figure E.78: Pipeline of the Top-Level Implementation