Enhancing Agricultural Efficiency: The Role of Robotics, Spatial Mapping, and Multi-modal Language Models in Crop Management

Alok Raj Didde

August 26, 2024

Contents

Chapter 1

Introduction

1.1 Motivation and Background

Agriculture, one of the world's oldest industries, faces increasing pressure to meet the growing global demand for food while contending with environmental challenges such as climate change, soil degradation, and water scarcity. [7] Traditional farming practices are no longer sufficient to address these challenges. As a result, technological innovations in robotics, artificial intelligence (AI), and data analytics are rapidly transforming agriculture into a more efficient, precise, and sustainable industry.

This report focuses on the application of advanced technologies such as robotics, spatial mapping, and multi-modal language models in crop management. These innovations have the potential to enhance agricultural efficiency by providing detailed environmental mapping, precision in crop health monitoring, and autonomous decision-making in real-time.

1.2 Objectives and Scope

The primary objective of this exercise is to explore the integration of cutting-edge technologies in agriculture to optimize crop management practices. The following key technologies have been investigated:

- **Gaussian Splatting for Real-Time Navigation and Mapping:** This technique transforms sparse 3D point clouds into continuous Gaussian distributions, enabling precise navigation and environmental understanding for agricultural robots. [4]
- **Convolutional Neural Networks (CNNs) for Disease and Pest Detection:** Deep learning models are employed to identify diseases and pests from plant images, facilitating targeted interventions. [22]
- **Visual Language Action Models (VLAMs):** These models integrate visual perception and language understanding to control agricultural robots, enabling them to perform complex tasks such as spraying pesticides or navigating through fields based on natural language instructions.

This aims to demonstrate how these technologies can be combined into a cohesive agricultural robot system that autonomously navigates fields, detects crop health issues, and takes appropriate actions, thereby improving productivity and reducing resource use. [19]

1.3 Overview of the Report

The structure of the report is as follows:

- **Chapter 2: Gaussian Splatting for Navigation and Mapping** - This chapter explains how Gaussian Splatting is applied to enable real-time navigation and mapping in agricultural environments, focusing on both theoretical aspects and practical implementations.
- **Chapter 3: Image-Based Disease and Pest Detection** - This chapter delves into the use of convolutional neural networks for detecting diseases and pests in crops, highlighting the dataset, model architecture, and performance evaluation.
- **Chapter 4: Visual Language Action Models (VLAMs) for Agricultural Robotics** - This chapter introduces the concept of VLAMs and their application in controlling agricultural robots through vision and language inputs.
- **Chapter 5: Agricultural Robot System** - This chapter describes the design, construction, and operation of an agricultural robot that autonomously navigates fields, detects crop issues, and takes corrective actions.
- **Chapter 6: Conclusion and Future Work** - The final chapter summarizes the key findings and discusses the potential future directions for research and development in agricultural robotics.

Chapter 2

Gaussian Splatting for Navigation, Mapping, and Semantic Understanding in Robotics

2.1 Overview

Gaussian Splatting (GSplat) offers an innovative approach to real-time navigation, mapping, and semantic understanding in robotics by transforming sparse 3D point clouds into continuous Gaussian distributions (splats). These splats enable efficient representation of the environment with adjustable precision, facilitating real-time updates and optimizations for robotics tasks. The GSplat framework combines traditional navigation and mapping techniques with advanced semantic understanding, object detection, and knowledge retrieval capabilities, enabling robots to interact more intelligently with their environment. [13]

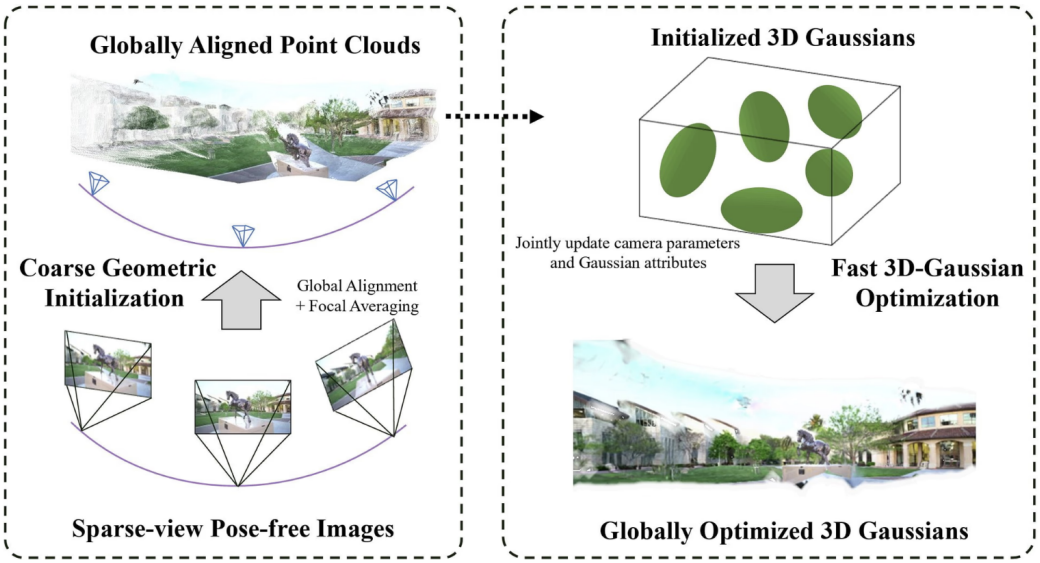

Figure 2.1: Gaussian Splatting Visualization: Transforming sparse 3D point clouds into continuous Gaussian distributions.

The GSplat framework can be divided into several key components:

1. **GSplat-based Navigation and Mapping:** This involves constructing and refining a representation of the environment using Gaussian splats for real-time navigation. The robot utilizes this representation for pose estimation, path planning, and motion control, allowing it to navigate complex environments safely, this method is much faster than NeRF based representation. As the robot moves through its environment, Gaussian splats are continually refined to ensure accurate mapping and efficient navigation.

2. **Image Segmentation and Embeddings:** Beyond navigation, GSplat-based environments enable more advanced perception tasks, such as object detection and semantic understanding. After reconstructing the environment with Gaussian splats, the robot performs image segmentation to identify objects and regions of interest. These segmented regions are converted into high-dimensional image embeddings that capture the essential features of each segment. The image embeddings provide a compact and informative representation of the environment's visual content, facilitating object recognition and other perception tasks.

3. **Vector Database Lookup:** The robot stores these image embeddings in a vector database along with their corresponding locations in the point cloud. This database allows for efficient retrieval of similar embeddings, enabling the robot to recognize previously seen objects and environments. By performing vector database lookups during navigation, the robot can recall the semantic information associated with objects and locations, enabling context-aware decision-making.

4. **Natural Language Query Processing:** GSplat-based environments also support natural language interactions. The robot can process natural language queries, convert them into embeddings using pre-trained language models, and perform a semantic lookup in the vector database. This allows the robot to answer questions, locate objects, and provide information based on both visual and linguistic inputs. For instance, a query like "Where is the nearest charging station?" will trigger the retrieval of the relevant information from the vector database, enabling the robot to provide accurate responses and assist in complex tasks.

By integrating these components, the GSplat framework combines efficient real-time navigation with advanced semantic understanding and natural language processing. This enables robots to perform tasks that go beyond basic navigation, allowing for meaningful interaction with the environment and providing valuable contextual information during exploration and task execution. [11]

2.2 GSplat-based Navigation and Mapping

2.2.1 GSplat-based Navigation

1. **GSplat Reconstruction**
   Gaussian Splatting initializes the navigation pipeline by transforming a sparse point cloud, obtained via camera calibration or other methods, into 3D Gaussian splats. This reconstruction allows for the environment's geometric representation to be optimized, forming a detailed

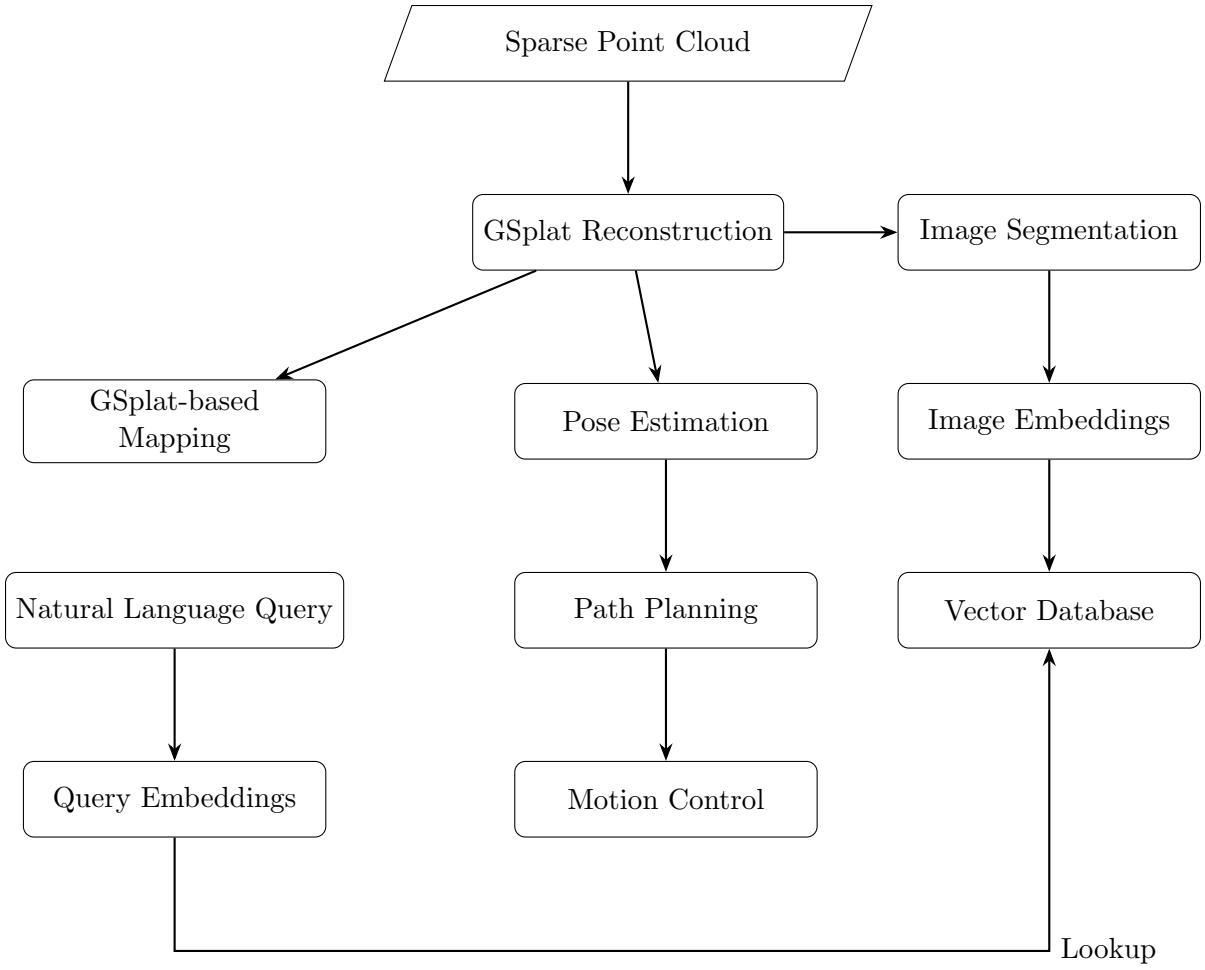

Figure 2.2: GSplat-based Navigation, Mapping, and Semantic Understanding Workflow

and efficient basis for real-time navigation tasks. The mathematical representation of a Gaussian splat is given by its mean $\mu \in \mathbb{R}^3$, covariance matrix $\Sigma \in S_{++}$, opacity $\alpha \in [0, 1]$, and spherical harmonics coefficients defining view-dependent colors [12]. The 3D covariance Σ is further decomposed as $\Sigma = RSS^TR^T$, where $R \in SO(3)$ is a rotation matrix and S is a diagonal scaling matrix [4].

To implement the GSplat framework, we utilize a NeRF-based [24] approach provided by the ‘nerfstudio‘ framework [21]. The following code snippet demonstrates how to construct a Gaussian splat model using images from a ‘data‘ folder.

```
    # nerfstudio installation and setup
    pip install nerfstudio

    # Prepare your dataset by placing images and camera parameters in the ‘data‘ folder
    # The following script initializes the splat model

    import nerfstudio as ns

```

```
    # Path to the dataset
    data_path = "./data"

    # Load configuration for the Gaussian Splatting model
    config = ns.configs.get_default_gaussian_splat_config(
        dataset_path=data_path,
        model_name="g_splat_model",
    )

    # Instantiate the NeRF Studio pipeline with Gaussian splatting
    pipeline = ns.pipeline.create_pipeline(config)

    # Start training the model
    pipeline.train()

    # Export the trained splat model for further use in navigation and
  mapping
    pipeline.save_model("splat_model.pth")
```

This code constructs a Gaussian splatting model from the provided images and camera parameters in the 'data' folder. The 'nerfstudio' framework is leveraged to optimize the Gaussian splats for an accurate and efficient environment representation. The configuration is automatically managed using the `get_default_gaussian_splat_config` function, which eliminates the need for manual specification of parameters like `num_gaussians`.

2. **Pose Estimation**
   Splat-Loc provides real-time pose estimation by leveraging the point cloud generated from Gaussian splatting. Pose estimation is done using a combination of global initialization and recursive localization based on monocular RGB images. Mathematically, the pose estimation problem can be formulated as a maximum a-posteriori (MAP) optimization problem. Given prior estimates of the robot's pose $\mathbf{x}_{t-1}$ and the measurements $\mathbf{y}_t$, the MAP estimate of the current pose $\mathbf{x}_t$ is computed by minimizing:

$$\mathbf{x}_t^\star = \arg\min_{\mathbf{x}_t} \left(\|\mathbf{x}_t - f(\mathbf{x}_{t-1}, \mathbf{u}_{t-1})\|^2_{Q^{-1}} + \|\mathbf{y}_t - h(\mathbf{x}_t)\|^2_{R^{-1}} \right)$$

   subject to constraints that ensure the pose remains within safe polytope corridors defined by the Gaussian splats [2] [4].

   We are solving the following optimization problem for pose estimation:

$$x_t^* = \arg\min_{x_t} \|x_t - f(x_{t-1}, u_{t-1})\|^2_{Q^{-1}} + \|y_t - h(x_t)\|^2_{R^{-1}}$$

   The Python implementation for this MAP optimization can be written as follows:

```
import numpy as np
from scipy.optimize import minimize

# Define the state transition function f(x_{t-1}, u_{t-1})
def state_transition(x_prev, u_prev):
    # Example linear transition model
    return np.dot(A, x_prev) + np.dot(B, u_prev)
```

```python

# Define the measurement function h(x_t)
def measurement_model(x_t):
    # Example measurement model
    return np.dot(C, x_t)

# Define the cost function to minimize
def cost_function(x_t, x_prev, u_prev, y_t, Q_inv, R_inv):
    # State transition error
    state_error = x_t - state_transition(x_prev, u_prev)
    state_cost = np.dot(state_error.T, np.dot(Q_inv, state_error))

    # Measurement error
    measurement_error = y_t - measurement_model(x_t)
    measurement_cost = np.dot(measurement_error.T, np.dot(R_inv,
   measurement_error))

    # Total cost
    return state_cost + measurement_cost

# Example matrices (replace with actual values)
A = np.eye(3)  # State transition matrix
B = np.eye(3)  # Control input matrix
C = np.eye(3)  # Measurement matrix
Q_inv = np.eye(3)  # Inverse of process covariance matrix
R_inv = np.eye(3)  # Inverse of measurement covariance matrix

# Previous state, control input, and measurement
x_prev = np.array([1.0, 2.0, 3.0])
u_prev = np.array([0.1, 0.2, 0.3])
y_t = np.array([1.5, 2.5, 3.5])

# Initial guess for the current state
x_t_initial = np.array([1.2, 2.2, 3.2])

# Perform the optimization
result = minimize(cost_function, x_t_initial, args=(x_prev, u_prev, y_t,
   Q_inv, R_inv))

# Optimal pose estimate
x_t_optimal = result.x
print("Optimal Pose Estimate:", x_t_optimal)
```

This code implements the MAP optimization problem for pose estimation. It uses the `scipy.optimize.minimize` function to find the optimal pose x_t by minimizing the cost function, which is a combination of the state transition and measurement model errors.

3. **Path Planning**
   The Splat-Plan module ensures safe navigation through the environment by constructing polytopic safe corridors. These polytopes are created by decomposing the configuration space into occupied and collision-free regions using the intersection of ellipsoidal approximations of the Gaussian splats. The collision-checking algorithm is based on solving generalized eigenvalue

problems and can be parallelized for efficiency. Given a set of safe polytopes, the trajectory planning problem is formulated as a quadratic program that minimizes the path length subject to safety constraints:

$$\min_{\mathbf{c}_i} \sum_{i=0}^{M-1} \|\mathbf{c}_{i+1} - \mathbf{c}_i\|_2^2 \quad \text{subject to } \mathbf{A}\mathbf{c}_i \leq \mathbf{b}$$

where $\mathbf{c}_i$ are the control points of the Bézier curve representing the trajectory [4].

The Splat-Plan module ensures safe navigation by constructing polytopic safe corridors. These polytopes are created by decomposing the configuration space into occupied and collision-free regions using the intersection of ellipsoidal approximations of Gaussian splats.

The trajectory planning problem can be formulated as a quadratic program (QP) that minimizes the path length subject to safety constraints:

$$\min_{\{c_i\}} \sum_{i=0}^{M-1} \|c_{i+1} - c_i\|_2^2 \quad \text{subject to} \quad Ac_i \leq b$$

Here, c_i are the control points of the Bézier curve representing the trajectory. The following Python code implements this optimization problem.

```python
import numpy as np
from scipy.optimize import minimize

# Example quadratic cost function
def trajectory_cost(control_points):
    total_cost = 0
    for i in range(len(control_points) - 1):
        total_cost += np.linalg.norm(control_points[i+1] - control_points[i])**2
    return total_cost

# Example constraint function for the polytope constraints
def polytope_constraints(control_points, A, b):
    constraints = []
    for c_i in control_points:
        constraints.append({'type': 'ineq', 'fun': lambda c_i, A=A, b=b: b - np.dot(A, c_i)})
    return constraints

# Example setup (replace with actual matrices and values)
M = 5  # Number of control points
A = np.array([[1, 0], [0, 1]])  # Example polytope constraint matrix
b = np.array([1, 1])  # Example constraint bounds

# Initial guess for control points (random initialization)
initial_control_points = np.random.rand(M, 2)

# Set up the constraints for all control points
constraints = polytope_constraints(initial_control_points, A, b)

```

```
# Perform the optimization (minimizing path length subject to polytope
    constraints)
result = minimize(trajectory_cost, initial_control_points, constraints=
    constraints)

# Optimal control points
optimal_control_points = result.x.reshape(M, 2)
print("Optimal Control Points:", optimal_control_points)
```

This code formulates and solves the quadratic program for trajectory planning. It minimizes the path length defined by the control points c_i of a Bézier curve, while ensuring that the trajectory remains within the safe polytope corridors by enforcing the constraint $Ac_i \leq b$.

4. **Motion Control**
   The planned trajectory is executed through motion control algorithms that ensure smooth and precise movement of the robot. [8] The trajectory is generated using Bézier curves within the safe polytopes, which guarantees safety and efficiency. The motion control system is integrated with the Gaussian splatting environment, allowing for dynamic adjustments in real-time navigation.

2.2.2 GSplat-based Mapping

1. **GSplat Reconstruction**
   For mapping, Gaussian Splatting is used to optimize the representation of 3D Gaussians, which enables efficient storage and real-time updates to the map. The anisotropic covariance matrices are adjusted to balance between map detail and computational efficiency [12].

2. **Exploration and Mapping**
   Robots using Gaussian Splatting can perform continuous exploration and mapping, adjusting their internal representation of the environment as they move. The exploration is driven by the robot's need to maintain an updated understanding of the surroundings, avoiding obstacles while generating a detailed map.

3. **Map Refinement**
   Gaussian splats are refined by adjusting their anisotropic covariances as the robot gathers more data from the environment. This allows for the continuous refinement of the environment representation, ensuring that the robot's map remains accurate and precise over time [12].

2.3 Semantic Understanding and Knowledge Retrieval in GSplat Environments

The continuous environment representation created by Gaussian splatting is leveraged for more advanced tasks, including semantic understanding, image-based query processing, and natural language queries. These processes enable robots to interact more meaningfully with their surroundings by providing both perceptual and cognitive capabilities. [3]

2.3.1 Image Segmentation and Embeddings

After the environment has been reconstructed using Gaussian splatting, the robot performs image segmentation to identify objects and regions of interest within its field of view. Image segmentation divides the visual input into segments corresponding to distinct objects or surfaces, helping the robot to parse the environment into meaningful components. [18]

Once the image segmentation process is complete, the segmented regions are passed through a neural network to generate high-dimensional image embeddings. These embeddings represent the visual content in a compact form, capturing the essential features of each segment. The embeddings are crucial for both object recognition and later retrieval tasks. [3]

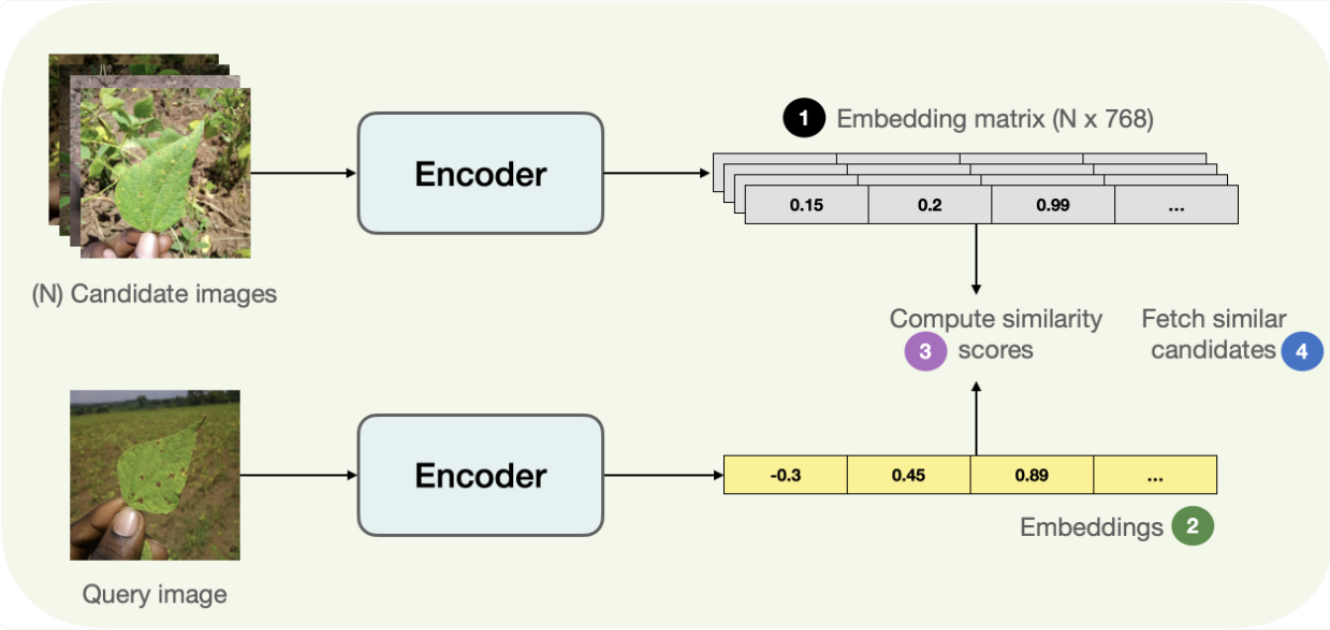

Figure 2.3: Image Embeddings for Semantic Understanding

1. Convert Image to Embeddings

We use the CLIP model from the ‘transformers‘ library to convert images into embeddings.

```
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import torch

# Load a pre-trained CLIP model and processor
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Function to convert an image to embeddings
def image_to_embeddings(image_path):
    image = Image.open(image_path)
    inputs = processor(images=image, return_tensors="pt")
    with torch.no_grad():
```

```
        embeddings = model.get_image_features(**inputs)
    return embeddings.squeeze().numpy()

# Example usage
image_path = "example_image.jpg"
image_embeddings = image_to_embeddings(image_path)
print("Image Embeddings:", image_embeddings)
```

2. Store Embeddings in ChromaDB

After converting the image to embeddings, we store the embeddings in ChromaDB which uses FAISS to lookup similar embeddings efficiently. [6], a vector database for efficient storage and retrieval of embeddings.

```
import chromadb
from chromadb.utils import embedding_store

# Initialize ChromaDB client
client = chromadb.Client()

# Create a collection to store embeddings
collection = client.create_collection("multimodal_embeddings")

# Function to store embeddings in ChromaDB
def store_embeddings(id, embeddings, metadata=None):
    collection.add(id=id, embedding=embeddings.tolist(), metadata=metadata)

# Store the image embeddings with an identifier
store_embeddings(id="image_1", embeddings=image_embeddings, metadata={"type": "
    image"})
```

3. Convert Text Query to Embeddings

We convert a text query into embeddings using the same CLIP model, allowing for multimodal comparisons. [3]

```
# Function to convert text query to embeddings
def text_to_embeddings(text_query):
    inputs = processor(text=text_query, return_tensors="pt")
    with torch.no_grad():
        embeddings = model.get_text_features(**inputs)
    return embeddings.squeeze().numpy()

# Example usage
text_query = "A cat sitting on a chair"
text_embeddings = text_to_embeddings(text_query)
print("Text Query Embeddings:", text_embeddings)
```

2.3.2 Vector Database Lookup

The image embeddings generated from the segmented regions are stored in a vector database along with their associated locations in the point cloud. The vector database enables fast and efficient retrieval of similar embeddings, facilitating object recognition, tracking, and semantic labeling.

During navigation, if the robot encounters a previously observed region or object, it can perform a vector database lookup by comparing the current image embeddings with the stored embeddings. This comparison allows the robot to recognize familiar objects and recall their previously associated semantic labels, enabling context-aware decision-making and interaction with the environment. [6]

```
    # Function to lookup similar embeddings in ChromaDB
    def query_similar_embeddings(embeddings, top_k=5):
        results = collection.query(embedding=embeddings.tolist(), n_results=
top_k)
        return results

    # Query similar embeddings using the text query embeddings
    similar_embeddings = query_similar_embeddings(text_embeddings)
    print("Similar Embeddings:", similar_embeddings)

```

2.3.3 Natural Language Query Processing

In addition to image-based processing, the robot is capable of understanding and processing natural language queries. The process begins with the robot receiving a natural language query, such as "Where is the nearest charging station?" or "Identify the red object in the room."

The query is first converted into a high-dimensional embedding using a pre-trained language model. This embedding captures the meaning of the query and is used to perform a semantic lookup in the vector database. The robot compares the query embedding with the stored image embeddings and retrieves the relevant objects or locations that match the query.

For example, if the query is about finding a specific object, the robot will locate the corresponding image embeddings in the vector database and return the object's location in the point cloud. This allows the robot to answer questions about its environment and provide meaningful responses based on its prior knowledge and observations. [3]

2.3.4 Integration with Navigation and Mapping

The semantic understanding and query processing capabilities are integrated with the GSplat-based navigation and mapping pipeline. As the robot explores its environment, it continuously updates its internal representation with new Gaussian splats, image embeddings, and semantic labels. This enables the robot to build a rich and detailed map of the environment that can be queried both visually and linguistically.

Furthermore, the vector database provides a way to efficiently store and retrieve this information, allowing the robot to recognize objects and locations that it has previously encountered. This integration ensures that the robot can perform advanced navigation tasks while maintaining an updated understanding of its surroundings.

By combining Gaussian splatting with image segmentation, embeddings, and natural language processing, the robot can achieve a high level of semantic understanding, enabling it to interact with complex environments in a meaningful and intelligent way.

Chapter 3

Image-Based Disease and Pest Detection

The detection of plant diseases and pests is a critical task in agriculture, as early identification can prevent widespread crop damage and reduce yield losses. Traditional methods of disease detection rely on visual inspection by human experts, which can be time-consuming and subjective. In recent years, the application of deep learning techniques, particularly convolutional neural networks (CNNs), has shown promising results in automating the detection of plant diseases from images. This chapter explores the use of CNNs for image-based disease and pest detection in crops, focusing on the dataset, model architecture, training process, and performance evaluation. [14]

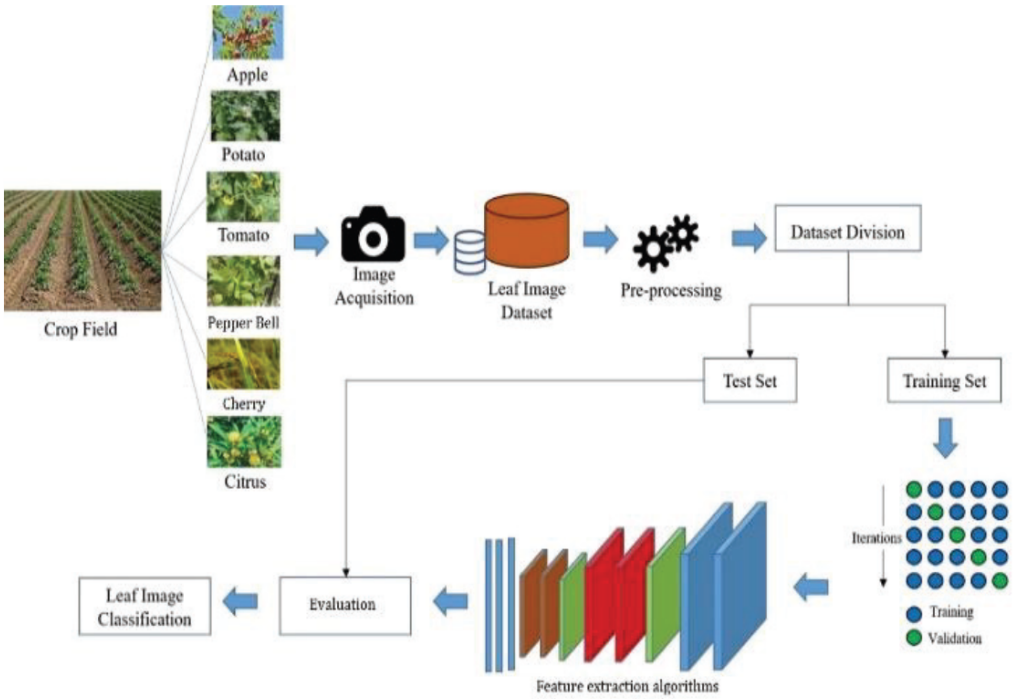

Figure 3.1: Convolutional Neural Network (CNN) Architecture for Disease Detection.

3.1 Dataset

The model is trained on the PlantVillage dataset, which contains 54,306 images of plant leaves categorized into 38 classes, representing both healthy and diseased leaves across various crops. The images are resized to 256x256 pixels for input into the CNN model. [10]

3.1.1 Preprocessing

Before feeding the images into the model, they are preprocessed to enhance the training efficiency. Preprocessing steps include resizing, normalization, and data augmentation techniques such as rotation, zoom, and flipping to improve the model's robustness. [**?**]

3.2 Model Architecture

The proposed method employs a CNN architecture, which consists of multiple convolutional layers, max-pooling layers, and fully connected layers. The architecture extracts features from the input images and classifies them into healthy or diseased categories.

3.2.1 Convolutional Layers

Convolutional layers are the foundation of CNNs. Each convolutional layer applies filters to the input images to detect specific features, such as edges, textures, and patterns. [16]

3.2.2 Pooling Layers

Pooling layers perform down-sampling operations to reduce the spatial dimensions of the feature maps. Max pooling is the most commonly used technique, where the maximum value within a sliding window is selected.

3.2.3 Fully Connected Layers

After feature extraction, fully connected layers map the features to output classes representing different plant diseases.

3.3 Training and Evaluation

The CNN model is trained using the PlantVillage dataset with 80% of the data used for training and 20% for testing. The model is optimized using the Adam optimizer, and the categorical cross-entropy loss function is employed. The training process is carried out for ten epochs.

3.3.1 Performance Metrics

The model's performance is evaluated using accuracy and loss metrics. The trained model achieves an accuracy of 95% on the training data and 94% on the test data, demonstrating its effectiveness in classifying plant diseases.

3.4 Fine-tuning the Model

Fine-tuning is an essential step in improving the model's accuracy by leveraging pre-trained models. Below is an example of fine-tuning a pre-trained CNN model (e.g., VGG16) on the PlantVillage dataset. [17]

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Load the pre-trained VGG16 model
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(256, 256, 3))

# Freeze the layers of the pre-trained model
for layer in base_model.layers:
    layer.trainable = False

# Add custom layers on top of the base model
x = Flatten()(base_model.output)
x = Dense(128, activation='relu')(x)
x = Dense(64, activation='relu')(x)
predictions = Dense(38, activation='softmax')(x)

# Define the final model
model = Model(inputs=base_model.input, outputs=predictions)

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])

# Data augmentation
train_datagen = ImageDataGenerator(rescale=1./255, rotation_range=20, width_shift_range=0.2, height_shift_range=0.2, horizontal_flip=True)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load training and testing data
train_generator = train_datagen.flow_from_directory('data/train', target_size=(256, 256), batch_size=32, class_mode='categorical')
test_generator = test_datagen.flow_from_directory('data/test', target_size=(256, 256), batch_size=32, class_mode='categorical')

# Train the model
history = model.fit(train_generator, epochs=10, validation_data=test_generator)
```

```
36
37 # Evaluate the model
38 loss, accuracy = model.evaluate(test_generator)
39 print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

Listing 3.1: Fine-tuning a Pre-trained CNN

Chapter 4

Visual Language Action Models (VLAMs) for Agricultural Robotics

4.1 Integration with Robotic Control

OpenVLA integrates pre-trained vision-language models (VLMs) [13] for controlling agricultural robots. By building on a Llama 2 [23] backbone combined with visual encoders from DINOv2 and SigLIP, OpenVLA allows the mapping of visual input $\mathbf{I}$ and language instructions $\mathbf{L}$ to robot control actions $\mathbf{A}$.

Let $\mathbf{I}$ be the input image from the robot's camera and $\mathbf{L}$ be the language instruction provided by the user. The model predicts the robot's action $\mathbf{A}$ by performing the following operation:

$$\mathbf{A} = \text{VLA}(\mathbf{I}, \mathbf{L}),$$

where VLA represents the vision-language-action model, which outputs the robot's control actions, such as gripper position, orientation, and grip force.

The architecture of OpenVLA is shown in Figure 4.1, where visual features from DINOv2 and SigLIP are concatenated and projected into the input space of the Llama 2 language model. The language model then generates robot actions in the form of discrete tokens.

For practical deployment, the OpenVLA model can be served over a REST API, enabling seamless integration with robotic control systems in real-time. The code snippet below demonstrates deploying OpenVLA models via a lightweight server using the HF AutoClass API:

```
import os.path
import json_numpy
json_numpy.patch()
import json
import logging
import traceback
from pathlib import Path
from typing import Any, Dict, Optional, Union

import torch
import uvicorn
from fastapi import FastAPI
from PIL import Image
```

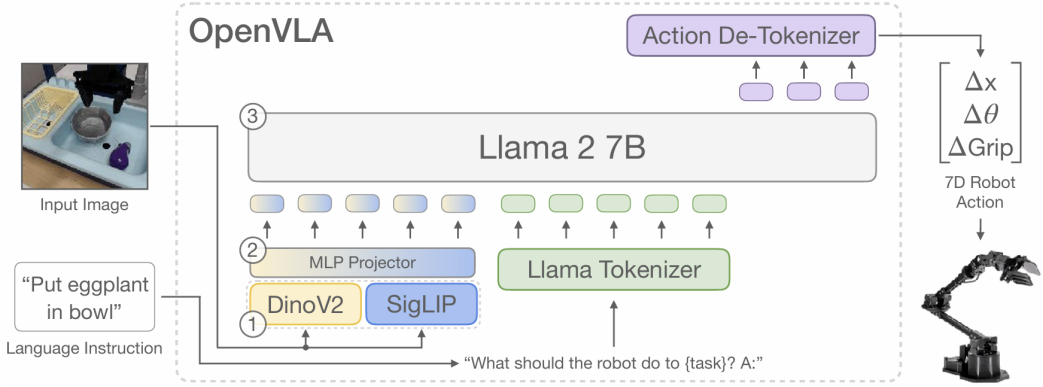

Figure 4.1: OpenVLA model architecture. The visual encoder extracts features from the input image, which are concatenated and projected into the input space of the Llama 2 language model. The model then generates robot control actions.

```python
from transformers import AutoModelForVision2Seq, AutoProcessor

class OpenVLAServer:
    def __init__(self, openvla_path: Union[str, Path]):
        self.device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
        self.processor = AutoProcessor.from_pretrained(openvla_path, trust_remote_code=True)
        self.vla = AutoModelForVision2Seq.from_pretrained(
            openvla_path, torch_dtype=torch.bfloat16, low_cpu_mem_usage=True, trust_remote_code=True
        ).to(self.device)

    def predict_action(self, payload: Dict[str, Any]) -> str:
        image, instruction = payload["image"], payload["instruction"]
        inputs = self.processor(instruction, Image.fromarray(image).convert("RGB")).to(self.device, dtype=torch.bfloat16)
        action = self.vla.generate(**inputs)
        return action

    def run(self, host: str = "0.0.0.0", port: int = 8000):
        app = FastAPI()
        app.post("/act")(self.predict_action)
        uvicorn.run(app, host=host, port=port)

if __name__ == "__main__":
    server = OpenVLAServer("path_to_model")
    server.run()
```

Listing 4.1: OpenVLA Deployment Server Example

4.2 Co-Fine-Tuning

OpenVLA supports efficient fine-tuning on new agricultural tasks using small datasets. The model leverages modern parameter-efficient techniques such as Low-Rank Adaptation (LoRA), which allows fine-tuning only a small percentage of the model parameters while maintaining performance [**?**].

The fine-tuning objective can be written as minimizing the cross-entropy loss $\mathcal{L}_{CE}$ between the predicted token sequence $\hat{\mathbf{T}} = \{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_n\}$ and the ground truth token sequence $\mathbf{T} = \{t_1, t_2, \ldots, t_n\}$, representing the robot actions:

$$\mathcal{L}_{CE}(\hat{\mathbf{T}}, \mathbf{T}) = -\sum_{i=1}^{n} t_i \log(\hat{t}_i).$$

During fine-tuning, the model updates its parameters θ by optimizing the following loss function:

$$\theta^* = \arg\min_{\theta} \mathbb{E}_{(\mathbf{I},\mathbf{L},\mathbf{T})\sim\mathcal{D}} \left[\mathcal{L}_{CE}(\hat{\mathbf{T}}, \mathbf{T}) \right],$$

where $\mathcal{D}$ represents the dataset of image-language-action triplets.

4.3 Action as Text Tokens

The OpenVLA model maps continuous robot actions into discrete tokens using the Llama 2 tokenizer [23]. The robot actions, $\mathbf{A}$, are first discretized into K bins per dimension, resulting in a sequence of discrete tokens $\mathbf{T}$:

$$\mathbf{T} = \text{Discretize}(\mathbf{A}) = \{t_1, t_2, \ldots, t_n\},$$

where $t_i \in \{0, 1, \ldots, K-1\}$.

The token sequence is then processed by the language model to predict the next action in a sequence-to-sequence manner [20]. The final action $\hat{\mathbf{A}}$ is reconstructed by decoding the predicted tokens:

$$\hat{\mathbf{A}} = \text{Decode}(\hat{\mathbf{T}}).$$

This approach allows the model to handle agricultural tasks such as precise gripper positioning and adjusting the robot arm's trajectory based on the environment.

4.4 Real-Time Inference

Real-time inference is crucial for agricultural robotics, where decisions need to be made instantly based on sensor inputs. OpenVLA's remote inference solution, released as part of the codebase, supports streaming of action predictions to agricultural robots in real-time.

The latency of the inference process can be represented as:

$$\text{Latency} = T_{\text{preprocess}} + T_{\text{forward}} + T_{\text{postprocess}},$$

where $T_{\text{preprocess}}$ is the time taken to preprocess the input image and text, T_{forward} is the forward pass through the model, and $T_{\text{postprocess}}$ is the time taken to convert the predicted tokens back into actions.

4.5 Generalization and Emergent Capabilities

OpenVLA demonstrates strong generalization capabilities across diverse agricultural tasks, such as handling unseen crops, varying lighting conditions, and multiple objects in complex environments. The model's ability to generalize is a result of both the large-scale Internet-pretrained data and the fine-tuning on agricultural-specific datasets.

The model's generalization can be evaluated by its performance on unseen tasks, which is quantified by the success rate S:

$$S = \frac{\text{Number of successful task completions}}{\text{Total number of task attempts}}.$$

OpenVLA achieves high success rates across a wide range of agricultural tasks, demonstrating its robustness and adaptability.

The emergence of such capabilities in VLAMs is attributed to the combination of large-scale Internet-pretrained data and task-specific fine-tuning, which allows for learning transferable skills across different agricultural domains.

Chapter 5

Agricultural Robot System

This chapter describes the design and functioning of an agricultural robot that autonomously navigates, detects diseases and pests, and sprays chemicals using advanced technologies such as Gaussian splatting and convolutional neural networks (CNNs). The robot is constructed with aluminum extrusion as its frame, motors for movement and arm control, and various sensors and components for navigation and operation. An Android application is used to control both the Arduino Uno and the ESP32 microcontrollers for precise movement and spraying actions.

5.1 System Overview

The robot uses a mobile phone, specifically a Google Pixel, as its primary processing unit. [15] The phone's camera captures images to create a 3D Gaussian splat map for navigation, allowing the robot to identify its position in the field and locate target objects for spraying. Additionally, the robot is capable of identifying diseases and pests using a CNN-based model trained on agricultural datasets. The system operates as follows:

1. The robot wakes up and captures an image using the phone's camera.
2. The image is processed to create a 3D Gaussian splat for localization and mapping. [4] [2]
3. Simultaneously, the image is analyzed using a CNN-based model to detect diseases or pests on plants. [22]
4. Based on the identified location and any detected issues, the robot receives instructions to move to a specific point. [8]
5. Using OpenVLA, the robot performs actions such as moving its arms and spraying chemicals on the target objects. [13]

The system connects to the web server to send images, receive instructions for navigation, and apply actions based on disease and pest identification. An Android application is also used to control the robot's movement and spraying mechanism, providing a user-friendly interface for manual control when needed.

5.2 Communication and Control

The Android phone uses USB-C to serial adapter to communicate with the Arduino Uno and ESP32 microcontrollers. The Arduino Uno controls the motors for arm movement and the hydraulic motor for spraying, while the ESP32 controls the base wheel motors for navigation. The phone captures images and sends them to the cloud server for processing, receiving navigation instructions and action tokens in return. The robot's operation is coordinated through a combination of image processing, navigation algorithms, and disease/pest detection logic.

5.3 Gaussian Splatting for Navigation and Mapping

Gaussian splatting offers an innovative approach to real-time navigation, mapping, and semantic understanding by transforming sparse 3D point clouds into continuous Gaussian distributions (splats). This technique provides an efficient representation of the environment, enabling the robot to perform real-time updates and optimizations for navigation tasks.

The key components of Gaussian splatting include:

- **GSplat Reconstruction:** The robot converts images captured by the camera into 3D Gaussian splats, forming a geometric representation of the environment. This allows the robot to optimize navigation in real-time. [4]
- **Pose Estimation:** Using the Gaussian splat map, the robot performs pose estimation to accurately determine its position in the field. [2]
- **Path Planning:** The robot plans its path by constructing polytopic safe corridors, ensuring collision-free navigation. [8]
- **Motion Control:** The planned trajectory is executed through smooth motion control, integrated with the Gaussian splat environment for dynamic adjustments during navigation. [8]

This approach enables the robot to navigate autonomously in complex agricultural environments, continuously refining its internal map as it moves through the field.

5.4 Electronics and Control System

The robot's electronics are controlled by an Arduino Uno, which drives the motors in the arms. The power system consists of an S4 9V battery, which is converted to 12V using a voltage converter. The motor control is achieved using TMC2208 drivers, providing smooth and precise control of the arm motors.

The base wheel motors are powered by a separate battery, identical to the one used for the arms. The ESP32 microcontroller, along with a relay, controls the motors and ensures proper movement of the robot. The hydraulic motor for the spraying mechanism is also controlled by the Arduino Uno.

5.5 Hydraulic System

The hydraulic system of the robot consists of a pump, a reservoir, and a spraying mechanism. The pump is powered by a separate power supply, providing the necessary pressure to drive the liquid spraying system. The reservoir holds the chemicals to be sprayed, which are released through the spraying mechanism when activated. The hydraulic system is controlled by the Arduino Uno, ensuring precise and efficient spraying of chemicals on target objects.

[1]

5.5.1 Power System

The power system of the robot consists of:

- An S4 9V battery for the arm motors and control system.
- A voltage converter to step up the voltage to 12V for motor operation.
- TMC2208 motor drivers to control the NEMA 17 motors.
- A separate S4 9V battery for the base wheel motors.
- Power supply for the hydraulic motor to drive the liquid spraying system.
- Hydraulic pump and reservoir for chemical spraying using a Relay.

5.6 Software and Communication

The robot's software is based on a combination of image processing, navigation algorithms, disease/pest detection, and motor control logic. The mobile phone serves as the main computational brain, performing the following tasks:

- Capturing images for localization and disease/pest detection.
- Sending images to the cloud server to create a 3D Gaussian splat map.
- Analyzing images using the CNN model to detect diseases and pests.
- Receiving instructions from the server based on the identified location and detected issues.
- Using OpenVLA to generate action tokens for arm movement and spraying.

The ESP32 microcontroller is responsible for controlling the base motors, using feedback from an accelerometer to maintain stability and ensure accurate navigation. The Android application provides manual control over the robot, enabling the user to issue movement commands or activate the spraying mechanism when needed.

5.7 Operation Workflow

The robot operates autonomously in the field with the following workflow:

1. The robot powers on and initializes all systems.
2. The mobile phone captures an image of the surroundings.
3. The image is sent to the cloud server for localization using Gaussian splatting.
4. Simultaneously, the image is analyzed for disease and pest detection using a CNN model.
5. The robot receives navigation instructions and moves to the specified location.
6. The arms are positioned based on action tokens generated by OpenVLA.
7. The spraying mechanism is activated using the hydraulic motor, and chemicals are applied to the target object if diseases or pests are detected.
8. The user can also manually control the robot's movement and spraying mechanism using the Android app.

5.8 IK and FK for Arm Movement

The robot's arm movement is controlled using Inverse Kinematics (IK) and Forward Kinematics (FK) algorithms. The IK algorithm calculates the joint angles required to position the end effector at a specific location, while the FK algorithm determines the end effector's position based on the joint angles. These algorithms enable precise and efficient control of the robot's arms, allowing it to perform complex tasks such as spraying chemicals on target objects. Using a grid based system with the image segmentation, the robot can identify the location of the target object and calculate the IK and FK to move the arm to the desired position.

5.9 Workflow Diagram

5.10 Tools Used

The following tools were used in the development of the agricultural robot system:

- **Arduino Uno:** Controls the arm motors and hydraulic motor.
- **ESP32:** Controls the base wheel motors for navigation.
- **Google Pixel:** Acts as the main processing unit for image capture and analysis.
- **Android App:** Provides manual control over the robot's movement and spraying mechanism.
- **OpenVLA:** Generates action tokens for arm movement and spraying based on visual and language inputs.

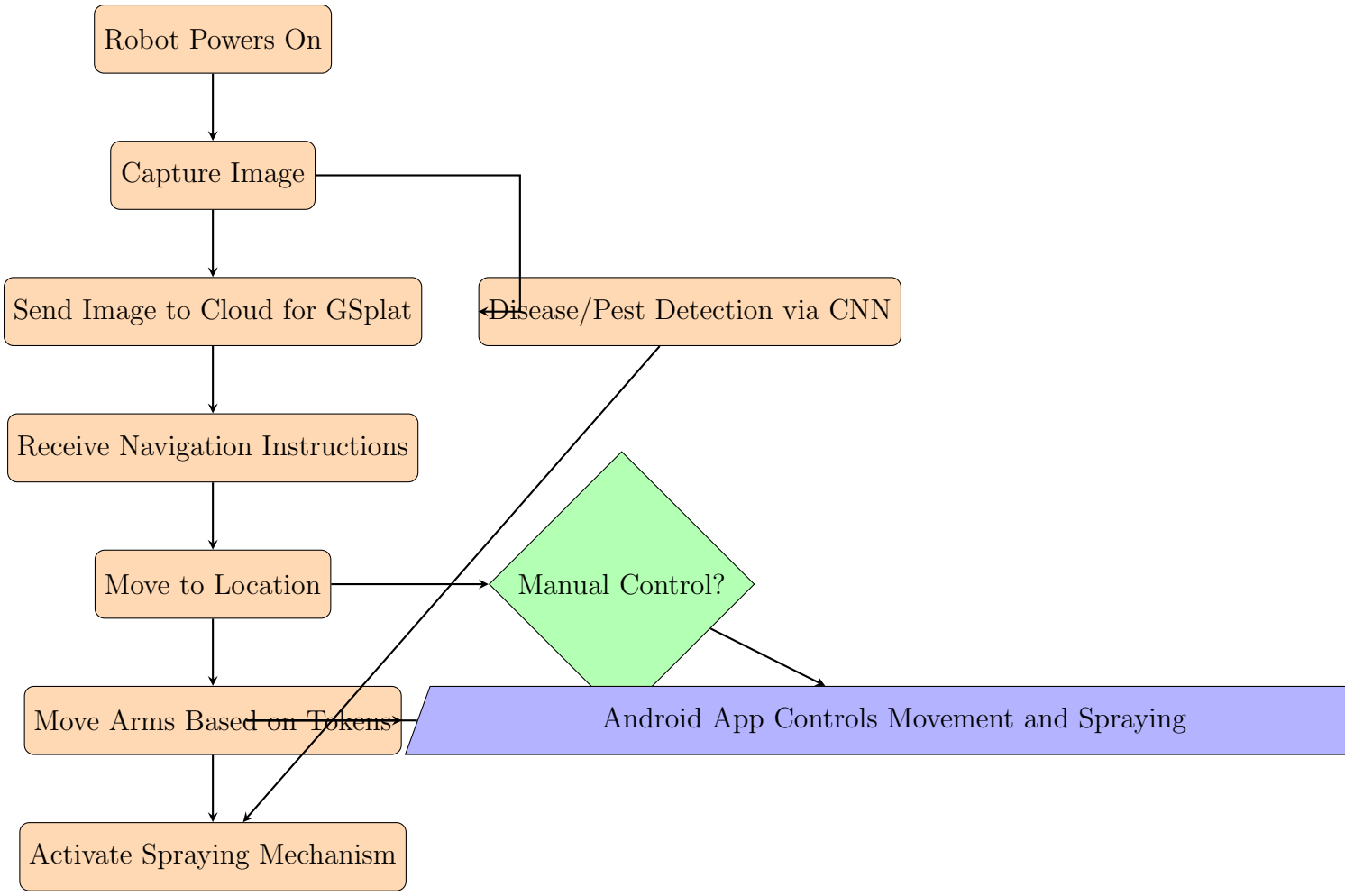

Figure 5.1: Workflow of the Agricultural Robot System

- **Gaussian Splatting:** Creates a 3D Gaussian splat map for navigation and mapping.
- **CNN Model:** Detects diseases and pests in plants from captured images.
- **Hydraulic System:** Sprays chemicals on target objects based on detected issues.
- **IK and FK Algorithms:** Control the arm movement for precise positioning.
- **Grid Based System:** Identifies the location of target objects for arm movement.
- **Image Segmentation:** Analyzes images to detect diseases and pests in plants.
- **Web Server:** Communicates with the robot to provide navigation instructions and action tokens.
- **Voltage Converter:** Steps up the voltage to 12V for motor operation.
- **TMC2208 Motor Drivers:** Control the NEMA 17 motors for smooth and precise movement.

- **Relay:** Controls the hydraulic motor for spraying chemicals on target objects.
- **Accelerometer:** Provides feedback for maintaining stability during navigation.
- **CAD - Onshape** Used for designing the robot and its components.
- **Python:** Programming language used for developing the robot's software.
- **FastAPI:** Web framework used for creating the REST API for communication.
- **Orca Slicer:** Used for slicing the 3D models for printing.
- **Ender 3:** 3D printer used for printing the robot's components.
- **S4 9V Battery:** Power source for the arm motors and control system.

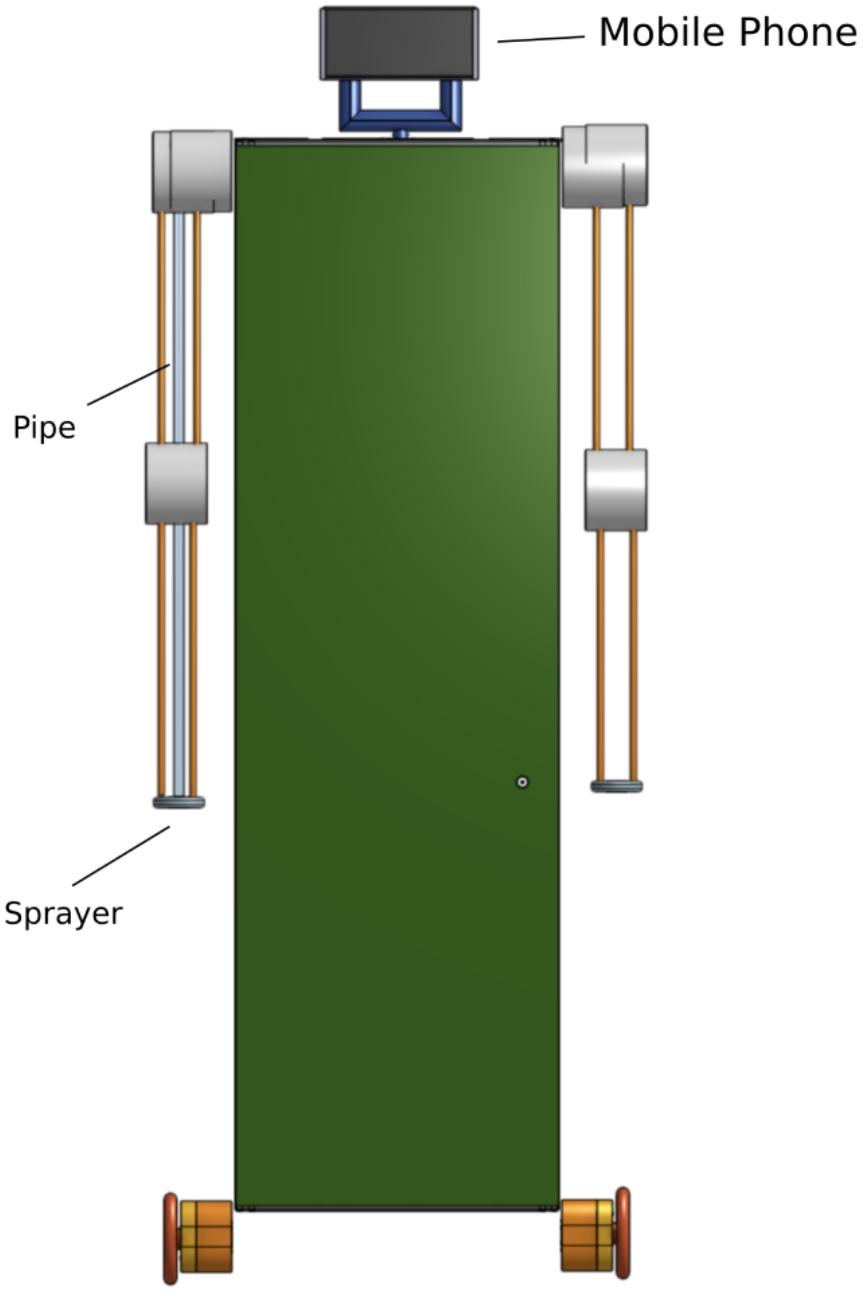

Figure 5.2: Diagram of the Agricultural Robot

Figure 5.3: Wheels of the Agricultural Robot

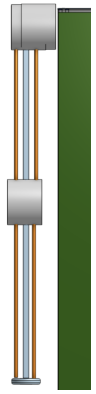

Figure 5.4: Arm of the Agricultural Robot

Chapter 6

Conclusion

The agricultural sector is undergoing a transformative phase, driven by the integration of advanced technologies such as robotics, spatial mapping, and multi-modal language models. This report has explored how these technologies, when applied effectively, can revolutionize traditional farming practices and address the pressing challenges faced by modern agriculture, such as climate change, resource scarcity, and the growing global demand for food.

One of the central contributions of this research has been the application of Gaussian splatting for real-time navigation and mapping in agricultural environments. By transforming sparse 3D point clouds into continuous Gaussian distributions, this method provides a precise and efficient representation of the environment. The ability of agricultural robots to navigate through complex terrains with safety and accuracy is greatly enhanced by this technique. Gaussian splatting also supports dynamic real-time updates, allowing robots to continually refine their understanding of the environment as they move through it. This level of precision in navigation is crucial for autonomous farming systems, where the ability to make real-time decisions can lead to more efficient field operations and reduce the margin of error in crop management.

Another major focus of this report has been on the use of convolutional neural networks (CNNs) for disease and pest detection in crops. The CNN-based models trained on agricultural datasets have demonstrated their effectiveness in accurately identifying diseases and pests from plant images. This capability is vital for precision agriculture, as it allows for targeted interventions that can prevent widespread crop damage, reduce the unnecessary use of pesticides, and promote healthier crop growth. By leveraging CNNs, farmers can move away from traditional blanket spraying techniques and instead adopt a more focused approach that conserves resources and minimizes environmental impact.

In addition to these advancements, the integration of Visual Language Action Models (VLAMs) into agricultural robotics represents a significant leap forward. VLAMs allow robots to interpret and respond to natural language instructions in combination with visual data, enabling them to perform complex tasks autonomously. For example, a robot equipped with VLAM capabilities can navigate a field, identify specific crops, and execute actions such as spraying pesticides or applying fertilizers based on spoken commands. This integration of language and vision into robotic control systems enhances the versatility and adaptability of agricultural robots, allowing them to handle a wider range of tasks with minimal human intervention.

While the current system shows significant promise, several areas for improvement have been identified. The following improvements could enhance the performance and efficiency of the system:

- **Onboard Computer:** The system would benefit from the inclusion of an onboard computer with sufficient computational power to run the necessary models directly on the robot. Currently, the processing relies heavily on external systems, which introduces latency and reduces real-time responsiveness. An onboard computer would enhance autonomy and reduce dependency on cloud-based infrastructure, allowing for quicker decision-making and more efficient operations in the field.

- **Wheelbase Optimization:** The robot's current wheelbase design is not optimal for navigating farm environments. [9] The existing design struggles with uneven terrain and lacks the necessary stability for consistent performance in rough agricultural settings. A track-based system with suspension would provide better traction, stability, and adaptability to the varying conditions found in farms, leading to improved navigation and task execution.

- **Frame Sturdiness:** The robot's frame, currently constructed from aluminum extrusion, has shown weaknesses under the demands of field operation. Replacing the frame with a more robust core material could significantly improve the robot's durability and longevity. A sturdier frame would better withstand the stresses of farm work, especially when carrying heavy equipment or operating in harsh conditions.

The implications of these findings are far-reaching. The successful integration of robotics, spatial mapping, and language models into agriculture can lead to significant improvements in productivity, sustainability, and cost-efficiency. Autonomous robots equipped with advanced perception and decision-making capabilities can perform repetitive and labor-intensive tasks, freeing up human labor for more strategic activities. Moreover, by optimizing the use of inputs such as water, fertilizers, and pesticides, these technologies can contribute to more sustainable farming practices that are better aligned with environmental conservation goals.

Despite the promising results, this research also highlights areas where further exploration is needed. The future of agricultural robotics will depend on continuous improvements in several key areas. First, advancements in real-time data processing and the development of more robust algorithms will be essential to ensure that robots can operate effectively in diverse and unpredictable agricultural environments. Second, the availability of larger and more diverse datasets for disease and pest detection will be critical in improving the accuracy and generalization capabilities of CNN models. [5] As agricultural environments vary widely across regions and climates, models must be trained on data that reflect this diversity to be effective on a global scale.

Furthermore, the generalization capabilities of Visual Language Action Models must be enhanced to handle more complex and nuanced instructions, especially in multilingual and culturally diverse farming contexts. Fine-tuning these models for specific agricultural tasks while ensuring they maintain their adaptability to new tasks is an ongoing challenge. Additionally, as these technologies continue to develop, ethical considerations such as data privacy, the impact on rural employment, and the equitable distribution of technological benefits must be addressed to ensure that the advancements in agricultural robotics contribute positively to society as a whole.

In conclusion, the combination of robotics, spatial mapping, and multi-modal language models holds immense potential for the future of agriculture. By continuing to innovate in these areas, we can create more efficient, resilient, and sustainable farming systems that are capable of meeting the demands of a growing global population. This research marks an important step toward realizing that vision, but there is still much work to be done. As we move forward, the integration of

cutting-edge technologies into agriculture will require a collaborative effort from researchers, engineers, farmers, and policymakers to ensure that these advancements are implemented in ways that maximize their benefits for both people and the planet.

Bibliography

[1] Mostafa Rahimi Azghadi, Alex Olsen, Jake Wood, Alzayat Saleh, Brendan Calvert, Terry Granshaw, Emilie Fillols, and Bronson Philippa. Precise robotic weed spot-spraying for reduced herbicide usage and improved environmental outcomes – a real-world case study, 2024.

[2] Matteo Bortolon, Theodore Tsesmelis, Stuart James, Fabio Poiesi, and Alessio Del Bue. 6dgs: 6d pose estimation from a single image and a 3d gaussian splatting model, 2024.

[3] Chang Che, Qunwei Lin, Xinyu Zhao, Jiaxin Huang, and Liqiang Yu. Enhancing multimodal understanding with clip-based image-to-text transformation, 2024.

[4] Timothy Chen, Ola Shorinwa, Joseph Bruno, Javier Yu, Weijia Zeng, Keiko Nagami, Philip Dames, and Mac Schwager. Splat-nav: Safe real-time robot navigation in gaussian splatting maps, 2024.

[5] Mikolaj Cieslak, Umabharathi Govindarajan, Alejandro Garcia, Anuradha Chandrashekar, Torsten Hädrich, Aleksander Mendoza-Drosik, Dominik L. Michels, Sören Pirk, Chia-Chun Fu, and Wojciech Pałubicki. Generating diverse agricultural data for vision-based farming applications, 2024.

[6] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. The faiss library, 2024.

[7] Tom Duckett, Simon Pearson, Simon Blackmore, Bruce Grieve, Wen-Hua Chen, Grzegorz Cielniak, Jason Cleaversmith, Jian Dai, Steve Davis, Charles Fox, Pål From, Ioannis Georgilas, Richie Gill, Iain Gould, Marc Hanheide, Alan Hunter, Fumiya Iida, Lyudmila Mihalyova, Samia Nefti-Meziani, Gerhard Neumann, Paolo Paoletti, Tony Pridmore, Dave Ross, Melvyn Smith, Martin Stoelen, Mark Swainson, Sam Wane, Peter Wilson, Isobel Wright, and Guang-Zhong Yang. Agricultural robotics: The future of robotic agriculture, 2018.

[8] Zafer Duraklı and Vasif Nabiyev. A new approach based on bezier curves to solve path planning problems for mobile robots. *Journal of Computational Science*, 58:101540, 2022.

[9] Mogeeb A. Elsheikh. Design of a special rigid wheel for traversing loose soil. *Scientific Reports*, 13(1):171, 2023.

[10] Arun Pandian J and Geetharamani Gopal. Data for: Identification of plant leaf diseases using a 9-layer deep convolutional neural network, 2019.

[11] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics (SIGGRAPH Conference Proceedings)*, 42(4), July 2023.

[12] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering, 2023.

[13] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model, 2024.

[14] Sharada P. Mohanty, David P. Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in Plant Science*, 7, 2016.

[15] Matthias Müller and Vladlen Koltun. Openbot: Turning smartphones into robots, 2021.

[16] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks, 2015.

[17] Filip Radenović, Giorgos Tolias, and Ondřej Chum. Fine-tuning cnn image retrieval with no human annotation, 2018.

[18] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, Eric Mintun, Junting Pan, Kalyan Vasudev Alwala, Nicolas Carion, Chao-Yuan Wu, Ross Girshick, Piotr Dollár, and Christoph Feichtenhofer. Sam 2: Segment anything in images and videos, 2024.

[19] Vasileios Sitokonstantinou, Emiliano Díaz Salas Porras, Jordi Cerdà Bautista, Maria Piles, Ioannis Athanasiadis, Hannah Kerner, Giulia Martini, Lily belle Sweet, Ilias Tsoumas, Jakob Zscheischler, and Gustau Camps-Valls. Causal machine learning for sustainable agroecosystems, 2024.

[20] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[21] Matthew Tancik, Ethan Weber, Evonne Ng, Ruilong Li, Brent Yi, Terrance Wang, Alexander Kristoffersen, Jake Austin, Kamyar Salahi, Abhik Ahuja, David Mcallister, Justin Kerr, and Angjoo Kanazawa. Nerfstudio: A modular framework for neural radiance field development. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings*, SIGGRAPH '23. ACM, July 2023.

[22] Puneeth N. Thotad, Shanta Kallur, and Anupama Nandeppanavar. An efficient model for plant disease detection in agriculture using deep learning approaches. In *2023 4th IEEE Global Conference for Advancement in Technology (GCAT)*, pages 1–6, 2023.

[23] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[24] Lintong Zhang, Yifu Tao, Jiarong Lin, Fu Zhang, and Maurice Fallon. Visual localization in 3d maps: Comparing point cloud, mesh, and nerf representations, 2024.